

# Senet

A Programming Language for Playing Board Games

**22<sup>th</sup> December 2015**

Lilia Nikolova - lpn2112  
Maxim Sigalov - ms4772  
Dhruvkumar Motwani - dgm2138  
Srihari Sridhar - ss4964  
Richard Muñoz - rtm2129

# Table of Contents

1. Overview
2. Language Tutorial
3. Language Reference Manual
  - 3.1. Lexical Conventions
  - 3.2. Meaning of Identifiers
  - 3.3. Expressions
  - 3.4. Declarations
  - 3.5. Statements
  - 3.6. Program Structure and Scope
  - 3.7. Example Program
4. Project Plan
  - 4.1. Process
  - 4.2. Style Guide
  - 4.3. Project Timeline
  - 4.4. Roles and Responsibilities
  - 4.5. Development Environment
  - 4.6. Project Log
5. Architectural Design
  - 5.1. Block Diagram
  - 5.2. Scanner
  - 5.3. Parser
  - 5.4. AST and Semantic Checks
  - 5.5. CAST and Code Generation
6. Test Plan
7. Lessons Learned
8. Appendix

# 1. Overview

Past projects for Programming Languages and Translators have included languages for expressing the setup and flow of playing card games. Inspired by such languages, we propose to extend the domain to general, two-dimensional board games. Examples of games that might be expressed in our proposed language are tic-tac-toe, checkers, and chess. A similar idea has been investigated by Romein, Bal and Grune (1995), who described a language called *Multigame* that compiled to a parallel game playing program.<sup>1</sup> The authors focused their research on parallelized artificial intelligence to find optimal moving while playing games created in *Multigame*. In part due to this research focus, the authors restricted the group of games that could be described in *Multigame* to those with fixed-sized boards (thereby excluding card games) and to those where all players have perfect information.

We propose a similar language focused on simple expression of board games; however, its compiler will create games that may be played interactively on the command line. The players will execute the game program of their choice after which they will be presented with prompts that navigate them through the game. We have named our new language *Senet* after one of the oldest-known board games, which traces its origins back to ancient Egypt.

## 1.1 Goals

With our proposed language, we aim to provide:

1. Intuitive, relatively high-level expression of the setup and flow of board games;
2. Simple description of boards and pieces; and
3. Static, strong typing, and a mix of C and Python syntax to minimize the learning curve.

# 2. Language Tutorial

The following shows a “hello world” program in *Senet*.

```
@setup { }

@turns {

func void begin() {
```

---

<sup>1</sup> J. Romein, H. Bal and D. Grune. (1995). Multigame - A Very High Level Language for Describing Board Games. ASCI 95, pp. 278-287.

```

        print("Hello World\n");
    end;
}
}
```

Every *senet* program consists of two major sections: @setup and @turns. The former section is used to declare global functions, groups (*Senet* classes), and variables. In the latter section, a set of functions can be declared, which automatically act though they might be turns in a board game. Every *Senet* program starts with the `begin()` turn function and the `end` keyword in a `turns` function is used to terminate the program.

The other way to end a turn function is to use a `pass` statement, which takes a turn function and an integer (which identifies a player number). The `pass` statement will then execute the turn with that player number. The first player number is always 0. In *Senet*, this turn passing can be used to create interactive games between one or more players, and also change the turn type. Below is a short example of turn passing:

```

@setup { }
@turns {

func void begin() {
    if (PLAYER_ON_MOVE = 1) {
        end;
    } else {
        pass(hello, 1);
    }
}

func void ending() {
    print("Hello!"); print("\n");
    pass(begin, 1);
}
```

In addition to the `turns` function, *Senet* groups allow object-oriented construction of board games inheriting from built in groups (see Section 4.3).

# 3. Language Reference Manual

## 3.1 Lexical Conventions

There are 6 kinds of tokens: identifiers, keywords, integer literals, string literals, expression operators, and other separators. Blank, tab, and newline characters are only used to separate tokens, and are discarded by the scanner.

### 3.1.1 Comments

Lines (or the remainder of a line) can be commented one at a time with `#`. *Senet* does not recognize multiline comments.

### 3.1.2 Identifiers

Identifiers consist of a sequence of letters, digits, and underscores. They must begin with either a letter or underscore. *Senet* considers two identifiers differing only by case to be different.

### 3.1.3 Keywords

The following are reserved keywords in *Senet* and may not be used for any other meaning:

- `if`
- `else`
- `elif`
- `for`
- `while`
- `break`
- `continue`
- `end`
- `return`
- `True`
- `False`
- `None`
- `in`
- `int`
- `str`
- `bool`
- `void`
- `list`
- `group`

- and
- or
- not
- assert
- func
- pass
- @setup
- @turns
- this

### 3.1.4 Literals

*Senet* includes three kinds of literals that have fixed values: integer, string, and list literals. In addition, *Senet* includes a literal representing `None`. Notably, *Senet* does not support floating point literals.

#### Integer

An `IntLiteral` is a sequence of digits. All integer literals in *Senet* are base 10.

#### String

A `StrLiteral` is a sequence of characters enclosed by double quotation marks. A double quotation mark inside a string must be written as “`\”`”. A newline character inside a string must be written as “`\n`”. A backslash character inside a string must be written as “`\\\`”.

#### Void

A `VoidLiteral` represents `None`, the absence of a value.

#### Bool

A `BoolLiteral` takes a `bool_lit` as an argument, which in turn is either `True` or `False`.

#### List

A `list_lit` consists of elements of a common type separated by commas and enclosed in brackets. In addition, a `list_lit` may be the empty list of length zero (`[]`). In that case, it will have type `void`. The expression list must consist of literals or field expressions.

```
list_lit -> []
    | [ expr_list ]
```

## 3.2 Meaning of Identifiers

Since *Senet* is a strongly-typed language, identifiers are associated with types. There are 5 kinds of identifiers in the language: basic types, derived types, group definitions, function definitions, and group instances.

### 3.2.1 Basic Types

*Senet* includes a number of basic types inspired from C and Python as shown in the table below:

Basic Types	Meaning
int	32-bit Integer
str	String
bool	Boolean (True or False)
void	type of None, a value used to represent the absence of a value

### 3.2.2 Derived Types

The following table lists *Senet* types are derivatives (collections) of a basic type. The list type takes a basic type, T, which indicates the type of its elements.

Derived Types	Meaning
list[T]	Linked lists; e.g. list[int] a; a = [1, 2, 3];
group T	Object instance of type group T.

### 3.2.3 Group Definitions

The language is object-oriented, with inheritance (but no multiple inheritance). New groups can be defined with the `group` keyword as described in Section 3.3. *Senet* includes the standard groups shown in the table below built-in and meant to be extended by the programmer. *Senet* does not currently support virtual functions. For boards,

<b>Standard groups</b>	<b>Meaning</b>	<b>Built-in Methods &amp; Attributes</b>
Object	Base object group; all groups extend this	Object(): default constructor, which contains a single statement: <code>return this;</code>
Board	Defines board geometries, win conditions, cleanup methods. Its constructor initializes the cells and occupied fields.	<p><code>place(Piece p, int x)</code>: places the piece on the board at index <code>x</code> of <code>b.cells</code></p> <p><code>remove(int x)</code>: removes the piece at index <code>x</code>, and sets <code>occupied[x]</code> to <code>False</code>;</p> <p><code>owns(int x)</code>: returns the number of the player at index <code>x</code> of <code>cells</code></p> <p><code>full()</code>: returns <code>True</code> if all spots are occupied, else <code>False</code></p> <p><code>cells</code>: list of board cells, each element is either a dummy <code>Piece</code> instance with <code>s = " "</code> or the <code>Piece</code> at that spot.</p> <p><code>occupied</code>: list of <code>bool</code> stating whether each cell is occupied.</p> <p><code>toi(list[int] l)</code>: takes human-readable coordinate list, maps it to an internal cell index</p> <p><code>tol(int x)</code>: takes an internal cell index, and maps to a human-readable coordinate list</p>
Piece	Defines possible moves, keeps track of position, owning player, and other needed variables. Contains a special <code>__repr__()</code> method that returns <code>s</code> .	<p><code>s</code>: string representation</p> <p><code>owner</code>: the owner of the piece</p> <p><code>fixed</code>: whether or not the piece can be overwritten</p>

### 3.2.4 Group Instances

Identifiers with types corresponding to group definitions can be created as:

```
group Id Id ( expr list ) ;
```

```
expr list -> expr
```

```
| expr list , expr
```

If the group's constructor requires parameters, the number of expressions matching the number of parameters must be included inside the parentheses.

### 3.2.5 Functions

An identifier is bound to a function using syntax described in Section 3.4. The identifier can then be used to call the function using a Call, which is described further in Section 3.3.

### 3.2.6 Boards Library

Senet comes with a standard library of Board subclasses:

Board Declaration	Meaning
Borads.Rect(int x, int y)	x by y size rectangular board
Borads.Loop(int x)	Loop-shaped board with x cells
Borads.Line(int x)	Linear board with x cells
Borads.Hex(int x)	Hexagonal lattice of radius x

## 3.3 Expressions

The precedence of expression operators in Senet follows the ordering of the following subsections 4.1 to 4.10, with the highest precedence first. All operators are left associative unless otherwise specified in the subsections below.

An expression can be any of the following:

```
expr -> IntLiteral
| StrLiteral
| list_lit
| bool_lit
| VoidLiteral
| field_expr
| Binop
| Assign
| Call
| Element
| Uminus
| Not
```

```
| Noexpr
| ( expression )
| Place
| Remove
```

There are a number of binary operation (`binop-expr`) expressions. Because they have different precedence, the following binary operation productions will be discussed in separate subsections, along with `field_expr`, `Element`, `Call`, and `assign-expr`.

```
binop-expr -> multiplicative-expr
    | additive-expr
    | relational-expr
    | equality-expr
    | log-and-expr
    | log-or-expr
```

The most basic, and highest precedence expression are literals (`integer`, `string`, and `list`), identifiers, and parenthesized expressions. In addition, `Noexpr` represents an empty expression.

Access to an object's fields (methods or attributes) is accomplished using a period ('.') between the object's identifier and the field as shown below. The type of the `field-expr` is the type of the accessed field.

```
field_expr -> Id
    | This
    | field_expr . Id
```

Functions are called as follows, with return type equal to the called function's return type. The identifier must be a function.

```
Call -> Id ( expr list )
```

### 3.3.1 List Element Access

A single element of a list can be accessed using an element-expression. The left expression must have type `list`. The right expression (in brackets) must have type `int`. The return type is the type of the list's elements.

```
Element -> expr [ expr ]
```

### 3.3.2 Unary Minus

The language includes the unary minus ('-') operator that negates the value to its right. It's nonassociative. The expression operand must have type `int`. The operator is described by the following production rule:

```
Uminus -> expr
```

### 3.3.3 Multiplicative Expressions

*Senet* also includes operators for multiplication ('\*'), integer division ('/'), and modulo ('%'). The language does not support floating point arithmetic. The expression operands must be of type `int`. The value has type `int`.

```
Mult expr -> expr * expr  
          | expr / expr  
          | expr % expr
```

### 3.3.4 Additive Expressions

Math operations in *Senet* are purely integral. The language includes operators for addition ('+') and subtraction ('-'). The types of the expression operands must be `int`. The value has type `int`.

```
Add expr -> expr + expr  
          | expr - expr
```

### 3.3.5 Relational Operators

*Senet* includes the following comparison operators: `>` (greater than), `<` (less than), `>=` (greater than or equal to), `<=` (less than or equal to). The operands must be of type `int`. The value has type `bool`.

```
Rel-op -> >  
          | <  
          | >=  
          | <=  
  
relational-expr -> expr Rel-op expr
```

### 3.3.6 Equality Operators

*Senet* includes the following equality operators `==` (equal to), `!=` (not equal to). The types of the operands must be the same. The value has type `bool`.

```
equality-expr -> expr == expr  
| expr != expr
```

### 3.3.7 Logical NOT

The logical not operator negates its operand, which must be of type `bool`. The return type is `bool`. It's nonassociative.

```
log-not-expr -> not expr
```

### 3.3.8 Logical AND

The logical and operator takes two operands of type `bool`, and has a value of type `bool`.

```
log-and-expr -> expr and expr
```

### 3.3.9 Logical OR

The logical or operator takes two operands of type `bool`, and returns a `bool`.

```
log-or-expr -> expr or expr
```

### 3.3.10 Assignment Expressions

The assignment operator in the language (`'=`) is right associative. The value of an assignment expression is the value of the rightmost of the expressions it is composed of.

```
assign-expr -> Id = expr  
| field_expr = expr
```

### 3.3.11 Board State Modification

The place (`>>`) and remove(`<<`) operators affect board state. The `>>` operator takes a `Piece` and places it on the board at a coordinate (described by a `list[int]`). Therefore, it requires its left operand to be derived from `Piece` and its right operand to be a `Board` and a `list[int]`, which describe the coordinate(s) on the board upon which to place the piece.

Similarly, the `<<` operator takes a piece off the board. The left operand must be of type `Board` and the right operand must be of type `list[int]`.

These operators automatically check to see if the move is legal before performing. If it is not legal, the statement has value `False`. Otherwise, it has value `True`.

```
board-mod-expr ->
    field_expr >> field_expr >> [ list<int> ]
    field_expr << [ list<int> ]
```

## 3.4 Declarations

Any given declaration may be one of the following:

```
decl -> /* nothing */
      | decl vdecl
      | decl fdecl
      | decl gdecl
```

Basic-typed variables are declared as follows:<sup>2</sup>

```
vdecl -> type_id Id
      | type_id Id = expr ;
```

```
type_id -> int | bool | str | void
      | list<type_id>
      | group Id
```

Group definitions follow, where the identifier in parenthesis must be a group from which the new groups is extended:

```
gdecl -> group Id ( Id ){ vdecl_list fdecl_list };
```

```
vdecl_list -> /* nothing */
      | vdecl_list vdecl
```

```
fdecl_list -> /* nothing */
      | fdecl_list fdecl
```

Functions can be declared as follows:

---

<sup>2</sup> If `type-id` is an identifier or `list<identifier>` the identifier has to be a defined group.

```

fdecl -> func type-id Id ( formals_opt ){ vdecl_list stmt_list }
| assert Id ( formals_opt ){ vdecl_list stmt_list }

formals_opt -> /* nothing */
| formals-list

formal_list -> type-id Id
| formal_list , type_id Id

```

There is an additional special class of `assert` functions. This special class of functions evaluate a list of conditionals sequentially and returns `True` if all of them evaluate to true. If any of the statements evaluates to false, it returns a `False` and breaks immediately.

Note that compound types (groups and lists) cannot be initialized in the setup block.

## 3.5 Statements

A statement is singular:

```

stmt -> expr-stmt
| selection-stmt
| iteration-stmt
| jump-stmt
| board-mod-stmt

```

A statement sequence allows many statements to be executed in order (left to right):

```

stmt_list -> /* nothing */
| stmt_list stmt

stmt_list_req -> stmt
| stmt_list_req stmt

```

### 3.5.1 Expression Statement

An expression statement executes a single expression. The expression statement has a value equal to the expression's value:

```
expr-stmt -> expr ;
```

### 3.5.2 Selection Statements

Branching on `if`, `elif`, and `else` are described by the following:

```
selection-stmt ->
  if ( expr ) { stmt_list }
  | if ( expr ) { stmt_list } elif (expr) { stmt_list }
  | if ( expr ) { stmt_list } else { stmt_list }
```

### 3.5.3 Iteration Statements

An iteration statement may be:

```
iteration-stmt -> while-stmt
  | for-stmt
```

A `while` loop statement executes the statement in brackets repeatedly as long as the expression in parentheses evaluates to `True`.

```
while-stmt -> while ( expr ) { stmt_list }
```

A `for` loop statement executes the statement in brackets repeatedly similarly to the `while` loop statement. However, the `for` loop statement requires a basic-typed variable to be declared, and it executes (incrementing the declared variable each time) until the variable is greater than the upper end of the range.

```
for-stmt -> for ( type_id Id { expr_list } ) { stmt_list }

expr_list -> expr
  | expr_list , expr
```

### 3.5.4 Jump Statements

A jump statement may be one of the following:

```
jump-stmt -> break ;
  | continue;
  | return expr ;
```

A `break` statement must be inside of a `for` or `while` loop. The effect of this statement is to terminate the loop and execute the next statement after the loop. A `continue` statement must be inside of a `for` or `while` loop. The effect of this statement is to jump to the begin of the next loop iteration. A `return` statement must be inside of a function block.

## 3.6 Program Structure and Scope

### 3.6.1 Structure

A *Senet* program consists of the following simple structure:

```
program -> setup-block turns-block EOF
```

Thus, all games must have two program sections: the `setup-block`, which contains global functions, groups, and parameters used to set up the game; and the `turns-block`, which contains a `function-list` which describe turn “phases” functions each of which operate as a “`while True`” loop but can call other functions in the `@turns` section. One of the phase functions must be named `begin`, this will be called when the game begins.

```
setup-block -> @setup { declaration-list statement-list }
turns-block -> @turns { function-list }
declaration-list -> /* nothing */
| declaration-list declaration
```

### 3.6.2 Scope

Identifiers declared in the `setup-block` are visible to the remainder of the program, in the order they are declared. First, the variable declarations are output, followed by groups, and then functions. Identifiers declared within functions are visible only to each function. Identifiers declared as part of groups can be accessed using the field access operator whenever the group instance can be accessed. Within a function that is within a group definition, other identifiers that are part of the group can be called via a field-expression using the `this` keyword. Instances of group definitions can only be created if the group definition can be accessed.

### 3.6.3 Other Built-Ins

A number of library functions are built into the language:

- `clear_input()`
- `exit()`
- `read(int)`
- `stoi(str)`
- `print(expr list)`
- `rand()`

To begin the game over, use `restart`. To quit the program, use `exit`. To read `x` number of integers from standard input, use `read(x)`. To convert a string `s` to an integer, use `stoi(s)` -- note that `s` must be a string of one character only. To pass a string `s` to standard output, use `print()`, which takes an expression list. `clear_input()` is useful when interacting with players so discard the remainder of standard in. `rand()` is the same as its C counterpart.

In addition, one variable is built into the language to control the flow of games:

- `PLAYER_ON_MOVE`

`PLAYER_ON_MOVE` begins at 0 and is used to know which player is currently taking his or her turn. The programmer can use a `pass` statement to change this variable.

## 3.7 Example Program

Tic-tac-toe is a two-player game that is played on a three row, three column board. The players take turns placing either an “X” or an “O” in each cell. A player wins if three of their pieces fall in a line (vertical, horizontal, or diagonal). The game ends in a draw if all cells are full and no player has won. Below, we describe how our language could be used to create an interactive tic-tac-toe game.

```
@setup
{
    list[group line] victory_conds;

    group line(Object) {
        list[int] loci;

        func group line __init__(list[int] l) {
            this.loci = l;
            return this;
        }
    };
}

group ttb(Rect(3, 3)) {

    assert owner_of_line(group line l, int player) {
        this.owns(l.loci[0]) == player;
        this.owns(l.loci[1]) == player;
        this.owns(l.loci[2]) == player;
    }

    func bool three_in_a_row(int player) {
        int i = 0;
        group line l;
```

```

while (i < 8) {
    l = victory_conds[i];
    # print("DEBUG: three_in_a_row, i = "); print(i); print("\n");

    if (this.owner_of_line(l, player)) {
        return True;
    }
    i = i + 1;
}
return False;
}

func bool won(int player) {
    if (this.three_in_a_row(player)) {
        return True;
    }
    return False;
}

assert draw() {
    this.full();
}

func str __str_of_row(int row) {
    group Piece l; group Piece m; group Piece r;
    str ret;
    # print("DEBUG: start of __str_of_row with row = "); print(row);
print("\n");
    # print("DEBUG: print [0, row] = "); print([0, row]); print("\n");
    l = this.cells[this.toi([0, row])];
    # print("DEBUG: l = this.cells call ok\n");
    m = this.cells[this.toi([1, row])];
    r = this.cells[this.toi([2, row])];
    ret = "[" + l.__repr__() + ", " + m.__repr__() + ", " + r.__repr__() +
"]\n";
    return ret;
}

func str __repr__() {
    return this.__str_of_row(0) + this.__str_of_row(1) +
this.__str_of_row(2);
}

group Mark(Piece) {
    func group Mark __init__(str s, int player) {
        this.s = s;
        this.owner = player;
        return this;
    }
};

int N_PLAYERS = 2;

```

```

group ttb b;
}

@turns
{

func void begin() {
    group line v0; group line v1; group line v2; group line v3;
    group line v4; group line v5; group line v6; group line v7;
    int x; int y; int z;
    # list[group line] victory_conds;

    b = ttb();

    x = b.toi([0, 0]); y = b.toi([1, 0]); z = b.toi([2, 0]);
    v0 = line([x, y, z]);
    x = b.toi([0, 1]); y = b.toi([1, 1]); z = b.toi([2, 1]);
    v1 = line([x, y, z]);
    x = b.toi([0, 2]); y = b.toi([1, 2]); z = b.toi([2, 2]);
    v2 = line([x, y, z]);
    x = b.toi([0, 0]); y = b.toi([0, 1]); z = b.toi([0, 2]);
    v3 = line([x, y, z]);
    x = b.toi([1, 0]); y = b.toi([1, 1]); z = b.toi([1, 2]);
    v4 = line([x, y, z]);
    x = b.toi([2, 0]); y = b.toi([2, 1]); z = b.toi([2, 2]);
    v5 = line([x, y, z]);
    x = b.toi([0, 0]); y = b.toi([1, 1]); z = b.toi([2, 2]);
    v6 = line([x, y, z]);
    x = b.toi([0, 2]); y = b.toi([1, 1]); z = b.toi([2, 0]);
    v7 = line([x, y, z]);

    # print("DEBUG: v7 ok\n");

    victory_conds = [v0, v1, v2, v3, v4, v5, v6, v7];
    # print("DEBUG: built victory_conds\n");

    # b = ttb();

    # print("DEBUG: built b\n");

    pass(prompt, 0);
}

func void prompt() {
    int a;
    int c;
    group Mark m;
    int i;

    if (PLAYER_ON_MOVE % 2 == 0) {

```

```

        m = Mark("X", PLAYER_ON_MOVE);
    } else {
        m = Mark("O", PLAYER_ON_MOVE);
    }

# print("DEBUG: start of prompt()\n");

print("\n"); print(b); print("\n");

# print("DEBUG: printed b\n");

# players input moves by typing coordinates, e.g. "11" or "02"
print("PLAYER "); print(PLAYER_ON_MOVE); print(": ");
print("Input coordinates of square to place ");
print("in i.e. \"22\" or \"10\".\n");
a = stoi(read(1));
c = stoi(read(1));
clear_input();
# print("index of input: ["); print(a); print(", "); print(c); print("] ->
");
# print(b.toi([a, c])); print("\n");

if (m >> b >> [a, c]) {

    # PLAYER_ON_MOVE is the index of the player
    if (b.won(PLAYER_ON_MOVE)) {
        pass(winner, PLAYER_ON_MOVE);
    }
    if (b.draw()) {
        pass(nowinner, PLAYER_ON_MOVE);
    }
} else {
    # A piece is already at [a, c]
    print("Cannot place a mark there, try again.\n\n");
    pass(prompt, PLAYER_ON_MOVE);
}
# if the move was legal, went through successfully,
# and the game is not over, pass the turn to the next player
pass(prompt, (PLAYER_ON_MOVE + 1) % N_PLAYERS);
}

func void winner() {
    print("\n"); print(b); print("\n");
    print("Player "); print(PLAYER_ON_MOVE); print(" wins.\n");
    print("Congratulations!\n");
    end;
}

func void nowinner() {
    print("\n"); print(b); print("\n");
    print("Game ends in a draw.\n");
    end;
}

```

}

## 4. Project Plan

### 4.1 Process

#### 4.1.1 Planning

During the first half of the semester our group met once a week on Fridays. We met in the mornings and worked as long as we needed to in order to complete the necessary work for each assignment that we had to turn in. This served as a good guide for the progression of our project throughout the semester. We set weekly tasks based on the deliverables and tried to complete most of them during our meetings. This allowed us to brainstorm ideas, share our knowledge, and contribute to various aspects of the project. There were weeks when we met with our mentor Lixin Yao in order to make sure that we were on the right track or consult about problems that we weren't able to figure out ourselves.

During the second half of the semester we met twice a week. We found it was necessary because we started working on developing and testing our code. Once again, we worked together during those meetings and although our weekly goals weren't as clearly outlined as before we progressed by building on our first "hello world" test case.

#### 4.1.2 Specification

Even though we had a good idea of what we wanted our language to do and the types of building blocks required, the construction of the specification for our language was not an easy project and was one that took the whole semester to refine. The task of writing the Language Reference Manual helped us with figuring out the syntax and semantics but it wasn't until we started coding each aspect of the compiler that we realized the mistakes we had made initially. We kept updating our LRM almost every time something changed as we were developing each component (scanner, AST, parser, etc.). We also defined a standard library for our language and did a second major revision of the LRM after we had coded and tested everything.

#### 4.1.3 Development and Testing

Development followed the stages specified in the class lectures. We started with developing the scanner, parser, AST, and enough compiling to C in order to print out the "hello world" message. From there, we expanded the code within the components to handle all the necessary test cases and include the required functionality to build a working program.

## 4.2 Style Guide

We generally used the following conventions while programming our compiler in order to ensure consistency, readability, and transparency:

- Ocaml editing and formatting style was generally followed when writing Ocaml code.
- C style editing and formatting style was used when writing Senet code.

## 4.3 Project Timeline

Date	Task
September 30th	Submitted Project Proposal
October 26th	Submitted Language Reference Manual
October 26th	Compiler Front End Complete
November 13th	Code Generation for Hello World Complete
November 13th	Hello World Runs
November 16th	Hello World Presentation
December 20th	Code Generation Complete
December 20th	Testing and Debugging Complete
December 21st	Presentation Complete
December 22nd	Submitted Final Project

## 4.4 Roles and Responsibilities

A big part of our project was done by the team working in the same physical space and a lot of our roles overlapped because everyone did a little bit of everything. The table below provides information about each team member's main responsibilities as outlined in the beginning of the semester along with specific aspects of the project they were involved in:

Team Member	Role	Responsibilities
Lilia Nikolova	Manager	Timely completion of deliverables, team organization, documentation, testing

Maxim Sigalov	Language Guru	Language design, semantics
Dhruvkumar Motwani	Language Guru	Language design, standard library
Srihari Sridhar	System Architect	Compiler architecture, standard library, code generation, semantics
Richard Muñoz	Verification and Validation	Test plan, test suites, code generation, semantics

## 4.5 Development Environment

- Github repository - version control; contains compiler code, tests, program examples, and standard library.
- OCaml 4.02.1 - parsing and semantic checking.
- gcc-5 - building C output of compiler.
- Valgrind - debugging C code.

## 4.6 Project Log

Date	Task
September 18th	Determined language function and purpose
September 25th	Determined main language features
September 30th	Completed language proposal
October 2nd	Defined grammar
October 9th	Defined grammar
October 16th	Compiler front end, semantics
October 23rd	Compiler front end, semantics, type checking
October 26th	Completed Language Reference Manual
October 26th	Compiler front end complete
October 30th	Hello World code generation
November 3rd	Hello World code generation
November 6th	Hello World code debugging
November 13th	Code generation for Hello World complete

November 13th	Hello World runs
November 16th	Hello World presentation
November 20th	Code generation, testing
December 24th	Code generation, testing
December 1st	Code generation, testing
December 4th	Code generation, testing
December 8th	Code generation, testing, standard library
December 11th	Code generation, testing, standard library
December 15th	Code generation, testing, debugging
December 18th	Code generation, testing, debugging
December 20th	Code generation complete
December 20th	Testing and debugging complete
December 21st	Presentation complete
December 22nd	Submitted final project

## 5. Architectural Design

### 5.1 Compiler Block Diagram

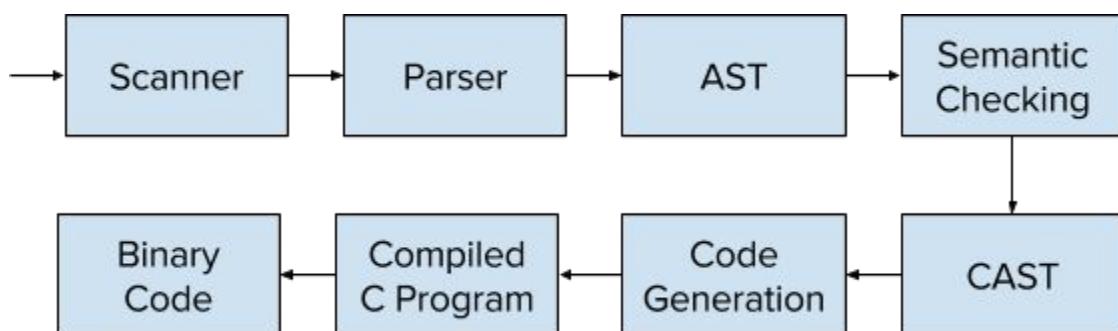


Figure 1 : Architecture of Senet compiler

## 5.2 Scanner

The scanner, implemented in OCamlLex takes Senet code as input and tokenizes it into defined keywords in the language, literals and constants. It also discards white spaces and designated comments in the language. These tokens are then passed on to the parser for building the Abstract Syntax Tree.

## 5.3 Parser

The parser takes the tokens from the scanner and produces the Abstract Syntax Tree using the grammar definitions in the parser.mly file. Additionally, this checks for consistency between the data types used in the program and the valid definitions listed in the ast.ml file.

## 5.4 AST and Semantic Checks

The Abstract Syntax Tree is the intermediary representation used by the parser and the subsequent stages of the compilation process which includes only the most relevant aspects of the tokenized code received from the scanner. Each individual AST type is listed as a node in the a tree representation. Semantic checks are run on this representation to check for consistency with the grammar rules defined for the language.

## 5.5 CAST and Code Generation

These two components encompass the backend of the compilation process where the semantically checked AST representation is converted modified into another AST that aligns with the C syntax to help ease the transition from Senet to C code that happens in the code generation process. In particular, the ordering of group attributes is standardized so that C can cast pointers between them. In addition, expressions with a group type are filled in with the full SAST group declaration in order to facilitate accessing attributes and functions and casting. Also, some list literals are tagged with temporary names for the C code to use to build linked lists (since the C program needs memory locations to use the address operator). From this intermediary representation, relevant C code is generated which can then be compiled and executed the same way as any generic C program.

# 6. Test Plan

First, below is the test file for assert functions, test-assert-func.snt:

```
@setup {
  assert test() {
    True;
```

```

}

assert test2() {
    2 + 2 > 3;
    4 - 2 < 0;
    2 == 2;
}

assert test3(int x) {
    if (x > 3) { True; } else { False; }
}

}

@turns {

func void begin() {
    print(True); print("\n");
    print(test()); print("\n");
    print(test2()); print("\n");
    print(test3(2)); print("\n");
    end;
}
}

@turns {

func void begin() {
    print(True); print("\n");
    print(test()); print("\n");
    print(test2()); print("\n");
    print(test3(2)); print("\n");
    end;
}
}

```

The following shows the C output for compiling test-assert-func.snt:

```

// @senet_header
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```
#include "temp/sen_linked_list.h"
#include "temp/sen_print_base_grps.h"
#include "temp/sen_init_base_grps.h"
#include "temp/sen_read.h"

struct SENET_NONE {
} SENET_NONE;

struct Sen_list snt_SEN_EMPTY_LIST;

char *SENET_STR_CONCAT(char* s1, char* s2) {
    char *temp = (char *) malloc(strlen(s1)+ strlen(s2) +1);
    strcpy(temp, s1);
    strcat(temp, s2);
    return temp;
}

void (*CUR_TURN)();
int snt_PLAYER_ON_MOVE = 0;

// @setup

bool snt_test() {

if (!(true)) { return false; }

return true;
}

bool snt_test2() {

if (!(((2 + 2) > 3))) { return false; }

if (!(((4 - 2) < 0))) { return false; }

if (!((2 == 2))) { return false; }

return true;
}

bool snt_test3(int snt_x) {

if ((snt_x > 3) ) {
if (!(true)) { return false; }

} else {
if (!(false)) { return false; }
}
```

```

}

return true;
}

// @turns
void snt_begin();

void snt_begin() {

printf("%s", true ? "true" : "false");
printf("\n");
printf("%s", snt_test() ? "true" : "false");
printf("\n");
printf("%s", snt_test2() ? "true" : "false");
printf("\n");
printf("%s", snt_test3(2) ? "true" : "false");
printf("\n");
exit(0);
}

// @senet_footer
int main() {
    CUR_TURN = &snt_begin;
    snt_PLAYER_ON_MOVE = 0;
    while (true) {
        CUR_TURN();
    }
    return 0;
}

```

Now, here is a second example of a test script, test-inherit-three-deg.snt, which tests inheriting functions through a grandchild.

```

@setup {

group A(Object) {
    int x;

    func group A __init__(int x) {
        this.x = x;
        return this;
    }

    func int test() {

```

```

        return this.x + 5;
    }

};

group B(A) {
    int y;
};

group C(B) {
    int z;
};

group A obj;
group B obj2;
group C obj3;

}

@turns {

func void begin() {
    int y;

    obj.__init__(1);
    obj2.__init__(2);
    obj3.__init__(3);

    print(obj.x); print("\n");
    print(obj2.x); print("\n");
    print(obj3.x); print("\n");

    print(obj.test()); print("\n");
    print(obj2.test()); print("\n");
    print(obj3.test()); print("\n");

    end;
}
}

```

The C output for test-test-inherit-three-deg.snt:

```

// @senet_header
#include <stdbool.h>
#include <stdio.h>

```

```

#include <stdlib.h>
#include <string.h>
#include "temp/sen_linked_list.h"
#include "temp/sen_print_base_grps.h"
#include "temp/sen_init_base_grps.h"
#include "temp/sen_read.h"

struct SENET_NONE {
} SENET_NONE;

struct Sen_list snt_SEN_EMPTY_LIST;

char *SENET_STR_CONCAT(char* s1, char* s2) {
    char *temp = (char *) malloc(strlen(s1)+ strlen(s2) +1);
    strcpy(temp, s1);
    strcat(temp, s2);
    return temp;
}

void (*CUR_TURN)();
int snt_PLAYER_ON_MOVE = 0;

// @setup
struct snt_A snt_obj;
struct snt_B snt_obj2;
struct snt_C snt_obj3;

struct snt_A{
    int snt_x;
} snt_A;

char* snt_A_snt__repr__(struct snt_A *this) {

return "<Group A instance>";
}

struct snt_A snt_A_snt__init__(struct snt_A *this, int snt_x) {

(*this).snt_x = snt_x;
return (*this);
}

int snt_A_snt_test(struct snt_A *this) {

return ((*this).snt_x + 5);
}

```

```

struct snt_B{
    int snt_x;
    int snt_y;
} snt_B;

char*  snt_B_snt__repr__(struct snt_B *this) {
    return "<Group B instance>";
}

struct snt_B  snt_B_snt__init__(struct snt_B *this, int snt_x) {
    (*this).snt_x = snt_x;
    return (*this);
}

struct snt_C{
    int snt_x;
    int snt_y;
    int snt_z;
} snt_C;

char*  snt_C_snt__repr__(struct snt_C *this) {
    return "<Group C instance>";
}

struct snt_C  snt_C_snt__init__(struct snt_C *this, int snt_x) {
    (*this).snt_x = snt_x;
    return (*this);
}

// @turns
void  snt_begin();

void  snt_begin() {

    int snt_y;
    snt_A_snt__init__((struct snt_A *) &snt_obj, 1);
    snt_B_snt__init__((struct snt_B *) &snt_obj2, 2);
    snt_C_snt__init__((struct snt_C *) &snt_obj3, 3);
    printf("%d", snt_obj.snt_x);
    printf("%s", "\n");
    printf("%d", snt_obj2.snt_x);
    printf("%s", "\n");
    printf("%d", snt_obj3.snt_x);
}

```

```

printf("%s", "\n");
printf("%d", snt_A_snt_test((struct snt_A *) &snt_obj));
printf("%s", "\n");
printf("%d", snt_A_snt_test((struct snt_A *) &snt_obj2));
printf("%s", "\n");
printf("%d", snt_A_snt_test((struct snt_A *) &snt_obj3));
printf("%s", "\n");
exit(0);
}

// @senet_footer
int main() {
    CUR_TURN = &snt_begin;
    snt_PLAYER_ON_MOVE = 0;
    while (true) {
        CUR_TURN();
    }
    return 0;
}

```

In total, the test suites used are below. We generally added tests for each new feature we added to the language or when we discovered that features that appeared to be work did not. The test automation uses a modified version of the `testall.sh` script from microc.

## 7. Lessons Learned

### 7.1 Lilia Nikolova

Scheduling and communication within a team are crucial elements when working on a large-scale group project. You have to make sure that everyone is on the same page at all times and not assume or expect that everyone knows exactly what to do. This is achieved by being proactive (in terms of reaching out and coordinating times to meet) and assertive (in terms of assigning tasks and making sure they are complete). There's no room for shyness.

### 7.2 Maxim Sigalov

Ambition needs to be checked by realistic expectations. A desire for more and more features can thin out work and make it harder to assemble a working subset compliant with the desired specification. At the same time, forward thinking necessary in order to avoid having to change large amounts of code to allow for more advanced features. Classes with true inheritance and virtual tables are not as easy as languages like C++ and Python make, and the temptation to overuse macros is hard to resist.

## **7.3 Richard Muñoz**

Creating test scripts was essential to making progress. It was nice to see that small pieces of the compiler began to work. Translating code to C proved to be more difficult than I anticipated. I would advise next year's groups to spend time during the LRM stage to specify as many details as possible, since they are needed in order to translate code.

## **7.4 Srihari Sridhar**

It is hard to write good code. It is infinitely harder to build a compiler that recognizes what good code is. Delving into the nuances of what used to be a mere click of the Compile button was a really enlightening experience. From here on, I am fairly sure a miniaturized version of our wrestling matches with OCaml and C will flash before my eyes every time I compile a piece of code. On the management side, when you have a bunch of really talented people to work with, fixed roles and responsibilities can end up being restrictive. Working together as often as possible and solving each other's problems was crucial to our success.

Lastly, when you think you know C, think again!

## **7.5 Dhruvkumar Motwani**

Creating a programming language is a daunting task in a semester. I learned that the best way to move forward in any project is to allow the best developer(s) to lead and support him/her in all possible ways. The project also allowed me to learn a lot about the production level code since we ended up breaking our code multiple times for small errors and had to spend hours debugging it.

## 8. Appendix

./ast.ml

```
1 type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater |
2     Geq
3         | Mod | And | Or
4
5 type id_type =
6     Int
7     | Bool
8     | Str
9     | Void
10    | List of id_type
11    | Group of string
12
13 type bool_lit =
14     True
15     | False
16
17 type field_expr =
18     Id of string
19     | This
20     | FieldCall of field_expr * string
21
22 type list_lit =
23     Elems of expr list
24     | EmptyList
25
26 and expr =
27     IntLiteral of int
28     | StrLiteral of string
29     | ListLiteral of list_lit
30     | BoolLiteral of bool_lit
31     | VoidLiteral
32     | Field of field_expr
33     | Binop of expr * op * expr
34     | Assign of field_expr * expr
35     | Call of field_expr * expr list
36     | Element of expr * expr
37     | Uminus of expr
38     | Not of expr
39     | Noexpr
40     | Remove of field_expr * list_lit
41     | Place of field_expr * field_expr * list_lit
42
43 type stmt =
44     Block of stmt list
45     | Expr of expr
46     | Return of expr
47     | Break
48     | Continue
49     | If of expr * stmt * expr option * stmt
50     | For of var_decl * expr list * stmt
```

```

50  | While of expr * stmt
51  | End
52  | Pass of string * expr
53
54 and init =
55  | ExprInit of expr
56  | NoInit
57
58 and var_decl = {
59  vname : string;
60  vtype : id_type;
61  vinit : init
62 }
63
64 type basic_func_decl = {
65  ftype : id_type;
66  fname : string;
67  formals : var_decl list;
68  locals : var_decl list;
69  body : stmt list;
70 }
71
72 type assert_decl = {
73  fname : string;
74  formals : var_decl list;
75  locals : var_decl list;
76  body : stmt list;
77 }
78
79 type func_decl =
80  BasicFunc of basic_func_decl
81  | AssertFunc of assert_decl
82
83 type group_decl = {
84  gname : string;
85  extends : field_expr option;
86  par_actualls : expr list option;
87  attributes : var_decl list;
88  methods : func_decl list;
89 }
90
91 type setup = var_decl list * func_decl list * group_decl list
92
93 type turns = func_decl list
94
95 type program = setup * turns
96
97 let rec escaped_string s =
98  Printf.sprintf "%S" s
99
100 let rec string_of_vtype = function
101  Int -> "int"
102  | Bool -> "bool"
103  | Str -> "str"

```

```

104 | Void -> "void"
105 | List(vt) ->
106   "list[" ^ string_of_vtype vt ^ "]"
107 | Group(s) -> s
108
109 let rec string_of_field = function
110   Id(s) -> s
111 | This -> "this"
112 | FieldCall(f,s) -> string_of_field f ^ "." ^ s
113
114 let rec string_of_list_lit = function
115   EmptyList -> "[]"
116 | Elems(e) ->
117   "[" ^ String.concat ", " (List.map string_of_expr e) ^ "]"
118
119 and string_of_expr = function
120   IntLiteral(l) -> string_of_int l (*[? | ?]*)
121 | Field(f) -> string_of_field f
122 | Binop(e1, o, e2) ->
123   string_of_expr e1 ^ " " ^
124   (match o with
125     Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
126     Equal -> "==" | Neq -> "!="
127     Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">="
128     Mod -> "%"
129     And -> "and" | Or -> "or" ) ^ " " ^
130   string_of_expr e2
131 | Assign(f, e) -> string_of_field f ^ " = " ^ string_of_expr e
132 | Call(f, el) ->
133   string_of_field f ^
134   "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
135 | Noexpr -> ""
136 | StrLiteral(s) -> escaped_string s
137 | Uminus(e) -> "-" ^ string_of_expr e
138 | Not(e) -> "not" ^ string_of_expr e
139 | Element(e1, e2) ->
140   string_of_expr e1 ^ "[" ^ string_of_expr e2 ^ "]"
141 | ListLiteral(l) -> string_of_list_lit l
142 | BoolLiteral(b) -> (match b with True -> "True" | False -> "False")
143 | VoidLiteral -> "None"
144 | Place(f1, f2, l) ->
145   string_of_field f1 ^ " >> " ^ string_of_field f2 ^ " >> " ^
146   string_of_list_lit l
147 | Remove(f1, l) ->
148   string_of_field f1 ^ " << " ^
149   string_of_list_lit l
150
151 let rec string_of_vinit = function
152   NoInit -> ""
153 | ExprInit(e) -> " = " ^ string_of_expr e
154
155 let rec string_of_vdecl vdecl =
156   string_of_vtype vdecl.vtype ^ " " ^ vdecl.vname ^
157   string_of_vinit vdecl.vinit

```

```

158 let rec string_of_stmt = function
159   Block(stmts) ->
160     "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
161   | Expr(expr) -> string_of_expr expr ^ ";\n";
162   | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
163   | If(e, s, None, Block([])) ->
164     "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
165   | If(e, s1, e2, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^
166     string_of_stmt s1 ^
167     (match e2 with
168      | None -> "else\n"
169      | Some(expr) -> string_of_expr e) ^
170     string_of_stmt s2
171   | For(vd, elist, s) ->
172     "for (" ^ string_of_vdecl vd ^ " in " ^
173       "\n" ^ String.concat ", " (List.map string_of_expr elist) ^ "\n" ^
174       ") " ^ string_of_stmt s
175   | While(e, s) -> "while (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt
176     s ^ "\n}\n"
177   | Pass(s, e) -> "pass (" ^ s ^ ", " ^ string_of_expr e ^ ")\n"
178   | Break -> "break;\n"
179   | Continue -> "continue;\n"
180   | End -> "end();\n"
181
182 let string_of_basic_fdecl fdecl =
183   "func" ^ " " ^ string_of_vtype fdecl.ftype ^ " " ^
184   fdecl.fname ^ "(" ^
185     String.concat ", " (List.map string_of_vdecl fdecl.formals) ^ ")\n"
186   { \n" ^
187     String.concat "" (List.map (fun v -> string_of_vdecl v ^ ";\n") fdecl.
188     locals) ^
189     String.concat "" (List.map string_of_stmt fdecl.body) ^
190     "}\n"
191
192 let string_of_assert_decl fdecl =
193   "assert" ^
194   fdecl.fname ^ "(" ^
195     String.concat ", " (List.map string_of_vdecl fdecl.formals) ^ ")\n"
196   { \n" ^
197     String.concat "" (List.map (fun v -> string_of_vdecl v ^ ";\n") fdecl.
198     locals) ^
199     String.concat "" (List.map string_of_stmt fdecl.body) ^
200     "}\n"
201
202 let string_of_fdecl = function
203   | BasicFunc(f) -> string_of_basic_fdecl f
204   | AssertFunc(f) -> string_of_assert_decl f
205
206 let string_of_gdecl gdecl =
207   "group" ^ gdecl.gname ^ "(" ^
208     (match gdecl.extends with
209      | Some(par) -> string_of_field par ^

```

```

206         (match gdecl.par_actu als with
207             Some(acts) ->
208                 (" " ^ String.concat ", " (List.map string_of_expr acts
209                  ) ^ ")"
210                  | None -> "")
211                  | None -> "") ^ "\n{\\n" ^
211 String.concat "" (List.map (fun v -> string_of_vdecl v ^ ";\\n") gdecl.
212 attributes) ^
212 String.concat "" (List.map string_of_fdecl gdecl.methods) ^
213 "};\\n"
214
215 let string_of_setup (vars, funcs, groups) =
216   "@setup {\\n" ^
217   String.concat "" (List.map (fun v -> string_of_vdecl v ^ ";\\n") vars) ^
218   String.concat "\\n" (List.map string_of_fdecl funcs) ^
219   String.concat "\\n" (List.map string_of_gdecl groups) ^
220 "}\\n"
221
222 let string_of_turns (funcs) =
223   "@turns {\\n" ^
224   String.concat "\\n" (List.map string_of_fdecl funcs) ^
225 "}\\n"
226
227 let string_of_program (s, t) =
228   string_of_setup s ^ string_of_turns t

```

**./c\_files/headers/all\_headers.h**

```

1 #ifndef __ALL_HEADERS__
2 #define __ALL_HEADERS__
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <stdbool.h>
6 #include <math.h>
7 #include <setjmp.h>
8 #include <string.h>
9 #include "sen_basic_type.h"
10 #include "sen_int.h"
11 #include "sen_object.h"
12 #include "sen_bool.h"
13 #include "sen_string.h"
14 #include "sen_array.h"
15 #include "sen_array_int.h"
16 #include "sen_board.h"
17 #include "sen_board_square.h"
18 #endif

```

**./c\_files/headers/sen\_array.h**

```

1 #include "sen_object.h"
2 #include "sen_basic_type.h"
3 #include "sen_int.h"
4
5 #ifndef SEN_ARRAY_H
6 #define SEN_ARRAY_H
7

```

```

8 #if !defined(ARRAY_SIZE)
9     #define ARRAY_SIZE(x) (sizeof((x)) / sizeof((x)[0]))
10 #endif
11
12 struct Sen_array_vtable;
13 typedef struct Sen_array_vtable Sen_array_vtable;
14
15 struct Sen_array_class;
16 typedef struct Sen_array_class Sen_array_class;
17
18 struct Sen_array;
19 typedef struct Sen_array Sen_array;
20
21 struct Sen_array_vtable {
22     void (*print) (Sen_object *);
23     //Sen_array *(*construct) (Sen_object **, int);
24     Sen_array *(*construct) (int);
25     void (*destruct) (Sen_array *);
26     Sen_array *(*copy) (Sen_array *);
27     Sen_object *(*access) (Sen_array *, Sen_int *);
28     Sen_array *(*concat) (Sen_array *, Sen_array *);
29 };
30
31 struct Sen_array_class {
32     Sen_object_class *superp;
33     Sen_array_vtable *tablep;
34 };
35
36 struct Sen_array {
37     bool bound;
38     Sen_array_class *classp;
39     Sen_object *superp;
40     Sen_object **arr;
41     int len;
42     char print_sep;
43 };
44
45 extern Sen_array_class Sen_array_class_;
46 extern Sen_array_vtable Sen_array_vtable_;
47
48 //Sen_array *construct_array (Sen_object **, int);
49 Sen_array *construct_array (int);
50 void destruct_array (Sen_array *);
51 Sen_array *copy_array (Sen_array *);
52 Sen_object *access_array (Sen_array *, Sen_int *);
53 Sen_array *concat_array (Sen_array *, Sen_array *);
54
55 #define CONCAT_ARRAY(x,y) ({ \
56     __auto_type __temp__ = x; \
57     __temp__->classp->tablep->concat((Sen_array *)__temp__, (Sen_array \
58     *)y); \
59 })
60 //#define CONSTRUCT_ARRAY(x) ((Sen_array *) construct_array(x))

```

```

61 #define CONSTRUCT_ARRAY(array, length) ({ \
62     __auto_type input_arr = array; \
63     __auto_type __temp_arr__ = construct_array(length); \
64     __temp_arr__->len = length; \
65     for (int i=0; i<length; i++) { \
66         __auto_type __temp_elem__ = input_arr[i]; \
67         __temp_arr__->arr[i] = (Sen_object *) COPY(input_arr[i]); \
68         __temp_arr__->bound = true; \
69         if (!__temp_elem__->bound) { \
70             DESTRUCT(__temp_elem__); \
71         } \
72     } \
73     __temp_arr__; \
74 } ) \
75 #endif

```

./c\_files/headers/sen\_array\_int.h

```

1 #include "sen_object.h" \
2 #include "sen_basic_type.h" \
3 #include "sen_int.h" \
4 \
5 #ifndef SEN_ARRAY_INT_H \
6 #define SEN_ARRAY_INT_H \
7 \
8 #if !defined(ARRAY_INT_SIZE) \
9     #define ARRAY_INT_SIZE(x) (sizeof((x)) / sizeof((x)[0])) \
10 #endif \
11 \
12 struct Sen_array_int_vtable; \
13 typedef struct Sen_array_int_vtable Sen_array_int_vtable; \
14 \
15 struct Sen_array_int_class; \
16 typedef struct Sen_array_int_class Sen_array_int_class; \
17 \
18 struct Sen_array_int; \
19 typedef struct Sen_array_int Sen_array_int; \
20 \
21 struct Sen_array_int_vtable { \
22     void (*print) (Sen_object *); \
23     //Sen_array_int *(*construct) (Sen_object **, int); \
24     Sen_array_int *(*construct) (int); \
25     void (*destruct) (Sen_array_int *); \
26     Sen_array_int *(*copy) (Sen_array_int *); \
27     Sen_int *(*access) (Sen_array_int *, Sen_int *); \
28     Sen_array_int *(*concat) (Sen_array_int *, Sen_array_int *); \
29 }; \
30 \
31 struct Sen_array_int_class { \
32     Sen_object_class *superp; \
33     Sen_array_int_vtable *tablep; \
34 }; \
35 \
36 struct Sen_array_int {

```

```

37     bool bound;
38     Sen_array_int_class *classp;
39     Sen_object *superp;
40     Sen_int **arr;
41     int len;
42     char print_sep;
43 };
44
45 extern Sen_array_int_class Sen_array_int_class_;
46 extern Sen_array_int_vtable Sen_array_int_vtable_;
47
48 //Sen_array_int *construct_array_int (Sen_object **, int);
49 Sen_array_int *construct_array_int (int);
50 void destruct_array_int (Sen_array_int *);
51 Sen_array_int *copy_array_int (Sen_array_int *);
52 Sen_int *access_array_int (Sen_array_int *, Sen_int *);
53 Sen_array_int *concat_array_int (Sen_array_int *, Sen_array_int *);
54
55 #define CONCAT_ARRAY_INT(x,y) ({\
56     __auto_type __temp__ = x; \
57     __temp__->classp->tablep->concat((Sen_array_int *)__temp__, (Sen_array_int \
58     *)y); \
59 }
60 //#define CONSTRUCT_ARRAY_INT(x) ((Sen_array_int *) construct_array_int(x))
61 #define CONSTRUCT_ARRAY_INT(array_int, length) ({ \
62     __auto_type input_arr = array_int; \
63     __auto_type __temp_arr__ = construct_array_int(length); \
64     __temp_arr__->len=length; \
65     for (int i=0; i<length; i++) { \
66         __auto_type __temp_elem__ = input_arr[i]; \
67         __auto_type __temp_cpy__ = COPY(input_arr[i]); \
68         __temp_arr__->arr[i] = __temp_cpy__; \
69         if (!__temp_elem__->bound) { \
70             DESTRUCT(__temp_elem__); \
71         } \
72     } \
73     __temp_arr__; \
74 }
75 #endif

```

**./c\_files/headers/sen\_basic\_type.h**

```

1 #include "sen_object.h"
2
3 #ifndef SEN_BASIC_TYPE_H
4 #define SEN_BASIC_TYPE_H
5
6 struct Sen_basic_type_vtable;
7 typedef struct Sen_basic_type_vtable Sen_basic_type_vtable;
8

```

```

9 struct Sen_basic_type_class;
10 typedef struct Sen_basic_type_class Sen_basic_type_class;
11
12 struct Sen_basic_type;
13 typedef struct Sen_basic_type Sen_basic_type;
14
15 typedef enum {BOOL, INT, STR, UNK} Type;
16
17 struct Sen_basic_type_vtable {
18     void (*print) (Sen_object *);
19     Sen_basic_type *(*construct) (void *);
20     void *(*get_val) (Sen_basic_type *);
21     void *(*set_val) (Sen_basic_type *, void *);
22     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
23 };
24
25 struct Sen_basic_type_class {
26     Sen_object_class *superp;
27     Sen_basic_type_vtable *tablep;
28     Type type;
29 };
30
31 struct Sen_basic_type {
32     bool bound;
33     Sen_basic_type_class *classp;
34     Sen_object *superp;
35 };
36
37 extern Sen_basic_type_class Sen_basic_type_class_;
38 extern Sen_basic_type_vtable Sen_basic_type_vtable_;
39
40 void * get_val_basic_type (Sen_basic_type *);
41 void * set_val_basic_type (Sen_basic_type *, void *);
42 Sen_basic_type * add_basic_type (Sen_basic_type *, Sen_basic_type *);
43
44 //#define ADD_BASIC_TYPE(x,y, target) x->classp->tablep->add((
45     Sen_basic_type *)x, (Sen_basic_type *)y)
45 #define ADD_BASIC_TYPE(x,y) ({                                \
46     __auto_type __temp__ = x;\                             \
47     __temp__->classp->tablep->add((Sen_basic_type *)__temp__, ( \
48         Sen_basic_type *)y);\
49 })
50
50 #endif

```

./c\_files/headers/sen\_board.h

```

1 #include "sen_object.h"
2 #include "sen_basic_type.h"
3 #include "sen_int.h"
4 #include "sen_array.h"
5
6 #ifndef SEN_BOARD_H
7 #define SEN_BOARD_H
8

```

```

9 struct Sen_board_vtable;
10 typedef struct Sen_board_vtable Sen_board_vtable;
11
12 struct Sen_board_class;
13 typedef struct Sen_board_class Sen_board_class;
14
15 struct Sen_board;
16 typedef struct Sen_board Sen_board;
17
18 struct Sen_board_vtable {
19     void (*print) (Sen_object *);
20     //Sen_board *(*construct) (Sen_object **, int);
21     Sen_board *(*construct) (int);
22     void (*destruct) (Sen_board *);
23     Sen_board *(*copy) (Sen_board *);
24     int *(*index) (Sen_array *);
25 };
26
27 struct Sen_board_class {
28     Sen_object_class *superp;
29     Sen_board_vtable *tablep;
30 };
31
32 struct Sen_board {
33     bool bound;
34     Sen_board_class *classp;
35     Sen_object *superp;
36     Sen_array *data;
37     int len;
38     char print_sep;
39 };
40
41 extern Sen_board_class Sen_board_class_;
42 extern Sen_board_vtable Sen_board_vtable_;
43
44 //Sen_board *construct_board (Sen_object **, int);
45 Sen_board *construct_board (int);
46 int board_index (Sen_array *);
47
48 #define CONSTRUCT_BOARD(array) ({
49     __auto_type input_arr = array;
50     __auto_type __temp_board__ = construct_board(input_arr->len);
51     \
52     __temp_board__->len=input_arr->len;
53     printf("OKAY\n");
54     __temp_board__->data=COPY(input_arr);
55     if (!input_arr->bound) {
56         DESTRUCT(input_arr);
57     }
58     __temp_board__;
59 })
59 #endif

```

**./c\_files/headers/sen\_board\_square.h**

```

1 #include "sen_object.h"
2 #include "sen_basic_type.h"
3 #include "sen_int.h"
4 #include "sen_array.h"
5
6 #ifndef SEN_BOARD_SQUARE_H
7 #define SEN_BOARD_SQUARE_H
8
9 struct Sen_board_square_vtable;
10 typedef struct Sen_board_square_vtable Sen_board_square_vtable;
11
12 struct Sen_board_square_class;
13 typedef struct Sen_board_square_class Sen_board_square_class;
14
15 struct Sen_board_square;
16 typedef struct Sen_board_square Sen_board_square;
17
18 struct Sen_board_square_vtable {
19     void (*print) (Sen_object *);
20     //Sen_board_square *(*construct) (Sen_object **, int);
21     Sen_board_square *(*construct) (int);
22     void (*destruct) (Sen_board_square *);
23     Sen_board_square *(*copy) (Sen_board_square *);
24     int (*index) (Sen_array *, int);
25 };
26
27 struct Sen_board_square_class {
28     Sen_object_class *superp;
29     Sen_board_square_vtable *tablep;
30 };
31
32 struct Sen_board_square {
33     bool bound;
34     Sen_board_square_class *classp;
35     Sen_object *superp;
36     Sen_array *data;
37     int len;
38     char print_sep;
39 };
40
41 extern Sen_board_square_class Sen_board_square_class_;
42 extern Sen_board_square_vtable Sen_board_square_vtable_;
43
44 //Sen_board_square *construct_board_square (Sen_object **, int);
45 Sen_board_square *construct_board_square (int);
46 int board_square_index (Sen_array *);
47
48 #define CONSTRUCT_BOARD_SQUARE(array) ({
49     __auto_type input_arr = array;
50     __auto_type __temp_board_square__ = construct_board_square(
51         input_arr->len); \
52         __temp_board_square__->len=input_arr->len;
53     \

```

```

52         printf("OKAY\n");
53         __temp_board_square__->data=COPY(input_arr);
54     \
55         if (!input_arr->bound) {
56             DESTRUCT(input_arr);
57         }
58     \
59 #endif

```

**./c\_files/headers/sen\_board\_square\_tictac.h**

```

1 #include "sen_object.h"
2 #include "sen_basic_type.h"
3 #include "sen_int.h"
4 #include "sen_array.h"
5
6 #ifndef SEN_BOARD_SQUARE_TICTAC_H
7 #define SEN_BOARD_SQUARE_TICTAC_H
8
9 struct Sen_board_square_tictac_vtable;
10 typedef struct Sen_board_square_tictac_vtable
11     Sen_board_square_tictac_vtable;
12
13 struct Sen_board_square_tictac_class;
14 typedef struct Sen_board_square_tictac_class Sen_board_square_tictac_class
15 ;
16
17 struct Sen_board_square_tictac;
18 typedef struct Sen_board_square_tictac Sen_board_square_tictac;
19
20 struct Sen_board_square_tictac_vtable {
21     void (*print) (Sen_object *);
22     //Sen_board_square_tictac *(*construct) (Sen_object **, int);
23     Sen_board_square_tictac *(*construct) (int);
24     void (*destruct) (Sen_board_square_tictac *);
25     Sen_board_square_tictac *(*copy) (Sen_board_square_tictac *);
26     int (*index) (Sen_array *, int);
27 };
28
29 struct Sen_board_square_tictac_class {
30     Sen_object_class *superp;
31     Sen_board_square_tictac_vtable *tablep;
32 };
33
34 struct Sen_board_square_tictac {
35     bool bound;
36     Sen_board_square_tictac_class *classp;
37     Sen_object *superp;
38     Sen_array *data;
39     int len;
40     char print_sep;
41 };

```

```

41 extern Sen_board_square_tictac_class Sen_board_square_tictac_class_;
42 extern Sen_board_square_tictac_vtable Sen_board_square_tictac_vtable_;
43
44 //Sen_board_square_tictac *construct_board_square_tictac (Sen_object **,
45 //int);
45 Sen_board_square_tictac *construct_board_square_tictac (int);
46 int board_square_tictac_index (Sen_array *);
47
48 #define CONSTRUCT_BOARD_SQUARE_TICTAC(array) ({ \
49     __auto_type input_arr = array; \
50     __auto_type __temp_board_square_tictac__ = \
51         construct_board_square_tictac(input_arr->len); \
52     __temp_board_square_tictac__->len=input_arr->len; \
53     printf("OKAY\n"); \
54     __temp_board_square_tictac__->data=COPY(input_arr); \
55     if (!input_arr->bound) { \
56         DESTRUCT(input_arr); \
57     } \
58     __temp_board_square_tictac__; \
59 })
59 #endif

```

./c\_files/headers/sen\_bool.h

```

1 #include "sen_basic_type.h"
2
3 #ifndef SEN_BOOL_H
4 #define SEN_BOOL_H
5
6 struct Sen_bool_vtable;
7 typedef struct Sen_bool_vtable Sen_bool_vtable;
8
9 struct Sen_bool_class;
10 typedef struct Sen_bool_class Sen_bool_class;
11
12 struct Sen_bool;
13 typedef struct Sen_bool Sen_bool;
14
15 struct Sen_bool_vtable {
16     void (*print) (Sen_object *);
17     void *(*get_val) (Sen_basic_type *);
18     void *(*set_val) (Sen_basic_type *, void *);
19     Sen_bool *(*construct) (bool);
20     void (*destruct) (Sen_bool *);
21     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
22 };
23
24 struct Sen_bool_class {
25     Sen_basic_type_class *superp;
26     Sen_bool_vtable *tablep;
27     Type type;
28 };
29
30 struct Sen_bool {
31     bool bound;

```

```

32     Sen_bool_class *classsp;
33     Sen_basic_type *superp;
34     bool val;
35 };
36
37 extern Sen_bool_class Sen_bool_class_;
38 extern Sen_bool_vtable Sen_bool_vtable_;
39
40 void print_bool (Sen_object *);
41 Sen_bool * construct_bool (bool);
42 void *get_val_bool (Sen_basic_type *);
43 void *set_val_bool (Sen_basic_type *, void *);
44
45 #define CONSTRUCT_BOOL(val) (Sen_bool*) construct_bool(val)
46
47 #endif

```

./c\_files/headers/sen\_int.h

```

1 #include "sen_basic_type.h"
2
3 #ifndef SEN_INT_H
4 #define SEN_INT_H
5
6 struct Sen_int_vtable;
7 typedef struct Sen_int_vtable Sen_int_vtable;
8
9 struct Sen_int_class;
10 typedef struct Sen_int_class Sen_int_class;
11
12 struct Sen_int;
13 typedef struct Sen_int Sen_int;
14
15 struct Sen_int_vtable {
16     void (*print) (Sen_object *);
17     void *(*get_val) (Sen_basic_type *);
18     void *(*set_val) (Sen_basic_type *, void *);
19     Sen_int *(*construct) (int);
20     void (*destruct) (Sen_int *);
21     Sen_int *(*copy) (Sen_int *);
22     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
23 };
24
25 struct Sen_int_class {
26     Sen_basic_type_class *superp;
27     Sen_int_vtable *tablep;
28     Type type;
29 };
30
31 struct Sen_int {
32     bool bound;
33     Sen_int_class *classsp;
34     Sen_basic_type *superp;
35     int val;
36 };

```

```

37
38 extern Sen_int_class Sen_int_class_;
39 extern Sen_int_vtable Sen_int_vtable_;
40
41 void print_int (Sen_object *);
42 Sen_int * construct_int (int);
43 void *get_val_int (Sen_basic_type *);
44 void *set_val_int (Sen_basic_type *, void *);
45
46 #define CONSTRUCT_INT(val) ((Sen_int*) construct_int(val))
47
48 #endif

```

**./c\_files/headers/sen\_object.h**

```

1 ///#include "stdio.h"
2 ///#include "stdlib.h"
3
4 #ifndef SEN_OBJECT_H
5 #define SEN_OBJECT_H
6
7 struct Sen_object_vtable;
8 typedef struct Sen_object_vtable Sen_object_vtable;
9
10 struct Sen_object_class;
11 typedef struct Sen_object_class Sen_object_class;
12
13 struct Sen_object;
14 typedef struct Sen_object Sen_object;
15
16 struct Sen_object_vtable {
17     void (*print) (Sen_object *);
18     Sen_object *(*construct) (void *);
19     void (*destruct) (Sen_object *);
20     Sen_object *(*copy) (Sen_object *);
21 };
22
23 //static Sen_object_vtable _Sen_object_vtable;
24
25 struct Sen_object_class {
26     void * superp;
27     Sen_object_vtable *tablep;
28 };
29
30 //static Sen_object_class _Sen_object_class;
31
32 struct Sen_object {
33     bool bound;
34     Sen_object_class *classp;
35 };
36
37 extern Sen_object_class Sen_object_class_;
38 extern Sen_object_vtable Sen_object_vtable_;
39
40 void print_object (Sen_object *);

