String Manipulation and Probability

(SMAP)

Final Project Report

Emily Melissa Sillars (ems2331)

Swapnil Paliwal (sp3991)

Andrew Magid (aam2302)

Tushar Arora (ta2673)

Table of Contents

Chapter 1

Introduction

String Manipulation and Probability (SMAP) is an imperative and statically typed language with C-like syntax. The goal of this project is to add in a special type, along with the basic C features, which reflects probability and so the language extends the functionality by adding a new and a very unique data type, called the prob type which in a limited manner incorporates this feature of probability into SMAP. The prob type is polymorphic which means any type can be made into a probabilistic type the prob type identifier and it models a discrete distribution by matching probabilities to values. The extended goal of this feature is to get incorporated into games where attacks, objects and other random things that the game picks can be based on this probability feature rather than making it completely random.

Chapter 2
Language Tutorial

2. 1 Getting Started

SMAP requires Ocaml and LLVM libraries to build and execute.

Mac OS Distribution

(TO INSTALL BREW)

```
$ curl -fsSL -o install.sh https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh
$ /bin/bash install.sh
```

(TO INSTALL OPAM AND LLVM)

```
$ brew install opam
$ brew install llvm
$ opam install llvm
```

Ubuntu Distribution

(TO INSTALL OPAM AND LLVM)

```
$ sudo apt-get install opam
$ sudo apt-get install llvm
$ opam install llvm
```

Other Linux Distributions:

```
(TO INSTALL BREW)
$ curl -fsSL -o install.sh https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh
$ /bin/bash install.sh
(TO INSTALL OPAM AND LLVM)
$ brew install llvm
$ opam depext conf-llvm.6.0.0
$ opam install llvm
$ export PATH=/usr/local/opt/llvm@6/bin:$PATH
```

2. 2 Building SMAP

Docker Environment setup

```
(TO INSTALL DOCKER)
$ apt install docker.io
(ADD YOURSELF TO THE DOCKER GROUP)
$ sudo usermod -aG docker <username>
(TEST INSTALLATION)
$ docker run hello-world
(RUN INSIDE THE CLONED DIRECTORY)
$ docker run –rm -it -v ‘pwd’ :/home/smap -w=/home/smap/plt
```

Clone and Build SMAP

(CLONE REPO)

```
$ git clone https://github.com/magidandrew/smap.git
```

(RUN THE DOCKER INSIDE SMAP)

```
$ cd smap; docker run –rm -it -v ‘pwd’ :/home/smap -w=/home/smap/plt
```

(MAKE FILES)

```
$ make all
```

(RUN THE TEST FILES)

```
$ ./testall.sh
```

(CLEAN THE EXECUTABLES)

```
$ make clean
```

2. 2 Running a single SMAP program

```
$ ./testfile.sh <file path>
```

Chapter 3
Language Reference Manual

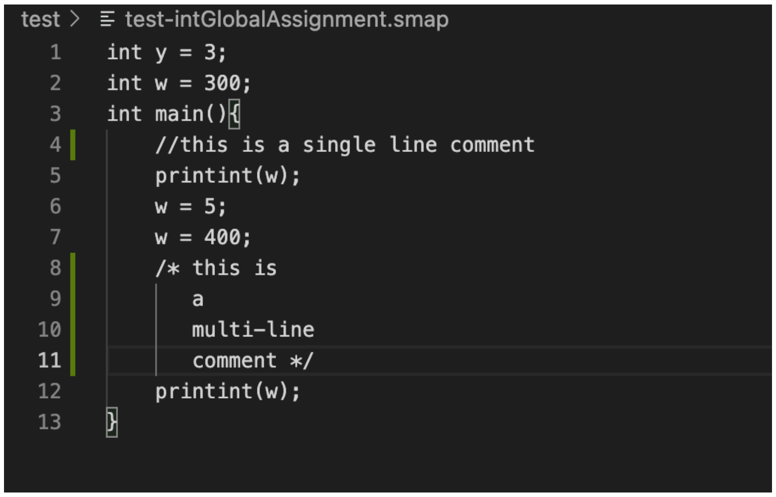
Lexical Conventions

3. 1 Tokens

There are five classes of tokens: identifiers, keywords, string literals, operators and other separators. Other tokens are ignored because they act as separators, i.e. they separate the tokens. Blanks, horizontal and vertical tabs, newlines, formfeeds and comments are part of the group of tokens that are ignored while parsing the program.

3. 2 Comments

Single-line comments are denoted by the characters // which ignore all text following them until a newline is met. Multi-line comments are denoted by the characters /* and */ which introduce and terminate the multi-line comment, respectively. Everything between the paired multi-line comment characters is ignored. Comments do not nest and do not occur in literals.



```
int y = 3;
int w = 300;
int main(){
    //this is a single line comment
    printint(w);
    w = 5;
    w = 400;
    /* this is
       a
       multi-line
       comment */
    printint(w);
}
```

3. 3 Identifiers

Identifiers are a sequence of letters and digits. SMAP is case-sensitive since lowercase and uppercase letters are distinguished to allow for camel case, and the underscore _ is included to allow for snake case. Identifiers must begin with an underscore or letter.

`okayToUseCamelCase`	✓
`okay_to_use_snake_case`	✓
`OkayToStartWithCapital`	✓
`_alsoOkay`	✓
`9_not_okay_to_start_with_number`	✗
`&not_okay_non_alphanumeric-=@#%`	✗

3. 4 Keywords

The following identifiers are reserved keywords in SMAP:

`bool`	`elif`	`if`	`prob`	`void`
	`else`	`int`	`return`	`while`
`case`	`false`	`list`	`string`	`continue`
`char`	`float`	`null`	`switch`	
	`for`		`true`	

3. 5 Data Types (Primitive)

Data Types	**Description**
int	A sequence of digits in decimal format. Cannot start with a zero.
bool	Either **`true`** or **`false`**, implemented as integers 0 or 1.
char	A text character in single quotes, for example **`‘a’`**. Escape characters **`‘\n’`**, **`‘\0’`** , **`‘\\’`**, **`‘\’’`**, **`‘\”’`**, represent new line, null terminator, regular slash, regular single quote and regular double quote respectively.
float	A sequence of digits in decimal format containing a single . for the decimal point.
string	A sequence of characters in double quotes, for example, **`“hello”`**. String literals are implemented as dynamic lists of characters.

3. 6 Operators

3.6.1 Binary Operators

There are four basic categories of binary operators, as well as concatenation.

Arithmetic:	**+, -, *, /**
Bitwise:	**&,|,^, >>, <<**
Comparison:	**>,<, >=, <=, ==,** and **!=**
Logical:	**||, &&**

These expressions group left to right and follow the general form

```
type expr =
   Binop of expr * binary_op * expr
```

// where **binop_op** is a binary operator and

// the expressions on either side of the operation are **of the same type**

```
 expr PLUS   expr          { Binop($1, Add, $3)        }
| expr MINUS  expr          { Binop($1, Sub,  $3)       }
| expr TIMES  expr          { Binop($1, Mul, $3)        }
| expr DIVIDE expr          { Binop($1, Div, $3)        }
| expr COMPEQ expr          { Binop($1, CompEq, $3)     }
| expr COMPLT expr          { Binop($1, CompLt, $3)     }
| expr COMPLEQ expr         { Binop($1, CompLeq, $3)    }
| expr COMPGT expr          { Binop($1, CompGt, $3)     }
| expr COMPGEQ expr         { Binop($1, CompEq, $3)     }
| expr COMPNEQ expr         { Binop($1, CompNeq, $3)    }
| expr RSHIFT expr          { Binop($1, RShift, $3)     }
| expr LSHIFT expr          { Binop($1, LShift, $3)     }
| expr BITAND expr          { Binop($1, BitAnd, $3)     }
| expr BITOR expr           { Binop($1, BitOr, $3)      }
| expr XOR    expr          { Binop($1, Xor, $3)        }
```

The only exceptions to the same type rule requirement are arithmetic operations between floats and ints (in which ints are implicitly cast to floats), concatenation operator between a string and float, string and int, or string and char (where floats, ints, and chars are implicitly cast to strings), and right and left shifts on lists for deleting elements, which require an argument of type int on the right hand side.

3.6.2 Assignment Operator

The assignment operator is a special case of binary operators.

```
/* assignment expressions (special case of binary)     */
| expr ASSIGN expr          { Assign($1, Equal, $3)       }
| expr ADDEQUAL expr        { Assign($1, PlusEqual,$3)    }
| expr MINUSEQUAL expr      { Assign($1, MinusEqual, $3)  }
| expr TIMESEQUAL expr      { Assign($1, TimesEqual, $3)  }
| expr DIVEQUAL expr        { Assign($1, DivEqual, $3)    }
```

3.6.3 Unary Operators

```
Unop of unary_op * expr
type unary_op = BitNot | Not | Bang | Octothorpe | Neg

| MINUS expr                            { Unop(Neg, $2)     }
| BITNOT expr                           { Unop(BitNot, $2)  }
| NOT expr                              { Unop(Not, $2)     }
| expr NOT                              { Unop(Bang, $1)    }
| expr OCTOTHORPE                       { Unop(Octothorpe, $1)  }
```

Unary expressions group left to right and are of the following forms:

~ *expr*   // BitNot; flip all the bits

! *expr*   // Not; a logical not; non zero values become 0, zeroes become 1

expr!   // Bang; evaluate a probability type

expr#   //Octothorpe; return the list of probabilities contained in the probability type

-expr   //Neg; denotes a negative number

3. 7 Types (Built-in types)

SMAP supports two fundamental types: primitive types and built-in data types.

Primitive types :   int, float, bool, string and char.

Built-in types: list and prob

```
type typ = Int | Bool | Float | Void | Char | String | Prob | List
type typ_name = typ list
type bind = typ_name * string
```

The SMAP AST represents types as a list of built-in and primitive types. Type checking in semant.ml rules out invalid sequences of types. For example, [Char,Char] is an invalid sequence of types.

3.7.1 Built-in types - Probability

```
expr PROBCOLON expr  { ProbColon($1,$3)   }
```

The probability type is polymorphic; any type can be made into a probabilistic type with the **`prob`** type qualifier. The probability type models a discrete distribution by matching probabilities to values. Probability types are initialized with a pair of non-empty lists separated by a colon. The left hand side list represents probabilities and must be of type `list float`. The right hand side list represents values and must be of type list <type>, where <type> is the type declared after the **`prob`** type qualifier in the declaration. Once initialized, the list of values inside the probability type is immutable.

```
prob type-identifier id = list float : list type-identifier;
//   ^where the two type-identifiers must match^!

prob int someNumbers = list float : list int;
// note that int matches int!

prob int randNum = [0,25, 0.25, 0.50] : [1, 87, 6];
// ^ example with concrete types
```

Accessing Fields

```
expr LENGTH       { Length($1) }
```

The "length" of a probability type returns the minimum of the lengths of its two lists and is retrieved with **`.length`**:

```
prob int aNum = [0.3, 0.4, 0.3 ] : [7, 5]; // aNum#.length is 3, and
                                           // aNum[0,1].length is 2
int len = aNum.length;                     // len is min(3,2) = 2
```

The **`#`** operator extracts the list of probabilities:

```
prob int randNum = [0,25, 0.25, 0.50] : [1, 87, 6];
list float vals = randNum#;                 // get the list of probabilities
```

```
 .                                            // inside randNum
 // vals == randNum# ==  [0,25, 0.25, 0.50]  // evaluates to true
```

The **!** pulls out a value based on the underlying distribution:

```
int num = randNum!    //num os 1, 87 or 6
```

Note that if the length of the two lists are unequal, the value-list or probability-list is treated as "trimmed" to the minimum of the two list lengths. When the probability list is treated as "trimmed" its remaining elements are treated as normalized to sum to one.

```
randNum = [0.60,0.40] : randNum[0,1];   // [0.60,0.40]:[1, 87, 6]
num = randNum!                          // num3 is 1 or 87 (6 trimmed off)
randNum = [0.20, 0.25, 0.55]: [4, 2];   // randNum# is treated like
[0.45,0.55]
num = randNum!;                         // num is 4 or 2
```

```
randNum = [0.60,0.40] : randNum[0,1];   // [0.60,0.40]:[1, 87, 6]
num = randNum!                          // num3 is 1 or 87 (6 trimmed off)
randNum = [0.20, 0.25, 0.55]: [4, 2];   // randNum# is treated like [0.45,0.55]
num = randNum!;                         // num is 4 or 2
```

Modifying Probability Types with Transformations

We use transformations to safely modify a probability type's underlying distribution. Transformations use the arithmetic operators **+, - , *** and / as part of a "zipwith" function, and then normalize the resulting list of floats to represent a new distribution. Transformation expressions can be generalized into four forms, with a probability type required on the left side of the operation.

probType OP anotherProbType

probType OP list float

probType OP list int

probType OP float

probType OP int

Let’s take a look at these transformations in action.

Ex. probType OP anotherProbType:

```
prob int  num    = [0.25, 0.15, 0.15, 0.45] : [6, 87, 4, 302] ;
prob char letter = [0.25, 0.75 ] : [‘a’,’b’];
letter = letter * num; // letter is [0.36,0.64] : [‘a’,’b’]
```

How did we get the result [0.36,0.64] : [‘a’,’b’]? First, letter and num’s values lists are immutable. Since the letter is on the left hand side of the transformation, its values are chosen to appear in the resulting probability type. **Note that this means a transformation between two probability types is associative but not commutative!** Next, letter# and num# are zipped with operator *. Transformations use the probability type’s length (the minimum of its two list lengths) when performing zipwith, so the expression can be broken down as follows.

```
letter * num
zipwith letter# num# *
zipwith [0.25, 0.75]  [0.25, 0.15, 0.15, 0.45] *
[0.25*0.25, 0.75*0.15]
[0.0625, 0.1125]
```

Then the transformation normalizes the probabilities,

N([0.0625, 0.1125]) = [0.0625/0.157, 0.1125/0.157] = [0.36,0.64]

and final result is

[0.36,0.64] : [‘a’,’b’]

Ex. probType OP list float:

```
prob int changedNum = Num * [0.2, 0.3] ;    // [0.53, 0.047]:Num[0,1]
// zipwith Num# letter# *
// zipwith [0.25, 0.15, 0.15, 0.45] [0.2, 0.3] *
// [0.25*0.2, 0.15*0.3]
// [0.05, 0.045]
// Normalize([0.05, 0.045]) = [0.05/0.095, 0.045/0.095] = [0.53, 0.047]
// final result is [0.53, 0.047] : [6, 87, 4, 302]
```

Since integers are implicitly converted to floats in float op int or int op float expressions, probType OP list int expressions are implicitly converted expressions of probType OP list float.

Ex. probType OP float:
Expressions of **probType OP float** are implicitly converted to expressions of **probType OP list float,** where the float transforming the probability is promoted to a list of equal length with a single repeated value.

```
letter = letter + 3     // [0.48, 0.52]:['a', 'b']
// since letter.length = 2, 3 gets promoted to [3, 3]
// now we can do zip [0.36, 0.64] [3,3] + = [3.36, 3.64]
// Normalize([3.36, 3.64]) = [3.36/7, 3.64/7,] = [0.48, 0.52]
// final result is [0.48, 0.52]:['a', 'b']
```

Since integers are implicitly converted to floats in float op int or int op float expressions, probType OP int expressions are implicitly converted expressions of probType OP float.

Normalization

When a probability type is first initialized, or whenever a probability type is transformed, its internal probability list is normalized to ensure it represents a valid distribution. There are three steps in the normalization process.

1) Check both the probability list and the value list are non-empty. If this check fails, throw an error.
2) Check the probability list contains only non-negative elements, and that the sum of the elements > 0. If this check fails, throw an error.
3) Divide each element by the sum

3.7.2 Primitive types - List

A list is a collection of homogenous mutable data, i.e. all of the elements inside the list would be of the same type and can be modified. A list is also dynamic, i.e. the list size is mutable.

Lists are defined by the list keyword followed by the singular type that the list will contain. Lists can be created with an initial set of data or empty and can have elements added to them dynamically.

```
| LBRACKET expr_list_opt RBRACKET  { List_lit(List.rev $2)    }
```

Note that square brackets surround an optional expr_list. This allows the empty list to beparsed as `[]`.

Creating a list:

```
list int myints = [3, 2, 5];
list int emptylist;
```

Indexing a list using positional notation:

```
list_elt: ID index index_list_opt  { ListElement($1,$2,$3) }
index: LBRACKET expr RBRACKET      { Index($2)   }
index_list_opt:
                                   { []    }
```

```
| index index_list_opt              { $1::$2  }
```

An indexed list element consists of an identifier followed by at least one index. The index_list_opt allows for multidimensional array indexing.

---

myints[0] == 3; //evaluates to true

myints[0] = 1;

myints[0] == 1; //evaluates to true

Oftentimes, we need to append a value at the end or beginning of the list. This is easily done with the > and < symbol within list indexing:

```
| id ADDHEAD ASSIGN expr     { ListAddHead($1,$4)        }
| id ADDTAIL ASSIGN expr     { ListAddTail($1,$4)        }
```

---

myints[>] = 42;

myints[<] = 37;

// myints now looks like: [37, 3, 2, 5, 42]

```
| expr LENGTH                         { Length($1)                }
```

Getting the length of the list is possible by using the .length operator which returns the length as an integer.

---

myints.length == 5; //evaluates to true

```
| expr CONCAT expr                 { Binop($1, Concat, $3)     }
```

We allow the ++ operator on the list type to concatenate two lists of the same type together.

```
list int numsToAdd = [100];
list int concatResult = myints ++ numsToAdd;
// concatResult == [37, 3, 2, 5, 42, 100]
```

Delete elements from a list using the right and left shift operators:

```
| expr RSHIFT expr          { Binop($1, RShift, $3)      }
| expr LSHIFT expr          { Binop($1, LShift, $3)      }
```

```
myInts << 1;
// myints now looks like: [3, 2, 5, 42]
myInts >> 2;
// myints now looks like: [3, 2]
```

3. 8 Program Structure

Program is the top level node in the syntax tree. Since we parse bottom-up all the parsing should end here.

program: vdecl list * func decl list

A program contains multiple (top-level) variable declarations and then multiple function declarations with arguments if necessary.

```
vdecl:
 typ_decl ID SEMICOLON { Vdecl  (($1, $2),Noexpr) }
| typ_decl ID ASSIGN expr SEMICOLON { Vdecl  (($1, $2),$4)}
```

A type decl consists of a list of type specifiers and type qualifiers.

```
Typ_decl:
```

```
| spec_qual_list      {$1}

typ_spec:
 INT      { Int     }
| BOOL     { Bool    }
| FLOAT    { Float   }
| VOID     { Void    }
| CHAR     { Char    }
| STRING   { String }

typ_qual:
| PROB     { Prob }
| LIST     { List }

spec_qual_list:
| typ_spec spec_qual_list_opt {$1::$2}
| typ_qual spec_qual_list {$1::$2}

spec_qual_list_opt:
| /*nothing*/     {[]}
| spec_qual_list {$1}
```

3. 9 Declarations

There are two types of declarations that SMAP supports. One is the variable declarations and the other is function declarations (which also includes the main function).

3.9.1 Variable Declarations

Variable declarations happen at the start of the .smap file, or at the very start of each function.

Five Type specifiers:	**int, bool, float, string**,and **char**
Two type qualifiers:	**prob,** and **list**

Types are declared with or without an initial value. In the case of no initial expression,

```
vdecl:
 typ_decl ID SEMICOLON { Vdecl  (($1, $2),Noexpr) }
| typ_decl ID ASSIGN expr SEMICOLON { Vdecl  (($1, $2),$4)}
```

Note that the type of these declarations are distinct. The standard syntax for declaring the various types of variables are :

```
prob list int specialList;       // one of a group of possible lists
list prob int listOfSpecialNums; //a list where each elt is
                                 //one of a group of possible integers
list prob list list int num;   //a list of where each elt is
                             // one of a group of possible lists of int lists
prob list list int num;      // one of a group of possible lists of int lists
```

Type Initialization

Types are initialized after declaration using the assignment operator. Initialization can occur immediately after declaration.

```
<qualifier>* <type-specifier> identifier = initializer;
    or some time later in a separate statement:
<qualifier>* <type-specifier> identifier; //declared here
...
identifier = initializer;                 //initialized here
```

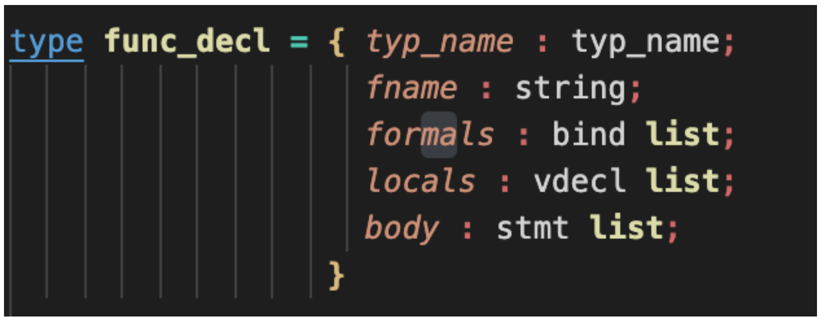
```
/* identifier is either an indexed list element
or regular text identifier  */
id:
 ID                            { Id($1)                    }
| list_elt                     { $1                        }
```

Note that an identifier can be either a regular text identifier, or a list_elt. This ensures that indexed elements within an array can be assigned to.

3.9.2 Function Declarations

The main function is the place where all the code written inside or outside gets executed (code outside has to be called inside the main to get executed).

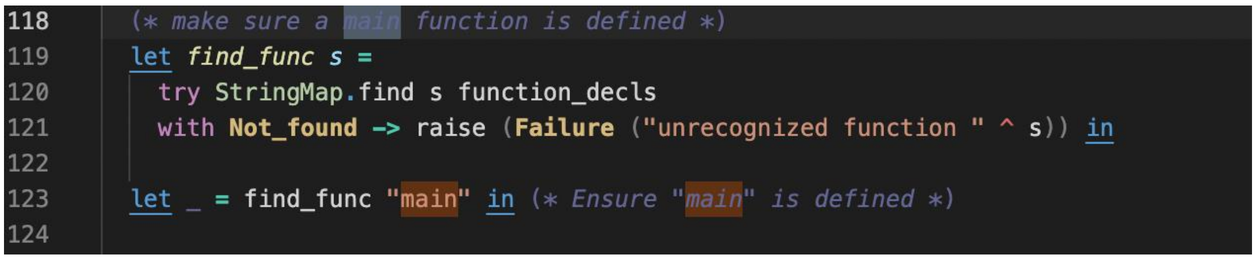
The functions in SMAP have the following declaration in the ast.ml



```
type func_decl = { typ_name : typ_name;
                    fname : string;
                    formals : bind list;
                    locals : vdecl list;
                    body : stmt list;
                  }
```

Here the typ_name is the return type of the function, the fname is the name of the function, the formals is a list of arguments to the functions, the locals is the list of local variable declaration to the function and the body is a list of statements that gets executed when there is a function call.

Finding the main function gets handled in the semant file where the “find_func” function is responsible for checking for the existence of the function.



```
  (* make sure a main function is defined *)
  let find_func s =
    try StringMap.find s function_decls
    with Not_found -> raise (Failure ("unrecognized function " ^ s)) in

  let _ = find_func "main" in (* Ensure "main" is defined *)

```

The function declarator consists of a return type followed by the function signature. The function signature lists the return type (void keyword is used if no return type) , name of the function, and a list of parameters inside parentheses (if the function has 0 arguments, an empty pair of parentheses are used).

The function definition consists of the function signature, and function body. The latter is a block of statements denoted with curly braces.

A note on Blocks: *Curly braces surrounding a group of statements denotes a "block" of statements. Entering a block signals entrance into a new, more local scope. Contents like bindings (identifiers and their values) declared inside a block are not visible outside the block. Statements inside a block are executed in sequence.*

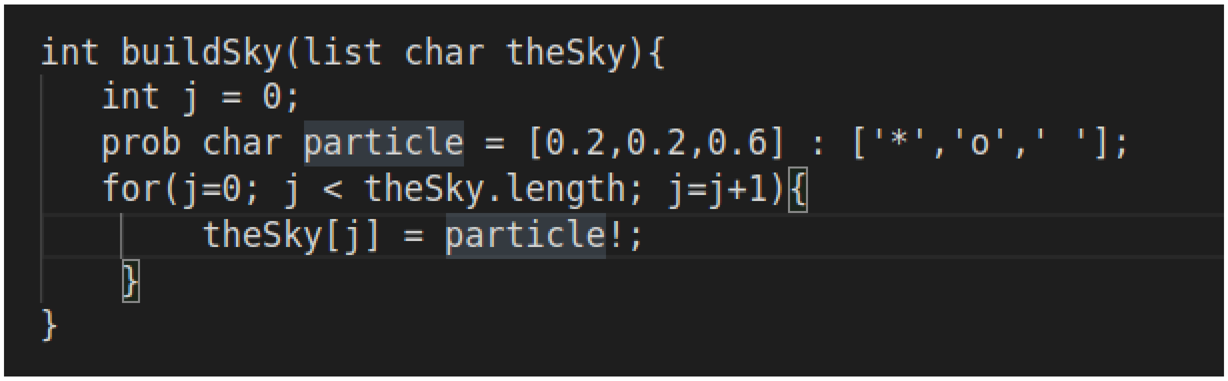
To declare a function without immediately specifying its definition, write out the type signature followed by a semicolon.

```
return-type name (type-specifier arg1_id, ... type-specifier argn_id)
```

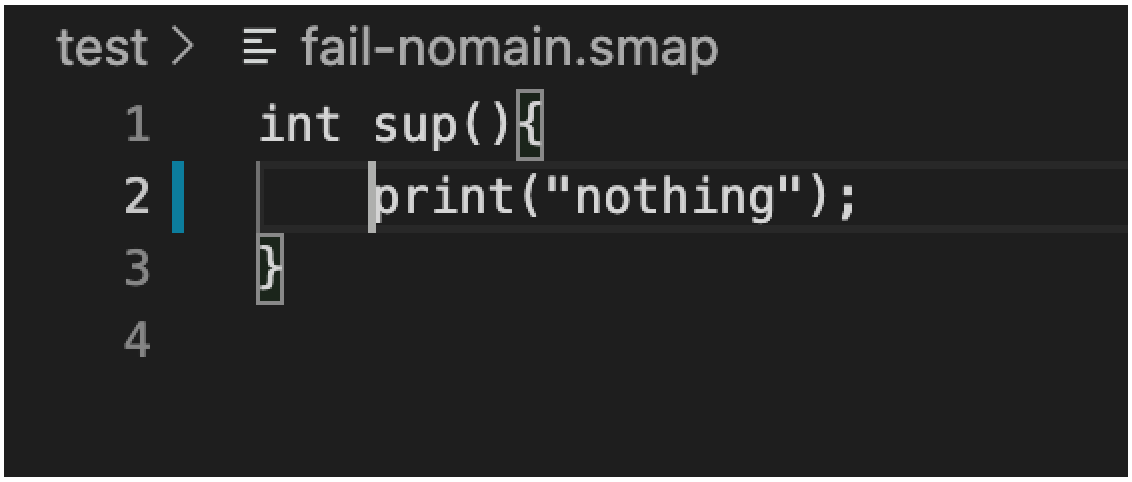
To declare and define a function or to define a function declared earlier, write the type signature followed by the body of the function enclosed in curly braces.

```
return-type name (type-specifier arg1_id, ... type-specifier argn_id){
        statement;
        …
        statement;
        return identifier; //where identifier is of type return-type
}
```

An example of a function declaration in the SMAP is as follows -


```
int buildSky(list char theSky){
    int j = 0;
    prob char particle = [0.2,0.2,0.6] : ['*','o',' '];
    for(j=0; j < theSky.length; j=j+1){
        theSky[j] = particle!;
    }
}
```

An example of an invalid program in SMAP is as follows -

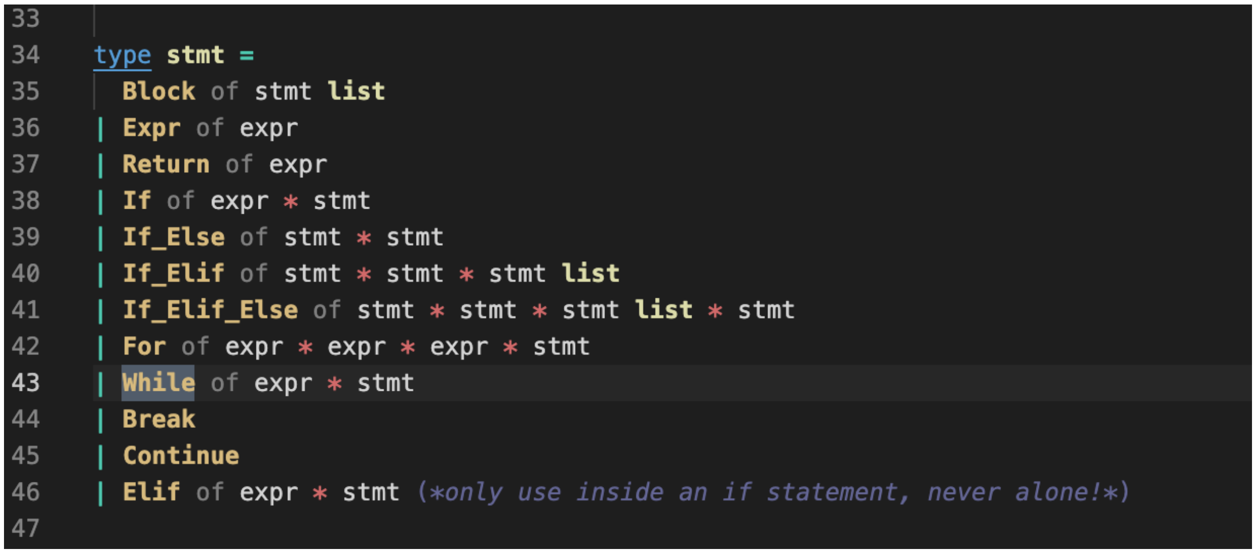


```
int sup(){
    print("nothing");
}
```

3.10 Statements

Statements are usually executed sequentially, i.e. in the order they are defined within a program. There are 3 special control flow statements that usually jump out of the scope or exit the program. Those 3 special statements are - return, exit and continue.

All the statements are defined in the ast file as follows -



```
type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt
  | If_Else of stmt * stmt
  | If_Elif of stmt * stmt * stmt list
  | If_Elif_Else of stmt * stmt * stmt list * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Break
  | Continue
  | Elif of expr * stmt (*only use inside an if statement, never alone!*)
```

3.10.1 Conditional Statement

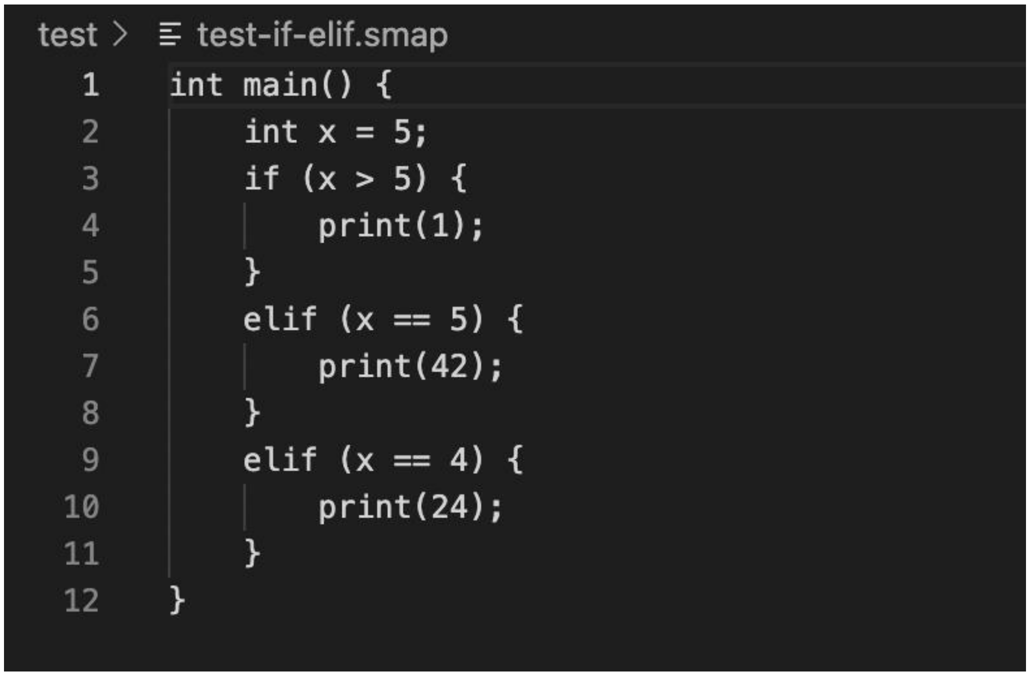
The conditional statements execute based on the boolean condition of the block. If the result of the test expression inside the block is true, then the following list of statements inside the block is executed but if the test expression is false then it either goes in the else if block (if it exists) or in the else block and executes the list of statements based on the conditions.

The SMAP interprepts true as 1 and false as 0 for the conditions. An example of how the parser interprets it as is -

```
if (expr) {
     // this block only executes if expr evaluates to true
}
elif(expr2){ //expr2 is only evaluated if expr evaluates to false
     // this block only executes if expr2 evaluates to true
}
else{
    // this block only executes if none of the other blocks executed
```

}

An example code of the if - elif -else is as follows -



test > test-if-elif.smap

```
int main() {
    int x = 5;
    if (x > 5) {
        print(1);
    }
    elif (x == 5) {
        print(42);
    }
    elif (x == 4) {
        print(24);
    }
}
```

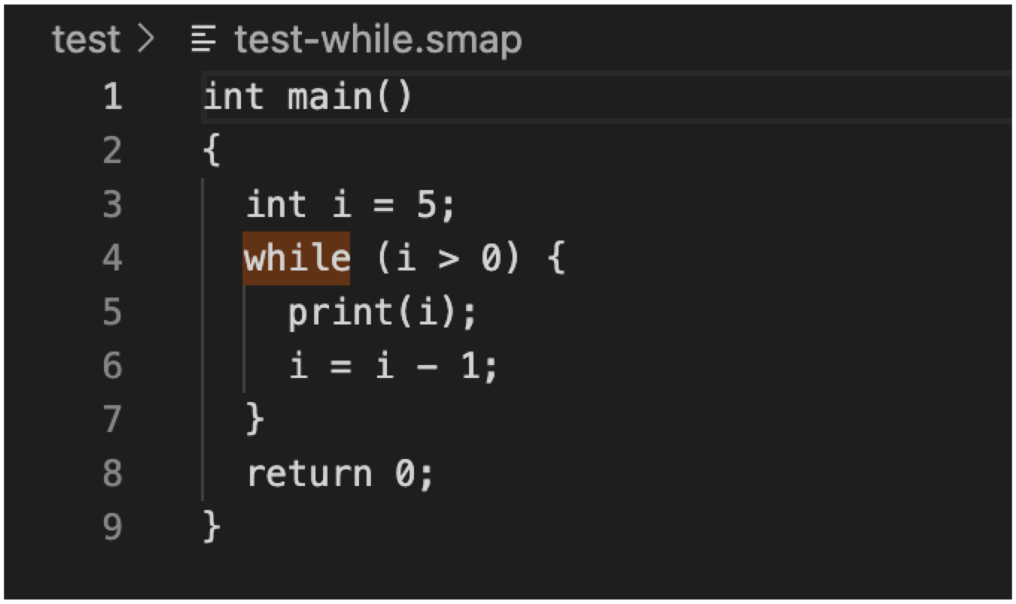
The output of this block will be 42

3.10.2 While Statement

The code inside the while statement will get executed until the conditional statement returns false.

```
while(condition){
      //statements
}
```

An example code from SMAP for while statement -



```
int main()
{
    int i = 5;
    while (i > 0) {
        print(i);
        i = i - 1;
    }
    return 0;
}
```

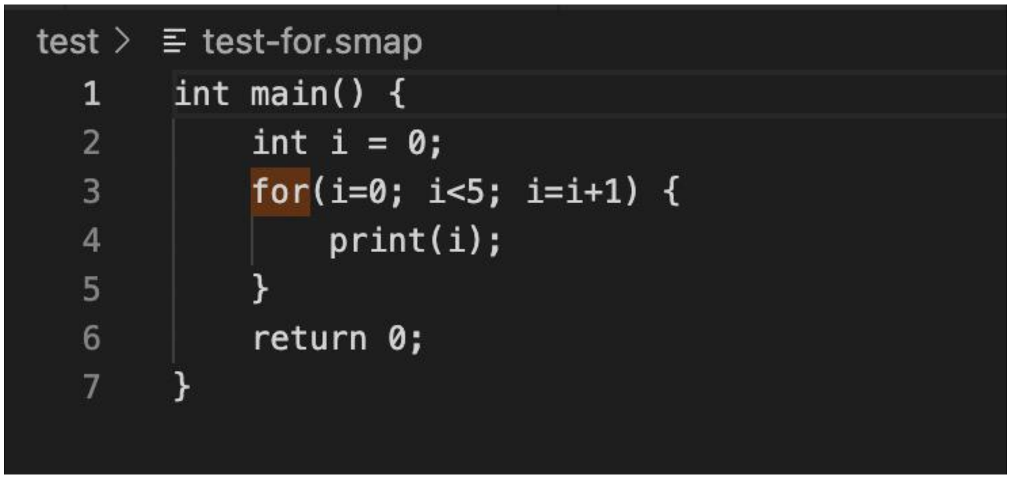
3.10.3 For Statement

A for loop is used to repeat a specific statement block a given number of times by initializing a counter and incrementing/decrementing the counter until the given condition becomes false. For loops can be used to execute a set of statements, once for each item in a given sequence.

The ast handles the for statements in the following manner -

```
for(scoped variable assignment; condition; statement){
      //statements
}
```

An example from the SMAP is as follows -



test > test-for.smap

```
int main() {
    int i = 0;
    for(i=0; i<5; i=i+1) {
        print(i);
    }
    return 0;
}
```

3.10.4 Return Statement

The return statement exits the scope that it is in and either stores the return value to the caller of the function.