```

```

41 Sen_object * construct_object (void *);
42 void destruct_object (Sen_object *);
43 Sen_object * copy_object (Sen_object *);
44
45 #define PRINT(self) {
46     typeof(self) __temp__ = self;
47     __temp__-> classp-> tablep-> print(((Sen_object *)__temp__));
48 }
49
50 #define DESTRUCT(self) ({
51     __auto_type __temp__ = self;
52     __temp__-> classp-> tablep-> destruct(__temp__);
53 })
54
55 #define COPY(self) ({
56     typeof(self) __temp__ = self;
57     (typeof (__temp__)) __temp__-> classp-> tablep-> copy(__temp__);
58 })
59 #endif

```

./c\_files/headers/sen\_string.h

```

1 #include "sen_basic_type.h"
2
3 #ifndef SEN_STRING_H
4 #define SEN_STRING_H
5
6 struct Sen_string_vtable;
7 typedef struct Sen_string_vtable Sen_string_vtable;
8
9 struct Sen_string_class;
10 typedef struct Sen_string_class Sen_string_class;
11
12 struct Sen_string;
13 typedef struct Sen_string Sen_string;
14
15 struct Sen_string_vtable {
16     void (*print) (Sen_object *);
17     Sen_string *(*construct) (char *);
18     void (*destruct) (Sen_string *);
19     Sen_string *(*copy) (Sen_string *);
20     void *(*get_val) (Sen_basic_type *);
21     void *(*set_val) (Sen_basic_type *, void *);
22     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
23 };
24
25 struct Sen_string_class {
26     Sen_basic_type_class *superp;
27     Sen_string_vtable *tablep;
28     Type type;
29 };
30
31 struct Sen_string {
32     bool bound;

```

```

33     Sen_string_class *classp;
34     Sen_basic_type *superp;
35     char *val;
36 };
37
38 extern Sen_string_class Sen_string_class_;
39 extern Sen_string_vtable Sen_string_vtable_;
40
41 void print_string (Sen_object *);
42 Sen_string * construct_string (char *);
43 void destruct (Sen_string *);
44 Sen_string * copy_string (Sen_string *);
45 void *get_val_string (Sen_basic_type *);
46 void *set_val_string (Sen_basic_type *, void *);
47
48
49 #define CONSTRUCT_STRING(val) ((Sen_string*) construct_string(val))
50
51 #endif

```

**./c\_files/headers/sen\_tictac.h**

```

1 struct GameBoard
2 {
3     int length;
4     int width;
5     int *board;
6     int game_step;
7 };
8 typedef struct GameBoard GameBoard;
9
10 //Functions
11 GameBoard* init_board( int length , int width);
12 void display(GameBoard);
13 int board_evaluate(GameBoard);
14 int valid_turn(GameBoard,int x,int y);
15 void turn(GameBoard* game_board);
16 void reset_game(GameBoard* game_board);

```

**./c\_files/main.c**

```

1 #include "headers/all_headers.h"
2
3 int main() {
4     __auto_type x = CONSTRUCT_INT(100);
5     x->bound = true;
6     __auto_type f = CONSTRUCT_INT(50);
7     f->bound=true;
8     printf("%d\n", x->val);
9     PRINT(x);
10    printf("\n");
11    PRINT(f);
12    printf("%d %d\n", x->bound , f->bound);
13    {
14        typeof(x) __temp__ = (typeof(x)) ADD_BASIC_TYPE(x, ((Sen_int *)
ADD_BASIC_TYPE(x, f)));

```

```

15     free(x);
16     x = __temp__;
17     x->bound=true;
18 }
19 printf("%d %d\n", x->bound, f->bound);
20 x->bound=true;
21 PRINT(x);
22 printf("\n");
23 printf("%d %d\n", x->bound, f->bound);
24 DESTRUCT(x);
25 DESTRUCT(f);
26 __auto_type s = CONSTRUCT_STRING("tttestingggg ");
27 s->bound=true;
28 __auto_type ss = CONSTRUCT_STRING("hooray!!\n");
29 ss->bound=true;
30 //PRINT(arr_[0]);
31
32 PRINT(((Sen_string *) ss));
33 PRINT(((Sen_string *) ADD_BASIC_TYPE(s, ss)));
34 PRINT(((Sen_string *) ADD_BASIC_TYPE(((Sen_string *) ADD_BASIC_TYPE(s,
ss)), ((Sen_string *) ADD_BASIC_TYPE(s, ss)))));
35 DESTRUCT(((Sen_string *) s));
36 DESTRUCT(((Sen_string *) ss));
37 //Sen_array *arr = (Sen_array_vtable_.construct(arr_, ARRAY_SIZE(arr_))
);
38 //Sen_array *arr = (Sen_array_vtable_.construct((Sen_int*[]){
CONSTRUCT_INT(100), CONSTRUCT_INT(50)}, 2));
39 Sen_array *arr = CONSTRUCT_ARRAY(((Sen_int*[]){
CONSTRUCT_INT(1900),
CONSTRUCT_INT(50)}), 2);
40 arr->bound = true;
41 //Sen_array *arr = CONSTRUCT_ARRAY(arr_, 2);
42 __auto_type xx = CONSTRUCT_INT(123);
43 xx->bound=true;
44 __auto_type yy = COPY(xx);
45 PRINT(yy);
46 PRINT(xx);
47 DESTRUCT(xx);
48 printf("%d %d asdsad\n", ((Sen_int *)((arr->arr)[0]))->val, arr->len);
49 printf("OKAY\n");
50 __auto_type xxx = arr->arr[1];
51 PRINT(((Sen_int *)xxx));
52 printf("\nOKAY\n");
53 PRINT(xxx);
54 printf("\nOKAY\n");

55
56 printf("OKAY\n");
57 PRINT((arr->arr)[1]);
58 printf("OKAY\n");
59 //DESTRUCT(xxx);
60 Sen_board *board = CONSTRUCT_BOARD(arr);
61 printf("%d %d\n", ((Sen_int *)((arr->arr)[0]))->val, arr->len);
62 //DESTRUCT(arr);
63 printf("OKAYFINALS\n");
64

```

```

65 //DESTRUCT(board);
66 //printf("OKAY\n");
67 return 0;
68 }
```

**./c\_files/Makefile**

```

1 CC=gcc
2 CFLAGS= -Wall -g
3
4 INC= -I ./headers
5
6 SRC=$(wildcard *.c)
7 SRC:=$(filter-out tictactoe.c, $(SRC))
8
9 all: main
10
11 main: $(SRC)
12     $(CC) -o out $(CFLAGS) $(INC) $(SRC)
13
14 clean:
15     rm -f *.o out
```

**./c\_files/sen\_array.c**

```

1 #include "headers/all_headers.h"
2
3 /*
4 Sen_array *construct_array(Sen_object *val[], int len) {
5     Sen_array *ret = malloc(sizeof(Sen_array));
6     ret->len = len;
7     printf("%d %d\n", (int)ret->len, (int)sizeof(Sen_object *));
8     ret->arr = malloc(sizeof(typeof(val[0])) * (len));
9     for (int i=0; i<len; i++) {
10         (ret->arr)[i] = COPY(((typeof(val[i])) val[i]));
11         (ret->arr)[i]->bound=true;
12         if (!val[i]->bound) {
13             DESTRUCT(val[i]);
14         }
15     }
16     ret->bound=false;
17     return ret;
18 }
19 */
20
21 Sen_array *construct_array(int len) {
22     Sen_array *ret = malloc(sizeof(Sen_array));
23     ret->len = len;
24     ret->classp = &Sen_array_class_;
25     ret->bound = false;
26     ret->arr = malloc(sizeof(Sen_object *) * len);
27     ret->print_sep=' ';
28     return ret;
29 }
30
31 void destruct_array(Sen_array *self) {
```

```

32     for (int i=0; i<self->len; i++) {
33         free(self->arr[i]);
34     }
35     free(self);
36 }
37
38 Sen_array *copy_array(Sen_array *other) {
39     Sen_array *ret = construct_array(other->len);
40     for (int i=0; i<ret->len; i++) {
41         //printf("%d %d\n", other->len, ((Sen_int *) (other->arr[i]))->val)
42         ;
43         //printf("OKAY1\n");
44         ret->arr[i] = COPY(((other->arr)[i]));
45         ret->arr[i]->bound=true;
46         //printf("OKAY2\n");
47     }
48     //printf("OKAY1\n");
49     return ret;
50 }
51
52 Sen_object *access_array(Sen_array *self, Sen_int *index) {
53     return self->arr[index->val];
54 }
55
56 Sen_array *add_array(Sen_array *x, Sen_array *y) {
57     // Need to check that types are the same
58     Sen_array *ret = malloc(sizeof(ret));
59     *ret=*x;
60     return ret;
61 }
62
63 Sen_array_vtable Sen_array_vtable_ = {
64     print_object,
65     construct_array,
66     destruct_array,
67     copy_array,
68     access_array,
69     add_array
70 };
71
72 Sen_array_class Sen_array_class_ = {
73     &Sen_object_class_,
74     &Sen_array_vtable_,
75 };

```

**./c\_files/sen\_array\_int.c**

```

1 #include "headers/all_headers.h"
2
3 /*
4  Sen_array_int *construct_array_int(Sen_object *val[], int len) {
5      Sen_array_int *ret = malloc (sizeof(Sen_array_int));
6      ret->len = len;
7      printf("%d %d\n", (int)ret->len, (int)sizeof(Sen_object *));
8      ret->arr = malloc(sizeof(typeof (val[0])) * (len));

```

```

9   for (int i=0; i<len; i++) {
10     (ret->arr)[i] = COPY(((typeof (val[i])) val[i]));
11     (ret->arr)[i]->bound=true;
12     if (!val[i]->bound) {
13       DESTRUCT(val[i]);
14     }
15   }
16   ret->bound=false;
17   return ret;
18 }
*/
20
21 Sen_array_int *construct_array_int(int len) {
22   Sen_array_int *ret = malloc(sizeof(Sen_array_int));
23   ret->len = len;
24   ret->classp = &Sen_array_int_class_;
25   ret->bound = false;
26   ret->arr = malloc(sizeof(Sen_int *) * len);
27   ret->print_sep=' ';
28   return ret;
29 }
30
31 void destruct_array_int(Sen_array_int *self) {
32   for (int i=0; i<self->len; i++) {
33     free(self->arr[i]);
34   }
35   free(self);
36 }
37
38 Sen_array_int *copy_array_int(Sen_array_int *other) {
39   Sen_array_int *ret = construct_array_int(other->len);
40   for (int i=0; i<ret->len; i++) {
41     printf("%d %d\n", other->len, ((Sen_int *) (other->arr[i]))->val);
42     printf("OKAY1\n");
43     ret->arr[i] = COPY(((other->arr)[i]));
44     printf("OKAY2\n");
45   }
46   printf("OKAY1\n");
47   return ret;
48 }
49
50 Sen_int *access_array_int(Sen_array_int *self, Sen_int *index) {
51   return self->arr[index->val];
52 }
53
54 Sen_array_int *add_array_int(Sen_array_int *x, Sen_array_int *y) {
55   // Need to check that types are the same
56   Sen_array_int *ret = malloc(sizeof(ret));
57   *ret=*x;
58   return ret;
59 }
60
61 Sen_array_int_vtable Sen_array_int_vtable_ = {
62   print_object,

```

```

63     construct_array_int ,
64     destruct_array_int ,
65     copy_array_int ,
66     access_array_int ,
67     add_array_int
68 };
69
70 Sen_array_int_class Sen_array_int_class_ = {
71     &Sen_object_class_ ,
72     &Sen_array_int_vtable_ ,
73 };

```

**./c\_files/sen\_basic\_type.c**

```

1 #include "headers/all_headers.h"
2
3 void *get_val_basic_type(Sen_basic_type *self) {
4     int *ret = malloc(sizeof(*ret));
5     *ret=((Sen_int *) self)->val;
6     return ret;
7 }
8
9 void *set_val_basic_type(Sen_basic_type *self, void *val) {
10
11     //self->val=val;
12     return val;
13 }
14
15 Sen_basic_type *construct_basic_type(void *val) {
16     Sen_basic_type *ret = malloc(sizeof(ret));
17     return ret;
18 }
19
20 Sen_basic_type *add_basic_type(Sen_basic_type *x, Sen_basic_type *y) {
21     // Need to check that types are the same
22     Sen_basic_type *ret = malloc(sizeof(ret));
23     *ret=**x;
24     return ret;
25 }
26
27 Sen_basic_type_vtable Sen_basic_type_vtable_ = {
28     print_object ,
29     construct_basic_type ,
30     get_val_basic_type ,
31     set_val_basic_type ,
32     add_basic_type
33 };
34
35 Sen_basic_type_class Sen_basic_type_class_ = {
36     &Sen_object_class_ ,
37     &Sen_basic_type_vtable_ ,
38     UNK
39 };

```

**./c\_files/sen\_board.c**

```

1 #include "headers/all_headers.h"
2
3 /*
4 Sen_board *construct_board(Sen_object *val[], int len) {
5     Sen_board *ret = malloc(sizeof(Sen_board));
6     ret->len = len;
7     printf("%d %d\n", (int)ret->len, (int)sizeof(Sen_object *));
8     ret->arr = malloc(sizeof(typeof(val[0])) * (len));
9     for (int i=0; i<len; i++) {
10         (ret->arr)[i] = COPY(((typeof(val[i])) val[i]));
11         (ret->arr)[i]->bound=true;
12         if (!val[i]->bound) {
13             DESTRUCT(val[i]);
14         }
15     }
16     ret->bound=false;
17     return ret;
18 }
*/
20
21 Sen_board *construct_board(int len) {
22     Sen_board *ret = malloc(sizeof(Sen_board));
23     ret->len = len;
24     ret->bound = false;
25     ret->data = construct_array(len);
26     ret->print_sep = ',';
27     printf("hi\n");
28     return ret;
29 }
30
31 void destruct_board(Sen_board *self) {
32     DESTRUCT(self->data);
33     free(self);
34 }
35
36 Sen_board *copy_board(Sen_board *other) {
37     Sen_board *ret = malloc(sizeof(Sen_board));
38     ret->len = other->len;
39     ret->data = COPY(other->data);
40     return ret;
41 }
42
43 Sen_board_vtable Sen_board_vtable_ = {
44     print_object,
45     construct_board,
46     destruct_board,
47 };
48
49 Sen_board_class Sen_board_class_ = {
50     &Sen_object_class_,
51     &Sen_board_vtable_,
52 };

```

./c\_files/sen\_board\_square.c

```

1 #include "headers/all_headers.h"
2
3 /*
4 Sen_board_square *construct_board_square(Sen_object *val[], int len) {
5     Sen_board_square *ret = malloc(sizeof(Sen_board_square));
6     ret->len = len
7     printf("%d %d\n", (int)ret->len, (int)sizeof(Sen_object *));
8     ret->arr = malloc(sizeof(typeof(val[0])) * (len));
9     for (int i=0; i<len; i++) {
10         (ret->arr)[i] = COPY(((typeof(val[i])) val[i]));
11         (ret->arr)[i]->bound=true;
12         if (!val[i]->bound) {
13             DESTRUCT(val[i]);
14         }
15     }
16     ret->bound=false;
17     return ret;
18 }
*/
20
21 Sen_board_square *construct_board_square(int len) {
22     Sen_board_square *ret = malloc(sizeof(Sen_board_square));
23     ret->classp = &Sen_board_square_class_;
24     ret->len = len;
25     ret->bound = false;
26     ret->data = construct_array(len);
27     ret->print_sep = ',';
28     printf("hi\n");
29     return ret;
30 }
31
32 void destruct_board_square(Sen_board_square *self) {
33     DESTRUCT(self->data);
34     free(self);
35 }
36
37
38 Sen_board_square *copy_board_square(Sen_board_square *other) {
39     Sen_board_square *ret = malloc(sizeof(Sen_board_square));
40     ret->len = other->len;
41     ret->data = COPY(other->data);
42     return ret;
43 }
44
45 int index_board_square(Sen_array *coord, int len) {
46     int x = ((Sen_int *)coord->arr[0])->val;
47     int y = ((Sen_int *)coord->arr[1])->val;
48     return x + y % len;
49 }
50
51 Sen_board_square_vtable Sen_board_square_vtable_ = {
52     print_object,
53     construct_board_square,
54     destruct_board_square,

```

```

55     copy_board_square,
56     index_board_square
57 };
58
59 Sen_board_square_class Sen_board_square_class_ = {
60     &Sen_object_class_,
61     &Sen_board_square_vtable_,
62 };

```

**./c\_files/sen\_bool.c**

```

1 #include "headers/all_headers.h"
2
3 Sen_bool *construct_bool(bool val) {
4     Sen_bool *ret = malloc(sizeof(Sen_bool));
5     ret->classp = &Sen_bool_class_;
6     ret->val = val;
7     ret->bound = false;
8     return ret;
9 }
10
11 void destruct_bool(Sen_bool *self) {
12     free(self);
13 }
14
15 void print_bool(Sen_object *self) {
16     if (((Sen_bool*) self)->val==true) {
17         printf("True");
18     } else {
19         printf("False");
20     }
21     if (!((Sen_bool *)self)->bound) {
22         free(self);
23     }
24 }
25
26 void *get_val_bool(Sen_basic_type *self) {
27     bool *ret = malloc(sizeof *ret);
28     *ret=*((Sen_bool *) self)->val;
29     if (!((Sen_bool *)self)->bound) {
30         free(self);
31     }
32     return ret;
33 }
34
35 void *set_val_bool(Sen_basic_type *self, void *val) {
36     ((Sen_bool *)self)->val=*(bool*)val;
37     if (!((Sen_bool *)self)->bound) {
38         free(self);
39     }
40     return val;
41 }
42
43 Sen_basic_type *add_bool(Sen_basic_type *x, Sen_basic_type *y) {
44     // Need to check types for safety

```

```

45     Sen_bool *ret = construct_bool(((Sen_bool *)x)->val);
46     ret->val = ret->val != ((Sen_bool *) y)->val;
47     if (!x->bound) {
48         free(x);
49     }
50     if (!y->bound) {
51         free(y);
52     }
53     return (Sen_basic_type *) ret;
54 }
55
56 Sen_bool_vtable Sen_bool_vtable_ = {
57     print_bool,
58     get_val_bool,
59     set_val_bool,
60     construct_bool,
61     destruct_bool,
62     add_bool
63 };
64
65 //Sen_basic_type_class temp; /* NEED TO FIX */
66 Sen_bool_class Sen_bool_class_ = {
67     &Sen_basic_type_class_,
68     &Sen_bool_vtable_,
69     BOOL
70 };

```

./c\_files/sen\_int.c

```

1 #include "headers/all_headers.h"
2
3 Sen_int *construct_int(int val) {
4     Sen_int *ret = malloc(sizeof(Sen_int));
5     ret->classp = &Sen_int_class_;
6     ret->val = val;
7     ret->bound = false;
8     return ret;
9 }
10
11 void destruct_int(Sen_int *self) {
12     free(self);
13 }
14
15 Sen_int *copy_int(Sen_int *self) {
16     Sen_int *ret = construct_int(self->val);
17     return ret;
18 }
19
20 void print_int(Sen_object *self) {
21     printf("%d", ((Sen_int *) self)->val);
22     if (!((Sen_int *) self)->bound) {
23         free(self);
24     }
25 }
26

```

```

27 void *get_val_int(Sen_basic_type *self) {
28     int *ret = malloc(sizeof *ret);
29     *ret=((Sen_int *) self)->val;
30     if (!self->bound) {
31         free(self);
32     }
33     return ret;
34 }
35
36 void *set_val_int(Sen_basic_type *self, void *val) {
37     ((Sen_int *)self)->val=*(int*)val;
38     if (!self->bound) {
39         free(self);
40     }
41     return val;
42 }
43
44 Sen_basic_type *add_int(Sen_basic_type *x, Sen_basic_type *y) {
45     // Need to check types for safety
46     Sen_int *ret = construct_int(((Sen_int *)x)->val);
47     ret->val+=((Sen_int *) y)->val;
48     if (!x->bound) {
49         free(x);
50     }
51     if (!y->bound) {
52         free(y);
53     }
54     return (Sen_basic_type *) ret;
55 }
56
57 Sen_int_vtable Sen_int_vtable_ = {
58     print_int,
59     get_val_int,
60     set_val_int,
61     construct_int,
62     destruct_int,
63     copy_int,
64     add_int
65 };
66
67 //Sen_basic_type_class temp; /* NEED TO FIX */
68 Sen_int_class Sen_int_class_ = {
69     &Sen_basic_type_class_,
70     &Sen_int_vtable_,
71     INT
72 };

```

**./c\_files/sen\_object.c**

```

1 #include "headers/all_headers.h"
2
3 void print_object(Sen_object *self) {
4
5 }
6

```

```

7 Sen_object *construct_object(void *val) {
8     Sen_object *ret = malloc(sizeof(Sen_object *));
9     return ret;
10 }
11
12 void destruct_object(Sen_object *val) {
13     free(val);
14 }
15
16 Sen_object *copy_object(Sen_object *self) {
17     Sen_object *ret = malloc(sizeof(Sen_object *));
18     ret->classp = &Sen_object_class_;
19     return ret;
20 }
21
22 Sen_object_vtable Sen_object_vtable_ = {
23     print_object,
24     construct_object,
25     destruct_object,
26     copy_object
27 };
28
29 Sen_object_class Sen_object_class_ = {
30     NULL,
31     &Sen_object_vtable_
32 };

```

**./c\_files/sen\_string.c**

```

1 #include "headers/all_headers.h"
2
3 Sen_string *construct_string(char *val) {
4     int l = strlen(val);
5     Sen_string *ret = malloc(sizeof(Sen_string));
6     ret->classp = &Sen_string_class_;
7     ret->val = (char *)malloc(l+1);
8     strncpy(ret->val, val, l+1);
9     ret->bound = false;
10    return ret;
11 }
12
13 void destruct_string(Sen_string *self) {
14     free(self->val);
15     self->val = NULL;
16     free(self);
17     self=NULL;
18 }
19
20
21 Sen_string *copy_string(Sen_string *other) {
22     return construct_string(other->val);
23 }
24
25 void print_string(Sen_object *self) {
26     printf("%s", ((Sen_string *) self)->val);

```

```

27     if (!self->bound) {
28         destruct_string((Sen_string *) self);
29     }
30 }
31
32 void *get_val_string(Sen_basic_type *self) {
33     char **ret = malloc(sizeof *ret);
34     *ret=((Sen_string *) self)->val;
35     if (!self->bound) {
36         destruct_string((Sen_string *) self);
37     }
38     return ret;
39 }
40
41 void *set_val_string(Sen_basic_type *self, void *val) {
42     ((Sen_string *)self)->val=*(char **)val;
43     if (!self->bound) {
44         destruct_string((Sen_string *) self);
45     }
46     return val;
47 }
48
49 Sen_basic_type *add_string(Sen_basic_type *x, Sen_basic_type *y) {
50     // Need to check types for safety
51     char *new_string = malloc(strlen(((Sen_string*)x)->val) + strlen(((Sen_string*)y)->val) + 1);
52     strcpy(new_string, ((Sen_string*) x)->val);
53     strcat(new_string, ((Sen_string*) y)->val);
54     Sen_string *ret = construct_string(new_string);
55     free(new_string);
56     if (!x->bound) {
57         destruct_string((Sen_string*) x);
58     }
59     if (!y->bound) {
60         destruct_string((Sen_string*) y);
61     }
62     return (Sen_basic_type *) ret;
63 }
64
65 Sen_string_vtable Sen_string_vtable_ = {
66     print_string,
67     construct_string,
68     destruct_string,
69     copy_string,
70     get_val_string,
71     set_val_string,
72     add_string
73 };
74
75 //Sen_basic_type_class temp; /* NEED TO FIX */
76 Sen_string_class Sen_string_class_ = {
77     &Sen_basic_type_class_,
78     &Sen_string_vtable_,
79     STR

```

```

80 };
./c_files/temp

1 # 1 "main.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 1 "/usr/include/stdc-predef.h" 1 3 4
5 # 1 "<command-line>" 2
6 # 1 "main.c"
7 # 1 "headers/all_headers.h" 1
8
9
10 # 1 "/usr/include/stdlib.h" 1 3 4
11 # 24 "/usr/include/stdlib.h" 3 4
12 # 1 "/usr/include/features.h" 1 3 4
13 # 374 "/usr/include/features.h" 3 4
14 # 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
15 # 385 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
16 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
17 # 386 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
18 # 375 "/usr/include/features.h" 2 3 4
19 # 398 "/usr/include/features.h" 3 4
20 # 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4
21 # 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4
22 # 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4
23 # 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4
24 # 399 "/usr/include/features.h" 2 3 4
25 # 25 "/usr/include/stdlib.h" 2 3 4
26
27
28
29
30
31
32
33 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
34 # 216 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 3 4
35
36 # 216 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 3 4
37 typedef long unsigned int size_t;
38 # 328 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 3 4
39 typedef int wchar_t;
40 # 33 "/usr/include/stdlib.h" 2 3 4
41
42
43
44
45
46
47
48
49 # 1 "/usr/include/x86_64-linux-gnu/bits/waitflags.h" 1 3 4
50 # 50 "/usr/include/x86_64-linux-gnu/bits/waitflags.h" 3 4
51 typedef enum

```

```

52 {
53     P_ALL,
54     P_PID,
55     P_PGID
56 } idtype_t;
57 # 42 "/usr/include/stdlib.h" 2 3 4
58 # 1 "/usr/include/x86_64-linux-gnu/bits/waitstatus.h" 1 3 4
59 # 64 "/usr/include/x86_64-linux-gnu/bits/waitstatus.h" 3 4
60 # 1 "/usr/include/endian.h" 1 3 4
61 # 36 "/usr/include/endian.h" 3 4
62 # 1 "/usr/include/x86_64-linux-gnu/bits/endian.h" 1 3 4
63 # 37 "/usr/include/endian.h" 2 3 4
64 # 60 "/usr/include/endian.h" 3 4
65 # 1 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 1 3 4
66 # 27 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 3 4
67 # 1 "/usr/include/x86_64-linux-gnu/bits/types.h" 1 3 4
68 # 27 "/usr/include/x86_64-linux-gnu/bits/types.h" 3 4
69 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
70 # 28 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
71
72
73 typedef unsigned char __u_char;
74 typedef unsigned short int __u_short;
75 typedef unsigned int __u_int;
76 typedef unsigned long int __u_long;
77
78
79 typedef signed char __int8_t;
80 typedef unsigned char __uint8_t;
81 typedef signed short int __int16_t;
82 typedef unsigned short int __uint16_t;
83 typedef signed int __int32_t;
84 typedef unsigned int __uint32_t;
85
86 typedef signed long int __int64_t;
87 typedef unsigned long int __uint64_t;
88
89
90
91
92
93
94
95 typedef long int __quad_t;
96 typedef unsigned long int __u_quad_t;
97 # 121 "/usr/include/x86_64-linux-gnu/bits/types.h" 3 4
98 # 1 "/usr/include/x86_64-linux-gnu/bits/typesizes.h" 1 3 4
99 # 122 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
100
101
102 typedef unsigned long int __dev_t;
103 typedef unsigned int __uid_t;
104 typedef unsigned int __gid_t;
105 typedef unsigned long int __ino_t;

```

```

106 typedef unsigned long int __ino64_t;
107 typedef unsigned int __mode_t;
108 typedef unsigned long int __nlink_t;
109 typedef long int __off_t;
110 typedef long int __off64_t;
111 typedef int __pid_t;
112 typedef struct { int __val[2]; } __fsid_t;
113 typedef long int __clock_t;
114 typedef unsigned long int __rlim_t;
115 typedef unsigned long int __rlim64_t;
116 typedef unsigned int __id_t;
117 typedef long int __time_t;
118 typedef unsigned int __useconds_t;
119 typedef long int __suseconds_t;
120
121 typedef int __daddr_t;
122 typedef int __key_t;
123
124
125 typedef int __clockid_t;
126
127
128 typedef void * __timer_t;
129
130
131 typedef long int __blksize_t;
132
133
134
135
136 typedef long int __blkcnt_t;
137 typedef long int __blkcnt64_t;
138
139
140 typedef unsigned long int __fsblkcnt_t;
141 typedef unsigned long int __fsblkcnt64_t;
142
143
144 typedef unsigned long int __fsfilcnt_t;
145 typedef unsigned long int __fsfilcnt64_t;
146
147
148 typedef long int __fword_t;
149
150 typedef long int __ssize_t;
151
152
153 typedef long int __syscall_slong_t;
154
155 typedef unsigned long int __syscall_ulong_t;
156
157
158
159 typedef __off64_t __loff_t;

```

```

160 typedef __quad_t *__qaddr_t;
161 typedef char *__caddr_t;
162
163
164 typedef long int __intptr_t;
165
166
167 typedef unsigned int __socklen_t;
168 # 28 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 2 3 4
169 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
170 # 29 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 2 3 4
171
172
173
174
175
176
177 # 1 "/usr/include/x86_64-linux-gnu/bits/bytesswap-16.h" 1 3 4
178 # 36 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 2 3 4
179 # 44 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 3 4
180 static __inline unsigned int
181 __bswap_32 (unsigned int __bsx)
182 {
183     return __builtin_bswap32 (__bsx);
184 }
185 # 108 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 3 4
186 static __inline __uint64_t
187 __bswap_64 (__uint64_t __bsx)
188 {
189     return __builtin_bswap64 (__bsx);
190 }
191 # 61 "/usr/include/endian.h" 2 3 4
192 # 65 "/usr/include/x86_64-linux-gnu/bits/waitstatus.h" 2 3 4
193
194 union wait
195 {
196     int w_status;
197     struct
198     {
199
200         unsigned int __w_termsig:7;
201         unsigned int __w_coredump:1;
202         unsigned int __w_retcode:8;
203         unsigned int :16;
204
205
206
207
208
209
210
211         } __wait_terminated;
212     struct
213     {

```

```

214
215     unsigned int __w_stopval:8;
216     unsigned int __w_stopsig:8;
217     unsigned int :16;
218
219
220
221
222
223
224         } __wait_stopped;
225     };
226 # 43 "/usr/include/stdlib.h" 2 3 4
227 # 67 "/usr/include/stdlib.h" 3 4
228 typedef union
229 {
230     union wait *__uptr;
231     int * __iptr;
232 } __WAIT_STATUS __attribute__((__transparent_union__));
233 # 95 "/usr/include/stdlib.h" 3 4
234
235
236 typedef struct
237 {
238     int quot;
239     int rem;
240 } div_t;
241
242
243
244 typedef struct
245 {
246     long int quot;
247     long int rem;
248 } ldiv_t;
249
250
251
252
253
254
255
256 __extension__ typedef struct
257 {
258     long long int quot;
259     long long int rem;
260 } lldiv_t;
261
262
263 # 139 "/usr/include/stdlib.h" 3 4
264 extern size_t __ctype_get_mb_cur_max (void) __attribute__((__nothrow__, __leaf__));
265
266

```

```

267
268
269 extern double atof (const char *__nptra)
270     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
271     __attribute__ ((__nonnull__(1))) ;
272
273 extern int atoi (const char *__nptra)
274     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
275     __attribute__ ((__nonnull__(1))) ;
276
277
278
279
280
281
282 __extension__ extern long long int atoll (const char *__nptra)
283     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
284     __attribute__ ((__nonnull__(1))) ;
285
286
287
288
289 extern double strtod (const char *__restrict __nptra,
290     char **__restrict __endptr)
291     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__(1)));
292
293
294
295
296
297 extern float strtodf (const char *__restrict __nptra,
298     char **__restrict __endptr) __attribute__ ((__nothrow__, __leaf__))
299     __attribute__ ((__nonnull__(1)));
300
301 extern long double strtold (const char *__restrict __nptra,
302     char **__restrict __endptr)
303     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__(1)));
304
305
306
307
308 extern long int strtol (const char *__restrict __nptra,
309     char **__restrict __endptr, int __base)
310     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__(1)));
311
312 extern unsigned long int strtoul (const char *__restrict __nptra,

```

```

313     char **__restrict __endptr, int __base)
314     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
315     (1)));
316
317
318
319 --extension--
320 extern long long int strtoll (const char *__restrict __nptr,
321     char **__restrict __endptr, int __base)
322     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
323     (1)));
324
325 --extension--
326 extern unsigned long long int strtoull (const char *__restrict __nptr,
327     char **__restrict __endptr, int __base)
328     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
329     (1)));
330
331
332
333 --extension--
334 extern long long int strtouq (const char *__restrict __nptr,
335     char **__restrict __endptr, int __base)
336     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
337     (1)));
338
339 --extension--
340 extern unsigned long long int a64l (const char *__s)
341     __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
342     __attribute__((__nonnull__(1)));
343
344 # 305 "/usr/include/libc.h" 3 4
345 extern char *l64a (long int __n) __attribute__((__nothrow__, __leaf__))
346 ;
347
348
349
350
351
352
353 # 1 "/usr/include/x86_64-linux-gnu/sys/types.h" 1 3 4
354 # 27 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
355
356
357
358
359

```

```
360
361 typedef __u_char u_char;
362 typedef __u_short u_short;
363 typedef __u_int u_int;
364 typedef __u_long u_long;
365 typedef __quad_t quad_t;
366 typedef __u_quad_t u_quad_t;
367 typedef __fsid_t fsid_t;
368
369
370
371
372 typedef __loff_t loff_t;
373
374
375
376 typedef __ino_t ino_t;
377 # 60 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
378 typedef __dev_t dev_t;
379
380
381
382
383 typedef __gid_t gid_t;
384
385
386
387
388 typedef __mode_t mode_t;
389
390
391
392
393 typedef __nlink_t nlink_t;
394
395
396
397
398 typedef __uid_t uid_t;
399
400
401
402
403
404 typedef __off_t off_t;
405 # 98 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
406 typedef __pid_t pid_t;
407
408
409
410
411
412 typedef __id_t id_t;
413
```

```

414
415
416
417 typedef __ssize_t ssize_t;
418
419
420
421
422
423 typedef __daddr_t daddr_t;
424 typedef __caddr_t caddr_t;
425
426
427
428
429
430 typedef __key_t key_t;
431 # 132 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
432 # 1 "/usr/include/time.h" 1 3 4
433 # 57 "/usr/include/time.h" 3 4
434
435
436 typedef __clock_t clock_t;
437
438
439
440 # 73 "/usr/include/time.h" 3 4
441
442
443 typedef __time_t time_t;
444
445
446
447 # 91 "/usr/include/time.h" 3 4
448 typedef __clockid_t clockid_t;
449 # 103 "/usr/include/time.h" 3 4
450 typedef __timer_t timer_t;
451 # 133 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
452 # 146 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
453 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
454 # 147 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
455
456
457
458 typedef unsigned long int ulong;
459 typedef unsigned short int ushort;
460 typedef unsigned int uint;
461 # 194 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
462 typedef int int8_t __attribute__((__mode__(__QI__)));
463 typedef int int16_t __attribute__((__mode__(__HI__)));
464 typedef int int32_t __attribute__((__mode__(__SI__)));
465 typedef int int64_t __attribute__((__mode__(__DI__)));
466
467

```

```

468 typedef unsigned int u_int8_t __attribute__((__mode__(__QI__)));
469 typedef unsigned int u_int16_t __attribute__((__mode__(__HI__)));
470 typedef unsigned int u_int32_t __attribute__((__mode__(__SI__)));
471 typedef unsigned int u_int64_t __attribute__((__mode__(__DI__)));
472
473 typedef int register_t __attribute__((__mode__(__word__)));
474 # 219 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
475 # 1 "/usr/include/x86_64-linux-gnu/sys/select.h" 1 3 4
476 # 30 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
477 # 1 "/usr/include/x86_64-linux-gnu/bits/select.h" 1 3 4
478 # 22 "/usr/include/x86_64-linux-gnu/bits/select.h" 3 4
479 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
480 # 23 "/usr/include/x86_64-linux-gnu/bits/select.h" 2 3 4
481 # 31 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
482
483
484 # 1 "/usr/include/x86_64-linux-gnu/bits/sigset.h" 1 3 4
485 # 22 "/usr/include/x86_64-linux-gnu/bits/sigset.h" 3 4
486 typedef int __sig_atomic_t;
487
488
489
490
491 typedef struct
492 {
493     unsigned long int __val[(1024 / (8 * sizeof (unsigned long int))]];
494 } __sigset_t;
495 # 34 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
496
497
498
499 typedef __sigset_t sigset_t;
500
501
502
503
504
505 # 1 "/usr/include/time.h" 1 3 4
506 # 120 "/usr/include/time.h" 3 4
507 struct timespec
508 {
509     __time_t tv_sec;
510     __syscall_slong_t tv_nsec;
511 };
512 # 44 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
513
514 # 1 "/usr/include/x86_64-linux-gnu/bits/time.h" 1 3 4
515 # 30 "/usr/include/x86_64-linux-gnu/bits/time.h" 3 4
516 struct timeval
517 {
518     __time_t tv_sec;
519     __suseconds_t tv_usec;
520 };
521 # 46 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4

```

```

522
523
524 typedef __suseconds_t suseconds_t;
525
526
527
528
529
530 typedef long int __fd_mask;
531 # 64 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
532 typedef struct
533 {
534
535
536
537
538
539
540     __fd_mask __fds_bits[1024 / (8 * (int) sizeof (__fd_mask))];
541
542
543 } fd_set;
544
545
546
547
548
549
550 typedef __fd_mask fd_mask;
551 # 96 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
552
553 # 106 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
554 extern int select (int __nfds, fd_set *__restrict __readfds,
555                 fd_set *__restrict __writefds,
556                 fd_set *__restrict __exceptfds,
557                 struct timeval *__restrict __timeout);
558 # 118 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
559 extern int pselect (int __nfds, fd_set *__restrict __readfds,
560                     fd_set *__restrict __writefds,
561                     fd_set *__restrict __exceptfds,
562                     const struct timespec *__restrict __timeout,
563                     const __sigset_t *__restrict __sigmask);
564 # 131 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
565
566 # 220 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
567
568
569 # 1 "/usr/include/x86_64-linux-gnu/sys/sysmacros.h" 1 3 4
570 # 24 "/usr/include/x86_64-linux-gnu/sys/sysmacros.h" 3 4
571
572
573 --extension--
574 extern unsigned int gnu_dev_major (unsigned long long int __dev)
575     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__const__));

```

```

576 --extension--
577 extern unsigned int gnu_dev_minor (unsigned long long int __dev)
578     --attribute__ ((--nothrow__ , --leaf__) --attribute__ ((--const__));
579 --extension--
580 extern unsigned long long int gnu_dev_makedev (unsigned int __major ,
581         unsigned int __minor)
582     --attribute__ ((--nothrow__ , --leaf__) --attribute__ ((--const__));
583 # 58 "/usr/include/x86_64-linux-gnu/sys/sysmacros.h" 3 4
584
585 # 223 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
586
587
588
589
590
591 typedef __blksize_t blksize_t;
592
593
594
595
596
597
598 typedef __blkcnt_t blkcnt_t;
599
600
601
602 typedef __fsblkcnt_t fsblkcnt_t;
603
604
605
606 typedef __fsfilcnt_t fsfilcnt_t;
607 # 270 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
608 # 1 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 1 3 4
609 # 21 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
610 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
611 # 22 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 2 3 4
612 # 60 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
613 typedef unsigned long int pthread_t;
614
615
616 union pthread_attr_t
617 {
618     char __size[56];
619     long int __align;
620 };
621
622 typedef union pthread_attr_t pthread_attr_t;
623
624
625
626
627
628 typedef struct __pthread_internal_list
629 {

```

```

630     struct __pthread_internal_list *__prev;
631     struct __pthread_internal_list *__next;
632 } __pthread_list_t;
633 # 90 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
634 typedef union
635 {
636     struct __pthread_mutex_s
637     {
638         int __lock;
639         unsigned int __count;
640         int __owner;
641
642         unsigned int __nusers;
643
644
645         int __kind;
646
647         short __spins;
648         short __elision;
649         __pthread_list_t __list;
650 # 124 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
651     } __data;
652     char __size[40];
653     long int __align;
654 } pthread_mutex_t;
655
656
657 typedef union
658 {
659     char __size[4];
660     int __align;
661 } pthread_mutexattr_t;
662
663
664
665
666 typedef union
667 {
668     struct
669     {
670         int __lock;
671         unsigned int __futex;
672         __extension__ unsigned long long int __total_seq;
673         __extension__ unsigned long long int __wakeup_seq;
674         __extension__ unsigned long long int __woken_seq;
675         void *__mutex;
676         unsigned int __nwaiters;
677         unsigned int __broadcast_seq;
678     } __data;
679     char __size[48];
680     __extension__ long long int __align;
681 } pthread_cond_t;
682
683 typedef union

```

```

684 {
685     char __size[4];
686     int __align;
687 } pthread_condattr_t;
688
689
690
691 typedef unsigned int pthread_key_t;
692
693
694
695 typedef int pthread_once_t;
696
697
698
699
700
701 typedef union
702 {
703
704     struct
705     {
706         int __lock;
707         unsigned int __nr_readers;
708         unsigned int __readers_wakeup;
709         unsigned int __writer_wakeup;
710         unsigned int __nr_readers_queued;
711         unsigned int __nr_writers_queued;
712         int __writer;
713         int __shared;
714         unsigned long int __pad1;
715         unsigned long int __pad2;
716
717
718         unsigned int __flags;
719
720     } __data;
721 # 211 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
722     char __size[56];
723     long int __align;
724 } pthread_rwlock_t;
725
726
727
728
729
730 } pthread_rwlockattr_t;
731
732
733
734
735
736 typedef volatile int pthread_spinlock_t;
737

```

```

738
739
740
741 typedef union
742 {
743     char __size[32];
744     long int __align;
745 } pthread_barrier_t;
746
747 typedef union
748 {
749     char __size[4];
750     int __align;
751 } pthread_barrierattr_t;
752 # 271 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
753
754
755
756 # 315 "/usr/include/stdlib.h" 2 3 4
757
758
759
760
761
762
763 extern long int random (void) __attribute__((__nothrow__, __leaf__));
764
765
766 extern void srandom (unsigned int __seed) __attribute__((__nothrow__, __leaf__));
767
768
769
770
771
772 extern char *initstate (unsigned int __seed, char *__statebuf,
773 __size_t __statelen) __attribute__((__nothrow__, __leaf__))
774 __attribute__((__nonnull__(2)));
775
776
777 extern char *setstate (char *__statebuf) __attribute__((__nothrow__, __leaf__))
778 __attribute__((__nonnull__(1)));
779
780
781
782
783
784
785 struct random_data
786 {
787     int32_t *fptr;
788     int32_t *rptr;

```

```

789     int32_t *state;
790     int rand_type;
791     int rand_deg;
792     int rand_sep;
793     int32_t *end_ptr;
794 };
795
796 extern int random_r (struct random_data *__restrict __buf,
797     int32_t *__restrict __result) __attribute__((__nothrow__, __leaf__))
798     __attribute__((__nonnull__(1, 2)));
799
800 extern int srand_r (unsigned int __seed, struct random_data *__buf)
801     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(2)));
802
803 extern int initstate_r (unsigned int __seed, char *__restrict __statebuf,
804     size_t __statelen,
805     struct random_data *__restrict __buf)
806     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(2, 4)));
807
808 extern int setstate_r (char *__restrict __statebuf,
809     struct random_data *__restrict __buf)
810     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1, 2)));
811
812
813
814
815
816 extern int rand (void) __attribute__((__nothrow__, __leaf__));
817
818 extern void srand (unsigned int __seed) __attribute__((__nothrow__, __leaf__));
819
820
821
822
823 extern int rand_r (unsigned int *__seed) __attribute__((__nothrow__, __leaf__));
824
825
826
827
828
829
830
831 extern double drand48 (void) __attribute__((__nothrow__, __leaf__));
832 extern double erand48 (unsigned short int __xsubi[3]) __attribute__((__nothrow__, __leaf__))
833     __attribute__((__nonnull__(1)));
834
835 extern long int lrand48 (void) __attribute__((__nothrow__, __leaf__));

```

```

836 extern long int nrand48 (unsigned short int __xsubi[3])
837     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
838     (1)));
839
840 extern long int mrand48 (void) __attribute__ ((__nothrow__ , __leaf__));
841 extern long int jrand48 (unsigned short int __xsubi[3])
842     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
843     (1)));
844
845 extern void srand48 (long int __seedval) __attribute__ ((__nothrow__ ,
846     __leaf__));
847 extern unsigned short int *seed48 (unsigned short int __seed16v[3])
848     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
849     (1)));
850 extern void lcong48 (unsigned short int __param[7]) __attribute__ (((
851     __nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1)));
852
853
854 struct drand48_data
855 {
856     unsigned short int __x[3];
857     unsigned short int __old_x[3];
858     unsigned short int __c;
859     unsigned short int __init;
860     __extension__ unsigned long long int __a;
861
862 };
863
864
865 extern int drand48_r (struct drand48_data *__restrict __buffer,
866     double *__restrict __result) __attribute__ ((__nothrow__ ,
867     __leaf__)) __attribute__ ((__nonnull__ (1, 2)));
868 extern int erand48_r (unsigned short int __xsubi[3],
869     struct drand48_data *__restrict __buffer,
870     double *__restrict __result) __attribute__ ((__nothrow__ ,
871     __leaf__)) __attribute__ ((__nonnull__ (1, 2)));
872
873 extern int lrand48_r (struct drand48_data *__restrict __buffer,
874     long int *__restrict __result)
875     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
876     (1, 2)));
877 extern int nrnd48_r (unsigned short int __xsubi[3],
878     struct drand48_data *__restrict __buffer,
879     long int *__restrict __result)
880     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
881     (1, 2)));

```

```

881 extern int mrand48_r (struct drand48_data *__restrict __buffer,
882     long int *__restrict __result)
883     __attribute__ ((__nothrow__ , __leaf__))
884     __attribute__ ((__nonnull__(1, 2)));
885 extern int jrand48_r (unsigned short int __xsubi[3],
886     struct drand48_data *__restrict __buffer,
887     long int *__restrict __result)
888     __attribute__ ((__nothrow__ , __leaf__))
889     __attribute__ ((__nonnull__(1, 2)));
890
891 extern int srand48_r (long int __seedval, struct drand48_data *__buffer)
892     __attribute__ ((__nothrow__ , __leaf__))
893     __attribute__ ((__nonnull__(2)));
894
895 extern int seed48_r (unsigned short int __seed16v[3],
896     struct drand48_data *__buffer) __attribute__ ((__nothrow__ ,
897     __leaf__))
898     __attribute__ ((__nonnull__(1, 2)));
899
900
901
902
903
904
905
906
907
908 extern void *malloc (size_t __size) __attribute__ ((__nothrow__ , __leaf__))
909     __attribute__ ((__malloc__));
910 extern void *calloc (size_t __nmemb, size_t __size)
911     __attribute__ ((__nothrow__ , __leaf__))
912     __attribute__ ((__malloc__));
913
914
915
916
917
918
919
920
921
922 extern void *realloc (void *__ptr, size_t __size)
923     __attribute__ ((__nothrow__ , __leaf__))
924     __attribute__ ((__warn_unused_result__));
925 extern void free (void *__ptr) __attribute__ ((__nothrow__ , __leaf__));
926

```

```

927
928
929
930 extern void cfree (void * __ptr) __attribute__ ((__nothrow__, __leaf__));
931
932
933
934 # 1 "/usr/include/alloca.h" 1 3 4
935 # 24 "/usr/include/alloca.h" 3 4
936 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
937 # 25 "/usr/include/alloca.h" 2 3 4
938
939
940
941
942
943
944
945 extern void * alloca (size_t __size) __attribute__ ((__nothrow__, __leaf__));
946
947
948
949
950
951
952 # 493 "/usr/include/stdlib.h" 2 3 4
953
954
955
956
957
958 extern void * valloc (size_t __size) __attribute__ ((__nothrow__, __leaf__))
959     __attribute__ ((__malloc__));
960
961
962
963 extern int posix_memalign (void ** __memptr, size_t __alignment, size_t
964     __size)
965     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__(1)));
966
967
968
969 extern void * aligned_alloc (size_t __alignment, size_t __size)
970     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__malloc__))
971     __attribute__ ((__alloc_size__(2)));
972
973
974

```

```

975 extern void abort (void) __attribute__((__nothrow__, __leaf__))
976     __attribute__((__noreturn__));
977
978
979 extern int atexit (void (*__func) (void)) __attribute__((__nothrow__, __leaf__))
980     __attribute__((__nonnull__(1)));
981
982
983
984
985
986
987 extern int at_quick_exit (void (*__func) (void)) __attribute__((
988     __nothrow__, __leaf__)) __attribute__((__nonnull__(1)));
989
990
991
992
993
994
995 extern int on_exit (void (*__func) (int __status, void *__arg), void *
996     __arg)
997     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1)));
998
999
1000
1001
1002
1003 extern void exit (int __status) __attribute__((__nothrow__, __leaf__))
1004     __attribute__((__noreturn__));
1005
1006
1007
1008
1009 extern void quick_exit (int __status) __attribute__((__nothrow__, __leaf__))
1010     __attribute__((__noreturn__));
1011
1012
1013
1014
1015
1016
1017 extern void _Exit (int __status) __attribute__((__nothrow__, __leaf__))
1018     __attribute__((__noreturn__));
1019
1020

```

```

1021
1022
1023
1024 extern char *getenv (const char *__name) __attribute__ ((__nothrow__ ,
1025   __leaf__)) __attribute__ ((__nonnull__ (1))) ;
1026 # 578 "/usr/include/stdlib.h" 3 4
1027 extern int putenv (char *__string) __attribute__ ((__nothrow__ , __leaf__))
1028   __attribute__ ((__nonnull__ (1))) ;
1029
1030
1031
1032
1033 extern int setenv (const char *__name, const char *__value, int __replace)
1034   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (2)));
1035
1036
1037 extern int unsetenv (const char *__name) __attribute__ ((__nothrow__ ,
1038   __leaf__)) __attribute__ ((__nonnull__ (1)));
1039
1040
1041
1042
1043
1044 extern int clearenv (void) __attribute__ ((__nothrow__ , __leaf__));
1045 # 606 "/usr/include/stdlib.h" 3 4
1046 extern char *mktemp (char *__template) __attribute__ ((__nothrow__ ,
1047   __leaf__)) __attribute__ ((__nonnull__ (1)));
1048 # 620 "/usr/include/stdlib.h" 3 4
1049 extern int mkstemp (char *__template) __attribute__ ((__nonnull__ (1))) ;
1050 # 642 "/usr/include/stdlib.h" 3 4
1051 extern int mkstems (char *__template, int __suffixlen) __attribute__ ((
1052   __nonnull__ (1))) ;
1053 # 663 "/usr/include/stdlib.h" 3 4
1054 extern char *mkdtemp (char *__template) __attribute__ ((__nothrow__ ,
1055   __leaf__)) __attribute__ ((__nonnull__ (1))) ;
1056 # 712 "/usr/include/stdlib.h" 3 4
1057
1058
1059 extern int system (const char *__command) ;
1060
1061 # 734 "/usr/include/stdlib.h" 3 4
1062 extern char *realpath (const char *__restrict __name,
1063   char *__restrict __resolved) __attribute__ ((__nothrow__ ,
1064   __leaf__));
1065
1066

```

```

1067
1068
1069
1070 typedef int (*__compar_fn_t) (const void *, const void *);
1071 # 752 "/usr/include/stdlib.h" 3 4
1072
1073
1074
1075 extern void *bsearch (const void *_key, const void *_base,
1076     size_t _nmemb, size_t _size, __compar_fn_t _compar)
1077     --attribute__ ((_nonnull_ (1, 2, 5))) ;
1078
1079
1080
1081
1082
1083
1084
1085 extern void qsort (void *_base, size_t _nmemb, size_t _size,
1086     __compar_fn_t _compar) --attribute__ ((_nonnull_ (1, 4)));
1087 # 775 "/usr/include/stdlib.h" 3 4
1088 extern int abs (int _x) --attribute__ ((_nothrow_ , _leaf_)
1089     --attribute__ ((_const_)) ;
1090 extern long int labs (long int _x) --attribute__ ((_nothrow_ , _leaf_)
1091     --attribute__ ((_const_)) ;
1092
1093 --extension__ extern long long int llabs (long long int _x)
1094     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_const_))
1095     ;
1096
1097
1098
1099
1100
1101
1102 extern div_t div (int _numer, int _denom)
1103     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_const_))
1104     ;
1105 extern ldiv_t ldiv (long int _numer, long int _denom)
1106     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_const_))
1107     ;
1108
1109
1110 --extension__ extern lldiv_t lldiv (long long int _numer,
1111     long long int _denom)
1112     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_const_))
1113     ;
1114 # 812 "/usr/include/stdlib.h" 3 4

```

```

1115 extern char *ecvt (double __value, int __ndigit, int *__restrict __decpt,
1116     int *__restrict __sign) __attribute__ ((__nothrow__, __leaf__))
1117     __attribute__ ((__nonnull__ (3, 4))) ;
1118
1119
1120
1121 extern char *fcvt (double __value, int __ndigit, int *__restrict __decpt,
1122     int *__restrict __sign) __attribute__ ((__nothrow__, __leaf__))
1123     __attribute__ ((__nonnull__ (3, 4))) ;
1124
1125
1126
1127 extern char *gcvt (double __value, int __ndigit, char *__buf)
1128     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
1129     (3))) ;
1130
1131
1132
1133 extern char *qecvt (long double __value, int __ndigit,
1134     int *__restrict __decpt, int *__restrict __sign)
1135     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
1136     (3, 4))) ;
1137 extern char *qfcvt (long double __value, int __ndigit,
1138     int *__restrict __decpt, int *__restrict __sign)
1139     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
1140     (3, 4))) ;
1141 extern char *qgcvt (long double __value, int __ndigit, char *__buf)
1142     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
1143     (3))) ;
1144
1145 extern int ecvt_r (double __value, int __ndigit, int *__restrict __decpt,
1146     int *__restrict __sign, char *__restrict __buf,
1147     size_t __len) __attribute__ ((__nothrow__, __leaf__)) __attribute__ 
1148     ((__nonnull__ (3, 4, 5)));
1149 extern int fcvt_r (double __value, int __ndigit, int *__restrict __decpt,
1150     int *__restrict __sign, char *__restrict __buf,
1151     size_t __len) __attribute__ ((__nothrow__, __leaf__)) __attribute__ 
1152     ((__nonnull__ (3, 4, 5)));
1153 extern int qecvt_r (long double __value, int __ndigit,
1154     int *__restrict __decpt, int *__restrict __sign,
1155     char *__restrict __buf, size_t __len)
1156     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
1157     (3, 4, 5)));
1158 extern int qfcvt_r (long double __value, int __ndigit,
1159     int *__restrict __decpt, int *__restrict __sign,
1160     char *__restrict __buf, size_t __len)

```

```

1159     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__))
1160     (3, 4, 5));
1161
1162
1163
1164
1165
1166 extern int mblen (const char *_s, size_t _n) __attribute__((__nothrow__,
1167     __leaf__));
1168
1169 extern int mbtowc (wchar_t *__restrict __pwc,
1170     const char *__restrict __s, size_t _n) __attribute__((__nothrow__,
1171     __leaf__));
1172
1173 extern int wctomb (char *_s, wchar_t __wchar) __attribute__((__nothrow__,
1174     __leaf__));
1175
1176
1177 extern size_t mbstowcs (wchar_t *__restrict __pwcs,
1178     const char *__restrict __s, size_t _n) __attribute__((__nothrow__,
1179     __leaf__));
1180
1181 extern size_t wcstombs (char *__restrict __s,
1182     const wchar_t *__restrict __pwcs, size_t _n)
1183     __attribute__((__nothrow__, __leaf__));
1184
1185
1186
1187
1188
1189
1190
1191 extern int rpmatch (const char *_response) __attribute__((__nothrow__,
1192     __leaf__)) __attribute__((__nonnull__(1))) ;
1193 # 899 "/usr/include/stdlib.h" 3 4
1194 extern int getsubopt (char **__restrict __optionp,
1195     char *const __restrict __tokens,
1196     char **__restrict __valuep)
1197     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
1198     (1, 2, 3))) ;
1199 # 951 "/usr/include/stdlib.h" 3 4
1200 extern int getloadavg (double __loadavg[], int __nelem)
1201     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
1202     (1)));
1203 # 1 "/usr/include/x86_64-linux-gnu/bits/stdc-float.h" 1 3 4
1204 # 956 "/usr/include/stdlib.h" 2 3 4
1205 # 968 "/usr/include/stdlib.h" 3 4

```

```

1205
1206 # 4 "headers/all_headers.h" 2
1207 # 1 "/usr/include/stdio.h" 1 3 4
1208 # 29 "/usr/include/stdio.h" 3 4
1209
1210
1211
1212
1213 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
1214 # 34 "/usr/include/stdio.h" 2 3 4
1215 # 44 "/usr/include/stdio.h" 3 4
1216 struct _IO_FILE;
1217
1218
1219
1220 typedef struct _IO_FILE FILE;
1221
1222
1223
1224
1225
1226 # 64 "/usr/include/stdio.h" 3 4
1227 typedef struct _IO_FILE __FILE;
1228 # 74 "/usr/include/stdio.h" 3 4
1229 # 1 "/usr/include/libio.h" 1 3 4
1230 # 31 "/usr/include/libio.h" 3 4
1231 # 1 "/usr/include/_G_config.h" 1 3 4
1232 # 15 "/usr/include/_G_config.h" 3 4
1233 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
1234 # 16 "/usr/include/_G_config.h" 2 3 4
1235
1236
1237
1238
1239 # 1 "/usr/include/wchar.h" 1 3 4
1240 # 82 "/usr/include/wchar.h" 3 4
1241 typedef struct
1242 {
1243     int __count;
1244     union
1245     {
1246         unsigned int __wch;
1247
1248
1249
1250
1251         char __wchb[4];
1252     } __value;
1253 } __mbstate_t;
1254 # 21 "/usr/include/_G_config.h" 2 3 4
1255 typedef struct
1256 {
1257     __off_t __pos;
1258     __mbstate_t __state;

```

```

1259 } _G_fpos_t;
1260 typedef struct
1261 {
1262     __off64_t __pos;
1263     __mbstate_t __state;
1264 } _G_fpos64_t;
1265 # 32 "/usr/include/libio.h" 2 3 4
1266 # 49 "/usr/include/libio.h" 3 4
1267 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stdarg.h" 1 3 4
1268 # 40 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stdarg.h" 3 4
1269 typedef __builtin_va_list __gnuc_va_list;
1270 # 50 "/usr/include/libio.h" 2 3 4
1271 # 144 "/usr/include/libio.h" 3 4
1272 struct _IO_jump_t; struct _IO_FILE;
1273 # 154 "/usr/include/libio.h" 3 4
1274 typedef void _IO_lock_t;
1275
1276
1277
1278
1279
1280 struct _IO_marker {
1281     struct _IO_marker *_next;
1282     struct _IO_FILE *_sbuf;
1283
1284
1285     int _pos;
1286 # 177 "/usr/include/libio.h" 3 4
1287 };
1288
1289
1290
1291 enum __codecvt_result
1292 {
1293     __codecvt_ok,
1294     __codecvt_partial,
1295     __codecvt_error,
1296     __codecvt_noconv
1297 };
1298 # 245 "/usr/include/libio.h" 3 4
1299 struct _IO_FILE {
1300     int _flags;
1301
1302
1303
1304
1305     char* _IO_read_ptr;
1306     char* _IO_read_end;
1307     char* _IO_read_base;
1308     char* _IO_write_base;
1309     char* _IO_write_ptr;
1310     char* _IO_write_end;
1311     char* _IO_buf_base;
1312     char* _IO_buf_end;

```

```

1313
1314     char *_IO_save_base;
1315     char *_IO_backup_base;
1316     char *_IO_save_end;
1317
1318     struct _IO_marker *_markers;
1319
1320     struct _IO_FILE *_chain;
1321
1322     int _fileno;
1323
1324
1325
1326     int _flags2;
1327
1328     __off_t _old_offset;
1329
1330
1331
1332     unsigned short _cur_column;
1333     signed char _vtable_offset;
1334     char _shortbuf[1];
1335
1336
1337
1338     _IO_lock_t *_lock;
1339 # 293 "/usr/include/libio.h" 3 4
1340     __off64_t _offset;
1341 # 302 "/usr/include/libio.h" 3 4
1342     void *_pad1;
1343     void *_pad2;
1344     void *_pad3;
1345     void *_pad4;
1346     size_t _pad5;
1347
1348     int _mode;
1349
1350     char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)
1351     ];
1352 };
1353
1354
1355 typedef struct _IO_FILE _IO_FILE;
1356
1357
1358 struct _IO_FILE_plus;
1359
1360 extern struct _IO_FILE_plus _IO_2_1_stdin_;
1361 extern struct _IO_FILE_plus _IO_2_1_stdout_;
1362 extern struct _IO_FILE_plus _IO_2_1_stderr_;
1363 # 338 "/usr/include/libio.h" 3 4
1364 typedef __ssize_t __io_read_fn (void *__cookie, char *__buf, size_t
1365     __nbytes);

```

```

1365
1366
1367
1368
1369
1370
1371
1372 typedef _ssize_t __io_write_fn (void *__cookie, const char *__buf,
1373     size_t __n);
1374
1375
1376
1377
1378
1379
1380
1381 typedef int __io_seek_fn (void *__cookie, __off64_t *__pos, int __w);
1382
1383
1384 typedef int __io_close_fn (void *__cookie);
1385 # 390 "/usr/include/libio.h" 3 4
1386 extern int __underflow (_IO_FILE *);
1387 extern int __uflow (_IO_FILE *);
1388 extern int __overflow (_IO_FILE *, int);
1389 # 434 "/usr/include/libio.h" 3 4
1390 extern int _IO_getc (_IO_FILE *__fp);
1391 extern int _IO_putc (int __c, _IO_FILE *__fp);
1392 extern int _IO_feof (_IO_FILE *__fp) __attribute__ ((__nothrow__ ,
1393     __leaf__));
1394 extern int _IO_ferror (_IO_FILE *__fp) __attribute__ ((__nothrow__ ,
1395     __leaf__));
1396
1397
1398
1399
1400
1401 extern void _IO_flockfile (_IO_FILE *) __attribute__ ((__nothrow__ ,
1402     __leaf__));
1403 extern void _IO_funlockfile (_IO_FILE *) __attribute__ ((__nothrow__ ,
1404     __leaf__));
1405 extern int _IO_ftrylockfile (_IO_FILE *) __attribute__ ((__nothrow__ ,
1406     __leaf__));
1407 # 464 "/usr/include/libio.h" 3 4
1408 extern int _IO_vfscanf (_IO_FILE * __restrict, const char * __restrict,
1409     _gnuc_va_list, int * __restrict);
1410 extern int _IO_vfprintf (_IO_FILE * __restrict, const char * __restrict,
1411     _gnuc_va_list);
1412 extern _ssize_t _IO_padvn (_IO_FILE *, int, _ssize_t);
1413 extern size_t _IO_sgetn (_IO_FILE *, void *, size_t);
1414
1415 extern __off64_t _IO_seekoff (_IO_FILE *, __off64_t, int, int);
1416 extern __off64_t _IO_seekpos (_IO_FILE *, __off64_t, int);

```

```

1414
1415 extern void _IO_free_backup_area (_IO_FILE *) __attribute__ ((__nothrow__
1416   , __leaf__));
1417 # 75 "/usr/include/stdio.h" 2 3 4
1418
1419
1420
1421 typedef __gnuc_va_list va_list;
1422 # 108 "/usr/include/stdio.h" 3 4
1423
1424
1425 typedef _G_fpos_t fpos_t;
1426
1427
1428
1429
1430 # 164 "/usr/include/stdio.h" 3 4
1431 # 1 "/usr/include/x86_64-linux-gnu/bits/stdio_lim.h" 1 3 4
1432 # 165 "/usr/include/stdio.h" 2 3 4
1433
1434
1435
1436 extern struct _IO_FILE *stdin;
1437 extern struct _IO_FILE *stdout;
1438 extern struct _IO_FILE *stderr;
1439
1440
1441
1442
1443
1444
1445
1446 extern int remove (const char *__filename) __attribute__ ((__nothrow__,
1447   __leaf__));
1448 extern int rename (const char *__old, const char *__new) __attribute__ ((
1449   __nothrow__, __leaf__));
1450
1451
1452
1453 extern int renameat (int __oldfd, const char *__old, int __newfd,
1454   const char *__new) __attribute__ ((__nothrow__, __leaf__));
1455
1456
1457
1458
1459
1460
1461
1462
1463 extern FILE *tmpfile (void);
1464 # 209 "/usr/include/stdio.h" 3 4

```

```

1465 extern char *tmpnam (char *_s) __attribute__ ((__nothrow__, __leaf__)) ;
1466
1467
1468
1469
1470
1471 extern char *tmpnam_r (char *_s) __attribute__ ((__nothrow__, __leaf__))
1472 ;
1472 # 227 "/usr/include/stdio.h" 3 4
1473 extern char *tempnam (const char *_dir, const char *_pfx)
1474     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__malloc__))
1474 ;
1475
1476
1477
1478
1479
1480
1481
1482
1483 extern int fclose (FILE *_stream);
1484
1485
1486
1487
1488 extern int fflush (FILE *_stream);
1489
1490 # 252 "/usr/include/stdio.h" 3 4
1491 extern int fflush_unlocked (FILE *_stream);
1492 # 266 "/usr/include/stdio.h" 3 4
1493
1494
1495
1496
1497
1498
1499 extern FILE *fopen (const char *__restrict __filename,
1500     const char *__restrict __modes) ;
1501
1502
1503
1504
1505 extern FILE *freopen (const char *__restrict __filename,
1506     const char *__restrict __modes,
1507     FILE *__restrict __stream) ;
1508 # 295 "/usr/include/stdio.h" 3 4
1509
1510 # 306 "/usr/include/stdio.h" 3 4
1511 extern FILE *fdopen (int _fd, const char *_modes) __attribute__ (((
1512     __nothrow__, __leaf__)) ;
1512 # 319 "/usr/include/stdio.h" 3 4
1513 extern FILE *fmemopen (void *_s, size_t _len, const char *_modes)
1514     __attribute__ ((__nothrow__, __leaf__)) ;
1515

```

```

1516
1517
1518
1519 extern FILE *open_memstream (char **__bufloc, size_t *__sizeloc)
1520     __attribute__ ((__nothrow__, __leaf__));
1521
1522
1523
1524
1525
1526 extern void setbuf (FILE *__restrict __stream, char *__restrict __buf)
1527     __attribute__ ((__nothrow__, __leaf__));
1528
1529
1530 extern int setvbuf (FILE *__restrict __stream, char *__restrict __buf,
1531     int __modes, size_t __n) __attribute__ ((__nothrow__, __leaf__));
1532
1533
1534
1535
1536
1537 extern void setbuffer (FILE *__restrict __stream, char *__restrict __buf,
1538     size_t __size) __attribute__ ((__nothrow__, __leaf__));
1539
1540
1541 extern void setlinebuf (FILE *__stream) __attribute__ ((__nothrow__,
1542     __leaf__));
1543
1544
1545
1546
1547
1548
1549
1550 extern int fprintf (FILE *__restrict __stream,
1551     const char *__restrict __format, ...);
1552
1553
1554
1555
1556 extern int printf (const char *__restrict __format, ...);
1557
1558
1559 extern int sprintf (char *__restrict __s,
1560     const char *__restrict __format, ...) __attribute__ ((__nothrow__));
1561
1562
1563
1564
1565 extern int vfprintf (FILE *__restrict __s, const char *__restrict __format
1566     ,

```

```

1566     __gnuc_va_list __arg);
1567
1568
1569
1570
1571 extern int vprintf (const char *__restrict __format, __gnuc_va_list __arg)
1572 ;
1573
1574 extern int vsprintf (char *__restrict __s, const char *__restrict __format
1575   ,
1576   __gnuc_va_list __arg) __attribute__ ((__nothrow__));
1577
1578
1579
1580 extern int snprintf (char *__restrict __s, size_t __ maxlen,
1581   const char *__restrict __format, ...)
1582   __attribute__ ((__nothrow__)) __attribute__ ((__format__ (__printf__,
1583   3, 4)));
1584
1585 extern int vsnprintf (char *__restrict __s, size_t __ maxlen,
1586   const char *__restrict __format, __gnuc_va_list __arg)
1587   __attribute__ ((__nothrow__)) __attribute__ ((__format__ (__printf__,
1588   3, 0)));
1589
1590 # 412 "/usr/include/stdio.h" 3 4
1591 extern int vdprintf (int __fd, const char *__restrict __fmt,
1592   __gnuc_va_list __arg)
1593   __attribute__ ((__format__ (__printf__, 2, 0)));
1594 extern int dprintf (int __fd, const char *__restrict __fmt, ...)
1595   __attribute__ ((__format__ (__printf__, 2, 3)));
1596
1597
1598
1599
1600
1601
1602 extern int fscanf (FILE *__restrict __stream,
1603   const char *__restrict __format, ...) ;
1604
1605
1606
1607
1608 extern int scanf (const char *__restrict __format, ...) ;
1609
1610 extern int sscanf (const char *__restrict __s,
1611   const char *__restrict __format, ...) __attribute__ ((__nothrow__,
1612   __leaf__));
1613
1614 # 443 "/usr/include/stdio.h" 3 4
1615 extern int fscanf (FILE *__restrict __stream, const char *__restrict
1616   __format, ...) __asm__ ("\" __isoc99_fscanf")

```

```

1614
1615             ;
1616 extern int scanf (const char * __restrict __format, ...) __asm__ (""
1617             "__isoc99_scanf")
1618             ;
1619 extern int sscanf (const char * __restrict __s, const char * __restrict
1620             __format, ...) __asm__ ("" "__isoc99_sscanf") __attribute__ ((
1621             __nothrow__, __leaf__))
1622             ;
1623 # 463 "/usr/include/stdio.h" 3 4
1624
1625
1626
1627
1628
1629
1630 extern int vfscanf (FILE * __restrict __s, const char * __restrict __format,
1631             __gnuc_va_list __arg)
1632             __attribute__ ((__format__ (__scanf__, 2, 0))) ;
1633
1634
1635
1636
1637
1638 extern int vscanf (const char * __restrict __format, __gnuc_va_list __arg)
1639             __attribute__ ((__format__ (__scanf__, 1, 0))) ;
1640
1641
1642 extern int vscanf (const char * __restrict __s,
1643             const char * __restrict __format, __gnuc_va_list __arg)
1644             __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__format__ (
1645             __scanf__, 2, 0)));
1646 # 494 "/usr/include/stdio.h" 3 4
1647 extern int vfscanf (FILE * __restrict __s, const char * __restrict __format,
1648             __gnuc_va_list __arg) __asm__ ("" "__isoc99_vfscanf")
1649             __attribute__ ((__format__ (__scanf__, 2, 0))) ;
1650
1651 extern int vscanf (const char * __restrict __format, __gnuc_va_list __arg)
1652             __asm__ ("" "__isoc99_vscanf")
1653             __attribute__ ((__format__ (__scanf__, 1, 0))) ;
1654
1655 extern int vscanf (const char * __restrict __s, const char * __restrict
1656             __format, __gnuc_va_list __arg) __asm__ ("" "__isoc99_vscanf")
1657             __attribute__ ((__nothrow__, __leaf__))
1658             __attribute__ ((__format__ (__scanf__, 2, 0)));
1659 # 522 "/usr/include/stdio.h" 3 4

```

```
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669 extern int fgetc (FILE *_stream);
1670 extern int getc (FILE *_stream);
1671
1672
1673
1674
1675
1676 extern int getchar (void);
1677
1678 # 550 "/usr/include/stdio.h" 3 4
1679 extern int getc_unlocked (FILE *_stream);
1680 extern int getchar_unlocked (void);
1681 # 561 "/usr/include/stdio.h" 3 4
1682 extern int fgetc_unlocked (FILE *_stream);
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694 extern int fputc (int __c, FILE *_stream);
1695 extern int putc (int __c, FILE *_stream);
1696
1697
1698
1699
1700
1701 extern int putchar (int __c);
1702
1703 # 594 "/usr/include/stdio.h" 3 4
1704 extern int fputc_unlocked (int __c, FILE *_stream);
1705
1706
1707
1708
1709
1710
1711
1712 extern int putc_unlocked (int __c, FILE *_stream);
1713 extern int putchar_unlocked (int __c);
```

```

1714
1715
1716
1717
1718
1719
1720 extern int getw (FILE *_stream);
1721
1722
1723 extern int putw (int _w, FILE *_stream);
1724
1725
1726
1727
1728
1729
1730
1731
1732 extern char *fgets (char *_restrict _s, int _n, FILE *_restrict
1733     __stream)
1734 ;
1735 # 640 "/usr/include/stdio.h" 3 4
1736 # 665 "/usr/include/stdio.h" 3 4
1737 extern __ssize_t __getdelim (char **__restrict __lineptr,
1738     size_t *__restrict __n, int __delimiter,
1739     FILE *__restrict __stream) ;
1740 extern __ssize_t getdelim (char **__restrict __lineptr,
1741     size_t *__restrict __n, int __delimiter,
1742     FILE *__restrict __stream) ;
1743
1744
1745
1746
1747
1748
1749
1750 extern __ssize_t getline (char **__restrict __lineptr,
1751     size_t *__restrict __n,
1752     FILE *__restrict __stream) ;
1753
1754
1755
1756
1757
1758
1759
1760
1761 extern int fputs (const char *_restrict _s, FILE *_restrict __stream);
1762
1763
1764
1765
1766
```

```

1767 extern int puts (const char *__s);
1768
1769
1770
1771
1772
1773
1774 extern int ungetc (int __c, FILE *__stream);
1775
1776
1777
1778
1779
1780
1781 extern size_t fread (void *__restrict __ptr, size_t __size,
1782                      size_t __n, FILE *__restrict __stream) ;
1783
1784
1785
1786
1787 extern size_t fwrite (const void *__restrict __ptr, size_t __size,
1788                      size_t __n, FILE *__restrict __s);
1789
1790 # 737 "/usr/include/stdio.h" 3 4
1791 extern size_t fread_unlocked (void *__restrict __ptr, size_t __size,
1792                               size_t __n, FILE *__restrict __stream) ;
1793 extern size_t fwrite_unlocked (const void *__restrict __ptr, size_t __size
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803 extern int fseek (FILE *__stream, long int __off, int __whence);
1804
1805
1806
1807
1808 extern long int ftell (FILE *__stream) ;
1809
1810
1811
1812
1813 extern void rewind (FILE *__stream);
1814
1815 # 773 "/usr/include/stdio.h" 3 4
1816 extern int fseeko (FILE *__stream, __off_t __off, int __whence);
1817
1818
1819
```

```

1820
1821 extern __off_t ftello (FILE *__stream) ;
1822 # 792 "/usr/include/stdio.h" 3 4
1823
1824
1825
1826
1827
1828
1829 extern int fgetpos (FILE *__restrict __stream, fpos_t *__restrict __pos);
1830
1831
1832
1833
1834 extern int fsetpos (FILE *__stream, const fpos_t *__pos);
1835 # 815 "/usr/include/stdio.h" 3 4
1836
1837 # 824 "/usr/include/stdio.h" 3 4
1838
1839
1840 extern void clearerr (FILE *__stream) __attribute__((__nothrow__ ,
1841 __leaf__));
1842 extern int feof (FILE *__stream) __attribute__((__nothrow__ , __leaf__))
1843 ;
1844 extern int perror (FILE *__stream) __attribute__((__nothrow__ , __leaf__))
1845 ;
1846
1847
1848
1849 extern void clearerr_unlocked (FILE *__stream) __attribute__((__nothrow__ ,
1850 __leaf__));
1851 extern int feof_unlocked (FILE *__stream) __attribute__((__nothrow__ ,
1852 __leaf__));
1853 extern int perror_unlocked (FILE *__stream) __attribute__((__nothrow__ ,
1854 __leaf__));
1855
1856
1857
1858
1859
1860 extern void perror (const char *_s);
1861
1862
1863
1864
1865
1866
1867 # 1 "/usr/include/x86_64-linux-gnu/bits/sys_errlist.h" 1 3 4

```

```

1868 # 26 "/usr/include/x86_64-linux-gnu/bits/sys_errlist.h" 3 4
1869 extern int sys_nerr;
1870 extern const char *const sys_errlist[];
1871 # 854 "/usr/include/stdio.h" 2 3 4
1872
1873
1874
1875
1876 extern int fileno (FILE *_stream) __attribute__ ((__nothrow__, __leaf__))
1877   ;
1878
1879
1880
1881 extern int fileno_unlocked (FILE *_stream) __attribute__ ((__nothrow__, __leaf__));
1882 # 873 "/usr/include/stdio.h" 3 4
1883 extern FILE *popen (const char *_command, const char *_modes);
1884
1885
1886
1887
1888
1889 extern int pclose (FILE *_stream);
1890
1891
1892
1893
1894
1895 extern char *ctermid (char *_s) __attribute__ ((__nothrow__, __leaf__));
1896 # 913 "/usr/include/stdio.h" 3 4
1897 extern void flockfile (FILE *_stream) __attribute__ ((__nothrow__, __leaf__));
1898
1899
1900
1901 extern int ftrylockfile (FILE *_stream) __attribute__ ((__nothrow__, __leaf__));
1902
1903
1904 extern void funlockfile (FILE *_stream) __attribute__ ((__nothrow__, __leaf__));
1905 # 943 "/usr/include/stdio.h" 3 4
1906
1907 # 5 "headers/all_headers.h" 2
1908 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stdbool.h" 1 3 4
1909 # 6 "headers/all_headers.h" 2
1910 # 1 "/usr/include/math.h" 1 3 4
1911 # 28 "/usr/include/math.h" 3 4
1912
1913
1914
1915
1916 # 1 "/usr/include/x86_64-linux-gnu/bits/huge_val.h" 1 3 4

```

```

1917 # 33 "/usr/include/math.h" 2 3 4
1918
1919 # 1 "/usr/include/x86_64-linux-gnu/bits/huge_valf.h" 1 3 4
1920 # 35 "/usr/include/math.h" 2 3 4
1921 # 1 "/usr/include/x86_64-linux-gnu/bits/huge_vall.h" 1 3 4
1922 # 36 "/usr/include/math.h" 2 3 4
1923
1924
1925 # 1 "/usr/include/x86_64-linux-gnu/bits/inf.h" 1 3 4
1926 # 39 "/usr/include/math.h" 2 3 4
1927
1928
1929 # 1 "/usr/include/x86_64-linux-gnu/bits/nan.h" 1 3 4
1930 # 42 "/usr/include/math.h" 2 3 4
1931
1932
1933
1934 # 1 "/usr/include/x86_64-linux-gnu/bits/mathdef.h" 1 3 4
1935 # 28 "/usr/include/x86_64-linux-gnu/bits/mathdef.h" 3 4
1936 typedef float float_t;
1937 typedef double double_t;
1938 # 46 "/usr/include/math.h" 2 3 4
1939 # 69 "/usr/include/math.h" 3 4
1940 # 1 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 1 3 4
1941 # 52 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
1942
1943
1944 extern double acos (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __acos (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1945
1946 extern double asin (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __asin (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1947
1948 extern double atan (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __atan (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1949
1950 extern double atan2 (double __y, double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
      extern double __atan2 (double __y, double __x)
      __attribute__ ((__nothrow__ , __leaf__));
1951
1952
1953 extern double cos (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __cos (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1954
1955 extern double sin (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __sin (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1956
1957 extern double tan (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __tan (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));

```

```

));
1958
1959
1960
1961
1962 extern double cosh (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __cosh (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
1963
1964 extern double sinh (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __sinh (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
1965
1966 extern double tanh (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __tanh (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
1967
1968 # 86 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
1969
1970
1971 extern double acosh (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __acosh (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
1972
1973 extern double asinh (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __asinh (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
1974
1975 extern double atanh (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __atanh (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
1976
1977
1978
1979
1980
1981
1982
1983 extern double exp (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __exp (double __x) __attribute__ ((__nothrow__ , __leaf__));
1984
1985
1986 extern double frexp (double __x, int *__exponent) __attribute__ (((
__nothrow__ , __leaf__)); extern double __frexp (double __x, int *
__exponent) __attribute__ ((__nothrow__ , __leaf__));
1987
1988
1989 extern double ldexp (double __x, int __exponent) __attribute__ (((
__nothrow__ , __leaf__)); extern double __ldexp (double __x, int
__exponent) __attribute__ ((__nothrow__ , __leaf__));
1990
1991
```

```

1992 extern double log (double __x) __attribute__ ((__nothrow__ , __leaf__));
    extern double __log (double __x) __attribute__ ((__nothrow__ , __leaf__));
);
1993
1994
1995 extern double log10 (double __x) __attribute__ ((__nothrow__ , __leaf__));
    extern double __log10 (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
);
1996
1997
1998 extern double modf (double __x, double *__iptr) __attribute__ ((
__nothrow__ , __leaf__));
    extern double __modf (double __x, double *
__iptr) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((
__nonnull__ (2)));
1999
2000 # 126 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2001
2002
2003 extern double expm1 (double __x) __attribute__ ((__nothrow__ , __leaf__));
    extern double __expm1 (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
2004
2005
2006 extern double log1p (double __x) __attribute__ ((__nothrow__ , __leaf__));
    extern double __log1p (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
2007
2008
2009 extern double logb (double __x) __attribute__ ((__nothrow__ , __leaf__));
    extern double __logb (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
2010
2011
2012
2013
2014
2015
2016 extern double exp2 (double __x) __attribute__ ((__nothrow__ , __leaf__));
    extern double __exp2 (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
2017
2018
2019 extern double log2 (double __x) __attribute__ ((__nothrow__ , __leaf__));
    extern double __log2 (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
2020
2021
2022
2023
2024
2025
2026
2027

```

```

2028 extern double pow (double __x, double __y) __attribute__ ((__nothrow__ ,
2029   __leaf__));
2030
2031 extern double sqrt (double __x) __attribute__ ((__nothrow__ , __leaf__));
2032   extern double __sqrt (double __x) __attribute__ ((__nothrow__ ,
2033     __leaf__));
2034
2035
2036
2037 extern double hypot (double __x, double __y) __attribute__ ((__nothrow__ ,
2038   __leaf__));
2039   extern double __hypot (double __x, double __y)
2040     __attribute__ ((__nothrow__ , __leaf__));
2041
2042
2043
2044 extern double cbrt (double __x) __attribute__ ((__nothrow__ , __leaf__));
2045   extern double __cbrt (double __x) __attribute__ ((__nothrow__ ,
2046     __leaf__));
2047
2048
2049
2050
2051
2052
2053 extern double ceil (double __x) __attribute__ ((__nothrow__ , __leaf__))
2054   __attribute__ ((__const__));
2055   extern double __ceil (double __x)
2056     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2057
2058
2059
2060 extern double fabs (double __x) __attribute__ ((__nothrow__ , __leaf__))
2061   __attribute__ ((__const__));
2062   extern double __fabs (double __x)
2063     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2064
2065
2066
2067 extern double floor (double __x) __attribute__ ((__nothrow__ , __leaf__))
2068   __attribute__ ((__const__));
2069   extern double __floor (double __x)
2070     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2071
2072
2073
2074 extern double fmod (double __x, double __y) __attribute__ ((__nothrow__ ,
2075   __leaf__));
2076   extern double __fmod (double __x, double __y) __attribute__ (
2077     __nothrow__ , __leaf__));
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165

```

```

2066
2067 extern int __isinf (double __value) __attribute__ ((__nothrow__ , __leaf__))
2068   __attribute__ ((__const__));
2069
2070 extern int __finite (double __value) __attribute__ ((__nothrow__ ,
2071   __leaf__)) __attribute__ ((__const__));
2072
2073
2074
2075
2076 extern int isinf (double __value) __attribute__ ((__nothrow__ , __leaf__))
2077   __attribute__ ((__const__));
2078
2079 extern int finite (double __value) __attribute__ ((__nothrow__ , __leaf__))
2080   __attribute__ ((__const__));
2081
2082 extern double drem (double __x, double __y) __attribute__ ((__nothrow__ ,
2083   __leaf__)); extern double __drem (double __x, double __y) __attribute__ (
2084   __nothrow__ , __leaf__);
2085
2086 extern double significand (double __x) __attribute__ ((__nothrow__ ,
2087   __leaf__)); extern double __significand (double __x) __attribute__ (
2088   __nothrow__ , __leaf__);
2089
2090
2091
2092 extern double copysign (double __x, double __y) __attribute__ (((
2093   __nothrow__ , __leaf__)) __attribute__ ((__const__)); extern double
2094   __copysign (double __x, double __y) __attribute__ ((__nothrow__ ,
2095   __leaf__)) __attribute__ ((__const__));
2096
2097
2098
2099 extern double nan (const char *__tagb) __attribute__ ((__nothrow__ ,
2100   __leaf__)) __attribute__ ((__const__)); extern double __nan (const char
2101   *__tagb) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ (((
2102   __const__));
2103
2104

```

```

2105 extern int __isnan (double __value) __attribute__ ((__nothrow__ , __leaf__))
2106   __attribute__ ((__const__));
2107
2108
2109 extern int isnan (double __value) __attribute__ ((__nothrow__ , __leaf__))
2110   __attribute__ ((__const__));
2111
2112 extern double j0 (double) __attribute__ ((__nothrow__ , __leaf__)); extern
2113   double __j0 (double) __attribute__ ((__nothrow__ , __leaf__));
2114 extern double j1 (double) __attribute__ ((__nothrow__ , __leaf__)); extern
2115   double __j1 (double) __attribute__ ((__nothrow__ , __leaf__));
2116 extern double jn (int, double) __attribute__ ((__nothrow__ , __leaf__));
2117   extern double __jn (int, double) __attribute__ ((__nothrow__ , __leaf__));
2118
2119
2120
2121
2122
2123
2124 extern double erf (double) __attribute__ ((__nothrow__ , __leaf__));
2125   extern double __erf (double) __attribute__ ((__nothrow__ , __leaf__));
2126 extern double erfc (double) __attribute__ ((__nothrow__ , __leaf__));
2127   extern double __erfc (double) __attribute__ ((__nothrow__ , __leaf__));
2128 extern double lgamma (double) __attribute__ ((__nothrow__ , __leaf__));
2129   extern double __lgamma (double) __attribute__ ((__nothrow__ , __leaf__));
2130
2131
2132
2133 extern double tgamma (double) __attribute__ ((__nothrow__ , __leaf__));
2134   extern double __tgamma (double) __attribute__ ((__nothrow__ , __leaf__));
2135
2136
2137
2138
2139 extern double gamma (double) __attribute__ ((__nothrow__ , __leaf__));
2140   extern double __gamma (double) __attribute__ ((__nothrow__ , __leaf__));

```

```

2141
2142
2143
2144
2145
2146 extern double lgamma_r (double, int * __signgamp) __attribute__ ((  

2147     __nothrow__, __leaf__)); extern double __lgamma_r (double, int *  

2148     __signgamp) __attribute__ ((__nothrow__, __leaf__));
2149
2150
2151
2152
2153
2154 extern double rint (double __x) __attribute__ ((__nothrow__, __leaf__));
2155     extern double __rint (double __x) __attribute__ ((__nothrow__,  

2156     __leaf__));
2157
2158
2159 extern double nextafter (double __x, double __y) __attribute__ ((  

2160     __nothrow__, __leaf__)) __attribute__ ((__const__)); extern double  

2161     __nextafter (double __x, double __y) __attribute__ ((__nothrow__,  

2162     __leaf__)) __attribute__ ((__const__));
2163
2164
2165
2166
2167 extern double nexttoward (double __x, long double __y) __attribute__ ((  

2168     __nothrow__, __leaf__)) __attribute__ ((__const__)); extern double  

2169     __nexttoward (double __x, long double __y) __attribute__ ((__nothrow__,  

2170     __leaf__)) __attribute__ ((__const__));
2171
2172
2173
2174
2175
2176 extern double remainder (double __x, double __y) __attribute__ ((  

2177     __nothrow__, __leaf__)); extern double __remainder (double __x, double  

2178     __y) __attribute__ ((__nothrow__, __leaf__));
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

2177
2178
2179
2180 extern double nearbyint (double __x) __attribute__ ((__nothrow__ ,
2181   __leaf__));
2182 extern double __nearbyint (double __x) __attribute__ (((
2183   __nothrow__ , __leaf__)));
2184
2185 extern double round (double __x) __attribute__ ((__nothrow__ , __leaf__))
2186   __attribute__ ((__const__));
2187 extern double __round (double __x)
2188   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2189
2190
2191
2192
2193 extern double trunc (double __x) __attribute__ ((__nothrow__ , __leaf__))
2194   __attribute__ ((__const__));
2195 extern double __trunc (double __x)
2196   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2197
2198
2199
2200 extern long int lrint (double __x) __attribute__ ((__nothrow__ , __leaf__))
2201   __attribute__ ((__const__));
2202 extern long int __lrint (double __x) __attribute__ ((__nothrow__ ,
2203   __leaf__));
2204
2205
2206
2207 __extension__
2208 extern long long int llrint (double __x) __attribute__ ((__nothrow__ ,
2209   __leaf__));
2210 extern long long int __llrint (double __x) __attribute__ (((
2211   __nothrow__ , __leaf__)));
2212
2213
2214
2215
2216
2217 __extension__
2218 extern long long int llround (double __x) __attribute__ ((__nothrow__ ,
2219   __leaf__));
2220 extern long long int __llround (double __x) __attribute__ (((
2221   __nothrow__ , __leaf__)));
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

2213
2214
2215 extern double fmax (double __x, double __y) __attribute__ ((__nothrow__ ,
2216   __leaf__)) __attribute__ ((__const__));
2217   extern double __fmax (double __x, double __y) __attribute__ ((__nothrow__ ,
2218   __leaf__)) __attribute__ ((__const__));
2219
2220
2221
2222 extern int __fpclassify (double __value) __attribute__ ((__nothrow__ ,
2223   __leaf__))
2224     __attribute__ ((__const__));
2225
2226 extern int __signbit (double __value) __attribute__ ((__nothrow__ ,
2227   __leaf__))
2228     __attribute__ ((__const__));
2229
2230
2231 extern double fma (double __x, double __y, double __z) __attribute__ ((
2232   __nothrow__ , __leaf__));
2233   extern double __fma (double __x, double __y,
2234   double __z) __attribute__ ((__nothrow__ , __leaf__));
2235
2236 # 371 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2237 extern double scalb (double __x, double __n) __attribute__ ((__nothrow__ ,
2238   __leaf__));
2239   extern double __scalb (double __x, double __n)
2240     __attribute__ ((__nothrow__ , __leaf__));
2241 # 70 "/usr/include/math.h" 2 3 4
2242 # 88 "/usr/include/math.h" 3 4
2243 # 1 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 1 3 4
2244 # 52 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2245
2246
2247 extern float acosf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2248   extern float __acosf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2249
2250
2251 extern float asinf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2252   extern float __asinf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2253
2254
2255 extern float atanf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2256   extern float __atanf (float __x) __attribute__ ((__nothrow__ , __leaf__));

```

```

2249
2250 extern float atan2f (float __y, float __x) __attribute__ ((__nothrow__ ,
2251     __leaf__));
2252 extern float __atan2f (float __y, float __x) __attribute__ ((__nothrow__ ,
2253     __leaf__));
2254
2255 extern float cosf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2256 extern float __cosf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2257
2258 extern float sinf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2259 extern float __sinf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2260
2261
2262 extern float tanf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2263 extern float __tanf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2264
2265 extern float coshf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2266 extern float __coshf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2267
2268 extern float sinhf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2269 extern float __sinhf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2270
2271 extern float tanhf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2272 extern float __tanhf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2273
2274 # 86 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2275
2276
2277
2278
2279
2280
2281
2282

```

```

2283 extern float expf (float __x) __attribute__((__nothrow__, __leaf__));
    extern float __expf (float __x) __attribute__((__nothrow__, __leaf__));
);
2284
2285
2286 extern float frexpf (float __x, int *__exponent) __attribute__((
    __nothrow__, __leaf__)); extern float __frexpf (float __x, int *
__exponent) __attribute__((__nothrow__, __leaf__));
2287
2288
2289 extern float ldexpf (float __x, int __exponent) __attribute__(((
    __nothrow__, __leaf__)); extern float __ldexpf (float __x, int
__exponent) __attribute__((__nothrow__, __leaf__));
2290
2291
2292 extern float logf (float __x) __attribute__((__nothrow__, __leaf__));
    extern float __logf (float __x) __attribute__((__nothrow__, __leaf__));
);
2293
2294
2295 extern float log10f (float __x) __attribute__((__nothrow__, __leaf__));
    extern float __log10f (float __x) __attribute__((__nothrow__,
__leaf__));
2296
2297
2298 extern float modff (float __x, float *__iptr) __attribute__((__nothrow__
, __leaf__)); extern float __modff (float __x, float *__iptr)
__attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(
2)));
2299
2300 # 126 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2301
2302
2303 extern float expm1f (float __x) __attribute__((__nothrow__, __leaf__));
    extern float __expm1f (float __x) __attribute__((__nothrow__,
__leaf__));
2304
2305
2306 extern float log1pf (float __x) __attribute__((__nothrow__, __leaf__));
    extern float __log1pf (float __x) __attribute__((__nothrow__,
__leaf__));
2307
2308
2309 extern float logbf (float __x) __attribute__((__nothrow__, __leaf__));
    extern float __logbf (float __x) __attribute__((__nothrow__,
__leaf__));
2310
2311
2312
2313
2314
2315
2316 extern float exp2f (float __x) __attribute__((__nothrow__, __leaf__));
    extern float __exp2f (float __x) __attribute__((__nothrow__,
__leaf__));

```

```
2317 );
2318
2319 extern float log2f (float __x) __attribute__((__nothrow__, __leaf__));
2320   extern float __log2f (float __x) __attribute__((__nothrow__, __leaf__));
2321 );
2322
2323
2324
2325
2326
2327
2328 extern float powf (float __x, float __y) __attribute__((__nothrow__, __leaf__));
2329   extern float __powf (float __x, float __y) __attribute__((__nothrow__, __leaf__));
2330
2331 extern float sqrtf (float __x) __attribute__((__nothrow__, __leaf__));
2332   extern float __sqrtf (float __x) __attribute__((__nothrow__, __leaf__));
2333
2334
2335
2336
2337 extern float hypotf (float __x, float __y) __attribute__((__nothrow__, __leaf__));
2338   extern float __hypotf (float __x, float __y) __attribute__((__nothrow__, __leaf__));
2339
2340
2341
2342
2343
2344 extern float cbrtf (float __x) __attribute__((__nothrow__, __leaf__));
2345   extern float __cbrtf (float __x) __attribute__((__nothrow__, __leaf__));
2346
2347
2348
2349
2350
2351
2352
2353 extern float ceilf (float __x) __attribute__((__nothrow__, __leaf__))
2354   __attribute__((__const__));
2355   extern float __ceilf (float __x)
2356   __attribute__((__nothrow__, __leaf__)) __attribute__((__const__));
2357
2358
2359
2360
2361 extern float fabsf (float __x) __attribute__((__nothrow__, __leaf__))
2362   __attribute__((__const__));
2363   extern float __fabsf (float __x)
```

```

2357
2358
2359 extern float floorf (float __x) __attribute__ ((_nothrow_ , _leaf_)) __attribute__ ((_const_));
2360
2361
2362 extern float fmodf (float __x, float __y) __attribute__ ((_nothrow_ ,
2363     _leaf_)); extern float __fmodf (float __x, float __y) __attribute__ ((_nothrow_ ,
2364     _leaf_));
2365
2366
2367 extern int __isinff (float __value) __attribute__ ((_nothrow_ , _leaf_))
2368     __attribute__ ((_const_));
2369
2370 extern int __finitef (float __value) __attribute__ ((_nothrow_ ,
2371     _leaf_)) __attribute__ ((_const_));
2372
2373
2374
2375
2376 extern int isinff (float __value) __attribute__ ((_nothrow_ , _leaf_))
2377     __attribute__ ((_const_));
2378
2379 extern int finitef (float __value) __attribute__ ((_nothrow_ , _leaf_))
2380     __attribute__ ((_const_));
2381
2382 extern float dremf (float __x, float __y) __attribute__ ((_nothrow_ ,
2383     _leaf_)); extern float __dremf (float __x, float __y) __attribute__ ((_nothrow_ ,
2384     _leaf_));
2385
2386
2387 extern float significandf (float __x) __attribute__ ((_nothrow_ ,
2388     _leaf_)); extern float __significandf (float __x) __attribute__ ((_nothrow_ ,
2389     _leaf_));
2390
2391
2392 extern float copysignf (float __x, float __y) __attribute__ ((_nothrow_ ,
2393     _leaf_)) __attribute__ ((_const_)); extern float __copysignf (
2394     float __x, float __y) __attribute__ ((_nothrow_ , _leaf_))
2395     __attribute__ ((_const_));

```

```

2395
2396
2397
2398
2399 extern float nanf (const char *_tagb) __attribute__((__nothrow__ ,
2400   __leaf__)) __attribute__((__const__));
2401   extern float __nanf (const char
2402     *_tagb) __attribute__((__nothrow__ , __leaf__)) __attribute__((
2403       __const__));
2404
2405 extern int __isnanf (float __value) __attribute__((__nothrow__ , __leaf__))
2406   __attribute__((__const__));
2407
2408
2409 extern int isnanf (float __value) __attribute__((__nothrow__ , __leaf__))
2410   __attribute__((__const__));
2411
2412 extern float j0f (float) __attribute__((__nothrow__ , __leaf__));
2413   extern float __j0f (float) __attribute__((__nothrow__ , __leaf__));
2414 extern float j1f (float) __attribute__((__nothrow__ , __leaf__));
2415   extern float __j1f (float) __attribute__((__nothrow__ , __leaf__));
2416 extern float jnf (int, float) __attribute__((__nothrow__ , __leaf__));
2417   extern float __jnf (int, float) __attribute__((__nothrow__ , __leaf__));
2418
2419
2420
2421
2422
2423
2424 extern float erff (float) __attribute__((__nothrow__ , __leaf__));
2425   extern float __erff (float) __attribute__((__nothrow__ , __leaf__));
2426 extern float erfcf (float) __attribute__((__nothrow__ , __leaf__));
2427   extern float __erfcf (float) __attribute__((__nothrow__ , __leaf__));
2428 extern float lgammaf (float) __attribute__((__nothrow__ , __leaf__));
2429   extern float __lgammaf (float) __attribute__((__nothrow__ , __leaf__));
2430
2431

```

```

2432
2433 extern float tgammaf (float) __attribute__((__nothrow__, __leaf__));
2434     extern float __tgammaf (float) __attribute__((__nothrow__, __leaf__));
2435     ;
2436
2437
2438
2439 extern float gammaf (float) __attribute__((__nothrow__, __leaf__));
2440     extern float __gammaf (float) __attribute__((__nothrow__, __leaf__));
2441
2442
2443
2444
2445
2446 extern float lgammaf_r (float, int *__signgamp) __attribute__((
2447     __nothrow__, __leaf__)); extern float __lgammaf_r (float, int *
2448     __signgamp) __attribute__((__nothrow__, __leaf__));
2449
2450
2451
2452
2453
2454 extern float rintf (float __x) __attribute__((__nothrow__, __leaf__));
2455     extern float __rintf (float __x) __attribute__((__nothrow__, __leaf__));
2456
2457
2458
2459 extern float nextafterf (float __x, float __y) __attribute__((__nothrow__,
2460     __leaf__)) __attribute__((__const__)); extern float __nextafterf (
2461     float __x, float __y) __attribute__((__nothrow__, __leaf__))
2462     __attribute__((__const__));
2463
2464
2465
2466
2467 extern float nexttowardf (float __x, long double __y) __attribute__((
2468     __nothrow__, __leaf__)) __attribute__((__const__)); extern float
2469     __nexttowardf (float __x, long double __y) __attribute__((__nothrow__,
2470     __leaf__)) __attribute__((__const__));
2471
2472
2473
2474
2475 extern float remainderf (float __x, float __y) __attribute__((__nothrow__,
2476     __leaf__)); extern float __remainderf (float __x, float __y)
2477     __attribute__((__nothrow__, __leaf__));
2478
2479
2480
2481
2482
2483 extern float scalbnf (float __x, int __n) __attribute__((__nothrow__,
2484     __leaf__)); extern float __scalbnf (float __x, int __n) __attribute__
2485     ((__nothrow__, __leaf__));
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

2469
2470
2471 extern int ilogbf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2472     extern int __ilogbf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2473
2474
2475
2476 extern float scalblnf (float __x, long int __n) __attribute__ (( __nothrow__ , __leaf__));
2477     extern float __scalblnf (float __x, long int __n) __attribute__ ((__nothrow__ , __leaf__));
2478
2479
2480 extern float nearbyintf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2481     extern float __nearbyintf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2482
2483
2484 extern float roundf (float __x) __attribute__ ((__nothrow__ , __leaf__))
2485     __attribute__ ((__const__));
2486     extern float __roundf (float __x)
2487         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2488
2489
2490
2491
2492
2493 extern float truncf (float __x) __attribute__ ((__nothrow__ , __leaf__))
2494     __attribute__ ((__const__));
2495     extern float __truncf (float __x)
2496         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2497
2498
2499
2500 extern long int lrintf (float __x) __attribute__ ((__nothrow__ , __leaf__));
2501     extern long int __lrintf (float __x) __attribute__ ((__nothrow__ ,
2502         __leaf__));
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605

```

```

2506 extern long int lroundf (float __x) __attribute__ ((__nothrow__ , __leaf__))
2507   ); extern long int __lroundf (float __x) __attribute__ ((__nothrow__ ,
2508   __leaf__));
2509 __extension__
2510 extern long long int llroundf (float __x) __attribute__ ((__nothrow__ ,
2511   __leaf__));
2512 extern long long int __llroundf (float __x) __attribute__ ((__nothrow__ ,
2513   __leaf__));
2514
2515 extern float fdimf (float __x, float __y) __attribute__ ((__nothrow__ ,
2516   __leaf__));
2517 extern float __fdimf (float __x, float __y) __attribute__ ((__nothrow__ ,
2518   __leaf__));
2519
2520
2521 extern float fmaxf (float __x, float __y) __attribute__ ((__nothrow__ ,
2522   __leaf__)) __attribute__ ((__const__));
2523 extern float __fmaxf (float __x, float __y) __attribute__ ((__nothrow__ ,
2524   __leaf__)) __attribute__ ((__const__));
2525
2526 extern float fminf (float __x, float __y) __attribute__ ((__nothrow__ ,
2527   __leaf__)) __attribute__ ((__const__));
2528 extern float __fminf (float __x, float __y) __attribute__ ((__nothrow__ ,
2529   __leaf__)) __attribute__ ((__const__));
2530
2531 extern int __fpclassifyf (float __value) __attribute__ ((__nothrow__ ,
2532   __leaf__))
2533   __attribute__ ((__const__));
2534
2535 extern int __signbitf (float __value) __attribute__ ((__nothrow__ ,
2536   __leaf__))
2537   __attribute__ ((__const__));
2538
2539 extern float fmaf (float __x, float __y, float __z) __attribute__ (((
2540   __nothrow__ , __leaf__));
2541 extern float __fmaf (float __x, float __y,
2542   float __z) __attribute__ ((__nothrow__ , __leaf__));
2543
2544
2545 # 371 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2546 extern float scalbf (float __x, float __n) __attribute__ ((__nothrow__ ,
2547   __leaf__));
2548 extern float __scalbf (float __x, float __n) __attribute__ ((__nothrow__ ,
2549   __leaf__));
2550
2551 # 89 "/usr/include/math.h" 2 3 4
2552 # 132 "/usr/include/math.h" 3 4
2553 # 1 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 1 3 4
2554 # 52 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4

```

```

2542
2543
2544 extern long double acosl (long double __x) __attribute__ ((__nothrow__ ,
2545   __leaf__));
2546 extern long double __acosl (long double __x) __attribute__ ((__nothrow__ ,
2547   __leaf__));
2548 extern long double atanl (long double __x) __attribute__ ((__nothrow__ ,
2549   __leaf__));
2550 extern long double __atanl (long double __x) __attribute__ ((__nothrow__ ,
2551   __leaf__));
2552
2553 extern long double atan2l (long double __y, long double __x) __attribute__ ((__nothrow__ ,
2554   __leaf__));
2555 extern long double __atan2l (long double __y, long double __x) __attribute__ ((__nothrow__ ,
2556   __leaf__));
2557
2558
2559
2560
2561
2562 extern long double coshl (long double __x) __attribute__ ((__nothrow__ ,
2563   __leaf__));
2564 extern long double __coshl (long double __x) __attribute__ ((__nothrow__ ,
2565   __leaf__));
2566
2567 # 86 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2568
2569
2570
2571 extern long double sinh1l (long double __x) __attribute__ ((__nothrow__ ,
2572   __leaf__));
2573 extern long double __sinh1l (long double __x) __attribute__ ((__nothrow__ ,
2574   __leaf__));

```

```

2573 extern long double asinhl (long double __x) __attribute__ ((__nothrow__ ,
2574   __leaf__));
2575 extern long double __asinhl (long double __x) __attribute__ ((__nothrow__ ,
2576   __leaf__));
2577
2578
2579
2580
2581
2582
2583 extern long double expl (long double __x) __attribute__ ((__nothrow__ ,
2584   __leaf__));
2585 extern long double __expl (long double __x) __attribute__ ((__nothrow__ ,
2586   __leaf__));
2587
2588
2589 extern long double frexp1 (long double __x, int *__exponent) __attribute__ ((__nothrow__ ,
2590   __leaf__));
2591 extern long double __frexp1 (long double __x, int *__exponent) __attribute__ ((__nothrow__ ,
2592   __leaf__));
2593
2594
2595 extern long double ldexpl (long double __x, int __exponent) __attribute__ ((__nothrow__ ,
2596   __leaf__));
2597 extern long double __ldexpl (long double __x, int __exponent) __attribute__ ((__nothrow__ ,
2598   __leaf__));
2599
2600 # 126 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2601
2602
2603 extern long double expm1 (long double __x) __attribute__ ((__nothrow__ ,
2604   __leaf__));
2605 extern long double __expm1 (long double __x) __attribute__ ((__nothrow__ ,
2606   __leaf__));
2607
2608
2609 extern long double log1pl (long double __x) __attribute__ ((__nothrow__ ,
2610   __leaf__));
2611 extern long double __log1pl (long double __x) __attribute__ ((__nothrow__ ,
2612   __leaf__));

```

```

((__nothrow__ , __leaf__));
2607
2608
2609 extern long double logbl (long double __x) __attribute__ ((__nothrow__ ,
2610   __leaf__)); extern long double __logbl (long double __x) __attribute__ 
2611 ((__nothrow__ , __leaf__));
2612
2613
2614
2615
2616 extern long double exp2l (long double __x) __attribute__ ((__nothrow__ ,
2617   __leaf__)); extern long double __exp2l (long double __x) __attribute__ 
2618 ((__nothrow__ , __leaf__));
2619
2620
2621 extern long double log2l (long double __x) __attribute__ ((__nothrow__ ,
2622   __leaf__)); extern long double __log2l (long double __x) __attribute__ 
2623 ((__nothrow__ , __leaf__));
2624
2625
2626
2627
2628 extern long double powl (long double __x, long double __y) __attribute__ 
2629 ((__nothrow__ , __leaf__)); extern long double __powl (long double __x,
2630   long double __y) __attribute__ ((__nothrow__ , __leaf__));
2631
2632
2633 extern long double sqrtl (long double __x) __attribute__ ((__nothrow__ ,
2634   __leaf__)); extern long double __sqrtl (long double __x) __attribute__ 
2635 ((__nothrow__ , __leaf__));
2636
2637
2638 extern long double hypotl (long double __x, long double __y) __attribute__ 
2639 ((__nothrow__ , __leaf__)); extern long double __hypotl (long double 
2640   __x, long double __y) __attribute__ ((__nothrow__ , __leaf__));
2641
2642
2643
2644 extern long double cbrtl (long double __x) __attribute__ ((__nothrow__ ,
2645   __leaf__)); extern long double __cbrtl (long double __x) __attribute__ 
((__nothrow__ , __leaf__));

```

```

2646
2647
2648
2649
2650
2651
2652
2653 extern long double ceil (long double __x) __attribute__ ((__nothrow__ ,
2654   __leaf__)) __attribute__ ((__const__)); extern long double __ceil (
2655   long double __x) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ (
2656   __const__));
2657
2658
2659 extern long double fabsl (long double __x) __attribute__ ((__nothrow__ ,
2660   __leaf__)) __attribute__ ((__const__)); extern long double __fabsl (
2661   long double __x) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ (
2662   __const__));
2663
2664
2665
2666
2667 extern long double floorl (long double __x) __attribute__ ((__nothrow__ ,
2668   __leaf__)) __attribute__ ((__const__)); extern long double __floorl (
2669   long double __x) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ (
2670   __const__));
2671
2672
2673
2674
2675
2676 extern int __isinfl (long double __value) __attribute__ ((__nothrow__ ,
2677   __leaf__)) __attribute__ ((__const__));
2678
2679 extern int __finitel (long double __value) __attribute__ ((__nothrow__ ,
2680   __leaf__)) __attribute__ ((__const__));
2681
2682 extern long double dreml (long double __x, long double __y) __attribute__ (
2683   __nothrow__ , __leaf__); extern long double __dreml (long double __x
2684   , long double __y) __attribute__ ((__nothrow__ , __leaf__));

```

```

2683
2684
2685
2686 extern long double significndl (long double __x) __attribute__ ((  

2687     __nothrow__ , __leaf__)); extern long double __significndl (long  

2688     double __x) __attribute__ ((__nothrow__ , __leaf__));
2689
2690
2691
2692 extern long double copysignl (long double __x, long double __y)  

2693     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2694     extern long double __copysignl (long double __x, long double __y)  

2695     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2696
2697
2698
2699 extern long double nanl (const char *__tagb) __attribute__ ((__nothrow__ ,  

2700     __leaf__)) __attribute__ ((__const__)); extern long double __nanl (  

2701     const char *__tagb) __attribute__ ((__nothrow__ , __leaf__))
2702     __attribute__ ((__const__));
2703
2704
2705 extern int __isnanl (long double __value) __attribute__ ((__nothrow__ ,  

2706     __leaf__)) __attribute__ ((__const__));
2707
2708
2709 extern int isnanl (long double __value) __attribute__ ((__nothrow__ ,  

2710     __leaf__)) __attribute__ ((__const__));
2711
2712
2713 extern long double j0l (long double) __attribute__ ((__nothrow__ ,  

2714     __leaf__)); extern long double __j0l (long double) __attribute__ ((  

2715     __nothrow__ , __leaf__));
2716 extern long double j1l (long double) __attribute__ ((__nothrow__ ,  

2717     __leaf__)); extern long double __j1l (long double) __attribute__ ((  

2718     __nothrow__ , __leaf__));
2719 extern long double jnl (int, long double) __attribute__ ((__nothrow__ ,  

2720     __leaf__)); extern long double __jnl (int, long double) __attribute__  

2721     ((__nothrow__ , __leaf__));
2722 extern long double y0l (long double) __attribute__ ((__nothrow__ ,  

2723     __leaf__)); extern long double __y0l (long double) __attribute__ ((  

2724     __nothrow__ , __leaf__));
2725 extern long double y1l (long double) __attribute__ ((__nothrow__ ,  

2726     __leaf__)); extern long double __y1l (long double) __attribute__ ((  

2727     __nothrow__ , __leaf__));

```

```

2717 extern long double ynl (int, long double) __attribute__ ((__nothrow__ ,
2718   __leaf__));
2719
2720
2721
2722
2723
2724 extern long double erfl (long double) __attribute__ ((__nothrow__ ,
2725   __leaf__));
2726 extern long double __erfl (long double) __attribute__ (((
2727   __nothrow__ , __leaf__)));
2728
2729
2730
2731
2732
2733 extern long double erfcl (long double) __attribute__ ((__nothrow__ ,
2734   __leaf__));
2735 extern long double __erfcl (long double) __attribute__ (((
2736   __nothrow__ , __leaf__)));
2737
2738
2739 extern long double lgammal (long double) __attribute__ ((__nothrow__ ,
2740   __leaf__));
2741 extern long double __lgammal (long double) __attribute__ (((
2742   __nothrow__ , __leaf__)));
2743
2744
2745
2746 extern long double tgammal (long double) __attribute__ ((__nothrow__ ,
2747   __leaf__));
2748 extern long double __tgammal (long double) __attribute__ (((
2749   __nothrow__ , __leaf__)));
2750
2751
2752
2753
2754 extern long double gammal (long double) __attribute__ ((__nothrow__ ,
2755   __leaf__));
2756 extern long double __gammal (long double) __attribute__ (((
2757   __nothrow__ , __leaf__)));
2758
2759
2760
2761 extern long double lgammal_r (long double, int *__signgamp) __attribute__ (
2762   __nothrow__ , __leaf__);
2763 extern long double __lgammal_r (long double ,
2764   int *__signgamp) __attribute__ ((__nothrow__ , __leaf__));
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774 extern long double rintl (long double __x) __attribute__ ((__nothrow__ ,
2775   __leaf__));
2776 extern long double __rintl (long double __x) __attribute__ (((
2777   __nothrow__ , __leaf__)));

```

```

2755
2756
2757 extern long double nextafterl (long double __x, long double __y)
    __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__const__));
extern long double __nextafterl (long double __x, long double __y)
    __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__const__));
2758
2759 extern long double nexttowardl (long double __x, long double __y)
    __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__const__));
extern long double __nexttowardl (long double __x, long double __y)
    __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__const__));
2760
2761
2762
2763 extern long double remainderl (long double __x, long double __y)
    __attribute__ ((__nothrow__, __leaf__)); extern long double
    __remainderl (long double __x, long double __y) __attribute__ (((
    __nothrow__, __leaf__)));
2764
2765
2766
2767 extern long double scalbnl (long double __x, int __n) __attribute__ (((
    __nothrow__, __leaf__)); extern long double __scalbnl (long double __x
    , int __n) __attribute__ ((__nothrow__, __leaf__));
2768
2769
2770
2771 extern int ilogbl (long double __x) __attribute__ ((__nothrow__, __leaf__))
    ); extern int __ilogbl (long double __x) __attribute__ ((__nothrow__,
    __leaf__));
2772
2773
2774
2775
2776 extern long double scalblnl (long double __x, long int __n) __attribute__ (((
    __nothrow__, __leaf__)); extern long double __scalblnl (long double
    __x, long int __n) __attribute__ ((__nothrow__, __leaf__));
2777
2778
2779
2780 extern long double nearbyintl (long double __x) __attribute__ (((
    __nothrow__, __leaf__)); extern long double __nearbyintl (long double
    __x) __attribute__ ((__nothrow__, __leaf__));
2781
2782
2783
2784 extern long double roundl (long double __x) __attribute__ ((__nothrow__,
    __leaf__)) __attribute__ ((__const__)); extern long double __roundl (
    long double __x) __attribute__ ((__nothrow__, __leaf__)) __attribute__ (((
    __const__)));
2785
2786
2787

```

```

2788 extern long double trunc1 (long double __x) __attribute__((__nothrow__ ,
2789   __leaf__)) __attribute__((__const__)); extern long double __trunc1 (
2790   long double __x) __attribute__((__nothrow__ , __leaf__)) __attribute__(
2791   ((__const__)));
2792
2793 extern long double remquol (long double __x, long double __y, int *__quo)
2794   __attribute__((__nothrow__ , __leaf__)); extern long double __remquol
2795   (long double __x, long double __y, int *__quo) __attribute__((
2796   __nothrow__ , __leaf__));
2797
2798
2799
2800 extern long int lrintl (long double __x) __attribute__((__nothrow__ ,
2801   __leaf__)); extern long int __lrintl (long double __x) __attribute__((
2802   __nothrow__ , __leaf__));
2803 --extension--
2804 extern long long int llrintl (long double __x) __attribute__((__nothrow__ ,
2805   __leaf__)); extern long long int __llrintl (long double __x)
2806   __attribute__((__nothrow__ , __leaf__));
2807
2808 --extension--
2809 extern long long int llroundl (long double __x) __attribute__(((
2810   __nothrow__ , __leaf__)); extern long long int __llroundl (long double
2811   __x) __attribute__((__nothrow__ , __leaf__));
2812
2813
2814
2815 extern long double fdiml (long double __x, long double __y) __attribute__(
2816   ((__nothrow__ , __leaf__)); extern long double __fdiml (long double __x
2817   , long double __y) __attribute__((__nothrow__ , __leaf__));
2818
2819
2820
2821 extern long double fmaxl (long double __x, long double __y) __attribute__(
2822   ((__nothrow__ , __leaf__)) __attribute__((__const__)); extern long
2823   double __fmaxl (long double __x, long double __y) __attribute__((
2824   __nothrow__ , __leaf__)) __attribute__((__const__));
2825
2826
2827
2828 extern long double fminl (long double __x, long double __y) __attribute__(
2829   ((__nothrow__ , __leaf__)) __attribute__((__const__)); extern long
2830   double __fminl (long double __x, long double __y) __attribute__((
2831   __nothrow__ , __leaf__)) __attribute__((__const__));
2832
2833
2834
2835

```

```

2820
2821
2822 extern int __fpclassifyl (long double __value) __attribute__ ((__nothrow__
2823   , __leaf__))
2824   __attribute__ ((__const__));
2825
2826 extern int __signbitl (long double __value) __attribute__ ((__nothrow__ ,
2827   __leaf__))
2828   __attribute__ ((__const__));
2829
2830
2831 extern long double fmal (long double __x, long double __y, long double __z
2832   ) __attribute__ ((__nothrow__ , __leaf__)); extern long double __fmal (
2833   long double __x, long double __y, long double __z) __attribute__ (((
2834   __nothrow__ , __leaf__));
2835
2836 # 371 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2837 extern long double scalbl (long double __x, long double __n) __attribute__
2838   ((__nothrow__ , __leaf__)); extern long double __scalbl (long double
2839   __x, long double __n) __attribute__ ((__nothrow__ , __leaf__));
2840 # 133 "/usr/include/math.h" 2 3 4
2841 # 148 "/usr/include/math.h" 3 4
2842 extern int signgam;
2843 # 189 "/usr/include/math.h" 3 4
2844 enum
2845 {
2846   FP_NAN =
2847
2848   0,
2849   FP_INFINITE =
2850
2851   1,
2852   FP_ZERO =
2853
2854   2,
2855   FP_SUBNORMAL =
2856
2857   3,
2858   FP_NORMAL =
2859
2860   4
2861 };
2862 # 301 "/usr/include/math.h" 3 4
2863 typedef enum
2864 {
2865   _IEEE_ = -1,
2866   _SVID_,
2867   _XOPEN_,
2868   _POSIX_

```

```

2867     _ISOC_
2868 } _LIB_VERSION_TYPE;
2869
2870
2871
2872
2873 extern _LIB_VERSION_TYPE _LIB_VERSION;
2874 # 326 "/usr/include/math.h" 3 4
2875 struct exception
2876 {
2877     int type;
2878     char *name;
2879     double arg1;
2880     double arg2;
2881     double retval;
2882 };
2883
2884
2885
2886
2887
2888 extern int matherr (struct exception *__exc);
2889 # 488 "/usr/include/math.h" 3 4
2890
2891 # 7 "headers/all_headers.h" 2
2892 # 1 "/usr/include/setjmp.h" 1 3 4
2893 # 27 "/usr/include/setjmp.h" 3 4
2894
2895
2896 # 1 "/usr/include/x86_64-linux-gnu/bits/setjmp.h" 1 3 4
2897 # 26 "/usr/include/x86_64-linux-gnu/bits/setjmp.h" 3 4
2898 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
2899 # 27 "/usr/include/x86_64-linux-gnu/bits/setjmp.h" 2 3 4
2900
2901
2902
2903
2904 typedef long int __jmp_buf[8];
2905 # 30 "/usr/include/setjmp.h" 2 3 4
2906 # 1 "/usr/include/x86_64-linux-gnu/bits/sigset.h" 1 3 4
2907 # 31 "/usr/include/setjmp.h" 2 3 4
2908
2909
2910
2911 struct __jmp_buf_tag
2912 {
2913
2914
2915
2916
2917     __jmp_buf __jmpbuf;
2918     int __mask_was_saved;
2919     __sigset_t __saved_mask;
2920 };

```

```

2921
2922
2923
2924
2925 typedef struct __jmp_buf_tag jmp_buf [1];
2926
2927
2928
2929 extern int setjmp (jmp_buf __env) __attribute__ ((__nothrow__));
2930
2931
2932
2933
2934
2935
2936 extern int __sigsetjmp (struct __jmp_buf_tag __env[1], int __savemask)
     __attribute__ ((__nothrow__));
2937
2938
2939
2940 extern int _setjmp (struct __jmp_buf_tag __env[1]) __attribute__ ((
     __nothrow__));
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951 extern void longjmp (struct __jmp_buf_tag __env[1], int __val)
     __attribute__ ((__nothrow__)) __attribute__ ((__noreturn__));
2952
2953
2954
2955
2956
2957
2958
2959
2960 extern void _longjmp (struct __jmp_buf_tag __env[1], int __val)
     __attribute__ ((__nothrow__)) __attribute__ ((__noreturn__));
2961
2962
2963
2964
2965
2966
2967
2968
2969 typedef struct __jmp_buf_tag sigjmp_buf [1];
2970 # 102 "/usr/include/setjmp.h" 3 4
2971 extern void siglongjmp (sigjmp_buf __env, int __val)
     __attribute__ ((__nothrow__)) __attribute__ ((__noreturn__));
2972

```

```

2973 # 112 "/usr/include/setjmp.h" 3 4
2974
2975 # 8 "headers/all_headers.h" 2
2976 # 1 "/usr/include/string.h" 1 3 4
2977 # 27 "/usr/include/string.h" 3 4
2978
2979
2980
2981
2982
2983 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
2984 # 33 "/usr/include/string.h" 2 3 4
2985 # 44 "/usr/include/string.h" 3 4
2986
2987
2988 extern void *memcpy (void *__restrict __dest, const void *__restrict __src
2989   ,
2990   size_t __n) __attribute__((__nothrow__, __leaf__)) __attribute__
2991 ((__nonnull__(1, 2)));
2992
2993 extern void *memmove (void *__dest, const void *__src, size_t __n)
2994   __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
2995 (1, 2)));
2996
2997
2998
2999
3000 extern void *memccpy (void *__restrict __dest, const void *__restrict
3001   __src,
3002   int __c, size_t __n)
3003   __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
3004 (1, 2)));
3005
3006
3007
3008 extern void *memset (void *__s, int __c, size_t __n) __attribute__
3009 ((__nothrow__, __leaf__)) __attribute__((__nonnull__(1)));
3010
3011 extern int memcmp (const void *__s1, const void *__s2, size_t __n)
3012   __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
3013   __attribute__((__nonnull__(1, 2)));
3014 # 96 "/usr/include/string.h" 3 4
3015 extern void *memchr (const void *__s, int __c, size_t __n)
3016   __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
3017   __attribute__((__nonnull__(1)));
3018 # 127 "/usr/include/string.h" 3 4

```

```

3019
3020
3021 extern char *strcpy (char *__restrict __dest, const char *__restrict __src
3022   )
3023   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3024 (1, 2)));
3025
3026 extern char *strncpy (char *__restrict __dest,
3027   const char *__restrict __src, size_t __n)
3028   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3029 (1, 2)));
3030
3031 extern char *strcat (char *__restrict __dest, const char *__restrict __src
3032   )
3033   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3034 (1, 2)));
3035
3036 extern char *strncat (char *__restrict __dest, const char *__restrict
3037   __src,
3038   size_t __n) __attribute__ ((__nothrow__, __leaf__)) __attribute__ 
3039 ((__nonnull__(1, 2)));
3040
3041
3042 extern int strcmp (const char *__s1, const char *__s2)
3043   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3044   __attribute__ ((__nonnull__(1, 2)));
3045
3046 extern int strncmp (const char *__s1, const char *__s2, size_t __n)
3047   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3048   __attribute__ ((__nonnull__(1, 2)));
3049
3050
3051
3052
3053
3054
3055 # 1 "/usr/include/xlocale.h" 1 3 4
3056 # 27 "/usr/include/xlocale.h" 3 4
3057 typedef struct __locale_struct
3058 {
3059   struct __locale_data *__locales[13];
3060

```

```

3062
3063 const unsigned short int *_ctype_b;
3064 const int *_ctype_tolower;
3065 const int *_ctype_toupper;
3066
3067
3068 const char *_names[13];
3069 } *_locale_t;
3070
3071
3072 typedef _locale_t locale_t;
3073 # 164 "/usr/include/string.h" 2 3 4
3074
3075
3076 extern int strcoll_l (const char *_s1, const char *_s2, __locale_t __l)
3077     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3078     __attribute__ ((__nonnull__(1, 2, 3)));
3079
3080 extern size_t strxfrm_l (char *_dest, const char *_src, size_t __n,
3081     __locale_t __l) __attribute__ ((__nothrow__, __leaf__)) __attribute__
3082     ((__nonnull__(2, 4)));
3083
3084
3085
3086 extern char *strdup (const char *_s)
3087     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__malloc__))
3088     __attribute__ ((__nonnull__(1)));
3089
3090
3091
3092
3093
3094 extern char *strndup (const char *_string, size_t __n)
3095     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__malloc__))
3096     __attribute__ ((__nonnull__(1)));
3097 # 211 "/usr/include/string.h" 3 4
3098
3099 # 236 "/usr/include/string.h" 3 4
3100 extern char *strchr (const char *_s, int __c)
3101     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3102     __attribute__ ((__nonnull__(1)));
3103 # 263 "/usr/include/string.h" 3 4
3104 extern char * strrchr (const char *_s, int __c)
3105     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3106     __attribute__ ((__nonnull__(1)));
3107
3108
3109 # 282 "/usr/include/string.h" 3 4

```

```

3110 extern size_t strcspn (const char *_s, const char *_reject)
3111     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3112     __attribute__ ((__nonnull__(1, 2)));
3113
3114 extern size_t strspn (const char *_s, const char *_accept)
3115     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3116     __attribute__ ((__nonnull__(1, 2)));
3117 # 315 "/usr/include/string.h" 3 4
3118 extern char *strpbrk (const char *_s, const char *_accept)
3119     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3120     __attribute__ ((__nonnull__(1, 2)));
3121 # 342 "/usr/include/string.h" 3 4
3122 extern char *strstr (const char *_haystack, const char *_needle)
3123     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3124     __attribute__ ((__nonnull__(1, 2)));
3125
3126 extern char *strtok (char *__restrict __s, const char *__restrict __delim)
3127     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3128     (2)));
3129
3130
3131
3132 extern char *__strtok_r (char *__restrict __s,
3133     const char *__restrict __delim,
3134     char **__restrict __save_ptr)
3135     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3136     (2, 3)));
3137 extern char *strtok_r (char *__restrict __s, const char *__restrict
3138     __delim,
3139         char **__restrict __save_ptr)
3140     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3141     (2, 3)));
3142 # 397 "/usr/include/string.h" 3 4
3143
3144
3145
3146
3147
3148
3149
3150 extern size_t strlen (const char *_s)
3151     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3152     __attribute__ ((__nonnull__(1)));
3153

```

```

3154
3155
3156
3157 extern char *strerror (int __errnum) __attribute__ ((__nothrow__ ,
3158   __leaf__));
3158
3159 # 427 "/usr/include/string.h" 3 4
3160 extern int strerror_r (int __errnum, char *__buf, size_t __buflen) __asm__
3161   ("\" __xpg_strerror_r") __attribute__ ((__nothrow__ , __leaf__))
3162
3163 # 445 "/usr/include/string.h" 3 4
3164 extern char *strerror_l (int __errnum, __locale_t __l) __attribute__ ((
3165   __nothrow__ , __leaf__));
3166
3167
3168
3169
3170 extern void __bzero (void *__s, size_t __n) __attribute__ ((__nothrow__ ,
3171   __leaf__)) __attribute__ ((__nonnull__ (1)));
3172
3173
3174 extern void bcopy (const void *__src, void *__dest, size_t __n)
3175   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1,
3176   2)));
3177
3178 extern void bzero (void *__s, size_t __n) __attribute__ ((__nothrow__ ,
3179   __leaf__)) __attribute__ ((__nonnull__ (1)));
3180
3181
3182 extern int bcmp (const void *__s1, const void *__s2, size_t __n)
3183   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3184   __attribute__ ((__nonnull__ (1, 2)));
3185 # 489 "/usr/include/string.h" 3 4
3186 extern char *index (const char *__s, int __c)
3187   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3188   __attribute__ ((__nonnull__ (1)));
3189 # 517 "/usr/include/string.h" 3 4
3190 extern char *rindex (const char *__s, int __c)
3191   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3192   __attribute__ ((__nonnull__ (1)));
3193
3194
3195 extern int ffs (int __i) __attribute__ ((__nothrow__ , __leaf__))
3196   __attribute__ ((__const__));
3197 # 534 "/usr/include/string.h" 3 4
3198 extern int strcasecmp (const char *__s1, const char *__s2)
3199   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3200   __attribute__ ((__nonnull__ (1, 2)));

```

```

3197
3198
3199 extern int strncasecmp (const char *__s1, const char *__s2, size_t __n)
3200     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3201     __attribute__ ((__nonnull__(1, 2)));
3201 # 557 "/usr/include/string.h" 3 4
3202 extern char *strsep (char **__restrict __stringp,
3203     const char *__restrict __delim)
3204     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3204     (1, 2)));
3205
3206
3207
3208
3209 extern char *strsignal (int __sig) __attribute__ ((__nothrow__, __leaf__))
3209 );
3210
3211
3212 extern char *__stpcpy (char *__restrict __dest, const char *__restrict
3212     __src)
3213     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3213     (1, 2)));
3214 extern char *stpcpy (char *__restrict __dest, const char *__restrict __src
3214 )
3215     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3215     (1, 2)));
3216
3217
3218
3219 extern char *__stpncpy (char *__restrict __dest,
3220     const char *__restrict __src, size_t __n)
3221     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3221     (1, 2)));
3222 extern char *stpncpy (char *__restrict __dest,
3223     const char *__restrict __src, size_t __n)
3224     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3224     (1, 2)));
3225 # 644 "/usr/include/string.h" 3 4
3226
3227 # 9 "headers/all_headers.h" 2
3228 # 1 "headers/sen_basic_type.h" 1
3229 # 1 "headers/sen_object.h" 1
3230
3231
3232
3233
3234
3235
3236
3237 # 7 "headers/sen_object.h"
3238 struct Sen_object_vtable;
3239 typedef struct Sen_object_vtable Sen_object_vtable;
3240
3241 struct Sen_object_class;