Return in ast -

type stmt =

| Return of expr

3.10.5 Statement Block

Statement Block consists of statements that are inside the curly braces, the ones which need to be executed when there is a call to a function which has the block. SMAP uses semicolon as a delimiter to indicate the end of a statement.

```
block:    LBRACE stmt_list RBRACE      { Block(List.rev $2)}
```

3.11 Expressions

Expressions are a sequence of operators, data types and operands that are meant to be evaluated.

```
type expr =
   Binop of expr * binary_op * expr
|   Unop of unary_op * expr
|   Assign of expr * assign_op * expr
|   Int_lit of int
|   Bool_lit of bool
|   Float_lit of float
|   Char_lit of char
|   String_lit of string
|   List_lit of expr list
|   Id of string
|   Cast of typ_name * expr
|   ListElement of string * expr * expr list
|   ListAddHead of expr * expr
|   ListAddTail of expr * expr
|   Length of expr
|   ProbColon of expr * expr
|   FunCall of string * expr list
|   Index of expr
|   Noexpr
```

Noexpr appears inside optional expressions, for example the expr_opt inside the for loop predicate. It also appears inside vdecls when a declared variable is left uninitialized.

Noexpr inside vdecl:

```
vdecl:
 typ_decl ID SEMICOLON { Vdecl (($1, $2),Noexpr) }
| typ_decl ID ASSIGN expr SEMICOLON { Vdecl (($1, $2),$4)}
```

Optional expressions inside for loop:

```
| FOR LPAREN expr_opt SEMICOLON expr_opt SEMICOLON expr_opt RPAREN
block            { For($3,$5,$7,$9) }
```

Noexpr in optional expressions:

```
expr_opt:      /* zero or one expressions */
                       { Noexpr      }
| expr                  { $1          }

expr_list_opt:    /* zero or more expressions */{  []           }
| expr_list             { $1          }

expr_list:                /* one or more expressions */
 expr                  { [$1]        }
| expr_list COMMA expr  { $3 :: $1    }
```

3.11.5 Function Call

```
| ID LPAREN args_opt RPAREN            { FunCall ($1,$3)           }
```

Chapter 4
Standard Library

4.1 List functions

- **init_list (list *l)** - Initialises a list and gives it an INITIAL_LIST_CAPACITY of 10 elements.
- **check_resizing(list *l)** - this function checks if the list is at capacity and returns 1 if resizing is required.
- **check_empty (list *l) -** checks if the list is empty and returns a value of 1 if it is empty and 0 otherwise.
- **resize (list *l) -** this function resizes the list and increases the size by multiplying the initial size by 2.
- **push_back (list *l, char *item) -** add the item to the back of the list l. This function has a void return type.
- **push_front (list *l, char *item) -** add the item to the front of the list l. This function has a void return type.
- **del_back (list *l) -** deletes the last element from the list , returns 0 on success and -1 on failure.
- **del_front (list *l) -** deletes the first element from the list, returns 0 on success and -1 on failure.
- **del_at (list *l, int i) -** deletes the ith element from the list l and returns 0 on success and -1 on failure.
- **get_back(list *l) -** returns the last element from the list. This function has a return type of char.
- **get_front(list *l) -** returns the first element from the list. This function has a return type of char.
- **get_at(list *l, int i) -** return the ith element from the list. Returns a NULL of the i value is out of bounds or if the list is empty and char otherwise.
- **print_list_int(list *l) -** prints all the elements from the list.

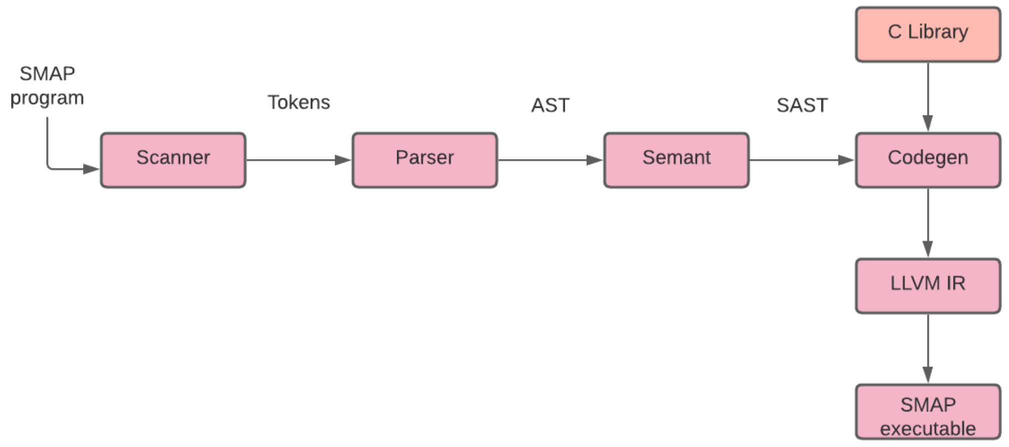
4.1 Prob functions

- **get_probs (prob *p) -**
- **get_vals (prob *p) -**
- **get_length (prob *p) -**
- **peek (prob *p) -**

4.2 String functions

- **corresponding_int(list* x, int index) -** Returns encoded version of the string at the index.
- **stringReverse(list* x)** - Reverse the input list and print a casted char of the values.
- **characterLocation(list* x, list* y)** - Returns the location of the character (taken from y) found in x.
- **stringLength(list* x)** - Returns the length of the list.

Chapter 6
Architecture Design



6.1 Scanner

The scanner takes in the SMAP program file (.smap) and generates tokens for all the code including the identifiers, keywords, operators, and literals specific to the SMAP language.

6.2 Parser

The parser takes in the tokens generated from the scanner and converts them into an abstract syntax tree (AST) based on the rules of SMAP grammar that are defined in the AST file.

6.3 Semantic Checking

This is the stage where the type check happens. The compiler checks the AST file for all the variables, functions and anything else that requires a check and converts it into an SAST file, which is similar to the AST but has grammar that has been type checked.

6.4 Code Generation

During code generation, semantically checked AST (which is the SAST now) is passed TO generate an intermediate representation (IR) of lower- level instructions. Outside library functions are also linked in this stage. Finally, the SMAP source code is converted into LLVM IR, which can then be compiled further into machine code.

Chapter 7

7.1 Testing Suite and Automation

SMAP was developed progressively over multiple branches. The main branch being the dune branch which implements all the prob type API and the list API including over 75 tests. All the team members contributed to the testing and the testing suite includes both the success and failing test cases.

7.2 Example Test program

7.2.1 SMAP Code 1

This code prints recursively. The code and the .ll file are -

SMAP CODE1 .smap file

```
//ToDo: Implement the military enigma

int character_location(list int x,int y){

        int j = 0;

        int i = 0;

        for(i=0;i<26;i=i+1) {

        if(x[i]==y) {

        return i;

        }

        }

        return j;


}
```

```
list int rotor_shift(list int rotor_input,int position){

        int character_holder=rotor_input[0];

        int k=0;

        int l=0;

        int m=0;

        for(k=0;k<position;k=k+1){

        character_holder=rotor_input[0];

        for(l=0;l<25;l=l+1){

        m = l + 1;

                rotor_input[l]=rotor_input[m];

                if(l==24){

                rotor_input[25]=character_holder;

                }

        }

        }

        return rotor_input;


}


int return_cor_int(string what_message, int what_index) {

        int res = corresponding_int(what_message,what_index);

        return res;

}
```

```
list int initialize_input_text(string message_user,list int input_list_message) {


        int message_length_input = stringLength(message_user);

        int message_length_init = 0;

        int current_int_val = 0;


        for(message_length_init = 0; message_length_init <
message_length_input;message_length_init=message_length_init+1) {

        current_int_val = return_cor_int(message_user,message_length_init);

    input_list_message[message_length_init] = current_int_val;

        }


        return input_list_message;

}




int character_location_status(list int xx, int yy) {


        int my_response = character_location(xx,yy);


        return my_response;


}


int my_modulus_result(int base) {

        int modulus_results = modulus_operation(base,26);
```

```
            return modulus_results;

    }



    int rotor_direction_output(int input_output_char_loc,list int rotor_one_input_direct,list int
    rotor_one_output_direct,list int rotor_two_input_direct,list int rotor_two_output_direct,list int
    rotor_three_input_direct,list int rotor_three_output_direct){



            int index_reverse = 0;

            int characher_location_holder=input_output_char_loc;

            int characher_location_holder_two = 0;

            list int reflector= [0,1,2,3,4,5,6,3,8,9,10,6,12,10,12,8,4,1,5,19,2,21,21,9,0,19];




            //Getting character on rotor3 input tray


            characher_location_holder=character_location(rotor_three_input_direct,rotor_three_output_direct[characher_location_holder]);




            //Getting character on rotor2 input tray


            characher_location_holder=character_location(rotor_two_input_direct,rotor_two_output_direct[characher_location_holder]);



            //Getting character on rotor1 input tray


            characher_location_holder=character_location(rotor_one_input_direct,rotor_one_output_direct[characher_location_holder]);



            //Verified till above
```

```
        //Getting the reflected elment and the other reflected value

        for(index_reverse=0;index_reverse<26;index_reverse=index_reverse+1){

        if(characher_location_holder==index_reverse){


        }

        else{

        if(reflector[characher_location_holder]==reflector[index_reverse]){

            characher_location_holder_two=index_reverse;

        }

        }

        }


        //Getting character on rotor1 output tray

        characher_location_holder=character_location(rotor_one_output_direct,rotor_one_input_direct[characher_l
ocation_holder_two]);

        //Getting character on rotor2 output tray

        characher_location_holder=character_location(rotor_two_output_direct,rotor_two_input_direct[characher_
location_holder]);

        //Getting character on rotor3 output tray

        characher_location_holder=character_location(rotor_three_output_direct,rotor_three_input_direct[characher_
r_location_holder]);

        //Cipher Character

        return characher_location_holder;
```

```
}




list int encryption_decryption(list int rotor_one_e_input,list int rotor_one_e_output,list int rotor_two_e_input,list
int rotor_two_e_output,list int rotor_three_e_input,list int rotor_three_e_output, list int text, int length_input_text){


        int modulus_result = 0;

        int modulus_result_t = 0;

        int char_loc_holder = 0;

        int count_rotor_one=0;

        int count_rotor_two=0;

        int count_rotor_three=0;

        int text_len = length_input_text;

        int loop_counter_e_d = 0;

        list int input_output_char = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25];

        list int cipher_output_character =
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
,1,1,1];



        for(loop_counter_e_d=0;loop_counter_e_d<text_len;loop_counter_e_d=loop_counter_e_d+1) {


        char_loc_holder = character_location_status(input_output_char, text[loop_counter_e_d]);

        rotor_shift(rotor_three_e_input,1);

        rotor_shift(rotor_three_e_output,1);
        count_rotor_three = count_rotor_three + 1;


    if(count_rotor_three>0) {

        modulus_result = my_modulus_result(count_rotor_three);

        if(modulus_result == 0) {

        rotor_shift(rotor_two_e_input,1);

                                rotor_shift(rotor_two_e_output,1);

        count_rotor_two = count_rotor_two + 1;


        if(count_rotor_two>0) {

                modulus_result_t = my_modulus_result(count_rotor_two);

                if(modulus_result_t == 0) {

                  rotor_shift(rotor_one_e_input,1);

                rotor_shift(rotor_one_e_output,1);

                count_rotor_one = count_rotor_one + 1;

                }


        }


        }


        }
```

```
char_loc_holder=rotor_direction_output(char_loc_holder,rotor_one_e_input,rotor_one_e_output,rotor_two_e_input,
rotor_two_e_output,rotor_three_e_input,rotor_three_e_output);

        cipher_output_character[loop_counter_e_d]=input_output_char[char_loc_holder];

        }

        return cipher_output_character;

}



//ToDo: Implement the plugboard redirect

list int plugboard_redirect(list int text_redirect,int count_for_input, list int first_plugs, list int second_plugs, int
message_length_plug){

        //substitute a character with different character

        list int list_of_char_redirect = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25];

        list int sub_list_char = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25];

        list int temp_sub_val_holder = [1,1];

        list int loc_holder = [1,1];

        int loc_holding_for_text = 0;

        int counter=0;

        int string_redirect_length = 0;
```

```
//provide machine setting credentials

if(count_for_input==0) {

while(counter!=10) {

temp_sub_val_holder[0] = first_plugs[counter];

temp_sub_val_holder[1] = second_plugs[counter];

loc_holder[0] = character_location(list_of_char_redirect,list_of_char_redirect[temp_sub_val_holder[0]]);

loc_holder[1] = character_location(list_of_char_redirect,list_of_char_redirect[temp_sub_val_holder[1]]);

sub_list_char[(loc_holder[0])] = list_of_char_redirect[(loc_holder[1])];

sub_list_char[(loc_holder[1])] = list_of_char_redirect[(loc_holder[0])];

counter=counter+1;

}

}




for(string_redirect_length=0;string_redirect_length<message_length_plug;string_redirect_length=string_re
direct_length+1){

loc_holding_for_text = character_location(list_of_char_redirect,text_redirect[string_redirect_length]);

text_redirect[string_redirect_length] = sub_list_char[loc_holding_for_text];

}




return text_redirect;
```

```
}



int main() {


            int plugboard_input_count = 0;

            int final_output_counter = 0;

            int location_on_initialization_array = 0;

            list int temp = [1];

            list int input_text =
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
,1,1,1];

            string messagez =
"ZEROSIXHUNDREEDHOURSWEATHERTODAYISCLEARRAININTHEEVENINGHEILHITLER";

            int length_input = stringLength(messagez);


            //ToDo: Define the key -> SIG (Rotor) PMP (Ring)

            list int key = [18,8,6];

            list int ring_setting = [15,12,15];


            //ToDo: Create plugboard settings

            list int first_plugs_init = [1,2,3,4,5,6,9,13,10,21];

            list int second_plugs_init = [18,8,12,16,20,24,11,15,25,22];
```

```
//ToDo: Rotor 3 initialize

list int rotor_three_out = [1,3,5,7,9,11,2,15,17,19,23,21,25,13,24,4,8,22,6,0,10,12,20,18,16,14];

list int rotor_three_in = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25];



//ToDo: Rotor 2 initialize

list int rotor_two_out = [0,9,3,10,18,8,17,20,23,1,11,7,22,19,12,2,16,6,25,13,15,24,5,21,14,4];

list int rotor_two_in = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25];


//ToDo: Rotor 1 initialize

list int rotor_one_out = [4,10,12,5,11,6,3,16,21,25,13,19,14,22,24,7,23,20,18,15,0,8,1,17,2,9];

list int rotor_one_in = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25];



//ToDo: Shift the inputs based on key and ring settings


//For Rotor - I

location_on_initialization_array=character_location(rotor_one_in,key[0]);

rotor_one_in = rotor_shift(rotor_one_in,location_on_initialization_array);



location_on_initialization_array=character_location(rotor_one_out,ring_setting[0]);

rotor_one_out = rotor_shift(rotor_one_out,location_on_initialization_array);



//For Rotor - II
```

```
location_on_initialization_array=character_location(rotor_two_in,key[1]);

rotor_two_in = rotor_shift(rotor_two_in,location_on_initialization_array);



location_on_initialization_array=character_location(rotor_two_out,ring_setting[1]);

rotor_two_out = rotor_shift(rotor_two_out,location_on_initialization_array);



//For Rotor - III

location_on_initialization_array=character_location(rotor_three_in,key[2]);

rotor_three_in = rotor_shift(rotor_three_in,location_on_initialization_array);


location_on_initialization_array=character_location(rotor_three_out,ring_setting[2]);

rotor_three_out = rotor_shift(rotor_three_out,location_on_initialization_array);




//ToDo: Initialize the message

input_text = initialize_input_text(messagez,input_text);


//ToDo: VERY IMPORTANT  ->   Plug re-direct the input message


input_text =
plugboard_redirect(input_text,plugboard_input_count,first_plugs_init,second_plugs_init,length_input);
```

```
        //ToDo: Initialize the encryption_decryption function

        temp =
encryption_decryption(rotor_one_in,rotor_one_out,rotor_two_in,rotor_two_out,rotor_three_in,rotor_three_out,input
_text,length_input);

        temp = plugboard_redirect(temp,plugboard_input_count,first_plugs_init,second_plugs_init,length_input);

        print("Output text: ");



        for(final_output_counter=0;final_output_counter<length_input;final_output_counter=final_output_counter
+1) {

        corresponding_char(temp[final_output_counter]);

        }







        return 0;

}
```