```

```

3242 typedef struct Sen_object_class Sen_object_class;
3243
3244 struct Sen_object;
3245 typedef struct Sen_object Sen_object;
3246
3247 struct Sen_object_vtable {
3248     void (*print) (Sen_object *);
3249     Sen_object *(*construct) (void *);
3250     void (*destruct) (Sen_object *);
3251     Sen_object *(*copy) (Sen_object *);
3252 };
3253
3254
3255
3256 struct Sen_object_class {
3257     Sen_object_vtable *tablep;
3258 };
3259
3260
3261
3262 struct Sen_object {
3263
3264 # 32 "headers/sen_object.h" 3 4
3265     _Bool
3266 # 32 "headers/sen_object.h"
3267     bound;
3268     Sen_object_class *classp;
3269 };
3270
3271 extern Sen_object_class Sen_object_class_;
3272 extern Sen_object_vtable Sen_object_vtable_;
3273
3274 void print_object (Sen_object *);
3275 Sen_object * construct_object (void *);
3276 void destruct_object (Sen_object *);
3277 Sen_object * copy_object (Sen_object *);
3278 # 2 "headers/sen_basic_type.h" 2
3279
3280
3281
3282
3283 struct Sen_basic_type_vtable;
3284 typedef struct Sen_basic_type_vtable Sen_basic_type_vtable;
3285
3286 struct Sen_basic_type_class;
3287 typedef struct Sen_basic_type_class Sen_basic_type_class;
3288
3289 struct Sen_basic_type;
3290 typedef struct Sen_basic_type Sen_basic_type;
3291
3292 typedef enum {BOOL, INT, STR, UNK} Type;
3293
3294 struct Sen_basic_type_vtable {
3295     void (*print) (Sen_object *);

```

```

3296     Sen_basic_type *(*construct) (void *);
3297     void *(*get_val) (Sen_basic_type *);
3298     void *(*set_val) (Sen_basic_type *, void *);
3299     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
3300 };
3301
3302 struct Sen_basic_type_class {
3303     Sen_object_class *superp;
3304     Sen_basic_type_vtable *tablep;
3305     Type type;
3306 };
3307
3308 struct Sen_basic_type {
3309 # 32 "headers/sen_basic_type.h" 3 4
3310     _Bool
3311 # 32 "headers/sen_basic_type.h"
3312     bound;
3313     Sen_basic_type_class *classp;
3314     Sen_object *superp;
3315 };
3316
3317
3318 extern Sen_basic_type_class Sen_basic_type_class_;
3319 extern Sen_basic_type_vtable Sen_basic_type_vtable_;
3320
3321 void * get_val_basic_type (Sen_basic_type *);
3322 void * set_val_basic_type (Sen_basic_type *, void *);
3323 Sen_basic_type * add_basic_type (Sen_basic_type *, Sen_basic_type *);
3324 # 10 "headers/all_headers.h" 2
3325 # 1 "headers/sen_int.h" 1
3326 # 1 "headers/sen_basic_type.h" 1
3327 # 2 "headers/sen_int.h" 2
3328
3329
3330
3331
3332 struct Sen_int_vtable;
3333 typedef struct Sen_int_vtable Sen_int_vtable;
3334
3335 struct Sen_int_class;
3336 typedef struct Sen_int_class Sen_int_class;
3337
3338 struct Sen_int;
3339 typedef struct Sen_int Sen_int;
3340
3341 struct Sen_int_vtable {
3342     void (*print) (Sen_object *);
3343     void *(*get_val) (Sen_basic_type *);
3344     void *(*set_val) (Sen_basic_type *, void *);
3345     Sen_int *(*construct) (int);
3346     void (*destruct) (Sen_int *);
3347     Sen_int *(*copy) (Sen_int *);
3348     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
3349 };

```

```

3350
3351 struct Sen_int_class {
3352     Sen_basic_type_class *superp;
3353     Sen_int_vtable *tablep;
3354     Type type;
3355 };
3356
3357 struct Sen_int {
3358
3359 # 32 "headers/sen_int.h" 3 4
3360     _Bool
3361 # 32 "headers/sen_int.h"
3362     bound;
3363     Sen_int_class *classp;
3364     Sen_basic_type *superp;
3365     int val;
3366 };
3367
3368 extern Sen_int_class Sen_int_class_;
3369 extern Sen_int_vtable Sen_int_vtable_;
3370
3371 void print_int (Sen_object *);
3372 Sen_int * construct_int (int);
3373 void *get_val_int (Sen_basic_type *);
3374 void *set_val_int (Sen_basic_type *, void *);
3375 # 11 "headers/all_headers.h" 2
3376
3377 # 1 "headers/sen_bool.h" 1
3378 # 1 "headers/sen_basic_type.h" 1
3379 # 2 "headers/sen_bool.h" 2
3380
3381
3382
3383
3384 struct Sen_bool_vtable;
3385 typedef struct Sen_bool_vtable Sen_bool_vtable;
3386
3387 struct Sen_bool_class;
3388 typedef struct Sen_bool_class Sen_bool_class;
3389
3390 struct Sen_bool;
3391 typedef struct Sen_bool Sen_bool;
3392
3393 struct Sen_bool_vtable {
3394     void (*print) (Sen_object *);
3395     void *(*get_val) (Sen_basic_type *);
3396     void *(*set_val) (Sen_basic_type *, void *);
3397     Sen_bool *(*construct) (
3398 # 19 "headers/sen_bool.h" 3 4
3399             _Bool
3400 # 19 "headers/sen_bool.h"
3401             );
3402     void (*destruct) (Sen_bool *);
3403     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);

```

```

3404 };
3405
3406 struct Sen_bool_class {
3407     Sen_basic_type_class *superp;
3408     Sen_bool_vtable *tablep;
3409     Type type;
3410 };
3411
3412 struct Sen_bool {
3413
3414 # 31 "headers/sen_bool.h" 3 4
3415     _Bool
3416 # 31 "headers/sen_bool.h"
3417     bound;
3418     Sen_bool_class *classp;
3419     Sen_basic_type *superp;
3420
3421 # 34 "headers/sen_bool.h" 3 4
3422     _Bool
3423 # 34 "headers/sen_bool.h"
3424     val;
3425 };
3426
3427 extern Sen_bool_class Sen_bool_class_;
3428 extern Sen_bool_vtable Sen_bool_vtable_;
3429
3430 void print_bool (Sen_object *);
3431 Sen_bool * construct_bool (
3432 # 41 "headers/sen_bool.h" 3 4
3433             _Bool
3434 # 41 "headers/sen_bool.h"
3435             );
3436 void *get_val_bool (Sen_basic_type *);
3437 void *set_val_bool (Sen_basic_type *, void *);
3438 # 13 "headers/all_headers.h" 2
3439 # 1 "headers/sen_string.h" 1
3440 # 1 "headers/sen_basic_type.h" 1
3441 # 2 "headers/sen_string.h" 2
3442
3443
3444
3445
3446 struct Sen_string_vtable;
3447 typedef struct Sen_string_vtable Sen_string_vtable;
3448
3449 struct Sen_string_class;
3450 typedef struct Sen_string_class Sen_string_class;
3451
3452 struct Sen_string;
3453 typedef struct Sen_string Sen_string;
3454
3455 struct Sen_string_vtable {
3456     void (*print) (Sen_object *);
3457     Sen_string *(*construct) (char *);

```

```

3458     void (*destruct) (Sen_string *);
3459     Sen_string *(*copy) (Sen_string *);
3460     void *(*get_val) (Sen_basic_type *);
3461     void *(*set_val) (Sen_basic_type *, void *);
3462     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
3463 };
3464
3465 struct Sen_string_class {
3466     Sen_basic_type_class *superp;
3467     Sen_string_vtable *tablep;
3468     Type type;
3469 };
3470
3471 struct Sen_string {
3472 # 32 "headers/sen_string.h" 3 4
3473     _Bool
3474 # 32 "headers/sen_string.h"
3475     bound;
3476     Sen_string_class *classp;
3477     Sen_basic_type *superp;
3478     char *val;
3479 };
3480 };
3481
3482 extern Sen_string_class Sen_string_class_;
3483 extern Sen_string_vtable Sen_string_vtable_;
3484
3485 void print_string (Sen_object *);
3486 Sen_string * construct_string (char *);
3487 void destruct (Sen_string *);
3488 Sen_string * copy_string (Sen_string *);
3489 void *get_val_string (Sen_basic_type *);
3490 void *set_val_string (Sen_basic_type *, void *);
3491 # 14 "headers/all_headers.h" 2
3492 # 1 "headers/sen_array.h" 1
3493
3494 # 1 "headers/sen_basic_type.h" 1
3495 # 3 "headers/sen_array.h" 2
3496 # 1 "headers/sen_int.h" 1
3497 # 1 "headers/sen_basic_type.h" 1
3498 # 2 "headers/sen_int.h" 2
3499 # 4 "headers/sen_array.h" 2
3500 # 12 "headers/sen_array.h"
3501 struct Sen_array_vtable;
3502 typedef struct Sen_array_vtable Sen_array_vtable;
3503
3504 struct Sen_array_class;
3505 typedef struct Sen_array_class Sen_array_class;
3506
3507 struct Sen_array;
3508 typedef struct Sen_array Sen_array;
3509
3510 struct Sen_array_vtable {
3511     void (*print) (Sen_object *);

```

```

3512     Sen_array *(*construct) (Sen_object **, int);
3513     void (*destruct) (Sen_array *);
3514     Sen_object *(*access) (Sen_array *, Sen_int *);
3515     Sen_array *(*concat) (Sen_array *, Sen_array *);
3516 };
3517
3518 struct Sen_array_class {
3519     Sen_object_class *superp;
3520     Sen_array_vtable *tablep;
3521 };
3522
3523 struct Sen_array {
3524
3525 # 35 "headers/sen_array.h" 3 4
3526     _Bool
3527 # 35 "headers/sen_array.h"
3528     bound;
3529     Sen_array_class *classp;
3530     Sen_object *superp;
3531     void **arr;
3532     int len;
3533     char print_sep;
3534 };
3535
3536 extern Sen_array_class Sen_array_class_;
3537 extern Sen_array_vtable Sen_array_vtable_;
3538
3539 Sen_array *construct_array (Sen_object **, int);
3540 Sen_object *access_array (Sen_array *, Sen_int *);
3541 Sen_array *concat_array (Sen_array *, Sen_array *);
3542 # 15 "headers/all_headers.h" 2
3543 # 2 "main.c" 2
3544
3545 int main() {
3546 # 29 "main.c"
3547     __auto_type s = ((Sen_string*) construct_string("tttesting "));
3548     s->bound=
3549 # 30 "main.c" 3 4
3550         1
3551 # 30 "main.c"
3552             ;
3553     __auto_type ss = ((Sen_string*) construct_string("hooray!!\n"));
3554     ss->bound=
3555 # 32 "main.c" 3 4
3556         1
3557 # 32 "main.c"
3558         ;
3559
3560
3561     { typeof(((Sen_string *) ss)) __temp__ = ((Sen_string *) ss); __temp__
3562     ->classp->tablep->print(((Sen_object *) __temp__));
3563     { typeof(((Sen_string *))({ __auto_type __temp__ = s; __temp__->classp
3564     ->tablep->add((Sen_basic_type *)__temp__, (Sen_basic_type *)ss); }))
3565     __temp__ = ((Sen_string *)({ __auto_type __temp__ = s; __temp__->classp

```

```

    ->tablep->add((Sen_basic_type *)__temp__, (Sen_basic_type *)ss); }));  

__temp__->classp->tablep->print(((Sen_object *)__temp__)); };  

{ typeof(((Sen_string *)){ __auto_type __temp__ = ((Sen_string *)){  

__auto_type __temp__ = s; __temp__->classp->tablep->add((Sen_basic_type  

*)__temp__, (Sen_basic_type *)ss); }}); __temp__->classp->tablep->add  

((Sen_basic_type *)__temp__, (Sen_basic_type *)((Sen_string *)){  

__auto_type __temp__ = s; __temp__->classp->tablep->add((Sen_basic_type  

*)__temp__, (Sen_basic_type *)ss); }})); __temp__ = ((Sen_string  

*)){ __auto_type __temp__ = ((Sen_string *)){ __auto_type __temp__ = s;  

__temp__->classp->tablep->add((Sen_basic_type *)__temp__, (  

Sen_basic_type *)ss); }}); __temp__->classp->tablep->add((  

Sen_basic_type *)__temp__, (Sen_basic_type *)((Sen_string *)){  

__auto_type __temp__ = s; __temp__->classp->tablep->add((Sen_basic_type  

*)__temp__, (Sen_basic_type *)ss); }})); __temp__->classp->tablep  

->print(((Sen_object *)__temp__)); };  

((Sen_string *) s)->classp->tablep->destruct(((Sen_string *) s));  

((Sen_string *) ss)->classp->tablep->destruct(((Sen_string *) ss));  

3566 Sen_int *arr_[2] = {((Sen_int*) construct_int(100)), ((Sen_int*)  

construct_int(50))};  

3567 printf ("%d %d\n", (int)sizeof(arr_), arr_[0]->bound);  

3568 Sen_array *arr = (Sen_array_vtable_.construct(arr_, (sizeof((arr_)) /  

sizeof((arr_)[0]))));  

3569 printf ("%d %d\n", (int)sizeof(arr_), arr_[0]->bound);  

3570 printf("%d %d %d\n", (int)((arr->arr)[0]), (sizeof((arr_)) / sizeof((  

arr_)[0])), arr_[0]->bound);  

3571  

3572     return 0;  

3573 }

```

./c\_files/test

```

1 # 1 "main.c"  

2 # 1 "<built-in>"  

3 # 1 "<command-line>"  

4 # 1 "/usr/include/stdc-predef.h" 1 3 4  

5 # 1 "<command-line>" 2  

6 # 1 "main.c"  

7 # 1 "headers/all_headers.h" 1  

8  

9  

10 # 1 "/usr/include/stdlib.h" 1 3 4  

11 # 24 "/usr/include/stdlib.h" 3 4  

12 # 1 "/usr/include/features.h" 1 3 4  

13 # 374 "/usr/include/features.h" 3 4  

14 # 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4  

15 # 385 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4  

16 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4  

17 # 386 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4  

18 # 375 "/usr/include/features.h" 2 3 4  

19 # 398 "/usr/include/features.h" 3 4  

20 # 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4  

21 # 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4  

22 # 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4  

23 # 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4  

24 # 399 "/usr/include/features.h" 2 3 4

```

```

25 # 25 "/usr/include/stdlib.h" 2 3 4
26
27
28
29
30
31
32
33 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
34 # 216 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 3 4
35
36 # 216 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 3 4
37 typedef long unsigned int size_t;
38 # 328 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 3 4
39 typedef int wchar_t;
40 # 33 "/usr/include/stdlib.h" 2 3 4
41
42
43
44
45
46
47
48
49 # 1 "/usr/include/x86_64-linux-gnu/bits/waitflags.h" 1 3 4
50 # 50 "/usr/include/x86_64-linux-gnu/bits/waitflags.h" 3 4
51 typedef enum
52 {
53     P_ALL,
54     P_PID,
55     P_PGID
56 } idtype_t;
57 # 42 "/usr/include/stdlib.h" 2 3 4
58 # 1 "/usr/include/x86_64-linux-gnu/bits/waitstatus.h" 1 3 4
59 # 64 "/usr/include/x86_64-linux-gnu/bits/waitstatus.h" 3 4
60 # 1 "/usr/include/endian.h" 1 3 4
61 # 36 "/usr/include/endian.h" 3 4
62 # 1 "/usr/include/x86_64-linux-gnu/bits/endian.h" 1 3 4
63 # 37 "/usr/include/endian.h" 2 3 4
64 # 60 "/usr/include/endian.h" 3 4
65 # 1 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 1 3 4
66 # 27 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 3 4
67 # 1 "/usr/include/x86_64-linux-gnu/bits/types.h" 1 3 4
68 # 27 "/usr/include/x86_64-linux-gnu/bits/types.h" 3 4
69 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
70 # 28 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
71
72
73 typedef unsigned char __u_char;
74 typedef unsigned short int __u_short;
75 typedef unsigned int __u_int;
76 typedef unsigned long int __u_long;
77
78

```

```

79 typedef signed char __int8_t;
80 typedef unsigned char __uint8_t;
81 typedef signed short int __int16_t;
82 typedef unsigned short int __uint16_t;
83 typedef signed int __int32_t;
84 typedef unsigned int __uint32_t;
85
86 typedef signed long int __int64_t;
87 typedef unsigned long int __uint64_t;
88
89
90
91
92
93
94
95 typedef long int __quad_t;
96 typedef unsigned long int __u_quad_t;
97 # 121 "/usr/include/x86_64-linux-gnu/bits/types.h" 3 4
98 # 1 "/usr/include/x86_64-linux-gnu/bits/typesizes.h" 1 3 4
99 # 122 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
100
101
102 typedef unsigned long int __dev_t;
103 typedef unsigned int __uid_t;
104 typedef unsigned int __gid_t;
105 typedef unsigned long int __ino_t;
106 typedef unsigned long int __ino64_t;
107 typedef unsigned int __mode_t;
108 typedef unsigned long int __nlink_t;
109 typedef long int __off_t;
110 typedef long int __off64_t;
111 typedef int __pid_t;
112 typedef struct { int __val[2]; } __fsid_t;
113 typedef long int __clock_t;
114 typedef unsigned long int __rlim_t;
115 typedef unsigned long int __rlim64_t;
116 typedef unsigned int __id_t;
117 typedef long int __time_t;
118 typedef unsigned int __useconds_t;
119 typedef long int __suseconds_t;
120
121 typedef int __daddr_t;
122 typedef int __key_t;
123
124
125 typedef int __clockid_t;
126
127
128 typedef void * __timer_t;
129
130
131 typedef long int __blksize_t;
132
```

```

133
134
135
136 typedef long int __blkcnt_t;
137 typedef long int __blkcnt64_t;
138
139
140 typedef unsigned long int __fsblkcnt_t;
141 typedef unsigned long int __fsblkcnt64_t;
142
143
144 typedef unsigned long int __fsfilcnt_t;
145 typedef unsigned long int __fsfilcnt64_t;
146
147
148 typedef long int __fsword_t;
149
150 typedef long int __ssize_t;
151
152
153 typedef long int __syscall_slong_t;
154
155 typedef unsigned long int __syscall_ulong_t;
156
157
158
159 typedef __off64_t __loff_t;
160 typedef __quad_t * __qaddr_t;
161 typedef char * __caddr_t;
162
163
164 typedef long int __intptr_t;
165
166
167 typedef unsigned int __socklen_t;
168 # 28 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 2 3 4
169 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
170 # 29 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 2 3 4
171
172
173
174
175
176
177 # 1 "/usr/include/x86_64-linux-gnu/bits/bytesswap-16.h" 1 3 4
178 # 36 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 2 3 4
179 # 44 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 3 4
180 static __inline unsigned int
181 __bswap_32 (unsigned int __bsx)
182 {
183     return __builtin_bswap32 (__bsx);
184 }
185 # 108 "/usr/include/x86_64-linux-gnu/bits/bytesswap.h" 3 4
186 static __inline __uint64_t

```

```

187 __bswap_64 (__uint64_t __bsx)
188 {
189     return __builtin_bswap64 (__bsx);
190 }
# 61 "/usr/include/endian.h" 2 3 4
# 65 "/usr/include/x86_64-linux-gnu/bits/waitstatus.h" 2 3 4
193
194 union wait
195 {
196     int w_status;
197     struct
198     {
199         unsigned int __w_termsig:7;
200         unsigned int __w_coredump:1;
201         unsigned int __w_retcode:8;
202         unsigned int:16;
203
204
205
206
207
208
209
210
211         } __wait_terminated;
212         struct
213         {
214             unsigned int __w_stopval:8;
215             unsigned int __w_stopsig:8;
216             unsigned int:16;
217
218
219
220
221
222
223
224         } __wait_stopped;
225     };
# 43 "/usr/include/stdlib.h" 2 3 4
# 67 "/usr/include/stdlib.h" 3 4
228 typedef union
229 {
230     union wait *__uptr;
231     int * __iptr;
232 } __WAIT_STATUS __attribute__((__transparent_union__));
# 95 "/usr/include/stdlib.h" 3 4
234
235
236 typedef struct
237 {
238     int quot;
239     int rem;
240 } div_t;

```

```

241
242
243
244 typedef struct
245     {
246         long int quot;
247         long int rem;
248     } ldiv_t;
249
250
251
252
253
254
255
256 --extension-- typedef struct
257     {
258         long long int quot;
259         long long int rem;
260     } lldiv_t;
261
262
263 # 139 "/usr/include/stdlib.h" 3 4
264 extern size_t __ctype_get_mb_cur_max (void) __attribute__ ((__nothrow__ ,
--leaf__)) ;
265
266
267
268
269 extern double atof (const char *__nptra)
270     --attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__ ))
271     --attribute__ ((__nonnull__ (1))) ;
272
273 extern int atoi (const char *__nptra)
274     --attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__ ))
275     --attribute__ ((__nonnull__ (1))) ;
276
277 extern long int atol (const char *__nptra)
278     --attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__ ))
279     --attribute__ ((__nonnull__ (1))) ;
280
281
282 --extension-- extern long long int atoll (const char *__nptra)
283     --attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__ ))
284     --attribute__ ((__nonnull__ (1))) ;
285
286
287
288
289 extern double strtod (const char *__restrict __nptra ,

```

```

290         char **__restrict __endptr)
291     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
292     (1)));
293
294
295
296
297 extern float strtod (const char *__restrict __nptra,
298     char **__restrict __endptr) __attribute__((__nothrow__, __leaf__))
299     __attribute__((__nonnull__(1)));
300
301 extern long double strtold (const char *__restrict __nptra,
302     char **__restrict __endptr)
303     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
304     (1)));
305
306
307
308 extern long int strtol (const char *__restrict __nptra,
309     char **__restrict __endptr, int __base)
310     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
311     (1)));
312
313 extern unsigned long int strtoul (const char *__restrict __nptra,
314     char **__restrict __endptr, int __base)
315     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
316     (1)));
317
318
319 __extension__
320 extern long long int strtoq (const char *__restrict __nptra,
321     char **__restrict __endptr, int __base)
322     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
323     (1)));
324
325 __extension__
326 extern unsigned long long int strtouq (const char *__restrict __nptra,
327     char **__restrict __endptr, int __base)
328     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
329     (1)));
330
331
332
333 __extension__
334 extern long long int strtoll (const char *__restrict __nptra,
335     char **__restrict __endptr, int __base)

```

```

336     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__))
337     (1));
338
339 __extension__
340 extern unsigned long long int strtoull (const char *__restrict __nptra,
341     char **__restrict __endptr, int __base)
342     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__))
343     (1));
344
345
346
347 extern long int a64l (const char *__s)
348     __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
349     __attribute__((__nonnull__(1)));
350
351
352
353 # 1 "/usr/include/x86_64-linux-gnu/sys/types.h" 1 3 4
354 # 27 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
355
356
357
358
359
360
361 typedef __u_char u_char;
362 typedef __u_short u_short;
363 typedef __u_int u_int;
364 typedef __u_long u_long;
365 typedef __quad_t quad_t;
366 typedef __u_quad_t u_quad_t;
367 typedef __fsid_t fsid_t;
368
369
370
371
372 typedef __loff_t loff_t;
373
374
375
376 typedef __ino_t ino_t;
377 # 60 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
378 typedef __dev_t dev_t;
379
380
381
382
383 typedef __gid_t gid_t;
384
385
```

```
386
387
388 typedef __mode_t mode_t;
389
390
391
392
393 typedef __nlink_t nlink_t;
394
395
396
397
398 typedef __uid_t uid_t;
399
400
401
402
403
404 typedef __off_t off_t;
405 # 98 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
406 typedef __pid_t pid_t;
407
408
409
410
411
412 typedef __id_t id_t;
413
414
415
416
417 typedef __ssize_t ssize_t;
418
419
420
421
422
423 typedef __daddr_t daddr_t;
424 typedef __caddr_t caddr_t;
425
426
427
428
429
430 typedef __key_t key_t;
431 # 132 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
432 # 1 "/usr/include/time.h" 1 3 4
433 # 57 "/usr/include/time.h" 3 4
434
435
436 typedef __clock_t clock_t;
437
438
439
```

```

440 # 73 "/usr/include/time.h" 3 4
441
442
443 typedef __time_t time_t;
444
445
446
447 # 91 "/usr/include/time.h" 3 4
448 typedef __clockid_t clockid_t;
449 # 103 "/usr/include/time.h" 3 4
450 typedef __timer_t timer_t;
451 # 133 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
452 # 146 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
453 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
454 # 147 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
455
456
457
458 typedef unsigned long int ulong;
459 typedef unsigned short int ushort;
460 typedef unsigned int uint;
461 # 194 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
462 typedef int int8_t __attribute__((__mode__(__QI__)));
463 typedef int int16_t __attribute__((__mode__(__HI__)));
464 typedef int int32_t __attribute__((__mode__(__SI__)));
465 typedef int int64_t __attribute__((__mode__(__DI__)));
466
467
468 typedef unsigned int u_int8_t __attribute__((__mode__(__QI__)));
469 typedef unsigned int u_int16_t __attribute__((__mode__(__HI__)));
470 typedef unsigned int u_int32_t __attribute__((__mode__(__SI__)));
471 typedef unsigned int u_int64_t __attribute__((__mode__(__DI__)));
472
473 typedef int register_t __attribute__((__mode__(__word__)));
474 # 219 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
475 # 1 "/usr/include/x86_64-linux-gnu/sys/select.h" 1 3 4
476 # 30 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
477 # 1 "/usr/include/x86_64-linux-gnu/bits/select.h" 1 3 4
478 # 22 "/usr/include/x86_64-linux-gnu/bits/select.h" 3 4
479 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
480 # 23 "/usr/include/x86_64-linux-gnu/bits/select.h" 2 3 4
481 # 31 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
482
483
484 # 1 "/usr/include/x86_64-linux-gnu/bits/sigset.h" 1 3 4
485 # 22 "/usr/include/x86_64-linux-gnu/bits/sigset.h" 3 4
486 typedef int __sig_atomic_t;
487
488
489
490
491 typedef struct
492 {
493     unsigned long int __val[(1024 / (8 * sizeof (unsigned long int)))];
```

```

494     } __sigset_t;
495 # 34 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
496
497
498
499 typedef __sigset_t sigset_t;
500
501
502
503
504
505 # 1 "/usr/include/time.h" 1 3 4
506 # 120 "/usr/include/time.h" 3 4
507 struct timespec
508 {
509     __time_t tv_sec;
510     __syscall_slong_t tv_nsec;
511 };
512 # 44 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
513
514 # 1 "/usr/include/x86_64-linux-gnu/bits/time.h" 1 3 4
515 # 30 "/usr/include/x86_64-linux-gnu/bits/time.h" 3 4
516 struct timeval
517 {
518     __time_t tv_sec;
519     __suseconds_t tv_usec;
520 };
521 # 46 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
522
523
524 typedef __suseconds_t suseconds_t;
525
526
527
528
529
530 typedef long int __fd_mask;
531 # 64 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
532 typedef struct
533 {
534
535
536
537
538
539
540     __fd_mask __fds_bits[1024 / (8 * (int) sizeof (__fd_mask))];
541
542 }
543 fd_set;
544
545
546
547

```

```

548
549
550 typedef __fd_mask fd_mask;
551 # 96 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
552
553 # 106 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
554 extern int select (int __nfds, fd_set *__restrict __readfds,
555     fd_set *__restrict __writefds,
556     fd_set *__restrict __exceptfds,
557     struct timeval *__restrict __timeout);
558 # 118 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
559 extern int pselect (int __nfds, fd_set *__restrict __readfds,
560     fd_set *__restrict __writefds,
561     fd_set *__restrict __exceptfds,
562     const struct timespec *__restrict __timeout,
563     const __sigset_t *__restrict __sigmask);
564 # 131 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
565
566 # 220 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
567
568
569 # 1 "/usr/include/x86_64-linux-gnu/sys/sysmacros.h" 1 3 4
570 # 24 "/usr/include/x86_64-linux-gnu/sys/sysmacros.h" 3 4
571
572
573 --extension--
574 extern unsigned int gnu_dev_major (unsigned long long int __dev)
575     --attribute__ ((--nothrow__, --leaf__)) --attribute__ ((--const__));
576 --extension--
577 extern unsigned int gnu_dev_minor (unsigned long long int __dev)
578     --attribute__ ((--nothrow__, --leaf__)) --attribute__ ((--const__));
579 --extension--
580 extern unsigned long long int gnu_dev_makedev (unsigned int __major,
581     unsigned int __minor)
582     --attribute__ ((--nothrow__, --leaf__)) --attribute__ ((--const__));
583 # 58 "/usr/include/x86_64-linux-gnu/sys/sysmacros.h" 3 4
584
585 # 223 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
586
587
588
589
590
591 typedef __blksize_t blksize_t;
592
593
594
595
596
597
598 typedef __blkcnt_t blkcnt_t;
599
600
601

```

```

602 typedef __fsblkcnt_t fsblkcnt_t;
603
604
605
606 typedef __fsfilcnt_t fsfilcnt_t;
607 # 270 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
608 # 1 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 1 3 4
609 # 21 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
610 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
611 # 22 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 2 3 4
612 # 60 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
613 typedef unsigned long int pthread_t;
614
615
616 union pthread_attr_t
617 {
618     char __size[56];
619     long int __align;
620 };
621
622 typedef union pthread_attr_t pthread_attr_t;
623
624
625
626
627
628 typedef struct __pthread_internal_list
629 {
630     struct __pthread_internal_list *__prev;
631     struct __pthread_internal_list *__next;
632 } __pthread_list_t;
633 # 90 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
634 typedef union
635 {
636     struct __pthread_mutex_s
637     {
638         int __lock;
639         unsigned int __count;
640         int __owner;
641
642         unsigned int __nusers;
643
644
645         int __kind;
646
647         short __spins;
648         short __elision;
649         __pthread_list_t __list;
650 # 124 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
651     } __data;
652     char __size[40];
653     long int __align;
654 } pthread_mutex_t;

```

```

656
657 typedef union
658 {
659     char __size[4];
660     int __align;
661 } pthread_mutexattr_t;
662
663
664
665
666 typedef union
667 {
668     struct
669     {
670         int __lock;
671         unsigned int __futex;
672         __extension__ unsigned long long int __total_seq;
673         __extension__ unsigned long long int __wakeup_seq;
674         __extension__ unsigned long long int __woken_seq;
675         void *__mutex;
676         unsigned int __nwaiters;
677         unsigned int __broadcast_seq;
678     } __data;
679     char __size[48];
680     __extension__ long long int __align;
681 } pthread_cond_t;
682
683
684
685
686
687 } pthread_condattr_t;
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704     struct
705     {
706         int __lock;
707         unsigned int __nr_readers;
708         unsigned int __readers_wakeup;
709         unsigned int __writer_wakeup;

```

```

710     unsigned int __nr_readers_queued;
711     unsigned int __nr_writers_queued;
712     int __writer;
713     int __shared;
714     unsigned long int __pad1;
715     unsigned long int __pad2;
716
717
718     unsigned int __flags;
719
720 } __data;
721 # 211 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
722     char __size[56];
723     long int __align;
724 } pthread_rwlock_t;
725
726 typedef union
727 {
728     char __size[8];
729     long int __align;
730 } pthread_rwlockattr_t;
731
732
733
734
735
736 typedef volatile int pthread_spinlock_t;
737
738
739
740
741 typedef union
742 {
743     char __size[32];
744     long int __align;
745 } pthread_barrier_t;
746
747 typedef union
748 {
749     char __size[4];
750     int __align;
751 } pthread_barrierattr_t;
752 # 271 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
753
754
755
756 # 315 "/usr/include/stdlib.h" 2 3 4
757
758
759
760
761
762
763 extern long int random (void) __attribute__((__nothrow__, __leaf__));

```

```

764
765
766 extern void srand (unsigned int __seed) __attribute__ ((__nothrow__ ,
767   __leaf__));
768
769
770
771
772 extern char *initstate (unsigned int __seed, char *__statebuf,
773   size_t __statelen) __attribute__ ((__nothrow__ , __leaf__))
774   __attribute__ ((__nonnull__ (2)));
775
776
777 extern char *setstate (char *__statebuf) __attribute__ ((__nothrow__ ,
778   __leaf__)) __attribute__ ((__nonnull__ (1)));
779
780
781
782
783
784
785 struct random_data
786 {
787   int32_t *fptr;
788   int32_t *rptr;
789   int32_t *state;
790   int rand_type;
791   int rand_deg;
792   int rand_sep;
793   int32_t *end_ptr;
794 };
795
796 extern int random_r (struct random_data *__restrict __buf,
797   int32_t *__restrict __result) __attribute__ ((__nothrow__ ,
798   __leaf__)) __attribute__ ((__nonnull__ (1, 2)));
799
800 extern int srand_r (unsigned int __seed, struct random_data *__buf)
801   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (2)));
802
803 extern int initstate_r (unsigned int __seed, char *__restrict __statebuf,
804   size_t __statelen,
805   struct random_data *__restrict __buf)
806   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (2, 4)));
807
808 extern int setstate_r (char *__restrict __statebuf,
809   struct random_data *__restrict __buf)
810   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1, 2)));

```

```

811
812
813
814
815
816 extern int rand (void) __attribute__((__nothrow__ , __leaf__));
817
818 extern void srand (unsigned int __seed) __attribute__((__nothrow__ ,
819   __leaf__));
820
821
822
823 extern int rand_r (unsigned int * __seed) __attribute__((__nothrow__ ,
824   __leaf__));
825
826
827
828
829
830
831 extern double drand48 (void) __attribute__((__nothrow__ , __leaf__));
832 extern double erand48 (unsigned short int __xsubi[3]) __attribute__((
833   __nothrow__ , __leaf__)) __attribute__((__nonnull__(1)));
834
835 extern long int lrand48 (void) __attribute__((__nothrow__ , __leaf__));
836 extern long int nrand48 (unsigned short int __xsubi[3])
837   __attribute__((__nothrow__ , __leaf__)) __attribute__((__nonnull__(
838     1)));
839
840
841 extern long int mrand48 (void) __attribute__((__nothrow__ , __leaf__));
842 extern long int jrand48 (unsigned short int __xsubi[3])
843   __attribute__((__nothrow__ , __leaf__)) __attribute__((__nonnull__(
844     1)));
845
846 extern void srand48 (long int __seedval) __attribute__((__nothrow__ ,
847   __leaf__));
848 extern unsigned short int *seed48 (unsigned short int __seed16v[3])
849   __attribute__((__nothrow__ , __leaf__)) __attribute__((__nonnull__(
850     1)));
851
852
853
854 struct drand48_data
855 {
856   unsigned short int __x[3];

```

```

857     unsigned short int __old_x[3];
858     unsigned short int __c;
859     unsigned short int __init;
860     __extension__ unsigned long long int __a;
861
862 };
863
864
865 extern int drand48_r (struct drand48_data *__restrict __buffer,
866     double *__restrict __result) __attribute__((__nothrow__, __leaf__))
867     __attribute__((__nonnull__(1, 2)));
868 extern int erand48_r (unsigned short int __xsubi[3],
869     struct drand48_data *__restrict __buffer,
870     double *__restrict __result) __attribute__((__nothrow__, __leaf__))
871     __attribute__((__nonnull__(1, 2)));
872
873 extern int lrand48_r (struct drand48_data *__restrict __buffer,
874     long int *__restrict __result)
875     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1, 2)));
876 extern int nrand48_r (unsigned short int __xsubi[3],
877     struct drand48_data *__restrict __buffer,
878     long int *__restrict __result)
879     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1, 2)));
880
881 extern int mrand48_r (struct drand48_data *__restrict __buffer,
882     long int *__restrict __result)
883     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1, 2)));
884 extern int jrand48_r (unsigned short int __xsubi[3],
885     struct drand48_data *__restrict __buffer,
886     long int *__restrict __result)
887     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1, 2)));
888
889 extern int srand48_r (long int __seedval, struct drand48_data *__buffer)
890     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(2)));
891
892 extern int seed48_r (unsigned short int __seed16v[3],
893     struct drand48_data *__buffer) __attribute__((__nothrow__, __leaf__))
894     __attribute__((__nonnull__(1, 2)));
895
896 extern int lcong48_r (unsigned short int __param[7],
897     struct drand48_data *__buffer)
898     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1, 2)));
899
900
901

```

```

902
903
904
905
906
907
908 extern void *malloc (size_t __size) __attribute__ ((__nothrow__ , __leaf__))
909   __attribute__ ((__malloc__));
910
911 extern void *calloc (size_t __nmemb, size_t __size)
912   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__malloc__));
913
914
915
916
917
918
919
920
921
922 extern void *realloc (void *__ptr, size_t __size)
923   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__warn_unused_result__));
924
925 extern void free (void *__ptr) __attribute__ ((__nothrow__ , __leaf__));
926
927
928
929
930 extern void cfree (void *__ptr) __attribute__ ((__nothrow__ , __leaf__));
931
932
933
934 # 1 "/usr/include/alloca.h" 1 3 4
935 # 24 "/usr/include/alloca.h" 3 4
936 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
937 # 25 "/usr/include/alloca.h" 2 3 4
938
939
940
941
942
943
944
945 extern void *alloca (size_t __size) __attribute__ ((__nothrow__ , __leaf__));
946
947
948
949
950
951

```

```

952 # 493 "/usr/include/stdlib.h" 2 3 4
953
954
955
956
957
958 extern void *valloc (size_t __size) __attribute__ ((__nothrow__, __leaf__))
959   __attribute__ ((__malloc__));
960
961
962
963 extern int posix_memalign (void **__memptr, size_t __alignment, size_t
964   __size)
965   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__(1)));
966
967
968
969 extern void *aligned_alloc (size_t __alignment, size_t __size)
970   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__malloc__))
971   __attribute__ ((__alloc_size__(2)));
972
973
974
975 extern void abort (void) __attribute__ ((__nothrow__, __leaf__))
976   __attribute__ ((__noreturn__));
977
978
979 extern int atexit (void (*__func) (void)) __attribute__ ((__nothrow__,
980   __leaf__)) __attribute__ ((__nonnull__(1)));
981
982
983
984
985
986
987 extern int at_quick_exit (void (*__func) (void)) __attribute__ (((
988   __nothrow__, __leaf__))) __attribute__ ((__nonnull__(1)));
989
990
991
992
993
994
995 extern int on_exit (void (*__func) (int __status, void *__arg), void *
996   __arg)
997   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__(1)));

```

```

997
998
999
1000
1001
1002
1003 extern void exit (int __status) __attribute__ ((__nothrow__ , __leaf__))
1004     __attribute__ ((__noreturn__));
1005
1006
1007
1008
1009 extern void quick_exit (int __status) __attribute__ ((__nothrow__ ,
1010     __leaf__)) __attribute__ ((__noreturn__));
1011
1012
1013
1014
1015
1016
1017 extern void _Exit (int __status) __attribute__ ((__nothrow__ , __leaf__))
1018     __attribute__ ((__noreturn__));
1019
1020
1021
1022
1023
1024 extern char *getenv (const char *__name) __attribute__ ((__nothrow__ ,
1025     __leaf__)) __attribute__ ((__nonnull__ (1)));
1026 # 578 "/usr/include/stdlib.h" 3 4
1027 extern int putenv (char *__string) __attribute__ ((__nothrow__ , __leaf__))
1028     __attribute__ ((__nonnull__ (1)));
1029
1030
1031
1032
1033 extern int setenv (const char *__name, const char *__value, int __replace)
1034     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (2)));
1035
1036
1037 extern int unsetenv (const char *__name) __attribute__ ((__nothrow__ ,
1038     __leaf__)) __attribute__ ((__nonnull__ (1)));
1039
1040
1041
1042
1043
```

```

1044 extern int clearenv (void) __attribute__ ((__nothrow__, __leaf__));
1045 # 606 "/usr/include/stdlib.h" 3 4
1046 extern char *mktemp (char *__template) __attribute__ ((__nothrow__,
1047   __leaf__)) __attribute__ ((__nonnull__(1)));
1047 # 620 "/usr/include/stdlib.h" 3 4
1048 extern int mkstemp (char *__template) __attribute__ ((__nonnull__(1))) ;
1049 # 642 "/usr/include/stdlib.h" 3 4
1050 extern int mkstemp (char *__template, int __suffixlen) __attribute__ (((
1051   __nonnull__(1))) ;
1051 # 663 "/usr/include/stdlib.h" 3 4
1052 extern char *mkdtemp (char *__template) __attribute__ ((__nothrow__,
1053   __leaf__)) __attribute__ ((__nonnull__(1))) ;
1053 # 712 "/usr/include/stdlib.h" 3 4
1054
1055
1056
1057
1058
1059 extern int system (const char *__command) ;
1060
1061 # 734 "/usr/include/stdlib.h" 3 4
1062 extern char *realpath (const char *__restrict __name,
1063   char *__restrict __resolved) __attribute__ ((__nothrow__,
1064   __leaf__)) ;
1065
1066
1067
1068
1069
1070 typedef int (*__compar_fn_t) (const void *, const void *);
1071 # 752 "/usr/include/stdlib.h" 3 4
1072
1073
1074
1075 extern void *bsearch (const void *__key, const void *__base,
1076   size_t __nmemb, size_t __size, __compar_fn_t __compar)
1077   __attribute__ ((__nonnull__(1, 2, 5))) ;
1078
1079
1080
1081
1082
1083
1084
1085 extern void qsort (void *__base, size_t __nmemb, size_t __size,
1086   __compar_fn_t __compar) __attribute__ ((__nonnull__(1, 4)));
1087 # 775 "/usr/include/stdlib.h" 3 4
1088 extern int abs (int __x) __attribute__ ((__nothrow__, __leaf__))
1089   __attribute__ ((__const__));
1089 extern long int labs (long int __x) __attribute__ ((__nothrow__, __leaf__))
1090   __attribute__ ((__const__));

```

```

1092
1093 --extension__ extern long long int llabs (long long int __x)
1094     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_const_))
1095     ;
1096
1097
1098
1099
1100
1101
1102 extern div_t div (int __numer, int __denom)
1103     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_const_))
1104     ;
1105 extern ldiv_t ldiv (long int __numer, long int __denom)
1106     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_const_))
1107     ;
1108
1109
1110 --extension__ extern lldiv_t lldiv (long long int __numer,
1111     long long int __denom)
1112     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_const_))
1113     ;
1114 # 812 "/usr/include/stdlib.h" 3 4
1115 extern char *ecvt (double __value, int __ndigit, int *_restrict __decpt,
1116     int *_restrict __sign) --attribute__ ((_nothrow_ , _leaf_))
1117     --attribute__ ((_nonnull_ (3, 4))) ;
1118
1119
1120
1121 extern char *fcvt (double __value, int __ndigit, int *_restrict __decpt,
1122     int *_restrict __sign) --attribute__ ((_nothrow_ , _leaf_))
1123     --attribute__ ((_nonnull_ (3, 4))) ;
1124
1125
1126
1127 extern char *gcvt (double __value, int __ndigit, char *_buf)
1128     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_nonnull_
1129     (3))) ;
1130
1131
1132
1133 extern char *qecvt (long double __value, int __ndigit,
1134     int *_restrict __decpt, int *_restrict __sign)
1135     --attribute__ ((_nothrow_ , _leaf_)) --attribute__ ((_nonnull_
1136     (3, 4))) ;
1137 extern char *qfcvt (long double __value, int __ndigit,
1138     int *_restrict __decpt, int *_restrict __sign)
```

```

1138     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
1139     (3, 4))) ;
1140 extern char *qgcvt (long double __value, int __ndigit, char *__buf)
1141     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
1142     (3))) ;
1143
1144
1145 extern int ecvt_r (double __value, int __ndigit, int *__restrict __decpt,
1146     int *__restrict __sign, char *__restrict __buf,
1147     size_t __len) __attribute__((__nothrow__, __leaf__)) __attribute__
1148     ((__nonnull__(3, 4, 5)));
1149 extern int fcvt_r (double __value, int __ndigit, int *__restrict __decpt,
1150     int *__restrict __sign, char *__restrict __buf,
1151     size_t __len) __attribute__((__nothrow__, __leaf__)) __attribute__
1152     ((__nonnull__(3, 4, 5)));
1153
1154 extern int qecvt_r (long double __value, int __ndigit,
1155     int *__restrict __decpt, int *__restrict __sign,
1156     char *__restrict __buf, size_t __len)
1157     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
1158     (3, 4, 5)));
1159 extern int qfcvt_r (long double __value, int __ndigit,
1160     int *__restrict __decpt, int *__restrict __sign,
1161     char *__restrict __buf, size_t __len)
1162     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
1163     (3, 4, 5)));
1164
1165
1166 extern int mblen (const char *__s, size_t __n) __attribute__((__nothrow__
1167     , __leaf__));
1168
1169 extern int mbtowc (wchar_t *__restrict __pwc,
1170     const char *__restrict __s, size_t __n) __attribute__((__nothrow__,
1171     __leaf__));
1172
1173 extern int wctomb (char *__s, wchar_t __wchar) __attribute__((__nothrow__
1174     , __leaf__));
1175
1176
1177 extern size_t mbstowcs (wchar_t *__restrict __pwcs,
1178     const char *__restrict __s, size_t __n) __attribute__((__nothrow__,
1179     __leaf__));
1180
1181 extern size_t wcstombs (char *__restrict __s,
1182     const wchar_t *__restrict __pwcs, size_t __n)

```

```

1182     __attribute__((__nothrow__, __leaf__));
1183
1184
1185
1186
1187
1188
1189
1190
1191 extern int rpmatch (const char *__response) __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1))) ;
1192 # 899 "/usr/include/stdlib.h" 3 4
1193 extern int getsubopt (char **__restrict __optionp,
1194                      char *const __restrict __tokens,
1195                      char **__restrict __valuep)
1196 __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1, 2, 3))) ;
1197 # 951 "/usr/include/stdlib.h" 3 4
1198 extern int getloadavg (double __loadavg[], int __nelem)
1199 __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__(1)));
1200
1201
1202 # 1 "/usr/include/x86_64-linux-gnu/bits/stdc-float.h" 1 3 4
1203 # 956 "/usr/include/stdlib.h" 2 3 4
1204 # 968 "/usr/include/stdlib.h" 3 4
1205
1206 # 4 "headers/all_headers.h" 2
1207 # 1 "/usr/include/stdio.h" 1 3 4
1208 # 29 "/usr/include/stdio.h" 3 4
1209
1210
1211
1212
1213 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
1214 # 34 "/usr/include/stdio.h" 2 3 4
1215 # 44 "/usr/include/stdio.h" 3 4
1216 struct _IO_FILE;
1217
1218
1219
1220 typedef struct _IO_FILE FILE;
1221
1222
1223
1224
1225
1226 # 64 "/usr/include/stdio.h" 3 4
1227 typedef struct _IO_FILE __FILE;
1228 # 74 "/usr/include/stdio.h" 3 4
1229 # 1 "/usr/include/libio.h" 1 3 4
1230 # 31 "/usr/include/libio.h" 3 4
1231 # 1 "/usr/include/_G_config.h" 1 3 4
1232 # 15 "/usr/include/_G_config.h" 3 4

```

```

1233 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
1234 # 16 "/usr/include/_G_config.h" 2 3 4
1235
1236
1237
1238
1239 # 1 "/usr/include/wchar.h" 1 3 4
1240 # 82 "/usr/include/wchar.h" 3 4
1241 typedef struct
1242 {
1243     int __count;
1244     union
1245     {
1246         unsigned int __wch;
1247
1248
1249
1250         char __wchb[4];
1251     } __value;
1252 } __mbstate_t;
1253 # 21 "/usr/include/_G_config.h" 2 3 4
1254 typedef struct
1255 {
1256     __off_t __pos;
1257     __mbstate_t __state;
1258 } __G_fpos_t;
1259 typedef struct
1260 {
1261     __off64_t __pos;
1262     __mbstate_t __state;
1263 } __G_fpos64_t;
1264 # 32 "/usr/include/libio.h" 2 3 4
1265 # 49 "/usr/include/libio.h" 3 4
1266 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stdarg.h" 1 3 4
1267 # 40 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stdarg.h" 3 4
1268 typedef __builtin_va_list __gnuc_va_list;
1269 # 50 "/usr/include/libio.h" 2 3 4
1270 # 144 "/usr/include/libio.h" 3 4
1271 struct _IO_jump_t; struct _IO_FILE;
1272 # 154 "/usr/include/libio.h" 3 4
1273 typedef void _IO_lock_t;
1274
1275
1276
1277
1278
1279
1280 struct _IO_marker {
1281     struct _IO_marker *_next;
1282     struct _IO_FILE *_sbuf;
1283
1284
1285     int _pos;

```

```

1287 # 177 "/usr/include/libio.h" 3 4
1288 };
1289
1290
1291 enum __codecvt_result
1292 {
1293     __codecvt_ok,
1294     __codecvt_partial,
1295     __codecvt_error,
1296     __codecvt_noconv
1297 };
1298 # 245 "/usr/include/libio.h" 3 4
1299 struct _IO_FILE {
1300     int _flags;
1301
1302
1303
1304
1305     char* _IO_read_ptr;
1306     char* _IO_read_end;
1307     char* _IO_read_base;
1308     char* _IO_write_base;
1309     char* _IO_write_ptr;
1310     char* _IO_write_end;
1311     char* _IO_buf_base;
1312     char* _IO_buf_end;
1313
1314     char *_IO_save_base;
1315     char *_IO_backup_base;
1316     char *_IO_save_end;
1317
1318     struct _IO_marker *_markers;
1319
1320     struct _IO_FILE *_chain;
1321
1322     int _fileno;
1323
1324
1325
1326     int _flags2;
1327
1328     __off_t _old_offset;
1329
1330
1331
1332     unsigned short _cur_column;
1333     signed char _vtable_offset;
1334     char _shortbuf[1];
1335
1336
1337
1338     _IO_lock_t *_lock;
1339 # 293 "/usr/include/libio.h" 3 4
1340     __off64_t _offset;

```

```

1341 # 302 "/usr/include/libio.h" 3 4
1342 void *_ __pad1;
1343 void *_ __pad2;
1344 void *_ __pad3;
1345 void *_ __pad4;
1346 size_t _ __pad5;
1347
1348 int _ mode;
1349
1350 char _ unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)
1351 ];
1352 };
1353
1354
1355 typedef struct _IO_FILE _IO_FILE;
1356
1357
1358 struct _IO_FILE_plus;
1359
1360 extern struct _IO_FILE_plus _IO_2_1_stdin_;
1361 extern struct _IO_FILE_plus _IO_2_1_stdout_;
1362 extern struct _IO_FILE_plus _IO_2_1_stderr_;
1363 # 338 "/usr/include/libio.h" 3 4
1364 typedef __ssize_t __io_read_fn (void *_ __cookie, char *_ __buf, size_t
1365 _ __nbytes);
1366
1367
1368
1369
1370
1371
1372 typedef __ssize_t __io_write_fn (void *_ __cookie, const char *_ __buf,
1373 size_t _ __n);
1374
1375
1376
1377
1378
1379
1380
1381 typedef int __io_seek_fn (void *_ __cookie, __off64_t *_ __pos, int _ __w);
1382
1383
1384 typedef int __io_close_fn (void *_ __cookie);
1385 # 390 "/usr/include/libio.h" 3 4
1386 extern int __underflow (_IO_FILE *);
1387 extern int __uflow (_IO_FILE *);
1388 extern int __overflow (_IO_FILE *, int);
1389 # 434 "/usr/include/libio.h" 3 4
1390 extern int _IO_getc (_IO_FILE *_ __fp);
1391 extern int _IO_putc (int _ __c, _IO_FILE *_ __fp);

```

```

1392 extern int _IO_feof (_IO_FILE *__fp) __attribute__ ((__nothrow__ ,
1393   __leaf__));
1394 extern int _IO_ferror (_IO_FILE *__fp) __attribute__ ((__nothrow__ ,
1395   __leaf__));
1396
1397
1398
1399
1400
1401 extern void _IO_flockfile (_IO_FILE *) __attribute__ ((__nothrow__ ,
1402   __leaf__));
1403 extern void _IO_funlockfile (_IO_FILE *) __attribute__ ((__nothrow__ ,
1404   __leaf__));
1405 extern int _IO_ftrylockfile (_IO_FILE *) __attribute__ ((__nothrow__ ,
1406   __leaf__));
1407 # 464 "/usr/include/libio.h" 3 4
1408 extern int _IO_vfscanf (_IO_FILE * __restrict, const char * __restrict,
1409   __gnuc_va_list, int * __restrict);
1410 extern int _IO_vfprintf (_IO_FILE * __restrict, const char * __restrict,
1411   __gnuc_va_list);
1412 extern __ssize_t _IO_padvn (_IO_FILE *, int, __ssize_t);
1413 extern size_t _IO_sgetn (_IO_FILE *, void *, size_t);
1414
1415 extern void _IO_free_backup_area (_IO_FILE *) __attribute__ ((__nothrow__ ,
1416   __leaf__));
1417 # 75 "/usr/include/stdio.h" 2 3 4
1418
1419
1420
1421 typedef __gnuc_va_list va_list;
1422 # 108 "/usr/include/stdio.h" 3 4
1423
1424
1425 typedef _G_fpos_t fpos_t;
1426
1427
1428
1429
1430 # 164 "/usr/include/stdio.h" 3 4
1431 # 1 "/usr/include/x86_64-linux-gnu/bits/stdio_lim.h" 1 3 4
1432 # 165 "/usr/include/stdio.h" 2 3 4
1433
1434
1435
1436 extern struct _IO_FILE *stdin;
1437 extern struct _IO_FILE *stdout;
1438 extern struct _IO_FILE *stderr;
1439
```

```

1440
1441
1442
1443
1444
1445
1446 extern int remove (const char *__filename) __attribute__((__nothrow__, __leaf__));
1447
1448 extern int rename (const char *__old, const char *__new) __attribute__((__nothrow__, __leaf__));
1449
1450
1451
1452
1453 extern int renameat (int __oldfd, const char *__old, int __newfd,
1454 const char *__new) __attribute__((__nothrow__, __leaf__));
1455
1456
1457
1458
1459
1460
1461
1462
1463 extern FILE *tmpfile (void) ;
1464 # 209 "/usr/include/stdio.h" 3 4
1465 extern char *tmpnam (char *__s) __attribute__((__nothrow__, __leaf__)) ;
1466
1467
1468
1469
1470
1471 extern char *tmpnam_r (char *__s) __attribute__((__nothrow__, __leaf__))
1472 ;
1473 # 227 "/usr/include/stdio.h" 3 4
1474 extern char *tempnam (const char *__dir, const char *__prefix)
1475 __attribute__((__nothrow__, __leaf__)) __attribute__((__malloc__));
1476
1477
1478
1479
1480
1481
1482
1483 extern int fclose (FILE *__stream);
1484
1485
1486
1487
1488 extern int fflush (FILE *__stream);
1489
```

```

1490 # 252 "/usr/include/stdio.h" 3 4
1491 extern int fflush_unlocked (FILE *__restrict __stream);
1492 # 266 "/usr/include/stdio.h" 3 4
1493
1494
1495
1496
1497
1498
1499 extern FILE *fopen (const char *__restrict __filename,
1500     const char *__restrict __modes) ;
1501
1502
1503
1504
1505 extern FILE *freopen (const char *__restrict __filename,
1506     const char *__restrict __modes,
1507     FILE *__restrict __stream) ;
1508 # 295 "/usr/include/stdio.h" 3 4
1509
1510 # 306 "/usr/include/stdio.h" 3 4
1511 extern FILE *fdopen (int __fd, const char *__modes) __attribute__((
1512     __nothrow__, __leaf__));
1513 # 319 "/usr/include/stdio.h" 3 4
1514 extern FILE *fmemopen (void *__s, size_t __len, const char *__modes)
1515     __attribute__((__nothrow__, __leaf__));
1516
1517
1518
1519 extern FILE *open_memstream (char **__bufloc, size_t __sizeloc)
1520     __attribute__((__nothrow__, __leaf__));
1521
1522
1523
1524
1525
1526 extern void setbuf (FILE *__restrict __stream, char *__restrict __buf)
1527     __attribute__((__nothrow__, __leaf__));
1528
1529
1530 extern int setvbuf (FILE *__restrict __stream, char *__restrict __buf,
1531     int __modes, size_t __n) __attribute__((__nothrow__, __leaf__));
1532
1533
1534
1535
1536 extern void setbuffer (FILE *__restrict __stream, char *__restrict __buf,
1537     size_t __size) __attribute__((__nothrow__, __leaf__));
1538
1539

```

```

1541 extern void setlinebuf (FILE *_stream) __attribute__ ((__nothrow__ ,
1542   __leaf__));
1543
1544
1545
1546
1547
1548
1549
1550 extern int fprintf (FILE *__restrict __stream ,
1551   const char * __restrict __format, ...);
1552
1553
1554
1555
1556 extern int printf (const char * __restrict __format, ...);
1557
1558 extern int sprintf (char * __restrict __s ,
1559   const char * __restrict __format, ...) __attribute__ ((__nothrow__));
1560
1561
1562
1563
1564
1565 extern int vfprintf (FILE * __restrict __s, const char * __restrict __format
1566   ,
1567   __gnuc_va_list __arg);
1568
1569
1570
1571 extern int vprintf (const char * __restrict __format, __gnuc_va_list __arg)
1572   ;
1573
1574 extern int vsprintf (char * __restrict __s, const char * __restrict __format
1575   ,
1576   __gnuc_va_list __arg) __attribute__ ((__nothrow__));
1577
1578
1579
1580 extern int snprintf (char * __restrict __s, size_t __ maxlen,
1581   const char * __restrict __format, ...)
1582   __attribute__ ((__nothrow__)) __attribute__ ((__format__(__printf__,
1583   3, 4)));
1584
1585 extern int vsnprintf (char * __restrict __s, size_t __ maxlen,
1586   const char * __restrict __format, __gnuc_va_list __arg)
1587   __attribute__ ((__nothrow__)) __attribute__ ((__format__(__printf__,
1588   3, 0)));
# 412 "/usr/include/stdio.h" 3 4

```

```

1589 extern int vdprintf (int __fd, const char * __restrict __fmt,
1590   __gnuc_va_list __arg)
1591   __attribute__ ((__format__ (__printf__, 2, 0)));
1592 extern int dprintf (int __fd, const char * __restrict __fmt, ...)
1593   __attribute__ ((__format__ (__printf__, 2, 3)));
1594
1595
1596
1597
1598
1599
1600
1601
1602 extern int fscanf (FILE * __restrict __stream,
1603   const char * __restrict __format, ...);
1604
1605
1606
1607
1608 extern int scanf (const char * __restrict __format, ...);
1609
1610 extern int sscanf (const char * __restrict __s,
1611   const char * __restrict __format, ...) __attribute__ ((__nothrow__,
1612   __leaf__));
1612 # 443 "/usr/include/stdio.h" 3 4
1613 extern int fscanf (FILE * __restrict __stream, const char * __restrict
1614   __format, ...) __asm__ ("\" __isoc99_fscanf")
1615
1616
1617 extern int scanf (const char * __restrict __format, ...) __asm__ (""
1618   __isoc99_scanf")
1619
1620
1621 extern int sscanf (const char * __restrict __s, const char * __restrict
1622   __format, ...) __asm__ ("\" __isoc99_sscanf") __attribute__ ((
1623   __nothrow__, __leaf__))
1624
1625
1626
1627
1628
1629
1630 extern int vfscanf (FILE * __restrict __s, const char * __restrict __format,
1631   __gnuc_va_list __arg)
1632   __attribute__ ((__format__ (__scanf__, 2, 0)));
1633
1634
1635
1636
1637

```

```

1638 extern int vscanf (const char *__restrict __format, __gnuc_va_list __arg)
1639     __attribute__ ((__format__ (__scanf__, 1, 0))) ;
1640
1641
1642 extern int vsscanf (const char *__restrict __s,
1643     const char *__restrict __format, __gnuc_va_list __arg)
1644     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__format__ (
1645     __scanf__, 2, 0)));
1645 # 494 "/usr/include/stdio.h" 3 4
1646 extern int vfscanf (FILE *__restrict __s, const char *__restrict __format,
1647     __gnuc_va_list __arg) __asm__ ("__isoc99_vfscanf")
1648
1649
1650     __attribute__ ((__format__ (__scanf__, 2, 0))) ;
1651 extern int vscanf (const char *__restrict __format, __gnuc_va_list __arg)
1652     __asm__ ("__isoc99_vscanf")
1653
1654     __attribute__ ((__format__ (__scanf__, 1, 0))) ;
1655 extern int vsscanf (const char *__restrict __s, const char *__restrict
1656     __format, __gnuc_va_list __arg) __asm__ ("__isoc99_vsscanf")
1657     __attribute__ ((__nothrow__, __leaf__))
1658
1659     __attribute__ ((__format__ (__scanf__, 2, 0)));
1660 # 522 "/usr/include/stdio.h" 3 4
1661
1662
1663
1664
1665
1666
1667
1668
1669 extern int fgetc (FILE *__stream);
1670 extern int getc (FILE *__stream);
1671
1672
1673
1674
1675
1676 extern int getchar (void);
1677
1678 # 550 "/usr/include/stdio.h" 3 4
1679 extern int getc_unlocked (FILE *__stream);
1680 extern int getchar_unlocked (void);
1681 # 561 "/usr/include/stdio.h" 3 4
1682 extern int fgetc_unlocked (FILE *__stream);
1683
1684
1685
1686

```

```

1687
1688
1689
1690
1691
1692
1693
1694 extern int fputc (int __c, FILE *__stream);
1695 extern int putc (int __c, FILE *__stream);
1696
1697
1698
1699
1700
1701 extern int putchar (int __c);
1702
1703 # 594 "/usr/include/stdio.h" 3 4
1704 extern int fputc_unlocked (int __c, FILE *__stream);
1705
1706
1707
1708
1709
1710
1711
1712 extern int putc_unlocked (int __c, FILE *__stream);
1713 extern int putchar_unlocked (int __c);
1714
1715
1716
1717
1718
1719
1720 extern int getw (FILE *__stream);
1721
1722
1723 extern int putw (int __w, FILE *__stream);
1724
1725
1726
1727
1728
1729
1730
1731
1732 extern char *fgets (char *__restrict __s, int __n, FILE *__restrict
1733   __stream)
1734   ;
1735 # 640 "/usr/include/stdio.h" 3 4
1736 # 665 "/usr/include/stdio.h" 3 4
1737 extern __ssize_t __getdelim (char **__restrict __lineptr,
1738   size_t *__restrict __n, int __delimiter,
1739   FILE *__restrict __stream) ;

```

```

1740 extern __ssize_t getdelim (char **__restrict __lineptr,
1741     size_t *__restrict __n, int __delimiter,
1742     FILE *__restrict __stream) ;
1743
1744
1745
1746
1747
1748
1749
1750 extern __ssize_t getline (char **__restrict __lineptr,
1751     size_t *__restrict __n,
1752     FILE *__restrict __stream) ;
1753
1754
1755
1756
1757
1758
1759
1760
1761 extern int fputs (const char *__restrict __s, FILE *__restrict __stream);
1762
1763
1764
1765
1766
1767 extern int puts (const char *__s);
1768
1769
1770
1771
1772
1773
1774 extern int ungetc (int __c, FILE *__stream);
1775
1776
1777
1778
1779
1780
1781 extern size_t fread (void *__restrict __ptr, size_t __size,
1782     size_t __n, FILE *__restrict __stream) ;
1783
1784
1785
1786
1787 extern size_t fwrite (const void *__restrict __ptr, size_t __size,
1788     size_t __n, FILE *__restrict __s);
1789
1790 # 737 "/usr/include/stdio.h" 3 4
1791 extern size_t fread_unlocked (void *__restrict __ptr, size_t __size,
1792     size_t __n, FILE *__restrict __stream) ;

```

```

1793 extern size_t fwrite_unlocked (const void * __restrict __ptr, size_t __size
1794   ,
1795   size_t __n, FILE * __restrict __stream);
1796
1797
1798
1799
1800
1801
1802
1803 extern int fseek (FILE * __stream, long int __off, int __whence);
1804
1805
1806
1807
1808 extern long int ftell (FILE * __stream) ;
1809
1810
1811
1812
1813 extern void rewind (FILE * __stream);
1814
1815 # 773 "/usr/include/stdio.h" 3 4
1816 extern int fseeko (FILE * __stream, __off_t __off, int __whence);
1817
1818
1819
1820
1821 extern __off_t ftello (FILE * __stream) ;
1822 # 792 "/usr/include/stdio.h" 3 4
1823
1824
1825
1826
1827
1828
1829 extern int fgetpos (FILE * __restrict __stream, fpos_t * __restrict __pos);
1830
1831
1832
1833
1834 extern int fsetpos (FILE * __stream, const fpos_t * __pos);
1835 # 815 "/usr/include/stdio.h" 3 4
1836
1837 # 824 "/usr/include/stdio.h" 3 4
1838
1839
1840 extern void clearerr (FILE * __stream) __attribute__ (( __nothrow__ ,
1841   __leaf__));
1842 extern int feof (FILE * __stream) __attribute__ (( __nothrow__ , __leaf__))
1843   ;