SMAP Code1 .ll file

```
; ModuleID = 'Smap'
source_filename = "Smap"

%list = type { i8**, i32, i32 }
%prob = type { %list, %list, i32 }

@test = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.1 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@test.2 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.3 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.4 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.5 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@test.6 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.7 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.8 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.9 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@test.10 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.11 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.12 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.13 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@test.14 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.15 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.16 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.17 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@test.18 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.19 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.20 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.21 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@test.22 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.23 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.24 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.25 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@test.26 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.27 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.28 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.29 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
```

```
@test.30 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.31 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.32 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.33 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@test.34 = private unnamed_addr constant [17 x i8] c"test test test!\0A\00", align 1
@newLine.35 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1
@fmt.36 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.37 = private unnamed_addr constant [3 x i8] c"%f\00", align 1

declare i32 @printstr(i8*)

declare i32 @corresponding_char(i32)

declare i32 @printchar(i8)

declare i32 @print_list_char(%list*)

declare i32 @printb(i1)

declare i32 @stringLength(%list*)

declare i32 @stringReverse(%list*)

declare i32 @absolute(i32)

declare i32 @key_test(i32, i32)

declare i32 @characterLocation(%list*, %list*)

declare i32 @isCompDivisible(i32)

declare i32 @divisible(i32, i32)

declare i32 @power(i32, i32)

declare i32 @ceilFloat(double)

declare i32 @ascii(%list*)

declare %list* @int_to_char(i32)

declare i32 @corresponding_int(%list*, i32)

declare { i8*, i32 } @testMakeStruct(i8*, i32)

declare i32 @printint(i32)

declare i32 @printf(i8*, ...)

declare i32 @push_front(%list*, i8*, ...)

declare i32 @push_back(%list*, i8*, ...)

declare i32 @init_list(%list*, ...)

declare i32 @init_prob(%prob*, %list*, %list*, ...)

declare i8* @peek(%prob*, ...)

declare i32 @list_length(%list*, ...)

declare i32 @modulus_operation(i32, i32, ...)

declare i32 @get_length(%prob*, ...)

declare i8* @get_at(%list*, i32, ...)

declare %list* @set_at(%list*, i32, i8*, ...)

declare i32 @print_list_int(%list*, ...)

declare i32 @print_list_float(%list*, ...)

declare %list* @get_vals(%prob*, ...)

declare %list* @get_probs(%prob*, ...)

declare i32 @print_prob_int_debug(%prob*, ...)

declare i32 @bad_add_head(%list*, i8*, ...)

declare i32 @very_bad_get_head(%list*, ...)

define i32 @main() {
entry:
  %plugboard_input_count = alloca i32
  %final_output_counter = alloca i32
  %location_on_initialization_array = alloca i32
  %temp = alloca %list*
  %input_text = alloca %list*
  %messagez = alloca %list*
  %length_input = alloca i32
  %key = alloca %list*
  %ring_setting = alloca %list*
  %first_plugs_init = alloca %list*
  %second_plugs_init = alloca %list*
  %rotor_three_out = alloca %list*
  %rotor_three_in = alloca %list*
  %rotor_two_out = alloca %list*
  %rotor_two_in = alloca %list*
  %rotor_one_out = alloca %list*
  %rotor_one_in = alloca %list*
  store i32 0, i32* %plugboard_input_count
  store i32 0, i32* %final_output_counter
  store i32 0, i32* %location_on_initialization_array
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr = bitcast i8* %malloccall to %list*
  %init_list = call i32 (%list*, ...) @init_list(%list* %listAddr)
  %tmp = alloca i32*
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr = bitcast i8* %malloccall1 to i32*
  store i32* %tmpAddr, i32** %tmp
  %tmpWithVal = load i32*, i32** %tmp
  store i32 1, i32* %tmpWithVal
  %asChar = bitcast i32* %tmpWithVal to i8*
  %push_back = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar)
  store %list* %listAddr, %list** %temp
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr3 = bitcast i8* %malloccall2 to %list*
  %init_list4 = call i32 (%list*, ...) @init_list(%list* %listAddr3)
  %tmp5 = alloca i32*
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr7 = bitcast i8* %malloccall6 to i32*
  store i32* %tmpAddr7, i32** %tmp5
  %tmpWithVal8 = load i32*, i32** %tmp5
  store i32 1, i32* %tmpWithVal8
  %asChar9 = bitcast i32* %tmpWithVal8 to i8*
  %tmp10 = alloca i32*
  %malloccall11 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr12 = bitcast i8* %malloccall11 to i32*
  store i32* %tmpAddr12, i32** %tmp10
  %tmpWithVal13 = load i32*, i32** %tmp10
  store i32 1, i32* %tmpWithVal13
  %asChar14 = bitcast i32* %tmpWithVal13 to i8*
  %tmp15 = alloca i32*
  %malloccall16 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr17 = bitcast i8* %malloccall16 to i32*
  store i32* %tmpAddr17, i32** %tmp15
  %tmpWithVal18 = load i32*, i32** %tmp15
  store i32 1, i32* %tmpWithVal18
  %asChar19 = bitcast i32* %tmpWithVal18 to i8*
  %tmp20 = alloca i32*
  %malloccall21 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr22 = bitcast i8* %malloccall21 to i32*
  store i32* %tmpAddr22, i32** %tmp20
  %tmpWithVal23 = load i32*, i32** %tmp20
  store i32 1, i32* %tmpWithVal23
  %asChar24 = bitcast i32* %tmpWithVal23 to i8*
  %tmp25 = alloca i32*
  %malloccall26 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr27 = bitcast i8* %malloccall26 to i32*
  store i32* %tmpAddr27, i32** %tmp25
  %tmpWithVal28 = load i32*, i32** %tmp25
  store i32 1, i32* %tmpWithVal28
  %asChar29 = bitcast i32* %tmpWithVal28 to i8*
  %tmp30 = alloca i32*
  %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr32 = bitcast i8* %malloccall31 to i32*
  store i32* %tmpAddr32, i32** %tmp30
  %tmpWithVal33 = load i32*, i32** %tmp30
  store i32 1, i32* %tmpWithVal33
  %asChar34 = bitcast i32* %tmpWithVal33 to i8*
  %tmp35 = alloca i32*
  %malloccall36 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr37 = bitcast i8* %malloccall36 to i32*
  store i32* %tmpAddr37, i32** %tmp35
  %tmpWithVal38 = load i32*, i32** %tmp35
  store i32 1, i32* %tmpWithVal38
  %asChar39 = bitcast i32* %tmpWithVal38 to i8*
  %tmp40 = alloca i32*
  %malloccall41 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr42 = bitcast i8* %malloccall41 to i32*
  store i32* %tmpAddr42, i32** %tmp40
  %tmpWithVal43 = load i32*, i32** %tmp40
  store i32 1, i32* %tmpWithVal43
  %asChar44 = bitcast i32* %tmpWithVal43 to i8*
  %tmp45 = alloca i32*
  %malloccall46 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr47 = bitcast i8* %malloccall46 to i32*
  store i32* %tmpAddr47, i32** %tmp45
  %tmpWithVal48 = load i32*, i32** %tmp45
  store i32 1, i32* %tmpWithVal48
  %asChar49 = bitcast i32* %tmpWithVal48 to i8*
  %tmp50 = alloca i32*
  %malloccall51 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr52 = bitcast i8* %malloccall51 to i32*
  store i32* %tmpAddr52, i32** %tmp50
  %tmpWithVal53 = load i32*, i32** %tmp50
  store i32 1, i32* %tmpWithVal53
  %asChar54 = bitcast i32* %tmpWithVal53 to i8*
  %tmp55 = alloca i32*
  %malloccall56 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr57 = bitcast i8* %malloccall56 to i32*
  store i32* %tmpAddr57, i32** %tmp55
  %tmpWithVal58 = load i32*, i32** %tmp55
  store i32 1, i32* %tmpWithVal58
  %asChar59 = bitcast i32* %tmpWithVal58 to i8*
  %tmp60 = alloca i32*
  %malloccall61 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr62 = bitcast i8* %malloccall61 to i32*
  store i32* %tmpAddr62, i32** %tmp60
```

```
	%tmpWithVal63 = load i32*, i32** %tmp60
	store i32 1, i32* %tmpWithVal63
	%asChar64 = bitcast i32* %tmpWithVal63 to i8*
	%tmp65 = alloca i32*
	%malloccall66 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr67 = bitcast i8* %malloccall66 to i32*
	store i32* %tmpAddr67, i32** %tmp65
	%tmpWithVal68 = load i32*, i32** %tmp65
	store i32 1, i32* %tmpWithVal68
	%asChar69 = bitcast i32* %tmpWithVal68 to i8*
	%tmp70 = alloca i32*
	%malloccall71 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr72 = bitcast i8* %malloccall71 to i32*
	store i32* %tmpAddr72, i32** %tmp70
	%tmpWithVal73 = load i32*, i32** %tmp70
	store i32 1, i32* %tmpWithVal73
	%asChar74 = bitcast i32* %tmpWithVal73 to i8*
	%tmp75 = alloca i32*
	%malloccall76 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr77 = bitcast i8* %malloccall76 to i32*
	store i32* %tmpAddr77, i32** %tmp75
	%tmpWithVal78 = load i32*, i32** %tmp75
	store i32 1, i32* %tmpWithVal78
	%asChar79 = bitcast i32* %tmpWithVal78 to i8*
	%tmp80 = alloca i32*
	%malloccall81 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr82 = bitcast i8* %malloccall81 to i32*
	store i32* %tmpAddr82, i32** %tmp80
	%tmpWithVal83 = load i32*, i32** %tmp80
	store i32 1, i32* %tmpWithVal83
	%asChar84 = bitcast i32* %tmpWithVal83 to i8*
	%tmp85 = alloca i32*
	%malloccall86 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr87 = bitcast i8* %malloccall86 to i32*
	store i32* %tmpAddr87, i32** %tmp85
	%tmpWithVal88 = load i32*, i32** %tmp85
	store i32 1, i32* %tmpWithVal88
	%asChar89 = bitcast i32* %tmpWithVal88 to i8*
	%tmp90 = alloca i32*
```

```
%malloccall91 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr92 = bitcast i8* %malloccall91 to i32*
store i32* %tmpAddr92, i32** %tmp90
%tmpWithVal93 = load i32*, i32** %tmp90
store i32 1, i32* %tmpWithVal93
%asChar94 = bitcast i32* %tmpWithVal93 to i8*
%tmp95 = alloca i32*
%malloccall96 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr97 = bitcast i8* %malloccall96 to i32*
store i32* %tmpAddr97, i32** %tmp95
%tmpWithVal98 = load i32*, i32** %tmp95
store i32 1, i32* %tmpWithVal98
%asChar99 = bitcast i32* %tmpWithVal98 to i8*
%tmp100 = alloca i32*
%malloccall101 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr102 = bitcast i8* %malloccall101 to i32*
store i32* %tmpAddr102, i32** %tmp100
%tmpWithVal103 = load i32*, i32** %tmp100
store i32 1, i32* %tmpWithVal103
%asChar104 = bitcast i32* %tmpWithVal103 to i8*
%tmp105 = alloca i32*
%malloccall106 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr107 = bitcast i8* %malloccall106 to i32*
store i32* %tmpAddr107, i32** %tmp105
%tmpWithVal108 = load i32*, i32** %tmp105
store i32 1, i32* %tmpWithVal108
%asChar109 = bitcast i32* %tmpWithVal108 to i8*
%tmp110 = alloca i32*
%malloccall111 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr112 = bitcast i8* %malloccall111 to i32*
store i32* %tmpAddr112, i32** %tmp110
%tmpWithVal113 = load i32*, i32** %tmp110
store i32 1, i32* %tmpWithVal113
%asChar114 = bitcast i32* %tmpWithVal113 to i8*
%tmp115 = alloca i32*
%malloccall116 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr117 = bitcast i8* %malloccall116 to i32*
store i32* %tmpAddr117, i32** %tmp115
```

```
  %tmpWithVal118 = load i32*, i32** %tmp115
  store i32 1, i32* %tmpWithVal118
  %asChar119 = bitcast i32* %tmpWithVal118 to i8*
  %tmp120 = alloca i32*
  %malloccall121 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr122 = bitcast i8* %malloccall121 to i32*
  store i32* %tmpAddr122, i32** %tmp120
  %tmpWithVal123 = load i32*, i32** %tmp120
  store i32 1, i32* %tmpWithVal123
  %asChar124 = bitcast i32* %tmpWithVal123 to i8*
  %tmp125 = alloca i32*
  %malloccall126 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr127 = bitcast i8* %malloccall126 to i32*
  store i32* %tmpAddr127, i32** %tmp125
  %tmpWithVal128 = load i32*, i32** %tmp125
  store i32 1, i32* %tmpWithVal128
  %asChar129 = bitcast i32* %tmpWithVal128 to i8*
  %tmp130 = alloca i32*
  %malloccall131 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr132 = bitcast i8* %malloccall131 to i32*
  store i32* %tmpAddr132, i32** %tmp130
  %tmpWithVal133 = load i32*, i32** %tmp130
  store i32 1, i32* %tmpWithVal133
  %asChar134 = bitcast i32* %tmpWithVal133 to i8*
  %tmp135 = alloca i32*
  %malloccall136 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr137 = bitcast i8* %malloccall136 to i32*
  store i32* %tmpAddr137, i32** %tmp135
  %tmpWithVal138 = load i32*, i32** %tmp135
  store i32 1, i32* %tmpWithVal138
  %asChar139 = bitcast i32* %tmpWithVal138 to i8*
  %tmp140 = alloca i32*
  %malloccall141 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr142 = bitcast i8* %malloccall141 to i32*
  store i32* %tmpAddr142, i32** %tmp140
  %tmpWithVal143 = load i32*, i32** %tmp140
  store i32 1, i32* %tmpWithVal143
  %asChar144 = bitcast i32* %tmpWithVal143 to i8*
  %tmp145 = alloca i32*
```

```
%malloccall146 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr147 = bitcast i8* %malloccall146 to i32*
store i32* %tmpAddr147, i32** %tmp145
%tmpWithVal148 = load i32*, i32** %tmp145
store i32 1, i32* %tmpWithVal148
%asChar149 = bitcast i32* %tmpWithVal148 to i8*
%tmp150 = alloca i32*
%malloccall151 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr152 = bitcast i8* %malloccall151 to i32*
store i32* %tmpAddr152, i32** %tmp150
%tmpWithVal153 = load i32*, i32** %tmp150
store i32 1, i32* %tmpWithVal153
%asChar154 = bitcast i32* %tmpWithVal153 to i8*
%tmp155 = alloca i32*
%malloccall156 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr157 = bitcast i8* %malloccall156 to i32*
store i32* %tmpAddr157, i32** %tmp155
%tmpWithVal158 = load i32*, i32** %tmp155
store i32 1, i32* %tmpWithVal158
%asChar159 = bitcast i32* %tmpWithVal158 to i8*
%tmp160 = alloca i32*
%malloccall161 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr162 = bitcast i8* %malloccall161 to i32*
store i32* %tmpAddr162, i32** %tmp160
%tmpWithVal163 = load i32*, i32** %tmp160
store i32 1, i32* %tmpWithVal163
%asChar164 = bitcast i32* %tmpWithVal163 to i8*
%tmp165 = alloca i32*
%malloccall166 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr167 = bitcast i8* %malloccall166 to i32*
store i32* %tmpAddr167, i32** %tmp165
%tmpWithVal168 = load i32*, i32** %tmp165
store i32 1, i32* %tmpWithVal168
%asChar169 = bitcast i32* %tmpWithVal168 to i8*
%tmp170 = alloca i32*
%malloccall171 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr172 = bitcast i8* %malloccall171 to i32*
store i32* %tmpAddr172, i32** %tmp170
```

```
    %tmpWithVal173 = load i32*, i32** %tmp170
    store i32 1, i32* %tmpWithVal173
    %asChar174 = bitcast i32* %tmpWithVal173 to i8*
    %tmp175 = alloca i32*
    %malloccall176 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr177 = bitcast i8* %malloccall176 to i32*
    store i32* %tmpAddr177, i32** %tmp175
    %tmpWithVal178 = load i32*, i32** %tmp175
    store i32 1, i32* %tmpWithVal178
    %asChar179 = bitcast i32* %tmpWithVal178 to i8*
    %tmp180 = alloca i32*
    %malloccall181 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr182 = bitcast i8* %malloccall181 to i32*
    store i32* %tmpAddr182, i32** %tmp180
    %tmpWithVal183 = load i32*, i32** %tmp180
    store i32 1, i32* %tmpWithVal183
    %asChar184 = bitcast i32* %tmpWithVal183 to i8*
    %tmp185 = alloca i32*
    %malloccall186 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr187 = bitcast i8* %malloccall186 to i32*
    store i32* %tmpAddr187, i32** %tmp185
    %tmpWithVal188 = load i32*, i32** %tmp185
    store i32 1, i32* %tmpWithVal188
    %asChar189 = bitcast i32* %tmpWithVal188 to i8*
    %tmp190 = alloca i32*
    %malloccall191 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr192 = bitcast i8* %malloccall191 to i32*
    store i32* %tmpAddr192, i32** %tmp190
    %tmpWithVal193 = load i32*, i32** %tmp190
    store i32 1, i32* %tmpWithVal193
    %asChar194 = bitcast i32* %tmpWithVal193 to i8*
    %tmp195 = alloca i32*
    %malloccall196 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr197 = bitcast i8* %malloccall196 to i32*
    store i32* %tmpAddr197, i32** %tmp195
    %tmpWithVal198 = load i32*, i32** %tmp195
    store i32 1, i32* %tmpWithVal198
    %asChar199 = bitcast i32* %tmpWithVal198 to i8*
    %tmp200 = alloca i32*
```

```
  %malloccall201 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr202 = bitcast i8* %malloccall201 to i32*
  store i32* %tmpAddr202, i32** %tmp200
  %tmpWithVal203 = load i32*, i32** %tmp200
  store i32 1, i32* %tmpWithVal203
  %asChar204 = bitcast i32* %tmpWithVal203 to i8*
  %tmp205 = alloca i32*
  %malloccall206 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr207 = bitcast i8* %malloccall206 to i32*
  store i32* %tmpAddr207, i32** %tmp205
  %tmpWithVal208 = load i32*, i32** %tmp205
  store i32 1, i32* %tmpWithVal208
  %asChar209 = bitcast i32* %tmpWithVal208 to i8*
  %tmp210 = alloca i32*
  %malloccall211 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr212 = bitcast i8* %malloccall211 to i32*
  store i32* %tmpAddr212, i32** %tmp210
  %tmpWithVal213 = load i32*, i32** %tmp210
  store i32 1, i32* %tmpWithVal213
  %asChar214 = bitcast i32* %tmpWithVal213 to i8*
  %tmp215 = alloca i32*
  %malloccall216 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr217 = bitcast i8* %malloccall216 to i32*
  store i32* %tmpAddr217, i32** %tmp215
  %tmpWithVal218 = load i32*, i32** %tmp215
  store i32 1, i32* %tmpWithVal218
  %asChar219 = bitcast i32* %tmpWithVal218 to i8*
  %tmp220 = alloca i32*
  %malloccall221 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr222 = bitcast i8* %malloccall221 to i32*
  store i32* %tmpAddr222, i32** %tmp220
  %tmpWithVal223 = load i32*, i32** %tmp220
  store i32 1, i32* %tmpWithVal223
  %asChar224 = bitcast i32* %tmpWithVal223 to i8*
  %tmp225 = alloca i32*
  %malloccall226 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr227 = bitcast i8* %malloccall226 to i32*
  store i32* %tmpAddr227, i32** %tmp225
  %tmpWithVal228 = load i32*, i32** %tmp225
  store i32 1, i32* %tmpWithVal228
  %asChar229 = bitcast i32* %tmpWithVal228 to i8*
  %tmp230 = alloca i32*
  %malloccall231 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr232 = bitcast i8* %malloccall231 to i32*
  store i32* %tmpAddr232, i32** %tmp230
  %tmpWithVal233 = load i32*, i32** %tmp230
  store i32 1, i32* %tmpWithVal233
  %asChar234 = bitcast i32* %tmpWithVal233 to i8*
  %tmp235 = alloca i32*
  %malloccall236 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr237 = bitcast i8* %malloccall236 to i32*
  store i32* %tmpAddr237, i32** %tmp235
  %tmpWithVal238 = load i32*, i32** %tmp235
  store i32 1, i32* %tmpWithVal238
  %asChar239 = bitcast i32* %tmpWithVal238 to i8*
  %tmp240 = alloca i32*
  %malloccall241 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr242 = bitcast i8* %malloccall241 to i32*
  store i32* %tmpAddr242, i32** %tmp240
  %tmpWithVal243 = load i32*, i32** %tmp240
  store i32 1, i32* %tmpWithVal243
  %asChar244 = bitcast i32* %tmpWithVal243 to i8*
  %tmp245 = alloca i32*
  %malloccall246 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr247 = bitcast i8* %malloccall246 to i32*
  store i32* %tmpAddr247, i32** %tmp245
  %tmpWithVal248 = load i32*, i32** %tmp245
  store i32 1, i32* %tmpWithVal248
  %asChar249 = bitcast i32* %tmpWithVal248 to i8*
  %tmp250 = alloca i32*
  %malloccall251 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr252 = bitcast i8* %malloccall251 to i32*
  store i32* %tmpAddr252, i32** %tmp250
  %tmpWithVal253 = load i32*, i32** %tmp250
  store i32 1, i32* %tmpWithVal253
  %asChar254 = bitcast i32* %tmpWithVal253 to i8*
  %tmp255 = alloca i32*
  %malloccall256 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr257 = bitcast i8* %malloccall256 to i32*
  store i32* %tmpAddr257, i32** %tmp255
  %tmpWithVal258 = load i32*, i32** %tmp255
  store i32 1, i32* %tmpWithVal258
  %asChar259 = bitcast i32* %tmpWithVal258 to i8*
  %tmp260 = alloca i32*
  %malloccall261 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr262 = bitcast i8* %malloccall261 to i32*
  store i32* %tmpAddr262, i32** %tmp260
  %tmpWithVal263 = load i32*, i32** %tmp260
  store i32 1, i32* %tmpWithVal263
  %asChar264 = bitcast i32* %tmpWithVal263 to i8*
  %tmp265 = alloca i32*
  %malloccall266 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr267 = bitcast i8* %malloccall266 to i32*
  store i32* %tmpAddr267, i32** %tmp265
  %tmpWithVal268 = load i32*, i32** %tmp265
  store i32 1, i32* %tmpWithVal268
  %asChar269 = bitcast i32* %tmpWithVal268 to i8*
  %tmp270 = alloca i32*
  %malloccall271 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr272 = bitcast i8* %malloccall271 to i32*
  store i32* %tmpAddr272, i32** %tmp270
  %tmpWithVal273 = load i32*, i32** %tmp270
  store i32 1, i32* %tmpWithVal273
  %asChar274 = bitcast i32* %tmpWithVal273 to i8*
  %tmp275 = alloca i32*
  %malloccall276 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr277 = bitcast i8* %malloccall276 to i32*
  store i32* %tmpAddr277, i32** %tmp275
  %tmpWithVal278 = load i32*, i32** %tmp275
  store i32 1, i32* %tmpWithVal278
  %asChar279 = bitcast i32* %tmpWithVal278 to i8*
  %tmp280 = alloca i32*
  %malloccall281 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr282 = bitcast i8* %malloccall281 to i32*
  store i32* %tmpAddr282, i32** %tmp280
```

```
    %tmpWithVal283 = load i32*, i32** %tmp280
    store i32 1, i32* %tmpWithVal283
    %asChar284 = bitcast i32* %tmpWithVal283 to i8*
    %tmp285 = alloca i32*
    %malloccall286 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr287 = bitcast i8* %malloccall286 to i32*
    store i32* %tmpAddr287, i32** %tmp285
    %tmpWithVal288 = load i32*, i32** %tmp285
    store i32 1, i32* %tmpWithVal288
    %asChar289 = bitcast i32* %tmpWithVal288 to i8*
    %tmp290 = alloca i32*
    %malloccall291 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr292 = bitcast i8* %malloccall291 to i32*
    store i32* %tmpAddr292, i32** %tmp290
    %tmpWithVal293 = load i32*, i32** %tmp290
    store i32 1, i32* %tmpWithVal293
    %asChar294 = bitcast i32* %tmpWithVal293 to i8*
    %tmp295 = alloca i32*
    %malloccall296 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr297 = bitcast i8* %malloccall296 to i32*
    store i32* %tmpAddr297, i32** %tmp295
    %tmpWithVal298 = load i32*, i32** %tmp295
    store i32 1, i32* %tmpWithVal298
    %asChar299 = bitcast i32* %tmpWithVal298 to i8*
    %tmp300 = alloca i32*
    %malloccall301 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr302 = bitcast i8* %malloccall301 to i32*
    store i32* %tmpAddr302, i32** %tmp300
    %tmpWithVal303 = load i32*, i32** %tmp300
    store i32 1, i32* %tmpWithVal303
    %asChar304 = bitcast i32* %tmpWithVal303 to i8*
    %tmp305 = alloca i32*
    %malloccall306 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr307 = bitcast i8* %malloccall306 to i32*
    store i32* %tmpAddr307, i32** %tmp305
    %tmpWithVal308 = load i32*, i32** %tmp305
    store i32 1, i32* %tmpWithVal308
    %asChar309 = bitcast i32* %tmpWithVal308 to i8*
    %tmp310 = alloca i32*
    %malloccall311 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr312 = bitcast i8* %malloccall311 to i32*
    store i32* %tmpAddr312, i32** %tmp310
    %tmpWithVal313 = load i32*, i32** %tmp310
    store i32 1, i32* %tmpWithVal313
    %asChar314 = bitcast i32* %tmpWithVal313 to i8*
    %tmp315 = alloca i32*
    %malloccall316 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr317 = bitcast i8* %malloccall316 to i32*
    store i32* %tmpAddr317, i32** %tmp315
    %tmpWithVal318 = load i32*, i32** %tmp315
    store i32 1, i32* %tmpWithVal318
    %asChar319 = bitcast i32* %tmpWithVal318 to i8*
    %tmp320 = alloca i32*
    %malloccall321 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr322 = bitcast i8* %malloccall321 to i32*
    store i32* %tmpAddr322, i32** %tmp320
    %tmpWithVal323 = load i32*, i32** %tmp320
    store i32 1, i32* %tmpWithVal323
    %asChar324 = bitcast i32* %tmpWithVal323 to i8*
    %tmp325 = alloca i32*
    %malloccall326 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr327 = bitcast i8* %malloccall326 to i32*
    store i32* %tmpAddr327, i32** %tmp325
    %tmpWithVal328 = load i32*, i32** %tmp325
    store i32 1, i32* %tmpWithVal328
    %asChar329 = bitcast i32* %tmpWithVal328 to i8*
    %push_back330 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar9)
    %push_back331 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar14)
    %push_back332 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar19)
    %push_back333 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar24)
    %push_back334 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar29)
    %push_back335 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar34)
```

```
%push_back336 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar39)
%push_back337 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar44)
%push_back338 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar49)
%push_back339 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar54)
%push_back340 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar59)
%push_back341 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar64)
%push_back342 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar69)
%push_back343 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar74)
%push_back344 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar79)
%push_back345 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar84)
%push_back346 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar89)
%push_back347 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar94)
%push_back348 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar99)
%push_back349 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar104)
%push_back350 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar109)
%push_back351 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar114)
%push_back352 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar119)
%push_back353 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar124)
%push_back354 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar129)
%push_back355 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar134)
%push_back356 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar139)
%push_back357 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar144)
```

```
  %push_back358 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar149)
  %push_back359 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar154)
  %push_back360 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar159)
  %push_back361 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar164)
  %push_back362 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar169)
  %push_back363 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar174)
  %push_back364 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar179)
  %push_back365 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar184)
  %push_back366 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar189)
  %push_back367 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar194)
  %push_back368 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar199)
  %push_back369 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar204)
  %push_back370 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar209)
  %push_back371 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar214)
  %push_back372 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar219)
  %push_back373 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar224)
  %push_back374 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar229)
  %push_back375 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar234)
  %push_back376 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar239)
  %push_back377 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar244)
  %push_back378 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar249)
  %push_back379 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar254)
  %push_back380 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar259)
  %push_back381 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar264)
  %push_back382 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar269)
  %push_back383 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar274)
  %push_back384 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar279)
  %push_back385 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar284)
  %push_back386 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar289)
  %push_back387 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar294)
  %push_back388 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar299)
  %push_back389 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar304)
  %push_back390 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar309)
  %push_back391 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar314)
  %push_back392 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar319)
  %push_back393 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar324)
  %push_back394 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr3, i8* %asChar329)
  store %list* %listAddr3, %list** %input_text
  %malloccall395 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr396 = bitcast i8* %malloccall395 to %list*
  %init_list397 = call i32 (%list*, ...) @init_list(%list* %listAddr396)
  %tmp398 = alloca i8*
  %tmpAddr400 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr400, i8** %tmp398
  %tmpWithVal401 = load i8*, i8** %tmp398
  store i8 90, i8* %tmpWithVal401
  %tmp402 = alloca i8*
  %tmpAddr404 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr404, i8** %tmp402
  %tmpWithVal405 = load i8*, i8** %tmp402
  store i8 69, i8* %tmpWithVal405
  %tmp406 = alloca i8*
  %tmpAddr408 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr408, i8** %tmp406
  %tmpWithVal409 = load i8*, i8** %tmp406
  store i8 82, i8* %tmpWithVal409
  %tmp410 = alloca i8*
  %tmpAddr412 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr412, i8** %tmp410
  %tmpWithVal413 = load i8*, i8** %tmp410
  store i8 79, i8* %tmpWithVal413
  %tmp414 = alloca i8*
  %tmpAddr416 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr416, i8** %tmp414
  %tmpWithVal417 = load i8*, i8** %tmp414
  store i8 83, i8* %tmpWithVal417
  %tmp418 = alloca i8*
  %tmpAddr420 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr420, i8** %tmp418
  %tmpWithVal421 = load i8*, i8** %tmp418
  store i8 73, i8* %tmpWithVal421
  %tmp422 = alloca i8*
  %tmpAddr424 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr424, i8** %tmp422
  %tmpWithVal425 = load i8*, i8** %tmp422
  store i8 88, i8* %tmpWithVal425
  %tmp426 = alloca i8*
  %tmpAddr428 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr428, i8** %tmp426
  %tmpWithVal429 = load i8*, i8** %tmp426
  store i8 72, i8* %tmpWithVal429
  %tmp430 = alloca i8*
  %tmpAddr432 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr432, i8** %tmp430
  %tmpWithVal433 = load i8*, i8** %tmp430
```

```
store i8 85, i8* %tmpWithVal433
%tmp434 = alloca i8*
%tmpAddr436 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr436, i8** %tmp434
%tmpWithVal437 = load i8*, i8** %tmp434
store i8 78, i8* %tmpWithVal437
%tmp438 = alloca i8*
%tmpAddr440 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr440, i8** %tmp438
%tmpWithVal441 = load i8*, i8** %tmp438
store i8 68, i8* %tmpWithVal441
%tmp442 = alloca i8*
%tmpAddr444 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr444, i8** %tmp442
%tmpWithVal445 = load i8*, i8** %tmp442
store i8 82, i8* %tmpWithVal445
%tmp446 = alloca i8*
%tmpAddr448 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr448, i8** %tmp446
%tmpWithVal449 = load i8*, i8** %tmp446
store i8 69, i8* %tmpWithVal449
%tmp450 = alloca i8*
%tmpAddr452 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr452, i8** %tmp450
%tmpWithVal453 = load i8*, i8** %tmp450
store i8 69, i8* %tmpWithVal453
%tmp454 = alloca i8*
%tmpAddr456 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr456, i8** %tmp454
%tmpWithVal457 = load i8*, i8** %tmp454
store i8 68, i8* %tmpWithVal457
%tmp458 = alloca i8*
%tmpAddr460 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr460, i8** %tmp458
%tmpWithVal461 = load i8*, i8** %tmp458
store i8 72, i8* %tmpWithVal461
%tmp462 = alloca i8*
```

```
%tmpAddr464 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr464, i8** %tmp462
%tmpWithVal465 = load i8*, i8** %tmp462
store i8 79, i8* %tmpWithVal465
%tmp466 = alloca i8*
%tmpAddr468 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr468, i8** %tmp466
%tmpWithVal469 = load i8*, i8** %tmp466
store i8 85, i8* %tmpWithVal469
%tmp470 = alloca i8*
%tmpAddr472 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr472, i8** %tmp470
%tmpWithVal473 = load i8*, i8** %tmp470
store i8 82, i8* %tmpWithVal473
%tmp474 = alloca i8*
%tmpAddr476 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr476, i8** %tmp474
%tmpWithVal477 = load i8*, i8** %tmp474
store i8 83, i8* %tmpWithVal477
%tmp478 = alloca i8*
%tmpAddr480 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr480, i8** %tmp478
%tmpWithVal481 = load i8*, i8** %tmp478
store i8 87, i8* %tmpWithVal481
%tmp482 = alloca i8*
%tmpAddr484 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr484, i8** %tmp482
%tmpWithVal485 = load i8*, i8** %tmp482
store i8 69, i8* %tmpWithVal485
%tmp486 = alloca i8*
%tmpAddr488 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr488, i8** %tmp486
%tmpWithVal489 = load i8*, i8** %tmp486
store i8 65, i8* %tmpWithVal489
%tmp490 = alloca i8*
%tmpAddr492 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
```

```
store i8* %tmpAddr492, i8** %tmp490
%tmpWithVal493 = load i8*, i8** %tmp490
store i8 84, i8* %tmpWithVal493
%tmp494 = alloca i8*
%tmpAddr496 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr496, i8** %tmp494
%tmpWithVal497 = load i8*, i8** %tmp494
store i8 72, i8* %tmpWithVal497
%tmp498 = alloca i8*
%tmpAddr500 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr500, i8** %tmp498
%tmpWithVal501 = load i8*, i8** %tmp498
store i8 69, i8* %tmpWithVal501
%tmp502 = alloca i8*
%tmpAddr504 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr504, i8** %tmp502
%tmpWithVal505 = load i8*, i8** %tmp502
store i8 82, i8* %tmpWithVal505
%tmp506 = alloca i8*
%tmpAddr508 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr508, i8** %tmp506
%tmpWithVal509 = load i8*, i8** %tmp506
store i8 84, i8* %tmpWithVal509
%tmp510 = alloca i8*
%tmpAddr512 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr512, i8** %tmp510
%tmpWithVal513 = load i8*, i8** %tmp510
store i8 79, i8* %tmpWithVal513
%tmp514 = alloca i8*
%tmpAddr516 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr516, i8** %tmp514
%tmpWithVal517 = load i8*, i8** %tmp514
store i8 68, i8* %tmpWithVal517
%tmp518 = alloca i8*
%tmpAddr520 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr520, i8** %tmp518
%tmpWithVal521 = load i8*, i8** %tmp518
```

```
  store i8 65, i8* %tmpWithVal521
  %tmp522 = alloca i8*
  %tmpAddr524 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr524, i8** %tmp522
  %tmpWithVal525 = load i8*, i8** %tmp522
  store i8 89, i8* %tmpWithVal525
  %tmp526 = alloca i8*
  %tmpAddr528 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr528, i8** %tmp526
  %tmpWithVal529 = load i8*, i8** %tmp526
  store i8 73, i8* %tmpWithVal529
  %tmp530 = alloca i8*
  %tmpAddr532 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr532, i8** %tmp530
  %tmpWithVal533 = load i8*, i8** %tmp530
  store i8 83, i8* %tmpWithVal533
  %tmp534 = alloca i8*
  %tmpAddr536 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr536, i8** %tmp534
  %tmpWithVal537 = load i8*, i8** %tmp534
  store i8 67, i8* %tmpWithVal537
  %tmp538 = alloca i8*
  %tmpAddr540 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr540, i8** %tmp538
  %tmpWithVal541 = load i8*, i8** %tmp538
  store i8 76, i8* %tmpWithVal541
  %tmp542 = alloca i8*
  %tmpAddr544 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr544, i8** %tmp542
  %tmpWithVal545 = load i8*, i8** %tmp542
  store i8 69, i8* %tmpWithVal545
  %tmp546 = alloca i8*
  %tmpAddr548 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr548, i8** %tmp546
  %tmpWithVal549 = load i8*, i8** %tmp546
  store i8 65, i8* %tmpWithVal549
  %tmp550 = alloca i8*
```

```
    %tmpAddr552 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr552, i8** %tmp550
    %tmpWithVal553 = load i8*, i8** %tmp550
    store i8 82, i8* %tmpWithVal553
    %tmp554 = alloca i8*
    %tmpAddr556 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr556, i8** %tmp554
    %tmpWithVal557 = load i8*, i8** %tmp554
    store i8 82, i8* %tmpWithVal557
    %tmp558 = alloca i8*
    %tmpAddr560 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr560, i8** %tmp558
    %tmpWithVal561 = load i8*, i8** %tmp558
    store i8 65, i8* %tmpWithVal561
    %tmp562 = alloca i8*
    %tmpAddr564 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr564, i8** %tmp562
    %tmpWithVal565 = load i8*, i8** %tmp562
    store i8 73, i8* %tmpWithVal565
    %tmp566 = alloca i8*
    %tmpAddr568 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr568, i8** %tmp566
    %tmpWithVal569 = load i8*, i8** %tmp566
    store i8 78, i8* %tmpWithVal569
    %tmp570 = alloca i8*
    %tmpAddr572 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr572, i8** %tmp570
    %tmpWithVal573 = load i8*, i8** %tmp570
    store i8 73, i8* %tmpWithVal573
    %tmp574 = alloca i8*
    %tmpAddr576 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr576, i8** %tmp574
    %tmpWithVal577 = load i8*, i8** %tmp574
    store i8 78, i8* %tmpWithVal577
    %tmp578 = alloca i8*
    %tmpAddr580 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
```

```
  store i8* %tmpAddr580, i8** %tmp578
  %tmpWithVal581 = load i8*, i8** %tmp578
  store i8 84, i8* %tmpWithVal581
  %tmp582 = alloca i8*
  %tmpAddr584 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr584, i8** %tmp582
  %tmpWithVal585 = load i8*, i8** %tmp582
  store i8 72, i8* %tmpWithVal585
  %tmp586 = alloca i8*
  %tmpAddr588 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr588, i8** %tmp586
  %tmpWithVal589 = load i8*, i8** %tmp586
  store i8 69, i8* %tmpWithVal589
  %tmp590 = alloca i8*
  %tmpAddr592 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr592, i8** %tmp590
  %tmpWithVal593 = load i8*, i8** %tmp590
  store i8 69, i8* %tmpWithVal593
  %tmp594 = alloca i8*
  %tmpAddr596 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr596, i8** %tmp594
  %tmpWithVal597 = load i8*, i8** %tmp594
  store i8 86, i8* %tmpWithVal597
  %tmp598 = alloca i8*
  %tmpAddr600 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr600, i8** %tmp598
  %tmpWithVal601 = load i8*, i8** %tmp598
  store i8 69, i8* %tmpWithVal601
  %tmp602 = alloca i8*
  %tmpAddr604 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr604, i8** %tmp602
  %tmpWithVal605 = load i8*, i8** %tmp602
  store i8 78, i8* %tmpWithVal605
  %tmp606 = alloca i8*
  %tmpAddr608 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr608, i8** %tmp606
  %tmpWithVal609 = load i8*, i8** %tmp606
```

```
store i8 73, i8* %tmpWithVal609
%tmp610 = alloca i8*
%tmpAddr612 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr612, i8** %tmp610
%tmpWithVal613 = load i8*, i8** %tmp610
store i8 78, i8* %tmpWithVal613
%tmp614 = alloca i8*
%tmpAddr616 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr616, i8** %tmp614
%tmpWithVal617 = load i8*, i8** %tmp614
store i8 71, i8* %tmpWithVal617
%tmp618 = alloca i8*
%tmpAddr620 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr620, i8** %tmp618
%tmpWithVal621 = load i8*, i8** %tmp618
store i8 72, i8* %tmpWithVal621
%tmp622 = alloca i8*
%tmpAddr624 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr624, i8** %tmp622
%tmpWithVal625 = load i8*, i8** %tmp622
store i8 69, i8* %tmpWithVal625
%tmp626 = alloca i8*
%tmpAddr628 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr628, i8** %tmp626
%tmpWithVal629 = load i8*, i8** %tmp626
store i8 73, i8* %tmpWithVal629
%tmp630 = alloca i8*
%tmpAddr632 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr632, i8** %tmp630
%tmpWithVal633 = load i8*, i8** %tmp630
store i8 76, i8* %tmpWithVal633
%tmp634 = alloca i8*
%tmpAddr636 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr636, i8** %tmp634
%tmpWithVal637 = load i8*, i8** %tmp634
store i8 72, i8* %tmpWithVal637
%tmp638 = alloca i8*
```

```
    %tmpAddr640 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr640, i8** %tmp638
    %tmpWithVal641 = load i8*, i8** %tmp638
    store i8 73, i8* %tmpWithVal641
    %tmp642 = alloca i8*
    %tmpAddr644 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr644, i8** %tmp642
    %tmpWithVal645 = load i8*, i8** %tmp642
    store i8 84, i8* %tmpWithVal645
    %tmp646 = alloca i8*
    %tmpAddr648 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr648, i8** %tmp646
    %tmpWithVal649 = load i8*, i8** %tmp646
    store i8 76, i8* %tmpWithVal649
    %tmp650 = alloca i8*
    %tmpAddr652 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr652, i8** %tmp650
    %tmpWithVal653 = load i8*, i8** %tmp650
    store i8 69, i8* %tmpWithVal653
    %tmp654 = alloca i8*
    %tmpAddr656 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
    store i8* %tmpAddr656, i8** %tmp654
    %tmpWithVal657 = load i8*, i8** %tmp654
    store i8 82, i8* %tmpWithVal657
    %push_back658 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal401)
    %push_back659 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal405)
    %push_back660 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal409)
    %push_back661 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal413)
    %push_back662 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal417)
    %push_back663 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal421)
    %push_back664 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal425)
```

```
%push_back665 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal429)
%push_back666 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal433)
%push_back667 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal437)
%push_back668 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal441)
%push_back669 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal445)
%push_back670 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal449)
%push_back671 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal453)
%push_back672 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal457)
%push_back673 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal461)
%push_back674 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal465)
%push_back675 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal469)
%push_back676 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal473)
%push_back677 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal477)
%push_back678 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal481)
%push_back679 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal485)
%push_back680 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal489)
%push_back681 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal493)
%push_back682 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal497)
%push_back683 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal501)
%push_back684 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal505)
%push_back685 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal509)
%push_back686 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal513)
```

```
%push_back687 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal517)
%push_back688 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal521)
%push_back689 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal525)
%push_back690 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal529)
%push_back691 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal533)
%push_back692 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal537)
%push_back693 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal541)
%push_back694 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal545)
%push_back695 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal549)
%push_back696 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal553)
%push_back697 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal557)
%push_back698 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal561)
%push_back699 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal565)
%push_back700 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal569)
%push_back701 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal573)
%push_back702 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal577)
%push_back703 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal581)
%push_back704 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal585)
%push_back705 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal589)
%push_back706 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal593)
%push_back707 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal597)
%push_back708 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal601)
```

```
%push_back709 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal605)
%push_back710 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal609)
%push_back711 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal613)
%push_back712 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal617)
%push_back713 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal621)
%push_back714 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal625)
%push_back715 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal629)
%push_back716 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal633)
%push_back717 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal637)
%push_back718 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal641)
%push_back719 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal645)
%push_back720 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal649)
%push_back721 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal653)
%push_back722 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr396, i8* %tmpWithVal657)
store %list* %listAddr396, %list** %messagez
%messagez723 = load %list*, %list** %messagez
%stringLength = call i32 @stringLength(%list* %messagez723)
store i32 %stringLength, i32* %length_input
%malloccall724 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr725 = bitcast i8* %malloccall724 to %list*
%init_list726 = call i32 (%list*, ...) @init_list(%list* %listAddr725)
%tmp727 = alloca i32*
%malloccall728 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr729 = bitcast i8* %malloccall728 to i32*
store i32* %tmpAddr729, i32** %tmp727
%tmpWithVal730 = load i32*, i32** %tmp727
store i32 18, i32* %tmpWithVal730
%asChar731 = bitcast i32* %tmpWithVal730 to i8*
```

```
    %tmp732 = alloca i32*
    %malloccall733 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr734 = bitcast i8* %malloccall733 to i32*
    store i32* %tmpAddr734, i32** %tmp732
    %tmpWithVal735 = load i32*, i32** %tmp732
    store i32 8, i32* %tmpWithVal735
    %asChar736 = bitcast i32* %tmpWithVal735 to i8*
    %tmp737 = alloca i32*
    %malloccall738 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr739 = bitcast i8* %malloccall738 to i32*
    store i32* %tmpAddr739, i32** %tmp737
    %tmpWithVal740 = load i32*, i32** %tmp737
    store i32 6, i32* %tmpWithVal740
    %asChar741 = bitcast i32* %tmpWithVal740 to i8*
    %push_back742 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr725, i8* %asChar731)
    %push_back743 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr725, i8* %asChar736)
    %push_back744 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr725, i8* %asChar741)
    store %list* %listAddr725, %list** %key
    %malloccall745 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
    %listAddr746 = bitcast i8* %malloccall745 to %list*
    %init_list747 = call i32 (%list*, ...) @init_list(%list* %listAddr746)
    %tmp748 = alloca i32*
    %malloccall749 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr750 = bitcast i8* %malloccall749 to i32*
    store i32* %tmpAddr750, i32** %tmp748
    %tmpWithVal751 = load i32*, i32** %tmp748
    store i32 15, i32* %tmpWithVal751
    %asChar752 = bitcast i32* %tmpWithVal751 to i8*
    %tmp753 = alloca i32*
    %malloccall754 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr755 = bitcast i8* %malloccall754 to i32*
    store i32* %tmpAddr755, i32** %tmp753
    %tmpWithVal756 = load i32*, i32** %tmp753
    store i32 12, i32* %tmpWithVal756
    %asChar757 = bitcast i32* %tmpWithVal756 to i8*
    %tmp758 = alloca i32*
```

```
%malloccall759 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr760 = bitcast i8* %malloccall759 to i32*
store i32* %tmpAddr760, i32** %tmp758
%tmpWithVal761 = load i32*, i32** %tmp758
store i32 15, i32* %tmpWithVal761
%asChar762 = bitcast i32* %tmpWithVal761 to i8*
%push_back763 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr746, i8* %asChar752)
%push_back764 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr746, i8* %asChar757)
%push_back765 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr746, i8* %asChar762)
store %list* %listAddr746, %list** %ring_setting
%malloccall766 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr767 = bitcast i8* %malloccall766 to %list*
%init_list768 = call i32 (%list*, ...) @init_list(%list* %listAddr767)
%tmp769 = alloca i32*
%malloccall770 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr771 = bitcast i8* %malloccall770 to i32*
store i32* %tmpAddr771, i32** %tmp769
%tmpWithVal772 = load i32*, i32** %tmp769
store i32 1, i32* %tmpWithVal772
%asChar773 = bitcast i32* %tmpWithVal772 to i8*
%tmp774 = alloca i32*
%malloccall775 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr776 = bitcast i8* %malloccall775 to i32*
store i32* %tmpAddr776, i32** %tmp774
%tmpWithVal777 = load i32*, i32** %tmp774
store i32 2, i32* %tmpWithVal777
%asChar778 = bitcast i32* %tmpWithVal777 to i8*
%tmp779 = alloca i32*
%malloccall780 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr781 = bitcast i8* %malloccall780 to i32*
store i32* %tmpAddr781, i32** %tmp779
%tmpWithVal782 = load i32*, i32** %tmp779
store i32 3, i32* %tmpWithVal782
%asChar783 = bitcast i32* %tmpWithVal782 to i8*
%tmp784 = alloca i32*
```

```
%malloccall785 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr786 = bitcast i8* %malloccall785 to i32*
store i32* %tmpAddr786, i32** %tmp784
%tmpWithVal787 = load i32*, i32** %tmp784
store i32 4, i32* %tmpWithVal787
%asChar788 = bitcast i32* %tmpWithVal787 to i8*
%tmp789 = alloca i32*
%malloccall790 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr791 = bitcast i8* %malloccall790 to i32*
store i32* %tmpAddr791, i32** %tmp789
%tmpWithVal792 = load i32*, i32** %tmp789
store i32 5, i32* %tmpWithVal792
%asChar793 = bitcast i32* %tmpWithVal792 to i8*
%tmp794 = alloca i32*
%malloccall795 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr796 = bitcast i8* %malloccall795 to i32*
store i32* %tmpAddr796, i32** %tmp794
%tmpWithVal797 = load i32*, i32** %tmp794
store i32 6, i32* %tmpWithVal797
%asChar798 = bitcast i32* %tmpWithVal797 to i8*
%tmp799 = alloca i32*
%malloccall800 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr801 = bitcast i8* %malloccall800 to i32*
store i32* %tmpAddr801, i32** %tmp799
%tmpWithVal802 = load i32*, i32** %tmp799
store i32 9, i32* %tmpWithVal802
%asChar803 = bitcast i32* %tmpWithVal802 to i8*
%tmp804 = alloca i32*
%malloccall805 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr806 = bitcast i8* %malloccall805 to i32*
store i32* %tmpAddr806, i32** %tmp804
%tmpWithVal807 = load i32*, i32** %tmp804
store i32 13, i32* %tmpWithVal807
%asChar808 = bitcast i32* %tmpWithVal807 to i8*
%tmp809 = alloca i32*
%malloccall810 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr811 = bitcast i8* %malloccall810 to i32*
store i32* %tmpAddr811, i32** %tmp809
```

```
%tmpWithVal812 = load i32*, i32** %tmp809
store i32 10, i32* %tmpWithVal812
%asChar813 = bitcast i32* %tmpWithVal812 to i8*
%tmp814 = alloca i32*
%malloccall815 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr816 = bitcast i8* %malloccall815 to i32*
store i32* %tmpAddr816, i32** %tmp814
%tmpWithVal817 = load i32*, i32** %tmp814
store i32 21, i32* %tmpWithVal817
%asChar818 = bitcast i32* %tmpWithVal817 to i8*
%push_back819 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar773)
%push_back820 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar778)
%push_back821 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar783)
%push_back822 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar788)
%push_back823 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar793)
%push_back824 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar798)
%push_back825 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar803)
%push_back826 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar808)
%push_back827 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar813)
%push_back828 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr767, i8* %asChar818)
store %list* %listAddr767, %list** %first_plugs_init
%malloccall829 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr830 = bitcast i8* %malloccall829 to %list*
%init_list831 = call i32 (%list*, ...) @init_list(%list* %listAddr830)
%tmp832 = alloca i32*
%malloccall833 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr834 = bitcast i8* %malloccall833 to i32*
store i32* %tmpAddr834, i32** %tmp832
%tmpWithVal835 = load i32*, i32** %tmp832
store i32 18, i32* %tmpWithVal835
%asChar836 = bitcast i32* %tmpWithVal835 to i8*
```

```
    %tmp837 = alloca i32*
    %malloccall838 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr839 = bitcast i8* %malloccall838 to i32*
    store i32* %tmpAddr839, i32** %tmp837
    %tmpWithVal840 = load i32*, i32** %tmp837
    store i32 8, i32* %tmpWithVal840
    %asChar841 = bitcast i32* %tmpWithVal840 to i8*
    %tmp842 = alloca i32*
    %malloccall843 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr844 = bitcast i8* %malloccall843 to i32*
    store i32* %tmpAddr844, i32** %tmp842
    %tmpWithVal845 = load i32*, i32** %tmp842
    store i32 12, i32* %tmpWithVal845
    %asChar846 = bitcast i32* %tmpWithVal845 to i8*
    %tmp847 = alloca i32*
    %malloccall848 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr849 = bitcast i8* %malloccall848 to i32*
    store i32* %tmpAddr849, i32** %tmp847
    %tmpWithVal850 = load i32*, i32** %tmp847
    store i32 16, i32* %tmpWithVal850
    %asChar851 = bitcast i32* %tmpWithVal850 to i8*
    %tmp852 = alloca i32*
    %malloccall853 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr854 = bitcast i8* %malloccall853 to i32*
    store i32* %tmpAddr854, i32** %tmp852
    %tmpWithVal855 = load i32*, i32** %tmp852
    store i32 20, i32* %tmpWithVal855
    %asChar856 = bitcast i32* %tmpWithVal855 to i8*
    %tmp857 = alloca i32*
    %malloccall858 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr859 = bitcast i8* %malloccall858 to i32*
    store i32* %tmpAddr859, i32** %tmp857
    %tmpWithVal860 = load i32*, i32** %tmp857
    store i32 24, i32* %tmpWithVal860
    %asChar861 = bitcast i32* %tmpWithVal860 to i8*
    %tmp862 = alloca i32*
    %malloccall863 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr864 = bitcast i8* %malloccall863 to i32*
```

```
	store i32* %tmpAddr864, i32** %tmp862
	%tmpWithVal865 = load i32*, i32** %tmp862
	store i32 11, i32* %tmpWithVal865
	%asChar866 = bitcast i32* %tmpWithVal865 to i8*
	%tmp867 = alloca i32*
	%malloccall868 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr869 = bitcast i8* %malloccall868 to i32*
	store i32* %tmpAddr869, i32** %tmp867
	%tmpWithVal870 = load i32*, i32** %tmp867
	store i32 15, i32* %tmpWithVal870
	%asChar871 = bitcast i32* %tmpWithVal870 to i8*
	%tmp872 = alloca i32*
	%malloccall873 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr874 = bitcast i8* %malloccall873 to i32*
	store i32* %tmpAddr874, i32** %tmp872
	%tmpWithVal875 = load i32*, i32** %tmp872
	store i32 25, i32* %tmpWithVal875
	%asChar876 = bitcast i32* %tmpWithVal875 to i8*
	%tmp877 = alloca i32*
	%malloccall878 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr879 = bitcast i8* %malloccall878 to i32*
	store i32* %tmpAddr879, i32** %tmp877
	%tmpWithVal880 = load i32*, i32** %tmp877
	store i32 22, i32* %tmpWithVal880
	%asChar881 = bitcast i32* %tmpWithVal880 to i8*
	%push_back882 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar836)
	%push_back883 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar841)
	%push_back884 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar846)
	%push_back885 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar851)
	%push_back886 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar856)
	%push_back887 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar861)
	%push_back888 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar866)
	%push_back889 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar871)
```

```
  %push_back890 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar876)
  %push_back891 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr830, i8* %asChar881)
  store %list* %listAddr830, %list** %second_plugs_init
  %malloccall892 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr893 = bitcast i8* %malloccall892 to %list*
  %init_list894 = call i32 (%list*, ...) @init_list(%list* %listAddr893)
  %tmp895 = alloca i32*
  %malloccall896 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr897 = bitcast i8* %malloccall896 to i32*
  store i32* %tmpAddr897, i32** %tmp895
  %tmpWithVal898 = load i32*, i32** %tmp895
  store i32 1, i32* %tmpWithVal898
  %asChar899 = bitcast i32* %tmpWithVal898 to i8*
  %tmp900 = alloca i32*
  %malloccall901 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr902 = bitcast i8* %malloccall901 to i32*
  store i32* %tmpAddr902, i32** %tmp900
  %tmpWithVal903 = load i32*, i32** %tmp900
  store i32 3, i32* %tmpWithVal903
  %asChar904 = bitcast i32* %tmpWithVal903 to i8*
  %tmp905 = alloca i32*
  %malloccall906 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr907 = bitcast i8* %malloccall906 to i32*
  store i32* %tmpAddr907, i32** %tmp905
  %tmpWithVal908 = load i32*, i32** %tmp905
  store i32 5, i32* %tmpWithVal908
  %asChar909 = bitcast i32* %tmpWithVal908 to i8*
  %tmp910 = alloca i32*
  %malloccall911 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr912 = bitcast i8* %malloccall911 to i32*
  store i32* %tmpAddr912, i32** %tmp910
  %tmpWithVal913 = load i32*, i32** %tmp910
  store i32 7, i32* %tmpWithVal913
  %asChar914 = bitcast i32* %tmpWithVal913 to i8*
  %tmp915 = alloca i32*
  %malloccall916 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
%tmpAddr917 = bitcast i8* %malloccall916 to i32*
store i32* %tmpAddr917, i32** %tmp915
%tmpWithVal918 = load i32*, i32** %tmp915
store i32 9, i32* %tmpWithVal918
%asChar919 = bitcast i32* %tmpWithVal918 to i8*
%tmp920 = alloca i32*
%malloccall921 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr922 = bitcast i8* %malloccall921 to i32*
store i32* %tmpAddr922, i32** %tmp920
%tmpWithVal923 = load i32*, i32** %tmp920
store i32 11, i32* %tmpWithVal923
%asChar924 = bitcast i32* %tmpWithVal923 to i8*
%tmp925 = alloca i32*
%malloccall926 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr927 = bitcast i8* %malloccall926 to i32*
store i32* %tmpAddr927, i32** %tmp925
%tmpWithVal928 = load i32*, i32** %tmp925
store i32 2, i32* %tmpWithVal928
%asChar929 = bitcast i32* %tmpWithVal928 to i8*
%tmp930 = alloca i32*
%malloccall931 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr932 = bitcast i8* %malloccall931 to i32*
store i32* %tmpAddr932, i32** %tmp930
%tmpWithVal933 = load i32*, i32** %tmp930
store i32 15, i32* %tmpWithVal933
%asChar934 = bitcast i32* %tmpWithVal933 to i8*
%tmp935 = alloca i32*
%malloccall936 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr937 = bitcast i8* %malloccall936 to i32*
store i32* %tmpAddr937, i32** %tmp935
%tmpWithVal938 = load i32*, i32** %tmp935
store i32 17, i32* %tmpWithVal938
%asChar939 = bitcast i32* %tmpWithVal938 to i8*
%tmp940 = alloca i32*
%malloccall941 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr942 = bitcast i8* %malloccall941 to i32*
store i32* %tmpAddr942, i32** %tmp940
%tmpWithVal943 = load i32*, i32** %tmp940
store i32 19, i32* %tmpWithVal943
```

```
%asChar944 = bitcast i32* %tmpWithVal943 to i8*
%tmp945 = alloca i32*
%malloccall946 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr947 = bitcast i8* %malloccall946 to i32*
store i32* %tmpAddr947, i32** %tmp945
%tmpWithVal948 = load i32*, i32** %tmp945
store i32 23, i32* %tmpWithVal948
%asChar949 = bitcast i32* %tmpWithVal948 to i8*
%tmp950 = alloca i32*
%malloccall951 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr952 = bitcast i8* %malloccall951 to i32*
store i32* %tmpAddr952, i32** %tmp950
%tmpWithVal953 = load i32*, i32** %tmp950
store i32 21, i32* %tmpWithVal953
%asChar954 = bitcast i32* %tmpWithVal953 to i8*
%tmp955 = alloca i32*
%malloccall956 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr957 = bitcast i8* %malloccall956 to i32*
store i32* %tmpAddr957, i32** %tmp955
%tmpWithVal958 = load i32*, i32** %tmp955
store i32 25, i32* %tmpWithVal958
%asChar959 = bitcast i32* %tmpWithVal958 to i8*
%tmp960 = alloca i32*
%malloccall961 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr962 = bitcast i8* %malloccall961 to i32*
store i32* %tmpAddr962, i32** %tmp960
%tmpWithVal963 = load i32*, i32** %tmp960
store i32 13, i32* %tmpWithVal963
%asChar964 = bitcast i32* %tmpWithVal963 to i8*
%tmp965 = alloca i32*
%malloccall966 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr967 = bitcast i8* %malloccall966 to i32*
store i32* %tmpAddr967, i32** %tmp965
%tmpWithVal968 = load i32*, i32** %tmp965
store i32 24, i32* %tmpWithVal968
%asChar969 = bitcast i32* %tmpWithVal968 to i8*
%tmp970 = alloca i32*
%malloccall971 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
  %tmpAddr972 = bitcast i8* %malloccall971 to i32*
  store i32* %tmpAddr972, i32** %tmp970
  %tmpWithVal973 = load i32*, i32** %tmp970
  store i32 4, i32* %tmpWithVal973
  %asChar974 = bitcast i32* %tmpWithVal973 to i8*
  %tmp975 = alloca i32*
  %malloccall976 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr977 = bitcast i8* %malloccall976 to i32*
  store i32* %tmpAddr977, i32** %tmp975
  %tmpWithVal978 = load i32*, i32** %tmp975
  store i32 8, i32* %tmpWithVal978
  %asChar979 = bitcast i32* %tmpWithVal978 to i8*
  %tmp980 = alloca i32*
  %malloccall981 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr982 = bitcast i8* %malloccall981 to i32*
  store i32* %tmpAddr982, i32** %tmp980
  %tmpWithVal983 = load i32*, i32** %tmp980
  store i32 22, i32* %tmpWithVal983
  %asChar984 = bitcast i32* %tmpWithVal983 to i8*
  %tmp985 = alloca i32*
  %malloccall986 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr987 = bitcast i8* %malloccall986 to i32*
  store i32* %tmpAddr987, i32** %tmp985
  %tmpWithVal988 = load i32*, i32** %tmp985
  store i32 6, i32* %tmpWithVal988
  %asChar989 = bitcast i32* %tmpWithVal988 to i8*
  %tmp990 = alloca i32*
  %malloccall991 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr992 = bitcast i8* %malloccall991 to i32*
  store i32* %tmpAddr992, i32** %tmp990
  %tmpWithVal993 = load i32*, i32** %tmp990
  store i32 0, i32* %tmpWithVal993
  %asChar994 = bitcast i32* %tmpWithVal993 to i8*
  %tmp995 = alloca i32*
  %malloccall996 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr997 = bitcast i8* %malloccall996 to i32*
  store i32* %tmpAddr997, i32** %tmp995
  %tmpWithVal998 = load i32*, i32** %tmp995
  store i32 10, i32* %tmpWithVal998
```

```
%asChar999 = bitcast i32* %tmpWithVal998 to i8*
%tmp1000 = alloca i32*
%malloccall1001 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1002 = bitcast i8* %malloccall1001 to i32*
store i32* %tmpAddr1002, i32** %tmp1000
%tmpWithVal1003 = load i32*, i32** %tmp1000
store i32 12, i32* %tmpWithVal1003
%asChar1004 = bitcast i32* %tmpWithVal1003 to i8*
%tmp1005 = alloca i32*
%malloccall1006 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1007 = bitcast i8* %malloccall1006 to i32*
store i32* %tmpAddr1007, i32** %tmp1005
%tmpWithVal1008 = load i32*, i32** %tmp1005
store i32 20, i32* %tmpWithVal1008
%asChar1009 = bitcast i32* %tmpWithVal1008 to i8*
%tmp1010 = alloca i32*
%malloccall1011 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1012 = bitcast i8* %malloccall1011 to i32*
store i32* %tmpAddr1012, i32** %tmp1010
%tmpWithVal1013 = load i32*, i32** %tmp1010
store i32 18, i32* %tmpWithVal1013
%asChar1014 = bitcast i32* %tmpWithVal1013 to i8*
%tmp1015 = alloca i32*
%malloccall1016 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1017 = bitcast i8* %malloccall1016 to i32*
store i32* %tmpAddr1017, i32** %tmp1015
%tmpWithVal1018 = load i32*, i32** %tmp1015
store i32 16, i32* %tmpWithVal1018
%asChar1019 = bitcast i32* %tmpWithVal1018 to i8*
%tmp1020 = alloca i32*
%malloccall1021 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1022 = bitcast i8* %malloccall1021 to i32*
store i32* %tmpAddr1022, i32** %tmp1020
%tmpWithVal1023 = load i32*, i32** %tmp1020
store i32 14, i32* %tmpWithVal1023
%asChar1024 = bitcast i32* %tmpWithVal1023 to i8*
%push_back1025 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar899)
```

```
%push_back1026 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar904)
%push_back1027 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar909)
%push_back1028 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar914)
%push_back1029 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar919)
%push_back1030 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar924)
%push_back1031 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar929)
%push_back1032 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar934)
%push_back1033 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar939)
%push_back1034 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar944)
%push_back1035 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar949)
%push_back1036 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar954)
%push_back1037 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar959)
%push_back1038 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar964)
%push_back1039 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar969)
%push_back1040 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar974)
%push_back1041 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar979)
%push_back1042 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar984)
%push_back1043 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar989)
%push_back1044 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar994)
%push_back1045 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar999)
%push_back1046 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar1004)
%push_back1047 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar1009)
```

```
%push_back1048 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar1014)
%push_back1049 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar1019)
%push_back1050 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr893, i8* %asChar1024)
store %list* %listAddr893, %list** %rotor_three_out
%malloccall1051 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr1052 = bitcast i8* %malloccall1051 to %list*
%init_list1053 = call i32 (%list*, ...) @init_list(%list* %listAddr1052)
%tmp1054 = alloca i32*
%malloccall1055 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1056 = bitcast i8* %malloccall1055 to i32*
store i32* %tmpAddr1056, i32** %tmp1054
%tmpWithVal1057 = load i32*, i32** %tmp1054
store i32 0, i32* %tmpWithVal1057
%asChar1058 = bitcast i32* %tmpWithVal1057 to i8*
%tmp1059 = alloca i32*
%malloccall1060 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1061 = bitcast i8* %malloccall1060 to i32*
store i32* %tmpAddr1061, i32** %tmp1059
%tmpWithVal1062 = load i32*, i32** %tmp1059
store i32 1, i32* %tmpWithVal1062
%asChar1063 = bitcast i32* %tmpWithVal1062 to i8*
%tmp1064 = alloca i32*
%malloccall1065 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1066 = bitcast i8* %malloccall1065 to i32*
store i32* %tmpAddr1066, i32** %tmp1064
%tmpWithVal1067 = load i32*, i32** %tmp1064
store i32 2, i32* %tmpWithVal1067
%asChar1068 = bitcast i32* %tmpWithVal1067 to i8*
%tmp1069 = alloca i32*
%malloccall1070 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1071 = bitcast i8* %malloccall1070 to i32*
store i32* %tmpAddr1071, i32** %tmp1069
%tmpWithVal1072 = load i32*, i32** %tmp1069
store i32 3, i32* %tmpWithVal1072
%asChar1073 = bitcast i32* %tmpWithVal1072 to i8*
%tmp1074 = alloca i32*
```

```
  %malloccall1075 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1076 = bitcast i8* %malloccall1075 to i32*
  store i32* %tmpAddr1076, i32** %tmp1074
  %tmpWithVal1077 = load i32*, i32** %tmp1074
  store i32 4, i32* %tmpWithVal1077
  %asChar1078 = bitcast i32* %tmpWithVal1077 to i8*
  %tmp1079 = alloca i32*
  %malloccall1080 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1081 = bitcast i8* %malloccall1080 to i32*
  store i32* %tmpAddr1081, i32** %tmp1079
  %tmpWithVal1082 = load i32*, i32** %tmp1079
  store i32 5, i32* %tmpWithVal1082
  %asChar1083 = bitcast i32* %tmpWithVal1082 to i8*
  %tmp1084 = alloca i32*
  %malloccall1085 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1086 = bitcast i8* %malloccall1085 to i32*
  store i32* %tmpAddr1086, i32** %tmp1084
  %tmpWithVal1087 = load i32*, i32** %tmp1084
  store i32 6, i32* %tmpWithVal1087
  %asChar1088 = bitcast i32* %tmpWithVal1087 to i8*
  %tmp1089 = alloca i32*
  %malloccall1090 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1091 = bitcast i8* %malloccall1090 to i32*
  store i32* %tmpAddr1091, i32** %tmp1089
  %tmpWithVal1092 = load i32*, i32** %tmp1089
  store i32 7, i32* %tmpWithVal1092
  %asChar1093 = bitcast i32* %tmpWithVal1092 to i8*
  %tmp1094 = alloca i32*
  %malloccall1095 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1096 = bitcast i8* %malloccall1095 to i32*
  store i32* %tmpAddr1096, i32** %tmp1094
  %tmpWithVal1097 = load i32*, i32** %tmp1094
  store i32 8, i32* %tmpWithVal1097
  %asChar1098 = bitcast i32* %tmpWithVal1097 to i8*
  %tmp1099 = alloca i32*
  %malloccall1100 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1101 = bitcast i8* %malloccall1100 to i32*
  store i32* %tmpAddr1101, i32** %tmp1099
  %tmpWithVal1102 = load i32*, i32** %tmp1099
  store i32 9, i32* %tmpWithVal1102
  %asChar1103 = bitcast i32* %tmpWithVal1102 to i8*
  %tmp1104 = alloca i32*
  %malloccall1105 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1106 = bitcast i8* %malloccall1105 to i32*
  store i32* %tmpAddr1106, i32** %tmp1104
  %tmpWithVal1107 = load i32*, i32** %tmp1104
  store i32 10, i32* %tmpWithVal1107
  %asChar1108 = bitcast i32* %tmpWithVal1107 to i8*
  %tmp1109 = alloca i32*
  %malloccall1110 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1111 = bitcast i8* %malloccall1110 to i32*
  store i32* %tmpAddr1111, i32** %tmp1109
  %tmpWithVal1112 = load i32*, i32** %tmp1109
  store i32 11, i32* %tmpWithVal1112
  %asChar1113 = bitcast i32* %tmpWithVal1112 to i8*
  %tmp1114 = alloca i32*
  %malloccall1115 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1116 = bitcast i8* %malloccall1115 to i32*
  store i32* %tmpAddr1116, i32** %tmp1114
  %tmpWithVal1117 = load i32*, i32** %tmp1114
  store i32 12, i32* %tmpWithVal1117
  %asChar1118 = bitcast i32* %tmpWithVal1117 to i8*
  %tmp1119 = alloca i32*
  %malloccall1120 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1121 = bitcast i8* %malloccall1120 to i32*
  store i32* %tmpAddr1121, i32** %tmp1119
  %tmpWithVal1122 = load i32*, i32** %tmp1119
  store i32 13, i32* %tmpWithVal1122
  %asChar1123 = bitcast i32* %tmpWithVal1122 to i8*
  %tmp1124 = alloca i32*
  %malloccall1125 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1126 = bitcast i8* %malloccall1125 to i32*
  store i32* %tmpAddr1126, i32** %tmp1124
  %tmpWithVal1127 = load i32*, i32** %tmp1124
  store i32 14, i32* %tmpWithVal1127
  %asChar1128 = bitcast i32* %tmpWithVal1127 to i8*
  %tmp1129 = alloca i32*
```

```
    %malloccall1130 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1131 = bitcast i8* %malloccall1130 to i32*
    store i32* %tmpAddr1131, i32** %tmp1129
    %tmpWithVal1132 = load i32*, i32** %tmp1129
    store i32 15, i32* %tmpWithVal1132
    %asChar1133 = bitcast i32* %tmpWithVal1132 to i8*
    %tmp1134 = alloca i32*
    %malloccall1135 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1136 = bitcast i8* %malloccall1135 to i32*
    store i32* %tmpAddr1136, i32** %tmp1134
    %tmpWithVal1137 = load i32*, i32** %tmp1134
    store i32 16, i32* %tmpWithVal1137
    %asChar1138 = bitcast i32* %tmpWithVal1137 to i8*
    %tmp1139 = alloca i32*
    %malloccall1140 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1141 = bitcast i8* %malloccall1140 to i32*
    store i32* %tmpAddr1141, i32** %tmp1139
    %tmpWithVal1142 = load i32*, i32** %tmp1139
    store i32 17, i32* %tmpWithVal1142
    %asChar1143 = bitcast i32* %tmpWithVal1142 to i8*
    %tmp1144 = alloca i32*
    %malloccall1145 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1146 = bitcast i8* %malloccall1145 to i32*
    store i32* %tmpAddr1146, i32** %tmp1144
    %tmpWithVal1147 = load i32*, i32** %tmp1144
    store i32 18, i32* %tmpWithVal1147
    %asChar1148 = bitcast i32* %tmpWithVal1147 to i8*
    %tmp1149 = alloca i32*
    %malloccall1150 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1151 = bitcast i8* %malloccall1150 to i32*
    store i32* %tmpAddr1151, i32** %tmp1149
    %tmpWithVal1152 = load i32*, i32** %tmp1149
    store i32 19, i32* %tmpWithVal1152
    %asChar1153 = bitcast i32* %tmpWithVal1152 to i8*
    %tmp1154 = alloca i32*
    %malloccall1155 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1156 = bitcast i8* %malloccall1155 to i32*
    store i32* %tmpAddr1156, i32** %tmp1154
```

```
  %tmpWithVal1157 = load i32*, i32** %tmp1154
  store i32 20, i32* %tmpWithVal1157
  %asChar1158 = bitcast i32* %tmpWithVal1157 to i8*
  %tmp1159 = alloca i32*
  %malloccall1160 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1161 = bitcast i8* %malloccall1160 to i32*
  store i32* %tmpAddr1161, i32** %tmp1159
  %tmpWithVal1162 = load i32*, i32** %tmp1159
  store i32 21, i32* %tmpWithVal1162
  %asChar1163 = bitcast i32* %tmpWithVal1162 to i8*
  %tmp1164 = alloca i32*
  %malloccall1165 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1166 = bitcast i8* %malloccall1165 to i32*
  store i32* %tmpAddr1166, i32** %tmp1164
  %tmpWithVal1167 = load i32*, i32** %tmp1164
  store i32 22, i32* %tmpWithVal1167
  %asChar1168 = bitcast i32* %tmpWithVal1167 to i8*
  %tmp1169 = alloca i32*
  %malloccall1170 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1171 = bitcast i8* %malloccall1170 to i32*
  store i32* %tmpAddr1171, i32** %tmp1169
  %tmpWithVal1172 = load i32*, i32** %tmp1169
  store i32 23, i32* %tmpWithVal1172
  %asChar1173 = bitcast i32* %tmpWithVal1172 to i8*
  %tmp1174 = alloca i32*
  %malloccall1175 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1176 = bitcast i8* %malloccall1175 to i32*
  store i32* %tmpAddr1176, i32** %tmp1174
  %tmpWithVal1177 = load i32*, i32** %tmp1174
  store i32 24, i32* %tmpWithVal1177
  %asChar1178 = bitcast i32* %tmpWithVal1177 to i8*
  %tmp1179 = alloca i32*
  %malloccall1180 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1181 = bitcast i8* %malloccall1180 to i32*
  store i32* %tmpAddr1181, i32** %tmp1179
  %tmpWithVal1182 = load i32*, i32** %tmp1179
  store i32 25, i32* %tmpWithVal1182
  %asChar1183 = bitcast i32* %tmpWithVal1182 to i8*
```

```
%push_back1184 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1058)
%push_back1185 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1063)
%push_back1186 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1068)
%push_back1187 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1073)
%push_back1188 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1078)
%push_back1189 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1083)
%push_back1190 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1088)
%push_back1191 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1093)
%push_back1192 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1098)
%push_back1193 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1103)
%push_back1194 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1108)
%push_back1195 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1113)
%push_back1196 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1118)
%push_back1197 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1123)
%push_back1198 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1128)
%push_back1199 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1133)
%push_back1200 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1138)
%push_back1201 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1143)
%push_back1202 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1148)
%push_back1203 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1153)
%push_back1204 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1158)
%push_back1205 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1163)
```

```
%push_back1206 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1168)
%push_back1207 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1173)
%push_back1208 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1178)
%push_back1209 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1052, i8* %asChar1183)
store %list* %listAddr1052, %list** %rotor_three_in
%malloccall1210 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr1211 = bitcast i8* %malloccall1210 to %list*
%init_list1212 = call i32 (%list*, ...) @init_list(%list* %listAddr1211)
%tmp1213 = alloca i32*
%malloccall1214 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1215 = bitcast i8* %malloccall1214 to i32*
store i32* %tmpAddr1215, i32** %tmp1213
%tmpWithVal1216 = load i32*, i32** %tmp1213
store i32 0, i32* %tmpWithVal1216
%asChar1217 = bitcast i32* %tmpWithVal1216 to i8*
%tmp1218 = alloca i32*
%malloccall1219 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1220 = bitcast i8* %malloccall1219 to i32*
store i32* %tmpAddr1220, i32** %tmp1218
%tmpWithVal1221 = load i32*, i32** %tmp1218
store i32 9, i32* %tmpWithVal1221
%asChar1222 = bitcast i32* %tmpWithVal1221 to i8*
%tmp1223 = alloca i32*
%malloccall1224 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1225 = bitcast i8* %malloccall1224 to i32*
store i32* %tmpAddr1225, i32** %tmp1223
%tmpWithVal1226 = load i32*, i32** %tmp1223
store i32 3, i32* %tmpWithVal1226
%asChar1227 = bitcast i32* %tmpWithVal1226 to i8*
%tmp1228 = alloca i32*
%malloccall1229 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1230 = bitcast i8* %malloccall1229 to i32*
store i32* %tmpAddr1230, i32** %tmp1228
%tmpWithVal1231 = load i32*, i32** %tmp1228
store i32 10, i32* %tmpWithVal1231
```

```
  %asChar1232 = bitcast i32* %tmpWithVal1231 to i8*
  %tmp1233 = alloca i32*
  %malloccall1234 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1235 = bitcast i8* %malloccall1234 to i32*
  store i32* %tmpAddr1235, i32** %tmp1233
  %tmpWithVal1236 = load i32*, i32** %tmp1233
  store i32 18, i32* %tmpWithVal1236
  %asChar1237 = bitcast i32* %tmpWithVal1236 to i8*
  %tmp1238 = alloca i32*
  %malloccall1239 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1240 = bitcast i8* %malloccall1239 to i32*
  store i32* %tmpAddr1240, i32** %tmp1238
  %tmpWithVal1241 = load i32*, i32** %tmp1238
  store i32 8, i32* %tmpWithVal1241
  %asChar1242 = bitcast i32* %tmpWithVal1241 to i8*
  %tmp1243 = alloca i32*
  %malloccall1244 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1245 = bitcast i8* %malloccall1244 to i32*
  store i32* %tmpAddr1245, i32** %tmp1243
  %tmpWithVal1246 = load i32*, i32** %tmp1243
  store i32 17, i32* %tmpWithVal1246
  %asChar1247 = bitcast i32* %tmpWithVal1246 to i8*
  %tmp1248 = alloca i32*
  %malloccall1249 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1250 = bitcast i8* %malloccall1249 to i32*
  store i32* %tmpAddr1250, i32** %tmp1248
  %tmpWithVal1251 = load i32*, i32** %tmp1248
  store i32 20, i32* %tmpWithVal1251
  %asChar1252 = bitcast i32* %tmpWithVal1251 to i8*
  %tmp1253 = alloca i32*
  %malloccall1254 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1255 = bitcast i8* %malloccall1254 to i32*
  store i32* %tmpAddr1255, i32** %tmp1253
  %tmpWithVal1256 = load i32*, i32** %tmp1253
  store i32 23, i32* %tmpWithVal1256
  %asChar1257 = bitcast i32* %tmpWithVal1256 to i8*
  %tmp1258 = alloca i32*
  %malloccall1259 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
    %tmpAddr1260 = bitcast i8* %malloccall1259 to i32*
    store i32* %tmpAddr1260, i32** %tmp1258
    %tmpWithVal1261 = load i32*, i32** %tmp1258
    store i32 1, i32* %tmpWithVal1261
    %asChar1262 = bitcast i32* %tmpWithVal1261 to i8*
    %tmp1263 = alloca i32*
    %malloccall1264 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1265 = bitcast i8* %malloccall1264 to i32*
    store i32* %tmpAddr1265, i32** %tmp1263
    %tmpWithVal1266 = load i32*, i32** %tmp1263
    store i32 11, i32* %tmpWithVal1266
    %asChar1267 = bitcast i32* %tmpWithVal1266 to i8*
    %tmp1268 = alloca i32*
    %malloccall1269 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1270 = bitcast i8* %malloccall1269 to i32*
    store i32* %tmpAddr1270, i32** %tmp1268
    %tmpWithVal1271 = load i32*, i32** %tmp1268
    store i32 7, i32* %tmpWithVal1271
    %asChar1272 = bitcast i32* %tmpWithVal1271 to i8*
    %tmp1273 = alloca i32*
    %malloccall1274 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1275 = bitcast i8* %malloccall1274 to i32*
    store i32* %tmpAddr1275, i32** %tmp1273
    %tmpWithVal1276 = load i32*, i32** %tmp1273
    store i32 22, i32* %tmpWithVal1276
    %asChar1277 = bitcast i32* %tmpWithVal1276 to i8*
    %tmp1278 = alloca i32*
    %malloccall1279 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1280 = bitcast i8* %malloccall1279 to i32*
    store i32* %tmpAddr1280, i32** %tmp1278
    %tmpWithVal1281 = load i32*, i32** %tmp1278
    store i32 19, i32* %tmpWithVal1281
    %asChar1282 = bitcast i32* %tmpWithVal1281 to i8*
    %tmp1283 = alloca i32*
    %malloccall1284 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1285 = bitcast i8* %malloccall1284 to i32*
    store i32* %tmpAddr1285, i32** %tmp1283
    %tmpWithVal1286 = load i32*, i32** %tmp1283
    store i32 12, i32* %tmpWithVal1286
```

```
%asChar1287 = bitcast i32* %tmpWithVal1286 to i8*
%tmp1288 = alloca i32*
%malloccall1289 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1290 = bitcast i8* %malloccall1289 to i32*
store i32* %tmpAddr1290, i32** %tmp1288
%tmpWithVal1291 = load i32*, i32** %tmp1288
store i32 2, i32* %tmpWithVal1291
%asChar1292 = bitcast i32* %tmpWithVal1291 to i8*
%tmp1293 = alloca i32*
%malloccall1294 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1295 = bitcast i8* %malloccall1294 to i32*
store i32* %tmpAddr1295, i32** %tmp1293
%tmpWithVal1296 = load i32*, i32** %tmp1293
store i32 16, i32* %tmpWithVal1296
%asChar1297 = bitcast i32* %tmpWithVal1296 to i8*
%tmp1298 = alloca i32*
%malloccall1299 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1300 = bitcast i8* %malloccall1299 to i32*
store i32* %tmpAddr1300, i32** %tmp1298
%tmpWithVal1301 = load i32*, i32** %tmp1298
store i32 6, i32* %tmpWithVal1301
%asChar1302 = bitcast i32* %tmpWithVal1301 to i8*
%tmp1303 = alloca i32*
%malloccall1304 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1305 = bitcast i8* %malloccall1304 to i32*
store i32* %tmpAddr1305, i32** %tmp1303
%tmpWithVal1306 = load i32*, i32** %tmp1303
store i32 25, i32* %tmpWithVal1306
%asChar1307 = bitcast i32* %tmpWithVal1306 to i8*
%tmp1308 = alloca i32*
%malloccall1309 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1310 = bitcast i8* %malloccall1309 to i32*
store i32* %tmpAddr1310, i32** %tmp1308
%tmpWithVal1311 = load i32*, i32** %tmp1308
store i32 13, i32* %tmpWithVal1311
%asChar1312 = bitcast i32* %tmpWithVal1311 to i8*
%tmp1313 = alloca i32*
%malloccall1314 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
    %tmpAddr1315 = bitcast i8* %malloccall1314 to i32*
    store i32* %tmpAddr1315, i32** %tmp1313
    %tmpWithVal1316 = load i32*, i32** %tmp1313
    store i32 15, i32* %tmpWithVal1316
    %asChar1317 = bitcast i32* %tmpWithVal1316 to i8*
    %tmp1318 = alloca i32*
    %malloccall1319 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1320 = bitcast i8* %malloccall1319 to i32*
    store i32* %tmpAddr1320, i32** %tmp1318
    %tmpWithVal1321 = load i32*, i32** %tmp1318
    store i32 24, i32* %tmpWithVal1321
    %asChar1322 = bitcast i32* %tmpWithVal1321 to i8*
    %tmp1323 = alloca i32*
    %malloccall1324 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1325 = bitcast i8* %malloccall1324 to i32*
    store i32* %tmpAddr1325, i32** %tmp1323
    %tmpWithVal1326 = load i32*, i32** %tmp1323
    store i32 5, i32* %tmpWithVal1326
    %asChar1327 = bitcast i32* %tmpWithVal1326 to i8*
    %tmp1328 = alloca i32*
    %malloccall1329 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1330 = bitcast i8* %malloccall1329 to i32*
    store i32* %tmpAddr1330, i32** %tmp1328
    %tmpWithVal1331 = load i32*, i32** %tmp1328
    store i32 21, i32* %tmpWithVal1331
    %asChar1332 = bitcast i32* %tmpWithVal1331 to i8*
    %tmp1333 = alloca i32*
    %malloccall1334 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1335 = bitcast i8* %malloccall1334 to i32*
    store i32* %tmpAddr1335, i32** %tmp1333
    %tmpWithVal1336 = load i32*, i32** %tmp1333
    store i32 14, i32* %tmpWithVal1336
    %asChar1337 = bitcast i32* %tmpWithVal1336 to i8*
    %tmp1338 = alloca i32*
    %malloccall1339 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1340 = bitcast i8* %malloccall1339 to i32*
    store i32* %tmpAddr1340, i32** %tmp1338
    %tmpWithVal1341 = load i32*, i32** %tmp1338
    store i32 4, i32* %tmpWithVal1341
```

```
%asChar1342 = bitcast i32* %tmpWithVal1341 to i8*
%push_back1343 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1217)
%push_back1344 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1222)
%push_back1345 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1227)
%push_back1346 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1232)
%push_back1347 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1237)
%push_back1348 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1242)
%push_back1349 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1247)
%push_back1350 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1252)
%push_back1351 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1257)
%push_back1352 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1262)
%push_back1353 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1267)
%push_back1354 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1272)
%push_back1355 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1277)
%push_back1356 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1282)
%push_back1357 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1287)
%push_back1358 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1292)
%push_back1359 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1297)
%push_back1360 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1302)
%push_back1361 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1307)
%push_back1362 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1312)
%push_back1363 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1317)
```

```
    %push_back1364 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1322)
    %push_back1365 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1327)
    %push_back1366 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1332)
    %push_back1367 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1337)
    %push_back1368 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1211, i8* %asChar1342)
    store %list* %listAddr1211, %list** %rotor_two_out
    %malloccall1369 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
    %listAddr1370 = bitcast i8* %malloccall1369 to %list*
    %init_list1371 = call i32 (%list*, ...) @init_list(%list* %listAddr1370)
    %tmp1372 = alloca i32*
    %malloccall1373 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1374 = bitcast i8* %malloccall1373 to i32*
    store i32* %tmpAddr1374, i32** %tmp1372
    %tmpWithVal1375 = load i32*, i32** %tmp1372
    store i32 0, i32* %tmpWithVal1375
    %asChar1376 = bitcast i32* %tmpWithVal1375 to i8*
    %tmp1377 = alloca i32*
    %malloccall1378 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1379 = bitcast i8* %malloccall1378 to i32*
    store i32* %tmpAddr1379, i32** %tmp1377
    %tmpWithVal1380 = load i32*, i32** %tmp1377
    store i32 1, i32* %tmpWithVal1380
    %asChar1381 = bitcast i32* %tmpWithVal1380 to i8*
    %tmp1382 = alloca i32*
    %malloccall1383 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1384 = bitcast i8* %malloccall1383 to i32*
    store i32* %tmpAddr1384, i32** %tmp1382
    %tmpWithVal1385 = load i32*, i32** %tmp1382
    store i32 2, i32* %tmpWithVal1385
    %asChar1386 = bitcast i32* %tmpWithVal1385 to i8*
    %tmp1387 = alloca i32*
    %malloccall1388 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr1389 = bitcast i8* %malloccall1388 to i32*
    store i32* %tmpAddr1389, i32** %tmp1387
```

```
  %tmpWithVal1390 = load i32*, i32** %tmp1387
  store i32 3, i32* %tmpWithVal1390
  %asChar1391 = bitcast i32* %tmpWithVal1390 to i8*
  %tmp1392 = alloca i32*
  %malloccall1393 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1394 = bitcast i8* %malloccall1393 to i32*
  store i32* %tmpAddr1394, i32** %tmp1392
  %tmpWithVal1395 = load i32*, i32** %tmp1392
  store i32 4, i32* %tmpWithVal1395
  %asChar1396 = bitcast i32* %tmpWithVal1395 to i8*
  %tmp1397 = alloca i32*
  %malloccall1398 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1399 = bitcast i8* %malloccall1398 to i32*
  store i32* %tmpAddr1399, i32** %tmp1397
  %tmpWithVal1400 = load i32*, i32** %tmp1397
  store i32 5, i32* %tmpWithVal1400
  %asChar1401 = bitcast i32* %tmpWithVal1400 to i8*
  %tmp1402 = alloca i32*
  %malloccall1403 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1404 = bitcast i8* %malloccall1403 to i32*
  store i32* %tmpAddr1404, i32** %tmp1402
  %tmpWithVal1405 = load i32*, i32** %tmp1402
  store i32 6, i32* %tmpWithVal1405
  %asChar1406 = bitcast i32* %tmpWithVal1405 to i8*
  %tmp1407 = alloca i32*
  %malloccall1408 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1409 = bitcast i8* %malloccall1408 to i32*
  store i32* %tmpAddr1409, i32** %tmp1407
  %tmpWithVal1410 = load i32*, i32** %tmp1407
  store i32 7, i32* %tmpWithVal1410
  %asChar1411 = bitcast i32* %tmpWithVal1410 to i8*
  %tmp1412 = alloca i32*
  %malloccall1413 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1414 = bitcast i8* %malloccall1413 to i32*
  store i32* %tmpAddr1414, i32** %tmp1412
  %tmpWithVal1415 = load i32*, i32** %tmp1412
  store i32 8, i32* %tmpWithVal1415
  %asChar1416 = bitcast i32* %tmpWithVal1415 to i8*
  %tmp1417 = alloca i32*
```

```
%malloccall1418 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1419 = bitcast i8* %malloccall1418 to i32*
store i32* %tmpAddr1419, i32** %tmp1417
%tmpWithVal1420 = load i32*, i32** %tmp1417
store i32 9, i32* %tmpWithVal1420
%asChar1421 = bitcast i32* %tmpWithVal1420 to i8*
%tmp1422 = alloca i32*
%malloccall1423 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1424 = bitcast i8* %malloccall1423 to i32*
store i32* %tmpAddr1424, i32** %tmp1422
%tmpWithVal1425 = load i32*, i32** %tmp1422
store i32 10, i32* %tmpWithVal1425
%asChar1426 = bitcast i32* %tmpWithVal1425 to i8*
%tmp1427 = alloca i32*
%malloccall1428 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1429 = bitcast i8* %malloccall1428 to i32*
store i32* %tmpAddr1429, i32** %tmp1427
%tmpWithVal1430 = load i32*, i32** %tmp1427
store i32 11, i32* %tmpWithVal1430
%asChar1431 = bitcast i32* %tmpWithVal1430 to i8*
%tmp1432 = alloca i32*
%malloccall1433 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1434 = bitcast i8* %malloccall1433 to i32*