```

```

1844 extern int perror (FILE *_stream) __attribute__ ((__nothrow__ , __leaf__))
1845
1846
1847
1848
1849 extern void clearerr_unlocked (FILE *_stream) __attribute__ ((__nothrow__
1850     , __leaf__));
1850 extern int feof_unlocked (FILE *_stream) __attribute__ ((__nothrow__ ,
1851     __leaf__)) ;
1851 extern int perror_unlocked (FILE *_stream) __attribute__ ((__nothrow__ ,
1852     __leaf__)) ;
1853
1854
1855
1856
1857
1858
1859
1860 extern void perror (const char *_s);
1861
1862
1863
1864
1865
1866
1867 # 1 "/usr/include/x86_64-linux-gnu/bits/sys_errlist.h" 1 3 4
1868 # 26 "/usr/include/x86_64-linux-gnu/bits/sys_errlist.h" 3 4
1869 extern int sys_nerr;
1870 extern const char *const sys_errlist[];
1871 # 854 "/usr/include/stdio.h" 2 3 4
1872
1873
1874
1875
1876 extern int fileno (FILE *_stream) __attribute__ ((__nothrow__ , __leaf__))
1877     ;
1878
1879
1880
1881 extern int fileno_unlocked (FILE *_stream) __attribute__ ((__nothrow__ ,
1882     __leaf__));
1882 # 873 "/usr/include/stdio.h" 3 4
1883 extern FILE *popen (const char *_command, const char *_modes) ;
1884
1885
1886
1887
1888
1889 extern int pclose (FILE *_stream);
1890
1891
```

```

1892
1893
1894
1895 extern char *ctermid (char *_s) __attribute__ ((__nothrow__ , __leaf__));
1896 # 913 "/usr/include/stdio.h" 3 4
1897 extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__ ,
1898                           __leaf__));
1899
1900
1901 extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__ ,
1902                           __leaf__));
1903
1904 extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ ,
1905                           __leaf__));
1906 # 943 "/usr/include/stdio.h" 3 4
1907
1908 # 5 "headers/all_headers.h" 2
1909 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stdbool.h" 1 3 4
1910 # 6 "headers/all_headers.h" 2
1911 # 1 "/usr/include/math.h" 1 3 4
1912 # 28 "/usr/include/math.h" 3 4
1913
1914
1915
1916 # 1 "/usr/include/x86_64-linux-gnu/bits/huge_val.h" 1 3 4
1917 # 33 "/usr/include/math.h" 2 3 4
1918
1919 # 1 "/usr/include/x86_64-linux-gnu/bits/huge_valf.h" 1 3 4
1920 # 35 "/usr/include/math.h" 2 3 4
1921 # 1 "/usr/include/x86_64-linux-gnu/bits/huge_vall.h" 1 3 4
1922 # 36 "/usr/include/math.h" 2 3 4
1923
1924
1925 # 1 "/usr/include/x86_64-linux-gnu/bits/inf.h" 1 3 4
1926 # 39 "/usr/include/math.h" 2 3 4
1927
1928
1929 # 1 "/usr/include/x86_64-linux-gnu/bits/nan.h" 1 3 4
1930 # 42 "/usr/include/math.h" 2 3 4
1931
1932
1933
1934 # 1 "/usr/include/x86_64-linux-gnu/bits/mathdef.h" 1 3 4
1935 # 28 "/usr/include/x86_64-linux-gnu/bits/mathdef.h" 3 4
1936 typedef float float_t;
1937 typedef double double_t;
1938 # 46 "/usr/include/math.h" 2 3 4
1939 # 69 "/usr/include/math.h" 3 4
1940 # 1 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 1 3 4
1941 # 52 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
1942

```

```

1943
1944 extern double acos (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __acos (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1945
1946 extern double asin (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __asin (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1947
1948 extern double atan (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __atan (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1949
1950 extern double atan2 (double __y, double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
      extern double __atan2 (double __y, double __x)
      __attribute__ ((__nothrow__ , __leaf__));
1951
1952
1953 extern double cos (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __cos (double __x) __attribute__ ((__nothrow__ , __leaf__));
1954
1955 extern double sin (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __sin (double __x) __attribute__ ((__nothrow__ , __leaf__));
1956
1957 extern double tan (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __tan (double __x) __attribute__ ((__nothrow__ , __leaf__));
1958
1959
1960
1961
1962 extern double cosh (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __cosh (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1963
1964 extern double sinh (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __sinh (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1965
1966 extern double tanh (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __tanh (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1967
1968 # 86 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
1969
1970
1971 extern double acosh (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __acosh (double __x) __attribute__ ((__nothrow__ ,
      __leaf__));
1972
1973 extern double asinh (double __x) __attribute__ ((__nothrow__ , __leaf__));
      extern double __asinh (double __x) __attribute__ ((__nothrow__ ,

```

```

    __leaf__)) ;

1974
1975 extern double atanh (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __atanh (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
1976
1977
1978
1979
1980
1981
1982
1983 extern double exp (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __exp (double __x) __attribute__ ((__nothrow__ , __leaf__));
1984
1985
1986 extern double frexp (double __x, int *__exponent) __attribute__ ((
__nothrow__ , __leaf__));
     extern double __frexp (double __x, int *
__exponent) __attribute__ ((__nothrow__ , __leaf__));
1987
1988
1989 extern double ldexp (double __x, int __exponent) __attribute__ (((
__nothrow__ , __leaf__));
     extern double __ldexp (double __x, int
__exponent) __attribute__ ((__nothrow__ , __leaf__));
1990
1991
1992 extern double log (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __log (double __x) __attribute__ ((__nothrow__ , __leaf__));
1993
1994
1995 extern double log10 (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __log10 (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
1996
1997
1998 extern double modf (double __x, double *__iptr) __attribute__ (((
__nothrow__ , __leaf__));
     extern double __modf (double __x, double *
__iptr) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((
__nonnull__ (2)));
1999
2000 # 126 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2001
2002
2003 extern double expm1 (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __expm1 (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
2004
2005
2006 extern double log1p (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __log1p (double __x) __attribute__ ((__nothrow__ ,
__leaf__));
2007
```

```

2008
2009 extern double logb (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __logb (double __x) __attribute__ ((__nothrow__ ,
     __leaf__));
2010
2011
2012
2013
2014
2015
2016 extern double exp2 (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __exp2 (double __x) __attribute__ ((__nothrow__ ,
     __leaf__));
2017
2018
2019 extern double log2 (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __log2 (double __x) __attribute__ ((__nothrow__ ,
     __leaf__));
2020
2021
2022
2023
2024
2025
2026
2027
2028 extern double pow (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__));
     extern double __pow (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__));
2029
2030
2031 extern double sqrt (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __sqrt (double __x) __attribute__ ((__nothrow__ ,
     __leaf__));
2032
2033
2034
2035
2036
2037 extern double hypot (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__));
     extern double __hypot (double __x, double __y)
     __attribute__ ((__nothrow__ , __leaf__));
2038
2039
2040
2041
2042
2043
2044 extern double cbrt (double __x) __attribute__ ((__nothrow__ , __leaf__));
     extern double __cbrt (double __x) __attribute__ ((__nothrow__ ,
     __leaf__));
2045
2046
2047

```

```

2048
2049
2050
2051
2052
2053 extern double ceil (double __x) __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__const__));
    extern double __ceil (double __x)
    __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2054
2055
2056 extern double fabs (double __x) __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__const__));
    extern double __fabs (double __x)
    __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2057
2058
2059 extern double floor (double __x) __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__const__));
    extern double __floor (double __x)
    __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2060
2061
2062 extern double fmod (double __x, double __y) __attribute__ ((__nothrow__ ,
    __leaf__));
    extern double __fmod (double __x, double __y) __attribute__
((__nothrow__ , __leaf__));
2063
2064
2065
2066
2067 extern int __isinf (double __value) __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__const__));
2068
2069
2070 extern int __finite (double __value) __attribute__ ((__nothrow__ ,
    __leaf__)) __attribute__ ((__const__));
2071
2072
2073
2074
2075
2076 extern int isinf (double __value) __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__const__));
2077
2078
2079 extern int finite (double __value) __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__const__));
2080
2081
2082 extern double drem (double __x, double __y) __attribute__ ((__nothrow__ ,
    __leaf__));
    extern double __drem (double __x, double __y) __attribute__
((__nothrow__ , __leaf__));
2083
2084
2085
2086 extern double significand (double __x) __attribute__ ((__nothrow__ ,
    __leaf__));
    extern double __significand (double __x) __attribute__ ((
```

```

    __nothrow__ , __leaf__));
2087
2088
2089
2090
2091
2092 extern double copysign (double __x, double __y) __attribute__((
    __nothrow__ , __leaf__)) __attribute__((__const__)); extern double
    __copysign (double __x, double __y) __attribute__((__nothrow__ ,
    __leaf__)) __attribute__((__const__));
2093
2094
2095
2096
2097
2098
2099 extern double nan (const char *__tagb) __attribute__((__nothrow__ ,
    __leaf__)) __attribute__((__const__)); extern double __nan (const char
    *__tagb) __attribute__((__nothrow__ , __leaf__)) __attribute__((
    __const__));
2100
2101
2102
2103
2104
2105 extern int __isnan (double __value) __attribute__((__nothrow__ , __leaf__))
    __attribute__((__const__));
2106
2107
2108
2109 extern int isnan (double __value) __attribute__((__nothrow__ , __leaf__))
    __attribute__((__const__));
2110
2111
2112 extern double j0 (double) __attribute__((__nothrow__ , __leaf__)); extern
    double __j0 (double) __attribute__((__nothrow__ , __leaf__));
2113 extern double j1 (double) __attribute__((__nothrow__ , __leaf__)); extern
    double __j1 (double) __attribute__((__nothrow__ , __leaf__));
2114 extern double jn (int, double) __attribute__((__nothrow__ , __leaf__));
    extern double __jn (int, double) __attribute__((__nothrow__ , __leaf__));
2115 extern double y0 (double) __attribute__((__nothrow__ , __leaf__)); extern
    double __y0 (double) __attribute__((__nothrow__ , __leaf__));
2116 extern double y1 (double) __attribute__((__nothrow__ , __leaf__)); extern
    double __y1 (double) __attribute__((__nothrow__ , __leaf__));
2117 extern double yn (int, double) __attribute__((__nothrow__ , __leaf__));
    extern double __yn (int, double) __attribute__((__nothrow__ , __leaf__));
2118
2119
2120
2121
2122
2123
```

```

2124 extern double erf (double) __attribute__((__nothrow__ , __leaf__));
2125     extern double __erf (double) __attribute__((__nothrow__ , __leaf__));
2126 extern double erfc (double) __attribute__((__nothrow__ , __leaf__));
2127     extern double __erfc (double) __attribute__((__nothrow__ , __leaf__));
2128 extern double lgamma (double) __attribute__((__nothrow__ , __leaf__));
2129     extern double __lgamma (double) __attribute__((__nothrow__ , __leaf__))
2130     ;
2131
2132
2133 extern double tgamma (double) __attribute__((__nothrow__ , __leaf__));
2134     extern double __tgamma (double) __attribute__((__nothrow__ , __leaf__))
2135     ;
2136
2137
2138
2139 extern double gamma (double) __attribute__((__nothrow__ , __leaf__));
2140     extern double __gamma (double) __attribute__((__nothrow__ , __leaf__))
2141     ;
2142
2143
2144
2145
2146 extern double lgamma_r (double, int *__signgamp) __attribute__((
2147     __nothrow__ , __leaf__)); extern double __lgamma_r (double, int *
2148     __signgamp) __attribute__((__nothrow__ , __leaf__));
2149
2150
2151
2152
2153
2154 extern double rint (double __x) __attribute__((__nothrow__ , __leaf__));
2155     extern double __rint (double __x) __attribute__((__nothrow__ ,
2156     __leaf__));
2157
2158
2159 extern double nextafter (double __x, double __y) __attribute__((
2160     __nothrow__ , __leaf__)) __attribute__((__const__)); extern double
2161     __nextafter (double __x, double __y) __attribute__((__nothrow__ ,
2162     __leaf__)) __attribute__((__const__));
2163
2164
2165 extern double nexttoward (double __x, long double __y) __attribute__((
2166     __nothrow__ , __leaf__)) __attribute__((__const__)); extern double
2167     __nexttoward (double __x, long double __y) __attribute__((__nothrow__
2168     , __leaf__)) __attribute__((__const__));

```

```

2160
2161
2162
2163 extern double remainder (double __x, double __y) __attribute__ ((  

2164     __nothrow__, __leaf__)); extern double __remainder (double __x, double  

2165     __y) __attribute__ ((__nothrow__, __leaf__));
2166
2167 extern double scalbn (double __x, int __n) __attribute__ ((__nothrow__,  

2168     __leaf__)); extern double __scalbn (double __x, int __n) __attribute__  

2169     ((__nothrow__, __leaf__));
2170
2171 extern int ilogb (double __x) __attribute__ ((__nothrow__, __leaf__));
2172     extern int __ilogb (double __x) __attribute__ ((__nothrow__, __leaf__));
2173
2174
2175
2176 extern double scalbln (double __x, long int __n) __attribute__ ((  

2177     __nothrow__, __leaf__)); extern double __scalbln (double __x, long int  

2178     __n) __attribute__ ((__nothrow__, __leaf__));
2179
2180
2181
2182
2183
2184 extern double round (double __x) __attribute__ ((__nothrow__, __leaf__))  

2185     __attribute__ ((__const__)); extern double __round (double __x)  

2186     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__const__));
2187
2188
2189
2190
2191
2192
2193 extern double remquo (double __x, double __y, int *__quo) __attribute__ ((  

2194     __nothrow__, __leaf__)); extern double __remquo (double __x, double  

2195     __y, int *__quo) __attribute__ ((__nothrow__, __leaf__));
2196
2197

```

```

2198
2199
2200 extern long int lint (double __x) __attribute__ ((__nothrow__ , __leaf__))
     ; extern long int __lint (double __x) __attribute__ ((__nothrow__ ,
     __leaf__));
2201 __extension__
2202 extern long long int llint (double __x) __attribute__ ((__nothrow__ ,
     __leaf__)); extern long long int __llint (double __x) __attribute__ ((
     __nothrow__ , __leaf__));
2203
2204
2205
2206 extern long int lround (double __x) __attribute__ ((__nothrow__ , __leaf__))
     ; extern long int __lround (double __x) __attribute__ ((__nothrow__ ,
     __leaf__));
2207 __extension__
2208 extern long long int llround (double __x) __attribute__ ((__nothrow__ ,
     __leaf__)); extern long long int __llround (double __x) __attribute__ ((
     __nothrow__ , __leaf__));
2209
2210
2211
2212 extern double fdim (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__));
     extern double __fdim (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__));
2213
2214
2215 extern double fmax (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__)) __attribute__ ((__const__));
     extern double __fmax (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__)) __attribute__ ((__const__));
2216
2217
2218 extern double fmin (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__)) __attribute__ ((__const__));
     extern double __fmin (double __x, double __y) __attribute__ ((__nothrow__ ,
     __leaf__)) __attribute__ ((__const__));
2219
2220
2221
2222 extern int __fpclassify (double __value) __attribute__ ((__nothrow__ ,
     __leaf__))
     __attribute__ ((__const__));
2223
2224
2225
2226 extern int __signbit (double __value) __attribute__ ((__nothrow__ ,
     __leaf__))
     __attribute__ ((__const__));
2227
2228
2229
2230
2231 extern double fma (double __x, double __y, double __z) __attribute__ ((
     __nothrow__ , __leaf__));
     extern double __fma (double __x, double __y,
     double __z) __attribute__ ((__nothrow__ , __leaf__));

```

```

2232
2233
2234
2235
2236 # 371 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2237 extern double scalb (double __x, double __n) __attribute__ ((__nothrow__ ,
2238   __leaf__));
2239   extern double __scalb (double __x, double __n)
2240   __attribute__ ((__nothrow__ , __leaf__));
2241 # 70 "/usr/include/math.h" 2 3 4
2242 # 88 "/usr/include/math.h" 3 4
2243 # 1 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 1 3 4
2244 # 52 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265

```

```

2266 extern float tanhf (float __x) __attribute__ ((__nothrow__ , __leaf__));
    extern float __tanhf (float __x) __attribute__ ((__nothrow__ , __leaf__));
);
2267
2268 # 86 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2269
2270
2271 extern float acoshf (float __x) __attribute__ ((__nothrow__ , __leaf__));
    extern float __acoshf (float __x) __attribute__ ((__nothrow__ ,
__leaf__));
;
2272
2273 extern float asinhf (float __x) __attribute__ ((__nothrow__ , __leaf__));
    extern float __asinhf (float __x) __attribute__ ((__nothrow__ ,
__leaf__));
;
2274
2275 extern float atanhf (float __x) __attribute__ ((__nothrow__ , __leaf__));
    extern float __atanhf (float __x) __attribute__ ((__nothrow__ ,
__leaf__));
;
2276
2277
2278
2279
2280
2281
2282
2283 extern float expf (float __x) __attribute__ ((__nothrow__ , __leaf__));
    extern float __expf (float __x) __attribute__ ((__nothrow__ , __leaf__));
);
2284
2285
2286 extern float frexpf (float __x, int *__exponent) __attribute__ (((
__nothrow__ , __leaf__)); extern float __frexpf (float __x, int *
__exponent) __attribute__ ((__nothrow__ , __leaf__));
;
2287
2288
2289 extern float ldexpf (float __x, int __exponent) __attribute__ (((
__nothrow__ , __leaf__)); extern float __ldexpf (float __x, int
__exponent) __attribute__ ((__nothrow__ , __leaf__));
;
2290
2291
2292 extern float logf (float __x) __attribute__ ((__nothrow__ , __leaf__));
    extern float __logf (float __x) __attribute__ ((__nothrow__ , __leaf__));
);
2293
2294
2295 extern float log10f (float __x) __attribute__ ((__nothrow__ , __leaf__));
    extern float __log10f (float __x) __attribute__ ((__nothrow__ ,
__leaf__));
;
2296
2297
2298 extern float modff (float __x, float *__iptr) __attribute__ ((__nothrow__
, __leaf__)); extern float __modff (float __x, float *__iptr)
__attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__(
2)));

```

```

2299
2300 # 126 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2301
2302
2303 extern float expm1f (float __x) __attribute__ ((__nothrow__, __leaf__));
2304     extern float __expm1f (float __x) __attribute__ ((__nothrow__, __leaf__));
2305
2306 extern float log1pf (float __x) __attribute__ ((__nothrow__, __leaf__));
2307     extern float __log1pf (float __x) __attribute__ ((__nothrow__, __leaf__));
2308
2309 extern float logbf (float __x) __attribute__ ((__nothrow__, __leaf__));
2310     extern float __logbf (float __x) __attribute__ ((__nothrow__, __leaf__));
2311
2312
2313
2314
2315
2316 extern float exp2f (float __x) __attribute__ ((__nothrow__, __leaf__));
2317     extern float __exp2f (float __x) __attribute__ ((__nothrow__, __leaf__));
2318
2319 extern float log2f (float __x) __attribute__ ((__nothrow__, __leaf__));
2320     extern float __log2f (float __x) __attribute__ ((__nothrow__, __leaf__));
2321
2322
2323
2324
2325
2326
2327
2328 extern float powf (float __x, float __y) __attribute__ ((__nothrow__, __leaf__));
2329     extern float __powf (float __x, float __y) __attribute__ ((__nothrow__, __leaf__));
2330
2331 extern float sqrtf (float __x) __attribute__ ((__nothrow__, __leaf__));
2332     extern float __sqrtf (float __x) __attribute__ ((__nothrow__, __leaf__));
2333
2334
2335
2336
2337 extern float hypotf (float __x, float __y) __attribute__ ((__nothrow__, __leaf__));
2338     extern float __hypotf (float __x, float __y) __attribute__ ((__nothrow__, __leaf__));

```

```

((__nothrow__ , __leaf__));
2338
2339
2340
2341
2342
2343
2344 extern float cbrtf (float __x) __attribute__((__nothrow__ , __leaf__));
2345     extern float __cbrtf (float __x) __attribute__((__nothrow__ , __leaf__));
2346         );
2347
2348
2349
2350
2351
2352
2353 extern float ceilf (float __x) __attribute__((__nothrow__ , __leaf__))
2354     __attribute__((__const__));
2355     extern float __ceilf (float __x)
2356         __attribute__((__nothrow__ , __leaf__)) __attribute__((__const__));
2357
2358
2359 extern float fabsf (float __x) __attribute__((__nothrow__ , __leaf__))
2360     __attribute__((__const__));
2361     extern float __fabsf (float __x)
2362         __attribute__((__nothrow__ , __leaf__)) __attribute__((__const__));
2363
2364
2365
2366
2367 extern int __isinff (float __value) __attribute__((__nothrow__ , __leaf__))
2368     __attribute__((__const__));
2369
2370 extern int __finitef (float __value) __attribute__((__nothrow__ ,
2371     __leaf__)) __attribute__((__const__));
2372
2373
2374
2375
2376 extern int isinff (float __value) __attribute__((__nothrow__ , __leaf__))
2377     __attribute__((__const__));

```

```

2378
2379 extern int finitef (float __value) __attribute__ ((__nothrow__ , __leaf__))
2380   __attribute__ ((__const__));
2381
2382 extern float dremf (float __x, float __y) __attribute__ ((__nothrow__ ,
2383   __leaf__));
extern float __dremf (float __x, float __y) __attribute__
2384   ((__nothrow__ , __leaf__));
2385
2386 extern float significandf (float __x) __attribute__ ((__nothrow__ ,
2387   __leaf__));
extern float __significandf (float __x) __attribute__ (((
2388   __nothrow__ , __leaf__));
2389
2390
2391
2392 extern float copysignf (float __x, float __y) __attribute__ ((__nothrow__
2393   , __leaf__)) __attribute__ ((__const__));
extern float __copysignf (
2394   float __x, float __y) __attribute__ ((__nothrow__ , __leaf__))
2395   __attribute__ ((__const__));
2396
2397
2398
2399 extern float nanf (const char *__tagb) __attribute__ ((__nothrow__ ,
2400   __leaf__)) __attribute__ ((__const__));
extern float __nanf (const char
2401   *__tagb) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ (((
2402   __const__));
2403
2404
2405 extern int __isnanf (float __value) __attribute__ ((__nothrow__ , __leaf__))
2406   __attribute__ ((__const__));
2407
2408
2409 extern int isnanf (float __value) __attribute__ ((__nothrow__ , __leaf__))
2410   __attribute__ ((__const__));
2411
2412 extern float j0f (float) __attribute__ ((__nothrow__ , __leaf__));
extern
2413   float __j0f (float) __attribute__ ((__nothrow__ , __leaf__));
2414 extern float j1f (float) __attribute__ ((__nothrow__ , __leaf__));
extern
2415   float __j1f (float) __attribute__ ((__nothrow__ , __leaf__));
2416 extern float jnf (int, float) __attribute__ ((__nothrow__ , __leaf__));
extern
2417   float __jnf (int, float) __attribute__ ((__nothrow__ , __leaf__));
2418

```

```

2415 extern float y0f (float) __attribute__((__nothrow__, __leaf__));
2416     float __y0f (float) __attribute__((__nothrow__, __leaf__));
2417 extern float y1f (float) __attribute__((__nothrow__, __leaf__));
2418     float __y1f (float) __attribute__((__nothrow__, __leaf__));
2419 extern float ynf (int, float) __attribute__((__nothrow__, __leaf__));
2420     extern float __ynf (int, float) __attribute__((__nothrow__, __leaf__));
2421 );
2422
2423
2424 extern float erff (float) __attribute__((__nothrow__, __leaf__));
2425     float __erff (float) __attribute__((__nothrow__, __leaf__));
2426 extern float erfcf (float) __attribute__((__nothrow__, __leaf__));
2427     extern float __erfcf (float) __attribute__((__nothrow__, __leaf__));
2428 extern float lgammaf (float) __attribute__((__nothrow__, __leaf__));
2429     extern float __lgammaf (float) __attribute__((__nothrow__, __leaf__));
2430 ;
2431
2432
2433 extern float tgammaf (float) __attribute__((__nothrow__, __leaf__));
2434     extern float __tgammaf (float) __attribute__((__nothrow__, __leaf__));
2435 ;
2436
2437
2438
2439 extern float gammaf (float) __attribute__((__nothrow__, __leaf__));
2440     extern float __gammaf (float) __attribute__((__nothrow__, __leaf__));
2441
2442
2443
2444
2445
2446 extern float lgammaf_r (float, int *__signgamp) __attribute__((
2447     __nothrow__, __leaf__));
2448     extern float __lgammaf_r (float, int *
2449     __signgamp) __attribute__((__nothrow__, __leaf__));
2450
2451
2452
2453
2454 extern float rintf (float __x) __attribute__((__nothrow__, __leaf__));
2455     extern float __rintf (float __x) __attribute__((__nothrow__, __leaf__));

```

```

));
2455
2456
2457 extern float nextafterf (float __x, float __y) __attribute__ ((__nothrow__,
2458   __leaf__)) __attribute__ ((__const__)); extern float __nextafterf (
2459   float __x, float __y) __attribute__ ((__nothrow__, __leaf__))
2460   __attribute__ ((__const__));
2461
2462
2463 extern float nexttowardf (float __x, long double __y) __attribute__ (((
2464   __nothrow__, __leaf__)) __attribute__ ((__const__)); extern float
2465   __nexttowardf (float __x, long double __y) __attribute__ ((__nothrow__,
2466   __leaf__)) __attribute__ ((__const__));
2467
2468
2469
2470
2471 extern int ilogbf (float __x) __attribute__ ((__nothrow__, __leaf__));
2472   extern int __ilogbf (float __x) __attribute__ ((__nothrow__, __leaf__));
2473
2474
2475
2476 extern float scalbnf (float __x, int __n) __attribute__ ((__nothrow__,
2477   __leaf__)); extern float __scalbnf (float __x, int __n) __attribute__
2478   ((__nothrow__, __leaf__));
2479
2480
2481
2482
2483
2484 extern float roundf (float __x) __attribute__ ((__nothrow__, __leaf__))
2485   __attribute__ ((__const__)); extern float __roundf (float __x)
2486   __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__const__));
2487
2488
2489
2490 extern float truncf (float __x) __attribute__ ((__nothrow__, __leaf__))
2491   __attribute__ ((__const__)); extern float __truncf (float __x)

```

```

2489
2490
2491
2492
2493 extern float remquof (float __x, float __y, int *__quo) __attribute__ ((  

2494     __nothrow__, __leaf__)); extern float __remquof (float __x, float __y,  

2495     int *__quo) __attribute__ ((__nothrow__, __leaf__));
2496
2497
2498
2499
2500 extern long int lrintf (float __x) __attribute__ ((__nothrow__, __leaf__))  

2501     ; extern long int __lrintf (float __x) __attribute__ ((__nothrow__,  

2502     __leaf__));
2503 --extension--
2504 extern long long int llrintf (float __x) __attribute__ ((__nothrow__,  

2505     __leaf__)); extern long long int __llrintf (float __x) __attribute__ ((  

2506     __nothrow__, __leaf__));
2507
2508
2509
2510
2511
2512 extern float fdimf (float __x, float __y) __attribute__ ((__nothrow__,  

2513     __leaf__)); extern float __fdimf (float __x, float __y) __attribute__  

2514     ((__nothrow__, __leaf__));
2515
2516
2517
2518 extern float fmaxf (float __x, float __y) __attribute__ ((__nothrow__,  

2519     __leaf__)) __attribute__ ((__const__)); extern float __fmaxf (float __x  

2520     , float __y) __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((  

2521     __const__));
2522
2523
2524
2525 extern float fminf (float __x, float __y) __attribute__ ((__nothrow__,  

2526     __leaf__)) __attribute__ ((__const__)); extern float __fminf (float __x  

2527     , float __y) __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((  

2528     __const__));
2529
2530
2531
2532 extern int __fpclassifyf (float __value) __attribute__ ((__nothrow__,  

2533     __leaf__));

```

```

2523     __attribute__((__const__));
2524
2525
2526 extern int __signbitf (float __value) __attribute__((__nothrow__ ,
2527     __leaf__))
2528     __attribute__((__const__));
2529
2530
2531 extern float fmaf (float __x, float __y, float __z) __attribute__((
2532     __nothrow__ , __leaf__));
extern float __fmaf (float __x, float __y,
2533     float __z) __attribute__((__nothrow__ , __leaf__));
2534
2535
2536 # 371 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2537 extern float scalbf (float __x, float __n) __attribute__((__nothrow__ ,
2538     __leaf__));
extern float __scalbf (float __x, float __n) __attribute__((
2539     __nothrow__ , __leaf__));
# 89 "/usr/include/math.h" 2 3 4
# 132 "/usr/include/math.h" 3 4
2540 # 1 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 1 3 4
2541 # 52 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2542
2543
2544 extern long double acosl (long double __x) __attribute__((__nothrow__ ,
2545     __leaf__));
extern long double __acosl (long double __x) __attribute__((
2546     __nothrow__ , __leaf__));
2547
2548 extern long double asinl (long double __x) __attribute__((__nothrow__ ,
2549     __leaf__));
extern long double __asinl (long double __x) __attribute__((
2550     __nothrow__ , __leaf__));
2551
2552
2553 extern long double atanl (long double __x) __attribute__((__nothrow__ ,
2554     __leaf__));
extern long double __atanl (long double __x) __attribute__((
2555     __nothrow__ , __leaf__));
2556
2557 extern long double atan2l (long double __y, long double __x) __attribute__((
2558     __nothrow__ , __leaf__));
extern long double __atan2l (long double __y, long double __x) __attribute__((
2559     __nothrow__ , __leaf__));
2560
2561
2562 extern long double cosl (long double __x) __attribute__((__nothrow__ ,
2563     __leaf__));
extern long double __cosl (long double __x) __attribute__((
2564     __nothrow__ , __leaf__));
2565
2566
2567 extern long double sinl (long double __x) __attribute__((__nothrow__ ,
2568     __leaf__));
extern long double __sinl (long double __x) __attribute__((
2569     __nothrow__ , __leaf__));
2570
2571
2572 extern long double tanl (long double __x) __attribute__((__nothrow__ ,
2573     __leaf__));
extern long double __tanl (long double __x) __attribute__((
2574     __nothrow__ , __leaf__));

```

```

2558
2559
2560
2561
2562 extern long double coshl (long double __x) __attribute__ ((__nothrow__ ,
2563   __leaf__));
extern long double __coshl (long double __x) __attribute__ ((__nothrow__ ,
2564   __leaf__));
2565
2566 extern long double sinhl (long double __x) __attribute__ ((__nothrow__ ,
2567   __leaf__));
extern long double __sinhl (long double __x) __attribute__ ((__nothrow__ ,
2568   __leaf__));
2569
2570
2571 extern long double tanhl (long double __x) __attribute__ ((__nothrow__ ,
2572   __leaf__));
extern long double __tanh (long double __x) __attribute__ ((__nothrow__ ,
2573   __leaf__));
2574
2575 # 86 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2576
2577
2578
2579
2580
2581
2582
2583 extern long double acoshl (long double __x) __attribute__ ((__nothrow__ ,
2584   __leaf__));
extern long double __acoshl (long double __x) __attribute__ ((__nothrow__ ,
2585   __leaf__));
2586
2587 extern long double asinhl (long double __x) __attribute__ ((__nothrow__ ,
2588   __leaf__));
extern long double __asinhl (long double __x) __attribute__ ((__nothrow__ ,
2589   __leaf__));
2590
2591 extern long double atanh (long double __x) __attribute__ ((__nothrow__ ,
2592   __leaf__));
extern long double __atanh (long double __x) __attribute__ ((__nothrow__ ,
2593   __leaf__));
2594
2595
2596 extern long double expl (long double __x) __attribute__ ((__nothrow__ ,
2597   __leaf__));
extern long double __expl (long double __x) __attribute__ ((__nothrow__ ,
2598   __leaf__));
2599
2600
2601 extern long double frexp (long double __x, int *__exponent) __attribute__ ((__nothrow__ ,
2602   __leaf__));
extern long double __frexp (long double __x, int *__exponent) __attribute__ ((__nothrow__ ,
2603   __leaf__));
2604
2605
2606 extern long double ldexp (long double __x, int __exponent) __attribute__ ((__nothrow__ ,
2607   __leaf__));
extern long double __ldexp (long double __x, int __exponent) __attribute__ ((__nothrow__ ,
2608   __leaf__));
2609
2610
2611 extern long double logl (long double __x) __attribute__ ((__nothrow__ ,
2612   __leaf__));
extern long double __logl (long double __x) __attribute__ ((__nothrow__ ,
2613   __leaf__));

```

```

    ((__nothrow__ , __leaf__));
2593
2594
2595 extern long double log10l (long double __x) __attribute__ ((__nothrow__ ,
2596     __leaf__)); extern long double __log10l (long double __x) __attribute__ 
2597     ((__nothrow__ , __leaf__));
2598
2599 extern long double modfl (long double __x, long double *__iptr)
2600     __attribute__ ((__nothrow__ , __leaf__)); extern long double __modfl (
2601     long double __x, long double *__iptr) __attribute__ ((__nothrow__ ,
2602     __leaf__)) __attribute__ ((__nonnull__ (2)));
2603
2604 # 126 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2605
2606
2607 extern long double expm1l (long double __x) __attribute__ ((__nothrow__ ,
2608     __leaf__)); extern long double __expm1l (long double __x) __attribute__ 
2609     ((__nothrow__ , __leaf__));
2610
2611
2612 extern long double log1pl (long double __x) __attribute__ ((__nothrow__ ,
2613     __leaf__)); extern long double __log1pl (long double __x) __attribute__ 
2614     ((__nothrow__ , __leaf__));
2615
2616
2617 extern long double logbl (long double __x) __attribute__ ((__nothrow__ ,
2618     __leaf__)); extern long double __logbl (long double __x) __attribute__ 
2619     ((__nothrow__ , __leaf__));
2620
2621
2622 extern long double exp2l (long double __x) __attribute__ ((__nothrow__ ,
2623     __leaf__)); extern long double __exp2l (long double __x) __attribute__ 
2624     ((__nothrow__ , __leaf__));
2625
2626
2627 extern long double log2l (long double __x) __attribute__ ((__nothrow__ ,
2628     __leaf__)); extern long double __log2l (long double __x) __attribute__ 
2629     ((__nothrow__ , __leaf__));
2630
2631
2632 extern long double powl (long double __x, long double __y) __attribute__ 
2633     ((__nothrow__ , __leaf__)); extern long double __powl (long double __x,
2634     long double __y) __attribute__ ((__nothrow__ , __leaf__));

```

```

2629
2630
2631 extern long double sqrtl (long double __x) __attribute__ ((__nothrow__ ,
2632   __leaf__));
2633 extern long double __sqrtl (long double __x) __attribute__ ((__nothrow__ ,
2634   __leaf__));
2635
2636
2637 extern long double hypotl (long double __x, long double __y) __attribute__ (
2638   (__nothrow__ , __leaf__));
2639 extern long double __hypotl (long double __x, long double __y) __attribute__ (
2640   (__nothrow__ , __leaf__));
2641
2642
2643
2644 extern long double cbrtl (long double __x) __attribute__ ((__nothrow__ ,
2645   __leaf__));
2646 extern long double __cbrtl (long double __x) __attribute__ ((__nothrow__ ,
2647   __leaf__));
2648
2649
2650
2651
2652
2653 extern long double ceil (long double __x) __attribute__ ((__nothrow__ ,
2654   __leaf__)) __attribute__ ((__const__));
2655 extern long double __ceil (long double __x) __attribute__ ((__nothrow__ ,
2656   __leaf__)) __attribute__ ((__const__));
2657
2658
2659 extern long double fabsl (long double __x) __attribute__ ((__nothrow__ ,
2660   __leaf__)) __attribute__ ((__const__));
2661 extern long double __fabsl (long double __x) __attribute__ ((__nothrow__ ,
2662   __leaf__)) __attribute__ ((__const__));
2663
2664
2665 extern long double floorl (long double __x) __attribute__ ((__nothrow__ ,
2666   __leaf__)) __attribute__ ((__const__));
2667 extern long double __floorl (long double __x) __attribute__ ((__nothrow__ ,
2668   __leaf__)) __attribute__ ((__const__));
2669
2670
2671 extern long double fmodl (long double __x, long double __y) __attribute__ (
2672   (__nothrow__ , __leaf__));
2673 extern long double __fmodl (long double __x, long double __y) __attribute__ (
2674   (__nothrow__ , __leaf__));
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

2666
2667 extern int __isinfl (long double __value) __attribute__ ((__nothrow__ ,
2668   __leaf__)) __attribute__ ((__const__));
2669
2670 extern int __finitel (long double __value) __attribute__ ((__nothrow__ ,
2671   __leaf__)) __attribute__ ((__const__));
2672
2673
2674
2675
2676 extern int isinfl (long double __value) __attribute__ ((__nothrow__ ,
2677   __leaf__)) __attribute__ ((__const__));
2678
2679 extern int finitel (long double __value) __attribute__ ((__nothrow__ ,
2680   __leaf__)) __attribute__ ((__const__));
2681
2682 extern long double dreml (long double __x, long double __y) __attribute__ (
2683   __nothrow__ , __leaf__); extern long double __dreml (long double __x
2684   , long double __y) __attribute__ ((__nothrow__ , __leaf__));
2685
2686 extern long double significndl (long double __x) __attribute__ (((
2687   __nothrow__ , __leaf__)); extern long double __significndl (long
2688   double __x) __attribute__ ((__nothrow__ , __leaf__));
2689
2690
2691
2692 extern long double copysignl (long double __x, long double __y)
2693   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2694   extern long double __copysignl (long double __x, long double __y)
2695   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2696
2697
2698
2699 extern long double nanl (const char * __tagb) __attribute__ ((__nothrow__ ,
2700   __leaf__)) __attribute__ ((__const__)); extern long double __nanl (
2701   const char * __tagb) __attribute__ ((__nothrow__ , __leaf__))
2702   __attribute__ ((__const__));
2703
2704

```

```

2705 extern int __isnanl (long double __value) __attribute__ ((__nothrow__ ,
2706   __leaf__)) __attribute__ ((__const__));
2707
2708
2709 extern int isnanl (long double __value) __attribute__ ((__nothrow__ ,
2710   __leaf__)) __attribute__ ((__const__));
2711
2712 extern long double j0l (long double) __attribute__ ((__nothrow__ ,
2713   __leaf__)); extern long double __j0l (long double) __attribute__ (((
2714   __nothrow__ , __leaf__)));
2715 extern long double j1l (long double) __attribute__ ((__nothrow__ ,
2716   __leaf__)); extern long double __j1l (long double) __attribute__ (((
2717   __nothrow__ , __leaf__)));
2718 extern long double jnl (int, long double) __attribute__ ((__nothrow__ ,
2719   __leaf__)); extern long double __jnl (int, long double) __attribute__ (((
2720   __nothrow__ , __leaf__)));
2721
2722
2723
2724 extern long double erfl (long double) __attribute__ ((__nothrow__ ,
2725   __leaf__)); extern long double __erfl (long double) __attribute__ (((
2726   __nothrow__ , __leaf__)));
2727 extern long double erfcl (long double) __attribute__ ((__nothrow__ ,
2728   __leaf__)); extern long double __erfcl (long double) __attribute__ (((
2729   __nothrow__ , __leaf__)));
2730
2731
2732
2733 extern long double lgammal (long double) __attribute__ ((__nothrow__ ,
2734   __leaf__)); extern long double __lgammal (long double) __attribute__ (((
2735   __nothrow__ , __leaf__)));
2736

```

```

2737
2738
2739 extern long double gammal (long double) __attribute__ ((__nothrow__ ,
2740   __leaf__));
2741 extern long double __gammal (long double) __attribute__ ((
2742   __nothrow__ , __leaf__));
2743
2744
2745
2746 extern long double lgammal_r (long double, int *__signgamp) __attribute__ (
2747   __nothrow__ , __leaf__);
2748 extern long double __lgammal_r (long double
2749   , int *__signgamp) __attribute__ ((__nothrow__ , __leaf__));
2750
2751
2752
2753
2754 extern long double rintl (long double __x) __attribute__ ((__nothrow__ ,
2755   __leaf__));
2756 extern long double __rintl (long double __x) __attribute__ (
2757   __nothrow__ , __leaf__);
2758
2759
2760
2761
2762
2763 extern long double nextafterl (long double __x, long double __y)
2764   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2765 extern long double __nextafterl (long double __x, long double __y)
2766   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2767
2768
2769
2770
2771 extern long double nexttowardl (long double __x, long double __y)
2772   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2773 extern long double __nexttowardl (long double __x, long double __y)
2774   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__));
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
3999

```

```

2772
2773
2774
2775
2776 extern long double scalblnl (long double __x, long int __n) __attribute__
2777   ((__nothrow__, __leaf__)); extern long double __scalblnl (long double
2778   __x, long int __n) __attribute__ ((__nothrow__, __leaf__));
2779
2780 extern long double nearbyintl (long double __x) __attribute__ (((
2781   __nothrow__, __leaf__)); extern long double __nearbyintl (long double
2782   __x) __attribute__ ((__nothrow__, __leaf__));
2783
2784 extern long double roundl (long double __x) __attribute__ ((__nothrow__,
2785   __leaf__)) __attribute__ ((__const__)); extern long double __roundl (
2786   long double __x) __attribute__ ((__nothrow__, __leaf__)) __attribute__
2787   ((__const__));
2788
2789 extern long double trunc1 (long double __x) __attribute__ ((__nothrow__,
2790   __leaf__)) __attribute__ ((__const__)); extern long double __trunc1 (
2791   long double __x) __attribute__ ((__nothrow__, __leaf__)) __attribute__
2792   ((__const__));
2793
2794 extern long double remquol (long double __x, long double __y, int *__quo)
2795   __attribute__ ((__nothrow__, __leaf__)); extern long double __remquol
2796   (long double __x, long double __y, int *__quo) __attribute__ (((
2797   __nothrow__, __leaf__));
2798
2799
2800 extern long int lrintl (long double __x) __attribute__ ((__nothrow__,
2801   __leaf__)); extern long int __lrintl (long double __x) __attribute__ (((
2802   __nothrow__, __leaf__));
2803 __extension__
2804 extern long long int llrintl (long double __x) __attribute__ ((__nothrow__,
2805   __leaf__)); extern long long int __llrintl (long double __x)
2806   __attribute__ ((__nothrow__, __leaf__));
2807
2808 extern long int lroundl (long double __x) __attribute__ ((__nothrow__,
2809   __leaf__)); extern long int __lroundl (long double __x) __attribute__
2810   ((__nothrow__, __leaf__));

```

```

2807 --extension--
2808 extern long long int llroundl (long double __x) --attribute__ ((  

2809     __nothrow__ , __leaf__)); extern long long int __llroundl (long double  

2810     __x) --attribute__ ((__nothrow__ , __leaf__));
2811
2812 extern long double fdiml (long double __x, long double __y) --attribute__  

2813     ((__nothrow__ , __leaf__)); extern long double __fdiml (long double __x  

2814     , long double __y) --attribute__ ((__nothrow__ , __leaf__));
2815
2816 extern long double fmaxl (long double __x, long double __y) --attribute__  

2817     ((__nothrow__ , __leaf__)) --attribute__ ((__const__)); extern long  

2818     double __fmaxl (long double __x, long double __y) --attribute__ ((  

2819         __nothrow__ , __leaf__)) --attribute__ ((__const__));
2820
2821 extern long double fminl (long double __x, long double __y) --attribute__  

2822     ((__nothrow__ , __leaf__)) --attribute__ ((__const__)); extern long  

2823     double __fminl (long double __x, long double __y) --attribute__ ((  

2824         __nothrow__ , __leaf__)) --attribute__ ((__const__));
2825
2826 extern int __fpclassifyl (long double __value) --attribute__ ((__nothrow__  

2827     , __leaf__))
2828     --attribute__ ((__const__));
2829
2830 extern int __signbitl (long double __value) --attribute__ ((__nothrow__ ,  

2831     __leaf__))
2832     --attribute__ ((__const__));
2833
2834
2835 extern long double fmal (long double __x, long double __y, long double __z  

2836     ) --attribute__ ((__nothrow__ , __leaf__)); extern long double __fmal (  

2837     long double __x, long double __y, long double __z) --attribute__ ((  

2838         __nothrow__ , __leaf__));
2839
2840 # 371 "/usr/include/x86_64-linux-gnu/bits/mathcalls.h" 3 4
2841 extern long double scalbl (long double __x, long double __n) --attribute__  

2842     ((__nothrow__ , __leaf__)); extern long double __scalbl (long double  

2843     __x, long double __n) --attribute__ ((__nothrow__ , __leaf__));
2844 # 133 "/usr/include/math.h" 2 3 4
2845 # 148 "/usr/include/math.h" 3 4
2846 extern int signgam;
2847 # 189 "/usr/include/math.h" 3 4
2848 enum
2849 {

```

```

2844 FP_NAN =
2845
2846     0,
2847 FP_INFINITE =
2848
2849     1,
2850 FP_ZERO =
2851
2852     2,
2853 FP_SUBNORMAL =
2854
2855     3,
2856 FP_NORMAL =
2857
2858     4
2859 };
2860 # 301 "/usr/include/math.h" 3 4
2861 typedef enum
2862 {
2863     _IEEE_ = -1,
2864     _SVID_,
2865     _XOPEN_,
2866     _POSIX_,
2867     _ISOC_
2868 } _LIB_VERSION_TYPE;
2869
2870
2871
2872
2873 extern _LIB_VERSION_TYPE _LIB_VERSION;
2874 # 326 "/usr/include/math.h" 3 4
2875 struct exception
2876 {
2877     int type;
2878     char *name;
2879     double arg1;
2880     double arg2;
2881     double retval;
2882 };
2883
2884
2885
2886
2887
2888 extern int matherr (struct exception *_exc);
2889 # 488 "/usr/include/math.h" 3 4
2890
2891 # 7 "headers/all_headers.h" 2
2892 # 1 "/usr/include/setjmp.h" 1 3 4
2893 # 27 "/usr/include/setjmp.h" 3 4
2894
2895
2896 # 1 "/usr/include/x86_64-linux-gnu/bits/setjmp.h" 1 3 4
2897 # 26 "/usr/include/x86_64-linux-gnu/bits/setjmp.h" 3 4

```

```

2898 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
2899 # 27 "/usr/include/x86_64-linux-gnu/bits/setjmp.h" 2 3 4
2900
2901
2902
2903
2904 typedef long int __jmp_buf[8];
2905 # 30 "/usr/include/setjmp.h" 2 3 4
2906 # 1 "/usr/include/x86_64-linux-gnu/bits/sigset.h" 1 3 4
2907 # 31 "/usr/include/setjmp.h" 2 3 4
2908
2909
2910
2911 struct __jmp_buf_tag
2912 {
2913
2914
2915
2916
2917     __jmp_buf __jmpbuf;
2918     int __mask_was_saved;
2919     __sigset_t __saved_mask;
2920 };
2921
2922
2923
2924
2925 typedef struct __jmp_buf_tag jmp_buf[1];
2926
2927
2928
2929 extern int setjmp (jmp_buf __env) __attribute__((__nothrow__));
2930
2931
2932
2933
2934
2935
2936 extern int __sigsetjmp (struct __jmp_buf_tag __env[1], int __savemask)
2937     __attribute__((__nothrow__));
2938
2939
2940 extern int _setjmp (struct __jmp_buf_tag __env[1]) __attribute__((
2941     __nothrow__));
2942
2943
2944
2945
2946
2947
2948
2949

```

```

2950
2951 extern void longjmp (struct __jmp_buf_tag __env[1], int __val)
2952     __attribute__ ((__nothrow__)) __attribute__ ((__noreturn__));
2953
2954
2955
2956
2957
2958
2959
2960 extern void _longjmp (struct __jmp_buf_tag __env[1], int __val)
2961     __attribute__ ((__nothrow__)) __attribute__ ((__noreturn__));
2962
2963
2964
2965
2966
2967
2968
2969 typedef struct __jmp_buf_tag sigjmp_buf [1];
2970 # 102 "/usr/include/setjmp.h" 3 4
2971 extern void siglongjmp (sigjmp_buf __env, int __val)
2972     __attribute__ ((__nothrow__)) __attribute__ ((__noreturn__));
2973 # 112 "/usr/include/setjmp.h" 3 4
2974
2975 # 8 "headers/all_headers.h" 2
2976 # 1 "/usr/include/string.h" 1 3 4
2977 # 27 "/usr/include/string.h" 3 4
2978
2979
2980
2981
2982
2983 # 1 "/usr/lib/gcc/x86_64-linux-gnu/5/include/stddef.h" 1 3 4
2984 # 33 "/usr/include/string.h" 2 3 4
2985 # 44 "/usr/include/string.h" 3 4
2986
2987
2988 extern void *memcpy (void *__restrict __dest, const void *__restrict __src
2989     ,
2990     size_t __n) __attribute__ ((__nothrow__, __leaf__)) __attribute__
2991 ((__nonnull__(1, 2)));
2992
2993 extern void *memmove (void *__dest, const void *__src, size_t __n)
2994     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
2995 (1, 2)));
2996
2997
2998
2999

```

```

3000 extern void *memccpy (void *_restrict __dest, const void *_restrict
3001     __src,
3002         int __c, size_t __n)
3003     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3004     (1, 2)));
3005
3006
3007
3008 extern void *memset (void *__s, int __c, size_t __n) __attribute__ (((
3009     __nothrow__, __leaf__)) __attribute__ ((__nonnull__ (1)));
3010
3011
3012 extern int memcmp (const void *__s1, const void *__s2, size_t __n)
3013     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3014     __attribute__ ((__nonnull__ (1, 2)));
3015 # 96 "/usr/include/string.h" 3 4
3016 extern void *memchr (const void *__s, int __c, size_t __n)
3017     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3018     __attribute__ ((__nonnull__ (1)));
3019
3020
3021
3022 extern char *strcpy (char *_restrict __dest, const char *_restrict __src
3023     )
3024     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3025     (1, 2)));
3026
3027
3028
3029 extern char *strncpy (char *_restrict __dest,
3030     const char *_restrict __src, size_t __n)
3031     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3032     (1, 2)));
3033
3034
3035
3036 extern char *strcat (char *_restrict __dest, const char *_restrict __src
3037     )
3038     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3039     (1, 2)));
3040
3041
3042
3043 extern char *strncat (char *_restrict __dest, const char *_restrict
3044     __src,
3045         size_t __n) __attribute__ ((__nothrow__, __leaf__)) __attribute__ (
3046         ((__nonnull__ (1, 2))));
3047
3048
3049
3050 extern int strcmp (const char *__s1, const char *__s2)
3051     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3052     __attribute__ ((__nonnull__ (1, 2)));
3053
3054
3055
3056 extern int strncmp (const char *__s1, const char *__s2, size_t __n)

```

```

3040     __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
3041     __attribute__((__nonnull__(1, 2)));
3042
3043 extern int strcoll (const char *__s1, const char *__s2)
3044     __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
3045     __attribute__((__nonnull__(1, 2)));
3046
3047 extern size_t strxfrm (char *__restrict __dest,
3048     const char *__restrict __src, size_t __n)
3049     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
3050     (2)));
3051
3052
3053
3054
3055 # 1 "/usr/include/xlocale.h" 1 3 4
3056 # 27 "/usr/include/xlocale.h" 3 4
3057 typedef struct __locale_struct
3058 {
3059
3060     struct __locale_data *__locales[13];
3061
3062
3063     const unsigned short int *__ctype_b;
3064     const int *__ctype_tolower;
3065     const int *__ctype_toupper;
3066
3067
3068     const char *__names[13];
3069 } *__locale_t;
3070
3071
3072 typedef __locale_t locale_t;
3073 # 164 "/usr/include/string.h" 2 3 4
3074
3075
3076 extern int strcoll_l (const char *__s1, const char *__s2, __locale_t __l)
3077     __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
3078     __attribute__((__nonnull__(1, 2, 3)));
3079
3080 extern size_t strxfrm_l (char *__dest, const char *__src, size_t __n,
3081     __locale_t __l) __attribute__((__nothrow__, __leaf__)) __attribute__((
3082     __nonnull__(2, 4)));
3083
3084
3085
3086 extern char *strdup (const char *__s)
3087     __attribute__((__nothrow__, __leaf__)) __attribute__((__malloc__))
3088     __attribute__((__nonnull__(1)));

```

```

3088
3089
3090
3091
3092
3093
3094 extern char *strndup (const char *_string, size_t _n)
3095     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__malloc__))
3096     __attribute__ ((__nonnull__(1)));
3097 # 211 "/usr/include/string.h" 3 4
3098
3099 # 236 "/usr/include/string.h" 3 4
3100 extern char *strchr (const char *_s, int _c)
3101     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3102     __attribute__ ((__nonnull__(1)));
3103 # 263 "/usr/include/string.h" 3 4
3104 extern char *strrchr (const char *_s, int _c)
3105     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3106     __attribute__ ((__nonnull__(1)));
3107
3108
3109
3110 extern size_t strcspn (const char *_s, const char *_reject)
3111     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3112     __attribute__ ((__nonnull__(1, 2)));
3113
3114 extern size_t strspn (const char *_s, const char *_accept)
3115     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3116     __attribute__ ((__nonnull__(1, 2)));
3117 # 315 "/usr/include/string.h" 3 4
3118 extern char *strpbrk (const char *_s, const char *_accept)
3119     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3120     __attribute__ ((__nonnull__(1, 2)));
3121 # 342 "/usr/include/string.h" 3 4
3122 extern char *strstr (const char *_haystack, const char *_needle)
3123     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__pure__))
3124     __attribute__ ((__nonnull__(1, 2)));
3125
3126 extern char *strtok (char *_restrict _s, const char *_restrict __delim)
3127     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__(2)));
3128
3129
3130
3131
3132 extern char *_strtok_r (char *_restrict _s,
3133     const char *_restrict __delim,

```

```

3134     char **__restrict __save_ptr)
3135     __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
3136     (2, 3)));
3137
3137 extern char *strtok_r (char *__restrict __s, const char *__restrict
3138   __delim,
3139   char **__restrict __save_ptr)
3140   __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
3141   (2, 3)));
3140 # 397 "/usr/include/string.h" 3 4
3141
3142
3143 extern size_t strlen (const char *__s)
3144   __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
3145   __attribute__((__nonnull__(1)));
3146
3147
3148
3149
3150 extern size_t strnlen (const char *__string, size_t __ maxlen)
3151   __attribute__((__nothrow__, __leaf__)) __attribute__((__pure__))
3152   __attribute__((__nonnull__(1)));
3153
3154
3155
3156
3157 extern char *strerror (int __errnum) __attribute__((__nothrow__,
3158   __leaf__));
3159 # 427 "/usr/include/string.h" 3 4
3160 extern int strerror_r (int __errnum, char *__buf, size_t __buflen) __asm__
3161   ("" "__xpg_strerror_r") __attribute__((__nothrow__, __leaf__))
3162   __attribute__((__nonnull__(2)));
3163 # 445 "/usr/include/string.h" 3 4
3164 extern char *strerror_l (int __errnum, __locale_t __l) __attribute__((
3165   __nothrow__, __leaf__));
3166
3167
3168
3169
3170 extern void __bzero (void *__s, size_t __n) __attribute__((__nothrow__,
3171   __leaf__)) __attribute__((__nonnull__(1)));
3172
3173
3174 extern void bcopy (const void *__src, void *__dest, size_t __n)
3175   __attribute__((__nothrow__, __leaf__)) __attribute__((__nonnull__
3176   (1, 2)));
3176
3177

```

```

3178 extern void bzero (void *__s, size_t __n) __attribute__ ((__nothrow__ ,
3179   __leaf__)) __attribute__ ((__nonnull__ (1)));
3180
3181 extern int bcmp (const void *__s1, const void *__s2, size_t __n)
3182   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3183   __attribute__ ((__nonnull__ (1, 2)));
3183 # 489 "/usr/include/string.h" 3 4
3184 extern char *index (const char *__s, int __c)
3185   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3186   __attribute__ ((__nonnull__ (1)));
3186 # 517 "/usr/include/string.h" 3 4
3187 extern char *rindex (const char *__s, int __c)
3188   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3189   __attribute__ ((__nonnull__ (1)));
3190
3191
3192
3193 extern int ffs (int __i) __attribute__ ((__nothrow__ , __leaf__))
3194   __attribute__ ((__const__));
3194 # 534 "/usr/include/string.h" 3 4
3195 extern int strcasecmp (const char *__s1, const char *__s2)
3196   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3197   __attribute__ ((__nonnull__ (1, 2)));
3198
3199 extern int strncasecmp (const char *__s1, const char *__s2, size_t __n)
3200   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
3201   __attribute__ ((__nonnull__ (1, 2)));
3201 # 557 "/usr/include/string.h" 3 4
3202 extern char *strsep (char **__restrict __stringp,
3203   const char *__restrict __delim)
3204   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
3205   (1, 2)));
3206
3207
3208
3209 extern char *strsignal (int __sig) __attribute__ ((__nothrow__ , __leaf__))
3210   ;
3211
3212 extern char *_stpcpy (char *__restrict __dest, const char *__restrict
3213   __src)
3214   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
3215   (1, 2)));
3214 extern char *stpcpy (char *__restrict __dest, const char *__restrict __src
3216   )
3217   __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
3218   (1, 2)));

```

```

3219 extern char *_stpncpy (char * __restrict __dest,
3220   const char * __restrict __src, size_t __n)
3221     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3222     (1, 2)));
3222 extern char *stpcpy (char * __restrict __dest,
3223   const char * __restrict __src, size_t __n)
3224     __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__
3225     (1, 2)));
3225 # 644 "/usr/include/string.h" 3 4
3226
3227 # 9 "headers/all_headers.h" 2
3228 # 1 "headers/sen_basic_type.h" 1
3229 # 1 "headers/sen_object.h" 1
3230
3231
3232
3233
3234
3235
3236
3237 # 7 "headers/sen_object.h"
3238 struct Sen_object_vtable;
3239 typedef struct Sen_object_vtable Sen_object_vtable;
3240
3241 struct Sen_object_class;
3242 typedef struct Sen_object_class Sen_object_class;
3243
3244 struct Sen_object;
3245 typedef struct Sen_object Sen_object;
3246
3247 struct Sen_object_vtable {
3248   void (*print) (Sen_object *);
3249   Sen_object *(*copy) (Sen_object *);
3250 };
3251
3252
3253
3254 struct Sen_object_class {
3255   Sen_object_vtable *tablep;
3256 };
3257
3258
3259
3260 struct Sen_object {
3261 # 30 "headers/sen_object.h" 3 4
3262   _Bool
3263 # 30 "headers/sen_object.h"
3264   bound;
3265   Sen_object_class *classp;
3266 };
3267
3268
3269 extern Sen_object_class Sen_object_class_;
3270 extern Sen_object_vtable Sen_object_vtable_;

```

```

3271 void print_object (Sen_object *);
3272 # 2 "headers/sen_basic_type.h" 2
3274
3275
3276
3277
3278 struct Sen_basic_type_vtable;
3279 typedef struct Sen_basic_type_vtable Sen_basic_type_vtable;
3280
3281 struct Sen_basic_type_class;
3282 typedef struct Sen_basic_type_class Sen_basic_type_class;
3283
3284 struct Sen_basic_type;
3285 typedef struct Sen_basic_type Sen_basic_type;
3286
3287 typedef enum {BOOL, INT, STR, UNK} Type;
3288
3289 struct Sen_basic_type_vtable {
3290     void (*print) (Sen_object *);
3291     Sen_basic_type *(*construct) (void *);
3292     void *(*get_val) (Sen_basic_type *);
3293     void *(*set_val) (Sen_basic_type *, void *);
3294     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
3295 };
3296
3297 struct Sen_basic_type_class {
3298     Sen_object_class *superp;
3299     Sen_basic_type_vtable *tablep;
3300     Type type;
3301 };
3302
3303 struct Sen_basic_type {
3304
3305 # 32 "headers/sen_basic_type.h" 3 4
3306     _Bool
3307 # 32 "headers/sen_basic_type.h"
3308         bound;
3309     Sen_basic_type_class *classp;
3310     Sen_object *superp;
3311 };
3312
3313 extern Sen_basic_type_class Sen_basic_type_class_;
3314 extern Sen_basic_type_vtable Sen_basic_type_vtable_;
3315
3316 void * get_val_basic_type (Sen_basic_type *);
3317 void * set_val_basic_type (Sen_basic_type *, void *);
3318 Sen_basic_type * add_basic_type (Sen_basic_type *, Sen_basic_type *);
3319 # 10 "headers/all_headers.h" 2
3320 # 1 "headers/sen_int.h" 1
3321 # 1 "headers/sen_basic_type.h" 1
3322 # 2 "headers/sen_int.h" 2
3323
3324

```

```

3325
3326
3327 struct Sen_int_vtable;
3328 typedef struct Sen_int_vtable Sen_int_vtable;
3329
3330 struct Sen_int_class;
3331 typedef struct Sen_int_class Sen_int_class;
3332
3333 struct Sen_int;
3334 typedef struct Sen_int Sen_int;
3335
3336 struct Sen_int_vtable {
3337     void (*print) (Sen_object *);
3338     void *(*get_val) (Sen_basic_type *);
3339     void *(*set_val) (Sen_basic_type *, void *);
3340     Sen_int *(*construct) (int);
3341     void (*destruct) (Sen_int *);
3342     Sen_int *(*copy) (Sen_int *);
3343     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
3344 };
3345
3346 struct Sen_int_class {
3347     Sen_basic_type_class *superp;
3348     Sen_int_vtable *tablep;
3349     Type type;
3350 };
3351
3352 struct Sen_int {
3353
3354 # 32 "headers/sen_int.h" 3 4
3355     _Bool
3356 # 32 "headers/sen_int.h"
3357         bound;
3358     Sen_int_class *classp;
3359     Sen_basic_type *superp;
3360     int val;
3361 };
3362
3363 extern Sen_int_class Sen_int_class_;
3364 extern Sen_int_vtable Sen_int_vtable_;
3365
3366 void print_int (Sen_object *);
3367 Sen_int * construct_int (int);
3368 void *get_val_int (Sen_basic_type *);
3369 void *set_val_int (Sen_basic_type *, void *);
3370 # 11 "headers/all_headers.h" 2
3371
3372 # 1 "headers/sen_bool.h" 1
3373 # 1 "headers/sen_basic_type.h" 1
3374 # 2 "headers/sen_bool.h" 2
3375
3376
3377
3378

```

```

3379 struct Sen_bool_vtable;
3380 typedef struct Sen_bool_vtable Sen_bool_vtable;
3381
3382 struct Sen_bool_class;
3383 typedef struct Sen_bool_class Sen_bool_class;
3384
3385 struct Sen_bool;
3386 typedef struct Sen_bool Sen_bool;
3387
3388 struct Sen_bool_vtable {
3389     void (*print) (Sen_object *);
3390     void *(*get_val) (Sen_basic_type *);
3391     void *(*set_val) (Sen_basic_type *, void *);
3392     Sen_bool *(*construct) (
3393 # 19 "headers/sen_bool.h" 3 4
3394             _Bool
3395 # 19 "headers/sen_bool.h"
3396             );
3397     void (*destruct) (Sen_bool *);
3398     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
3399 };
3400
3401 struct Sen_bool_class {
3402     Sen_basic_type_class *superp;
3403     Sen_bool_vtable *tablep;
3404     Type type;
3405 };
3406
3407 struct Sen_bool {
3408
3409 # 31 "headers/sen_bool.h" 3 4
3410     _Bool
3411 # 31 "headers/sen_bool.h"
3412         bound;
3413     Sen_bool_class *classp;
3414     Sen_basic_type *superp;
3415
3416 # 34 "headers/sen_bool.h" 3 4
3417     _Bool
3418 # 34 "headers/sen_bool.h"
3419         val;
3420 };
3421
3422 extern Sen_bool_class Sen_bool_class_;
3423 extern Sen_bool_vtable Sen_bool_vtable_;
3424
3425 void print_bool (Sen_object *);
3426 Sen_bool * construct_bool (
3427 # 41 "headers/sen_bool.h" 3 4
3428             _Bool
3429 # 41 "headers/sen_bool.h"
3430             );
3431 void *get_val_bool (Sen_basic_type *);
3432 void *set_val_bool (Sen_basic_type *, void *);

```

```

3433 # 13 "headers/all_headers.h" 2
3434 # 1 "headers/sen_string.h" 1
3435 # 1 "headers/sen_basic_type.h" 1
3436 # 2 "headers/sen_string.h" 2
3437
3438
3439
3440
3441 struct Sen_string_vtable;
3442 typedef struct Sen_string_vtable Sen_string_vtable;
3443
3444 struct Sen_string_class;
3445 typedef struct Sen_string_class Sen_string_class;
3446
3447 struct Sen_string;
3448 typedef struct Sen_string Sen_string;
3449
3450 struct Sen_string_vtable {
3451     void (*print) (Sen_object *);
3452     void *(*get_val) (Sen_basic_type *);
3453     void *(*set_val) (Sen_basic_type *, void *);
3454     Sen_string *(*construct) (char *);
3455     void (*destruct) (Sen_string *);
3456     Sen_basic_type *(*add) (Sen_basic_type *, Sen_basic_type *);
3457 };
3458
3459 struct Sen_string_class {
3460     Sen_basic_type_class *superp;
3461     Sen_string_vtable *tablep;
3462     Type type;
3463 };
3464
3465 struct Sen_string {
3466
3467 # 31 "headers/sen_string.h" 3 4
3468     _Bool
3469 # 31 "headers/sen_string.h"
3470         bound;
3471     Sen_string_class *classp;
3472     Sen_basic_type *superp;
3473     char *val;
3474 };
3475
3476 extern Sen_string_class Sen_string_class_;
3477 extern Sen_string_vtable Sen_string_vtable_;
3478
3479 void print_string (Sen_object *);
3480 Sen_string * construct_string (char *);
3481 void *get_val_string (Sen_basic_type *);
3482 void *set_val_string (Sen_basic_type *, void *);
3483 # 14 "headers/all_headers.h" 2
3484 # 1 "headers/sen_array.h" 1
3485
3486 # 1 "headers/sen_basic_type.h" 1

```

```

3487 # 3 "headers/sen_array.h" 2
3488 # 1 "headers/sen_int.h" 1
3489 # 1 "headers/sen_basic_type.h" 1
3490 # 2 "headers/sen_int.h" 2
3491 # 4 "headers/sen_array.h" 2
3492 # 12 "headers/sen_array.h"
3493 struct Sen_array_vtable;
3494 typedef struct Sen_array_vtable Sen_array_vtable;
3495
3496 struct Sen_array_class;
3497 typedef struct Sen_array_class Sen_array_class;
3498
3499 struct Sen_array;
3500 typedef struct Sen_array Sen_array;
3501
3502 struct Sen_array_vtable {
3503     void (*print) (Sen_object *);
3504     Sen_array *(*construct) (Sen_object **, int);
3505     void (*destruct) (Sen_array *);
3506     Sen_object *(*access) (Sen_array *, Sen_int *);
3507     Sen_array *(*concat) (Sen_array *, Sen_array *);
3508 };
3509
3510 struct Sen_array_class {
3511     Sen_object_class *superp;
3512     Sen_array_vtable *tablep;
3513 };
3514
3515 struct Sen_array {
3516
3517 # 35 "headers/sen_array.h" 3 4
3518     _Bool
3519 # 35 "headers/sen_array.h"
3520     bound;
3521     Sen_array_class *classp;
3522     Sen_object *superp;
3523     void **arr;
3524     int len;
3525     char print_sep;
3526 };
3527
3528 extern Sen_array_class Sen_array_class_;
3529 extern Sen_array_vtable Sen_array_vtable_;
3530
3531 Sen_array *construct_array (Sen_object **, int);
3532 Sen_object *access_array (Sen_array *, Sen_int *);
3533 Sen_array *concat_array (Sen_array *, Sen_array *);
3534 # 15 "headers/all_headers.h" 2
3535 # 2 "main.c" 2
3536
3537 int main() {
3538 # 29 "main.c"
3539     __auto_type s = ((Sen_string*) construct_string("tttesting "));
3540     s->bound =

```

```

3541 # 30 "main.c" 3 4
3542           1
3543 # 30 "main.c"
3544           ;
3545     __auto_type ss = ((Sen_string*) construct_string("hooray!!\n"));
3546     ss->bound=
3547 # 32 "main.c" 3 4
3548           1
3549 # 32 "main.c"
3550           ;
3551     Sen_int *arr_[2] = {((Sen_int*) construct_int(100)), ((Sen_int*) construct_int(50))};
3552     printf ("%d\n\n", (int)sizeof(arr_));
3553
3554
3555     { typeof(((Sen_string *) ss)) __temp__ = ((Sen_string *) ss); __temp__
3556     ->classp->tablep->print(((Sen_object *) __temp__));
3557     { typeof(((Sen_string *)s->classp->tablep->add((Sen_basic_type *)s, (Sen_basic_type *)ss))) __temp__ = ((Sen_string *)s->classp->tablep->add((Sen_basic_type *)s, (Sen_basic_type *)ss)); __temp__->classp->tablep->print(((Sen_object *) __temp__));
3558     ((Sen_string *) s)->classp->tablep->destruct(((Sen_string *) s));
3559     ((Sen_string *) ss)->classp->tablep->destruct(((Sen_string *) ss));
3560
3561     Sen_array *arr = (Sen_array_vtable_.construct(arr_, (sizeof((arr_)) / sizeof((arr_[0])))));
3562     printf ("%d\n", (int)sizeof((arr->arr)[0]));
3563
3564     return 0;
3565 }

```

./cast-test.snt

```

1 @setup
2 {
3
4 group A() {
5     int x;
6     int y;
7     int z;
8     func void __init__() {
9         1;
10        2;
11        3;
12    }
13 };
14
15 group B(A()) {
16     int x;
17     bool a;
18     bool b;
19 };
20
21 }

```

```

22
23
24 @turns
25 {
26
27 func void begin () {
28 }
29 }
30
31 func void test () {}
32
33 }
```

**./cast.ml**

```

1  (*
2   * C Abstract Syntax Tree (CAST)
3   *
4   * The CAST is a slightly modified version of the SAST, where inherited
5   * group attributes have been copied, in order, into groups.
6   *
7   * Also, the second argument to Group is filled in with Some(gdcl).
8   *)
9
10 open Types
11 open Sast
12
13 let rec get_attributes g = match g.extends with
14   None -> g.attributes
15 | Some(par) ->
16   let pa = get_attributes par in
17   pa @ g.attributes
18
19 let rec order_attrib = function
20   [] -> []
21 | g :: rest ->
22   let a = get_attributes g in
23   let new_g = {g with attributes = a} in
24   new_g :: order_attrib rest
25
26 let find_group g s =
27   if List.exists (fun x -> x.gname = s) g then
28     List.find (fun x -> x.gname = s) g
29   else
30     (* This is an internal dummy gdecl with name "" *)
31     { gname = ""; extends = None; par_actuals = None;
32       attributes = []; methods = [] }
33
34 let tag_groups_vdcl g v = match v.vtype with
35   Group(s, _) ->
36     let gdcl = find_group g s in
37     { v with vtype = Group(s, Some(gdcl)) }
38   | _ -> v
39
40 let rec tag_groups_field g = function
```

```

41 Var(vd) -> Var(tag_groups_vdcl g vd)
42 | Attrib(vd1, vd2) ->
43   let vd1 = tag_groups_vdcl g vd1 in
44   let vd2 = tag_groups_vdcl g vd2 in
45   Attrib(vd1, vd2)
46 | Fun(fd) -> Fun(tag_groups_func g fd)
47 | Method(vd, fd) ->
48   let vd = tag_groups_vdcl g vd in
49   let fd = tag_groups_func g fd in
50   Method(vd, fd)
51 | Grp(gd) -> Grp(gd)
52 | This -> This
53
54 and tag_groups_listlit g = function
55   Elems(el, name) -> Elems(List.map (tag_groups_expr g) el, name)
56   | EmptyList -> EmptyList
57
58 and tag_groups_expr g e =
59   let detail, typ = e in
60   let typ = match typ with
61     Group(s, _) -> Group(s, Some(find_group g s))
62     | _ as x -> x
63   in
64   let detail =
65     match detail with
66       ListLiteral(l1) ->
67         let l1 = tag_groups_listlit g l1 in
68         ListLiteral(l1)
69       | Field(fe) -> Field(tag_groups_field g fe)
70       | Binop(e1, op, e2) ->
71         let e1 = tag_groups_expr g e1 in
72         let e2 = tag_groups_expr g e2 in
73         Binop(e1, op, e2)
74       | Assign(fe, e) ->
75         let fe = tag_groups_field g fe in
76         let e = tag_groups_expr g e in
77         Assign(fe, e)
78       | Call(vd_opt, fd, el) ->
79         let vd_opt = match vd_opt with
80           Some(vd) -> Some(tag_groups_vdcl g vd)
81           | None -> None
82         in
83         let fd = tag_groups_func g fd in
84         let el = List.map (tag_groups_expr g) el in
85         Call(vd_opt, fd, el)
86       | Element(e1, e2) ->
87         let e1 = tag_groups_expr g e1 in
88         let e2 = tag_groups_expr g e2 in
89         Element(e1, e2)
90       | Uminus(e) -> Uminus(tag_groups_expr g e)
91       | Not(e) -> Not(tag_groups_expr g e)
92       | Remove(e) ->
93         let e = tag_groups_expr g e in
94         Remove(e)

```

```

95     | Place(e) ->
96         let e = tag_groups_expr g e in
97         Place(e)
98     | _ as x -> x
99   in
100  detail, typ
101
102 and tag_groups_stmt g = function
103   Block(scope, sl) -> Block(scope, List.map (tag_groups_stmt g) sl)
104   Expression(e) -> Expression(tag_groups_expr g e)
105   Return(e) -> Return(tag_groups_expr g e)
106   Break -> Break
107   Continue -> Continue
108   End -> End
109   Pass(fdcl, e) ->
110     let fdcl = tag_groups_func g fdcl in
111     let e = tag_groups_expr g e in
112     Pass(fdcl, e)
113   | If(e, s1, e_opt, s2) ->
114     let e = tag_groups_expr g e in
115     let e_opt = match e_opt with
116       None -> None
117       | Some(expr) -> Some(tag_groups_expr g expr)
118     in
119     let s1 = tag_groups_stmt g s1 in
120     let s2 = tag_groups_stmt g s2 in
121     If(e, s1, e_opt, s2)
122   | For(vd, el, s) ->
123     let vd = tag_groups_vdcl g vd in
124     let el = List.map (tag_groups_expr g) el in
125     let s = tag_groups_stmt g s in
126     For(vd, el, s)
127   | While(e, s) ->
128     let e = tag_groups_expr g e in
129     let s = tag_groups_stmt g s in
130     While(e, s)
131
132 and tag_groups_func g = function
133   BasicFunc(f) ->
134     let t = match f.ftype with
135       Group(s, _) -> Group(s, Some(find_group g s))
136       | _ -> f.ftype
137     in
138     let l = List.map (tag_groups_vdcl g) f.locals in
139     let fl = List.map (tag_groups_vdcl g) f.formals in
140     let b = List.map (tag_groups_stmt g) f.body in
141     BasicFunc({ f with ftype = t; locals = l; formals = fl; body = b })
142   | AssertFunc(f) ->
143     let l = List.map (tag_groups_vdcl g) f.alocals in
144     let fl = List.map (tag_groups_vdcl g) f.aformals in
145     let b = List.map (tag_groups_stmt g) f.abody in
146     AssertFunc({ f with alocals = l; aformals = fl; abody = b })
147
148 and tag_groups_grp g gd =

```

```

149 let a = List.map (tag_groups_vdcl g) gd.attributes in
150 let m = List.map (tag_groups_func g) gd.methods in
151 let pa = match gd.par_actualls with
152   Some(el) -> Some(List.map (tag_groups_expr g) el)
153 | None -> None
154 in
155 let ex = match gd.extends with
156   Some(par) -> Some(tag_groups_grp g par)
157 | None -> None
158 in
159 { gd with attributes = a; methods = m; par_actualls = pa; extends = ex }
160
161 let tag_program program =
162   let (v, f, g), turns = program in
163   let v = List.map (tag_groups_vdcl g) v in
164   let f = List.map (tag_groups_func g) f in
165   let g = List.map (tag_groups_grp g) g in
166   let turns = List.map (tag_groups_func g) turns in
167   (v, f, g), turns
168
169 let rec get_ll_type = function
170   Elems(el, _) ->
171     let _, typ = List.hd el in
172     typ
173   | EmptyList -> List_t(Void)
174
175 let fix_ll_lit_expr vars e =
176   let detail, typ = e in
177   let v_base =
178     { vname = ""; vtype = typ; vinit = Some(e); vloop = false }
179   in
180   match detail with
181     IntLiteral(i, name) -> { v_base with vname = name } :: vars
182   | StrLiteral(s, name) -> { v_base with vname = name } :: vars
183   (* | ListLiteral(ll) ->
184     literals.count <- literals.count + 1;
185     { v_base with vname = "__elem__" ^ literals.count; vtype = Int } :: vars *)
186   | BoolLiteral(bl, name) -> { v_base with vname = name } :: vars
187   | _ -> vars
188
189 let rec fix_ll vars = function
190   Elems(el, name) ->
191     let vars = List.fold_left fix_ll_lit_expr vars el in
192     let _, typ = List.hd el in
193     let typ = List_t(typ) in
194     let vdcl =
195       { vname = name;
196         vtype = typ;
197         vinit = Some(ListLiteral(Elems(el, name)), typ);
198         vloop = false }
199     in
200     vdcl :: vars
201   | EmptyList -> vars

```

```

202
203 and fix_ll_expr vars (expr_detail, expr_typ) = match expr_detail with
204   ListLiteral(ll) -> fix_ll vars ll
205   | Binop(e1, op, e2) ->
206     let vars = fix_ll_expr vars e1 in
207     let vars = fix_ll_expr vars e2 in
208     vars
209   | Assign(fe, e) -> fix_ll_expr vars e
210   | Call(vd_opt, fd, el) -> List.fold_left fix_ll_expr vars el
211   | Element(e1, e2) ->
212     let vars = fix_ll_expr vars e1 in
213     let vars = fix_ll_expr vars e2 in
214     vars
215   | Remove(e) -> fix_ll_expr vars e
216   | Place(e) -> fix_ll_expr vars e
217   | Uminus(e) -> fix_ll_expr vars e
218   | Not(e) -> fix_ll_expr vars e
219   | _ -> vars
220
221 let rec fix_ll_stmt vars = function
222   Block(_, sl) -> List.fold_left fix_ll_stmt vars sl
223   | Expression(e) -> fix_ll_expr vars e
224   | Return(e) -> fix_ll_expr vars e
225   | Pass(fd, e) -> fix_ll_expr vars e
226   | If(e, s1, e_opt, s2) ->
227     let vars = fix_ll_expr vars e in
228     let vars = fix_ll_stmt vars s1 in
229     let vars = match e_opt with
230       None -> vars
231       | Some(expr) -> fix_ll_expr vars expr in
232     let vars = fix_ll_stmt vars s2 in
233     vars
234   | For(vd, el, s) ->
235     let vars = List.fold_left fix_ll_expr vars el in
236     let vars = fix_ll_stmt vars s in
237     vars
238   | While(e, s) ->
239     let vars = fix_ll_expr vars e in
240     let vars = fix_ll_stmt vars s in
241     vars
242   | _ -> vars
243
244 let fix_ll_vdcl vars v = match v.vinit with
245   None -> v :: vars
246   | Some(e) -> (match e with
247     ListLiteral(ll), _ ->
248       let vdcl, vars = match ll with
249         Elems(el, name) ->
250           let vars = List.fold_left fix_ll_lit_expr vars el in
251           [{ v with vname = name }], vars
252         | EmptyList ->
253           [], vars
254       in
255       v :: vdcl @ vars

```

```

256     | _ -> v :: vars)
257
258 let fix_ll_vdcls vars =
259   let new_vars = List.fold_left fix_ll_vdcl [] vars in
260   new_vars (* @ vars *)
261
262 let fix_ll_fdcl = function
263   BasicFunc(f) ->
264     let new_vars = List.fold_left fix_ll_vdcl [] f.locals in
265     let new_vars = List.fold_left fix_ll_stmt new_vars f.body in
266     BasicFunc({ f with locals = List.rev new_vars (* @ f.locals *) })
267   | AssertFunc(f) ->
268     let new_vars = List.fold_left fix_ll_vdcl [] f.alocals in
269     let new_vars = List.fold_left fix_ll_stmt new_vars f.abody in
270     AssertFunc({ f with alocals = List.rev new_vars (* @ f.alocals *) })
271
272 let fix_ll_gdcl g =
273   let new_vars = List.fold_left fix_ll_vdcl [] g.attributes in
274   let g = { g with attributes = List.rev new_vars (* @ g.attributes *) }
275   in
276   let m = List.map fix_ll_fdcl g.methods in
277   { g with methods = m }
278
278 let correct_listlit program =
279   let (v, f, g), turns = program in
280   let v = List.rev (fix_ll_vdcls v) in
281   let f = List.map fix_ll_fdcl f in
282   let g = List.map fix_ll_gdcl g in
283   let turns = List.map fix_ll_fdcl turns in
284   (v, f, g), turns
285
286 let build_cast (program : Types.program) =
287   let (v, f, g), turns = program in
288   let g = order_attrib g in
289   let program = tag_program ((v, f, g), turns) in
290   let program = correct_listlit program in
291   program
292
293 (** -----
294 * -----
295 * Functions to convert the CAST to a string
296 * -----
297 *)
298
299 let rec string_of_t = function
300   Int -> "int"
301   | Bool -> "bool"
302   | Str -> "str"
303   | Void -> "void"
304   | List_t(vt) ->
305     "list[" ^ string_of_t vt ^ "]"
306   | Group(s, _) -> s
307
308 let string_of_field = function

```

```

309 Var(v) -> v.vname
310 | Fun(f) ->
311   (match f with
312     BasicFunc(x) -> x.fname
313     | AssertFunc(x) -> x.aname)
314 | Attrib(v1, v2) -> v1.vname ^ "." ^ v2.vname
315 | Method(v, f) -> v.vname ^ "." ^
316   (match f with
317     BasicFunc(x) -> x.fname
318     | AssertFunc(x) -> x.aname)
319 | Grp(g) -> g.gname
320 | This -> "this"
321
322 let rec string_of_list_lit = function
323   EmptyList -> "[]"
324   | Elems(e, n) ->
325     "[" ^ String.concat ", " (List.map string_of_expression e) ^ "]"
326
327 and string_of_expr_detail = function
328   IntLiteral(l, name) -> string_of_int l
329   | Field(f) -> string_of_field f
330   | Binop(e1, o, e2) ->
331     string_of_expression e1 ^ " " ^
332     (match o with
333       Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
334       | Equal -> "==" | Neq -> "!="
335       | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">="
336       | Mod -> "%"
337       | And -> "and" | Or -> "or" ) ^ " " ^
338     string_of_expression e2
339   | Assign(f, e) -> string_of_field f ^ " = " ^ string_of_expression e
340   | Call(vopt, f, el) ->
341     let par =
342       (match vopt with
343         None -> ""
344         | Some(v) -> v.vname) in
345     par ^ "." ^ string_of_field(Fun(f)) ^
346     "(" ^ String.concat ", " (List.map string_of_expression el) ^ ")"
347   | Noexpr -> "<Noexpr>"
348   | StrLiteral(s, name) -> Ast.escaped_string s
349   | Uminus(e) -> "-" ^ string_of_expression e
350   | Not(e) -> "not" ^ string_of_expression e
351   | Element(e1, e2) ->
352     string_of_expression e1 ^ "[" ^ string_of_expression e2 ^ "]"
353   | ListLiteral(l) -> string_of_list_lit l
354   | BoolLiteral(b, name) -> (match b with True -> "True" | False -> "False"
355   ")
356   | VoidLiteral -> "None"
357   | Place(e) -> string_of_expression e
358   | Remove(e) -> string_of_expression e
359
360 and string_of_expression e =
361   let detail, _ = e in
362   string_of_expr_detail detail

```

```

362
363 let string_of_vinit = function
364   None -> "<None>"
365   | Some(e) -> string_of_expression e
366
367 let string_of_vdecl vdecl =
368   "var_decl = { " ^
369   "vtype: " ^ string_of_t vdecl.vtype ^ "; " ^
370   "vname: " ^ vdecl.vname ^ "; " ^
371   "vinit: " ^ string_of_vinit vdecl.vinit ^
372   " }"
373
374 let rec string_of_stmt = function
375   Block(symbols, stmts) ->
376     "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
377   | Expression(e) -> string_of_expression e ^ ";\n";
378   | Return(e) -> "return " ^ string_of_expression e ^ ";\n";
379   | If(e, s1, e_opt, s2) -> "if (" ^ string_of_expression e ^ ")\n" ^
380     string_of_stmt s1 ^
381     (match e_opt with
382      None -> "else\n"
383      | Some(expr) -> string_of_expression e) ^
384     string_of_stmt s2
385   | For(vd, elist, s) ->
386     "for (" ^ string_of_vdecl vd ^ " in " ^
387     "{\n" ^ String.concat ", " (List.map string_of_expression
388     elist) ^ "}\n" ^
389     ") " ^ string_of_stmt s
390   | While(e, s) -> "while (" ^ string_of_expression e ^ "){\n" ^
391     string_of_stmt s ^ "\n}\n"
392   | Pass(e, s) -> "pass (" ^ string_of_field(Fun(e)) ^ ", " ^
393     string_of_expression s ^ ")\n"
394   | Break -> "break;\n"
395   | Continue -> "continue;\n"
396   | End -> "end();\n"
397
398 let string_of_basic_fdecl fdecl =
399   "basic_func_decl = {\n" ^
400   "  ftype: " ^ string_of_t fdecl.ftype ^ "; " ^
401   "  fname: " ^ fdecl.fname ^ "; " ^
402   "  turns_func: " ^ string_of_bool fdecl.turns_func ^ ";\n" ^
403   "  formals: {\n    " ^
404     String.concat ";\n    " (List.map string_of_vdecl fdecl.formals) ^ "\n  }\n" ^
405   "  locals: {\n    " ^
406     String.concat ";\n    " (List.map string_of_vdecl fdecl.locals) ^ "\n  }\n" ^
407   "  body: {\n    " ^
408     String.concat "" (List.map string_of_stmt fdecl.body) ^
409   " }\n"
410
411 let string_of_assert_decl fdecl =
412   "assert_func_decl = {\n" ^
413   "  fname: " ^ fdecl.fname ^ "; " ^

```

```

411 "  a_turns_func: " ^ string_of_bool fdecl.a_turns_func ^ ";\n" ^
412 "  aformals: {\n    " ^
413   String.concat ";\n      " (List.map string_of_vdecl fdecl.aformals) ^ "\n"
414   " }\n" ^
415 "  alocals: {\n    " ^
416   String.concat ";\n      " (List.map string_of_vdecl fdecl.alocals) ^ "\n"
417   " }\n" ^
418 "  abody: {\n    " ^
419   String.concat "" (List.map string_of_stmt fdecl.abody) ^
420 " }\n"
421
422 let string_of_fdecl = function
423   BasicFunc(f) -> string_of_basic_fdecl f
424   | AssertFunc(f) -> string_of_assert_decl f
425
426 let string_of_gdecl gdecl =
427   "group " ^ gdecl.gname ^ "(" ^
428   (match gdecl.extends with
429     Some(par) -> par.gname ^
430       (match gdecl.par_actuas with
431         Some(acts) ->
432           "(" ^ String.concat ", " (List.map
433             string_of_expression acts) ^ ")"
434           | None -> "")
435           | None -> "") ^ ")\n{\n" ^
436   String.concat "" (List.map (fun v -> string_of_vdecl v ^ ";\n") gdecl.
437   attributes) ^
438   String.concat "" (List.map string_of_fdecl gdecl.methods) ^
439 "};\n"
440
441 let string_of_setup s =
442   let vars, funcs, groups = s in
443   "@setup {\n\n" ^
444   String.concat "" (List.map (fun v -> string_of_vdecl v ^ ";\n") vars) ^
445   "\n" ^
446   String.concat "\n" (List.map string_of_fdecl funcs) ^
447   String.concat "\n" (List.map string_of_gdecl groups) ^
448   "\n}\n"
449
450 let string_of_turns t =
451   "@turns {\n\n" ^
452   String.concat "\n" (List.map string_of_fdecl t) ^
453   "\n}\n"
454
455 let string_of_program (program : program) =
456   let s, t = program in
457   string_of_setup s ^ string_of_turns t

```

./compile.ml

```

1 open Types
2 open Sast
3
4 type counter = {
5   mutable i : int

```

```

6 }
7
8 let count = { i = 0 }
9
10 let tmp_formal_prefix = "__tmp_form_"
11
12 let prefix_name n =
13   "snt_" ^ n
14
15 let senet_header =
16   "#include <stdbool.h>" ^ "\n" ^
17   "#include <stdio.h>" ^ "\n" ^
18   "#include <stdlib.h>" ^ "\n" ^
19   "#include <string.h>\n" ^
20   "#include \"temp/sen_linked_list.h\"\n" ^
21   "#include \"temp/sen_print_base_grps.h\"\n" ^
22   "#include \"temp/sen_init_base_grps.h\"\n" ^
23   "#include \"temp/sen_read.h\"\n" ^
24   "\n" ^
25   "struct SENET_NONE {\n" ^
26   "  } SENET_NONE;\n" ^
27   "\n" ^
28   "struct Sen_list snt_SEN_EMPTY_LIST;\n" ^
29   "\n" ^
30   "char *SENET_STR_CONCAT(char* s1, char* s2) {\n" ^
31   "  char *temp = (char *) malloc(strlen(s1)+ strlen(s2) +1);\n" ^
32   "  strcpy(temp, s1);\n" ^
33   "  strcat(temp, s2);\n" ^
34   "  return temp;\n" ^
35   "}\n" ^
36   "\n" ^
37   "void (*CUR_TURN)();\n" ^
38   "int snt_PLAYER_ON_MOVE = 0;\n" ^
39   "\n"
40
41 let senet_footer =
42   "int main() {\n" ^
43   "  CUR_TURN = &snt_begin;\n" ^
44   "  snt_PLAYER_ON_MOVE = 0;\n" ^
45   "  while (true) {\n" ^
46   "    CUR_TURN();\n" ^
47   "  }\n" ^
48   "  return 0;\n" ^
49   "}"
50
51 let binop_to_c = function
52   Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/" | Mod -> "%"
53   | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">="
54   | And -> "&&" | Or -> "||"
55   | Equal -> "==" | Neq -> "!="
56
57 let id_type_to_c = function
58   Int -> "int"
59   | Bool -> "bool"

```

```

60  | Str -> "char* "
61  | Void -> "void "
62  | List_t(typ) -> "struct Sen_list "
63  | Group(s, _) -> "struct " ^ prefix_name s ^ " "
64
65 let rec field_to_c = function
66   Var(v) ->
67     if v.vloop then
68       "*(" ^ prefix_name v.vname ^ " + __cnt__" ^ prefix_name v.vname ^ "
69     )"
70     else
71       prefix_name v.vname
72   Fun(f) -> prefix_name
73   (match f with
74    | BasicFunc(x) -> x.fname
75    | AssertFunc(x) -> x.aname)
76   Grp(g) -> prefix_name g.gname
77   Attrib(par, child) ->
78     let par_name, deref_op =
79       if par.vname = "this" then
80         "(*this)", "."
81       else if par.vloop then
82         "(" ^ prefix_name par.vname ^ " + __cnt__" ^ prefix_name par.vname
83         ^ ")", "->"
84       else
85         prefix_name par.vname, "."
86     in
87     par_name ^ deref_op ^ prefix_name child.vname
88   Method(v, f) -> raise (SemError ("Internal error: Method matched in
89   field_to_c(), use Call instead"))
90   This -> "(*this)"
91
92 let rec function_call_to_c = function
93   BasicFunc(f) -> f.fname
94   | AssertFunc(f) -> f.aname
95
96 let function_group_method = function
97   BasicFunc(f) -> f.group_method
98   | AssertFunc(f) -> f.a_group_method
99
100 let is_builtin_func = function
101   BasicFunc(f) -> f.f_is_builtin
102   | AssertFunc(f) -> f.a_is_builtin
103
104 let rec printf (detail, typ) =
105   let e_c_string = expression_to_c detail in
106   match typ with
107     Bool -> "printf(\"%s\", " ^ e_c_string ^ " ? \"true\" : \"false\" "
108     ")"
109     | Int -> "printf(\"%d\", " ^ e_c_string ^ ")"
110     | Str -> "printf(\"%s\", " ^ e_c_string ^ ")"
111     | Group(x, _) ->
112       "printf(\"%s\", " ^ prefix_name x ^ "_" ^ prefix_name "__repr__" ^
113       "((struct " ^ prefix_name x ^ "*) " ^ "&" ^ e_c_string ^ ")") ^ ")"

```

```

110 | Void -> "printf(\" ^ \"\\None\" \" ^ \")"
111 | List_t(l_typ) ->
112 let tmp_var, list_id = match detail with
113   Field(_) -> "", e_c_string
114   | Call(vd_opt, fd, _) ->
115     let tmp_name =
116       count.i <- count.i + 1;
117       "__tp__" ^ string_of_int count.i
118     in
119     id_type_to_c typ ^ tmp_name ^ " = " ^ e_c_string,
120     tmp_name
121   | _ -> "", prefix_name e_c_string
122 in
123 match l_typ with
124   | Group(x, g) ->
125     "printGroupList(&" ^ list_id ^ ", " ^
126     prefix_name x ^ "_" ^ prefix_name "__repr__" ^ ")"
127   | _ ->
128     let func = match l_typ with
129       Int -> "printInt"
130       | Str -> "printStr"
131       | Bool -> "printBool"
132       | Void -> "printEmptyList"
133       | Group(_, _) ->
134         raise (SemError ("Internal error: printList call with Group"
135       ))
136       | List_t(_) ->
137         raise (SemError ("Internal error: printList call with List_t"
138       ))
139       in
140       tmp_var ^ ";"^ "\n" ^
141       "printList(&" ^ list_id ^ ", " ^ func ^ ")"
142 and formal_to_c is_builtin_func v = match is_builtin_func, v.vtype with
143   false, Group(gname, _) ->
144     id_type_to_c v.vtype ^ "* " ^ tmp_formal_prefix ^ prefix_name v.
145     vname
146   | _, List_t(typ) ->
147     id_type_to_c v.vtype ^ "* " ^ tmp_formal_prefix ^ prefix_name v.
148     vname
149   | _, _ ->
150     id_type_to_c v.vtype ^ prefix_name v.vname
151 and function_group = function
152   BasicFunc(f) -> f.group_method
153   | AssertFunc(f) -> f.a_group_method
154 and ll_elem_to_c = function
155   IntLiteral(_, name)
156   | StrLiteral(_, name)
157   | BoolLiteral(_, name) -> prefix_name name
158   | _ as detail -> expression_to_c detail
159 and push_ll_to_new_list list_id = function

```

```

160 Elems(el, _) ->
161   let push_elem_to_new_list (detail, typ) =
162     "push(&" ^ prefix_name list_id ^
163     ", (void *) &(" ^ ll_elem_to_c detail ^ "))"
164   in
165   String.concat ";\n" (List.map push_elem_to_new_list el)
166 | EmptyList -> ""
167
168 and push_to_new_list list_id (det, typ) = match det with
169   ListLiteral(ll) -> push_ll_to_new_list list_id ll
170 | _ -> raise (SemError "Unsupported expression type to push to a new
list literal")
171
172 and var_decl_to_c v = match v.vtype, v.vinit with
173   _, None ->
174     id_type_to_c v.vtype ^
175     prefix_name v.vname ^ ";"
176 | List_t(typ), Some(e) ->
177   let detail, _ = e in
178   let name = expression_to_c detail in
179   id_type_to_c v.vtype ^
180   prefix_name v.vname ^
181   (if v.vname != "" && String.length v.vname > 5 &&
182     String.sub v.vname 0 6 = "__ll__" &&
183     String.sub name 0 6 = "__ll__" then
184     ";\n" ^
185     "new_Sen_list(&" ^ prefix_name name ^
186     ", sizeof(" ^ id_type_to_c typ ^ "));\n" ^
187     push_to_new_list name e ^ ";"
188   else
189     " = " ^ prefix_name name ^ ";")
190 | _, Some(e) ->
191   let detail, typ = e in
192   (match typ with
193     Group(_, _) ->
194       id_type_to_c typ ^ "__tmp__" ^ prefix_name v.vname ^
195       " = " ^ expression_to_c detail ^ ";\n" ^
196       id_type_to_c v.vtype ^ prefix_name v.vname ^
197       " = " ^ "*((" ^ id_type_to_c v.vtype ^ ")*) &" ^
198       " __tmp__" ^ prefix_name v.vname ^ ");"
199     | _ ->
200       id_type_to_c v.vtype ^ prefix_name v.vname ^
201       " = " ^ expression_to_c detail ^ ";")
202
203 and list_lit_to_c = function
204   Elems(el, name) -> name
205 | EmptyList -> "SEN_EMPTY_LIST"
206
207 and actual_to_c (detail, typ) =
208   let e_c_string = expression_to_c detail in
209   match typ with
210     List_t(l_typ) ->
211       (match detail with
212         Field(_) -> "&" ^ e_c_string

```

```

213     | _ -> "&" ^ prefix_name e_c_string)
214   | Group(name, gdcl) -> "&" ^ e_c_string
215   | _ -> e_c_string
216
217 and expression_to_c = function
218   IntLiteral(i, name) -> string_of_int i
219   StrLiteral(s, name) -> Ast.escaped_string s
220   ListLiteral(l1) -> list_lit_to_c l1
221   BoolLiteral(b, name) ->
222     (match b with True -> "true" | False -> "false")
223   VoidLiteral -> "SENET_NONE"
224   Field(fd) -> field_to_c fd
225   Binop(e1, op, e2) ->
226     let d1, t1 = e1 and d2, t2 = e2 in
227     let d1 = expression_to_c d1 and d2 = expression_to_c d2 in
228     (match t1, t2 with
229      Int, Int
230      | Bool, Bool ->
231        "(" ^ d1 ^ " " ^ binop_to_c op ^ " " ^ d2 ^ ")"
232      | Str, Str ->
233        let eval = match op with
234          Equal -> "? 0 : 1"
235          | Neq -> "? 1 : 0"
236          | _ -> ""
237        in
238        (match op with
239          Equal | Neq -> "(strcmp(" ^ d1 ^ ", " ^ d2 ^ ") " ^ eval ^ ")"
240
241          | Add -> "SENET_STR_CONCAT(" ^ d1 ^ ", " ^ d2 ^ ")"
242          | _ -> raise (SemError "Binop other than +, ==, or != has lhd
and rhs with type Str"))
243
244      | Void, Void ->
245        let ans = match op with
246          Equal -> True
247          | Neq -> False
248          | _ ->
249            raise (SemError "Binop other than == or != has lhs and rhs
with type Void")
250
251        in
252        expression_to_c (BoolLiteral(ans, ""))
253
254      | Group(_, gd1), Group(_, gd2) ->
255        let gd1, gd2 = match gd1, gd2 with
256          Some(x), Some(y) -> x, y
257          | _, _ ->
258            raise (SemError "Missing a group decl in expression type")
259
260        in
261        let eval = match op with
262          Equal -> "? 1 : 0"
263          | Neq -> "? 0 : 1"
264          | _ ->
265            raise (SemError "Binop other than == or != has lhs and rhs
with type Group(T)")
266
267        in
268        let compare_attrib e1 e2 a =

```

```

263         let e1, _ = e1 and e2, _ = e2 in
264         "(" ^ expression_to_c e1 ^ "." ^ prefix_name a.vname ^ " == "
265         " " ^
266         expression_to_c e2 ^ "." ^ prefix_name a.vname ^ ")"
267     in
268     if gd1.gname == gd2.gname then
269     let attrib = gd1.attributes in
270     "(" ^ "(" ^
271     String.concat " && " (List.map (compare_attrib e1 e2) attrib
272   ) ^
273   ") " ^ eval ^ ")"
274   else
275     "false"
276 (* | List_t(t1), List_t(t2) -> *)
277 | _, _ ->
278   raise (SemError ("Internal error: improper types in binop: " ^
279   d1 ^ " " ^ binop_to_c op ^ " " ^ d2)))
| Assign(fd, e) ->
280   let fd = field_to_c fd
281   and detail, typ = e in
282   fd ^ " = " ^
283   (match detail with
284   | ListLiteral(_) -> prefix_name (expression_to_c detail)
285   | _ -> expression_to_c detail)
| Call(vopt, fd, el) ->
286   (* let e = List.map (fun (detail, _) -> detail) el in *)
287   let argc = List.length el in
288   let fname = function_call_to_c fd in
289   let gname = function_group_method fd in
290   (* read and stoi take one argument, discard remainder *)
291   if fname = "print" && is_builtin_func fd then
292     String.concat ";\\n" (List.map printf el)
293   else if fname = "read" && is_builtin_func fd then
294     let detail, _ = List.hd el in
295     "_snt_read(" ^ expression_to_c detail ^ ")"
296   else if fname = "clear_input" && is_builtin_func fd then
297     "_snt_clear_input()"
298   else if fname = "stoi" && is_builtin_func fd then
299     let detail, _ = List.hd el in
300     "atoi(" ^ expression_to_c detail ^ ")"
301   else if fname = "exit" && is_builtin_func fd then
302     "exit(0)"
303   else if fname = "rand" && is_builtin_func fd then
304     "rand()"
305   else
306     let class_prefix = function_group fd in
307     let class_prefix =
308       if class_prefix <> "" then
309         prefix_name class_prefix ^ "_"
310       else
311         ""
312     in
313     let instance_addr = match vopt with
314       None -> ""

```

```

315     | Some(v) ->
316         let var_name =
317             if v.vloop then
318                 "(" ^ prefix_name v.vname ^ " + __cnt__" ^ prefix_name v.
319                 vname ^ ")"
320             else if v.vname = "this" then
321                 "this"
322             else
323                 "&" ^ prefix_name v.vname
324             in
325             "(struct " ^ prefix_name gname ^ " *) " ^
326             var_name ^
327             (if argc > 0 then ", " else ""))
328             in
329             class_prefix ^
330             field_to_c (Fun(fd)) ^ "(" ^ instance_addr ^
331             String.concat ", " (List.map actual_to_c el) ^ ")"
332     | Element(e1, e2) ->
333         let d1, typ = e1 in
334         let d2, _ = e2 in
335         let c_typ = match typ with
336             List_t(x) -> x
337             | _ as x -> x
338             in
339             "*(" ^ id_type_to_c c_typ ^ "*)" " "
340             "list_elem(&" ^ expression_to_c d1 ^ ", " ^ expression_to_c d2 ^ ")
341             )"
342     | Uminus(e) -> let detail, _ = e in "-(" ^ expression_to_c detail ^ ")"
343     | Not(e) -> let detail, _ = e in "!(" ^ expression_to_c detail ^ ")"
344     | Noexpr -> ""
345     | Remove(e) ->
346         let detail, _ = e in
347         expression_to_c detail
348     | Place(e) ->
349         let detail, _ = e in
350         expression_to_c detail

350 let reinitialize_and_push (detail, typ) = match detail with
351     ListLiteral(ll) -> (match ll with
352         EmptyList -> ""
353         | Elems(el, name) ->
354             let e = detail, typ in
355             "new_Sen_list(&" ^ prefix_name name ^
356             ", sizeof(" ^ id_type_to_c typ ^ "));\n" ^
357             push_to_new_list name e ^ "; \n")
358         | _ -> raise (SemError ("non-list passed to reinitialize_and_push"))
359
360 let rec update_ll_expr (detail, typ) = match detail with
361     ListLiteral(ll) -> reinitialize_and_push (detail, typ)
362     (* | Field(fe) -> *)
363     | Binop(e1, op, e2) -> update_ll_expr e1 ^ update_ll_expr e2
364     | Assign(fe, e) -> update_ll_expr e
365     | Call(vd_opt, fd, el) -> String.concat "" (List.map update_ll_expr el)
366     (* | Element(e1, e2) -> *)

```

```

367 (* | Uminus(e) of expression *)
368 | Not(e) -> update_ll_expr e
369 | Remove(e) -> update_ll_expr e
370 | Place(e) -> update_ll_expr e
371 | _ -> ""
372
373 let rec statement_to_c = function
374   Block(scope, slist) ->
375     " " ^ String.concat "\n" (List.map statement_to_c slist)
376   | Expression(e) ->
377     let detail, _ = e in
378     update_ll_expr e ^
379     expression_to_c(detail) ^ ";"
380   | Return(e) ->
381     let detail, _ = e in
382     update_ll_expr e ^
383     "return " ^ expression_to_c detail ^ ";"
384   | Break -> "break;"
385   | Continue -> "continue;"
386   | If(e, s1, e_opt, s2) ->
387     let det, _ = e in
388     update_ll_expr e ^
389     "if (" ^ expression_to_c det ^ " ) {\n" ^
390     statement_to_c s1 ^ "\n} " ^
391     (match e_opt with
392       None -> "else {\n"
393       | Some((detail, _)) -> "else if (" ^ expression_to_c detail ^ " )\n"
394     { \n } ^
395     statement_to_c s2 ^ "}\n"
396   | For(vd, elist, s) ->
397     let n = List.length elist in
398     let counter = "__cnt__" ^ prefix_name vd.vname in
399     String.concat "" (List.map update_ll_expr elist) ^
400     "int " ^ counter ^ " = 0;\n" ^
401     id_type_to_c vd.vtype ^ prefix_name vd.vname ^ "[] = " ^
402     "{" ^ String.concat ", " ^
403     (List.map (fun (det, typ) -> expression_to_c det) elist) ^ "};\n"
404     "
405     "for ( ; " ^ counter ^ " < " ^ string_of_int n ^ "; +" ^ counter ^ ")"
406     { \n } ^
407     statement_to_c s ^ "\n" ^
408     "}\n"
409   | End -> "exit(0);"
410   | Pass(e,s) -> let detaill, _ = s in
411     "CUR_TURN = &" ^ prefix_name (function_call_to_c e) ^
412     ";\n" ^
413     "snt_PLAYER_ON_MOVE = " ^ expression_to_c detaill ^ ";\n" ^
414     "return;\n"
415   | While(e, s) ->
416     let detail, _ = e in
417     "while (" ^ expression_to_c detail ^ ") {\n" ^
418     statement_to_c s ^ "\n} ^ "}\n"

```

```

416 let rec assert_stmt_to_c = function
417   Block(scope, sl) ->
418     " " ^ String.concat "\n" (List.map assert_stmt_to_c sl)
419   | Expression(e) ->
420     let detail, _ = e in
421     "if (!(" ^ expression_to_c detail ^ ")) { return false; }\n"
422   | If(e, s1, e_opt, s2) ->
423     let e, _ = e in
424     "if (" ^ expression_to_c e ^ " ) {\n" ^
425       assert_stmt_to_c s1 ^ "\n} " ^
426       (match e_opt with
427        None -> "else {\n"
428        | Some(expr) ->
429          let det, _ = expr in
430          "else if (" ^ expression_to_c det ^ " ) {\n" ^
431            assert_stmt_to_c s2 ^ "}\n"
432      (* | For(vd, el, s) -> *)
433    | While(e, s) ->
434      let detail, _ = e in
435      "while (" ^ expression_to_c detail ^ " ) {\n" ^
436        assert_stmt_to_c s ^ "\n" ^ "}\n"
437    | _ as s -> statement_to_c s
438
439 let deref_formal v = match v.vtype with
440   Group(gname, _) ->
441     id_type_to_c v.vtype ^ prefix_name v.vname ^ "
442     " = *(" ^ tmp_formal_prefix ^ prefix_name v.vname ^ ");"
443   | List_t(typ) ->
444     id_type_to_c v.vtype ^ prefix_name v.vname ^ "
445     " = *(" ^ tmp_formal_prefix ^ prefix_name v.vname ^ ");"
446   | _ -> ""
447
448 let basic_func_to_c gprefix f =
449   let self_arg =
450     if f.group_method <> "" then
451       "struct " ^ prefix_name f.group_method ^ " *this"
452     else
453       ""
454
455   in
456   let gprefix =
457     if gprefix = "" then "" else gprefix ^ "_"
458   in
459   (id_type_to_c f.ftype) ^ " " ^ gprefix ^ prefix_name f.fname ^ "(" ^
460     self_arg ^
461     (if self_arg <> "" && List.length f.formals > 0 then ", " else "") ^
462     String.concat ", " (List.map (formal_to_c f.f_is_built_in) f.formals) ^
463     " " ^ "\n" ^
464     String.concat "\n" (List.map deref_formal f.formals) ^ "\n" ^
465     String.concat "\n" (List.map var_decl_to_c f.locals) ^ "\n" ^
466     String.concat "\n" (List.map statement_to_c f.body) ^ "\n" ^
467     "}\n"
468
469 let assert_func_to_c gprefix f =
470   let self_arg =

```

```

469     if f.a_group_method <> "" then
470         "struct " ^ prefix_name f.a_group_method ^ " *this"
471     else
472         ""
473     in
474     let gprefix =
475         if gprefix = "" then "" else gprefix ^ "_"
476     in
477     (id_type_to_c Bool) ^ " " ^ gprefix ^ prefix_name f.aname ^ "(" ^
478     self_arg ^
479     (if self_arg <> "" && List.length f.aformals > 0 then ", " else "") ^
480     String.concat ", " (List.map (formal_to_c f.a_is_builtin) f.aformals)
481     ^ ") \n" ^
482     String.concat "\n" (List.map deref_formal f.aformals) ^ "\n" ^
483     String.concat "\n" (List.map var_decl_to_c f.alocals) ^ "\n" ^
484     String.concat "\n" (List.map assert_stmt_to_c f.abody) ^ "\n" ^
485     "\n"
486 let func_decl_to_c gprefix = function
487   BasicFunc(f) -> basic_func_to_c gprefix f
488 | AssertFunc(f) -> assert_func_to_c gprefix f
489
490 let group_decl_to_c g =
491   let c_name = prefix_name g.gname in
492   "struct " ^ c_name ^ "{\n" ^
493   " " ^ String.concat "\n" " " (List.map var_decl_to_c g.attributes) ^ "\n"
494   ^
495   "}" ^ c_name ^ ";" ^ "\n\n" ^
496   String.concat "\n" (List.map (func_decl_to_c c_name) g.methods) ^ "\n"
497
498 let setup_to_c s =
499   let v, f, g = s in
500   String.concat "\n" (List.map var_decl_to_c v) ^ "\n\n" ^
501   String.concat "\n" (List.map group_decl_to_c g) ^ "\n" ^
502   String.concat "\n" (List.map (func_decl_to_c "") f)
503
504 let declare_turn = function
505   BasicFunc(f) ->
506     (id_type_to_c f.ftype) ^ " " ^
507     prefix_name f.fname ^ "(" ^
508     String.concat ", " (List.map var_decl_to_c f.formals) ^ ");"
509 | AssertFunc(f) -> raise (SemError ("Assert function declared in turns:
510
511 let turns_to_c t =
512   String.concat "\n" (List.map declare_turn t) ^ "\n\n" ^
513   String.concat "\n" (List.map (func_decl_to_c "") t)
514
515 let senet_to_c (s, t) =
516   "// @senet_header\n" ^
517   senet_header ^ "\n" ^
518   "// @setup\n" ^
519   setup_to_c s ^ "\n" ^

```

```

520     "://" @turns \n" ^
521     turns_to_c t ^ "\n" ^
522     "://" @senet_footer \n" ^
523     senet_footer
524
525 let translate (program : Types.program) =
526   let outfile = open_out "output.c" in
527   let ctext = senet_to_c program in
528   output_string outfile ctext

```

./examples/chance.snt

```

1 # This is a game of intense skill and unparalleled intrigue
2 # Two players race against clock to reach the other side of a treacherous
   40-cell array
3 # Who will win? Only Lady Luck can tell.
4
5
6 @setup
7 {
8   int left = 0;
9   int right = 39;
10  group myline(Line(40)) {
11    func bool won(int player) {
12      if (this.owns((1-player)*39) == player) {
13        return True;
14      }
15      return False;
16    }
17
18    func str __repr__() {
19      str ret = "|\\n";
20      int i = 39;
21      group Piece temp;
22      while (i>=0) {
23        temp = this.cells[i];
24        ret = "|" + temp.__repr__() + ret;
25        i = i - 1;
26      }
27      return ret;
28    }
29  };
30
31  group Mark(Piece) {
32    func group Mark __init__(str s, int player) {
33      this.s = s;
34      this.owner = player;
35      return this;
36    }
37  };
38
39
40  int N_PLAYERS = 2;
41
42  group myline ml;

```

```

43 }
44 }
45
46
47 @turns
48 {
49
50 func void begin() {
51     group Mark m0 = Mark("X", 0);
52     group Mark m1 = Mark("O", 1);
53     ml = myline();
54     m0 >> ml >> [0];
55     m1 >> ml >> [39];
56     pass(prompt, 0);
57 }
58
59 func void prompt() {
60     int a;
61     int c;
62     int next;
63     group Mark m;
64     int i;
65
66     group Mark m0 = Mark("X", 0);
67     group Mark m1 = Mark("O", 1);
68
69     if (PLAYER_ON_MOVE % 2 == 0) {
70         m = Mark("X", PLAYER_ON_MOVE);
71     } else {
72         m = Mark("O", PLAYER_ON_MOVE);
73     }
74
75     print("\n"); print(ml); print("\n");
76
77     # players input moves by guessing even or odd
78     print("PLAYER "); print(PLAYER_ON_MOVE); print(": ");
79     print("Even (0) or odd (1)?\n");
80     a = stoi(read(1));
81     c = rand();
82     clear_input();
83     if (c % 2 == a % 2) {
84         print("\n Well guessed!! You are truly skilled!! \n");
85         if (PLAYER_ON_MOVE == 0) {
86             next = (left + (rand() % 6));
87             if (next > 39) {next = 39;}
88         } else {
89             next = (right - (rand() % 6));
90             if (next < 0) {next = 0;}
91         }
92         if ((next == 39 and PLAYER_ON_MOVE == 0) or (next == 0 and
93             PLAYER_ON_MOVE == 1)) {
94
95             pass(winner, PLAYER_ON_MOVE);
96         } else {

```

```

96     if (ml.owns(next) == 1-PLAYER_ON_MOVE) {
97         if (PLAYER_ON_MOVE == 1) {
98             ml << [left];
99             left = left -1;
100            m0 >> ml >> [left];
101        } else {
102            ml << [right];
103            right = right + 1;
104            m1 >> ml >> [right];
105        }
106    }
107    if (PLAYER_ON_MOVE == 0) {
108        ml << [left];
109        left = next;
110        m >> ml >> [next];
111    } else {
112        ml << [right];
113        right = next;
114        m >> ml >> [right];
115    }
116}
117} else {
118    print("\n Too bad, you guessed wrong, better luck next time!!\n");
119}
120 pass(prompt, (PLAYER_ON_MOVE + 1) % N_PLAYERS);
121
122}
123}
124
125 func void winner() {
126     print("\n"); print(ml); print("\n");
127     print("Player "); print(PLAYER_ON_MOVE); print(" wins.\n");
128     print("Congratulations!\n");
129     end;
130 }
131
132}

```

**./examples/checkers.snt**

```

1 @setup
2 {
3 group b (Boards.Rect(3,3)) {
4
5
6
7 bool checkWin(int player) {
8
9     int other_player=0;
10    if(player==0)
11    {
12        other_player=1;
13    }
14    int i=1;
15    while(i<64)

```

```

16     {
17         if(this.owns(i))==other_player)
18     {
19         return false;
20     }
21     i=i+1;
22 }
23
24     return true;
25 }
26
27 }
28 # owns checks the owner of a piece at an index,
29             # and returns -1 if the space is empty
30 }
31     return False;
32 }
33
34 void init_board()
35 { Mark mark_red_1 = mark("Red_1");
36     Mark mark_red_2 = mark("Red_2");
37     Mark mark_red_3 = mark("Red_3");
38     Mark mark_red_4 = mark("Red_4");
39     Mark mark_red_5 = mark("Red_5");
40     Mark mark_red_6 = mark("Red_6");
41     Mark mark_red_7 = mark("Red_7");
42     Mark mark_red_8 = mark("Red_8");
43     Mark mark_red_9 = mark("Red_10");
44     Mark mark_red_11 = mark("Red_11");
45     Mark mark_red_12 = mark("Red_12");
46
47     Mark mark_black_1 = mark("Black_1");
48     Mark mark_black_2 = mark("Black_2");
49     Mark mark_black_3 = mark("Black_3");
50     Mark mark_black_4 = mark("Black_4");
51     Mark mark_black_5 = mark("Black_5");
52     Mark mark_black_6 = mark("Black_6");
53     Mark mark_black_7 = mark("Black_7");
54     Mark mark_black_8 = mark("Black_8");
55     Mark mark_black_9 = mark("Black_9");
56     Mark mark_black_10 = mark("Black_10");
57     Mark mark_black_11 = mark("Black_11");
58     Mark mark_black_12 = mark("Black_12");
59
60     this.place(mark_red_1,2);
61     this.place(mark_red_2,4);
62     this.place(mark_red_3,6);
63     this.place(mark_red_4,8);
64         this.place(mark_red_5,9);
65     this.place(mark_red_6,11);
66     this.place(mark_red_7,13);
67     this.place(mark_red_8,15);
68         this.place(mark_red_9,18);
69     this.place(mark_red_10,20);

```

```

70     this.place(mark_red_11,22);
71     this.place(mark_red_12,24);
72
73         this.place(mark_black_1,41);
74     this.place(mark_black_2,43);
75     this.place(mark_black_3,45);
76     this.place(mark_black_4,47);
77         this.place(mark_black_5,50);
78     this.place(mark_black_6,52);
79     this.place(mark_black_7,54);
80     this.place(mark_black_8,56);
81         this.place(mark_black_9,57);
82     this.place(mark_black_10,59);
83     this.place(mark_black_11,61);
84     this.place(mark_black_12,63);
85
86 }
87
88     bool validatePiece(int c1, int c2,int player)
89 {
90     bool valid=false;
91     int index=toi([c1,c2]);
92     {
93         if(this.owns(index)==player)
94             {valid=true;
95             }
96     }
97     return valid;
98 }
99
100    bool validateCell(int p1,int p2,cur_p1,cur_p2,int player)
101    {int index=toi([c1,c2]);
102     int otherplayer=0;
103
104         if(player==0)
105     {
106             otherplayer=1;
107             if(p1<cur_p1+1)
108                 return false;
109             else if(p2<cur_p2+1)
110                 return false;
111             else if(this.owns(index)==player || this.owns(index==otherplayer))
112                 {return false;}
113         }
114         else
115     {
116             if(p1>cur_p1-1)
117                 return false;
118             else if(p2>cur_p2-1)
119                 return false;
120             else if(this.owns(index)==player || this.owns(index==otherplayer))
121                 {return false;}
122     }
123 }
```

```

124         this.remove(index);
125         return true;
126
127     }
128
129
130
131
132     bool won (int player) { # checks if the player won
133         if three_in_a_row(player)
134             return True;
135         return False;
136     }
137
138     bool draw() {
139         if this.full() {
140             return True;
141         }
142         return False;
143     }
144
145     assert draw() {
146         this.full();
147     }
148 };
149
150
151
152 group Mark (Piece) { # inherits from Piece
153
154     bool isKing;
155
156
157         func group Mark __init__(str s) {
158             this.s=s;
159             this.fixed = false;
160             isKing = false;
161             return this;
162         }
163
164     }
165 };
166
167
168     N_PLAYERS = 2; # number of players, this default of 1
169
170     int gamestep=0;
171
172
173 @turns
174 {
175     begin () {
176         gamestep=gamestep+1;
177         bool validpiece=false;

```

```

178 bool validcell=false;
179 int current_player=gamestep % 2;
180 # this is basically just "while True" with only 1 phase
181     # players input moves by typing coordinates, e.g. "11" or "02"
182 while(validPiece==false)
183 {
184 print( Input coordinates of the piece to you want to use for the turn
185     )
186 print( in i.e. \ 22\ or \ 10\ .\n ); # prompts the players
187     int p1 = stoi(read(1)); # reads one character and converts it to an
188     int
189     int p2 = stoi(read(1));
190     validpiece=b.validatePiece(p1,p2,current_player);
191 }
192
193 while(validcell==false)
194 {
195 print( Input coordinates of the cell you want your piece to move to )
196 print( in i.e. \ 22\ or \ 10\ .\n ); # prompts the players
197     int c1 = stoi(read(1)); # reads one character and converts it to an
198     int
199     int c2 = stoi(read(1));
200     validcell=b.validateCell(c1,c2,p1,p2,current_player);
201 }
202
203     if (Mark >> b [a,c]) {
204         if (b.won(current_player)) {
205             print("Player " + itos(current_player) + " wins.\n");
206             print("Congratulations!");
207         }
208     }
209 }
210 }
```

./examples/checkers\_updated.snt

```

1 @setup
2 {
3
4 list[group line] victory_conds;
5
6 group line(Object) {
7     list[int] loci;
8
9     func group line __init__(list[int] l) {
10         this.loci = l;
11         return this;
12     }
13 };
14
15 group Mark(Piece) {
16     func group Mark __init__(str s, int player) {
```

```

17     this.s = s;
18     this.owner = player;
19     return this;
20 }
21 };
22
23 group ttb(Rect(8, 8))
24 {
25
26     # NOTE: Need to add a way to call INIT_CELLS since this segfaults
27     # func group ttb __init__(list[group line] vc) {
28     #     group Piece x = Piece();
29     #     bool NO = false;
30     #     this.x = 3; this.y = 3;
31     #     this.victory_conds = vc;
32     #     this.cells = [x, x, x, x, x, x, x, x, x];
33     #     this.occupied = [NO, NO, NO, NO, NO, NO, NO, NO, NO];
34     #     return this;
35     # }
36
37     assert owner_of_cell(int index, int player) {
38         this.owns(index) == player;
39     }
40
41     func bool three_in_a_row(int player) {
42         int i = 0;
43         int other_player=0;
44         bool valid=False;
45         if(player == 0)
46         {
47             other_player=1;
48         }
49
50         while (i < 64) {
51             print("DEBUG: three_in_a_row, i = "); print(i); print("\n");
52             valid= False;
53             if (this.owner_of_cell(i, other_player)) {
54                 return False;
55             }
56             i = i + 1;
57         }
58         return True;
59     }
60
61     func bool won(int player) {
62         bool win=False;
63         if (this.three_in_a_row(player)) {
64             return True;
65         }
66         return False;
67     }
68 }
69
70     func void draw()

```

```

71     {
72         print ("Drawn");
73     }
74
75     func str __str_of_row(int row) {
76         group Piece p1; group Piece p2; group Piece p3; group Piece p4;
77         group Piece p5; group Piece p6; group Piece p7; group Piece p8;
78         str ret;
79         # print("DEUBG: start of __str_of_row with row = "); print(row);
80         print("\n");
81         # print("DEBUG: print [0, row] = "); print([0, row]); print("\n");
82         p1 = this.cells[this.toi([0, row])];
83         # print("DEBUG: l = this.cells call ok\n");
84         p2 = this.cells[this.toi([1, row])];
85         p3 = this.cells[this.toi([2, row])];
86         p4 = this.cells[this.toi([3, row])];
87         p5 = this.cells[this.toi([4, row])];
88         p6 = this.cells[this.toi([5, row])];
89         p7 = this.cells[this.toi([6, row])];
90         p8 = this.cells[this.toi([7, row])];
91         ret = "[" + p1.__repr__() + ", " + p2.__repr__() + ", " + p3.
92         __repr__() + ", " + p4.__repr__() + ", " + p5.__repr__() + ", " + p6.
93         __repr__() + ", " + p7.__repr__() + ", " + p8.__repr__() + "]\n";
94         return ret;
95     }
96
97     func str __repr__() {
98         return this.__str_of_row(0) + this.__str_of_row(1) + this.
99         __str_of_row(2)+this.__str_of_row(3)+this.__str_of_row(4)+this.
100        __str_of_row(5)+this.__str_of_row(6)+this.__str_of_row(7);
101    }
102
103    func bool validatePiece(int index,int player)
104    {
105        bool valid=False;
106        print("index");
107        if(this.owns(index)==player)
108            {valid=True;
109             }
110        }
111        return valid;
112    }
113
114    func bool validateCell(int p1, int p2, int cur_p1, int cur_p2, int
115    player)
116        {int index=this.toi([p1,p2]);
117         int otherplayer=0;
118         int temp1;
119         int temp2;
120         int tempindex;
121         if(player==0)
122         {
123             otherplayer=1;

```

```

118     if(p1-cur_p1 != p2-cur_p2)
119     {
120         return False;
121     }
122     if(p1<cur_p1+1)
123     {
124         return False;
125     }
126
127     if(p2<cur_p2+1)
128     {
129         return False;
130     }
131
132     if(this.owns(index)==player or this.owns(index)==otherplayer)
133     {
134         return False;
135     }
136     if(p1-cur_p1>1)
137     {
138         temp1=cur_p1+1;
139         temp2=cur_p2+1;
140         while(temp1<p1)
141         {
142             tempindex=this.toi([temp1,temp2]);
143             if(this.owns(tempindex)==otherplayer)
144             {
145                 temp1=temp1+1;
146                 temp2=temp2+1;
147                 continue;
148             }
149         }
150     }
151
152 }
153 else
154 {
155     if(cur_p1-p1 != cur_p2-p2)
156     {
157         return False;
158     }
159     if(p1>cur_p1-1)
160     {
161         return False;
162     }
163     if(p2>cur_p2-1)
164     {
165         return False;
166     }
167     if(this.owns(index) == player or this.owns(index)==otherplayer)
168     {
169         return False;
170     }
171     if(cur_p1-p1>1)
172     {
173         temp1=cur_p1-1;
174         temp2=cur_p2-1;
175         while(temp1>p1)
176         {
177             tempindex=this.toi([temp1,temp2]);
178             if(this.owns(tempindex)==otherplayer)
179             {
180                 temp1=temp1-1;
181             }
182         }
183     }
184 }
```

```

172         temp2=temp2-1;
173         continue;
174     }
175     else
176     {
177         return False;
178     }
179 }
180 }
181
182 }
183 }
184 }
185 this.remove(index);
186 return True;
187
188
189 }
190 }
191
192 func void removePieces(int p1,int p2, int cur_p1,int cur_p2)
193 {int temp1;
194     int temp2;
195     int tempindex;
196 if(cur_p1-p1>1)
197
198 {temp1=cur_p1-1;
199     temp2=cur_p2-1;
200     while(temp1>p1)
201     {    tempindex=this.toi([temp1,temp2]);
202         this.remove(tempindex);
203     }
204
205
206
207     }
208     if(p1-cur_p1>1)
209     {temp1=cur_p1+1;
210         temp2=cur_p2+1;
211         while(temp1<p1)
212         {    tempindex=this.toi([temp1,temp2]);
213
214             this.remove(tempindex);
215         }
216     }
217 }
218
219 func void init_board()
220 {int player=0;
221     int otherplayer=1;
222
223 group Mark mark_red_1 = Mark("R",player);
224     group Mark mark_red_2 = Mark("R",player);
225         group Mark mark_red_3 = Mark("R",player);

```

```

226     group Mark mark_red_4 = Mark("R",player);
227         group Mark mark_red_5 = Mark("R",player);
228         group Mark mark_red_6 = Mark("R",player);
229     group Mark mark_red_7 = Mark("R",player);
230         group Mark mark_red_8 = Mark("R",player);
231             group Mark mark_red_9 = Mark("R",player);
232             group Mark mark_red_10 = Mark("R",player);
233
234             group Mark mark_red_11 = Mark("R",player);
235     group Mark mark_red_12 = Mark("R",player);
236
237     group Mark mark_black_1 = Mark("B",otherplayer);
238         group Mark mark_black_2 = Mark("B",otherplayer);
239         group Mark mark_black_3 = Mark("B",otherplayer);
240             group Mark mark_black_4 = Mark("B",otherplayer);
241                 group Mark mark_black_5 = Mark("B",otherplayer);
242             group Mark mark_black_6 = Mark("B",otherplayer);
243                 group Mark mark_black_7 = Mark("B",otherplayer);
244             group Mark mark_black_8 = Mark("B",otherplayer);
245                 group Mark mark_black_9 = Mark("B",otherplayer);
246             group Mark mark_black_10 = Mark("B",otherplayer);
247                 group Mark mark_black_11 = Mark("B",otherplayer);
248             group Mark mark_black_12 = Mark("B",otherplayer);
249
250     this.place(mark_red_1,2);
251     this.place(mark_red_2,4);
252     this.place(mark_red_3,6);
253     this.place(mark_red_4,8);
254         this.place(mark_red_5,9);
255     this.place(mark_red_6,11);
256     this.place(mark_red_7,13);
257     this.place(mark_red_8,15);
258         this.place(mark_red_9,18);
259     this.place(mark_red_10,20);
260     this.place(mark_red_11,22);
261     this.place(mark_red_12,24);
262
263         this.place(mark_black_1,41);
264     this.place(mark_black_2,43);
265     this.place(mark_black_3,45);
266     this.place(mark_black_4,47);
267         this.place(mark_black_5,50);
268     this.place(mark_black_6,52);
269     this.place(mark_black_7,54);
270     this.place(mark_black_8,56);
271         this.place(mark_black_9,57);
272     this.place(mark_black_10,59);
273     this.place(mark_black_11,61);
274     this.place(mark_black_12,63);
275
276 }
277
278
279 };

```

```

280
281
282
283
284
285
286 int N_PLAYERS = 2;
287
288 group ttb b;
289
290
291 }
292
293
294
295 @turns
296 {
297
298 func void begin() {
299
300     # list[group line] victory_conds;
301
302     b = ttb();
303     b.init_board();
304
305     # print("DEBUG: v7 ok\n");
306
307
308     # print("DEBUG: built victory_conds\n");
309
310     # b = ttb();
311
312     # print("DEBUG: built b\n");
313
314     pass(prompt, 0);
315 }
316
317 func void prompt() {
318     int a;
319     int c;
320     int x;
321     int y;
322     int index;
323     bool valid_piece=False;
324     bool valid_cell=False;
325     group Mark m;
326     int i;
327
328     if (PLAYER_ON_MOVE % 2 == 0) {
329         m = Mark("R", PLAYER_ON_MOVE);
330     } else {
331         m = Mark("B", PLAYER_ON_MOVE);
332     }
333 }
```

```

334 # print("DEBUG: start of prompt()\n");
335
336 print("\n"); print(b); print("\n");
337
338 # print("DEBUG: printed b\n");
339
340 # players input moves by typing coordinates, e.g. "11" or "02"
341
342 print("PLAYER "); print(PLAYER_ON_MOVE); print(": ");
343 while(not valid_piece)
344 {
345     print("Input coordinates of the piece to be moved ");
346     print("in i.e. \"22\" or \"10\".\n");
347     a = stoi(read(1));
348     c = stoi(read(1));
349     index=b.toi([a,c]);
350     valid_piece=b.validatePiece(index,PLAYER_ON_MOVE);
351 }
352
353 while(not valid_cell)
354 {
355     print("Input coordinates of the piece to be moved ");
356     print("in i.e. \"22\" or \"10\".\n");
357     x= stoi(read(1));
358     y = stoi(read(1));
359     index=b.toi([x,y]);
360     valid_cell=b.validateCell(x,y,a,c,PLAYER_ON_MOVE);
361 }
362
363 clear_input();
364 # print("index of input: ["); print(a); print(", "); print(c); print("]");
365 # print(b.toi([a, c])); print("\n");
366
367 if (m >> b >> [x, y]) {
368     b.removePieces(x,y,a,c);
369     # PLAYER_ON_MOVE is the index of the player
370     if (b.won(PLAYER_ON_MOVE)) {
371         pass(winner, PLAYER_ON_MOVE);
372     }
373 }
374 else {
375     # A piece is already at [a, c]
376     print("Cannot place a mark there, try again.\n\n");
377     pass(prompt, PLAYER_ON_MOVE);
378 }
379 # if the move was legal, went through successfully,
380 # and the game is not over, pass the turn to the next player
381 pass(prompt, (PLAYER_ON_MOVE + 1) % N_PLAYERS);
382 }
383
384 func void winner() {
385
386     print("\n"); print(b); print("\n");

```

```

387     print("Player "); print(PLAYER_ON_MOVE); print(" wins.\n");
388     print("Congratulations!\n");
389     end;
390 }
391
392 func void nowinner() {
393     print("\n"); print(b); print("\n");
394     print("Game ends in a draw.\n");
395     end;
396 }
397
398 }