store i32* %tmpAddr1434, i32** %tmp1432
%tmpWithVal1435 = load i32*, i32** %tmp1432
store i32 12, i32* %tmpWithVal1435
%asChar1436 = bitcast i32* %tmpWithVal1435 to i8*
%tmp1437 = alloca i32*
%malloccall1438 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1439 = bitcast i8* %malloccall1438 to i32*
store i32* %tmpAddr1439, i32** %tmp1437
%tmpWithVal1440 = load i32*, i32** %tmp1437
store i32 13, i32* %tmpWithVal1440
%asChar1441 = bitcast i32* %tmpWithVal1440 to i8*
%tmp1442 = alloca i32*
%malloccall1443 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1444 = bitcast i8* %malloccall1443 to i32*
store i32* %tmpAddr1444, i32** %tmp1442
```

```
%tmpWithVal1445 = load i32*, i32** %tmp1442
store i32 14, i32* %tmpWithVal1445
%asChar1446 = bitcast i32* %tmpWithVal1445 to i8*
%tmp1447 = alloca i32*
%malloccall1448 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1449 = bitcast i8* %malloccall1448 to i32*
store i32* %tmpAddr1449, i32** %tmp1447
%tmpWithVal1450 = load i32*, i32** %tmp1447
store i32 15, i32* %tmpWithVal1450
%asChar1451 = bitcast i32* %tmpWithVal1450 to i8*
%tmp1452 = alloca i32*
%malloccall1453 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1454 = bitcast i8* %malloccall1453 to i32*
store i32* %tmpAddr1454, i32** %tmp1452
%tmpWithVal1455 = load i32*, i32** %tmp1452
store i32 16, i32* %tmpWithVal1455
%asChar1456 = bitcast i32* %tmpWithVal1455 to i8*
%tmp1457 = alloca i32*
%malloccall1458 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1459 = bitcast i8* %malloccall1458 to i32*
store i32* %tmpAddr1459, i32** %tmp1457
%tmpWithVal1460 = load i32*, i32** %tmp1457
store i32 17, i32* %tmpWithVal1460
%asChar1461 = bitcast i32* %tmpWithVal1460 to i8*
%tmp1462 = alloca i32*
%malloccall1463 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1464 = bitcast i8* %malloccall1463 to i32*
store i32* %tmpAddr1464, i32** %tmp1462
%tmpWithVal1465 = load i32*, i32** %tmp1462
store i32 18, i32* %tmpWithVal1465
%asChar1466 = bitcast i32* %tmpWithVal1465 to i8*
%tmp1467 = alloca i32*
%malloccall1468 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1469 = bitcast i8* %malloccall1468 to i32*
store i32* %tmpAddr1469, i32** %tmp1467
%tmpWithVal1470 = load i32*, i32** %tmp1467
store i32 19, i32* %tmpWithVal1470
%asChar1471 = bitcast i32* %tmpWithVal1470 to i8*
%tmp1472 = alloca i32*
```

```
%malloccall1473 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1474 = bitcast i8* %malloccall1473 to i32*
store i32* %tmpAddr1474, i32** %tmp1472
%tmpWithVal1475 = load i32*, i32** %tmp1472
store i32 20, i32* %tmpWithVal1475
%asChar1476 = bitcast i32* %tmpWithVal1475 to i8*
%tmp1477 = alloca i32*
%malloccall1478 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1479 = bitcast i8* %malloccall1478 to i32*
store i32* %tmpAddr1479, i32** %tmp1477
%tmpWithVal1480 = load i32*, i32** %tmp1477
store i32 21, i32* %tmpWithVal1480
%asChar1481 = bitcast i32* %tmpWithVal1480 to i8*
%tmp1482 = alloca i32*
%malloccall1483 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1484 = bitcast i8* %malloccall1483 to i32*
store i32* %tmpAddr1484, i32** %tmp1482
%tmpWithVal1485 = load i32*, i32** %tmp1482
store i32 22, i32* %tmpWithVal1485
%asChar1486 = bitcast i32* %tmpWithVal1485 to i8*
%tmp1487 = alloca i32*
%malloccall1488 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1489 = bitcast i8* %malloccall1488 to i32*
store i32* %tmpAddr1489, i32** %tmp1487
%tmpWithVal1490 = load i32*, i32** %tmp1487
store i32 23, i32* %tmpWithVal1490
%asChar1491 = bitcast i32* %tmpWithVal1490 to i8*
%tmp1492 = alloca i32*
%malloccall1493 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1494 = bitcast i8* %malloccall1493 to i32*
store i32* %tmpAddr1494, i32** %tmp1492
%tmpWithVal1495 = load i32*, i32** %tmp1492
store i32 24, i32* %tmpWithVal1495
%asChar1496 = bitcast i32* %tmpWithVal1495 to i8*
%tmp1497 = alloca i32*
%malloccall1498 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1499 = bitcast i8* %malloccall1498 to i32*
store i32* %tmpAddr1499, i32** %tmp1497
```

```
%tmpWithVal1500 = load i32*, i32** %tmp1497
store i32 25, i32* %tmpWithVal1500
%asChar1501 = bitcast i32* %tmpWithVal1500 to i8*
%push_back1502 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1376)
%push_back1503 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1381)
%push_back1504 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1386)
%push_back1505 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1391)
%push_back1506 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1396)
%push_back1507 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1401)
%push_back1508 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1406)
%push_back1509 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1411)
%push_back1510 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1416)
%push_back1511 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1421)
%push_back1512 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1426)
%push_back1513 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1431)
%push_back1514 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1436)
%push_back1515 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1441)
%push_back1516 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1446)
%push_back1517 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1451)
%push_back1518 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1456)
%push_back1519 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1461)
%push_back1520 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1466)
%push_back1521 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1471)
```

```
  %push_back1522 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1476)
  %push_back1523 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1481)
  %push_back1524 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1486)
  %push_back1525 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1491)
  %push_back1526 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1496)
  %push_back1527 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1370, i8* %asChar1501)
  store %list* %listAddr1370, %list** %rotor_two_in
  %malloccall1528 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr1529 = bitcast i8* %malloccall1528 to %list*
  %init_list1530 = call i32 (%list*, ...) @init_list(%list* %listAddr1529)
  %tmp1531 = alloca i32*
  %malloccall1532 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1533 = bitcast i8* %malloccall1532 to i32*
  store i32* %tmpAddr1533, i32** %tmp1531
  %tmpWithVal1534 = load i32*, i32** %tmp1531
  store i32 4, i32* %tmpWithVal1534
  %asChar1535 = bitcast i32* %tmpWithVal1534 to i8*
  %tmp1536 = alloca i32*
  %malloccall1537 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1538 = bitcast i8* %malloccall1537 to i32*
  store i32* %tmpAddr1538, i32** %tmp1536
  %tmpWithVal1539 = load i32*, i32** %tmp1536
  store i32 10, i32* %tmpWithVal1539
  %asChar1540 = bitcast i32* %tmpWithVal1539 to i8*
  %tmp1541 = alloca i32*
  %malloccall1542 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1543 = bitcast i8* %malloccall1542 to i32*
  store i32* %tmpAddr1543, i32** %tmp1541
  %tmpWithVal1544 = load i32*, i32** %tmp1541
  store i32 12, i32* %tmpWithVal1544
  %asChar1545 = bitcast i32* %tmpWithVal1544 to i8*
  %tmp1546 = alloca i32*
  %malloccall1547 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
	%tmpAddr1548 = bitcast i8* %malloccall1547 to i32*
	store i32* %tmpAddr1548, i32** %tmp1546
	%tmpWithVal1549 = load i32*, i32** %tmp1546
	store i32 5, i32* %tmpWithVal1549
	%asChar1550 = bitcast i32* %tmpWithVal1549 to i8*
	%tmp1551 = alloca i32*
	%malloccall1552 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr1553 = bitcast i8* %malloccall1552 to i32*
	store i32* %tmpAddr1553, i32** %tmp1551
	%tmpWithVal1554 = load i32*, i32** %tmp1551
	store i32 11, i32* %tmpWithVal1554
	%asChar1555 = bitcast i32* %tmpWithVal1554 to i8*
	%tmp1556 = alloca i32*
	%malloccall1557 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr1558 = bitcast i8* %malloccall1557 to i32*
	store i32* %tmpAddr1558, i32** %tmp1556
	%tmpWithVal1559 = load i32*, i32** %tmp1556
	store i32 6, i32* %tmpWithVal1559
	%asChar1560 = bitcast i32* %tmpWithVal1559 to i8*
	%tmp1561 = alloca i32*
	%malloccall1562 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr1563 = bitcast i8* %malloccall1562 to i32*
	store i32* %tmpAddr1563, i32** %tmp1561
	%tmpWithVal1564 = load i32*, i32** %tmp1561
	store i32 3, i32* %tmpWithVal1564
	%asChar1565 = bitcast i32* %tmpWithVal1564 to i8*
	%tmp1566 = alloca i32*
	%malloccall1567 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr1568 = bitcast i8* %malloccall1567 to i32*
	store i32* %tmpAddr1568, i32** %tmp1566
	%tmpWithVal1569 = load i32*, i32** %tmp1566
	store i32 16, i32* %tmpWithVal1569
	%asChar1570 = bitcast i32* %tmpWithVal1569 to i8*
	%tmp1571 = alloca i32*
	%malloccall1572 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr1573 = bitcast i8* %malloccall1572 to i32*
	store i32* %tmpAddr1573, i32** %tmp1571
	%tmpWithVal1574 = load i32*, i32** %tmp1571
	store i32 21, i32* %tmpWithVal1574
```

```
%asChar1575 = bitcast i32* %tmpWithVal1574 to i8*
%tmp1576 = alloca i32*
%malloccall1577 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1578 = bitcast i8* %malloccall1577 to i32*
store i32* %tmpAddr1578, i32** %tmp1576
%tmpWithVal1579 = load i32*, i32** %tmp1576
store i32 25, i32* %tmpWithVal1579
%asChar1580 = bitcast i32* %tmpWithVal1579 to i8*
%tmp1581 = alloca i32*
%malloccall1582 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1583 = bitcast i8* %malloccall1582 to i32*
store i32* %tmpAddr1583, i32** %tmp1581
%tmpWithVal1584 = load i32*, i32** %tmp1581
store i32 13, i32* %tmpWithVal1584
%asChar1585 = bitcast i32* %tmpWithVal1584 to i8*
%tmp1586 = alloca i32*
%malloccall1587 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1588 = bitcast i8* %malloccall1587 to i32*
store i32* %tmpAddr1588, i32** %tmp1586
%tmpWithVal1589 = load i32*, i32** %tmp1586
store i32 19, i32* %tmpWithVal1589
%asChar1590 = bitcast i32* %tmpWithVal1589 to i8*
%tmp1591 = alloca i32*
%malloccall1592 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1593 = bitcast i8* %malloccall1592 to i32*
store i32* %tmpAddr1593, i32** %tmp1591
%tmpWithVal1594 = load i32*, i32** %tmp1591
store i32 14, i32* %tmpWithVal1594
%asChar1595 = bitcast i32* %tmpWithVal1594 to i8*
%tmp1596 = alloca i32*
%malloccall1597 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1598 = bitcast i8* %malloccall1597 to i32*
store i32* %tmpAddr1598, i32** %tmp1596
%tmpWithVal1599 = load i32*, i32** %tmp1596
store i32 22, i32* %tmpWithVal1599
%asChar1600 = bitcast i32* %tmpWithVal1599 to i8*
%tmp1601 = alloca i32*
%malloccall1602 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
%tmpAddr1603 = bitcast i8* %malloccall1602 to i32*
store i32* %tmpAddr1603, i32** %tmp1601
%tmpWithVal1604 = load i32*, i32** %tmp1601
store i32 24, i32* %tmpWithVal1604
%asChar1605 = bitcast i32* %tmpWithVal1604 to i8*
%tmp1606 = alloca i32*
%malloccall1607 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1608 = bitcast i8* %malloccall1607 to i32*
store i32* %tmpAddr1608, i32** %tmp1606
%tmpWithVal1609 = load i32*, i32** %tmp1606
store i32 7, i32* %tmpWithVal1609
%asChar1610 = bitcast i32* %tmpWithVal1609 to i8*
%tmp1611 = alloca i32*
%malloccall1612 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1613 = bitcast i8* %malloccall1612 to i32*
store i32* %tmpAddr1613, i32** %tmp1611
%tmpWithVal1614 = load i32*, i32** %tmp1611
store i32 23, i32* %tmpWithVal1614
%asChar1615 = bitcast i32* %tmpWithVal1614 to i8*
%tmp1616 = alloca i32*
%malloccall1617 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1618 = bitcast i8* %malloccall1617 to i32*
store i32* %tmpAddr1618, i32** %tmp1616
%tmpWithVal1619 = load i32*, i32** %tmp1616
store i32 20, i32* %tmpWithVal1619
%asChar1620 = bitcast i32* %tmpWithVal1619 to i8*
%tmp1621 = alloca i32*
%malloccall1622 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1623 = bitcast i8* %malloccall1622 to i32*
store i32* %tmpAddr1623, i32** %tmp1621
%tmpWithVal1624 = load i32*, i32** %tmp1621
store i32 18, i32* %tmpWithVal1624
%asChar1625 = bitcast i32* %tmpWithVal1624 to i8*
%tmp1626 = alloca i32*
%malloccall1627 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1628 = bitcast i8* %malloccall1627 to i32*
store i32* %tmpAddr1628, i32** %tmp1626
%tmpWithVal1629 = load i32*, i32** %tmp1626
store i32 15, i32* %tmpWithVal1629
```

```
%asChar1630 = bitcast i32* %tmpWithVal1629 to i8*
%tmp1631 = alloca i32*
%malloccall1632 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1633 = bitcast i8* %malloccall1632 to i32*
store i32* %tmpAddr1633, i32** %tmp1631
%tmpWithVal1634 = load i32*, i32** %tmp1631
store i32 0, i32* %tmpWithVal1634
%asChar1635 = bitcast i32* %tmpWithVal1634 to i8*
%tmp1636 = alloca i32*
%malloccall1637 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1638 = bitcast i8* %malloccall1637 to i32*
store i32* %tmpAddr1638, i32** %tmp1636
%tmpWithVal1639 = load i32*, i32** %tmp1636
store i32 8, i32* %tmpWithVal1639
%asChar1640 = bitcast i32* %tmpWithVal1639 to i8*
%tmp1641 = alloca i32*
%malloccall1642 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1643 = bitcast i8* %malloccall1642 to i32*
store i32* %tmpAddr1643, i32** %tmp1641
%tmpWithVal1644 = load i32*, i32** %tmp1641
store i32 1, i32* %tmpWithVal1644
%asChar1645 = bitcast i32* %tmpWithVal1644 to i8*
%tmp1646 = alloca i32*
%malloccall1647 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1648 = bitcast i8* %malloccall1647 to i32*
store i32* %tmpAddr1648, i32** %tmp1646
%tmpWithVal1649 = load i32*, i32** %tmp1646
store i32 17, i32* %tmpWithVal1649
%asChar1650 = bitcast i32* %tmpWithVal1649 to i8*
%tmp1651 = alloca i32*
%malloccall1652 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1653 = bitcast i8* %malloccall1652 to i32*
store i32* %tmpAddr1653, i32** %tmp1651
%tmpWithVal1654 = load i32*, i32** %tmp1651
store i32 2, i32* %tmpWithVal1654
%asChar1655 = bitcast i32* %tmpWithVal1654 to i8*
%tmp1656 = alloca i32*
%malloccall1657 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
    %tmpAddr1658 = bitcast i8* %malloccall1657 to i32*
    store i32* %tmpAddr1658, i32** %tmp1656
    %tmpWithVal1659 = load i32*, i32** %tmp1656
    store i32 9, i32* %tmpWithVal1659
    %asChar1660 = bitcast i32* %tmpWithVal1659 to i8*
    %push_back1661 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1535)
    %push_back1662 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1540)
    %push_back1663 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1545)
    %push_back1664 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1550)
    %push_back1665 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1555)
    %push_back1666 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1560)
    %push_back1667 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1565)
    %push_back1668 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1570)
    %push_back1669 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1575)
    %push_back1670 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1580)
    %push_back1671 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1585)
    %push_back1672 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1590)
    %push_back1673 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1595)
    %push_back1674 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1600)
    %push_back1675 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1605)
    %push_back1676 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1610)
    %push_back1677 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1615)
    %push_back1678 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1620)
    %push_back1679 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1625)
```

```
  %push_back1680 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1630)
  %push_back1681 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1635)
  %push_back1682 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1640)
  %push_back1683 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1645)
  %push_back1684 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1650)
  %push_back1685 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1655)
  %push_back1686 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1529, i8* %asChar1660)
  store %list* %listAddr1529, %list** %rotor_one_out
  %malloccall1687 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr1688 = bitcast i8* %malloccall1687 to %list*
  %init_list1689 = call i32 (%list*, ...) @init_list(%list* %listAddr1688)
  %tmp1690 = alloca i32*
  %malloccall1691 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1692 = bitcast i8* %malloccall1691 to i32*
  store i32* %tmpAddr1692, i32** %tmp1690
  %tmpWithVal1693 = load i32*, i32** %tmp1690
  store i32 0, i32* %tmpWithVal1693
  %asChar1694 = bitcast i32* %tmpWithVal1693 to i8*
  %tmp1695 = alloca i32*
  %malloccall1696 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1697 = bitcast i8* %malloccall1696 to i32*
  store i32* %tmpAddr1697, i32** %tmp1695
  %tmpWithVal1698 = load i32*, i32** %tmp1695
  store i32 1, i32* %tmpWithVal1698
  %asChar1699 = bitcast i32* %tmpWithVal1698 to i8*
  %tmp1700 = alloca i32*
  %malloccall1701 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1702 = bitcast i8* %malloccall1701 to i32*
  store i32* %tmpAddr1702, i32** %tmp1700
  %tmpWithVal1703 = load i32*, i32** %tmp1700
  store i32 2, i32* %tmpWithVal1703
  %asChar1704 = bitcast i32* %tmpWithVal1703 to i8*
  %tmp1705 = alloca i32*
```

```
%malloccall1706 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1707 = bitcast i8* %malloccall1706 to i32*
store i32* %tmpAddr1707, i32** %tmp1705
%tmpWithVal1708 = load i32*, i32** %tmp1705
store i32 3, i32* %tmpWithVal1708
%asChar1709 = bitcast i32* %tmpWithVal1708 to i8*
%tmp1710 = alloca i32*
%malloccall1711 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1712 = bitcast i8* %malloccall1711 to i32*
store i32* %tmpAddr1712, i32** %tmp1710
%tmpWithVal1713 = load i32*, i32** %tmp1710
store i32 4, i32* %tmpWithVal1713
%asChar1714 = bitcast i32* %tmpWithVal1713 to i8*
%tmp1715 = alloca i32*
%malloccall1716 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1717 = bitcast i8* %malloccall1716 to i32*
store i32* %tmpAddr1717, i32** %tmp1715
%tmpWithVal1718 = load i32*, i32** %tmp1715
store i32 5, i32* %tmpWithVal1718
%asChar1719 = bitcast i32* %tmpWithVal1718 to i8*
%tmp1720 = alloca i32*
%malloccall1721 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1722 = bitcast i8* %malloccall1721 to i32*
store i32* %tmpAddr1722, i32** %tmp1720
%tmpWithVal1723 = load i32*, i32** %tmp1720
store i32 6, i32* %tmpWithVal1723
%asChar1724 = bitcast i32* %tmpWithVal1723 to i8*
%tmp1725 = alloca i32*
%malloccall1726 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1727 = bitcast i8* %malloccall1726 to i32*
store i32* %tmpAddr1727, i32** %tmp1725
%tmpWithVal1728 = load i32*, i32** %tmp1725
store i32 7, i32* %tmpWithVal1728
%asChar1729 = bitcast i32* %tmpWithVal1728 to i8*
%tmp1730 = alloca i32*
%malloccall1731 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1732 = bitcast i8* %malloccall1731 to i32*
store i32* %tmpAddr1732, i32** %tmp1730
```

```
%tmpWithVal1733 = load i32*, i32** %tmp1730
store i32 8, i32* %tmpWithVal1733
%asChar1734 = bitcast i32* %tmpWithVal1733 to i8*
%tmp1735 = alloca i32*
%malloccall1736 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1737 = bitcast i8* %malloccall1736 to i32*
store i32* %tmpAddr1737, i32** %tmp1735
%tmpWithVal1738 = load i32*, i32** %tmp1735
store i32 9, i32* %tmpWithVal1738
%asChar1739 = bitcast i32* %tmpWithVal1738 to i8*
%tmp1740 = alloca i32*
%malloccall1741 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1742 = bitcast i8* %malloccall1741 to i32*
store i32* %tmpAddr1742, i32** %tmp1740
%tmpWithVal1743 = load i32*, i32** %tmp1740
store i32 10, i32* %tmpWithVal1743
%asChar1744 = bitcast i32* %tmpWithVal1743 to i8*
%tmp1745 = alloca i32*
%malloccall1746 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1747 = bitcast i8* %malloccall1746 to i32*
store i32* %tmpAddr1747, i32** %tmp1745
%tmpWithVal1748 = load i32*, i32** %tmp1745
store i32 11, i32* %tmpWithVal1748
%asChar1749 = bitcast i32* %tmpWithVal1748 to i8*
%tmp1750 = alloca i32*
%malloccall1751 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1752 = bitcast i8* %malloccall1751 to i32*
store i32* %tmpAddr1752, i32** %tmp1750
%tmpWithVal1753 = load i32*, i32** %tmp1750
store i32 12, i32* %tmpWithVal1753
%asChar1754 = bitcast i32* %tmpWithVal1753 to i8*
%tmp1755 = alloca i32*
%malloccall1756 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1757 = bitcast i8* %malloccall1756 to i32*
store i32* %tmpAddr1757, i32** %tmp1755
%tmpWithVal1758 = load i32*, i32** %tmp1755
store i32 13, i32* %tmpWithVal1758
%asChar1759 = bitcast i32* %tmpWithVal1758 to i8*
%tmp1760 = alloca i32*
```

```
  %malloccall1761 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1762 = bitcast i8* %malloccall1761 to i32*
  store i32* %tmpAddr1762, i32** %tmp1760
  %tmpWithVal1763 = load i32*, i32** %tmp1760
  store i32 14, i32* %tmpWithVal1763
  %asChar1764 = bitcast i32* %tmpWithVal1763 to i8*
  %tmp1765 = alloca i32*
  %malloccall1766 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1767 = bitcast i8* %malloccall1766 to i32*
  store i32* %tmpAddr1767, i32** %tmp1765
  %tmpWithVal1768 = load i32*, i32** %tmp1765
  store i32 15, i32* %tmpWithVal1768
  %asChar1769 = bitcast i32* %tmpWithVal1768 to i8*
  %tmp1770 = alloca i32*
  %malloccall1771 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1772 = bitcast i8* %malloccall1771 to i32*
  store i32* %tmpAddr1772, i32** %tmp1770
  %tmpWithVal1773 = load i32*, i32** %tmp1770
  store i32 16, i32* %tmpWithVal1773
  %asChar1774 = bitcast i32* %tmpWithVal1773 to i8*
  %tmp1775 = alloca i32*
  %malloccall1776 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1777 = bitcast i8* %malloccall1776 to i32*
  store i32* %tmpAddr1777, i32** %tmp1775
  %tmpWithVal1778 = load i32*, i32** %tmp1775
  store i32 17, i32* %tmpWithVal1778
  %asChar1779 = bitcast i32* %tmpWithVal1778 to i8*
  %tmp1780 = alloca i32*
  %malloccall1781 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1782 = bitcast i8* %malloccall1781 to i32*
  store i32* %tmpAddr1782, i32** %tmp1780
  %tmpWithVal1783 = load i32*, i32** %tmp1780
  store i32 18, i32* %tmpWithVal1783
  %asChar1784 = bitcast i32* %tmpWithVal1783 to i8*
  %tmp1785 = alloca i32*
  %malloccall1786 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1787 = bitcast i8* %malloccall1786 to i32*
  store i32* %tmpAddr1787, i32** %tmp1785
  %tmpWithVal1788 = load i32*, i32** %tmp1785
  store i32 19, i32* %tmpWithVal1788
  %asChar1789 = bitcast i32* %tmpWithVal1788 to i8*
  %tmp1790 = alloca i32*
  %malloccall1791 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1792 = bitcast i8* %malloccall1791 to i32*
  store i32* %tmpAddr1792, i32** %tmp1790
  %tmpWithVal1793 = load i32*, i32** %tmp1790
  store i32 20, i32* %tmpWithVal1793
  %asChar1794 = bitcast i32* %tmpWithVal1793 to i8*
  %tmp1795 = alloca i32*
  %malloccall1796 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1797 = bitcast i8* %malloccall1796 to i32*
  store i32* %tmpAddr1797, i32** %tmp1795
  %tmpWithVal1798 = load i32*, i32** %tmp1795
  store i32 21, i32* %tmpWithVal1798
  %asChar1799 = bitcast i32* %tmpWithVal1798 to i8*
  %tmp1800 = alloca i32*
  %malloccall1801 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1802 = bitcast i8* %malloccall1801 to i32*
  store i32* %tmpAddr1802, i32** %tmp1800
  %tmpWithVal1803 = load i32*, i32** %tmp1800
  store i32 22, i32* %tmpWithVal1803
  %asChar1804 = bitcast i32* %tmpWithVal1803 to i8*
  %tmp1805 = alloca i32*
  %malloccall1806 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1807 = bitcast i8* %malloccall1806 to i32*
  store i32* %tmpAddr1807, i32** %tmp1805
  %tmpWithVal1808 = load i32*, i32** %tmp1805
  store i32 23, i32* %tmpWithVal1808
  %asChar1809 = bitcast i32* %tmpWithVal1808 to i8*
  %tmp1810 = alloca i32*
  %malloccall1811 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr1812 = bitcast i8* %malloccall1811 to i32*
  store i32* %tmpAddr1812, i32** %tmp1810
  %tmpWithVal1813 = load i32*, i32** %tmp1810
  store i32 24, i32* %tmpWithVal1813
  %asChar1814 = bitcast i32* %tmpWithVal1813 to i8*
  %tmp1815 = alloca i32*
```

```
%malloccall1816 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr1817 = bitcast i8* %malloccall1816 to i32*
store i32* %tmpAddr1817, i32** %tmp1815
%tmpWithVal1818 = load i32*, i32** %tmp1815
store i32 25, i32* %tmpWithVal1818
%asChar1819 = bitcast i32* %tmpWithVal1818 to i8*
%push_back1820 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1694)
%push_back1821 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1699)
%push_back1822 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1704)
%push_back1823 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1709)
%push_back1824 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1714)
%push_back1825 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1719)
%push_back1826 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1724)
%push_back1827 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1729)
%push_back1828 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1734)
%push_back1829 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1739)
%push_back1830 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1744)
%push_back1831 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1749)
%push_back1832 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1754)
%push_back1833 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1759)
%push_back1834 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1764)
%push_back1835 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1769)
%push_back1836 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1774)
%push_back1837 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1779)
```

```
  %push_back1838 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1784)
  %push_back1839 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1789)
  %push_back1840 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1794)
  %push_back1841 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1799)
  %push_back1842 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1804)
  %push_back1843 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1809)
  %push_back1844 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1814)
  %push_back1845 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1688, i8* %asChar1819)
  store %list* %listAddr1688, %list** %rotor_one_in
  %key1846 = load %list*, %list** %key
  %get_at = call i8* (%list*, i32, ...) @get_at(%list* %key1846, i32 0)
  %eltAsPtr = bitcast i8* %get_at to i32*
  %eltDeref = load i32, i32* %eltAsPtr
  %rotor_one_in1847 = load %list*, %list** %rotor_one_in
  %character_location_result = call i32 @character_location(%list* %rotor_one_in1847, i32 %eltDeref)
  store i32 %character_location_result, i32* %location_on_initialization_array
  %location_on_initialization_array1848 = load i32, i32* %location_on_initialization_array
  %rotor_one_in1849 = load %list*, %list** %rotor_one_in
  %rotor_shift_result = call %list* @rotor_shift(%list* %rotor_one_in1849, i32 %location_on_initialization_array1848)
  store %list* %rotor_shift_result, %list** %rotor_one_in
  %ring_setting1850 = load %list*, %list** %ring_setting
  %get_at1851 = call i8* (%list*, i32, ...) @get_at(%list* %ring_setting1850, i32 0)
  %eltAsPtr1852 = bitcast i8* %get_at1851 to i32*
  %eltDeref1853 = load i32, i32* %eltAsPtr1852
  %rotor_one_out1854 = load %list*, %list** %rotor_one_out
  %character_location_result1855 = call i32 @character_location(%list* %rotor_one_out1854, i32 %eltDeref1853)
  store i32 %character_location_result1855, i32* %location_on_initialization_array
  %location_on_initialization_array1856 = load i32, i32* %location_on_initialization_array
```

```
%rotor_one_out1857 = load %list*, %list** %rotor_one_out
%rotor_shift_result1858 = call %list* @rotor_shift(%list* %rotor_one_out1857, i32 %location_on_initialization_array1856)
store %list* %rotor_shift_result1858, %list** %rotor_one_out
%key1859 = load %list*, %list** %key
%get_at1860 = call i8* (%list*, i32, ...) @get_at(%list* %key1859, i32 1)
%eltAsPtr1861 = bitcast i8* %get_at1860 to i32*
%eltDeref1862 = load i32, i32* %eltAsPtr1861
%rotor_two_in1863 = load %list*, %list** %rotor_two_in
%character_location_result1864 = call i32 @character_location(%list* %rotor_two_in1863, i32 %eltDeref1862)
store i32 %character_location_result1864, i32* %location_on_initialization_array
%location_on_initialization_array1865 = load i32, i32* %location_on_initialization_array
%rotor_two_in1866 = load %list*, %list** %rotor_two_in
%rotor_shift_result1867 = call %list* @rotor_shift(%list* %rotor_two_in1866, i32 %location_on_initialization_array1865)
store %list* %rotor_shift_result1867, %list** %rotor_two_in
%ring_setting1868 = load %list*, %list** %ring_setting
%get_at1869 = call i8* (%list*, i32, ...) @get_at(%list* %ring_setting1868, i32 1)
%eltAsPtr1870 = bitcast i8* %get_at1869 to i32*
%eltDeref1871 = load i32, i32* %eltAsPtr1870
%rotor_two_out1872 = load %list*, %list** %rotor_two_out
%character_location_result1873 = call i32 @character_location(%list* %rotor_two_out1872, i32 %eltDeref1871)
store i32 %character_location_result1873, i32* %location_on_initialization_array
%location_on_initialization_array1874 = load i32, i32* %location_on_initialization_array
%rotor_two_out1875 = load %list*, %list** %rotor_two_out
%rotor_shift_result1876 = call %list* @rotor_shift(%list* %rotor_two_out1875, i32 %location_on_initialization_array1874)
store %list* %rotor_shift_result1876, %list** %rotor_two_out
%key1877 = load %list*, %list** %key
%get_at1878 = call i8* (%list*, i32, ...) @get_at(%list* %key1877, i32 2)
%eltAsPtr1879 = bitcast i8* %get_at1878 to i32*
%eltDeref1880 = load i32, i32* %eltAsPtr1879
%rotor_three_in1881 = load %list*, %list** %rotor_three_in
%character_location_result1882 = call i32 @character_location(%list* %rotor_three_in1881, i32 %eltDeref1880)
store i32 %character_location_result1882, i32* %location_on_initialization_array
```

```
  %location_on_initialization_array1883 = load i32, i32* %location_on_initialization_array
  %rotor_three_in1884 = load %list*, %list** %rotor_three_in
  %rotor_shift_result1885 = call %list* @rotor_shift(%list* %rotor_three_in1884, i32 %location_on_initialization_array1883)
  store %list* %rotor_shift_result1885, %list** %rotor_three_in
  %ring_setting1886 = load %list*, %list** %ring_setting
  %get_at1887 = call i8* (%list*, i32, ...) @get_at(%list* %ring_setting1886, i32 2)
  %eltAsPtr1888 = bitcast i8* %get_at1887 to i32*
  %eltDeref1889 = load i32, i32* %eltAsPtr1888
  %rotor_three_out1890 = load %list*, %list** %rotor_three_out
  %character_location_result1891 = call i32 @character_location(%list* %rotor_three_out1890, i32 %eltDeref1889)
  store i32 %character_location_result1891, i32* %location_on_initialization_array
  %location_on_initialization_array1892 = load i32, i32* %location_on_initialization_array
  %rotor_three_out1893 = load %list*, %list** %rotor_three_out
  %rotor_shift_result1894 = call %list* @rotor_shift(%list* %rotor_three_out1893, i32 %location_on_initialization_array1892)
  store %list* %rotor_shift_result1894, %list** %rotor_three_out
  %input_text1895 = load %list*, %list** %input_text
  %messagez1896 = load %list*, %list** %messagez
  %initialize_input_text_result = call %list* @initialize_input_text(%list* %messagez1896, %list* %input_text1895)
  store %list* %initialize_input_text_result, %list** %input_text
  %length_input1897 = load i32, i32* %length_input
  %second_plugs_init1898 = load %list*, %list** %second_plugs_init
  %first_plugs_init1899 = load %list*, %list** %first_plugs_init
  %plugboard_input_count1900 = load i32, i32* %plugboard_input_count
  %input_text1901 = load %list*, %list** %input_text
  %plugboard_redirect_result = call %list* @plugboard_redirect(%list* %input_text1901, i32 %plugboard_input_count1900, %list* %first_plugs_init1899, %list* %second_plugs_init1898, i32 %length_input1897)
  store %list* %plugboard_redirect_result, %list** %input_text
  %length_input1902 = load i32, i32* %length_input
  %input_text1903 = load %list*, %list** %input_text
  %rotor_three_out1904 = load %list*, %list** %rotor_three_out
  %rotor_three_in1905 = load %list*, %list** %rotor_three_in
  %rotor_two_out1906 = load %list*, %list** %rotor_two_out
  %rotor_two_in1907 = load %list*, %list** %rotor_two_in
  %rotor_one_out1908 = load %list*, %list** %rotor_one_out
  %rotor_one_in1909 = load %list*, %list** %rotor_one_in
```

```
%encryption_decryption_result = call %list* @encryption_decryption(%list* %rotor_one_in1909, %list* %rotor_one_out1908, %list* %rotor_two_in1907, %list* %rotor_two_out1906, %list* %rotor_three_in1905, %list* %rotor_three_out1904, %list* %input_text1903, i32 %length_input1902)
store %list* %encryption_decryption_result, %list** %temp
%length_input1910 = load i32, i32* %length_input
%second_plugs_init1911 = load %list*, %list** %second_plugs_init
%first_plugs_init1912 = load %list*, %list** %first_plugs_init
%plugboard_input_count1913 = load i32, i32* %plugboard_input_count
%temp1914 = load %list*, %list** %temp
%plugboard_redirect_result1915 = call %list* @plugboard_redirect(%list* %temp1914, i32 %plugboard_input_count1913, %list* %first_plugs_init1912, %list* %second_plugs_init1911, i32 %length_input1910)
store %list* %plugboard_redirect_result1915, %list** %temp
%malloccall1916 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr1917 = bitcast i8* %malloccall1916 to %list*
%init_list1918 = call i32 (%list*, ...) @init_list(%list* %listAddr1917)
%tmp1919 = alloca i8*
%tmpAddr1921 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1921, i8** %tmp1919
%tmpWithVal1922 = load i8*, i8** %tmp1919
store i8 79, i8* %tmpWithVal1922
%tmp1923 = alloca i8*
%tmpAddr1925 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1925, i8** %tmp1923
%tmpWithVal1926 = load i8*, i8** %tmp1923
store i8 117, i8* %tmpWithVal1926
%tmp1927 = alloca i8*
%tmpAddr1929 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1929, i8** %tmp1927
%tmpWithVal1930 = load i8*, i8** %tmp1927
store i8 116, i8* %tmpWithVal1930
%tmp1931 = alloca i8*
%tmpAddr1933 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1933, i8** %tmp1931
%tmpWithVal1934 = load i8*, i8** %tmp1931
store i8 112, i8* %tmpWithVal1934
```

```
%tmp1935 = alloca i8*
%tmpAddr1937 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1937, i8** %tmp1935
%tmpWithVal1938 = load i8*, i8** %tmp1935
store i8 117, i8* %tmpWithVal1938
%tmp1939 = alloca i8*
%tmpAddr1941 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1941, i8** %tmp1939
%tmpWithVal1942 = load i8*, i8** %tmp1939
store i8 116, i8* %tmpWithVal1942
%tmp1943 = alloca i8*
%tmpAddr1945 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1945, i8** %tmp1943
%tmpWithVal1946 = load i8*, i8** %tmp1943
store i8 32, i8* %tmpWithVal1946
%tmp1947 = alloca i8*
%tmpAddr1949 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1949, i8** %tmp1947
%tmpWithVal1950 = load i8*, i8** %tmp1947
store i8 116, i8* %tmpWithVal1950
%tmp1951 = alloca i8*
%tmpAddr1953 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1953, i8** %tmp1951
%tmpWithVal1954 = load i8*, i8** %tmp1951
store i8 101, i8* %tmpWithVal1954
%tmp1955 = alloca i8*
%tmpAddr1957 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1957, i8** %tmp1955
%tmpWithVal1958 = load i8*, i8** %tmp1955
store i8 120, i8* %tmpWithVal1958
%tmp1959 = alloca i8*
%tmpAddr1961 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
store i8* %tmpAddr1961, i8** %tmp1959
%tmpWithVal1962 = load i8*, i8** %tmp1959
store i8 116, i8* %tmpWithVal1962
%tmp1963 = alloca i8*
```

```
  %tmpAddr1965 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr1965, i8** %tmp1963
  %tmpWithVal1966 = load i8*, i8** %tmp1963
  store i8 58, i8* %tmpWithVal1966
  %tmp1967 = alloca i8*
  %tmpAddr1969 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i32))
  store i8* %tmpAddr1969, i8** %tmp1967
  %tmpWithVal1970 = load i8*, i8** %tmp1967
  store i8 32, i8* %tmpWithVal1970
  %push_back1971 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1922)
  %push_back1972 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1926)
  %push_back1973 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1930)
  %push_back1974 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1934)
  %push_back1975 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1938)
  %push_back1976 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1942)
  %push_back1977 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1946)
  %push_back1978 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1950)
  %push_back1979 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1954)
  %push_back1980 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1958)
  %push_back1981 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1962)
  %push_back1982 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1966)
  %push_back1983 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr1917, i8* %tmpWithVal1970)
  %print_list_char = call i32 @print_list_char(%list* %listAddr1917)
  store i32 0, i32* %final_output_counter
  br label %while

while:                                            ; preds = %while_body, %entry
  %final_output_counter1991 = load i32, i32* %final_output_counter
  %length_input1992 = load i32, i32* %length_input
  %tmp1993 = icmp slt i32 %final_output_counter1991,
%length_input1992
  br i1 %tmp1993, label %while_body, label %merge

while_body:                                     ; preds = %while
  %temp1984 = load %list*, %list** %temp
  %final_output_counter1985 = load i32, i32* %final_output_counter
  %get_at1986 = call i8* (%list*, i32, ...) @get_at(%list* %temp1984, i32
%final_output_counter1985)
  %eltAsPtr1987 = bitcast i8* %get_at1986 to i32*
  %eltDeref1988 = load i32, i32* %eltAsPtr1987
  %corresponding_char = call i32 @corresponding_char(i32
%eltDeref1988)
  %final_output_counter1989 = load i32, i32* %final_output_counter
  %tmp1990 = add i32 %final_output_counter1989, 1
  store i32 %tmp1990, i32* %final_output_counter
  br label %while

merge:                                          ; preds = %while
  ret i32 0
}

define %list* @plugboard_redirect(%list* %text_redirect, i32
%count_for_input, %list* %first_plugs, %list* %second_plugs, i32
%message_length_plug) {
entry:
  %text_redirect1 = alloca %list*
  store %list* %text_redirect, %list** %text_redirect1
  %count_for_input2 = alloca i32
  store i32 %count_for_input, i32* %count_for_input2
  %first_plugs3 = alloca %list*
  store %list* %first_plugs, %list** %first_plugs3
  %second_plugs4 = alloca %list*
  store %list* %second_plugs, %list** %second_plugs4
  %message_length_plug5 = alloca i32
  store i32 %message_length_plug, i32* %message_length_plug5
  %list_of_char_redirect = alloca %list*
  %sub_list_char = alloca %list*
  %temp_sub_val_holder = alloca %list*
  %loc_holder = alloca %list*
  %loc_holding_for_text = alloca i32
  %counter = alloca i32
  %string_redirect_length = alloca i32
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr = bitcast i8* %malloccall to %list*
  %init_list = call i32 (%list*, ...) @init_list(%list* %listAddr)
  %tmp = alloca i32*
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr = bitcast i8* %malloccall6 to i32*
  store i32* %tmpAddr, i32** %tmp
  %tmpWithVal = load i32*, i32** %tmp
  store i32 0, i32* %tmpWithVal
  %asChar = bitcast i32* %tmpWithVal to i8*
  %tmp7 = alloca i32*
  %malloccall8 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr9 = bitcast i8* %malloccall8 to i32*
  store i32* %tmpAddr9, i32** %tmp7
  %tmpWithVal10 = load i32*, i32** %tmp7
  store i32 1, i32* %tmpWithVal10
  %asChar11 = bitcast i32* %tmpWithVal10 to i8*
  %tmp12 = alloca i32*
  %malloccall13 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr14 = bitcast i8* %malloccall13 to i32*
  store i32* %tmpAddr14, i32** %tmp12
  %tmpWithVal15 = load i32*, i32** %tmp12
  store i32 2, i32* %tmpWithVal15
  %asChar16 = bitcast i32* %tmpWithVal15 to i8*
  %tmp17 = alloca i32*
  %malloccall18 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr19 = bitcast i8* %malloccall18 to i32*
  store i32* %tmpAddr19, i32** %tmp17
  %tmpWithVal20 = load i32*, i32** %tmp17
  store i32 3, i32* %tmpWithVal20
  %asChar21 = bitcast i32* %tmpWithVal20 to i8*
  %tmp22 = alloca i32*
  %malloccall23 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr24 = bitcast i8* %malloccall23 to i32*
  store i32* %tmpAddr24, i32** %tmp22
  %tmpWithVal25 = load i32*, i32** %tmp22
  store i32 4, i32* %tmpWithVal25
  %asChar26 = bitcast i32* %tmpWithVal25 to i8*
  %tmp27 = alloca i32*
  %malloccall28 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr29 = bitcast i8* %malloccall28 to i32*
  store i32* %tmpAddr29, i32** %tmp27
  %tmpWithVal30 = load i32*, i32** %tmp27
  store i32 5, i32* %tmpWithVal30
  %asChar31 = bitcast i32* %tmpWithVal30 to i8*
  %tmp32 = alloca i32*
  %malloccall33 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr34 = bitcast i8* %malloccall33 to i32*
  store i32* %tmpAddr34, i32** %tmp32
  %tmpWithVal35 = load i32*, i32** %tmp32
  store i32 6, i32* %tmpWithVal35
  %asChar36 = bitcast i32* %tmpWithVal35 to i8*
  %tmp37 = alloca i32*
  %malloccall38 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr39 = bitcast i8* %malloccall38 to i32*
  store i32* %tmpAddr39, i32** %tmp37
  %tmpWithVal40 = load i32*, i32** %tmp37
  store i32 7, i32* %tmpWithVal40
  %asChar41 = bitcast i32* %tmpWithVal40 to i8*
  %tmp42 = alloca i32*
  %malloccall43 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr44 = bitcast i8* %malloccall43 to i32*
  store i32* %tmpAddr44, i32** %tmp42
  %tmpWithVal45 = load i32*, i32** %tmp42
  store i32 8, i32* %tmpWithVal45
  %asChar46 = bitcast i32* %tmpWithVal45 to i8*
  %tmp47 = alloca i32*
  %malloccall48 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr49 = bitcast i8* %malloccall48 to i32*
  store i32* %tmpAddr49, i32** %tmp47
  %tmpWithVal50 = load i32*, i32** %tmp47
  store i32 9, i32* %tmpWithVal50
  %asChar51 = bitcast i32* %tmpWithVal50 to i8*
  %tmp52 = alloca i32*
  %malloccall53 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr54 = bitcast i8* %malloccall53 to i32*
```

```
store i32* %tmpAddr54, i32** %tmp52
%tmpWithVal55 = load i32*, i32** %tmp52
store i32 10, i32* %tmpWithVal55
%asChar56 = bitcast i32* %tmpWithVal55 to i8*
%tmp57 = alloca i32*
%malloccall58 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr59 = bitcast i8* %malloccall58 to i32*
store i32* %tmpAddr59, i32** %tmp57
%tmpWithVal60 = load i32*, i32** %tmp57
store i32 11, i32* %tmpWithVal60
%asChar61 = bitcast i32* %tmpWithVal60 to i8*
%tmp62 = alloca i32*
%malloccall63 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr64 = bitcast i8* %malloccall63 to i32*
store i32* %tmpAddr64, i32** %tmp62
%tmpWithVal65 = load i32*, i32** %tmp62
store i32 12, i32* %tmpWithVal65
%asChar66 = bitcast i32* %tmpWithVal65 to i8*
%tmp67 = alloca i32*
%malloccall68 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr69 = bitcast i8* %malloccall68 to i32*
store i32* %tmpAddr69, i32** %tmp67
%tmpWithVal70 = load i32*, i32** %tmp67
store i32 13, i32* %tmpWithVal70
%asChar71 = bitcast i32* %tmpWithVal70 to i8*
%tmp72 = alloca i32*
%malloccall73 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr74 = bitcast i8* %malloccall73 to i32*
store i32* %tmpAddr74, i32** %tmp72
%tmpWithVal75 = load i32*, i32** %tmp72
store i32 14, i32* %tmpWithVal75
%asChar76 = bitcast i32* %tmpWithVal75 to i8*
%tmp77 = alloca i32*
%malloccall78 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr79 = bitcast i8* %malloccall78 to i32*
store i32* %tmpAddr79, i32** %tmp77
%tmpWithVal80 = load i32*, i32** %tmp77
store i32 15, i32* %tmpWithVal80
%asChar81 = bitcast i32* %tmpWithVal80 to i8*
```

```
  %tmp82 = alloca i32*
  %malloccall83 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr84 = bitcast i8* %malloccall83 to i32*
  store i32* %tmpAddr84, i32** %tmp82
  %tmpWithVal85 = load i32*, i32** %tmp82
  store i32 16, i32* %tmpWithVal85
  %asChar86 = bitcast i32* %tmpWithVal85 to i8*
  %tmp87 = alloca i32*
  %malloccall88 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr89 = bitcast i8* %malloccall88 to i32*
  store i32* %tmpAddr89, i32** %tmp87
  %tmpWithVal90 = load i32*, i32** %tmp87
  store i32 17, i32* %tmpWithVal90
  %asChar91 = bitcast i32* %tmpWithVal90 to i8*
  %tmp92 = alloca i32*
  %malloccall93 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr94 = bitcast i8* %malloccall93 to i32*
  store i32* %tmpAddr94, i32** %tmp92
  %tmpWithVal95 = load i32*, i32** %tmp92
  store i32 18, i32* %tmpWithVal95
  %asChar96 = bitcast i32* %tmpWithVal95 to i8*
  %tmp97 = alloca i32*
  %malloccall98 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr99 = bitcast i8* %malloccall98 to i32*
  store i32* %tmpAddr99, i32** %tmp97
  %tmpWithVal100 = load i32*, i32** %tmp97
  store i32 19, i32* %tmpWithVal100
  %asChar101 = bitcast i32* %tmpWithVal100 to i8*
  %tmp102 = alloca i32*
  %malloccall103 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr104 = bitcast i8* %malloccall103 to i32*
  store i32* %tmpAddr104, i32** %tmp102
  %tmpWithVal105 = load i32*, i32** %tmp102
  store i32 20, i32* %tmpWithVal105
  %asChar106 = bitcast i32* %tmpWithVal105 to i8*
  %tmp107 = alloca i32*
  %malloccall108 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr109 = bitcast i8* %malloccall108 to i32*
```

```
	store i32* %tmpAddr109, i32** %tmp107
	%tmpWithVal110 = load i32*, i32** %tmp107
	store i32 21, i32* %tmpWithVal110
	%asChar111 = bitcast i32* %tmpWithVal110 to i8*
	%tmp112 = alloca i32*
	%malloccall113 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr114 = bitcast i8* %malloccall113 to i32*
	store i32* %tmpAddr114, i32** %tmp112
	%tmpWithVal115 = load i32*, i32** %tmp112
	store i32 22, i32* %tmpWithVal115
	%asChar116 = bitcast i32* %tmpWithVal115 to i8*
	%tmp117 = alloca i32*
	%malloccall118 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr119 = bitcast i8* %malloccall118 to i32*
	store i32* %tmpAddr119, i32** %tmp117
	%tmpWithVal120 = load i32*, i32** %tmp117
	store i32 23, i32* %tmpWithVal120