```

**./examples/tictactoe.snt**

```

1 @setup
2 {
3
4 list[group line] victory_conds;
5
6 group line(Object) {
7     list[int] loci;
8
9     func group line __init__(list[int] l) {
10         this.loci = l;
11         return this;
12     }
13 };
14
15 group ttb(Rect(3, 3)) {
16
17     # NOTE: Need to add a way to call INIT_CELLS since this segfaults
18     # func group ttb __init__(list[group line] vc) {
19     #     group Piece x = Piece();
20     #     bool NO = false;
21     #     this.x = 3; this.y = 3;
22     #     this.victory_conds = vc;
23     #     this.cells = [x, x, x, x, x, x, x, x, x];
24     #     this.occupied = [NO, NO, NO, NO, NO, NO, NO, NO, NO];
25     #     return this;
26     # }
27
28     assert owner_of_line(group line l, int player) {
29         this.owns(l.loci[0]) == player;
30         this.owns(l.loci[1]) == player;
31         this.owns(l.loci[2]) == player;
32     }
33
34     func bool three_in_a_row(int player) {
35         int i = 0;
36         group line l;
37         while (i < 8) {
38             l = victory_conds[i];
39             # print("DEBUG: three_in_a_row, i = "); print(i); print("\n");

```

```

41         if (this.owner_of_line(l, player)) {
42             return True;
43         }
44         i = i + 1;
45     }
46     return False;
47 }

48
49 func bool won(int player) {
50     if (this.three_in_a_row(player)) {
51         return True;
52     }
53     return False;
54 }

55
56 assert draw() {
57     this.full();
58 }

59
60 func str __str_of_row(int row) {
61     group Piece l; group Piece m; group Piece r;
62     str ret;
63     # print("DEUBG: start of __str_of_row with row = "); print(row);
print("\n");
64     # print("DEBUG: print [0, row] = "); print([0, row]); print("\n");
65     l = this.cells[this.toi([0, row])];
66     # print("DEBUG: l = this.cells call ok\n");
67     m = this.cells[this.toi([1, row])];
68     r = this.cells[this.toi([2, row])];
69     ret = "[" + l.__repr__() + ", " + m.__repr__() + ", " + r.__repr__();
() + "]\n";
70     return ret;
71 }

72
73 func str __repr__() {
74     return this.__str_of_row(0) + this.__str_of_row(1) + this.
__str_of_row(2);
75 }
76};

77
78
79
80 group Mark(Piece) {
81     func group Mark __init__(str s, int player) {
82         this.s = s;
83         this.owner = player;
84         return this;
85     }
86 };
87
88
89 int N_PLAYERS = 2;
90
91 group ttb b;

```

```

92 }
93 }
94
95
96 @turns
97 {
98
99 func void begin() {
100     group line v0; group line v1; group line v2; group line v3;
101     group line v4; group line v5; group line v6; group line v7;
102     int x; int y; int z;
103     # list[group line] victory_conds;
104
105     b = ttb();
106
107     x = b.toi([0, 0]); y = b.toi([1, 0]); z = b.toi([2, 0]);
108     v0 = line([x, y, z]);
109     x = b.toi([0, 1]); y = b.toi([1, 1]); z = b.toi([2, 1]);
110     v1 = line([x, y, z]);
111     x = b.toi([0, 2]); y = b.toi([1, 2]); z = b.toi([2, 2]);
112     v2 = line([x, y, z]);
113     x = b.toi([0, 0]); y = b.toi([0, 1]); z = b.toi([0, 2]);
114     v3 = line([x, y, z]);
115     x = b.toi([1, 0]); y = b.toi([1, 1]); z = b.toi([1, 2]);
116     v4 = line([x, y, z]);
117     x = b.toi([2, 0]); y = b.toi([2, 1]); z = b.toi([2, 2]);
118     v5 = line([x, y, z]);
119     x = b.toi([0, 0]); y = b.toi([1, 1]); z = b.toi([2, 2]);
120     v6 = line([x, y, z]);
121     x = b.toi([0, 2]); y = b.toi([1, 1]); z = b.toi([2, 0]);
122     v7 = line([x, y, z]);
123
124     # print("DEBUG: v7 ok\n");
125
126     victory_conds = [v0, v1, v2, v3, v4, v5, v6, v7];
127
128     # print("DEBUG: built victory_conds\n");
129
130     # b = ttb();
131
132     # print("DEBUG: built b\n");
133
134     pass(prompt, 0);
135 }
136
137 func void prompt() {
138     int a;
139     int c;
140     group Mark m;
141     int i;
142
143     if (PLAYER_ON_MOVE % 2 == 0) {
144         m = Mark("X", PLAYER_ON_MOVE);
145     } else {

```

```

146     m = Mark("0", PLAYER_ON_MOVE);
147 }
148
149 # print("DEBUG: start of prompt()\n");
150
151 print("\n"); print(b); print("\n");
152
153 # print("DEBUG: printed b\n");
154
155 # players input moves by typing coordinates, e.g. "11" or "02"
156 print("PLAYER "); print(PLAYER_ON_MOVE); print(": ");
157 print("Input coordinates of square to place ");
158 print("in i.e. \"22\" or \"10\".\n");
159 a = stoi(read(1));
160 c = stoi(read(1));
161 clear_input();
162 # print("index of input: ["); print(a); print(", "); print(c); print("]");
163 # print(b.toi([a, c])); print("\n");
164
165 if (m >> b >> [a, c]) {
166
167     # PLAYER_ON_MOVE is the index of the player
168     if (b.won(PLAYER_ON_MOVE)) {
169         pass(winner, PLAYER_ON_MOVE);
170     }
171     if (b.draw()) {
172         pass(nowinner, PLAYER_ON_MOVE);
173     }
174 } else {
175     # A piece is already at [a, c]
176     print("Cannot place a mark there, try again.\n\n");
177     pass(prompt, PLAYER_ON_MOVE);
178 }
179 # if the move was legal, went through successfully,
180 # and the game is not over, pass the turn to the next player
181 pass(prompt, (PLAYER_ON_MOVE + 1) % N_PLAYERS);
182 }
183
184 func void winner() {
185     print("\n"); print(b); print("\n");
186     print("Player "); print(PLAYER_ON_MOVE); print(" wins.\n");
187     print("Congratulations!\n");
188     end;
189 }
190
191 func void nowinner() {
192     print("\n"); print(b); print("\n");
193     print("Game ends in a draw.\n");
194     end;
195 }
196
197 }

```

./Makefile

```
1 OBJS = ast.cmo parser.cmo scanner.cmo types.cmo stdlib.cmo sast.cmo cast.cmo \
2           compile.cmo senet.cmo
3 VERBOSE_YACC = -v
4
5 # Choose one
6 # YACC = ocamlyacc
7 YACC = menhir --explain
8
9 senet : $(OBJS)
10    ocamlc -o senet $(OBJS)
11
12 scanner.ml : scanner.mll
13    ocamllex scanner.mll
14
15 parser.ml parser.mli : parser.mly
16    $(YACC) $(VERBOSE_YACC) parser.mly
17
18 %.cmo : %.ml
19    ocamlc -c $<
20
21 %.cmi : %.mli
22    ocamlc -c $<
23
24 .PHONY : clean
25 clean :
26    @rm -f senet parser.ml parser.mli scanner.ml \
27    *.cmo *.cmi *.out *.diff *.output *.conflicts *.automaton \
28    output.c testall.log
29
30 # Generated by ocamldep *.ml *.mli *.mly *.mll
31 # see http://caml.inria.fr/pub/docs/manual-ocaml/depend.html
32 ast.cmo :
33 ast.cmx :
34 cast.cmo : types.cmo sast.cmo ast.cmo
35 cast.cmx : types.cmx sast.cmx ast.cmx
36 compile.cmo : types.cmo sast.cmo ast.cmo
37 compile.cmx : types.cmx sast.cmx ast.cmx
38 parser.cmo : ast.cmo parser.cmi
39 parser.cmx : ast.cmx parser.cmi
40 sast.cmo : types.cmo stdlib.cmo ast.cmo
41 sast.cmx : types.cmx stdlib.cmx ast.cmx
42 scanner.cmo : parser.cmi
43 scanner.cmx : parser.cmx
44 senet.cmo : scanner.cmo sast.cmo parser.cmi compile.cmo cast.cmo ast.cmo
45 senet.cmx : scanner.cmx sast.cmx parser.cmx compile.cmx cast.cmx ast.cmx
46 stdlib.cmo : types.cmo
47 stdlib.cmx : types.cmx
48 types.cmo :
49 types.cmx :
50 parser.cmi : ast.cmo
```

./parser.mly

```

1 %{
2     open Ast
3     let fst_of_three (x, _, _) = x
4     let snd_of_three (_, x, _) = x
5     let trd_of_three (_, _, x) = x %
6
7 %token SEMI LPAREN RPAREN LBRACE RBRACE COMMA DOT
8 %token LBRACKET RBRACKET
9 %token PLUS MINUS TIMES DIVIDE ASSIGN MOD
10 %token AND OR NOT
11 %token EQ NEQ LT LEQ GT GEQ
12 %token RETURN IF ELSE FOR WHILE INT
13 %token <int> INTLITERAL
14 %token <string> STRLITERAL
15 %token <string> ID
16 %token EOF
17 %token IN
18 %token BREAK CONTINUE PASS END
19 %token ELIF
20 %token TRUE FALSE NONE
21 %token STR BOOL VOID LIST GROUP
22 %token ASSERT
23 %token REMOVE PLACE
24 %token SETUP TURNS FUNC
25 %token THIS
26 /* lowest precedence */
27
28 /* %nonassoc NOELSE */
29 /* %nonassoc ELSE */
30 %right ASSIGN
31 %left OR
32 %left AND
33 %nonassoc NOT
34 %left EQ NEQ
35 %left LT GT LEQ GEQ
36 %left PLUS MINUS
37 %left TIMES DIVIDE MOD
38 %nonassoc UMINUS
39 %nonassoc LBRACKET
40 /* %left DOT */
41
42 /* highest precedence */
43
44 %start program
45 %type <Ast.program> program
46
47 %%
48
49 program:
50     twoparts EOF { $1 }
51
52 twoparts:
53     SETUP LBRACE decls RBRACE TURNS LBRACE fdecl_list RBRACE
54     {(List.rev (fst_of_three $3),

```

```

55     List.rev (snd_of_three $3),
56     List.rev (trd_of_three $3)),
57     List.rev $7}

58 decls:
59   /* nothing */ { [], [], [] }
60   | decls vdecl { ($2 :: fst_of_three $1),
61                  snd_of_three $1,
62                  trd_of_three $1 }
63   | decls fdecl { fst_of_three $1,
64                  ($2 :: snd_of_three $1),
65                  trd_of_three $1 }
66   | decls gdecl { fst_of_three $1,
67                  snd_of_three $1,
68                  ($2 :: trd_of_three $1) }

69
70
71 gdecl:
72   GROUP ID LPAREN extend_opt RPAREN LBRACE vdecl_list fdecl_list RBRACE
73   SEMI
74   { { fname = $2;
75     extends = $4;
76     par_actualls = None;
77     attributes = List.rev $7;
78     methods = List.rev $8 } }
79   | GROUP ID LPAREN field_expr LPAREN actuals_opt RPAREN RPAREN LBRACE
80   vdecl_list fdecl_list RBRACE SEMI
81   { { fname = $2;
82     extends = Some($4);
83     par_actualls = Some($6);
84     attributes = List.rev $10;
85     methods = List.rev $11 } }

86 extend_opt:
87   /* nothing */ { None }
88   | field_expr { Some($1) }

89 fdecl_list:
90   /* nothing */ { [] }
91   | fdecl_list fdecl { $2 :: $1 }

92
93 fdecl:
94   FUNC type_id ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list
95   RBRACE
96   { BasicFunc({ ftype = $2;
97     fname = $3;
98     formals = $5;
99     locals = List.rev $8;
100    body = List.rev $9 }) }
101  | ASSERT ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
102  { AssertFunc({ fname = $2;
103    formals = $4;
104    locals = List.rev $7;
105    body = List.rev $8 }) }

```

```

106 formal_opts:
107   /* nothing */ { [] }
108   | formal_list { List.rev $1 }
109
110 formal_list:
111   type_id ID { [ { vtype = $1; vname = $2; vinit =
112     NoInit } ] }
113   | formal_list COMMA type_id ID { [ { vtype = $3; vname = $4; vinit =
114     NoInit } :: $1 ] }
115
116 vdecl_list:
117   /* nothing */ { [] }
118   | vdecl_list vdecl { $2 :: $1 }
119
120 vdecl:
121   type_id ID SEMI
122   { { vtype = $1;
123     vname = $2;
124     vinit = NoInit } }
125   | type_id ID ASSIGN expr SEMI
126   { { vtype = $1;
127     vname = $2;
128     vinit = ExprInit($4) } }
129
130 type_id:
131   INT { Int }
132   | BOOL { Bool }
133   | STR { Str }
134   | VOID { Void }
135   | LIST LBRACKET type_id RBRACKET { List($3) }
136   | GROUP ID { Group($2) }
137
138 stmt_list:
139   /* nothing */ { [] }
140   | stmt_list stmt { $2 :: $1 }
141
142 stmt_list_req:
143   stmt { [$1] }
144   | stmt_list_req stmt { $2 :: $1 }
145
146 stmt:
147   expr SEMI { Expr($1) }
148   | RETURN expr SEMI { Return($2) }
149   | LBRACE stmt_list RBRACE { Block(List.rev $2) }
150   | IF LPAREN expr RPAREN LBRACE stmt_list_req RBRACE /* %prec NOELSE */
151   { If($3, Block(List.rev $6), None, Block([])) }
152   | IF LPAREN expr RPAREN LBRACE stmt_list_req RBRACE ELIF expr LBRACE
153   { If($3, Block(List.rev $6), Some($9), Block(List.rev $11)) }
154   | FOR LPAREN type_id ID IN LBRACE expr_list RBRACE RPAREN LBRACE
155   { If($3, Block(List.rev $6), None, Block(List.rev $10)) }

```

```

155     { For({ vtype = $3;
156             vname = $4;
157             vinit = NoInit },
158             List.rev $7,
159             Block(List.rev $11)) }
160 WHILE LPAREN expr RPAREN LBRACE stmt_list_req RBRACE
161   { While($3, Block(List.rev $6)) }
162 BREAK SEMI      { Break}
163 CONTINUE SEMI { Continue }
164 END SEMI { End }
165 PASS LPAREN ID COMMA expr RPAREN SEMI { Pass($3, $5) }

166
167

168 expr:
169   INTLITERAL      { IntLiteral($1) }
170   STRLITERAL      { StrLiteral($1) }
171   NONE            { VoidLiteral }
172   bool_lit        { BoolLiteral($1) }
173   list_lit        { ListLiteral($1) }
174   field_expr       { Field($1) }
175   expr PLUS expr { Binop($1, Add,    $3) }
176   expr MINUS expr { Binop($1, Sub,    $3) }
177   expr TIMES expr { Binop($1, Mult,   $3) }
178   expr DIVIDE expr { Binop($1, Div,    $3) }
179   expr MOD expr { Binop($1, Mod,    $3) }
180   expr EQ expr { Binop($1, Equal,  $3) }
181   expr NEQ expr { Binop($1, Neq,    $3) }
182   expr LT expr { Binop($1, Less,   $3) }
183   expr LEQ expr { Binop($1, Leq,    $3) }
184   expr GT expr { Binop($1, Greater, $3) }
185   expr GEQ expr { Binop($1, Geq,    $3) }
186   expr AND expr { Binop($1, And,    $3) }
187   expr OR  expr { Binop($1, Or,     $3) }
188   field_expr ASSIGN expr { Assign($1, $3) }
189   field_expr LPAREN actuals_opt RPAREN { Call($1, $3) }
190   expr LBRACKET expr RBRACKET { Element($1, $3) }
191   LPAREN expr_opt RPAREN { $2 }
192   MINUS expr %prec UMINUS { Uminus($2) }
193   NOT expr { Not($2) }
194   field_expr PLACE field_expr PLACE list_lit { Place($1, $3, $5) }
195   field_expr REMOVE list_lit { Remove($1, $3) }

196
197 expr_opt:
198   /* nothing */ { Noexpr }
199   | expr          { $1 }

200
201

202 field_expr:
203   ID              { Id($1) }
204   | THIS           { This }
205   | field_expr DOT ID { FieldCall($1, $3) }

206
207 expr_list:
208   expr          { [$1] }

```

```

209 | expr_list COMMA expr { $3 :: $1 }
210
211 actuals_opt:
212   /* nothing */ { [] }
213   | actuals_list { List.rev $1 }
214
215 actuals_list:
216   expr { [$1] }
217   | actuals_list COMMA expr { $3 :: $1 }
218
219 list_lit:
220   LBRACKET RBRACKET { EmptyList }
221   | LBRACKET expr_list RBRACKET { Elems(List.rev $2) }
222
223 bool_lit:
224   TRUE { True }
225   | FALSE { False }

```

./sast.ml

```

1 open Types
2
3 exception SemError of string
4
5 type counter = {
6   mutable i : int
7 }
8
9 let count = { i = 0 }
10
11 let rec string_of_t = function
12   Int -> "int"
13   | Bool -> "bool"
14   | Str -> "str"
15   | Void -> "void"
16   | List_t(vt) ->
17     "list[" ^ string_of_t vt ^ "]"
18   | Group(s, _) -> "group " ^ s
19
20 let rec id_type_to_t = function
21   Ast.Int -> Int
22   | Ast.Bool -> Bool
23   | Ast.Str -> Str
24   | Ast.Void -> Void
25   | Ast.List(id_typ) -> List_t(id_type_to_t id_typ)
26   | Ast.Group(s) -> Group(s, None)
27
28 let rec find_variable (scope : symbol_table) name =
29   try
30     List.find (fun v -> v.vname = name) scope.variables
31   with Not_found ->
32     match scope.parent with
33       Some(parent) -> find_variable parent name
34     | _ -> raise Not_found
35

```

```

36 let rec find_function (scope : symbol_table) name =
37   try
38     List.find (fun f -> match f with
39                 BasicFunc(x) -> x.fname = name
40               | AssertFunc(x) -> x.aname = name) scope.functions
41   with Not_found ->
42     match scope.parent with
43       Some(parent) -> find_function parent name
44     | _ -> raise Not_found
45
46 let rec find_group (scope : symbol_table) name =
47   try
48     List.find (fun g -> g.gname = name) scope.groups
49   with Not_found ->
50     match scope.parent with
51       Some(parent) -> find_group parent name
52     | _ -> raise Not_found
53
54 let verify_args_signature fdcl formals actuals =
55   let rec helper check_types flist alist = match flist, alist with
56     [], [] -> true
57   | f :: frest, a :: arest ->
58     let tf = f.vtype
59     and _, ta = a in
60     if tf = ta || not check_types then
61       helper check_types frest arest
62     else
63       (match tf, ta with
64         Group(form_name, _), Group(act_name, _) ->
65           (* if form_name = act_name then *)
66           helper check_types frest arest
67         (* else
68           false *)
69         | _, _ -> false)
70   | _ :: _, [] -> false
71   | [], _ :: _ -> false
72   in
73   let fname, is_builtin =
74     (match fdcl with
75       BasicFunc(f) -> f.fname, f.f_is_builtin
76     | AssertFunc(f) -> f.aname, f.a_is_builtin)
77   in
78   let check_formal_types =
79     if fname = "print" && is_builtin then
80       let formal_type =
81         (match fdcl with
82           BasicFunc(f) -> List.hd f.formals
83         | _ -> raise (SemError ("Internal error: built-in print function
84           cannot be an assert function")))
85       in
86       (match formal_type.vtype with
87         Group("", _) -> false
88         | _ -> true)
89   else
90

```

```

89     true
90   in
91   helper check_formal_types formals actuals
92
93 let rec search_func_in_child (parent : group_decl) actuals name =
94   let rec helper = function
95     [] -> raise Not_found
96     | f :: rest ->
97       let fname, formals =
98         (match f with
99           BasicFunc(x) -> x.fname, x.formals
100          | AssertFunc(x) -> x.aname, x.aformals)
101      in
102      if name = fname &&
103        List.length formals = List.length actuals then
104        if verify_args_signature f formals actuals then
105          f
106        else
107          helper rest
108      else
109        helper rest
110    in
111    helper parent.methods
112
113 let search_func_in_scope scope actuals name =
114   let rec helper = function
115     [] -> raise (SemError ("Function name " ^ name ^ " exists in group
116                           scope " ^
117                           "but actuals signature not matched"))
118     | f :: rest ->
119       let fname, formals, built_in =
120         (match f with
121           BasicFunc(x) -> x.fname, x.formals, x.f_is_builtin
122           | AssertFunc(x) -> x.aname, x.aformals, x.a_is_builtin)
123       in
124       if name = "print" && name = fname && built_in then
125         f
126       else if name = fname &&
127         List.length formals = List.length actuals then
128         if verify_args_signature f formals actuals then
129           f
130         else
131           helper rest
132       else
133         helper rest
134   in
135   helper scope.functions
136
136 let rec find_child_in_group_def env (parent : group_decl) actuals name =
137   if List.exists (fun v -> v.vname = name) parent.attributes then
138     let vdcl = List.find (fun v -> v.vname = name) parent.attributes in
139     Var(vdcl), vdcl.vtype
140   else
141     if List.exists (fun f -> match f with

```

```

142             BasicFunc(x) -> x.fname = name
143             | AssertFunc(x) -> x.aname = name) parent.methods then
144     try
145         let fdcl = search_func_in_child parent actuals name in
146         let f_typ = (match fdcl with
147                         BasicFunc(x) -> x.ftype
148                         | AssertFunc(x) -> Bool) in
149         Fun(fdcl), f_typ
150     with Not_found ->
151         (match parent.extends with
152             None -> raise Not_found
153             | Some(g) -> find_child_in_group_def env g actuals name)
154     else
155         (match parent.extends with
156             None -> raise Not_found
157             | Some(g) -> find_child_in_group_def env g actuals name)
158
159 let rec find_child env (par_instance : var_decl) actuals name =
160     let class_name =
161         (match par_instance.vtype with
162             Group(s, _) -> s
163             | _ -> raise (SemError ("DOT operator does not work with non-group
variable: " ^ par_instance.vname)))
164     in
165     let parent =
166         try
167             find_group env.scope class_name
168         with Not_found ->
169             raise (SemError ("Group definition not found: " ^ class_name))
170     in
171     let child, child_typ =
172         try
173             find_child_in_group_def env parent actuals name
174         with Not_found ->
175             raise (SemError ("Child of " ^ parent.gname ^ " not found: " ^
name ^
";\nActual types: " ^ String.concat ", " (List.
176             map (fun (_, typ) -> string_of_t typ) actuals)))
177     in
178     (match child with
179         Var(v) -> Attrib(par_instance, v), child_typ
180         | Fun(f) -> Method(par_instance, f), child_typ
181         | _ -> raise (SemError ("Child is not a variable or function")))
182
183 let find_this_child env actuals name =
184     let info =
185         (match env.partial_group_info with
186             None -> (raise (SemError "'this' field call outside of group
definition"))
187             | Some(info) -> info) in
188     let this_dummy =
189         { vname = "this"; vtype = Group(info.group_name, None);
vinit = None; vloop = false } in
190     let scope = info.symbols in

```

```

192 if List.exists (fun v -> v.vname = name) scope.variables then
193   let vdcl = List.find (fun v -> v.vname = name) scope.variables in
194     Attrib(this_dummy, vdcl), vdcl.vtype
195 else
196 if List.exists (fun f -> match f with
197                 BasicFunc(x) -> x.fname = name
198                 | AssertFunc(x) -> x.aname = name) scope.functions then
199   let fdcl = search_func_in_scope scope actuals name in
200   let f_typ = (match fdcl with
201                 BasicFunc(x) -> x.ftype
202                 | AssertFunc(x) -> Bool) in
203     Method(this_dummy, fdcl), f_typ
204 else
205   (match info.par with
206     None -> raise Not_found
207     | Some(p) ->
208       let child, child_typ =
209         try
210           find_child_in_group_def env p actuals name
211         with Not_found ->
212             raise (SemError ("Child of 'this' not found: " ^ name))
213       in
214       (match child with
215         Var(v) -> Attrib(this_dummy, v), child_typ
216         | Fun(f) -> Method(this_dummy, f), child_typ
217         | _ -> raise (SemError ("Child is not a variable or function"))
218       )))
219
220 let rec search_func_in_parent scope actuals name =
221 let rec helper = function
222   [] ->
223     (match scope.parent with
224       Some(parent) -> search_func_in_parent parent actuals name
225       | None -> raise Not_found)
226   | f :: rest ->
227     let formals = (match f with
228                   BasicFunc(x) -> x.formals
229                   | AssertFunc(x) -> x.afomals)
230     in
231     if List.length formals = List.length actuals then
232       if verify_args_signature f formals actuals then
233         f
234       else
235         helper rest
236     else
237       helper rest
238   in
239   helper scope.functions
240
241 let rec search_field_local_first scope actuals name =
242   let fe_is_v =
243     List.exists (fun x -> x.vname = name) scope.variables
244   and fe_is_f =
245     List.exists (fun x -> match x with

```

```

245             BasicFunc(b) -> b.fname = name
246             | AssertFunc(a) -> a.aname = name )
247         scope.functions
248     and fe_is_g =
249         List.exists (fun x -> x.gname = name) scope.groups
250     in
251     if fe_is_v then
252         let vdecl =
253             List.find (fun v -> v.vname = name) scope.variables
254             in
255             Var(vdecl)
256     else if fe_is_f then
257         let rec helper = function
258             [] ->
259                 (match scope.parent with
260                  Some(parent) -> search_func_in_parent parent actuals name
261                  | None -> raise (SemError ("Function name " ^ name ^ "
262         exists in scope " ^
263                                     "but actuals signature not matched")))
264             | f :: rest ->
265                 let n, formals, built_in =
266                     (match f with
267                      BasicFunc(x) -> x.fname, x.formals, x.f_is_builtin
268                      | AssertFunc(x) -> x.aname, x.aformals, x.a_is_builtin)
269                 in
270                 if n = name && n = "print" && built_in then
271                     f
272                 else if n = name &&
273                     List.length formals = List.length actuals &&
274                     verify_args_signature f formals actuals then
275                         f
276                     else
277                         helper rest
278                     in
279                     let fdecl = helper scope.functions in
280                     Fun(fdecl)
281             else if fe_is_g then
282                 let gdecl =
283                     List.find (fun g -> g.gname = name) scope.groups
284                     in
285                     Grp(gdecl)
286             else
287                 match scope.parent with
288                     Some(parent) -> search_field_local_first parent actuals name
289                     | _ -> raise Not_found
290
291 let rec check_field env actuals = function
292     Ast.Id(name) ->
293         let dcl =
294             try
295                 search_field_local_first env.scope actuals name
296             with Not_found ->
297                 raise (SemError("Undeclared identifier: " ^ name))
298         in

```

```

298     let typ = match dcl with
299         Var(v) -> v.vtype
300     | Fun(x) -> (match x with
301         BasicFunc(f) -> f.ftype
302         | AssertFunc(a) -> Bool)
303     | Grp(g) -> Group(g.gname, None)
304     | This -> raise (SemError("Internal error: 'this' keyword match
with Ast.Id"))
305     | _ -> raise (SemError "Internal error: Ast.Id matched with Attrib
or Method")
306     in
307     dcl, typ
308 | Ast.This ->
309     let fname =
310         (match env.partial_group_info with
311             None -> raise (SemError "'this' keyword used outside of group
declaration"))
312         | Some(info) -> info.group_name)
313     in
314     This, Group(fname, None)
315 | Ast.FieldCall(fe, name) ->
316     let parent, _ = check_field env [] fe in
317     (match parent with
318         Var(par) ->
319             (try
320                 find_child env par actuals name
321             with Not_found ->
322                 raise (SemError("Undeclared child identifier: " ^ name)))
323         | This ->
324             (try
325                 find_this_child env actuals name
326             with Not_found ->
327                 raise (SemError("Undeclared 'this' child identifier: " ^
name)))
328         | _ ->
329             raise (SemError("Parent is either a function or variable, not
a group")))
330
331 let create_ll_name scope =
332     count.i <- count.i + 1;
333     "__ll__" ^ string_of_int count.i
334
335 let create_elem_name scope =
336     count.i <- count.i + 1;
337     "__elem__" ^ string_of_int count.i
338
339 let ast_op_to_sast_op = function
340     Ast.Add -> Add
341     | Ast.Sub -> Sub
342     | Ast.Mult -> Mult
343     | Ast.Div -> Div
344     | Ast.Equal -> Equal
345     | Ast.Neq -> Neq
346     | Ast.Less -> Less

```

```

347 | Ast.Leq -> Leq
348 | Ast.Greater -> Greater
349 | Ast.Geq -> Geq
350 | Ast.Mod -> Mod
351 | Ast.And -> And
352 | Ast.Or -> Or
353
354 let require_bool e msg = match e with
355   _, Bool -> ()
356 | _, _ -> raise (SemError msg)
357
358 let require_int e msg = match e with
359   _, Int -> ()
360 | _, _ -> raise (SemError msg)
361
362 let require_same e1 e2 msg =
363   let _, t1 = e1
364   and _, t2 = e2 in
365   if t1 = t2 then
366     ()
367   else raise (SemError msg)
368
369 let require_integer_list l msg = match l with
370   _, List_t(typ) ->
371     (match typ with
372       Int -> ()
373       | _ -> raise (SemError msg))
374   | _, _ -> raise (SemError msg)
375
376 let rec require_parent_helper pname gdcl msg =
377   if gdcl.gname = pname then
378     ()
379   else
380     (match gdcl.extends with
381       Some(gp) -> require_parent_helper pname gp msg
382       | _ -> raise (SemError msg))
383
384 let rec require_parent env pname fe msg = match fe with
385   Var(v) ->
386     (match v.vtype with
387       Group(s, _) ->
388         let gdcl =
389           try
390             find_group env.scope s
391           with Not_found ->
392             raise (SemError ("require_parent did not find group: " ^ pname))
393           in
394             require_parent_helper pname gdcl msg
395           | _ -> raise (SemError ("Variable is not a group: " ^ v.vname)))
396         | _ -> raise (SemError ("Field expr. is not a variable"))
397
398 let rec verify_args_helper f_typ a_typ check_types = match f_typ, a_typ
  with

```

```

399     [] , [] -> ()
400   | f_hd :: f_rest, a_hd :: a_rest ->
401     if f_hd = a_hd then
402       verify_args_helper f_rest a_rest check_types
403     else if not check_types then
404       ()
405     else
406       (match f_hd, a_hd with
407         Group(_, _), Group(_, _) -> ()
408       | _, _ ->
409         raise (SemError ("Formal type " ^ string_of_t f_hd ^ " " ^
410                     "does not match actual type " ^ string_of_t
411                     a_hd)))
412       | _, [] ->
413         raise (SemError ("Formal and actual argument lengths do not match"))
414       | [], _ ->
415         raise (SemError ("Formal and actual argument lengths do not match"))

416 let verify_args fdcl formals actuals =
417   let f_typ = List.map (fun v -> v.vtype) formals in
418   let a_typ = List.map (fun (_, typ) -> typ) actuals in
419   let fname, is_built_in =
420     (match fdcl with
421       BasicFunc(f) -> f.fname, f.f_is_builtin
422     | AssertFunc(f) -> f.aname, f.a_is_builtin)
423   in
424   if fname = "print" && is_built_in then
425     match fdcl with
426       BasicFunc(f) -> ()
427     | _ -> raise (SemError ("Internal error: built-in print function
428                             cannot be an assert function"))
429   else
430     verify_args_helper f_typ a_typ true

431 let rec verify_elems_list_type env typ = function
432   [] -> typ
433   | (a, b) :: rest ->
434     if b = typ then
435       verify_elems_list_type env typ rest
436     else
437       raise (SemError ("List elements are not all of same type: " ^
438                         string_of_t typ))

439 let rec check_single_elem env (detail, typ) = match detail with
440   IntLiteral(i, _) ->
441     let name = create_elem_name env.scope in
442     IntLiteral(i, name), typ
443   | StrLiteral(s, _) ->
444     let name = create_elem_name env.scope in
445     StrLiteral(s, name), typ
446   | ListLiteral(l1) ->
447     raise (SemError ("A list literal cannot be a list element."))
448   | BoolLiteral(bl, _) ->
449     let name = create_elem_name env.scope in

```

```

451     (match bl with
452         True -> BoolLiteral(True, name), typ
453         | False -> BoolLiteral(False, name), typ)
454     | Field(fe) ->
455         (match fe with
456             Var(_) -> detail, typ
457             | Attrib(_, _) -> detail, typ
458             | Method (_, _) -> detail, typ
459             | _ ->
460                 raise (SemError ("A list element is not a variable, " ^
461                               "attribute, or method")))
462     | Binop(e1, op, e2) ->
463         raise (SemError ("A list element cannot be binary expression."))
464     | Assign(fe, e) ->
465         raise (SemError ("A list element cannot be assign expression."))
466     | Call(vd_opt, fd, el) ->
467         raise (SemError ("A list element cannot be call expression."))
468     | Uminus(e) ->
469         raise (SemError ("A list element cannot be unary minus expression."))
470     )
471     | Not(e) ->
472         raise (SemError ("A list element cannot be not expression."))
473     | Noexpr ->
474         raise (SemError ("A list element cannot be an empty expression."))
475     | Remove(_) ->
476         raise (SemError ("A list element cannot be a remove expression."))
477     | Place(_) ->
478         raise (SemError ("A list element cannot be a place expression."))
479     | _ -> detail, typ
480
481 and check_listlit env = function
482     Ast.Elems(elems_list) ->
483         let el = List.map (check_expr env) elems_list in
484         let el = List.map (check_single_elem env) el in
485         let e, typ = List.hd el in
486         let typ = verify_elems_list_type env typ el in
487         let name = create_ll_name env.scope in
488             Elems(el, name), List_t(typ)
489     | Ast.EmptyList ->
490         EmptyList, List_t(Void)
491
492 and check_expr env = function
493     Ast.IntLiteral(i) -> IntLiteral(i, ""), Int
494     | Ast.StrLiteral(s) -> StrLiteral(s, ""), Str
495     | Ast.ListLiteral(ll) ->
496         let l, typ = check_listlit env ll in
497             ListLiteral(l), typ
498     | Ast.BoolLiteral(b) ->
499         (match b with
500             Ast.True -> BoolLiteral(True, ""), Bool
501             | Ast.False -> BoolLiteral(False, ""), Bool)
502     | Ast.VoidLiteral -> VoidLiteral, Void
503     | Ast.Field(fe) ->
504         let f, typ = check_field env [] fe in

```

```

504     Field(f), typ
505 | Ast.Binop(e1, op, e2) ->
506     let e1 = check_expr env e1
507     and e2 = check_expr env e2 in
508     if op = Ast.Add || op = Ast.Sub || op = Ast.Mult ||
509     op = Ast.Div || op = Ast.Mod then
510         let _, t1 = e1 and _, t2 = e2 in
511         let op = ast_op_to_sast_op op in
512         match t1, t2 with
513             Str, Str ->
514                 Binop(e1, op, e2), Str
515             | Int, Int -> Binop(e1, op, e2), Int
516             | _, _ ->
517                 raise (SemError ("Additional operation requires two integer "
518
519                         "or two string operands."))
520     else if op = Ast.Less || op = Ast.Leq ||
521     op = Ast.Greater || op = Ast.Geq then
522         ((require_int e1 "Left operand must be integer";
523           require_int e2 "Right operand must be integer";
524           let op = ast_op_to_sast_op op in
525             Binop(e1, op, e2), Bool))
526     else if op = Ast.Equal || op = Ast.Neq then
527         (require_same e1 e2 "Left and right operands in comparison must be
528         equal";
529         let op = ast_op_to_sast_op op in
530             Binop(e1, op, e2), Bool)
531     else (* op = Ast.And || op = Ast.Or *)
532         (require_bool e1 "Left operand must be boolean";
533           require_bool e2 "Right operand must be boolean";
534           let op = ast_op_to_sast_op op in
535             Binop(e1, op, e2), Bool)
536 | Ast.Assign(fd, e) ->
537     let field, tf = check_field env [] fd
538     and e = check_expr env e in
539     let _, te = e in
540     if tf = te then
541         Assign(field, e), tf
542     else
543         raise (SemError ("Types differ in assignment expression; expected:
544
545                         string_of_t tf))
546 | Ast.Call(fd, el) ->
547     let actuals = List.map (check_expr env) el in
548     let fd, typ = check_field env actuals fd in
549     (match fd with
550         Fun(f) ->
551             (match f with
552                 BasicFunc(bf) ->
553                     verify_args f bf.formals actuals
554                 | AssertFunc(af) ->
555                     verify_args f af.aformals actuals);
556         Call(None, f, actuals), typ
557     | This -> raise (SemError ("Not callable: 'this'")))

```

```

555     | Var(v) -> raise (SemError ("Not callable: " ^ v.vname))
556     | Attrib(v1, v2) -> raise (SemError ("Not callable: " ^ v1.vname
557     ^ "." ^ v2.vname))
558     | Grp(g) ->
559         let par =
560             { vname = g.gname ; vtype = Group(g.gname, None);
561               vinit = None; vloop = false }
562         in
563         let init =
564             List.find (fun f -> match f with
565                         BasicFunc(x) -> x.fname = "__init__"
566                         | AssertFunc(_) -> false) g.methods
567         in
568         let formals = match init with
569             BasicFunc(x) -> x.formals
570             | AssertFunc(_) -> raise (SemError ("Internal error:
571 __init__ is an assert function"))
572         in
573         verify_args init formals actuals;
574         Call(Some(par), init, actuals), Group(g.gname, None)
575     | Method(par, child) ->
576         (match par.vtype with
577             Group(s, _) -> ()
578             | _ -> raise (SemError ("Method call with parent that is
579 not a group")));
580         (match child with
581             BasicFunc(bf) ->
582                 verify_args child bf.formals actuals
583             | AssertFunc(af) ->
584                 verify_args child af.afomrals actuals);
585         Call(Some(par), child, actuals), typ)
586     | Ast.Element(e1, e2) ->
587         let e1 = check_expr env e1
588         and e2 = check_expr env e2 in
589         let _, t1 = e1 in
590         require_int e2 "Integer expected for element number";
591         (match t1 with
592             List_t(typ) -> Element(e1, e2), typ
593             | _ -> raise (SemError ("Expression not subscriptable")))
594     | Ast.Uminus(e) ->
595         let e = check_expr env e in
596         require_int e "Operand must be integer";
597         Uminus(e), Int
598     | Ast.Not(e) ->
599         let e = check_expr env e in
600         require_bool e "Operand must be boolean";
601         Not(e), Bool
602     | Ast.Noexpr ->
603         Noexpr, Void
604     | Ast.Remove(fd1, ll) ->
605         let board_name = match fd1 with
606             Ast.Id(s) -> s
607             | _ -> raise (SemError "Not implemented")
608         in

```

```

606   let toi_call = Ast.Call(Ast.FieldCall(Ast.Id(board_name), "toi"),
607                           [Ast.ListLiteral(l1)])
608   in
609   let rmv_call = Ast.Call(Ast.FieldCall(Ast.Id(board_name), "remove"),
610                           [toi_call])
611   in
612   let rmv_call = check_expr env rmv_call in
613   let fd1, _ = check_field env [] fd1 in
614   let checked_l1, l1_typ = check_listlit env l1 in
615   require_parent env "Board" fd1 "Board (sub)group expected";
616   require_integer_list (checked_l1, l1_typ) "List of integers expected
617   ";
618   Remove(rmv_call), Bool
619 | Ast.Place(fd1, fd2, l1) ->
620   let board_name = match fd2 with
621     Ast.Id(s) -> s
622     | _ -> raise (SemError "Not implemented")
623   in
624   let toi_call = Ast.Call(Ast.FieldCall(Ast.Id(board_name), "toi"),
625                           [Ast.ListLiteral(l1)])
626   in
627   let plc_call = Ast.Call(Ast.FieldCall(Ast.Id(board_name), "place"),
628                           [Ast.Field(fd1); toi_call])
629   in
630   let plc_call = check_expr env plc_call in
631   let fd1, _ = check_field env [] fd1
632   and fd2, _ = check_field env [] fd2
633   and l1, l1_typ = check_listlit env l1 in
634   require_parent env "Piece" fd1 "Piece (sub)group expected";
635   require_parent env "Board" fd2 "Board (sub)group expected";
636   require_integer_list (l1, l1_typ) "List of integers expected";
637   Place(plc_call), Bool
638
639 let rec verify_expr_list_type env typ = function
640   [] -> typ
641   | (a, b) :: rest ->
642     if b = typ then
643       verify_expr_list_type env typ rest
644     else
645       raise (SemError ("List of expressions are not all of same type: "
646
647 (* let verify_group_cast env to_name from_name =
648 let rec helper gdcl =
649   if gdcl.gname = to_name then
650     ()
651   else
652     (match gdcl.extends with
653      Some(g) -> helper g
654      | None -> raise (SemError ("Invalid group cast: " ^ to_name ^
655                      " is not an ancestor of " ^ from_name)))
656
657 in
658 let from_grp = find_group env.scope from_name in

```

```

658     helper from_grp *)
659
660 let check_init env v_name v_typ = function
661   Ast.ExprInit(e) ->
662     let e = check_expr env e in
663     let _, typ = e in
664     if v_typ = typ then
665       Some(e)
666     else
667       (match v_typ, typ with
668        Group(to_, _), Group(from, _) ->
669          (* verify_group_cast env to_ from; *)
670          Some(e)
671        | _, _ ->
672          raise (SemError ("Variable initiation type for identifier " ^
673                         v_name ^
674                         " does not match, expected: " ^ string_of_t
675                         v_typ)))
676   | Ast.NoInit -> None
677
678 let rec find_turn_name (scope : symbol_table) name =
679   try
680     List.find (fun t -> t = name) scope.turns
681   with Not_found ->
682     match scope.parent with
683       Some(parent) -> find_turn_name parent name
684     | _ -> raise Not_found
685
686 let rec check_stmt env = function
687   Ast.Block(sl) ->
688     let scope' =
689       { parent = Some(env.scope);
690         variables = [];
691         functions = [];
692         groups = [];
693         turns = [];
694         ll_count = 0;
695         elem_count = 0 } in
696     let env' =
697       { env with scope = scope'; } in
698     let sl = List.map (fun s -> check_stmt env' s) sl in
699     Block(scope', sl)
700   | Ast.Expr(e) -> Expression(check_expr env e)
701   | Ast.Pass(s, e) ->
702     let turn_name =
703       try
704         find_turn_name env.scope s
705       with Not_found ->
706         raise (SemError ("Turn name not found: " ^ s))
707     in
708     let dummy_turn_func =
709       { ftype = Void;
710         fname = turn_name;
711         formals = []};

```

```

710     locals = [];
711     body = [];
712     turns_func = true;
713     group_method = "";
714     f_is_built_in = false }
715   in
716   let e = check_expr env e in
717   require_int e "Must pass to an integer player.";
718   Pass(BasicFunc(dummy_turn_func), e)
719 | Ast.Return(e) ->
720   let e = check_expr env e in
721   let _, typ = e in
722   if env.return_type <> typ then
723     raise (SemError ("Return types do not match; expected: " ^
724             string_of_t env.return_type))
725   else
726     Return(e)
727 | Ast.Break ->
728   if env.in_loop = false then
729     raise (SemError ("Break outside of loop"))
730   else
731     Break
732 | Ast.Continue ->
733   if env.in_loop = false then
734     raise (SemError ("Continue outside of loop"))
735   else
736     Continue
737 | Ast.End -> End
738 | Ast.If(e, s1, e_opt, s2) ->
739   let e = check_expr env e in
740   let e_opt = match e_opt with
741     None -> None
742     | Some(expr) -> Some(check_expr env expr)
743   in
744   require_bool e "Predicate of if must be boolean";
745   If(e, check_stmt env s1, e_opt, check_stmt env s2)
746 | Ast.For(vd, el, s) ->
747   let name, t_vd = vd.Ast.vname, id_type_to_t vd.Ast.vtype in
748   let decl =
749   { vname = name;
750     vtype = t_vd;
751     vinit = check_init env name t_vd vd.Ast.vinit;
752     vloop = true }
753   and el = List.map (check_expr env) el in
754   let _, t_el = List.hd el in
755   let t_el = verify_expr_list_type env t_el el in
756   if t_vd <> t_el then
757     raise (SemError ("For loop elements and loop variable must be " ^
758                     "the same type. Variable type: " ^ string_of_t
759                     t_vd ^
760                     "; List type: " ^ string_of_t t_el))
761   else
762     let scope' =
763       { parent = Some(env.scope);

```

```

763     variables = [];
764     functions = [];
765     groups = [];
766     turns = [];
767     ll_count = 0;
768     elem_count = 0 } in
769   let env' = { env with scope = scope'; in_loop = true } in
770   scope'.variables <- decl :: scope'.variables;
771   For(decl, el, check_stmt env' s)
772 | Ast.While(e, s) ->
773   let e = check_expr env e in
774   let scope' =
775     { parent = Some(env.scope);
776     variables = [];
777     functions = [];
778     groups = [];
779     turns = [];
780     ll_count = 0;
781     elem_count = 0 } in
782   let env' = { env with scope = scope'; in_loop = true } in
783   require_bool e "While loop predicate must be Boolean";
784   While(e, check_stmt env' s)
785
786 let rec check_for_begin(turns_section) = match turns_section with
787   [] -> false
788 | Ast.BasicFunc(f) :: rest ->
789   if f.Ast.fname = "begin" then
790     true
791   else
792     check_for_begin(rest)
793 | Ast.AssertFunc(f) :: rest -> check_for_begin(rest)
794
795 let require_no_init = function
796   Ast.NoInit -> ()
797 | Ast.ExprInit(e) ->
798   raise (SemError "Function formal arguments cannot have default
values.")
799
800 let require_non_void v = match v.vtype with
801   Void -> raise (SemError (v.vname ^ " declared with void type"))
802 | _ -> ()
803
804 let verify_not_redeclaring scope name =
805   let is_var_or_fun_in_scope =
806     List.exists (fun x -> x.vname = name) scope.variables ||
807     List.exists (fun x -> match x with
808       BasicFunc(f) -> f.fname = name
809       | AssertFunc(f) -> f.aname = name) scope.functions
810   in
811   let gdcl_opt =
812     try
813       Some(find_group scope name)
814     with Not_found ->
815       None

```

```

816   in
817   match gdcl_opt with
818     Some(_) ->
819       raise (SemError ("Cannot redeclare an identifier that is also a
820                     group " ^ name))
821     | None ->
822       if is_var_or_fun_in_scope then
823         raise (SemError ("Cannot redeclare an indentifer matching with a " ^
824                     "variable or a function in current scope: " ^ name))
825     )
826   else
827     ()
828
829 let check_vdcl_helper env v init_ok =
830   let name = v.Ast.vname in
831   let t_vd = id_type_to_t v.Ast.vtype in
832   let init = check_init env name t_vd v.Ast.vinit in
833   let init =
834     (match init with
835       None -> init
836     | _ ->
837       if not init_ok then
838         raise (SemError ("Initiation not allowed here for variable: " ^ name))
839       else
840         init)
841   in
842   let decl =
843     { vname = name;
844       vtype = t_vd;
845       vinit = init;
846       vloop = false }
847   in
848   verify_not_redeclaring env.scope v.Ast.vname;
849   require_non_void decl;
850   decl
851
852 let check_formal env v =
853   let decl = check_vdcl_helper env v false in
854   let decl = {decl with vinit = None} in
855   require_no_init v.Ast.vinit;
856   env.scope.variables <- decl :: env.scope.variables;
857   decl
858
859 let check_vdcl env v =
860   let decl = check_vdcl_helper env v true in
861   env.scope.variables <- decl :: env.scope.variables;
862   decl
863
864 let rec verify_implicit_return_basic_fun name ftyp body=
865   let last_stmt =
866     try
867       List.hd (List.rev body)

```

```

867     with Failure("hd") ->
868         End (* empty body, so use non-Return dummy stmt *)
869     in
870     match last_stmt with
871     | Return(e) ->
872         () (* Last statement is not an implicit return *)
873     | Block(scope, sl) ->
874         verify_implicit_return_basic_fun name ftyp sl
875     | If(e, s1, e_opt, s2) ->
876         verify_implicit_return_basic_fun name ftyp [s1];
877         verify_implicit_return_basic_fun name ftyp [s2]
878     | For(vd, el, s) ->
879         verify_implicit_return_basic_fun name ftyp [s]
880     | While(e, s) ->
881         verify_implicit_return_basic_fun name ftyp [s]
882     | _ ->
883         (match ftyp with
884          | Void -> ()
885          | _ -> raise (SemError ("function " ^ name ^ " implicit return " ^
886                           "invalid type, expected: " ^ string_of_t ftyp)))
887
888 let rec verify_return_pass_end_turn_fun name body =
889 let error_on_return_stmt = function
890   | Return(_) ->
891       raise (SemError ("Return statement in turn function: " ^ name))
892   | _ -> ()
893 in
894 let last_stmt =
895   try
896     List.hd (List.rev body)
897   with Failure("hd") ->
898     raise (SemError ("Turn function " ^ name ^
899                   " does not end with 'end' or 'pass' statement."))
900 in
901 List.iter error_on_return_stmt body;
902 match last_stmt with
903 | Pass(fd, e) -> ()
904 | End -> ()
905 | Block(scope, sl) ->
906   verify_return_pass_end_turn_fun name sl
907 | If(e, s1, e_opt, s2) ->
908   verify_return_pass_end_turn_fun name [s1];
909   verify_return_pass_end_turn_fun name [s2]
910 | For(vd, el, s) ->
911   verify_return_pass_end_turn_fun name [s]
912 | While(e, s) ->
913   verify_return_pass_end_turn_fun name [s]
914 | _ -> raise (SemError ("Turn function " ^ name ^
915                           " does not end with 'end' or 'pass' statement."))
916
917 let verify_implicit_return = function
918   | AssertFunc(f) ->
919       () (* Implicit and explicit returns always Bool *)
920   | BasicFunc(f) ->

```

```

921         if f.turns_func then
922             verify_return_pass_end_turn_fun f.fname f.body
923         else
924             verify_implicit_return_basic_fun f.fname f.ftype f.body
925
926 let check_basic_func env in_turn_section (f : Ast.basic_func_decl) =
927     let scope' =
928         { parent = Some(env.scope);
929         variables = [];
930         functions = [];
931         groups = [];
932         turns = [];
933         ll_count = 0;
934         elem_count = 0 } in
935     let env' =
936         { env with scope = scope';
937             return_type = id_type_to_t f.Ast.ftype; } in
938     let fl = List.map (fun v -> check_formal env' v) f.Ast.formals in
939     let ll = List.map (fun dcl -> check_vdcl env' dcl) f.Ast.locals in
940     let sl = List.map (fun s -> check_stmt env' s) f.Ast.body in
941     let gname = (match env.partial_group_info with None -> "" | Some(g) ->
942     g.group_name) in
943     let fdecl =
944         BasicFunc({ ftype = id_type_to_t f.Ast.ftype;
945                     fname = f.Ast.fname;
946                     formals = fl;
947                     locals = ll;
948                     body = sl;
949                     turns_func = in_turn_section;
950                     group_method = gname;
951                     f_is_built_in = false })
952
953     in
954     verify_implicit_return fdecl;
955     env.scope.functions <- fdecl :: env.scope.functions;
956     fdecl
957
958 let rec verify_assert_func_stmt stmt =
959     let msg_bool = "Assert function expression statement or return statement"
960     ^
961             " must be of type boolean."
962     in
963     let msg_stmt = "Assert function statement cannot be " in
964     match stmt with
965     | Block(scope, sl) -> List.iter verify_assert_func_stmt sl
966     | Expression(e) -> require_bool e msg_bool
967     | Return(e) -> require_bool e msg_bool
968     | Break -> ()
969     | Continue -> ()
970     | End -> raise (SemError (msg_stmt ^ "end."))
971     | Pass(_, _) -> raise (SemError (msg_stmt ^ "pass."))
972     | If(e, s1, e_opt, s2) ->
973         verify_assert_func_stmt s1;
974         verify_assert_func_stmt s2
975     | For(vd, el, s) -> verify_assert_func_stmt s

```

```

973 | While(e, s) -> verify_assert_func_stmt s
974
975 let check_assert_func env in_turn_section (f : Ast.assert_decl) =
976   let scope' =
977     { parent = Some(env.scope);
978      variables = [];
979      functions = [];
980      groups = [];
981      turns = [];
982      ll_count = 0;
983      elem_count = 0 } in
984   let env' =
985     { env with scope = scope';
986       return_type = Bool; } in
987   let fl = List.map (fun v -> check_formal env' v) f.Ast.formals in
988   let ll = List.map (fun dcl -> check_vdcl env' dcl) f.Ast.locals in
989   let sl = List.map (fun s -> check_stmt env' s) f.Ast.body in
990   let ret_stmt = Return(BoolLiteral(True, ""), Bool) in
991   let sl = ret_stmt :: List.rev sl in
992   let sl = List.rev sl in
993   let gname = (match env.partial_group_info with None -> "" | Some(g) -> g
994     .group_name) in
995   let fdecl =
996     AssertFunc({ fname = f.Ast.fname;
997                 aformals = fl;
998                 alocals = ll;
999                 abody = sl ;
1000                a_turns_func = in_turn_section;
1001                a_group_method = gname;
1002                a_is_built_in = false })
1003   in
1004     List.iter verify_assert_func_stmt sl;
1005     env.scope.functions <- fdecl :: env.scope.functions;
1006     fdecl
1007
1008 let check_function env in_turn_section = function
1009   Ast.BasicFunc(f) ->
1010     verify_not_redeclaring env.scope f.Ast.fname;
1011     check_basic_func env in_turn_section f
1012   | Ast.AssertFunc(f) ->
1013     verify_not_redeclaring env.scope f.Ast.fname;
1014     check_assert_func env in_turn_section f
1015
1016 let find_init_func methods =
1017   List.find (fun x -> match x with
1018             BasicFunc(f) -> f.fname = "__init__"
1019             | AssertFunc(f) -> false) methods
1020
1021 let verify_extends parent par_actualls init_opt =
1022   match parent with
1023     Some(p) ->
1024       (match par_actualls with
1025         Some(el) ->
1026           (match init_opt with

```

```

1026             Some(init) ->
1027                 raise (SemError ("Defined child constructor while also
1028     using parent's constructor"))
1029             | None ->
1030                 let par_init = try
1031                     find_init_func p.methods
1032                 with Not_found ->
1033                     raise (SemError("Constructor function not found in
1034     parent"))
1035                     in
1036                     let fdcl = par_init in
1037                     let par_init = match par_init with
1038                         BasicFunc(f) -> f
1039                         | AssertFunc(f) ->
1040                             raise (SemError("Parent constructor cannot be an
1041     assert function"))
1042                             in
1043                             if List.length par_init.formals = List.length el then
1044                                 verify_args fdcl par_init.formals el
1045                             else
1046                                 raise (SemError("Number of constrctr variables for
1047     parent's constructor does not match")))
1048                         | None ->
1049                             match init_opt with
1050                               Some(init) -> ()
1051                               | None -> () )
1052                         | None ->
1053                             (match init_opt with
1054                               Some(init) -> ()
1055                               | None -> raise (SemError("No constructor function")))
1056
1057 let rec verify_single_attrib par_attr v = match par_attr with
1058     [] -> Some(v)
1059     | p :: rest ->
1060         if p.vname = v.vname then
1061             if p.vtype = v.vtype then
1062                 None
1063                 else
1064                     raise (SemError("Cannot change type of inherited attribute"))
1065             else
1066                 verify_single_attrib rest v
1067
1068 let rec verify_attributes par_attr = function
1069     [] -> []
1070     | v :: rest ->
1071         (match verify_single_attrib par_attr v with
1072           None -> verify_attributes par_attr rest
1073           | Some(var) -> var :: verify_attributes par_attr rest)
1074
1075 let check_attrib env v =
1076     let decl = check_vdcl_helper env v false in
1077     let name = decl.vname in
1078     let scope =
1079         (match env.partial_group_info with

```

```

1076     None -> raise (SemError "Internal error: check_attrib called
1077     outside of group definition.")
1078   | Some(info) -> info.symbols) in
1079 verify_not_redeclaring scope name;
1080 scope.variables <- decl :: scope.variables;
1081 decl
1082
1083 let check_method env new_fun =
1084   let fdcl = check_function env false new_fun in
1085   let name = match fdcl with
1086     BasicFunc(f) -> f.fname
1087   | AssertFunc(f) -> f.aname
1088   in
1089   let scope =
1090     (match env.partial_group_info with
1091       None -> raise (SemError "Internal error: check_attrib called
1092       outside of group definition.")
1093   | Some(info) -> info.symbols) in
1094 verify_not_redeclaring scope name;
1095 scope.functions <- fdcl :: scope.functions;
1096 fdcl
1097
1098 let add_parent_init parent par_actualls name methods = function
1099   Some(init_fun) ->
1100     (match init_fun with
1101       BasicFunc(f) ->
1102         if f.ftype = Group(name, None) then
1103           methods
1104         else
1105           raise (SemError ("Group " ^ name ^ " __init__ function has
1106 type not equal to itself"))
1107   | AssertFunc(f) ->
1108     raise (SemError ("Group " ^ name ^ " __init__ function not a
1109 basic function")))
1110 | None ->
1111   (match parent with
1112     None -> raise (SemError "No parent but using parent __init__")
1113   | Some(par) ->
1114     let par_init =
1115       try
1116         find_init_func par.methods
1117       with Not_found ->
1118         raise (SemError ("Parent __init__ function not found for child
1119 : " ^
1120                           name)))
1121       in
1122       let child_init =
1123         (match par_actualls with
1124           Some(el) ->
1125             (match par_init with
1126               AssertFunc(f) -> raise (SemError "Assert Function used as
1127 parent __init__")
1128             | BasicFunc(f) ->
1129               let dcls =

```

```

1124             List.map2 (fun vdcl act -> {vdcl with vinit = Some(act)})
1125         }) f.formals el
1126         in
1127         { ftype = Group(name, None);
1128           fname = "__init__";
1129           formals = [];
1130           locals = dcls;
1131           body = f.body;
1132           turns_func = false;
1133           group_method = name;
1134           f_is_builtin = false })
1135       | None ->
1136         (match par_init with
1137           AssertFunc(f) -> raise (SemError "Assert Function used as
parent __init__")
1138           | BasicFunc(f) ->
1139             { f with ftype = Group(name, None); group_method = name })
1140       )
1141     in
1142     BasicFunc(child_init) :: methods)
1143
1144 let verify_repr gname = function
1145   BasicFunc(f) ->
1146     if f.ftype <> Str then
1147       raise (SemError ("Group " ^ gname ^ " __repr__() does not have Str
return type"))
1148     else if List.length f.formals > 0 then
1149       raise (SemError ("Group " ^ gname ^ " __repr__() has more than 0
formals"))
1150     else
1151       ()
1152   | AssertFunc(f) ->
1153     raise (SemError ("Group " ^ gname ^ " __repr__() cannot be an assert
function"))
1154
1155 let rec find_repr gdcl =
1156   try
1157     List.find
1158     (fun f ->
1159       (match f with
1160         BasicFunc(x) -> x.fname = "__repr__"
1161         | AssertFunc(x) -> x.aname = "__repr__")) gdcl.methods
1162   with Not_found ->
1163     match gdcl.extends with
1164       Some(par) -> find_repr par
1165     | None -> raise Not_found
1166
1167 let rec instance_of name gdcl =
1168   if gdcl.gname = name then
1169     true
1170   else
1171     match gdcl.extends with
1172       None -> false
1173       | Some(par) -> instance_of name par

```

```

1172
1173 let check_for_repr env gdcl =
1174   try
1175     (* search locally first *)
1176     let repr = find_repr { gdcl with extends = None } in
1177     verify_repr gdcl.gname repr;
1178     gdcl
1179   with Not_found ->
1180     try
1181       (* not in group, search any parents *)
1182       let repr = find_repr gdcl in
1183       let built_in = match repr with
1184         BasicFunc(f) -> f.f_is_builtin
1185         | AssertFunc(f) -> raise (SemError ("Group " ^ gdcl.gname ^ "
1186 __repr__() cannot be an assert function"))
1187       in
1188       let not_a_piece = not (instance_of "Piece" gdcl) in
1189       let repr =
1190         if built_in && not_a_piece then
1191           let s = "<Group " ^ gdcl.gname ^ " instance>" in
1192           let body = Return(StrLiteral(s, ""), Str) in
1193           BasicFunc({ ftype = Str;
1194             fname = "__repr__";
1195             formals = [];
1196             locals = [];
1197             body = [body];
1198             turns_func = false;
1199             group_method = gdcl.gname;
1200             f_is_builtin = true})
1201         else
1202           repr
1203       in
1204       verify_repr gdcl.gname repr;
1205       { gdcl with methods = repr :: gdcl.methods }
1206   with Not_found ->
1207     let s = "<Group " ^ gdcl.gname ^ " instance>" in
1208     let body = Return(StrLiteral(s, ""), Str) in
1209     let repr =
1210       { ftype = Str;
1211         fname = "__repr__";
1212         formals = [];
1213         locals = [];
1214         body = [body];
1215         turns_func = false;
1216         group_method = gdcl.gname;
1217         f_is_builtin = true }
1218     in
1219     { gdcl with methods = BasicFunc(repr) :: gdcl.methods }
1220
1221 let rec check_group env g =
1222   let parent = match g.Ast.extends with
1223     Some(fe) ->
1224       let par, _ = check_field env [] fe in
1225       (match par with

```

```

1225         Grp(p) -> Some(p)
1226         | _ -> raise (SemError ("Parent is not a group")))
1227     | None ->
1228         if g.Ast.gname = "Object" then
1229             None
1230         else
1231             raise (SemError "Must inherit from Object or another group")
1232     in
1233 let par_actuels = match g.Ast.par_actuels with
1234     Some(el) -> Some(List.map (check_expr env) el)
1235     | None -> None
1236     in
1237 let scope' =
1238     { parent = Some(env.scope);
1239     variables = [];
1240     functions = [];
1241     groups = [];
1242     turns = [];
1243     ll_count = 0;
1244     elem_count = 0 } in
1245 let partial_scope = {scope' with parent = None} in
1246 let info =
1247     { group_name = g.Ast.gname;
1248     symbols = partial_scope;
1249     par = parent } in
1250 let env' =
1251     { env with scope = scope';
1252     return_type = Void;
1253     partial_group_info = Some(info) } in
1254 let attribs = List.map (check_attrib env') g.Ast.attributes in
1255 let attribs =
1256     (match parent with
1257     | Some(par) -> verify_attributes par.attributes attribs
1258     | None -> attribs) in
1259 let methods = List.map (check_method env') g.Ast.methods in
1260 let init_opt = try
1261     Some(find_init_func methods)
1262   with Not_found ->
1263     None
1264   in
1265 let methods = add_parent_init parent par_actuels g.Ast.gname methods
1266     init_opt in
1267 verify_extends parent par_actuels init_opt;
1268 let gdecl =
1269     { gname = g.Ast.gname;
1270     extends = parent;
1271     par_actuels = par_actuels;
1272     attributes = attribs;
1273     methods = methods }
1274   in
1275 let gdecl = check_for_repr env gdecl in
1276 verify_not_redeclaring env.scope gdecl.gname;
1277 env.scope.groups <- gdecl :: env.scope.groups;
gdecl

```

```

1278
1279 let require_no_init_list_groups v = match v.vtype, v.vinit with
1280   Group(_, _), Some(_) ->
1281     raise(SemError ("Cannot initialize groups in @setup"))
1282 | List_t(_), Some(_) ->
1283   raise(SemError ("Cannot initialize lists in @setup"))
1284 | _, _ -> ()
1285
1286 let check_setup env setup_section =
1287   let vars, funcs, groups = setup_section in
1288   let v = List.map (check_vdcl env) vars in
1289   let g = List.map (check_group env) groups in
1290   let f = List.map (check_function env false) funcs in
1291   ignore(List.map require_no_init_list_groups v);
1292   v, f, g
1293
1294 let rec gather_turn_names env = function
1295   [] -> ()
1296 | Ast.BasicFunc(t) :: rest ->
1297   env.scope.turns <- t.Ast.fname :: env.scope.turns;
1298   gather_turn_names env rest
1299 | Ast.AssertFunc(t) :: rest ->
1300   raise (SemError ("Assert function cannot be a turns function: " ^ t.
1301   Ast.fname ))
1302
1303 let check_turns env turns_section =
1304   if check_for_begin(turns_section) = false then
1305     raise (SemError "No begin() function in @turns section")
1306   else
1307     gather_turn_names env turns_section;
1308     List.map (check_function env true) turns_section
1309
1310 let fix_pass_stmts_helper turns = function
1311   Pass(s, e) ->
1312     let name = (match s with
1313       BasicFunc(x) -> x.fname
1314       | AssertFunc(x) -> x.aname)
1315     in
1316     let fdcl =
1317       try
1318         List.find
1319           (fun f -> match f with
1320             BasicFunc(x) -> x.fname = name
1321             | AssertFunc(x) -> x.aname = name)
1322           turns
1323         with Not_found ->
1324           raise (SemError ("Turns function declaration not found in pass
1325 statement: " ^ name ))
1326         in
1327           Pass(fdcl, e)
1328     | s -> s
1329
1330 let fix_pass_stmts turns = function
1331   BasicFunc(f) ->

```

```

1330 let new_f =
1331   {f with body = List.map (fix_pass_stmts_helper turns) f.body}
1332   in
1333   BasicFunc(new_f)
1334 | AssertFunc(f) ->
1335   let new_f =
1336     {f with abody = List.map (fix_pass_stmts_helper turns) f.abody}
1337   in
1338   AssertFunc(new_f)
1339
1340 let check_program (program : Ast.program) =
1341   let symbols =
1342     { parent = None ;
1343       variables = Stdlib.vars;
1344       functions = Stdlib.funcs;
1345       groups = Stdlib.grps;
1346       turns = [];
1347       ll_count = 0;
1348       elem_count = 0 } in
1349   let env =
1350     { scope = symbols;
1351       return_type = Void;
1352       in_loop = false;
1353       partial_group_info = None } in
1354   let setup_section, turns_section = program in
1355   let setup_section = check_setup env setup_section in
1356   let turns_section = check_turns env turns_section in
1357   let turns_section = List.map (fix_pass_stmts turns_section)
1358     turns_section in
1358 setup_section, turns_section

```

./scanner.mll

```

1 { open Parser
2   exception LexError of string * Lexing.lexbuf }
3
4 rule token = parse
5   '\n'           {Lexing.new_line lexbuf; token lexbuf} (* http://caml.
6     inria.fr/pub/docs/manual-ocaml/libref/Lexing.html; http://courses.
7     softlab.ntua.gr/compilers/ocamlyacc-tutorial.pdf *)
8   [',', '\t', '\r'] { token lexbuf } (* Whitespace *)
9   "#"            { comment lexbuf }          (* Comment until EOL *)
10  '"'             { str (Buffer.create 20) lexbuf }  (* String until next
11    unescaped quote *)
12  '.'             { DOT }
13  '('             { LPAREN }
14  ')'             { RPAREN }
15  '['             { LBRACKET }
16  ']'             { RBRACKET }
17  '{'             { LBRACE }
18  '}'             { RBRACE }
19  ';'             { SEMI }
20  ','             { COMMA }
21  '+'             { PLUS }
22  '-'             { MINUS }

```

```

20 |   '*'      { TIMES }
21 |   '/'      { DIVIDE }
22 |   '%'      { MOD }
23 |   '='      { ASSIGN }
24 |   "=="     { EQ }
25 |   "!="     { NEQ }
26 |   '<'      { LT }
27 |   "<="     { LEQ }
28 |   ">"      { GT }
29 |   ">="     { GEQ }
30 |   "if"      { IF }
31 |   "else"    { ELSE }
32 |   "elif"    { ELIF }
33 |   "for"     { FOR }
34 |   "while"   { WHILE }
35 |   "break"   { BREAK }
36 |   "continue" { CONTINUE }
37 |   "end"     { END }
38 |   "return"  { RETURN }
39 |   "True"    { TRUE }
40 |   "False"   { FALSE }
41 |   "None"    { NONE }
42 |   "in"      { IN }
43 |   "int"     { INT }
44 |   "str"     { STR }
45 |   "bool"    { BOOL }
46 |   "void"    { VOID }
47 |   "list"    { LIST }
48 |   "group"   { GROUP }
49 |   "and"     { AND }
50 |   "or"      { OR }
51 |   "not"     { NOT }
52 |   "assert"  { ASSERT }
53 |   "<<"     { REMOVE }
54 |   ">>"     { PLACE }
55 |   "func"    { FUNC }
56 |   "pass"    { PASS }
57 |   "@setup"  { SETUP }
58 |   "@turns"  { TURNS }
59 |   "this"    { THIS }
60 |   [ '0'-'9' ]+ as lxm { INTLITERAL(int_of_string lxm) }
61 |   [ 'a'-'z' 'A'-'Z' '_'] [ 'a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
62 |   eof { EOF }
63 |   _ as char
64 |   { let msg = "Lexical error: Illegal character: " ^ Char.escaped char
65 |     in
66 |       raise (LexError(msg, lexbuf)) }
67 and comment = parse
68   '\n' { Lexing.new_line lexbuf; token lexbuf }
69 |   _ { comment lexbuf }
70
71 and str buf = parse
72   (**
```

```

73  * String parsing. Modified version of:
74  * https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-
75  * menhir.html
76  *)
77  '\"", { STRLITERAL (Buffer.contents buf) }
78  '\\\\', '\\\\', { Buffer.add_char buf '\\\\'; str buf lexbuf }
79  '\\\\', 'n', { Buffer.add_char buf '\\n'; str buf lexbuf }
80  '\\\\', '\"', { Buffer.add_char buf '\"'; str buf lexbuf }
81  [^ '\"', '\\\\']+ as lxm
82  { Buffer.add_string buf (lxm);
83  str buf lexbuf
84  }
85  _ as char
86  { let msg = "Lexical error: Illegal string character: " ^ Char.escaped
87  char in
88  raise (LexError (msg, lexbuf)) }
89  eof
90  { let msg = "Lexical error: String is not terminated" in
91  raise (LexError (msg, lexbuf)) }

```

**./senet.ml**

```

1 open Lexing
2
3 type action = Ast | Semantic | Groupeval | Compile
4
5 (** 
6  * Printing of error line based on:
7  * https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir
8  *.html
9  * http://courses.softlab.ntua.gr/compilers/ocamlyacc-tutorial.pdf
10 * http://stackoverflow.com/questions/14046392/verbose-error-with-
11 ocamlyacc?lq=1
12 *)
13
14 let string_of_error msg lb =
15   let pos = lexeme_start_p lb in
16   let line = pos.pos_lnum in
17   let first_char = lexeme_start lb - pos.pos_bol + 1 in
18   let last_char = lexeme_end lb - pos.pos_bol + 1 in
19   msg
20
21
22
23
24 let _ =
25   let action = if Array.length Sys.argv > 1 then
26     List.assoc Sys.argv.(1) [ ("a", Ast) ;
27       ("-s", Semantic) ;
28       ("-g", Groupeval) ;
29       ("-c", Compile) ]
30   else Compile in
31   let lexbuf = Lexing.from_channel stdin in

```

```

32
33   try
34
35     let program = Parser.program Scanner.token lexbuf in
36
37     match action with
38       Ast -> let listing = Ast.string_of_program program
39                   in print_string listing
40     | Semantic -> let checked_program = Sast.check_program program
41                     in ignore(checked_program);
42                     print_string ("Success!\n")
43     | Groupeval -> let checked_program = Sast.check_program program in
44                     let checked_program = Cast.build_cast checked_program
45                     in
46                         print_string (Cast.string_of_program checked_program)
47     | Compile -> let checked_program = Sast.check_program program in
48                     Compile.translate (Cast.build_cast checked_program)
49
50   with Scanner.LexError(msg,lb) ->
51     print_string (string_of_error msg lb); print_newline ()
52   | Parser.Error ->
53     let msg = "Syntax error: " ^ Lexing.lexeme lexbuf in
54     print_string (string_of_error msg lexbuf); print_newline ()
55   | Sast.SemError(msg) ->
56     let msg = "Semantic error: " ^ msg in
57     print_string msg; print_newline ()

```

./stdlib.ml

```

1 open Types
2
3 let funcs =
4   let s =
5     { vname = "print_arg";
6      vtype = Str;
7      vinit = None;
8      vloop = false }
9   in
10  let b = { s with vtype = Bool } in
11  let i = { s with vtype = Int } in
12  let g = { s with vtype = Group("", None) } in
13  let print_str =
14    { ftype = Void;
15      fname = "print";
16      formals = [s];
17      locals = [];
18      body = [];
19      turns_func = false;
20      group_method = "";
21      f_is_built_in = true }
22  in
23  let print_bool = { print_str with formals = [b] } in
24  let print_int = { print_str with formals = [i] } in
25  let print_group = { print_str with formals = [g] } in

```

```

26 let read = { print_str with fname = "read"; ftype = Str; formals = [i] }
27   in
28 let clear = { print_str with fname = "clear_input"; ftype = Void;
29   formals = [] } in
30 let stoi = { print_str with fname = "stoi"; ftype = Int; formals = [s] }
31   in
32 let exit = { print_str with fname = "exit"; ftype = Void; formals = [] }
33   in
34 let rand = { print_str with fname = "rand"; ftype = Int; formals = [] }
35   in
36 [BasicFunc(print_str); BasicFunc(print_bool); BasicFunc(print_int);
37 BasicFunc(print_group); BasicFunc(read); BasicFunc(stoi); BasicFunc(
38   exit);
39 BasicFunc(clear); BasicFunc(rand)]
40
41
42 let vars =
43   let init = IntLiteral(0, "") in
44     let v = { vname = "PLAYER_ON_MOVE" ; vinit = Some(init, Int) ;
45               vtype = Int; vloop = false }
46   in
47   [v]
48
49
50 let base_init name =
51   let stmt = Return(Field(This), Group(name, None)) in
52     { ftype = Group(name, None);
53       fname = "__init__";
54       formals = [];
55       locals = [];
56       body = [stmt];
57       turns_func = false;
58       group_method = name;
59       f_is_builtin = true }
60
61
62 let base_repr name =
63   let s = "<Group " ^ name ^ " instance>" in
64     let body = Return(StrLiteral(s, ""), Str) in
65       BasicFunc({ ftype = Str;
66                   fname = "__repr__";
67                   formals = [];
68                   locals = [];
69                   body = [body];
70                   turns_func = false;
71                   group_method = name;
72                   f_is_builtin = true})
73
74
75 let obj =
76   let name = "Object" in
77     { gname = name;
78       extends = None;
79       par_actuas = None;
80       attributes = [];
81       methods = [BasicFunc(base_init name)] }
82
83
84 let board =

```

```

74 let v =
75   { vname = "cells"; vtype = List_t(Group("Piece", None));
76     vinit = None; vloop = false }
77 in
78 let occupied = { v with vname ="occupied"; vtype = List_t(Bool) } in
79 let attr = [v; occupied] in
80 let v = { v with vname = "x"; vtype = Int } in
81 let remove =
82   { ftype = Bool; fname = "remove"; formals = [v]; locals = []; body =
83     [];
84     turns_func = false; group_method = "Board"; f_is_built_in = true }
85 in
86 let owns = { remove with ftype = Int; fname = "owns" } in
87 let v = { v with vname = "l"; vtype = List_t(Int) } in
88 let toi = { owns with fname = "toi"; formals = [v] } in
89 let tol = { owns with ftype = List_t(Int); fname = "tol" } in
90 let full = { owns with ftype = Bool; fname = "full"; formals = [] } in
91 let x = { v with vname = "x"; vtype = Int } in
92 let p = { v with vname = "p"; vtype = Group("Piece", None) } in
93 let place =
94   { ftype = Bool; fname = "place"; formals = [p; x]; locals = []; body =
95     [];
96     turns_func = false; group_method = "Board"; f_is_built_in = true }
97 in
98 let meth =
99   [BasicFunc(remove); BasicFunc(owns); BasicFunc(owns);
100    BasicFunc(toi); BasicFunc(tol); BasicFunc(base_init "Board");
101    base_repr "Board"; BasicFunc(full); BasicFunc(place)]
102 in
103 { obj with gname = "Board"; extends = Some(obj);
104   attributes = attr; methods = meth }

let piece =
105 let v = { vname = ""; vtype = Void; vinit = None; vloop = false } in
106 let owner = { v with vname = "owner"; vtype = Int } in
107 let fixed = { v with vname = "fixed"; vtype = Bool } in
108 let s = { v with vname = "s"; vtype = Str } in
109 let this_dummy =
110   { vname = "this"; vtype = Group("Loop", None);
111     vinit = None; vloop = false }
112 in
113 let stmts =
114   [Expression(Assign(Attrib(this_dummy, s), (Field(Var(s)), Int)), Int),
115    Return(Field(This), Group("Piece", None))]
116 in
117 let init = base_init "Piece" in
118 let init = { init with body = stmts; formals = [s] } in
119 let stmts =
120   [Return(Field(Attrib(this_dummy, s)), Str)]
121 in
122 let repr = base_repr "Piece" in
123 let repr =
124   match repr with
125     BasicFunc(f) -> { f with body = stmts }

```

```

126     | _ -> raise (Failure ("__repr__ for piece matched to non-basicfunc"))
127   in
128 { obj with gname = "Piece"; extends = Some(obj);
129   attributes = [owner; fixed; s];
130   methods = [BasicFunc(base_init "Piece"); BasicFunc(init); BasicFunc(
131   repr)] }
132 let boards_lib =
133 let x = { vname = "x"; vtype = Int; vinit = None; vloop = false } in
134 let y = { x with vname = "y" } in
135 let this_dummy =
136   { vname = "this"; vtype = Group("Rect", None);
137   vinit = None; vloop = false }
138 in
139 let init_cells =
140   { ftype = Void;
141   fname = "INIT_CELLS";
142   formals = [this_dummy; x];
143   locals = [];
144   body = [];
145   turns_func = false;
146   group_method = "Board";
147   f_is_built_in = true }
148 in
149 let stmts =
150   [Expression(Assign(Attrib(this_dummy, x), (Field(Var(x)), Int)), Int);
151   Expression(Assign(Attrib(this_dummy, y), (Field(Var(y)), Int)), Int);
152   Expression(Call(Some(this_dummy), BasicFunc(init_cells),
153                 [Binop((Field(Var(x)), Int), Mult,
154                         (Field(Var(y)), Int)), Int]), Void));
155   Return(Field(This), Group("Rect", None))]
156 in
157 let init = base_init "Rect" in
158 let init = { init with formals = [x; y]; body = stmts } in
159 let coord = { x with vname = "coord"; vtype = List_t(Int) } in
160 let toi =
161   {init with ftype = Int; fname = "toi"; formals = [coord]; body = [] }
162 in
163 let tol =
164   {init with ftype = List_t(Int); fname = "tol"; formals = [x]; body =
165   [] }
166 in
167 let rect =
168   { board with gname = "Rect"; extends = Some(board); attributes = [x; y];
169   methods = [BasicFunc(init); base_repr "Rect"; BasicFunc(toi);
170             BasicFunc(tol)] }
171 in
172 let this_dummy =
173   { vname = "this"; vtype = Group("Loop", None);
174   vinit = None; vloop = false }
175 in
176 let stmts =
177   [Expression(Assign(Attrib(this_dummy, x), (Field(Var(x)), Int)), Int);

```

```

177     Expression(Call(Some(this_dummy), BasicFunc(init_cells),
178                  [(Field(Var(x)), Int)]), Void);
179     Return(Field(This), Group("Loop", None))]
180 in
181 let init = base_init "Loop" in
182 let init = { init with formals = [x]; body = stmts } in
183 let coord = { x with vname = "coord"; vtype = List_t(Int) } in
184 let toi =
185   {init with ftype = Int; fname = "toi"; formals = [coord]; body = [] }
186 in
187 let tol =
188   {init with ftype = List_t(Int); fname = "tol"; formals = [x]; body =
189   [] }
190 in
191 let loop =
192   { rect with gname = "Loop"; attributes = [x];
193     methods = [BasicFunc(init); base_repr "Loop"; BasicFunc(toi);
194     BasicFunc(tol)] }
195 in
196 let this_dummy =
197   { vname = "this"; vtype = Group("Line", None);
198     vinit = None; vloop = false }
199 in
200 let init = base_init "Line" in
201 let stmts =
202   [Expression(Assign(Attrib(this_dummy, x), (Field(Var(x)), Int)), Int);
203     Expression(Call(Some(this_dummy), BasicFunc(init_cells),
204                   [(Field(Var(x)), Int)]), Void);
205     Return(Field(This), Group("Line", None)) ]
206 in
207 let init = { init with formals = [x]; body = stmts } in
208 let coord = { x with vname = "coord"; vtype = List_t(Int) } in
209 let toi =
210   {init with ftype = Int; fname = "toi"; formals = [coord]; body = [] }
211 in
212 let tol =
213   {init with ftype = List_t(Int); fname = "tol"; formals = [x]; body =
214   [] }
215 in
216 let line =
217   { loop with gname = "Line";
218     methods = [BasicFunc(init); base_repr "Loop"; BasicFunc(toi);
219     BasicFunc(tol)] }
220 in
221 let this_dummy =
222   { vname = "this"; vtype = Group("Hex", None);
223     vinit = None; vloop = false }
224 in
225 let stmts =
226   [Expression(Assign(Attrib(this_dummy, x), (Field(Var(x)), Int)), Int);
227     Expression(Call(Some(this_dummy), BasicFunc(init_cells),
228                   [(Field(Var(x)), Int)]), Void);
229     Return(Field(This), Group("Hex", None)) ]

```

```

227 let init = base_init "Hex" in
228 let init = { init with formals = [x]; body = stmts } in
229 let coord = { x with vname = "coord"; vtype = List_t(Int) } in
230 let toi =
231   {init with ftype = Int; fname = "toi"; formals = [coord]; body = [] }
232 in
233 let tol =
234   {init with ftype = List_t(Int); fname = "tol"; formals = [x]; body =
235     [] }
236 in
237 let hex =
238   { loop with gname = "Hex";
239     methods = [BasicFunc(init); base_repr "Loop"; BasicFunc(toi);
240     BasicFunc(tol)] }
241 in
242 [rect; loop; line; hex]
243
244 let grps =
245   [obj; board; piece] @ boards_lib

```

./temp/sen\_init\_base\_grps.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #include "sen_linked_list.h"
5
6 #ifndef SEN_INIT_BASE_GRPS
7 #define SEN_INIT_BASE_GRPS
8
9 bool OCCUPIED = true;
10 bool NOT_OCCUPIED = false;
11
12 struct snt_Board {
13     Sen_list cells;
14     Sen_list occupied;
15 };
16
17 struct snt_Piece {
18     int snt_owner;
19     int snt_fixed;
20     char *snt_s;
21 } p;
22
23 struct snt_Rect{
24     Sen_list cells;
25     Sen_list occupied;
26     int x;
27     int y;
28 };
29
30 struct snt_Line{
31     Sen_list cells;
32     Sen_list occupied;
33     int x;

```

```

34 };
35 struct snt_Loop{
36     Sen_list cells;
37     Sen_list occupied;
38     int x;
39 };
40 struct snt_Hex{
41     Sen_list cells;
42     Sen_list occupied;
43     int x;
44 };
45
46
47
48 void snt_Board_snt_INIT_CELLS(struct snt_Board *b, int n) {
49     int i = 0;
50
51     struct snt_Piece *dummy_piece = malloc(sizeof(struct snt_Piece));
52     dummy_piece->snt_owner = -1;
53     dummy_piece->snt_fixed = 0;
54     dummy_piece->snt_s = " ";
55
56     new_Sen_list(&(b->cells), sizeof(struct snt_Piece));
57     new_Sen_list(&(b->occupied), sizeof(bool));
58     while (i < n) {
59         push(&(b->cells), dummy_piece);
60         push(&(b->occupied), &NOT_OCCUPIED);
61         ++i;
62     }
63 }
64
65 int snt_Board_snt_owns(struct snt_Board *b, int i) {
66     struct snt_Piece *p = (struct snt_Piece *) list_elem(&(b->cells), i);
67     return p->snt_owner;
68 }
69
70 bool snt_Board_snt_full(struct snt_Board *b) {
71     int i = 0;
72     while (i < b->occupied.len) {
73         bool elem = *((bool *) list_elem(&(b->occupied), i));
74         if (elem == false) {
75             return false;
76         }
77         ++i;
78     }
79     return true;
80 }
81
82 bool snt_Board_snt_remove(struct snt_Board *b, int x) {
83     bool *elem = (bool *) list_elem(&(b->occupied), x);
84     if (*elem) {
85         *elem = false;
86         struct snt_Piece *dummy_piece = malloc(sizeof(struct snt_Piece));
87         dummy_piece->snt_owner = -1;

```

```

88     dummy_piece->snt_fixed = 0;
89     dummy_piece->snt_s = " ";
90     replace(&(b->cells), dummy_piece, x);
91     return true;
92 } else {
93     return false;
94 }
95 }

96 bool snt_Board_snt_place(struct snt_Board *b, struct snt_Piece *p, int x)
{
98     bool *occ_elem = (bool *) list_elem(&(b->occupied), x);
99     if (*occ_elem) {
100         return false;
101     }
102     *occ_elem = true;
103     replace(&(b->cells), p, x);
104     return true;
105 }

106 int snt_Board_snt_toi(struct snt_Board *b, Sen_list *list) {
107     return 0; // must be overwritten in child classes
108 }

109 }

110 int snt_Rect_snt_toi(struct snt_Rect *b, Sen_list *list) {
111     // printf("DEBUG: snt_Rect_snt_toi\n");
112     // printf("DEBUG: converting: "); printList(list, printInt); printf("\n");
113     // n);

114     int x = *((int *) list_elem(list, 0));
115     int y = *((int *) list_elem(list, 1));

116     // printf("DEBUG: x: %d\n", x);
117     // printf("DEBUG: y: %d\n", y);
118     // printf("DEBUG: b->y: %d\n", b->y);
119     // printf("DEBUG: output: %d\n", (b->y) * y + x);

120     return (b->y) * y + x;
121 }

122 }

123 }

124 }

125 }

126 int snt_Line_snt_toi(struct snt_Line *b, Sen_list *list) {
127     return *((int *) list_elem(list, 0));
128 }

129 }

130 int snt_Loop_snt_toi(struct snt_Loop *b, Sen_list *list) {
131     int x = *((int *) list_elem(list, 0));
132     while (x < 0) {
133         x += (b->x);
134     }
135     return x % (b->x);
136 }

137 }

138 Sen_list snt_Board_snt_tol(struct snt_Board *b, int i) {
139     Sen_list list;

```

```

140     new_Sen_list(&list, sizeof(int));
141     return list; // must be overwritten in child classes
142 }
143
144 Sen_list snt_Rect_snt_tol(struct snt_Rect *b, int i) {
145     int max_x = b->x;
146     Sen_list list;
147     new_Sen_list(&list, sizeof(int));
148
149     int y = i / 3;
150     int x = i - (max_x * y);
151
152     push(&list, &x);
153     push(&list, &y);
154
155     return list;
156 }
157
158 Sen_list snt_Line_snt_tol(struct snt_Line *b, int i) {
159     Sen_list list;
160     new_Sen_list(&list, sizeof(int));
161     int x = i;
162     push(&list, &x);
163     return list;
164 }
165
166 Sen_list snt_Loop_snt_tol(struct snt_Loop *b, int i) {
167     Sen_list list;
168     new_Sen_list(&list, sizeof(int));
169     int x = i;
170     push(&list, &x);
171     return list;
172 }
173
174 #endif

```

**./temp/sen\_linked\_list.h**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4
5 #ifndef SEN_LINKED_LIST
6 #define SEN_LINKED_LIST
7
8 // Heavily based on
9 // http://pseudomuto.com/development/2013/05/02/implementing-a-generic-
10 // linked-list-in-c/
11
11 typedef struct Sen_node {
12     void *data;
13     struct Sen_node *next;
14     struct Sen_node *prev;
15 } Sen_node;
16

```

```

17 typedef struct Sen_list {
18     int len;
19     int data_size;
20     Sen_node *head;
21     Sen_node *tail;
22 } Sen_list;
23
24
25 void new_Sen_list(Sen_list *list, int data_size) {
26     list->len = 0;
27     list->data_size = data_size;
28     list->head = list->tail = NULL;
29 }
30
31 /* Function to add a Sen_node at the beginning of Linked List.
32    This function expects a pointer to the data to be added
33    and size of the data type */
34 void push(Sen_list *head_ref, void *new_data) {
35     // Allocate memory for Sen_node
36     Sen_node* new_Sen_node = malloc(sizeof(Sen_node));
37
38     new_Sen_node->data = malloc(head_ref->data_size);
39 //    memcpy(new_Sen_node->data, new_data, head_ref->data_size);
40
41
42     new_Sen_node->next = head_ref->head;
43     new_Sen_node->prev = NULL;
44     head_ref->head = new_Sen_node;
45
46     if (head_ref->len == 0) {
47         head_ref->tail = head_ref->head;
48     } else {
49         new_Sen_node->next->prev = new_Sen_node;
50     }
51
52     // Copy contents of new_data to newly allocated memory.
53     // Assumption: char takes 1 byte.
54     int i;
55     for (i = 0; i < head_ref->data_size; i++) {
56         *(char*)(new_Sen_node->data + i) = *(char*)(new_data + i);
57     }
58
59     // Change head pointer as new Sen_node is added at the beginning
60
61     head_ref->len++;
62 }
63
64 // Function to print an integer
65 void printInt(void *n) {
66     printf("%d", *(int *)n);
67 }
68
69 void printStr(void *n) {
70     printf("%s", *(char **)n);

```

```

71 }
72
73 void printBool(void *b) {
74     printf("%s", *(bool *)b ? "true" : "false");
75 }
76
77 void printEmptyList(void *el) {
78     printf("[] ");
79 }
80
81 void printGroupList(Sen_list *list, char *(*fptr)(void *)) {
82     int i = 0;
83     Sen_node *n = list->tail;
84     printf("[");
85     while (i < list->len) {
86         printf("%s", (*fptr)(n->data));
87         n = n->prev;
88         ++i;
89         if (i < list->len) {
90             printf(", ");
91         }
92     }
93     printf("]");
94 }
95
96 /* Function to print Sen_nodes in a given linked list. fptr is used
97    to access the function to be used for printing current Sen_node data.
98    Note that different data types need different specifier in printf() */
99 void printList(Sen_list *list, void (*fptr)(void *)) {
100    int i = 0;
101    Sen_node *n = list->tail;
102    printf("[");
103    while (i < list->len) {
104        (*fptr)(n->data);
105        n = n->prev;
106        ++i;
107        if (i < list->len) {
108            printf(", ");
109        }
110    }
111    printf("]");
112 }
113
114 void *list_elem(Sen_list *list, int i) {
115     int x = 0;
116     Sen_node *n = list->tail;
117     while (x < list->len && x != i) {
118         n = n->prev;
119         ++x;
120     }
121     return n->data;
122 }
123
124 void replace(Sen_list *list, void *new_data, int i) {

```

```

125 Sen_node *new_node = malloc(sizeof(Sen_node));
126 new_node->data = malloc(list->data_size);
127 memcpy(new_node->data, new_data, list->data_size);
128
129 int x = 0;
130 Sen_node *tmp;
131 Sen_node *n = list->tail;
132 while (x < list->len) {
133     if (x == i - 1) {
134         tmp = n->prev;
135         n->prev = new_node;
136         n = tmp;
137     } else if (x == i) {
138         new_node->prev = n->prev;
139         new_node->next = n->next;
140         n = n->prev;
141     } else if (x == i + 1) {
142         tmp = n->prev;
143         n->next = new_node;
144         n = tmp;
145     } else {
146         n = n->prev;
147     }
148     ++x;
149 }
150
151 if (list->len > 0) {
152     if (i == 0) {
153         list->tail = new_node;
154     }
155     if (i == list->len - 1) {
156         list->head = new_node;
157     }
158 }
159 }
160
161 #endif

```

./temp/sen\_print\_base\_grps.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #include "sen_init_base_grps.h"
5
6 #ifndef SEN_PRINT_BASE_GRPS
7 #define SEN_PRINT_BASE_GRPS
8
9 char* snt_Piece_snt___repr__(struct snt_Piece *this) {
10     return this->snt_s;
11 }
12
13 char* snt_Board_snt___repr__(struct snt_Board *this) {
14     return "<Group Board instance>";
15 }

```

```

16
17 char* snt_Line_snt___repr__(struct snt_Line *this) {
18     return "<Group Line instance>";
19 }
20
21 char* snt_Loop_snt___repr__(struct snt_Loop *this) {
22     return "<Group Loop instance>";
23 }
24
25 char* snt_Hex_snt___repr__(struct snt_Hex *this) {
26     return "<Group Hex instance>";
27 }
28
29 char* snt_Rect_snt___repr__(struct snt_Rect *this) {
30     return "<Group Rect instance>";
31 }
32
33 #endif

```

./temp/sen\_read.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #ifndef SEN_READ
5 #define SEN_READ
6
7 char SENET_BUFF[128];
8
9 char *_snt_read(int x) {
10     // http://stackoverflow.com/questions/4404368/using-scanf-to-read-in-
11     // certain-amount-of-characters-in-c
12     char *s = malloc(sizeof(char) * 128);
13     sprintf(SENET_BUFF, "%%dc", x);
14     scanf(SENET_BUFF, s);
15     return s;
16 }
17
18 void _snt_clear_input() {
19     int c;
20     while ( (c = getchar()) != '\n' && c != EOF ) ;
21 }
22
23 #endif

```

./testall.sh

```

1 #!/bin/sh
2
3 SENET="./senet"
4
5 # Set time limit for all operations
6 ulimit -t 30
7
8 globallog=testall.log
9 rm -f $globallog

```

```

10 rm -f output.c
11 rm -f a.out
12 error=0
13 globalerror=0
14
15 keep=0
16
17 Usage() {
18     echo "Usage: testall.sh [options] [.snt files]"
19     echo "-k      Keep intermediate files"
20     echo "-h      Print this help"
21     exit 1
22 }
23
24 SignalError() {
25     if [ $error -eq 0 ] ; then
26         echo "FAILED"
27         error=1
28     fi
29     echo "$1"
30 }
31
32 # Compare <outfile> <reffile> <difffile>
33 # Compares the outfile with reffile. Differences, if any, written to
34 # difffile
35 Compare() {
36     generatedfiles="$generatedfiles $3"
37     echo diff -b $1 $2 ">" $3 1>&2
38     diff -b "$1" "$2" > "$3" 2>&1 || {
39         SignalError "$1 differs"
40         echo "FAILED $1 differs from $2" 1>&2
41     }
42 }
43
44 # Run <args>
45 # Report the command, run it, and report any errors
46 Run() {
47     echo $* 1>&2
48     eval $* || {
49         SignalError "$1 failed on $*"
50         return 1
51     }
52 }
53
54 Check() {
55     error=0
56     basename='echo $1 | sed 's/.*/\///'
57             's/.snt//'
58     reffile='echo $1 | sed 's/.snt$//'
59     basedir="`echo $1 | sed 's/\//[^\/]*$//`."
60
61     echo -n "$basename..."
62     echo 1>&2

```

```

63 echo "##### Testing $basename" 1>&2
64
65 generatedfiles=""
66
67 generatedfiles="$generatedfiles ${basename}.c.out output.c a.out" &&
68 Run "$SENET" "-c" "<" $1 &&
69 Run "gcc output.c" &&
70 Run "./a.out >" ${basename}.c.out &&
71 Compare ${basename}.c.out ${reffile}.out ${basename}.c.diff
72
73 # Report the status and clean up the generated files
74
75 if [ $error -eq 0 ] ; then
76 if [ $keep -eq 0 ] ; then
77     rm -f $generatedfiles
78 fi
79 echo "OK"
80 echo "##### SUCCESS" 1>&2
81 else
82 echo "##### FAILED" 1>&2
83 globalerror=$error
84 fi
85 }
86
87 while getopts kdps h c; do
88     case $c in
89 k) # Keep intermediate files
90     keep=1
91     ;;
92 h) # Help
93     Usage
94     ;;
95 esac
96 done
97
98 shift `expr $OPTIND - 1`
99
100 if [ $# -ge 1 ]
101 then
102     files=$@
103 else
104     files="tests/fail-*.snt tests/test-*.snt"
105 fi
106
107 for file in $files
108 do
109     case $file in
110 *test-*)
111         Check $file 2>> $globallog
112         ;;
113 *fail-*)
114         CheckFail $file 2>> $globallog
115         ;;
116 *)
```

```
117     echo "unknown file type $file"
118     globalerror=1
119     ;;
120   esac
121 done
122
123 exit $globalerror
```

./tests/test-add.out

```
1 5
```

./tests/test-add.snt

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6   print(2 + 3);
7   end;
8 }
9
10 }
```

./tests/test-and.out

```
1 true
2 false
3 false
4 false
```

./tests/test-and.snt

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6   print(True and True); print("\n");
7   print(False and False); print("\n");
8   print(True and False); print("\n");
9   print(False and True); print("\n");
10  end;
11 }
12
13 }
```

./tests/test-assert-func.out

```
1 true
2 true
3 false
4 false
```

./tests/test-assert-func.snt

```
1 @setup {
```

```

3 assert test() {
4     True;
5 }
6
7 assert test2() {
8     2 + 2 > 3;
9     4 - 2 < 0;
10    2 == 2;
11 }
12
13 assert test3(int x) {
14     if (x > 3) { True; } else { False; }
15 }
16
17 }
18
19 @turns {
20
21 func void begin() {
22     print(True); print("\n");
23     print(test()); print("\n");
24     print(test2()); print("\n");
25     print(test3(2)); print("\n");
26     end;
27 }
28
29 }

```

**./tests/test-assign.out**

```
1 2
```

**./tests/test-assign.snt**

```

1 @setup {
2
3 @turns {
4
5 func void begin() {
6     int x = 3;
7     x = 2;
8     print(x); print("\n");
9     end;
10 }
11
12 }

```

**./tests/test-basic-func-call.out**

```
1 5
2 6
```

**./tests/test-basic-func-call.snt**

```

1 @setup {
2
3 func int test(int x) {
```

```

4     return x + 1;
5 }
6
7 }
8
9 @turns {
10
11 func void begin() {
12     int a = 5;
13     print(a); print("\n");
14     a = test(a);
15     print(a); print("\n");
16     end;
17 }
18
19 }

```

**./tests/test-board-cells.out**

```

1 [ , , , , , , , , ]
2
3 false
4 -1
5 false

```

**./tests/test-board-cells.snt**

```

1 @setup {
2
3 group table(Rect(3, 3)) { };
4
5 }
6
7 @turns {
8
9 func void begin() {
10     group table a = table();
11
12     print(a.cells); print("\n");
13     print(a.cells[2]); print("\n");
14     print(a.occupied[5]); print("\n");
15     print(a.owns(2)); print("\n");
16     print(a.full()); print("\n");
17     a.remove(6);
18
19     end;
20 }
21
22 }

```

**./tests/test-board-place.out**

```

1 M
2 did not remove

```

**./tests/test-board-place.snt**

```

1 @setup {
2
3 group table(Rect(3, 3)) { };
4 group mark(Piece) {
5     func group mark __init__(str s) { this.s = s; return this; }
6 };
7
8 }
9
10 @turns {
11
12 func void begin() {
13     group table t = table();
14     group mark m = mark("M");
15     group mark x = mark("X");
16
17     t.place(m, 3);
18     t.place(x, 3); # doesn't place X since it is occupied
19     print(t.cells[3]); print("\n");
20
21     if (t.remove(2)) {
22         print("remove error\n");
23     } else {
24         print("did not remove\n");
25     }
26
27     end;
28 }
29
30 }
```

**./tests/test-board-toi.out**

```

1 0
2 1
3 2
4 3
5 4
6 5
7 6
8 7
9 8
10 4
11 0
12 1
```

**./tests/test-board-toi.snt**

```

1 @setup {
2
3 group table(Rect(3, 3)) { };
4
5 group myloop(Loop(5)) { };
6
7 }
8
```

```

9 @turns {
10
11 func void begin() {
12     group table a = table();
13     group myloop b = myloop();
14     int x = -1;
15     list[int] l = [x];
16
17     print(a.toi([0, 0])); print("\n");
18     print(a.toi([1, 0])); print("\n");
19     print(a.toi([2, 0])); print("\n");
20     print(a.toi([0, 1])); print("\n");
21     print(a.toi([1, 1])); print("\n");
22     print(a.toi([2, 1])); print("\n");
23     print(a.toi([0, 2])); print("\n");
24     print(a.toi([1, 2])); print("\n");
25     print(a.toi([2, 2])); print("\n");
26
27
28     print(b.toi([x])); print("\n");
29     print(b.toi([0])); print("\n");
30     print(b.toi([1])); print("\n");
31
32     end;
33 }
34
35 }
```

**./tests/test-board-tol.out**

```

1 [0, 0]
2 [1, 0]
3 [2, 0]
4 [0, 1]
5 [1, 1]
6 [2, 1]
7 [0, 2]
8 [1, 2]
9 [2, 2]
```

**./tests/test-board-tol.snt**

```

1 @setup {
2
3 group table(Rect(3, 3)) { };
4
5 }
6
7 @turns {
8
9 func void begin() {
10     group table a = table();
11
12     print(a.tol(0)); print("\n");
13     print(a.tol(1)); print("\n");
14     print(a.tol(2)); print("\n");
```

```

15 print(a.tol(3)); print("\n");
16 print(a.tol(4)); print("\n");
17 print(a.tol(5)); print("\n");
18 print(a.tol(6)); print("\n");
19 print(a.tol(7)); print("\n");
20 print(a.tol(8)); print("\n");

21
22 end;
23 }
24
25 }
```

**./tests/test-board.out**

```

1 3 3
2 5
3 10
4 20
```

**./tests/test-board.snt**

```

1 @setup {
2
3 group rect(Rect(3, 3)) { };
4 group line(Line(5)) { };
5 group loop(Loop(10)) { };
6 group hex(Hex(20)) { };
7
8 group rect r;
9 group line li;
10 group loop lo;
11 group hex h;
12
13 }
14
15 @turns {
16
17 func void begin() {
18     r = rect();
19     li = line();
20     lo = loop();
21     h = hex();
22
23     print(r.x); print(" "); print(r.y); print("\n");
24     print(li.x); print("\n");
25     print(lo.x); print("\n");
26     print(h.x); print("\n");
27     end;
28 }
29 }
```

**./tests/test-bool-literal.out**

```

1 true
2 false
```

```
./tests/test-bool-literal.snt
```

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     print(True);
7     print("\n");
8     print(False);
9     end;
10 }
11
12 }
```

```
./tests/test-break.out
```

```
1 5
```

```
./tests/test-break.snt
```

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     int a = 5;
7     while (a > 0) {
8         print(a); print("\n");
9         a = a - 1;
10        break;
11    }
12    end;
13 }
14
15 }
```

```
./tests/test-continue.out
```

```
1 4
2 2
```

```
./tests/test-continue.snt
```

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     int a = 5;
7     while (a > 0) {
8         if (a % 2 == 1) {
9             a = a - 1;
10            continue;
11        }
12        print(a); print("\n");
13        a = a - 1;
14    }
15 }
```

```
15     end;
16 }
17
18 }
```

./tests/test-div.out

```
1 5
2 4
```

./tests/test-div.snt

```
1 @setup {
2
3 @turns {
4
5 func void begin() {
6     print(15 / 3); print("\n");
7     print(14 / 3); print("\n");
8     end;
9 }
10
11 }
```

./tests/test-element.out

```
1 1
2 400
3 <Group A instance>
4 400
```

./tests/test-element.snt

```
1 @setup {
2
3 group A(Object) {
4     int x;
5     func group A __init__(int x) {
6         this.x = x;
7         return this;
8     }
9 };
10
11 }
12
13 @turns {
14
15 func void begin() {
16     list[int] x = [0, 1, 2, 3, 4, 5];
17
18     print(x[1]); print("\n");
19
20     pass(test_groups, 0);
21 }
22
23 func void test_groups() {
24     group A obj1 = A(100);
```

```

25 group A obj2 = A(200);
26 group A obj3 = A(300);
27 group A obj4 = A(400);
28 list[group A] x = [obj1, obj2, obj3, obj4];
29 group A y;

30
31 print(obj4.x); print("\n");
32 print(x[3]); print("\n");

33
34 y = x[3];
35 print(y.x); print("\n");

36
37 end;
38 }
39
40 }
```

**./tests/test-equal.out**

```

1 false
2 true
3 true
4 false
5 true
```

**./tests/test-equal.snt**

```

1 @setup {
2
3 group base(Object) {
4     int x;
5     int y;
6     func group base __init__(int x, int y) {
7         this.x = x;
8         this.y = y;
9         return this;
10    }
11 };
12
13 group A(base) {
14     int x;
15     int y;
16 };
17
18 group B(base) {
19     int x;
20     int y;
21 };
22
23 group C(base) {
24     int a;
25     int b;
26 };
27
28 group A a;
29 group A aa;
```

```

30 group B b;
31 group C c;
32 }
33 }
34
35 @turns {
36
37 func void begin() {
38     print(2 == 3); print("\n");
39     print(2 == 2); print("\n");
40     print(2 == 2 == True); print("\n");
41     print("abc" == "def"); print("\n");
42
43     a.__init__(100, 200);
44     aa.__init__(100, 200);
45     print(a == aa); print("\n");
46     end;
47 }
48 }
49 }
```

**./tests/test-exit.out**

```

1 test
./tests/test-exit.snt
```

```

1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     print("test\n");
7     exit();
8     print("did not exit\n");
9     end;
10 }
11
12 }
```

**./tests/test-for.out**

```

1 1
2 2
3 3
4 abc
5 def
6 ghi
7 jkl
8 2 3
9 4 7
10 6 11
11 1
12 [1, 2, 3]
13 4
14 [4, 5]
15 6
```

```

16 [6, 7]
./tests/test-for.snt

1 @setup {
2
3 group A(Object) {
4     int x;
5     int y;
6     func group A __init__(int x, int y) {
7         this.x = x;
8         this.y = y;
9         return this;
10    }
11
12    func int test() {
13        return this.x + this.y;
14    }
15 };
16
17 }
18
19 @turns {
20
21 func void begin() {
22     for (int x in {1, 2, 3}) {
23         print(x); print("\n");
24     }
25     pass(test_str, 0);
26 }
27
28 func void test_str() {
29     for (str s in {"abc", "def", "ghi", "jkl"}) {
30         print(s); print("\n");
31     }
32     pass(test_grp, 0);
33 }
34
35 func void test_grp() {
36     group A a1 = A(1, 2);
37     group A a2 = A(3, 4);
38     group A a3 = A(5, 6);
39     for (group A x in {a1, a2, a3}) {
40         print(x.y); print(" ");
41         print(x.test()); print("\n");
42     }
43     pass(test_list, 0);
44 }
45
46 func void test_list() {
47     list[int] a = [1, 2, 3];
48     list[int] b = [4, 5];
49     list[int] c = [6, 7];
50     int y;
51     for (list[int] x in {a, b, c}) {

```

```
52     y = x[0];
53     print(y); print("\n");
54     print(x); print("\n");
55 }
56 end;
57 }
58 }
59 }
```

**./tests/test-geq.out**

```
1 false
2 true
3 true
```

**./tests/test-geq.snt**

```
1 @setup {
2
3 @turns {
4
5 func void begin() {
6     print(2 >= 3); print("\n");
7     print(2 >= 2); print("\n");
8     print(2 >= 1); print("\n");
9     end;
10 }
11 }
12 }
```

**./tests/test-greater.out**

```
1 false
2 false
3 true
```

**./tests/test-greater.snt**

```
1 @setup {
2
3 @turns {
4
5 func void begin() {
6     print(2 > 3); print("\n");
7     print(2 > 2); print("\n");
8     print(2 > 1); print("\n");
9     end;
10 }
11 }
12 }
```

**./tests/test-group-cast.out**

```
1 2
2 3
```

**./tests/test-group-cast.snt**

```

1 @setup {
2
3 group A(Object) {
4     int x;
5     int y;
6     func group A __init__(int x, int y) {
7         this.x = x; this.y = y;
8         return this;
9     }
10
11    func int test() {
12        return this.x + this.y;
13    }
14};
15
16 group B(A) {
17     int z;
18     func group B __init__(int x, int y, int z) {
19         this.x = x; this.y = y; this.z = z;
20         return this;
21     }
22
23    func int test() {
24        return this.x + this.y + this.z;
25    }
26
27};
28
29}
30
31 @turns {
32
33 func void begin() {
34     group A a = B(1, 2, 3);
35     # group B b = A(1, 2);
36
37     print(a.y); print("\n");
38     print(a.test()); print("\n");
39     # print(b.test()); print("\n");
40     end;
41 }
42
43}

```

**./tests/test-group.out**

```

1 2
2 3

```

**./tests/test-group.snt**

```

1 @setup {
2
3 group A(Object) {
4     int x;
5

```

```

6     func group A __init__() {
7         this.x = 2;
8         return this;
9     }
10    };
11
12 group A obj;
13
14 }
15
16 @turns {
17
18 func void begin() {
19     # group A obj;
20     obj.__init__();
21     print(obj.x); print("\n");
22     obj.x = obj.x + 1;
23     print(obj.x); print("\n");
24     end;
25 }
26
27 }

```

./tests/test-hello.out

```
1 Hello World
```

./tests/test-hello.snt

```
1 @setup {
2 @turns { func void begin() { print("Hello World\n"); end; } }
```

./tests/test-if.out

```
1 if ok
2 if else ok
3 if elif ok
```

./tests/test-if.snt

```

1 @setup {
2
3 }
4
5 @turns {
6
7 func void begin() {
8     if (5 > 0) {
9         print("if ok"); print("\n");
10    }
11    if (2 < 0) {
12        print("if error"); print("\n");
13    } else {
14        print("if else ok"); print("\n");
15    }
16    if (2 < 0) {
17        print("if error"); print("\n");

```

```

18     } elif (3 > 0) {
19         print("if elif ok"); print("\n");
20     }
21
22     end;
23 }
24
25 }
```

**./tests/test-inherit-func-override.out**

```

1 1
2 2
3 3
4 20
5 30
6 6
7 10
8 8
9 10
10 10
11 10
12 true
```

**./tests/test-inherit-func-override.snt**

```

1 @setup {
2
3 group A(Object) {
4     int x;
5
6     func group A __init__(int x) {
7         this.x = x;
8         return this;
9     }
10
11     func int test() {
12         return this.x + 5;
13     }
14
15 };
16
17
18 group B(A) {
19     int y;
20
21     func int test() {
22         this.y = 10;
23         return this.y;
24     }
25 };
26
27 group C(A) {
28
29};
```

```

31 group D(B) {
32     func group D __init__(int x, int y) {
33         this.x = x;
34         this.y = y;
35         return this;
36     }
37 };
38
39 group E(D) {
40     func bool test(int x) {
41         this.x = x;
42         if (this.x > 0) {
43             return (True);
44         } else {
45             return (False);
46         }
47     }
48 };
49
50 group A obj;
51 group B obj2;
52 group C obj3;
53 group D obj4;
54 group E obj5;
55
56 }
57
58 @turns {
59
60 func void begin() {
61     int y;
62
63     obj.__init__(1);
64     obj2.__init__(2);
65     obj3.__init__(3);
66     obj4.__init__(20, 30);
67     obj5.__init__(0, 1);
68
69     print(obj.x); print("\n");
70     print(obj2.x); print("\n");
71     print(obj3.x); print("\n");
72     print(obj4.x); print("\n");
73     print(obj4.y); print("\n");
74
75     print(obj.test()); print("\n");
76     print(obj2.test()); print("\n");
77     print(obj3.test()); print("\n");
78     print(obj4.test()); print("\n");
79
80     print(obj4.y); print("\n");
81
82     print(obj5.test()); print("\n");
83     print(obj5.test(5)); print("\n");
84 }
```

```
85     end;
86 }
87 }
88 }
89 }
```

**./tests/test-inherit-func.out**

```
1 1
2 10
3 21
4 2
5 3
6 15
7 22
```

**./tests/test-inherit-func.snt**

```
1 @setup {
2
3 group A(Object) {
4     int x;
5
6     func group A __init__() {
7         this.x = 1;
8         return this;
9     }
10
11    func int testA(int x) {
12        return x + 5;
13    }
14
15    func int testB() {
16        return this.x + 20;
17    }
18 };
19
20
21 group B(A) {
22     int y;
23     int x;
24
25     func group B __init__(int a, int b) {
26         this.x = a;
27         this.y = b;
28         return this;
29     }
30 };
31
32
33 group A obj;
34 group B obj2;
35
36 }
37
38 @turns {
```

```

39
40 func void begin() {
41     int y;
42
43     obj.__init__();
44     obj2.__init__(2, 3);
45
46     print(obj.x); print("\n");
47     print(obj.testA(5)); print("\n");
48     print(obj.testB()); print("\n");
49
50     print(obj2.x); print("\n");
51     print(obj2.y); print("\n");
52     print(obj2.testA(10)); print("\n");
53     print(obj2.testB()); print("\n");
54
55     end;
56 }
57
58 }
```

**./tests/test-inherit-three-deg.out**

```

1 1
2 2
3 3
4 6
5 7
6 8
```

**./tests/test-inherit-three-deg.snt**

```

1 @setup {
2
3 group A(Object) {
4     int x;
5
6     func group A __init__(int x) {
7         this.x = x;
8         return this;
9     }
10
11    func int test() {
12        return this.x + 5;
13    }
14
15 };
16
17
18 group B(A) {
19     int y;
20 };
21
22 group C(B) {
23     int z;
24 };
```

```

25
26 group A obj;
27 group B obj2;
28 group C obj3;
29
30 }
31
32 @turns {
33
34 func void begin() {
35     int y;
36
37     obj.__init__(1);
38     obj2.__init__(2);
39     obj3.__init__(3);
40
41     print(obj.x); print("\n");
42     print(obj2.x); print("\n");
43     print(obj3.x); print("\n");
44
45     print(obj.test()); print("\n");
46     print(obj2.test()); print("\n");
47     print(obj3.test()); print("\n");
48
49     end;
50 }
51
52 }

```

**./tests/test-inherit.out**

```

1 2
2 3
3 1
4 1
5 2
6 1
7 2
8 5
9 6

```

**./tests/test-inherit.snt**

```

1 @setup {
2
3 group A(Object) {
4     int x;
5
6     func group A __init__() {
7         this.x = 1;
8         return this;
9     }
10 };
11
12
13 group B(A) {

```

```

14     int y;
15
16     func group B __init__(int a, int b) {
17         this.x = a;
18         this.y = b;
19         return this;
20     }
21 };
22
23 group C(A()) {
24     int a;
25     int b;
26 };
27
28 group D(B(1, 2)) {
29     int a;
30 };
31
32 group B obj;
33 group C obj2;
34 group D obj3;
35 group D obj4;
36 group B obj5;
37
38 }
39
40 @turns {
41
42 func void begin() {
43     obj.__init__(2, 3);
44     obj2.__init__();
45     obj3.__init__();
46     obj4.__init__();
47     obj5.__init__(5, 6);
48     print(obj.x); print("\n");
49     print(obj.y); print("\n");
50     print(obj2.x); print("\n");
51     print(obj3.x); print("\n");
52     print(obj3.y); print("\n");
53     print(obj4.x); print("\n");
54     print(obj4.y); print("\n");
55     print(obj5.x); print("\n");
56     print(obj5.y); print("\n");
57     end;
58 }
59
60 }

```

**./tests/test-init.out**

```

1 5
2 hello
3 world
4 world
5 !!!

```

```
./tests/test-init.snt
```

```
1 @setup {
2
3 group A(Object) {
4     str x;
5
6     func group A __init__(str x) {
7         this.x = x;
8         return this;
9     }
10}
11};
12
13}
14
15@turns {
16
17 func void begin() {
18     int a = 2 + 3;
19     int b = a = 5;
20     group A g1 = A("hello");
21     group A g2;
22     group A g3 = g2 = A("world");
23
24     print(a); print("\n");
25     print(g1.x + "\n");
26     print(g2.x + "\n");
27     print(g3.x + "\n");
28     g2.x = "!!!";
29     print(g2.x + "\n");
30
31     end;
32 }
33}
34}
35}
```

```
./tests/test-int-literal.out
```

```
1 3
```

```
./tests/test-int-literal.snt
```

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     print(3);
7     end;
8 }
9}
10}
```

```
./tests/test-leq.out
```

```
1 true
2 true
3 false
```

./tests/test-leq.snt

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     print(2 <= 3); print("\n");
7     print(2 <= 2); print("\n");
8     print(2 <= 1); print("\n");
9     end;
10 }
11
12 }
```

./tests/test-less.out

```
1 true
2 false
3 false
```

./tests/test-less.snt

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     print(2 < 3); print("\n");
7     print(2 < 2); print("\n");
8     print(2 < 1); print("\n");
9     end;
10 }
11
12 }
```

./tests/test-list-literal-emptylist.out

```
1 []
2 []
```

./tests/test-list-literal-emptylist.snt

```
1 @setup {
2
3 }
4
5 @turns {
6
7 func void begin() {
8     list[void] x = [];
9
10    print([]); print("\n");
11    print(x); print("\n");
```

```

12     end;
13 }
14 }
15 }
16 }
```

**./tests/test-list-literal.out**

```

1 []
2 [1, 2, 3]
3 [3, 4, 5]
4 [abc, def]
5 [hello world!, 123]
6 [true, false, true]
7 [1, 2, 3]
8 [x, y, hello world!]
9 [obj1, obj2, obj3]
10 [obj1, obj2, obj3]
```

**./tests/test-list-literal.snt**

```

1 @setup {
2
3 group A(Object) {
4     str s;
5     func group A __init__(str s) {
6         this.s = s;
7         return this;
8     }
9     func str __repr__() {
10        return this.s;
11    }
12 };
13
14 }
15
16 @turns {
17
18 func void begin() {
19     int x = 1; int y = 2; int z = 3;
20     str s = "hello world!";
21     list[int] a = [1, 2, 3];
22     list[str] b = ["x", "y", s];
23
24     group A obj1 = A("obj1");
25     group A obj2 = A("obj2");
26     group A obj3 = A("obj3");
27     list[group A] g = [obj1, obj2, obj3];
28
29     print([]); print("\n");
30     print([x, y, z]); print("\n");
31     print([3, 4, 5]); print("\n");
32     print(["abc", "def"]); print("\n");
33     print([s, "123"]); print("\n");
34     print([True, False, True]); print("\n");
35     print(a); print("\n");
```

```
36     print(b); print("\n");
37
38     print([obj1, obj2, obj3]); print("\n");
39     print(g); print("\n");
40
41     end;
42 }
43 }
44 }
45 }
```

**./tests/test-min-not.out**

```
1 5
2 false
3 -3
```

**./tests/test-min-not.snt**

```
1 @setup {
2
3 @turns {
4
5 func void begin() {
6     print(5); print("\n");
7     print(not True); print("\n");
8     print(-3); print("\n");
9     end;
10 }
11 }
12 }
```

**./tests/test-mod.out**

```
1 1
2 2
3 0
```

**./tests/test-mod.snt**

```
1 @setup {
2
3 @turns {
4
5 func void begin() {
6     print(5 % 2); print("\n");
7     print(5 % 3); print("\n");
8     print(5 % 5); print("\n");
9     end;
10 }
11 }
12 }
```

**./tests/test-mult-turn.out**

```
1 true
2 false
```

**./tests/test-mult-turn.snt**

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     print(True); print("\n");
7     pass(ending, 2);
8 }
9
10 func void ending() {
11     print(False); print("\n");
12     end;
13 }
14
15 }
```

**./tests/test-mult.out**

```
1 -15
```

**./tests/test-mult.snt**

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     print(5 * -3);
7     end;
8 }
9
10 }
```

**./tests/test-neq.out**

```
1 true
2 false
3 false
4 true
5 true
6 false
7 false
```

**./tests/test-neq.snt**

```
1 @setup {
2
3 group A(Object) {
4     int x;
5     int y;
6
7     func group A __init__(int x, int y) {
8         this.x = x;
9         this.y = y;
10        return this;
11    }
12};
```

```

13
14 group B(A(2, 3)) {
15
16 };
17
18 group A a;
19 group A aa;
20 group B b;
21 group B bb;
22
23 }
24
25 @turns {
26
27 func void begin() {
28     print(2 != 3); print("\n");
29     print(2 != 2); print("\n");
30     print("abc" != "abc"); print("\n");
31     print("abcd" != "abc"); print("\n");
32
33     a.___init__(100, 200);
34     aa.___init__(200, 300);
35     b.___init__();
36     bb.___init__();
37
38     print(a != aa); print("\n");
39     print(b != b); print("\n");
40     print(b != bb); print("\n");
41     end;
42 }
43
44 }

```

**./tests/test-none.out**

```

1 None
2 true

```

**./tests/test-none.snt**

```

1 @setup {
2
3 group A(Object) {
4
5     func group A ___init__() { return this; }
6
7 };
8
9 group A test;
10
11 }
12
13 @turns {
14
15 func void begin() {
16

```

```

17     print(None); print("\n");
18     print(None == None); print("\n");
19
20     end;
21 }
22
23 }
```

**./tests/test-or.out**

```

1 true
2 false
3 true
4 true
```

**./tests/test-or.snt**

```

1 @setup {
2
3 @turns {
4
5 func void begin() {
6     print(True or True); print("\n");
7     print(False or False); print("\n");
8     print(True or False); print("\n");
9     print(False or True); print("\n");
10    end;
11 }
12
13 }
```

**./tests/test-pass-group.out**

```

1 X
2 false
```

**./tests/test-pass-group.snt**

```

1 @setup {
2
3 group Mark(Piece) {
4     func group Mark __init__(str s) {
5         this.s = s;
6         return this;
7     }
8 };
9
10 func str test(group Mark m) {
11     return m.s;
12 }
13
14 assert check(group Mark m) {
15     m.s == "M";
16 }
17
18 }
```

```

20 @turns {
21
22 func void begin() {
23     group Mark x = Mark("X");
24     print(test(x)); print("\n");
25     print(check(x)); print("\n");
26     end;
27 }
28
29 }
```

**./tests/test-pass-list.out**

```

1 3
2 true
```

**./tests/test-pass-list.snt**

```

1 @setup {
2
3 list[int] mylist;
4
5 func int test(list[int] x) {
6     return x[2];
7 }
8
9 assert check(list[int] x) {
10    x[0] == 1;
11 }
12
13 }
14
15 @turns {
16
17 func void begin() {
18     mylist = [1, 2, 3];
19     print(test(mylist)); print("\n");
20     print(check(mylist)); print("\n");
21     end;
22 }
23
24 }
```

**./tests/test-piece.out**

```

1 X
2 O
```

**./tests/test-piece.snt**

```

1 @setup {
2
3 group mark(Piece) {
4     str symbol;
5     func group mark __init__(str s) {
6         this.fixed = True;
7         this.symbol = s;
```

```

8         return this;
9     }
10
11    func str __repr__() {
12        return this.symbol;
13    }
14};
15
16 group mark x;
17 group mark o;
18
19}
20
21 @turns {
22
23 func void begin() {
24     x = mark("x");
25     o = mark("o");
26
27     print(x); print("\n");
28     print(o); print("\n");
29
30     end;
31 }
32
33}

```

**./tests/test-place.out**

```

1 true
2 false

```

**./tests/test-place.snt**

```

1 @setup {
2
3 group B(Rect(2, 2)) { };
4 group Mark(Piece) {
5     func group Mark __init__(str s) {
6         this.s = s;
7         return this;
8     }
9 };
10
11 group B brd;
12
13
14
15}
16
17 @turns {
18
19 func void begin() {
20     group Mark m = Mark("M");
21     brd = B();
22

```

```

23     print(m >> brd >> [1, 1]); print("\n");
24     print(m >> brd >> [1, 1]); print("\n");
25
26     end;
27 }
28
29 }
```

**./tests/test-print.out**

```

1 1
2 abc
3 true false
4 <Group A instance>
5 <Group A instance>
6 Hello from class B!
7 Hello from class B!
8 <Group D instance>
```

**./tests/test-print.snt**

```

1 @setup {
2
3 group A(Object) {
4     func group A __init__() { return this; }
5 };
6
7 group B(A) {
8     func str __repr__() {
9         return "Hello from class B!";
10    }
11 };
12
13 group C(B) {
14 };
15
16 group D(A) {
17 };
18
19
20 group A no_repr;
21 group B yes_repr;
22 group C inherited_repr;
23 group D inherited_repr_builtin;
24
25
26 }
27
28 @turns {
29
30 func void begin() {
31
32     print(1, "\n");
33     print("abc\n");
34     print(True, " ", False, "\n");
35     print(no_repr, "\n");
```

```

36     print(no_repr.__repr__(), "\n");
37     print(yes_repr, "\n");
38     print(inherited_repr, "\n");
39     print(inherited_repr_builtin, "\n");
40
41     end;
42 }
43 }
44 }
```

**./tests/test-rand.out**

```
1 success calling rand
```

**./tests/test-rand.snt**

```

1 @setup {
2
3 @turns {
4
5 func void begin() {
6     int x = rand();
7     print("success calling rand\n");
8     end;
9 }
10 }
11 }
```

**./tests/test-remove.out**

```
1 false
2 true
```

**./tests/test-remove.snt**

```

1 @setup {
2
3 group B(Rect(2, 2)) { };
4 group Mark(Piece) {
5     func group Mark __init__(str s) {
6         this.s = s;
7         return this;
8     }
9 };
10
11 group B brd;
12
13
14 }
15 }
16
17 @turns {
18
19 func void begin() {
20     group Mark m = Mark("M");
21     brd = B();
```

```

23     brd.place(m, brd.toi([0, 1]));
24
25     # print(brd.toi([1, 1])); print("\n");
26     print(brd << [1, 1]); print("\n");
27     print(brd << [0, 1]); print("\n");
28
29     end;
30 }
31
32 }
```

**./tests/test-scope.out**

```

1 6
2 25
```

**./tests/test-scope.snt**

```

1 @setup {
2     int x = 4;
3
4     group A(Object) {
5         int x;
6
7         func group A __init__(int x) {
8             this.x = x;
9             return this;
10        }
11
12        func void y() {
13
14        }
15    };
16
17 }
18
19 @turns {
20
21 func void begin() {
22     int x = 6;
23     group A a = A(25);
24
25     print(x); print("\n");
26     print(a.x); print("\n");
27
28     end;
29 }
30
31 }
```

**./tests/test-stoi.out**

```

1 12345
```

**./tests/test-stoi.snt**

```

1 @setup { }
```

```
2
3 @turns {
4
5 func void begin() {
6     int x = stoi("12345");
7
8     print(x); print("\n");
9
10    end;
11 }
12
13 }
```

**./tests/test-str-lit.out**

```
1 abcdef
```

**./tests/test-str-lit.snt**

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     str s = "abcdef";
7     print(s); print("\n");
8     end;
9 }
10
11 }
```

**./tests/test-strcat.out**

```
1 abcdef
```

**./tests/test-strcat.snt**

```
1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     str a = "abc";
7     str b = "def";
8
9     print(a + b); print("\n");
10
11    end;
12 }
13
14 }
```

**./tests/test-sub.out**

```
1 -1
```

**./tests/test-sub.snt**

```

1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     print(2 - 3);
7     end;
8 }
9
10 }
```

**./tests/test-turn-end.out**

```
1 passing
```

**./tests/test-turn-end.snt**

```

1 @setup { int x = 0; }
2
3 @turns {
4
5 func void begin() {
6     if (PLAYER_ON_MOVE == 1) {
7         end;
8     } else {
9         print("passing\n");
10        pass(ending, 1);
11    }
12 }
13
14 func void ending() {
15     pass (begin, 1);
16 }
17
18 }
```

**./tests/test-while.out**

```
1 5
2 4
3 3
4 2
5 1
```

**./tests/test-while.snt**

```

1 @setup { }
2
3 @turns {
4
5 func void begin() {
6     int a = 5;
7     while (a > 0) {
8         print(a); print("\n");
9         a = a - 1;
10    }
11    end;
```

```

12 }
13 }
14 }

./types.ml

1 type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater |
2     Geq
3         | Mod | And | Or
4
5 type bool_lit =
6     True
7     | False
8
9 type t =
10    Int
11    | Bool
12    | Str
13    | Void
14    | List_t of t
15    | Group of string * group_decl option (* 2nd value filled in by Cast *)
16
17 and list_lit =
18     Elems of expression list * string
19     | EmptyList
20
21 and var_decl = {
22     vname : string;
23     vtype : t;
24     vinit : expression option;
25     vloop : bool
26 }
27
28 and expr_detail =
29     IntLiteral of int * string
30     | StrLiteral of string * string
31     | ListLiteral of list_lit
32     | BoolLiteral of bool_lit * string
33     | VoidLiteral
34     | Field of field_expr
35     | Binop of expression * op * expression
36     | Assign of field_expr * expression
37     | Call of var_decl option * func_decl * expression list
38     | Element of expression * expression
39     | Uminus of expression
40     | Not of expression
41     | Noexpr
42     | Remove of expression
43     | Place of expression
44
45 and expression =
46     expr_detail * t
47
48 and field_expr =
49     Var of var_decl

```

```

49 |     Attrib of var_decl * var_decl
50 |     Fun of func_decl
51 |     Method of var_decl * func_decl
52 |     Grp of group_decl
53 |     This
54
55 and statement =
56 |     Block of symbol_table * statement list
57 |     Expression of expression
58 |     Return of expression
59 |     Break
60 |     Continue
61 |     End
62 |     Pass of func_decl * expression
63 |     If of expression * statement * expression option * statement
64 |     For of var_decl * expression list * statement
65 |     While of expression * statement
66
67 and basic_func_decl = {
68     ftype : t;
69     fname : string;
70     formals : var_decl list;
71     locals : var_decl list;
72     body : statement list;
73     turns_func : bool;
74     group_method : string;
75     f_is_builtin : bool
76 }
77
78 and assert_decl = {
79     fname : string;
80     aformals : var_decl list;
81     alocals : var_decl list;
82     abody : statement list;
83     a_turns_func : bool;
84     a_group_method : string;
85     a_is_builtin : bool
86 }
87
88 and func_decl =
89     BasicFunc of basic_func_decl
90     | AssertFunc of assert_decl
91
92 and group_decl = {
93     gname : string;
94     extends : group_decl option;
95     par_actualls : expression list option;
96     attributes : var_decl list;
97     methods : func_decl list;
98 }
99
100 and setup = var_decl list * func_decl list * group_decl list
101
102 and turns = func_decl list

```

```

103
104 and program = setup * turns
105
106
107 and symbol_table = {
108   parent : symbol_table option;
109   mutable variables : var_decl list;
110   mutable functions : func_decl list;
111   mutable groups : group_decl list;
112   mutable turns : string list;
113   mutable ll_count : int;
114   mutable elem_count : int
115 }
116
117 type partial_group_table = {
118   group_name : string;
119   par : group_decl option;
120   symbols : symbol_table;
121 }
122
123 type translation_environment = {
124   scope : symbol_table;
125   return_type : t;
126   in_loop : bool;
127   partial_group_info : partial_group_table option
128 }
```