	%asChar121 = bitcast i32* %tmpWithVal120 to i8*
	%tmp122 = alloca i32*
	%malloccall123 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr124 = bitcast i8* %malloccall123 to i32*
	store i32* %tmpAddr124, i32** %tmp122
	%tmpWithVal125 = load i32*, i32** %tmp122
	store i32 24, i32* %tmpWithVal125
	%asChar126 = bitcast i32* %tmpWithVal125 to i8*
	%tmp127 = alloca i32*
	%malloccall128 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
	%tmpAddr129 = bitcast i8* %malloccall128 to i32*
	store i32* %tmpAddr129, i32** %tmp127
	%tmpWithVal130 = load i32*, i32** %tmp127
	store i32 25, i32* %tmpWithVal130
	%asChar131 = bitcast i32* %tmpWithVal130 to i8*
	%push_back = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar)
	%push_back132 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar11)
	%push_back133 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar16)
	%push_back134 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar21)
```

```
%push_back135 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar26)
%push_back136 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar31)
%push_back137 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar36)
%push_back138 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar41)
%push_back139 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar46)
%push_back140 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar51)
%push_back141 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar56)
%push_back142 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar61)
%push_back143 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar66)
%push_back144 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar71)
%push_back145 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar76)
%push_back146 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar81)
%push_back147 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar86)
%push_back148 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar91)
%push_back149 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar96)
%push_back150 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar101)
%push_back151 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar106)
%push_back152 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar111)
%push_back153 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar116)
%push_back154 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar121)
%push_back155 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar126)
%push_back156 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar131)
```

```
  store %list* %listAddr, %list** %list_of_char_redirect
  %malloccall157 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr158 = bitcast i8* %malloccall157 to %list*
  %init_list159 = call i32 (%list*, ...) @init_list(%list* %listAddr158)
  %tmp160 = alloca i32*
  %malloccall161 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr162 = bitcast i8* %malloccall161 to i32*
  store i32* %tmpAddr162, i32** %tmp160
  %tmpWithVal163 = load i32*, i32** %tmp160
  store i32 0, i32* %tmpWithVal163
  %asChar164 = bitcast i32* %tmpWithVal163 to i8*
  %tmp165 = alloca i32*
  %malloccall166 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr167 = bitcast i8* %malloccall166 to i32*
  store i32* %tmpAddr167, i32** %tmp165
  %tmpWithVal168 = load i32*, i32** %tmp165
  store i32 1, i32* %tmpWithVal168
  %asChar169 = bitcast i32* %tmpWithVal168 to i8*
  %tmp170 = alloca i32*
  %malloccall171 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr172 = bitcast i8* %malloccall171 to i32*
  store i32* %tmpAddr172, i32** %tmp170
  %tmpWithVal173 = load i32*, i32** %tmp170
  store i32 2, i32* %tmpWithVal173
  %asChar174 = bitcast i32* %tmpWithVal173 to i8*
  %tmp175 = alloca i32*
  %malloccall176 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr177 = bitcast i8* %malloccall176 to i32*
  store i32* %tmpAddr177, i32** %tmp175
  %tmpWithVal178 = load i32*, i32** %tmp175
  store i32 3, i32* %tmpWithVal178
  %asChar179 = bitcast i32* %tmpWithVal178 to i8*
  %tmp180 = alloca i32*
  %malloccall181 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr182 = bitcast i8* %malloccall181 to i32*
  store i32* %tmpAddr182, i32** %tmp180
  %tmpWithVal183 = load i32*, i32** %tmp180
  store i32 4, i32* %tmpWithVal183
  %asChar184 = bitcast i32* %tmpWithVal183 to i8*
  %tmp185 = alloca i32*
  %malloccall186 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr187 = bitcast i8* %malloccall186 to i32*
  store i32* %tmpAddr187, i32** %tmp185
  %tmpWithVal188 = load i32*, i32** %tmp185
  store i32 5, i32* %tmpWithVal188
  %asChar189 = bitcast i32* %tmpWithVal188 to i8*
  %tmp190 = alloca i32*
  %malloccall191 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr192 = bitcast i8* %malloccall191 to i32*
  store i32* %tmpAddr192, i32** %tmp190
  %tmpWithVal193 = load i32*, i32** %tmp190
  store i32 6, i32* %tmpWithVal193
  %asChar194 = bitcast i32* %tmpWithVal193 to i8*
  %tmp195 = alloca i32*
  %malloccall196 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr197 = bitcast i8* %malloccall196 to i32*
  store i32* %tmpAddr197, i32** %tmp195
  %tmpWithVal198 = load i32*, i32** %tmp195
  store i32 7, i32* %tmpWithVal198
  %asChar199 = bitcast i32* %tmpWithVal198 to i8*
  %tmp200 = alloca i32*
  %malloccall201 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr202 = bitcast i8* %malloccall201 to i32*
  store i32* %tmpAddr202, i32** %tmp200
  %tmpWithVal203 = load i32*, i32** %tmp200
  store i32 8, i32* %tmpWithVal203
  %asChar204 = bitcast i32* %tmpWithVal203 to i8*
  %tmp205 = alloca i32*
  %malloccall206 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr207 = bitcast i8* %malloccall206 to i32*
  store i32* %tmpAddr207, i32** %tmp205
  %tmpWithVal208 = load i32*, i32** %tmp205
  store i32 9, i32* %tmpWithVal208
  %asChar209 = bitcast i32* %tmpWithVal208 to i8*
  %tmp210 = alloca i32*
  %malloccall211 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
%tmpAddr212 = bitcast i8* %malloccall211 to i32*
store i32* %tmpAddr212, i32** %tmp210
%tmpWithVal213 = load i32*, i32** %tmp210
store i32 10, i32* %tmpWithVal213
%asChar214 = bitcast i32* %tmpWithVal213 to i8*
%tmp215 = alloca i32*
%malloccall216 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr217 = bitcast i8* %malloccall216 to i32*
store i32* %tmpAddr217, i32** %tmp215
%tmpWithVal218 = load i32*, i32** %tmp215
store i32 11, i32* %tmpWithVal218
%asChar219 = bitcast i32* %tmpWithVal218 to i8*
%tmp220 = alloca i32*
%malloccall221 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr222 = bitcast i8* %malloccall221 to i32*
store i32* %tmpAddr222, i32** %tmp220
%tmpWithVal223 = load i32*, i32** %tmp220
store i32 12, i32* %tmpWithVal223
%asChar224 = bitcast i32* %tmpWithVal223 to i8*
%tmp225 = alloca i32*
%malloccall226 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr227 = bitcast i8* %malloccall226 to i32*
store i32* %tmpAddr227, i32** %tmp225
%tmpWithVal228 = load i32*, i32** %tmp225
store i32 13, i32* %tmpWithVal228
%asChar229 = bitcast i32* %tmpWithVal228 to i8*
%tmp230 = alloca i32*
%malloccall231 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr232 = bitcast i8* %malloccall231 to i32*
store i32* %tmpAddr232, i32** %tmp230
%tmpWithVal233 = load i32*, i32** %tmp230
store i32 14, i32* %tmpWithVal233
%asChar234 = bitcast i32* %tmpWithVal233 to i8*
%tmp235 = alloca i32*
%malloccall236 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr237 = bitcast i8* %malloccall236 to i32*
store i32* %tmpAddr237, i32** %tmp235
%tmpWithVal238 = load i32*, i32** %tmp235
store i32 15, i32* %tmpWithVal238
```

```
%asChar239 = bitcast i32* %tmpWithVal238 to i8*
%tmp240 = alloca i32*
%malloccall241 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr242 = bitcast i8* %malloccall241 to i32*
store i32* %tmpAddr242, i32** %tmp240
%tmpWithVal243 = load i32*, i32** %tmp240
store i32 16, i32* %tmpWithVal243
%asChar244 = bitcast i32* %tmpWithVal243 to i8*
%tmp245 = alloca i32*
%malloccall246 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr247 = bitcast i8* %malloccall246 to i32*
store i32* %tmpAddr247, i32** %tmp245
%tmpWithVal248 = load i32*, i32** %tmp245
store i32 17, i32* %tmpWithVal248
%asChar249 = bitcast i32* %tmpWithVal248 to i8*
%tmp250 = alloca i32*
%malloccall251 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr252 = bitcast i8* %malloccall251 to i32*
store i32* %tmpAddr252, i32** %tmp250
%tmpWithVal253 = load i32*, i32** %tmp250
store i32 18, i32* %tmpWithVal253
%asChar254 = bitcast i32* %tmpWithVal253 to i8*
%tmp255 = alloca i32*
%malloccall256 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr257 = bitcast i8* %malloccall256 to i32*
store i32* %tmpAddr257, i32** %tmp255
%tmpWithVal258 = load i32*, i32** %tmp255
store i32 19, i32* %tmpWithVal258
%asChar259 = bitcast i32* %tmpWithVal258 to i8*
%tmp260 = alloca i32*
%malloccall261 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr262 = bitcast i8* %malloccall261 to i32*
store i32* %tmpAddr262, i32** %tmp260
%tmpWithVal263 = load i32*, i32** %tmp260
store i32 20, i32* %tmpWithVal263
%asChar264 = bitcast i32* %tmpWithVal263 to i8*
%tmp265 = alloca i32*
%malloccall266 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
%tmpAddr267 = bitcast i8* %malloccall266 to i32*
store i32* %tmpAddr267, i32** %tmp265
%tmpWithVal268 = load i32*, i32** %tmp265
store i32 21, i32* %tmpWithVal268
%asChar269 = bitcast i32* %tmpWithVal268 to i8*
%tmp270 = alloca i32*
%malloccall271 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr272 = bitcast i8* %malloccall271 to i32*
store i32* %tmpAddr272, i32** %tmp270
%tmpWithVal273 = load i32*, i32** %tmp270
store i32 22, i32* %tmpWithVal273
%asChar274 = bitcast i32* %tmpWithVal273 to i8*
%tmp275 = alloca i32*
%malloccall276 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr277 = bitcast i8* %malloccall276 to i32*
store i32* %tmpAddr277, i32** %tmp275
%tmpWithVal278 = load i32*, i32** %tmp275
store i32 23, i32* %tmpWithVal278
%asChar279 = bitcast i32* %tmpWithVal278 to i8*
%tmp280 = alloca i32*
%malloccall281 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr282 = bitcast i8* %malloccall281 to i32*
store i32* %tmpAddr282, i32** %tmp280
%tmpWithVal283 = load i32*, i32** %tmp280
store i32 24, i32* %tmpWithVal283
%asChar284 = bitcast i32* %tmpWithVal283 to i8*
%tmp285 = alloca i32*
%malloccall286 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr287 = bitcast i8* %malloccall286 to i32*
store i32* %tmpAddr287, i32** %tmp285
%tmpWithVal288 = load i32*, i32** %tmp285
store i32 25, i32* %tmpWithVal288
%asChar289 = bitcast i32* %tmpWithVal288 to i8*
%push_back290 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar164)
%push_back291 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar169)
%push_back292 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar174)
```

```
%push_back293 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar179)
%push_back294 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar184)
%push_back295 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar189)
%push_back296 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar194)
%push_back297 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar199)
%push_back298 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar204)
%push_back299 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar209)
%push_back300 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar214)
%push_back301 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar219)
%push_back302 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar224)
%push_back303 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar229)
%push_back304 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar234)
%push_back305 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar239)
%push_back306 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar244)
%push_back307 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar249)
%push_back308 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar254)
%push_back309 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar259)
%push_back310 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar264)
%push_back311 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar269)
%push_back312 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar274)
%push_back313 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar279)
%push_back314 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar284)
```

```
%push_back315 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr158, i8* %asChar289)
store %list* %listAddr158, %list** %sub_list_char
%malloccall316 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr317 = bitcast i8* %malloccall316 to %list*
%init_list318 = call i32 (%list*, ...) @init_list(%list* %listAddr317)
%tmp319 = alloca i32*
%malloccall320 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr321 = bitcast i8* %malloccall320 to i32*
store i32* %tmpAddr321, i32** %tmp319
%tmpWithVal322 = load i32*, i32** %tmp319
store i32 1, i32* %tmpWithVal322
%asChar323 = bitcast i32* %tmpWithVal322 to i8*
%tmp324 = alloca i32*
%malloccall325 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr326 = bitcast i8* %malloccall325 to i32*
store i32* %tmpAddr326, i32** %tmp324
%tmpWithVal327 = load i32*, i32** %tmp324
store i32 1, i32* %tmpWithVal327
%asChar328 = bitcast i32* %tmpWithVal327 to i8*
%push_back329 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr317, i8* %asChar323)
%push_back330 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr317, i8* %asChar328)
store %list* %listAddr317, %list** %temp_sub_val_holder
%malloccall331 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr332 = bitcast i8* %malloccall331 to %list*
%init_list333 = call i32 (%list*, ...) @init_list(%list* %listAddr332)
%tmp334 = alloca i32*
%malloccall335 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr336 = bitcast i8* %malloccall335 to i32*
store i32* %tmpAddr336, i32** %tmp334
%tmpWithVal337 = load i32*, i32** %tmp334
store i32 1, i32* %tmpWithVal337
%asChar338 = bitcast i32* %tmpWithVal337 to i8*
%tmp339 = alloca i32*
%malloccall340 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr341 = bitcast i8* %malloccall340 to i32*
```

```
  store i32* %tmpAddr341, i32** %tmp339
  %tmpWithVal342 = load i32*, i32** %tmp339
  store i32 1, i32* %tmpWithVal342
  %asChar343 = bitcast i32* %tmpWithVal342 to i8*
  %push_back344 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr332, i8* %asChar338)
  %push_back345 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr332, i8* %asChar343)
  store %list* %listAddr332, %list** %loc_holder
  store i32 0, i32* %loc_holding_for_text
  store i32 0, i32* %counter
  store i32 0, i32* %string_redirect_length
  br label %if

if:                                               ; preds = %entry
  %count_for_input442 = load i32, i32* %count_for_input2
  %tmp443 = icmp eq i32 %count_for_input442, 0
  br i1 %tmp443, label %if_body, label %merge

if_body:                                          ; preds = %if
  br label %while

merge:                                            ; preds = %if, %merge441
  store i32 0, i32* %string_redirect_length
  br label %while444

while:                                            ; preds = %while_body, %if_body
  %counter439 = load i32, i32* %counter
  %tmp440 = icmp ne i32 %counter439, 10
  br i1 %tmp440, label %while_body, label %merge441

while_body:                                       ; preds = %while
  %temp_sub_val_holder346 = load %list*, %list** %temp_sub_val_holder
  %first_plugs347 = load %list*, %list** %first_plugs3
  %counter348 = load i32, i32* %counter
  %get_at = call i8* (%list*, i32, ...) @get_at(%list* %first_plugs347, i32 %counter348)
  %eltAsPtr = bitcast i8* %get_at to i32*
  %eltDeref = load i32, i32* %eltAsPtr
  %tmp349 = alloca i32*
  %malloccall350 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr351 = bitcast i8* %malloccall350 to i32*
  store i32* %tmpAddr351, i32** %tmp349
  %tmpWithVal352 = load i32*, i32** %tmp349
  store i32 %eltDeref, i32* %tmpWithVal352
  %asChar353 = bitcast i32* %tmpWithVal352 to i8*
  %set_at = call %list* (%list*, i32, i8*, ...) @set_at(%list* %temp_sub_val_holder346, i32 0, i8* %asChar353)
  %temp_sub_val_holder354 = load %list*, %list** %temp_sub_val_holder
  %second_plugs355 = load %list*, %list** %second_plugs4
  %counter356 = load i32, i32* %counter
  %get_at357 = call i8* (%list*, i32, ...) @get_at(%list* %second_plugs355, i32 %counter356)
  %eltAsPtr358 = bitcast i8* %get_at357 to i32*
  %eltDeref359 = load i32, i32* %eltAsPtr358
  %tmp360 = alloca i32*
  %malloccall361 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr362 = bitcast i8* %malloccall361 to i32*
  store i32* %tmpAddr362, i32** %tmp360
  %tmpWithVal363 = load i32*, i32** %tmp360
  store i32 %eltDeref359, i32* %tmpWithVal363
  %asChar364 = bitcast i32* %tmpWithVal363 to i8*
  %set_at365 = call %list* (%list*, i32, i8*, ...) @set_at(%list* %temp_sub_val_holder354, i32 1, i8* %asChar364)
  %loc_holder366 = load %list*, %list** %loc_holder
  %list_of_char_redirect367 = load %list*, %list** %list_of_char_redirect
  %temp_sub_val_holder368 = load %list*, %list** %temp_sub_val_holder
  %get_at369 = call i8* (%list*, i32, ...) @get_at(%list* %temp_sub_val_holder368, i32 0)
  %eltAsPtr370 = bitcast i8* %get_at369 to i32*
  %eltDeref371 = load i32, i32* %eltAsPtr370
  %get_at372 = call i8* (%list*, i32, ...) @get_at(%list* %list_of_char_redirect367, i32 %eltDeref371)
  %eltAsPtr373 = bitcast i8* %get_at372 to i32*
  %eltDeref374 = load i32, i32* %eltAsPtr373
  %list_of_char_redirect375 = load %list*, %list** %list_of_char_redirect
  %character_location_result = call i32 @character_location(%list* %list_of_char_redirect375, i32 %eltDeref374)
  %tmp376 = alloca i32*
  %malloccall377 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr378 = bitcast i8* %malloccall377 to i32*
  store i32* %tmpAddr378, i32** %tmp376
  %tmpWithVal379 = load i32*, i32** %tmp376
  store i32 %character_location_result, i32* %tmpWithVal379
  %asChar380 = bitcast i32* %tmpWithVal379 to i8*
```

```
    %set_at381 = call %list* (%list*, i32, i8*, ...) @set_at(%list* %loc_holder366, i32 0, i8* %asChar380)
    %loc_holder382 = load %list*, %list** %loc_holder
    %list_of_char_redirect383 = load %list*, %list** %list_of_char_redirect
    %temp_sub_val_holder384 = load %list*, %list** %temp_sub_val_holder
    %get_at385 = call i8* (%list*, i32, ...) @get_at(%list* %temp_sub_val_holder384, i32 1)
    %eltAsPtr386 = bitcast i8* %get_at385 to i32*
    %eltDeref387 = load i32, i32* %eltAsPtr386
    %get_at388 = call i8* (%list*, i32, ...) @get_at(%list* %list_of_char_redirect383, i32 %eltDeref387)
    %eltAsPtr389 = bitcast i8* %get_at388 to i32*
    %eltDeref390 = load i32, i32* %eltAsPtr389
    %list_of_char_redirect391 = load %list*, %list** %list_of_char_redirect
    %character_location_result392 = call i32 @character_location(%list* %list_of_char_redirect391, i32 %eltDeref390)
    %tmp393 = alloca i32*
    %malloccall394 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr395 = bitcast i8* %malloccall394 to i32*
    store i32* %tmpAddr395, i32** %tmp393
    %tmpWithVal396 = load i32*, i32** %tmp393
    store i32 %character_location_result392, i32* %tmpWithVal396
    %asChar397 = bitcast i32* %tmpWithVal396 to i8*
    %set_at398 = call %list* (%list*, i32, i8*, ...) @set_at(%list* %loc_holder382, i32 1, i8* %asChar397)
    %sub_list_char399 = load %list*, %list** %sub_list_char
    %loc_holder400 = load %list*, %list** %loc_holder
    %get_at401 = call i8* (%list*, i32, ...) @get_at(%list* %loc_holder400, i32 0)
    %eltAsPtr402 = bitcast i8* %get_at401 to i32*
    %eltDeref403 = load i32, i32* %eltAsPtr402
    %list_of_char_redirect404 = load %list*, %list** %list_of_char_redirect
    %loc_holder405 = load %list*, %list** %loc_holder
    %get_at406 = call i8* (%list*, i32, ...) @get_at(%list* %loc_holder405, i32 1)
    %eltAsPtr407 = bitcast i8* %get_at406 to i32*
    %eltDeref408 = load i32, i32* %eltAsPtr407
    %get_at409 = call i8* (%list*, i32, ...) @get_at(%list* %list_of_char_redirect404, i32 %eltDeref408)
    %eltAsPtr410 = bitcast i8* %get_at409 to i32*
    %eltDeref411 = load i32, i32* %eltAsPtr410
    %tmp412 = alloca i32*
```

```
  %malloccall413 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr414 = bitcast i8* %malloccall413 to i32*
  store i32* %tmpAddr414, i32** %tmp412
  %tmpWithVal415 = load i32*, i32** %tmp412
  store i32 %eltDeref411, i32* %tmpWithVal415
  %asChar416 = bitcast i32* %tmpWithVal415 to i8*
  %set_at417 = call %list* (%list*, i32, i8*, ...) @set_at(%list* %sub_list_char399, i32 %eltDeref403, i8* %asChar416)
  %sub_list_char418 = load %list*, %list** %sub_list_char
  %loc_holder419 = load %list*, %list** %loc_holder
  %get_at420 = call i8* (%list*, i32, ...) @get_at(%list* %loc_holder419, i32 1)
  %eltAsPtr421 = bitcast i8* %get_at420 to i32*
  %eltDeref422 = load i32, i32* %eltAsPtr421
  %list_of_char_redirect423 = load %list*, %list** %list_of_char_redirect
  %loc_holder424 = load %list*, %list** %loc_holder
  %get_at425 = call i8* (%list*, i32, ...) @get_at(%list* %loc_holder424, i32 0)
  %eltAsPtr426 = bitcast i8* %get_at425 to i32*
  %eltDeref427 = load i32, i32* %eltAsPtr426
  %get_at428 = call i8* (%list*, i32, ...) @get_at(%list* %list_of_char_redirect423, i32 %eltDeref427)
  %eltAsPtr429 = bitcast i8* %get_at428 to i32*
  %eltDeref430 = load i32, i32* %eltAsPtr429
  %tmp431 = alloca i32*
  %malloccall432 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr433 = bitcast i8* %malloccall432 to i32*
  store i32* %tmpAddr433, i32** %tmp431
  %tmpWithVal434 = load i32*, i32** %tmp431
  store i32 %eltDeref430, i32* %tmpWithVal434
  %asChar435 = bitcast i32* %tmpWithVal434 to i8*
  %set_at436 = call %list* (%list*, i32, i8*, ...) @set_at(%list* %sub_list_char418, i32 %eltDeref422, i8* %asChar435)
  %counter437 = load i32, i32* %counter
  %tmp438 = add i32 %counter437, 1
  store i32 %tmp438, i32* %counter
  br label %while

merge441:                                         ; preds = %while
  br label %merge

while444:                                         ; preds = %while_body445, %merge
```

```
  %string_redirect_length468 = load i32, i32* %string_redirect_length
  %message_length_plug469 = load i32, i32* %message_length_plug5
  %tmp470 = icmp slt i32 %string_redirect_length468, %message_length_plug469
  br i1 %tmp470, label %while_body445, label %merge471

while_body445:                                    ; preds = %while444
  %text_redirect446 = load %list*, %list** %text_redirect1
  %string_redirect_length447 = load i32, i32* %string_redirect_length
  %get_at448 = call i8* (%list*, i32, ...) @get_at(%list* %text_redirect446, i32 %string_redirect_length447)
  %eltAsPtr449 = bitcast i8* %get_at448 to i32*
  %eltDeref450 = load i32, i32* %eltAsPtr449
  %list_of_char_redirect451 = load %list*, %list** %list_of_char_redirect
  %character_location_result452 = call i32 @character_location(%list* %list_of_char_redirect451, i32 %eltDeref450)
  store i32 %character_location_result452, i32* %loc_holding_for_text
  %text_redirect453 = load %list*, %list** %text_redirect1
  %string_redirect_length454 = load i32, i32* %string_redirect_length
  %sub_list_char455 = load %list*, %list** %sub_list_char
  %loc_holding_for_text456 = load i32, i32* %loc_holding_for_text
  %get_at457 = call i8* (%list*, i32, ...) @get_at(%list* %sub_list_char455, i32 %loc_holding_for_text456)
  %eltAsPtr458 = bitcast i8* %get_at457 to i32*
  %eltDeref459 = load i32, i32* %eltAsPtr458
  %tmp460 = alloca i32*
  %malloccall461 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr462 = bitcast i8* %malloccall461 to i32*
  store i32* %tmpAddr462, i32** %tmp460
  %tmpWithVal463 = load i32*, i32** %tmp460
  store i32 %eltDeref459, i32* %tmpWithVal463
  %asChar464 = bitcast i32* %tmpWithVal463 to i8*
  %set_at465 = call %list* (%list*, i32, i8*, ...) @set_at(%list* %text_redirect453, i32 %string_redirect_length454, i8* %asChar464)
  %string_redirect_length466 = load i32, i32* %string_redirect_length
  %tmp467 = add i32 %string_redirect_length466, 1
  store i32 %tmp467, i32* %string_redirect_length
  br label %while444

merge471:                                         ; preds = %while444
  %text_redirect472 = load %list*, %list** %text_redirect1
  ret %list* %text_redirect472
}
```

```

define %list* @encryption_decryption(%list* %rotor_one_e_input, %list* %rotor_one_e_output, %list* %rotor_two_e_input, %list* %rotor_two_e_output, %list* %rotor_three_e_input, %list* %rotor_three_e_output, %list* %text, i32 %length_input_text) {
entry:
  %rotor_one_e_input1 = alloca %list*
  store %list* %rotor_one_e_input, %list** %rotor_one_e_input1
  %rotor_one_e_output2 = alloca %list*
  store %list* %rotor_one_e_output, %list** %rotor_one_e_output2
  %rotor_two_e_input3 = alloca %list*
  store %list* %rotor_two_e_input, %list** %rotor_two_e_input3
  %rotor_two_e_output4 = alloca %list*
  store %list* %rotor_two_e_output, %list** %rotor_two_e_output4
  %rotor_three_e_input5 = alloca %list*
  store %list* %rotor_three_e_input, %list** %rotor_three_e_input5
  %rotor_three_e_output6 = alloca %list*
  store %list* %rotor_three_e_output, %list** %rotor_three_e_output6
  %text7 = alloca %list*
  store %list* %text, %list** %text7
  %length_input_text8 = alloca i32
  store i32 %length_input_text, i32* %length_input_text8
  %modulus_result = alloca i32
  %modulus_result_t = alloca i32
  %char_loc_holder = alloca i32
  %count_rotor_one = alloca i32
  %count_rotor_two = alloca i32
  %count_rotor_three = alloca i32
  %text_len = alloca i32
  %loop_counter_e_d = alloca i32
  %input_output_char = alloca %list*
  %cipher_output_character = alloca %list*
  store i32 0, i32* %modulus_result
  store i32 0, i32* %modulus_result_t
  store i32 0, i32* %char_loc_holder
  store i32 0, i32* %count_rotor_one
  store i32 0, i32* %count_rotor_two
  store i32 0, i32* %count_rotor_three
  %length_input_text9 = load i32, i32* %length_input_text8
  store i32 %length_input_text9, i32* %text_len
  store i32 0, i32* %loop_counter_e_d
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr = bitcast i8* %malloccall to %list*
  %init_list = call i32 (%list*, ...) @init_list(%list* %listAddr)
  %tmp = alloca i32*
  %malloccall10 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr = bitcast i8* %malloccall10 to i32*
  store i32* %tmpAddr, i32** %tmp
  %tmpWithVal = load i32*, i32** %tmp
  store i32 0, i32* %tmpWithVal
  %asChar = bitcast i32* %tmpWithVal to i8*
  %tmp11 = alloca i32*
  %malloccall12 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr13 = bitcast i8* %malloccall12 to i32*
  store i32* %tmpAddr13, i32** %tmp11
  %tmpWithVal14 = load i32*, i32** %tmp11
  store i32 1, i32* %tmpWithVal14
  %asChar15 = bitcast i32* %tmpWithVal14 to i8*
  %tmp16 = alloca i32*
  %malloccall17 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr18 = bitcast i8* %malloccall17 to i32*
  store i32* %tmpAddr18, i32** %tmp16
  %tmpWithVal19 = load i32*, i32** %tmp16
  store i32 2, i32* %tmpWithVal19
  %asChar20 = bitcast i32* %tmpWithVal19 to i8*
  %tmp21 = alloca i32*
  %malloccall22 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr23 = bitcast i8* %malloccall22 to i32*
  store i32* %tmpAddr23, i32** %tmp21
  %tmpWithVal24 = load i32*, i32** %tmp21
  store i32 3, i32* %tmpWithVal24
  %asChar25 = bitcast i32* %tmpWithVal24 to i8*
  %tmp26 = alloca i32*
  %malloccall27 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr28 = bitcast i8* %malloccall27 to i32*
  store i32* %tmpAddr28, i32** %tmp26
  %tmpWithVal29 = load i32*, i32** %tmp26
  store i32 4, i32* %tmpWithVal29
  %asChar30 = bitcast i32* %tmpWithVal29 to i8*
  %tmp31 = alloca i32*
  %malloccall32 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr33 = bitcast i8* %malloccall32 to i32*
  store i32* %tmpAddr33, i32** %tmp31
  %tmpWithVal34 = load i32*, i32** %tmp31
  store i32 5, i32* %tmpWithVal34
  %asChar35 = bitcast i32* %tmpWithVal34 to i8*
  %tmp36 = alloca i32*
  %malloccall37 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr38 = bitcast i8* %malloccall37 to i32*
  store i32* %tmpAddr38, i32** %tmp36
  %tmpWithVal39 = load i32*, i32** %tmp36
  store i32 6, i32* %tmpWithVal39
  %asChar40 = bitcast i32* %tmpWithVal39 to i8*
  %tmp41 = alloca i32*
  %malloccall42 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr43 = bitcast i8* %malloccall42 to i32*
  store i32* %tmpAddr43, i32** %tmp41
  %tmpWithVal44 = load i32*, i32** %tmp41
  store i32 7, i32* %tmpWithVal44
  %asChar45 = bitcast i32* %tmpWithVal44 to i8*
  %tmp46 = alloca i32*
  %malloccall47 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr48 = bitcast i8* %malloccall47 to i32*
  store i32* %tmpAddr48, i32** %tmp46
  %tmpWithVal49 = load i32*, i32** %tmp46
  store i32 8, i32* %tmpWithVal49
  %asChar50 = bitcast i32* %tmpWithVal49 to i8*
  %tmp51 = alloca i32*
  %malloccall52 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr53 = bitcast i8* %malloccall52 to i32*
  store i32* %tmpAddr53, i32** %tmp51
  %tmpWithVal54 = load i32*, i32** %tmp51
  store i32 9, i32* %tmpWithVal54
  %asChar55 = bitcast i32* %tmpWithVal54 to i8*
  %tmp56 = alloca i32*
  %malloccall57 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr58 = bitcast i8* %malloccall57 to i32*
  store i32* %tmpAddr58, i32** %tmp56
  %tmpWithVal59 = load i32*, i32** %tmp56
  store i32 10, i32* %tmpWithVal59
```

```
%asChar60 = bitcast i32* %tmpWithVal59 to i8*
%tmp61 = alloca i32*
%malloccall62 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr63 = bitcast i8* %malloccall62 to i32*
store i32* %tmpAddr63, i32** %tmp61
%tmpWithVal64 = load i32*, i32** %tmp61
store i32 11, i32* %tmpWithVal64
%asChar65 = bitcast i32* %tmpWithVal64 to i8*
%tmp66 = alloca i32*
%malloccall67 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr68 = bitcast i8* %malloccall67 to i32*
store i32* %tmpAddr68, i32** %tmp66
%tmpWithVal69 = load i32*, i32** %tmp66
store i32 12, i32* %tmpWithVal69
%asChar70 = bitcast i32* %tmpWithVal69 to i8*
%tmp71 = alloca i32*
%malloccall72 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr73 = bitcast i8* %malloccall72 to i32*
store i32* %tmpAddr73, i32** %tmp71
%tmpWithVal74 = load i32*, i32** %tmp71
store i32 13, i32* %tmpWithVal74
%asChar75 = bitcast i32* %tmpWithVal74 to i8*
%tmp76 = alloca i32*
%malloccall77 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr78 = bitcast i8* %malloccall77 to i32*
store i32* %tmpAddr78, i32** %tmp76
%tmpWithVal79 = load i32*, i32** %tmp76
store i32 14, i32* %tmpWithVal79
%asChar80 = bitcast i32* %tmpWithVal79 to i8*
%tmp81 = alloca i32*
%malloccall82 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr83 = bitcast i8* %malloccall82 to i32*
store i32* %tmpAddr83, i32** %tmp81
%tmpWithVal84 = load i32*, i32** %tmp81
store i32 15, i32* %tmpWithVal84
%asChar85 = bitcast i32* %tmpWithVal84 to i8*
%tmp86 = alloca i32*
%malloccall87 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
```

```
    %tmpAddr88 = bitcast i8* %malloccall87 to i32*
    store i32* %tmpAddr88, i32** %tmp86
    %tmpWithVal89 = load i32*, i32** %tmp86
    store i32 16, i32* %tmpWithVal89
    %asChar90 = bitcast i32* %tmpWithVal89 to i8*
    %tmp91 = alloca i32*
    %malloccall92 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr93 = bitcast i8* %malloccall92 to i32*
    store i32* %tmpAddr93, i32** %tmp91
    %tmpWithVal94 = load i32*, i32** %tmp91
    store i32 17, i32* %tmpWithVal94
    %asChar95 = bitcast i32* %tmpWithVal94 to i8*
    %tmp96 = alloca i32*
    %malloccall97 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr98 = bitcast i8* %malloccall97 to i32*
    store i32* %tmpAddr98, i32** %tmp96
    %tmpWithVal99 = load i32*, i32** %tmp96
    store i32 18, i32* %tmpWithVal99
    %asChar100 = bitcast i32* %tmpWithVal99 to i8*
    %tmp101 = alloca i32*
    %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr103 = bitcast i8* %malloccall102 to i32*
    store i32* %tmpAddr103, i32** %tmp101
    %tmpWithVal104 = load i32*, i32** %tmp101
    store i32 19, i32* %tmpWithVal104
    %asChar105 = bitcast i32* %tmpWithVal104 to i8*
    %tmp106 = alloca i32*
    %malloccall107 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr108 = bitcast i8* %malloccall107 to i32*
    store i32* %tmpAddr108, i32** %tmp106
    %tmpWithVal109 = load i32*, i32** %tmp106
    store i32 20, i32* %tmpWithVal109
    %asChar110 = bitcast i32* %tmpWithVal109 to i8*
    %tmp111 = alloca i32*
    %malloccall112 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr113 = bitcast i8* %malloccall112 to i32*
    store i32* %tmpAddr113, i32** %tmp111
    %tmpWithVal114 = load i32*, i32** %tmp111
    store i32 21, i32* %tmpWithVal114
```

```
%asChar115 = bitcast i32* %tmpWithVal114 to i8*
%tmp116 = alloca i32*
%malloccall117 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr118 = bitcast i8* %malloccall117 to i32*
store i32* %tmpAddr118, i32** %tmp116
%tmpWithVal119 = load i32*, i32** %tmp116
store i32 22, i32* %tmpWithVal119
%asChar120 = bitcast i32* %tmpWithVal119 to i8*
%tmp121 = alloca i32*
%malloccall122 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr123 = bitcast i8* %malloccall122 to i32*
store i32* %tmpAddr123, i32** %tmp121
%tmpWithVal124 = load i32*, i32** %tmp121
store i32 23, i32* %tmpWithVal124
%asChar125 = bitcast i32* %tmpWithVal124 to i8*
%tmp126 = alloca i32*
%malloccall127 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr128 = bitcast i8* %malloccall127 to i32*
store i32* %tmpAddr128, i32** %tmp126
%tmpWithVal129 = load i32*, i32** %tmp126
store i32 24, i32* %tmpWithVal129
%asChar130 = bitcast i32* %tmpWithVal129 to i8*
%tmp131 = alloca i32*
%malloccall132 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr133 = bitcast i8* %malloccall132 to i32*
store i32* %tmpAddr133, i32** %tmp131
%tmpWithVal134 = load i32*, i32** %tmp131
store i32 25, i32* %tmpWithVal134
%asChar135 = bitcast i32* %tmpWithVal134 to i8*
%push_back = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar)
%push_back136 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar15)
%push_back137 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar20)
%push_back138 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar25)
%push_back139 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar30)
```

```
%push_back140 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar35)
%push_back141 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar40)
%push_back142 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar45)
%push_back143 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar50)
%push_back144 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar55)
%push_back145 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar60)
%push_back146 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar65)
%push_back147 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar70)
%push_back148 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar75)
%push_back149 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar80)
%push_back150 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar85)
%push_back151 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar90)
%push_back152 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar95)
%push_back153 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar100)
%push_back154 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar105)
%push_back155 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar110)
%push_back156 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar115)
%push_back157 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar120)
%push_back158 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar125)
%push_back159 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar130)
%push_back160 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar135)
store %list* %listAddr, %list** %input_output_char
```

```
%malloccall161 = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
%listAddr162 = bitcast i8* %malloccall161 to %list*
%init_list163 = call i32 (%list*, ...) @init_list(%list* %listAddr162)
%tmp164 = alloca i32*
%malloccall165 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr166 = bitcast i8* %malloccall165 to i32*
store i32* %tmpAddr166, i32** %tmp164
%tmpWithVal167 = load i32*, i32** %tmp164
store i32 1, i32* %tmpWithVal167
%asChar168 = bitcast i32* %tmpWithVal167 to i8*
%tmp169 = alloca i32*
%malloccall170 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr171 = bitcast i8* %malloccall170 to i32*
store i32* %tmpAddr171, i32** %tmp169
%tmpWithVal172 = load i32*, i32** %tmp169
store i32 1, i32* %tmpWithVal172
%asChar173 = bitcast i32* %tmpWithVal172 to i8*
%tmp174 = alloca i32*
%malloccall175 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr176 = bitcast i8* %malloccall175 to i32*
store i32* %tmpAddr176, i32** %tmp174
%tmpWithVal177 = load i32*, i32** %tmp174
store i32 1, i32* %tmpWithVal177
%asChar178 = bitcast i32* %tmpWithVal177 to i8*
%tmp179 = alloca i32*
%malloccall180 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr181 = bitcast i8* %malloccall180 to i32*
store i32* %tmpAddr181, i32** %tmp179
%tmpWithVal182 = load i32*, i32** %tmp179
store i32 1, i32* %tmpWithVal182
%asChar183 = bitcast i32* %tmpWithVal182 to i8*
%tmp184 = alloca i32*
%malloccall185 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr186 = bitcast i8* %malloccall185 to i32*
store i32* %tmpAddr186, i32** %tmp184
%tmpWithVal187 = load i32*, i32** %tmp184
store i32 1, i32* %tmpWithVal187
%asChar188 = bitcast i32* %tmpWithVal187 to i8*
```

```
  %tmp189 = alloca i32*
  %malloccall190 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr191 = bitcast i8* %malloccall190 to i32*
  store i32* %tmpAddr191, i32** %tmp189
  %tmpWithVal192 = load i32*, i32** %tmp189
  store i32 1, i32* %tmpWithVal192
  %asChar193 = bitcast i32* %tmpWithVal192 to i8*
  %tmp194 = alloca i32*
  %malloccall195 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr196 = bitcast i8* %malloccall195 to i32*
  store i32* %tmpAddr196, i32** %tmp194
  %tmpWithVal197 = load i32*, i32** %tmp194
  store i32 1, i32* %tmpWithVal197
  %asChar198 = bitcast i32* %tmpWithVal197 to i8*
  %tmp199 = alloca i32*
  %malloccall200 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr201 = bitcast i8* %malloccall200 to i32*
  store i32* %tmpAddr201, i32** %tmp199
  %tmpWithVal202 = load i32*, i32** %tmp199
  store i32 1, i32* %tmpWithVal202
  %asChar203 = bitcast i32* %tmpWithVal202 to i8*
  %tmp204 = alloca i32*
  %malloccall205 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr206 = bitcast i8* %malloccall205 to i32*
  store i32* %tmpAddr206, i32** %tmp204
  %tmpWithVal207 = load i32*, i32** %tmp204
  store i32 1, i32* %tmpWithVal207
  %asChar208 = bitcast i32* %tmpWithVal207 to i8*
  %tmp209 = alloca i32*
  %malloccall210 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr211 = bitcast i8* %malloccall210 to i32*
  store i32* %tmpAddr211, i32** %tmp209
  %tmpWithVal212 = load i32*, i32** %tmp209
  store i32 1, i32* %tmpWithVal212
  %asChar213 = bitcast i32* %tmpWithVal212 to i8*
  %tmp214 = alloca i32*
  %malloccall215 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr216 = bitcast i8* %malloccall215 to i32*
  store i32* %tmpAddr216, i32** %tmp214
  %tmpWithVal217 = load i32*, i32** %tmp214
  store i32 1, i32* %tmpWithVal217
  %asChar218 = bitcast i32* %tmpWithVal217 to i8*
  %tmp219 = alloca i32*
  %malloccall220 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr221 = bitcast i8* %malloccall220 to i32*
  store i32* %tmpAddr221, i32** %tmp219
  %tmpWithVal222 = load i32*, i32** %tmp219
  store i32 1, i32* %tmpWithVal222
  %asChar223 = bitcast i32* %tmpWithVal222 to i8*
  %tmp224 = alloca i32*
  %malloccall225 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr226 = bitcast i8* %malloccall225 to i32*
  store i32* %tmpAddr226, i32** %tmp224
  %tmpWithVal227 = load i32*, i32** %tmp224
  store i32 1, i32* %tmpWithVal227
  %asChar228 = bitcast i32* %tmpWithVal227 to i8*
  %tmp229 = alloca i32*
  %malloccall230 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr231 = bitcast i8* %malloccall230 to i32*
  store i32* %tmpAddr231, i32** %tmp229
  %tmpWithVal232 = load i32*, i32** %tmp229
  store i32 1, i32* %tmpWithVal232
  %asChar233 = bitcast i32* %tmpWithVal232 to i8*
  %tmp234 = alloca i32*
  %malloccall235 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr236 = bitcast i8* %malloccall235 to i32*
  store i32* %tmpAddr236, i32** %tmp234
  %tmpWithVal237 = load i32*, i32** %tmp234
  store i32 1, i32* %tmpWithVal237
  %asChar238 = bitcast i32* %tmpWithVal237 to i8*
  %tmp239 = alloca i32*
  %malloccall240 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr241 = bitcast i8* %malloccall240 to i32*
  store i32* %tmpAddr241, i32** %tmp239
  %tmpWithVal242 = load i32*, i32** %tmp239
  store i32 1, i32* %tmpWithVal242
  %asChar243 = bitcast i32* %tmpWithVal242 to i8*
```

```
%tmp244 = alloca i32*
%malloccall245 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr246 = bitcast i8* %malloccall245 to i32*
store i32* %tmpAddr246, i32** %tmp244
%tmpWithVal247 = load i32*, i32** %tmp244
store i32 1, i32* %tmpWithVal247
%asChar248 = bitcast i32* %tmpWithVal247 to i8*
%tmp249 = alloca i32*
%malloccall250 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr251 = bitcast i8* %malloccall250 to i32*
store i32* %tmpAddr251, i32** %tmp249
%tmpWithVal252 = load i32*, i32** %tmp249
store i32 1, i32* %tmpWithVal252
%asChar253 = bitcast i32* %tmpWithVal252 to i8*
%tmp254 = alloca i32*
%malloccall255 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr256 = bitcast i8* %malloccall255 to i32*
store i32* %tmpAddr256, i32** %tmp254
%tmpWithVal257 = load i32*, i32** %tmp254
store i32 1, i32* %tmpWithVal257
%asChar258 = bitcast i32* %tmpWithVal257 to i8*
%tmp259 = alloca i32*
%malloccall260 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr261 = bitcast i8* %malloccall260 to i32*
store i32* %tmpAddr261, i32** %tmp259
%tmpWithVal262 = load i32*, i32** %tmp259
store i32 1, i32* %tmpWithVal262
%asChar263 = bitcast i32* %tmpWithVal262 to i8*
%tmp264 = alloca i32*
%malloccall265 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr266 = bitcast i8* %malloccall265 to i32*
store i32* %tmpAddr266, i32** %tmp264
%tmpWithVal267 = load i32*, i32** %tmp264
store i32 1, i32* %tmpWithVal267
%asChar268 = bitcast i32* %tmpWithVal267 to i8*
%tmp269 = alloca i32*
%malloccall270 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr271 = bitcast i8* %malloccall270 to i32*
```

```
store i32* %tmpAddr271, i32** %tmp269
%tmpWithVal272 = load i32*, i32** %tmp269
store i32 1, i32* %tmpWithVal272
%asChar273 = bitcast i32* %tmpWithVal272 to i8*
%tmp274 = alloca i32*
%malloccall275 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr276 = bitcast i8* %malloccall275 to i32*
store i32* %tmpAddr276, i32** %tmp274
%tmpWithVal277 = load i32*, i32** %tmp274
store i32 1, i32* %tmpWithVal277
%asChar278 = bitcast i32* %tmpWithVal277 to i8*
%tmp279 = alloca i32*
%malloccall280 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr281 = bitcast i8* %malloccall280 to i32*
store i32* %tmpAddr281, i32** %tmp279
%tmpWithVal282 = load i32*, i32** %tmp279
store i32 1, i32* %tmpWithVal282
%asChar283 = bitcast i32* %tmpWithVal282 to i8*
%tmp284 = alloca i32*
%malloccall285 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr286 = bitcast i8* %malloccall285 to i32*
store i32* %tmpAddr286, i32** %tmp284
%tmpWithVal287 = load i32*, i32** %tmp284
store i32 1, i32* %tmpWithVal287
%asChar288 = bitcast i32* %tmpWithVal287 to i8*
%tmp289 = alloca i32*
%malloccall290 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr291 = bitcast i8* %malloccall290 to i32*
store i32* %tmpAddr291, i32** %tmp289
%tmpWithVal292 = load i32*, i32** %tmp289
store i32 1, i32* %tmpWithVal292
%asChar293 = bitcast i32* %tmpWithVal292 to i8*
%tmp294 = alloca i32*
%malloccall295 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr296 = bitcast i8* %malloccall295 to i32*
store i32* %tmpAddr296, i32** %tmp294
%tmpWithVal297 = load i32*, i32** %tmp294
store i32 1, i32* %tmpWithVal297
%asChar298 = bitcast i32* %tmpWithVal297 to i8*
```

```
  %tmp299 = alloca i32*
  %malloccall300 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr301 = bitcast i8* %malloccall300 to i32*
  store i32* %tmpAddr301, i32** %tmp299
  %tmpWithVal302 = load i32*, i32** %tmp299
  store i32 1, i32* %tmpWithVal302
  %asChar303 = bitcast i32* %tmpWithVal302 to i8*
  %tmp304 = alloca i32*
  %malloccall305 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr306 = bitcast i8* %malloccall305 to i32*
  store i32* %tmpAddr306, i32** %tmp304
  %tmpWithVal307 = load i32*, i32** %tmp304
  store i32 1, i32* %tmpWithVal307
  %asChar308 = bitcast i32* %tmpWithVal307 to i8*
  %tmp309 = alloca i32*
  %malloccall310 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr311 = bitcast i8* %malloccall310 to i32*
  store i32* %tmpAddr311, i32** %tmp309
  %tmpWithVal312 = load i32*, i32** %tmp309
  store i32 1, i32* %tmpWithVal312
  %asChar313 = bitcast i32* %tmpWithVal312 to i8*
  %tmp314 = alloca i32*
  %malloccall315 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr316 = bitcast i8* %malloccall315 to i32*
  store i32* %tmpAddr316, i32** %tmp314
  %tmpWithVal317 = load i32*, i32** %tmp314
  store i32 1, i32* %tmpWithVal317
  %asChar318 = bitcast i32* %tmpWithVal317 to i8*
  %tmp319 = alloca i32*
  %malloccall320 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr321 = bitcast i8* %malloccall320 to i32*
  store i32* %tmpAddr321, i32** %tmp319
  %tmpWithVal322 = load i32*, i32** %tmp319
  store i32 1, i32* %tmpWithVal322
  %asChar323 = bitcast i32* %tmpWithVal322 to i8*
  %tmp324 = alloca i32*
  %malloccall325 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr326 = bitcast i8* %malloccall325 to i32*
```

```
store i32* %tmpAddr326, i32** %tmp324
%tmpWithVal327 = load i32*, i32** %tmp324
store i32 1, i32* %tmpWithVal327
%asChar328 = bitcast i32* %tmpWithVal327 to i8*
%tmp329 = alloca i32*
%malloccall330 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr331 = bitcast i8* %malloccall330 to i32*
store i32* %tmpAddr331, i32** %tmp329
%tmpWithVal332 = load i32*, i32** %tmp329
store i32 1, i32* %tmpWithVal332
%asChar333 = bitcast i32* %tmpWithVal332 to i8*
%tmp334 = alloca i32*
%malloccall335 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr336 = bitcast i8* %malloccall335 to i32*
store i32* %tmpAddr336, i32** %tmp334
%tmpWithVal337 = load i32*, i32** %tmp334
store i32 1, i32* %tmpWithVal337
%asChar338 = bitcast i32* %tmpWithVal337 to i8*
%tmp339 = alloca i32*
%malloccall340 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr341 = bitcast i8* %malloccall340 to i32*
store i32* %tmpAddr341, i32** %tmp339
%tmpWithVal342 = load i32*, i32** %tmp339
store i32 1, i32* %tmpWithVal342
%asChar343 = bitcast i32* %tmpWithVal342 to i8*
%tmp344 = alloca i32*
%malloccall345 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr346 = bitcast i8* %malloccall345 to i32*
store i32* %tmpAddr346, i32** %tmp344
%tmpWithVal347 = load i32*, i32** %tmp344
store i32 1, i32* %tmpWithVal347
%asChar348 = bitcast i32* %tmpWithVal347 to i8*
%tmp349 = alloca i32*
%malloccall350 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr351 = bitcast i8* %malloccall350 to i32*
store i32* %tmpAddr351, i32** %tmp349
%tmpWithVal352 = load i32*, i32** %tmp349
store i32 1, i32* %tmpWithVal352
%asChar353 = bitcast i32* %tmpWithVal352 to i8*
```

```
%tmp354 = alloca i32*
%malloccall355 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr356 = bitcast i8* %malloccall355 to i32*
store i32* %tmpAddr356, i32** %tmp354
%tmpWithVal357 = load i32*, i32** %tmp354
store i32 1, i32* %tmpWithVal357
%asChar358 = bitcast i32* %tmpWithVal357 to i8*
%tmp359 = alloca i32*
%malloccall360 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr361 = bitcast i8* %malloccall360 to i32*
store i32* %tmpAddr361, i32** %tmp359
%tmpWithVal362 = load i32*, i32** %tmp359
store i32 1, i32* %tmpWithVal362
%asChar363 = bitcast i32* %tmpWithVal362 to i8*
%tmp364 = alloca i32*
%malloccall365 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr366 = bitcast i8* %malloccall365 to i32*
store i32* %tmpAddr366, i32** %tmp364
%tmpWithVal367 = load i32*, i32** %tmp364
store i32 1, i32* %tmpWithVal367
%asChar368 = bitcast i32* %tmpWithVal367 to i8*
%tmp369 = alloca i32*
%malloccall370 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr371 = bitcast i8* %malloccall370 to i32*
store i32* %tmpAddr371, i32** %tmp369
%tmpWithVal372 = load i32*, i32** %tmp369
store i32 1, i32* %tmpWithVal372
%asChar373 = bitcast i32* %tmpWithVal372 to i8*
%tmp374 = alloca i32*
%malloccall375 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr376 = bitcast i8* %malloccall375 to i32*
store i32* %tmpAddr376, i32** %tmp374
%tmpWithVal377 = load i32*, i32** %tmp374
store i32 1, i32* %tmpWithVal377
%asChar378 = bitcast i32* %tmpWithVal377 to i8*
%tmp379 = alloca i32*
%malloccall380 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr381 = bitcast i8* %malloccall380 to i32*
```

```
store i32* %tmpAddr381, i32** %tmp379
%tmpWithVal382 = load i32*, i32** %tmp379
store i32 1, i32* %tmpWithVal382
%asChar383 = bitcast i32* %tmpWithVal382 to i8*
%tmp384 = alloca i32*
%malloccall385 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr386 = bitcast i8* %malloccall385 to i32*
store i32* %tmpAddr386, i32** %tmp384
%tmpWithVal387 = load i32*, i32** %tmp384
store i32 1, i32* %tmpWithVal387
%asChar388 = bitcast i32* %tmpWithVal387 to i8*
%tmp389 = alloca i32*
%malloccall390 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr391 = bitcast i8* %malloccall390 to i32*
store i32* %tmpAddr391, i32** %tmp389
%tmpWithVal392 = load i32*, i32** %tmp389
store i32 1, i32* %tmpWithVal392
%asChar393 = bitcast i32* %tmpWithVal392 to i8*
%tmp394 = alloca i32*
%malloccall395 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr396 = bitcast i8* %malloccall395 to i32*
store i32* %tmpAddr396, i32** %tmp394
%tmpWithVal397 = load i32*, i32** %tmp394
store i32 1, i32* %tmpWithVal397
%asChar398 = bitcast i32* %tmpWithVal397 to i8*
%tmp399 = alloca i32*
%malloccall400 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr401 = bitcast i8* %malloccall400 to i32*
store i32* %tmpAddr401, i32** %tmp399
%tmpWithVal402 = load i32*, i32** %tmp399
store i32 1, i32* %tmpWithVal402
%asChar403 = bitcast i32* %tmpWithVal402 to i8*
%tmp404 = alloca i32*
%malloccall405 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr406 = bitcast i8* %malloccall405 to i32*
store i32* %tmpAddr406, i32** %tmp404
%tmpWithVal407 = load i32*, i32** %tmp404
store i32 1, i32* %tmpWithVal407
%asChar408 = bitcast i32* %tmpWithVal407 to i8*
```

```
%tmp409 = alloca i32*
%malloccall410 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr411 = bitcast i8* %malloccall410 to i32*
store i32* %tmpAddr411, i32** %tmp409
%tmpWithVal412 = load i32*, i32** %tmp409
store i32 1, i32* %tmpWithVal412
%asChar413 = bitcast i32* %tmpWithVal412 to i8*
%tmp414 = alloca i32*
%malloccall415 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr416 = bitcast i8* %malloccall415 to i32*
store i32* %tmpAddr416, i32** %tmp414
%tmpWithVal417 = load i32*, i32** %tmp414
store i32 1, i32* %tmpWithVal417
%asChar418 = bitcast i32* %tmpWithVal417 to i8*
%tmp419 = alloca i32*
%malloccall420 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr421 = bitcast i8* %malloccall420 to i32*
store i32* %tmpAddr421, i32** %tmp419
%tmpWithVal422 = load i32*, i32** %tmp419
store i32 1, i32* %tmpWithVal422
%asChar423 = bitcast i32* %tmpWithVal422 to i8*
%tmp424 = alloca i32*
%malloccall425 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr426 = bitcast i8* %malloccall425 to i32*
store i32* %tmpAddr426, i32** %tmp424
%tmpWithVal427 = load i32*, i32** %tmp424
store i32 1, i32* %tmpWithVal427
%asChar428 = bitcast i32* %tmpWithVal427 to i8*
%tmp429 = alloca i32*
%malloccall430 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr431 = bitcast i8* %malloccall430 to i32*
store i32* %tmpAddr431, i32** %tmp429
%tmpWithVal432 = load i32*, i32** %tmp429
store i32 1, i32* %tmpWithVal432
%asChar433 = bitcast i32* %tmpWithVal432 to i8*
%tmp434 = alloca i32*
%malloccall435 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr436 = bitcast i8* %malloccall435 to i32*
```

```
store i32* %tmpAddr436, i32** %tmp434
%tmpWithVal437 = load i32*, i32** %tmp434
store i32 1, i32* %tmpWithVal437
%asChar438 = bitcast i32* %tmpWithVal437 to i8*
%tmp439 = alloca i32*
%malloccall440 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr441 = bitcast i8* %malloccall440 to i32*
store i32* %tmpAddr441, i32** %tmp439
%tmpWithVal442 = load i32*, i32** %tmp439
store i32 1, i32* %tmpWithVal442
%asChar443 = bitcast i32* %tmpWithVal442 to i8*
%tmp444 = alloca i32*
%malloccall445 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr446 = bitcast i8* %malloccall445 to i32*
store i32* %tmpAddr446, i32** %tmp444
%tmpWithVal447 = load i32*, i32** %tmp444
store i32 1, i32* %tmpWithVal447
%asChar448 = bitcast i32* %tmpWithVal447 to i8*
%tmp449 = alloca i32*
%malloccall450 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr451 = bitcast i8* %malloccall450 to i32*
store i32* %tmpAddr451, i32** %tmp449
%tmpWithVal452 = load i32*, i32** %tmp449
store i32 1, i32* %tmpWithVal452
%asChar453 = bitcast i32* %tmpWithVal452 to i8*
%tmp454 = alloca i32*
%malloccall455 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr456 = bitcast i8* %malloccall455 to i32*
store i32* %tmpAddr456, i32** %tmp454
%tmpWithVal457 = load i32*, i32** %tmp454
store i32 1, i32* %tmpWithVal457
%asChar458 = bitcast i32* %tmpWithVal457 to i8*
%tmp459 = alloca i32*
%malloccall460 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr461 = bitcast i8* %malloccall460 to i32*
store i32* %tmpAddr461, i32** %tmp459
%tmpWithVal462 = load i32*, i32** %tmp459
store i32 1, i32* %tmpWithVal462
%asChar463 = bitcast i32* %tmpWithVal462 to i8*
```

```
  %tmp464 = alloca i32*
  %malloccall465 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr466 = bitcast i8* %malloccall465 to i32*
  store i32* %tmpAddr466, i32** %tmp464
  %tmpWithVal467 = load i32*, i32** %tmp464
  store i32 1, i32* %tmpWithVal467
  %asChar468 = bitcast i32* %tmpWithVal467 to i8*
  %tmp469 = alloca i32*
  %malloccall470 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr471 = bitcast i8* %malloccall470 to i32*
  store i32* %tmpAddr471, i32** %tmp469
  %tmpWithVal472 = load i32*, i32** %tmp469
  store i32 1, i32* %tmpWithVal472
  %asChar473 = bitcast i32* %tmpWithVal472 to i8*
  %tmp474 = alloca i32*
  %malloccall475 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr476 = bitcast i8* %malloccall475 to i32*
  store i32* %tmpAddr476, i32** %tmp474
  %tmpWithVal477 = load i32*, i32** %tmp474
  store i32 1, i32* %tmpWithVal477
  %asChar478 = bitcast i32* %tmpWithVal477 to i8*
  %tmp479 = alloca i32*
  %malloccall480 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr481 = bitcast i8* %malloccall480 to i32*
  store i32* %tmpAddr481, i32** %tmp479
  %tmpWithVal482 = load i32*, i32** %tmp479
  store i32 1, i32* %tmpWithVal482
  %asChar483 = bitcast i32* %tmpWithVal482 to i8*
  %tmp484 = alloca i32*
  %malloccall485 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr486 = bitcast i8* %malloccall485 to i32*
  store i32* %tmpAddr486, i32** %tmp484
  %tmpWithVal487 = load i32*, i32** %tmp484
  store i32 1, i32* %tmpWithVal487
  %asChar488 = bitcast i32* %tmpWithVal487 to i8*
  %push_back489 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar168)
  %push_back490 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar173)
```

```
%push_back491 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar178)
%push_back492 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar183)
%push_back493 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar188)
%push_back494 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar193)
%push_back495 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar198)
%push_back496 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar203)
%push_back497 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar208)
%push_back498 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar213)
%push_back499 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar218)
%push_back500 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar223)
%push_back501 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar228)
%push_back502 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar233)
%push_back503 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar238)
%push_back504 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar243)
%push_back505 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar248)
%push_back506 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar253)
%push_back507 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar258)
%push_back508 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar263)
%push_back509 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar268)
%push_back510 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar273)
%push_back511 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar278)
%push_back512 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar283)
```

```
%push_back513 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar288)
%push_back514 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar293)
%push_back515 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar298)
%push_back516 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar303)
%push_back517 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar308)
%push_back518 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar313)
%push_back519 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar318)
%push_back520 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar323)
%push_back521 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar328)
%push_back522 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar333)
%push_back523 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar338)
%push_back524 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar343)
%push_back525 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar348)
%push_back526 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar353)
%push_back527 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar358)
%push_back528 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar363)
%push_back529 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar368)
%push_back530 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar373)
%push_back531 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar378)
%push_back532 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar383)
%push_back533 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar388)
%push_back534 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar393)
```

```
  %push_back535 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar398)
  %push_back536 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar403)
  %push_back537 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar408)
  %push_back538 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar413)
  %push_back539 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar418)
  %push_back540 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar423)
  %push_back541 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar428)
  %push_back542 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar433)
  %push_back543 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar438)
  %push_back544 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar443)
  %push_back545 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar448)
  %push_back546 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar453)
  %push_back547 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar458)
  %push_back548 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar463)
  %push_back549 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar468)
  %push_back550 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar473)
  %push_back551 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar478)
  %push_back552 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar483)
  %push_back553 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr162, i8* %asChar488)
  store %list* %listAddr162, %list** %cipher_output_character
  store i32 0, i32* %loop_counter_e_d
  br label %while

while:                                            ; preds = %merge, %entry
  %loop_counter_e_d615 = load i32, i32* %loop_counter_e_d
  %text_len616 = load i32, i32* %text_len
  %tmp617 = icmp slt i32 %loop_counter_e_d615, %text_len616
  br i1 %tmp617, label %while_body, label %merge618

while_body:                                       ; preds = %while
  %text554 = load %list*, %list** %text7
  %loop_counter_e_d555 = load i32, i32* %loop_counter_e_d
  %get_at = call i8* (%list*, i32, ...) @get_at(%list* %text554, i32 %loop_counter_e_d555)
  %eltAsPtr = bitcast i8* %get_at to i32*
  %eltDeref = load i32, i32* %eltAsPtr
  %input_output_char556 = load %list*, %list** %input_output_char
  %character_location_status_result = call i32 @character_location_status(%list* %input_output_char556, i32 %eltDeref)
  store i32 %character_location_status_result, i32* %char_loc_holder
  %rotor_three_e_input557 = load %list*, %list** %rotor_three_e_input5
  %rotor_shift_result = call %list* @rotor_shift(%list* %rotor_three_e_input557, i32 1)
  %rotor_three_e_output558 = load %list*, %list** %rotor_three_e_output6
  %rotor_shift_result559 = call %list* @rotor_shift(%list* %rotor_three_e_output558, i32 1)
  %count_rotor_three560 = load i32, i32* %count_rotor_three
  %tmp561 = add i32 %count_rotor_three560, 1
  store i32 %tmp561, i32* %count_rotor_three
  br label %if

if:                                               ; preds = %while_body
  %count_rotor_three592 = load i32, i32* %count_rotor_three
  %tmp593 = icmp sgt i32 %count_rotor_three592, 0
  br i1 %tmp593, label %if_body, label %merge

if_body:                                          ; preds = %if
  %count_rotor_three562 = load i32, i32* %count_rotor_three
  %my_modulus_result_result = call i32 @my_modulus_result(i32 %count_rotor_three562)
  store i32 %my_modulus_result_result, i32* %modulus_result
  br label %if563

merge:                                            ; preds = %if, %merge565
  %rotor_three_e_output594 = load %list*, %list** %rotor_three_e_output6
  %rotor_three_e_input595 = load %list*, %list** %rotor_three_e_input5
  %rotor_two_e_output596 = load %list*, %list** %rotor_two_e_output4
  %rotor_two_e_input597 = load %list*, %list** %rotor_two_e_input3
  %rotor_one_e_output598 = load %list*, %list** %rotor_one_e_output2
  %rotor_one_e_input599 = load %list*, %list** %rotor_one_e_input1
  %char_loc_holder600 = load i32, i32* %char_loc_holder
  %rotor_direction_output_result = call i32 @rotor_direction_output(i32 %char_loc_holder600, %list* %rotor_one_e_input599, %list* %rotor_one_e_output598, %list* %rotor_two_e_input597, %list* %rotor_two_e_output596, %list* %rotor_three_e_input595, %list* %rotor_three_e_output594)
  store i32 %rotor_direction_output_result, i32* %char_loc_holder
  %cipher_output_character601 = load %list*, %list** %cipher_output_character
  %loop_counter_e_d602 = load i32, i32* %loop_counter_e_d
  %input_output_char603 = load %list*, %list** %input_output_char
  %char_loc_holder604 = load i32, i32* %char_loc_holder
  %get_at605 = call i8* (%list*, i32, ...) @get_at(%list* %input_output_char603, i32 %char_loc_holder604)
  %eltAsPtr606 = bitcast i8* %get_at605 to i32*
  %eltDeref607 = load i32, i32* %eltAsPtr606
  %tmp608 = alloca i32*
  %malloccall609 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr610 = bitcast i8* %malloccall609 to i32*
  store i32* %tmpAddr610, i32** %tmp608
  %tmpWithVal611 = load i32*, i32** %tmp608
  store i32 %eltDeref607, i32* %tmpWithVal611
  %asChar612 = bitcast i32* %tmpWithVal611 to i8*
  %set_at = call %list* (%list*, i32, i8*, ...) @set_at(%list* %cipher_output_character601, i32 %loop_counter_e_d602, i8* %asChar612)
  %loop_counter_e_d613 = load i32, i32* %loop_counter_e_d
  %tmp614 = add i32 %loop_counter_e_d613, 1
  store i32 %tmp614, i32* %loop_counter_e_d
  br label %while

if563:                                            ; preds = %if_body
  %modulus_result590 = load i32, i32* %modulus_result
  %tmp591 = icmp eq i32 %modulus_result590, 0
  br i1 %tmp591, label %if_body564, label %merge565

if_body564:                                       ; preds = %if563
  %rotor_two_e_input566 = load %list*, %list** %rotor_two_e_input3
  %rotor_shift_result567 = call %list* @rotor_shift(%list* %rotor_two_e_input566, i32 1)
  %rotor_two_e_output568 = load %list*, %list** %rotor_two_e_output4
  %rotor_shift_result569 = call %list* @rotor_shift(%list* %rotor_two_e_output568, i32 1)
  %count_rotor_two570 = load i32, i32* %count_rotor_two
  %tmp571 = add i32 %count_rotor_two570, 1
  store i32 %tmp571, i32* %count_rotor_two
  br label %if572

merge565:                                         ; preds = %if563, %merge574
  br label %merge

if572:                                            ; preds = %if_body564
  %count_rotor_two588 = load i32, i32* %count_rotor_two
  %tmp589 = icmp sgt i32 %count_rotor_two588, 0
  br i1 %tmp589, label %if_body573, label %merge574

if_body573:                                       ; preds = %if572
  %count_rotor_two575 = load i32, i32* %count_rotor_two
  %my_modulus_result_result576 = call i32 @my_modulus_result(i32 %count_rotor_two575)
  store i32 %my_modulus_result_result576, i32* %modulus_result_t
  br label %if577

merge574:                                         ; preds = %if572, %merge579
  br label %merge565

if577:                                            ; preds = %if_body573
  %modulus_result_t586 = load i32, i32* %modulus_result_t
  %tmp587 = icmp eq i32 %modulus_result_t586, 0
  br i1 %tmp587, label %if_body578, label %merge579

if_body578:                                       ; preds = %if577
  %rotor_one_e_input580 = load %list*, %list** %rotor_one_e_input1
  %rotor_shift_result581 = call %list* @rotor_shift(%list* %rotor_one_e_input580, i32 1)
  %rotor_one_e_output582 = load %list*, %list** %rotor_one_e_output2
  %rotor_shift_result583 = call %list* @rotor_shift(%list* %rotor_one_e_output582, i32 1)
  %count_rotor_one584 = load i32, i32* %count_rotor_one
  %tmp585 = add i32 %count_rotor_one584, 1
  store i32 %tmp585, i32* %count_rotor_one
  br label %merge579

merge579:                                         ; preds = %if577, %if_body578
  br label %merge574
```

```

merge618:                                         ; preds = %while
  %cipher_output_character619 = load %list*, %list** %cipher_output_character
  ret %list* %cipher_output_character619
}

define i32 @rotor_direction_output(i32 %input_output_char_loc, %list* %rotor_one_input_direct, %list* %rotor_one_output_direct, %list* %rotor_two_input_direct, %list* %rotor_two_output_direct, %list* %rotor_three_input_direct, %list* %rotor_three_output_direct) {
entry:
  %input_output_char_loc1 = alloca i32
  store i32 %input_output_char_loc, i32* %input_output_char_loc1
  %rotor_one_input_direct2 = alloca %list*
  store %list* %rotor_one_input_direct, %list** %rotor_one_input_direct2
  %rotor_one_output_direct3 = alloca %list*
  store %list* %rotor_one_output_direct, %list** %rotor_one_output_direct3
  %rotor_two_input_direct4 = alloca %list*
  store %list* %rotor_two_input_direct, %list** %rotor_two_input_direct4
  %rotor_two_output_direct5 = alloca %list*
  store %list* %rotor_two_output_direct, %list** %rotor_two_output_direct5
  %rotor_three_input_direct6 = alloca %list*
  store %list* %rotor_three_input_direct, %list** %rotor_three_input_direct6
  %rotor_three_output_direct7 = alloca %list*
  store %list* %rotor_three_output_direct, %list** %rotor_three_output_direct7
  %index_reverse = alloca i32
  %characher_location_holder = alloca i32
  %characher_location_holder_two = alloca i32
  %reflector = alloca %list*
  store i32 0, i32* %index_reverse
  %input_output_char_loc8 = load i32, i32* %input_output_char_loc1
  store i32 %input_output_char_loc8, i32* %characher_location_holder
  store i32 0, i32* %characher_location_holder_two
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%list* getelementptr (%list, %list* null, i32 1) to i32))
  %listAddr = bitcast i8* %malloccall to %list*
  %init_list = call i32 (%list*, ...) @init_list(%list* %listAddr)
  %tmp = alloca i32*
  %malloccall9 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr = bitcast i8* %malloccall9 to i32*
  store i32* %tmpAddr, i32** %tmp
  %tmpWithVal = load i32*, i32** %tmp
  store i32 0, i32* %tmpWithVal
  %asChar = bitcast i32* %tmpWithVal to i8*
  %tmp10 = alloca i32*
  %malloccall11 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr12 = bitcast i8* %malloccall11 to i32*
  store i32* %tmpAddr12, i32** %tmp10
  %tmpWithVal13 = load i32*, i32** %tmp10
  store i32 1, i32* %tmpWithVal13
  %asChar14 = bitcast i32* %tmpWithVal13 to i8*
  %tmp15 = alloca i32*
  %malloccall16 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr17 = bitcast i8* %malloccall16 to i32*
  store i32* %tmpAddr17, i32** %tmp15
  %tmpWithVal18 = load i32*, i32** %tmp15
  store i32 2, i32* %tmpWithVal18
  %asChar19 = bitcast i32* %tmpWithVal18 to i8*
  %tmp20 = alloca i32*
  %malloccall21 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr22 = bitcast i8* %malloccall21 to i32*
  store i32* %tmpAddr22, i32** %tmp20
  %tmpWithVal23 = load i32*, i32** %tmp20
  store i32 3, i32* %tmpWithVal23
  %asChar24 = bitcast i32* %tmpWithVal23 to i8*
  %tmp25 = alloca i32*
  %malloccall26 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr27 = bitcast i8* %malloccall26 to i32*
  store i32* %tmpAddr27, i32** %tmp25
  %tmpWithVal28 = load i32*, i32** %tmp25
  store i32 4, i32* %tmpWithVal28
  %asChar29 = bitcast i32* %tmpWithVal28 to i8*
  %tmp30 = alloca i32*
  %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr32 = bitcast i8* %malloccall31 to i32*
  store i32* %tmpAddr32, i32** %tmp30
```

```
%tmpWithVal33 = load i32*, i32** %tmp30
store i32 5, i32* %tmpWithVal33
%asChar34 = bitcast i32* %tmpWithVal33 to i8*
%tmp35 = alloca i32*
%malloccall36 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr37 = bitcast i8* %malloccall36 to i32*
store i32* %tmpAddr37, i32** %tmp35
%tmpWithVal38 = load i32*, i32** %tmp35
store i32 6, i32* %tmpWithVal38
%asChar39 = bitcast i32* %tmpWithVal38 to i8*
%tmp40 = alloca i32*
%malloccall41 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr42 = bitcast i8* %malloccall41 to i32*
store i32* %tmpAddr42, i32** %tmp40
%tmpWithVal43 = load i32*, i32** %tmp40
store i32 3, i32* %tmpWithVal43
%asChar44 = bitcast i32* %tmpWithVal43 to i8*
%tmp45 = alloca i32*
%malloccall46 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr47 = bitcast i8* %malloccall46 to i32*
store i32* %tmpAddr47, i32** %tmp45
%tmpWithVal48 = load i32*, i32** %tmp45
store i32 8, i32* %tmpWithVal48
%asChar49 = bitcast i32* %tmpWithVal48 to i8*
%tmp50 = alloca i32*
%malloccall51 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr52 = bitcast i8* %malloccall51 to i32*
store i32* %tmpAddr52, i32** %tmp50
%tmpWithVal53 = load i32*, i32** %tmp50
store i32 9, i32* %tmpWithVal53
%asChar54 = bitcast i32* %tmpWithVal53 to i8*
%tmp55 = alloca i32*
%malloccall56 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr57 = bitcast i8* %malloccall56 to i32*
store i32* %tmpAddr57, i32** %tmp55
%tmpWithVal58 = load i32*, i32** %tmp55
store i32 10, i32* %tmpWithVal58
%asChar59 = bitcast i32* %tmpWithVal58 to i8*
%tmp60 = alloca i32*
```

```
  %malloccall61 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr62 = bitcast i8* %malloccall61 to i32*
  store i32* %tmpAddr62, i32** %tmp60
  %tmpWithVal63 = load i32*, i32** %tmp60
  store i32 6, i32* %tmpWithVal63
  %asChar64 = bitcast i32* %tmpWithVal63 to i8*
  %tmp65 = alloca i32*
  %malloccall66 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr67 = bitcast i8* %malloccall66 to i32*
  store i32* %tmpAddr67, i32** %tmp65
  %tmpWithVal68 = load i32*, i32** %tmp65
  store i32 12, i32* %tmpWithVal68
  %asChar69 = bitcast i32* %tmpWithVal68 to i8*
  %tmp70 = alloca i32*
  %malloccall71 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr72 = bitcast i8* %malloccall71 to i32*
  store i32* %tmpAddr72, i32** %tmp70
  %tmpWithVal73 = load i32*, i32** %tmp70
  store i32 10, i32* %tmpWithVal73
  %asChar74 = bitcast i32* %tmpWithVal73 to i8*
  %tmp75 = alloca i32*
  %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr77 = bitcast i8* %malloccall76 to i32*
  store i32* %tmpAddr77, i32** %tmp75
  %tmpWithVal78 = load i32*, i32** %tmp75
  store i32 12, i32* %tmpWithVal78
  %asChar79 = bitcast i32* %tmpWithVal78 to i8*
  %tmp80 = alloca i32*
  %malloccall81 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr82 = bitcast i8* %malloccall81 to i32*
  store i32* %tmpAddr82, i32** %tmp80
  %tmpWithVal83 = load i32*, i32** %tmp80
  store i32 8, i32* %tmpWithVal83
  %asChar84 = bitcast i32* %tmpWithVal83 to i8*
  %tmp85 = alloca i32*
  %malloccall86 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr87 = bitcast i8* %malloccall86 to i32*
  store i32* %tmpAddr87, i32** %tmp85
```

```
%tmpWithVal88 = load i32*, i32** %tmp85
store i32 4, i32* %tmpWithVal88
%asChar89 = bitcast i32* %tmpWithVal88 to i8*
%tmp90 = alloca i32*
%malloccall91 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr92 = bitcast i8* %malloccall91 to i32*
store i32* %tmpAddr92, i32** %tmp90
%tmpWithVal93 = load i32*, i32** %tmp90
store i32 1, i32* %tmpWithVal93
%asChar94 = bitcast i32* %tmpWithVal93 to i8*
%tmp95 = alloca i32*
%malloccall96 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr97 = bitcast i8* %malloccall96 to i32*
store i32* %tmpAddr97, i32** %tmp95
%tmpWithVal98 = load i32*, i32** %tmp95
store i32 5, i32* %tmpWithVal98
%asChar99 = bitcast i32* %tmpWithVal98 to i8*
%tmp100 = alloca i32*
%malloccall101 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr102 = bitcast i8* %malloccall101 to i32*
store i32* %tmpAddr102, i32** %tmp100
%tmpWithVal103 = load i32*, i32** %tmp100
store i32 19, i32* %tmpWithVal103
%asChar104 = bitcast i32* %tmpWithVal103 to i8*
%tmp105 = alloca i32*
%malloccall106 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr107 = bitcast i8* %malloccall106 to i32*
store i32* %tmpAddr107, i32** %tmp105
%tmpWithVal108 = load i32*, i32** %tmp105
store i32 2, i32* %tmpWithVal108
%asChar109 = bitcast i32* %tmpWithVal108 to i8*
%tmp110 = alloca i32*
%malloccall111 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
%tmpAddr112 = bitcast i8* %malloccall111 to i32*
store i32* %tmpAddr112, i32** %tmp110
%tmpWithVal113 = load i32*, i32** %tmp110
store i32 21, i32* %tmpWithVal113
%asChar114 = bitcast i32* %tmpWithVal113 to i8*
%tmp115 = alloca i32*
```

```
    %malloccall116 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr117 = bitcast i8* %malloccall116 to i32*
    store i32* %tmpAddr117, i32** %tmp115
    %tmpWithVal118 = load i32*, i32** %tmp115
    store i32 21, i32* %tmpWithVal118
    %asChar119 = bitcast i32* %tmpWithVal118 to i8*
    %tmp120 = alloca i32*
    %malloccall121 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr122 = bitcast i8* %malloccall121 to i32*
    store i32* %tmpAddr122, i32** %tmp120
    %tmpWithVal123 = load i32*, i32** %tmp120
    store i32 9, i32* %tmpWithVal123
    %asChar124 = bitcast i32* %tmpWithVal123 to i8*
    %tmp125 = alloca i32*
    %malloccall126 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr127 = bitcast i8* %malloccall126 to i32*
    store i32* %tmpAddr127, i32** %tmp125
    %tmpWithVal128 = load i32*, i32** %tmp125
    store i32 0, i32* %tmpWithVal128
    %asChar129 = bitcast i32* %tmpWithVal128 to i8*
    %tmp130 = alloca i32*
    %malloccall131 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
    %tmpAddr132 = bitcast i8* %malloccall131 to i32*
    store i32* %tmpAddr132, i32** %tmp130
    %tmpWithVal133 = load i32*, i32** %tmp130
    store i32 19, i32* %tmpWithVal133
    %asChar134 = bitcast i32* %tmpWithVal133 to i8*
    %push_back = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar)
    %push_back135 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar14)
    %push_back136 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar19)
    %push_back137 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar24)
    %push_back138 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar29)
    %push_back139 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar34)
```

```
%push_back140 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar39)
%push_back141 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar44)
%push_back142 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar49)
%push_back143 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar54)
%push_back144 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar59)
%push_back145 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar64)
%push_back146 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar69)
%push_back147 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar74)
%push_back148 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar79)
%push_back149 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar84)
%push_back150 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar89)
%push_back151 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar94)
%push_back152 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar99)
%push_back153 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar104)
%push_back154 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar109)
%push_back155 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar114)
%push_back156 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar119)
%push_back157 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar124)
%push_back158 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar129)
%push_back159 = call i32 (%list*, i8*, ...) @push_back(%list* %listAddr, i8* %asChar134)
store %list* %listAddr, %list** %reflector
%rotor_three_output_direct160 = load %list*, %list** %rotor_three_output_direct7
```

```
  %characher_location_holder161 = load i32, i32* %characher_location_holder
  %get_at = call i8* (%list*, i32, ...) @get_at(%list* %rotor_three_output_direct160, i32 %characher_location_holder161)
  %eltAsPtr = bitcast i8* %get_at to i32*
  %eltDeref = load i32, i32* %eltAsPtr
  %rotor_three_input_direct162 = load %list*, %list** %rotor_three_input_direct6
  %character_location_result = call i32 @character_location(%list* %rotor_three_input_direct162, i32 %eltDeref)
  store i32 %character_location_result, i32* %characher_location_holder
  %rotor_two_output_direct163 = load %list*, %list** %rotor_two_output_direct5
  %characher_location_holder164 = load i32, i32* %characher_location_holder
  %get_at165 = call i8* (%list*, i32, ...) @get_at(%list* %rotor_two_output_direct163, i32 %characher_location_holder164)
  %eltAsPtr166 = bitcast i8* %get_at165 to i32*
  %eltDeref167 = load i32, i32* %eltAsPtr166
  %rotor_two_input_direct168 = load %list*, %list** %rotor_two_input_direct4
  %character_location_result169 = call i32 @character_location(%list* %rotor_two_input_direct168, i32 %eltDeref167)
  store i32 %character_location_result169, i32* %characher_location_holder
  %rotor_one_output_direct170 = load %list*, %list** %rotor_one_output_direct3
  %characher_location_holder171 = load i32, i32* %characher_location_holder
  %get_at172 = call i8* (%list*, i32, ...) @get_at(%list* %rotor_one_output_direct170, i32 %characher_location_holder171)
  %eltAsPtr173 = bitcast i8* %get_at172 to i32*
  %eltDeref174 = load i32, i32* %eltAsPtr173
  %rotor_one_input_direct175 = load %list*, %list** %rotor_one_input_direct2
  %character_location_result176 = call i32 @character_location(%list* %rotor_one_input_direct175, i32 %eltDeref174)
  store i32 %character_location_result176, i32* %characher_location_holder
  store i32 0, i32* %index_reverse
  br label %while

while:                                            ; preds = %merge, %entry
  %index_reverse197 = load i32, i32* %index_reverse
  %tmp198 = icmp slt i32 %index_reverse197, 26
  br i1 %tmp198, label %while_body, label %merge199

while_body:                                       ; preds = %while
  br label %if

if:                                               ; preds = %while_body
  %characher_location_holder192 = load i32, i32* %characher_location_holder
  %index_reverse193 = load i32, i32* %index_reverse
  %tmp194 = icmp eq i32 %characher_location_holder192, %index_reverse193
  br i1 %tmp194, label %if_body, label %else_body

if_body:                                          ; preds = %if
  br label %merge

else_body:                                        ; preds = %if
  br label %if177

merge:                                            ; preds = %merge179, %if_body
  %index_reverse195 = load i32, i32* %index_reverse
  %tmp196 = add i32 %index_reverse195, 1
  store i32 %tmp196, i32* %index_reverse
  br label %while

if177:                                            ; preds = %else_body
  %reflector181 = load %list*, %list** %reflector
  %characher_location_holder182 = load i32, i32* %characher_location_holder
  %get_at183 = call i8* (%list*, i32, ...) @get_at(%list* %reflector181, i32 %characher_location_holder182)
  %eltAsPtr184 = bitcast i8* %get_at183 to i32*
  %eltDeref185 = load i32, i32* %eltAsPtr184
  %reflector186 = load %list*, %list** %reflector
  %index_reverse187 = load i32, i32* %index_reverse
  %get_at188 = call i8* (%list*, i32, ...) @get_at(%list* %reflector186, i32 %index_reverse187)
  %eltAsPtr189 = bitcast i8* %get_at188 to i32*
  %eltDeref190 = load i32, i32* %eltAsPtr189
  %tmp191 = icmp eq i32 %eltDeref185, %eltDeref190
  br i1 %tmp191, label %if_body178, label %merge179

if_body178:                                       ; preds = %if177
```

```
  %index_reverse180 = load i32, i32* %index_reverse
  store i32 %index_reverse180, i32* %characher_location_holder_two
  br label %merge179

merge179:                                         ; preds = %if177, %if_body178
  br label %merge

merge199:                                         ; preds = %while
  %rotor_one_input_direct200 = load %list*, %list** %rotor_one_input_direct2
  %characher_location_holder_two201 = load i32, i32* %characher_location_holder_two
  %get_at202 = call i8* (%list*, i32, ...) @get_at(%list* %rotor_one_input_direct200, i32 %characher_location_holder_two201)
  %eltAsPtr203 = bitcast i8* %get_at202 to i32*
  %eltDeref204 = load i32, i32* %eltAsPtr203
  %rotor_one_output_direct205 = load %list*, %list** %rotor_one_output_direct3
  %character_location_result206 = call i32 @character_location(%list* %rotor_one_output_direct205, i32 %eltDeref204)
  store i32 %character_location_result206, i32* %characher_location_holder
  %rotor_two_input_direct207 = load %list*, %list** %rotor_two_input_direct4
  %characher_location_holder208 = load i32, i32* %characher_location_holder
  %get_at209 = call i8* (%list*, i32, ...) @get_at(%list* %rotor_two_input_direct207, i32 %characher_location_holder208)
  %eltAsPtr210 = bitcast i8* %get_at209 to i32*
  %eltDeref211 = load i32, i32* %eltAsPtr210
  %rotor_two_output_direct212 = load %list*, %list** %rotor_two_output_direct5
  %character_location_result213 = call i32 @character_location(%list* %rotor_two_output_direct212, i32 %eltDeref211)
  store i32 %character_location_result213, i32* %characher_location_holder
  %rotor_three_input_direct214 = load %list*, %list** %rotor_three_input_direct6
  %characher_location_holder215 = load i32, i32* %characher_location_holder
  %get_at216 = call i8* (%list*, i32, ...) @get_at(%list* %rotor_three_input_direct214, i32 %characher_location_holder215)
  %eltAsPtr217 = bitcast i8* %get_at216 to i32*
  %eltDeref218 = load i32, i32* %eltAsPtr217
  %rotor_three_output_direct219 = load %list*, %list** %rotor_three_output_direct7
  %character_location_result220 = call i32 @character_location(%list* %rotor_three_output_direct219, i32 %eltDeref218)
  store i32 %character_location_result220, i32* %characher_location_holder
  %characher_location_holder221 = load i32, i32* %characher_location_holder
  ret i32 %characher_location_holder221
}

define i32 @my_modulus_result(i32 %base) {
entry:
  %base1 = alloca i32
  store i32 %base, i32* %base1
  %modulus_results = alloca i32
  %base2 = load i32, i32* %base1
  %modulus_operation = call i32 (i32, i32, ...) @modulus_operation(i32 %base2, i32 26)
  store i32 %modulus_operation, i32* %modulus_results
  %modulus_results3 = load i32, i32* %modulus_results
  ret i32 %modulus_results3
}

define i32 @character_location_status(%list* %xx, i32 %yy) {
entry:
  %xx1 = alloca %list*
  store %list* %xx, %list** %xx1
  %yy2 = alloca i32
  store i32 %yy, i32* %yy2
  %my_response = alloca i32
  %yy3 = load i32, i32* %yy2
  %xx4 = load %list*, %list** %xx1
  %character_location_result = call i32 @character_location(%list* %xx4, i32 %yy3)
  store i32 %character_location_result, i32* %my_response
  %my_response5 = load i32, i32* %my_response
  ret i32 %my_response5
}

define %list* @initialize_input_text(%list* %message_user, %list* %input_list_message) {
entry:
  %message_user1 = alloca %list*
  store %list* %message_user, %list** %message_user1
  %input_list_message2 = alloca %list*
  store %list* %input_list_message, %list** %input_list_message2
  %message_length_input = alloca i32
  %message_length_init = alloca i32
  %current_int_val = alloca i32
  %message_user3 = load %list*, %list** %message_user1
  %stringLength = call i32 @stringLength(%list* %message_user3)
  store i32 %stringLength, i32* %message_length_input
  store i32 0, i32* %message_length_init
  store i32 0, i32* %current_int_val
  store i32 0, i32* %message_length_init
  br label %while

while:                                            ; preds = %while_body, %entry
  %message_length_init11 = load i32, i32* %message_length_init
  %message_length_input12 = load i32, i32* %message_length_input
  %tmp13 = icmp slt i32 %message_length_init11, %message_length_input12
  br i1 %tmp13, label %while_body, label %merge

while_body:                                       ; preds = %while
  %message_length_init4 = load i32, i32* %message_length_init
  %message_user5 = load %list*, %list** %message_user1
  %return_cor_int_result = call i32 @return_cor_int(%list* %message_user5, i32 %message_length_init4)
  store i32 %return_cor_int_result, i32* %current_int_val
  %input_list_message6 = load %list*, %list** %input_list_message2
  %message_length_init7 = load i32, i32* %message_length_init
  %current_int_val8 = load i32, i32* %current_int_val
  %tmp = alloca i32*
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr = bitcast i8* %malloccall to i32*
  store i32* %tmpAddr, i32** %tmp
  %tmpWithVal = load i32*, i32** %tmp
  store i32 %current_int_val8, i32* %tmpWithVal
  %asChar = bitcast i32* %tmpWithVal to i8*
  %set_at = call %list* (%list*, i32, i8*, ...) @set_at(%list* %input_list_message6, i32 %message_length_init7, i8* %asChar)
  %message_length_init9 = load i32, i32* %message_length_init
  %tmp10 = add i32 %message_length_init9, 1
  store i32 %tmp10, i32* %message_length_init
  br label %while
```

```

merge:                                            ; preds = %while
  %input_list_message14 = load %list*, %list** %input_list_message2
  ret %list* %input_list_message14
}

define i32 @return_cor_int(%list* %what_message, i32 %what_index) {
entry:
  %what_message1 = alloca %list*
  store %list* %what_message, %list** %what_message1
  %what_index2 = alloca i32
  store i32 %what_index, i32* %what_index2
  %res = alloca i32
  %what_index3 = load i32, i32* %what_index2
  %what_message4 = load %list*, %list** %what_message1
  %corresponding_int = call i32 @corresponding_int(%list*
%what_message4, i32 %what_index3)
  store i32 %corresponding_int, i32* %res
  %res5 = load i32, i32* %res
  ret i32 %res5
}

define %list* @rotor_shift(%list* %rotor_input, i32 %position) {
entry:
  %rotor_input1 = alloca %list*
  store %list* %rotor_input, %list** %rotor_input1
  %position2 = alloca i32
  store i32 %position, i32* %position2
  %character_holder = alloca i32
  %k = alloca i32
  %l = alloca i32
  %m = alloca i32
  %rotor_input3 = load %list*, %list** %rotor_input1
  %get_at = call i8* (%list*, i32, ...) @get_at(%list* %rotor_input3, i32 0)
  %eltAsPtr = bitcast i8* %get_at to i32*
  %eltDeref = load i32, i32* %eltAsPtr
  store i32 %eltDeref, i32* %character_holder
  store i32 0, i32* %k
  store i32 0, i32* %l
  store i32 0, i32* %m
  store i32 0, i32* %k
  br label %while

while:                                            ; preds = %merge33, %entry
```

```
  %k36 = load i32, i32* %k
  %position37 = load i32, i32* %position2
  %tmp38 = icmp slt i32 %k36, %position37
  br i1 %tmp38, label %while_body, label %merge39

while_body:                                       ; preds = %while
  %rotor_input4 = load %list*, %list** %rotor_input1
  %get_at5 = call i8* (%list*, i32, ...) @get_at(%list* %rotor_input4, i32 0)
  %eltAsPtr6 = bitcast i8* %get_at5 to i32*
  %eltDeref7 = load i32, i32* %eltAsPtr6
  store i32 %eltDeref7, i32* %character_holder
  store i32 0, i32* %l
  br label %while8

while8:                                           ; preds = %merge, %while_body
  %l31 = load i32, i32* %l
  %tmp32 = icmp slt i32 %l31, 25
  br i1 %tmp32, label %while_body9, label %merge33

while_body9:                                      ; preds = %while8
  %l10 = load i32, i32* %l
  %tmp = add i32 %l10, 1
  store i32 %tmp, i32* %m
  %rotor_input11 = load %list*, %list** %rotor_input1
  %l12 = load i32, i32* %l
  %rotor_input13 = load %list*, %list** %rotor_input1
  %m14 = load i32, i32* %m
  %get_at15 = call i8* (%list*, i32, ...) @get_at(%list* %rotor_input13, i32 %m14)
  %eltAsPtr16 = bitcast i8* %get_at15 to i32*
  %eltDeref17 = load i32, i32* %eltAsPtr16
  %tmp18 = alloca i32*
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr = bitcast i8* %malloccall to i32*
  store i32* %tmpAddr, i32** %tmp18
  %tmpWithVal = load i32*, i32** %tmp18
  store i32 %eltDeref17, i32* %tmpWithVal
  %asChar = bitcast i32* %tmpWithVal to i8*
  %set_at = call %list* (%list*, i32, i8*, ...) @set_at(%list* %rotor_input11, i32 %l12, i8* %asChar)
  br label %if

if:                                               ; preds = %while_body9
```

```
  %l27 = load i32, i32* %l
  %tmp28 = icmp eq i32 %l27, 24
  br i1 %tmp28, label %if_body, label %merge

if_body:                                          ; preds = %if
  %rotor_input19 = load %list*, %list** %rotor_input1
  %character_holder20 = load i32, i32* %character_holder
  %tmp21 = alloca i32*
  %malloccall22 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %tmpAddr23 = bitcast i8* %malloccall22 to i32*
  store i32* %tmpAddr23, i32** %tmp21
  %tmpWithVal24 = load i32*, i32** %tmp21
  store i32 %character_holder20, i32* %tmpWithVal24
  %asChar25 = bitcast i32* %tmpWithVal24 to i8*
  %set_at26 = call %list* (%list*, i32, i8*, ...) @set_at(%list* %rotor_input19, i32 25, i8* %asChar25)
  br label %merge

merge:                                            ; preds = %if, %if_body
  %l29 = load i32, i32* %l
  %tmp30 = add i32 %l29, 1
  store i32 %tmp30, i32* %l
  br label %while8

merge33:                                          ; preds = %while8
  %k34 = load i32, i32* %k
  %tmp35 = add i32 %k34, 1
  store i32 %tmp35, i32* %k
  br label %while

merge39:                                          ; preds = %while
  %rotor_input40 = load %list*, %list** %rotor_input1
  ret %list* %rotor_input40
}

define i32 @character_location(%list* %x, i32 %y) {
entry:
  %x1 = alloca %list*
  store %list* %x, %list** %x1
  %y2 = alloca i32
  store i32 %y, i32* %y2
  %j = alloca i32
  %i = alloca i32
  store i32 0, i32* %j
  store i32 0, i32* %i
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge, %entry
  %i9 = load i32, i32* %i
  %tmp10 = icmp slt i32 %i9, 26
  br i1 %tmp10, label %while_body, label %merge11

while_body:                                       ; preds = %while
  br label %if

if:                                               ; preds = %while_body
  %x4 = load %list*, %list** %x1
  %i5 = load i32, i32* %i
  %get_at = call i8* (%list*, i32, ...) @get_at(%list* %x4, i32 %i5)
  %eltAsPtr = bitcast i8* %get_at to i32*
  %eltDeref = load i32, i32* %eltAsPtr
  %y6 = load i32, i32* %y2
  %tmp = icmp eq i32 %eltDeref, %y6
  br i1 %tmp, label %if_body, label %merge

if_body:                                          ; preds = %if
  %i3 = load i32, i32* %i
  ret i32 %i3

merge:                                            ; preds = %if
  %i7 = load i32, i32* %i
  %tmp8 = add i32 %i7, 1
  store i32 %tmp8, i32* %i
  br label %while

merge11:                                          ; preds = %while
  %j12 = load i32, i32* %j
  ret i32 %j12
}

declare noalias i8* @malloc(i32)
```

Chapter 8

Makefile

```
CC = cc
CFLAGS = -g -Wall
LIBS = -lm -lncurses

.PHONY : dune
dune:
    dune build src/smap.exe

.PHONY : dunehello
dunehello :
    dune exec src/smap.exe test/hello.smap
# calc : parser.cmo scanner.cmo calc.cmo
#   ocamlc -w A -o calc $^

parser : parser.cmo scanner.cmo

%.cmo : %.ml
   ocamlc -w A -c $<

%.cmi : %.mli
   ocamlc -w A -c $<

scanner.ml : scanner.mll
   ocamllex $^

parser.ml parser.mli : parser.mly
   ocamlyacc $^

# Depedencies from ocamldep
# calc.cmo : scanner.cmo parser.cmi ast.cmi
# calc.cmx : scanner.cmx parser.cmx ast.cmi
parser.cmo : ast.cmi parser.cmi
parser.cmx : ast.cmi parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx



##############################
```

```
.PHONY : clean
clean :
   rm -rf *.cmi *.cmo parser.ml parser.mli scanner.ml calc.out calc *.o *.dSYM; make
clean2

.PHONY : smap
smap : smap.native

# Everything below the line was copied from the microc makefile
# I changed the make clean rule to a make clean2 rule
# so it wouldn't clash with our current make clean rule
# I also changed the microc.native rule to be smap.native

############### microc makefile stuff ########################################

# "make test" Compiles everything and runs the regression tests
.PHONY : testhello
testhello :
    ./testhello.sh

.PHONY : test
test : all testall.sh
   ./testall.sh

# "make all" builds the executable as well as the "printbig" library designed
# to test linking external code

.PHONY : all
all : smap.native prob.o list.o testMakeStruct.o polymorphicPrint.o c_deps

# "make microc.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by including
# packages, enabling warnings
#
# See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

smap.native :
   opam config exec -- \
   ocamlbuild -use-ocamlfind src/smap.native
```

```
# "make clean" removes all generated files

.PHONY : clean2
clean2 :
   ocamlbuild -clean
   rm -rf testall.log ocamlllvm *.diff

c_deps: list.o prob.o

c_tests: prob list

# Testing the "printbig" example
list.o:
   $(CC) $(CFLAGS) -c -o list.o runtime/list.c $(LIBS)

list: list.o runtime/list.c
   $(CC) $(CFLAGS) -o list -DBUILD_TEST $(LIBS)

prob.o:
   $(CC) $(CFLAGS) -c -o prob.o runtime/prob.c $(LIBS)

prob: list.o runtime/prob.c
   $(CC) $(CFLAGS) list.o -o prob -DBUILD_TEST runtime/prob.c $(LIBS)

printbig : printbig.c
   cc -o printbig -DBUILD_TEST printbig.c

polymorphicPrint.o :
   $(CC) $(CFLAGS) -c -o polymorphicPrint.o runtime/polymorphicPrint.c

testMakeStruct.o :
   $(CC) $(CFLAGS) -c -o testMakeStruct.o runtime/testMakeStruct.c

ncurses.o :
   $(CC) $(CFLAGS) -c -o ncurses.o runtime/ncurses.c

# Building the tarball

TESTS = \
 add1 arith1 arith2 arith3 fib float1 float2 float3 for1 for2 func1 \
 func2 func3 func4 func5 func6 func7 func8 func9 gcd2 gcd global1 \
 global2 global3 hello if1 if2 if3 if4 if5 if6 local1 local2 ops1 \
 ops2 printbig var1 var2 while1 while2

FAILS = \
 assign1 assign2 assign3 dead1 dead2 expr1 expr2 expr3 float1 float2 \
 for1 for2 for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 \
 func8 func9 global1 global2 if1 if2 if3 nomain printbig printb print \
 return1 return2 while1 while2

TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
	    $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)

TARFILES = ast.ml sast.ml codegen.ml Makefile _tags smap.ml microcparse.mly \
   README scanner.mll semant.ml testall.sh \
   printbig.c arcade-font.pbm font2c \
   Dockerfile \
   $(TESTFILES:%=tests/%)

microc.tar.gz : $(TARFILES)
   cd .. && tar czf microc/microc.tar.gz \
	$(TARFILES:%=microc/%)
```

scanner.mll

```
{ open Parser }
let letter = ['a'-'z' 'A'-'Z' ]
let digit = ['0'-'9']
let punc = ['.' ',' ':' ';' '!' '?' '-' '`' '(' ')' '?']
let otherChar = ['@' '#' '$' '%' '^' '|' '^' '&' '*' '~' '[' ']' '{' '}' '\\' '/' '+' '-' '*' '_' '>' '<' '=']
let whitespace = [' ' '\t' '\r' '\n']

rule tokenize = parse
(* scoping *)
| '['              { LBRACKET }
| ']'              { RBRACKET }
| '{'              { LBRACE }
| '}'              { RBRACE }
| '('              { LPAREN }
| ')'              { RPAREN }
```

```

(* operators  *)
| '+'                  { PLUS }
| '-'                  { MINUS }
| '*'                  { TIMES }
| '/'                  { DIVIDE }
| '='                  { ASSIGN }
| "+="                 { ADDEQUAL }
| "-="                 { MINUSEQUAL }
| "*="                 { TIMESEQUAL }
| "/="                 { DIVEQUAL }
| ">>"                 { RSHIFT }
| "<<"                 { LSHIFT }
| '~'                  { BITNOT }
| '&'                  { BITAND }
| '|'                  { BITOR  }
| '^'                  { XOR    }
| "++"                 { CONCAT }
| "[<]"                { ADDHEAD }
| "[>]"                { ADDTAIL }


(* statement symbols *)
| ';'                  { SEMICOLON }
| ':'                  { PROBCOLON }
| ','                  { COMMA }

(* SMAP types *)
| "list"               { LIST }
| "prob"               { PROB}

(* loops *)
| "while"              { WHILE }
| "for"                { FOR }

(* primitive types *)
| "int"                { INT }
| "char"               { CHAR }
| "bool"               { BOOL }
| "string"             { STRING }
| "float"              { FLOAT }
| "void"               { VOID }

(* function declaration *)
| "fn"                 { FUNCTION }

(* comments *)
| "/*"                 { comment lexbuf }
| "//"                 { lineComment lexbuf }

(* whitespace *)
| whitespace { tokenize lexbuf }

(* comparison operators *)
| "=="                 { COMPEQ }
| '<'                  { COMPLT }
| "<="                 { COMPLEQ }
| ">"                  { COMPGT }
| ">="                 { COMPGEQ }
| "!="                 { COMPNEQ }
| "&&"                 { AND }
| "||"                 { OR }
| '!'                  { NOT }

(* Accessor *)
(* if we put dot and length together, "length" doesn't have to be a keyword!*)
| ".length"            { LENGTH  }
| '#'                  { OCTOTHORPE}

(* control flow *)
| "if"                 { IF }
| "elif"               { ELIF }
| "else"               { ELSE }
| "switch"             { SWITCH }
| "case"               { CASE }
| "default"            { DEFAULT }
| "break"              { BREAK }
| "continue"           { CONTINUE }
| "return"             { RETURN }

```

```
(* literals *)
| '\'' '\\' 'n' '\''                              { CHAR_LIT('\n')}
| ['''] ((letter | digit | punc | ['"'] | otherChar | whitespace ) as c)['''] { CHAR_LIT(c)   }  (* char literal   *)
| ['"'] ((letter | digit | punc | ['''] | otherChar| whitespace)* as s )['"'] { STRING_LIT(s) }  (* string literal *)
        | ['0'-'9']+ as lit                        { INT_LIT(int_of_string lit) }     (* int literal    *)
        | ['0'-'9']* ['.'] ['0'-'9']+ as flit        { FLOAT_LIT(float_of_string flit)} (* float literal  *)
        | "true"                                  { BOOL_LIT(true) }
        | "false"                                 { BOOL_LIT(false) }

        (*identifiers *)
        | ['a'-'z' 'A'-'Z' '_']['a'-'z' 'A'-'Z' '0'-'9' '_']* as var_id  { ID(var_id) }

        (* EOF *)
        | eof              { EOF }
        | _ as ch            { raise (Failure("illegal character " ^ Char.escaped ch)) }

        and lineComment =
          parse '\n'          { tokenize lexbuf }
          | _               { lineComment lexbuf}

        and comment =
          parse "*/"          { tokenize lexbuf }
          | _               { comment lexbuf}
```

ast.ml

```
type binary_op = Add | Sub | Mul | Div | Concat | CompEq | CompGeq | CompLt | CompLeq
| CompGt | CompNeq | RShift | LShift | BitAnd | BitOr | Xor
type unary_op = BitNot | Not | Bang | Octothorpe | Neg
type assign_op = PlusEqual | MinusEqual | TimesEqual | DivEqual | Equal

type typ = Int | Bool | Float | Void | Char | String | Prob | List
type typ_name = typ list

type bind = typ_name * string

type expr =
   Binop of expr * binary_op * expr
|   Unop of unary_op * expr
|   Assign of expr * assign_op * expr
|   Int_lit of int
|   Bool_lit of bool
|   Float_lit of float
|   Char_lit of char
|   String_lit of string
|   List_lit of expr list
|   Id of string
|   Cast of typ_name * expr
|   ListElement of string * expr * expr list
|   ListAddHead of expr * expr
|   ListAddTail of expr * expr
|   Length of expr
|   ProbColon of expr * expr
|   FunCall of string * expr list
|   Index of expr
|   Noexpr

type vdecl =
 Vdecl of bind * expr

type stmt =
 Block of stmt list
| Expr of expr
| Return of expr
| If of expr * stmt
| If_Else of stmt * stmt
| If_Elif of stmt * stmt * stmt list
| If_Elif_Else of stmt * stmt * stmt list * stmt
| For of expr * expr * expr * stmt
| While of expr * stmt
| Break
| Continue
| Elif of expr * stmt (*only use inside an if statement, never alone!*)

type func_decl = { typ_name : typ_name;
                 fname : string;
                 formals : bind list;
                 locals : vdecl list;
                 body : stmt list;
```

```
                }

type program = vdecl list * func_decl list

(* dummy pretty printer*)
(*                 (vars, funcs)                              *)
let string_of_program (_,_) = "someday I'll pretty print the AST! \n"

let string_of_uop = function
 Neg -> "-"
| Not -> "!"
| BitNot -> "~"
| Bang -> "!"
| Octothorpe -> "#"

let string_of_typ = function
 Int -> "Int"
| Bool -> "Bool"
| Float -> "Float"
| Char -> "Char"
| String -> "String"
| List -> "List"
| Void -> "Voerid"
| Prob -> "Prob"


let string_of_op = function
   Add -> "+"
 | Sub -> "-"
 | Mul -> "*"
 | Div -> "/"
 | Concat -> "++"
 | CompEq -> "=="
 | CompLt -> "<"
 | CompNeq -> "!="
 | CompLeq -> "<="
 | CompGt -> ">"
 | CompGeq -> ">="
 | RShift -> ">>"
 | LShift -> "<<"
 | BitAnd -> "&"
 | BitOr -> "|"
```

```
  | Xor -> "^"

let string_of_assign = function
    PlusEqual -> "+="
  | MinusEqual -> "-="
  | TimesEqual -> "*="
  | DivEqual -> "/="
  | Equal -> "="

(* !NON EXHUASTIVE PATTERN MATCHING! *)
let rec string_of_expr = function
    Int_lit(l) -> string_of_int l
  | Float_lit(l) -> string_of_float l     (*check floats again*)
  | Bool_lit(true) -> "true"
  | Bool_lit(false) -> "false"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, o, e) -> string_of_expr v ^ " " ^ string_of_assign o ^ " " ^
string_of_expr e
  | FunCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | _ -> "can't print this expr yet"

(* !NON EXHUASTIVE PATTERN MATCHING! *)
let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  (*| If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2*)
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_typ_name typ_lst =
 List.fold_left (fun acc elt -> acc^elt^" ") "" (List.map string_of_typ typ_lst)
```

parser.mly

```
%{ open Ast %}

/* TOKENS
***********************************************************************
***********************/

%token EOF ASSIGN SEMICOLON PROBCOLON COMMA IF ELIF ELSE
// scoping
%token LBRACKET RBRACKET LBRACE RBRACE LPAREN RPAREN

// operators
%token BITNOT BITAND BITOR XOR CONCAT
%token PLUS MINUS TIMES DIVIDE RSHIFT LSHIFT
%token ADDEQUAL MINUSEQUAL TIMESEQUAL DIVEQUAL WHILE FOR
%token AND OR NOT LENGTH OCTOTHORPE
%token ADDHEAD ADDTAIL

// primitives
%token INT CHAR BOOL STRING FLOAT PROB LIST VOID RETURN

%token SWITCH CASE DEFAULT BREAK CONTINUE FUNCTION

//comparison
%token COMPEQ COMPLT COMPLEQ COMPGT COMPGEQ COMPNEQ

//literals
%token <bool>   BOOL_LIT
%token <int>    INT_LIT
%token <char>   CHAR_LIT
%token <float>  FLOAT_LIT
%token <string> STRING_LIT
//identifier
%token <string> ID


```

```
/* OPERATOR PRECEDENCE ************************************************************************************/

%left SEMICOLON
%right ASSIGN PROBCOLON


%left CONCAT
%left PLUS MINUS
%left TIMES DIVIDE

%left ADDEQUAL MINUSEQUAL TIMESEQUAL DIVEQUAL COMPEQ COMPLT COMPLEQ RSHIFT LSHIFT
%left COMPGT COMPGEQ COMPNEQ
%left BITOR BITAND BITNOT XOR
%left OCTOTHORPE

%left COMMA LENGTH

%nonassoc ELIF
%nonassoc ELSE
%nonassoc CAST
%right NOT

%start program
%type <Ast.program> program

%%

/* EXPRESSIONS *******************************************************************************************/
expr:
/* binary expressions                              */
  expr PLUS   expr        { Binop($1, Add, $3)       }
| expr MINUS  expr        { Binop($1, Sub,  $3)      }
| expr TIMES  expr        { Binop($1, Mul, $3)       }
| expr DIVIDE expr        { Binop($1, Div, $3)       }
| expr COMPEQ expr        { Binop($1, CompEq, $3)    }
| expr COMPLT expr        { Binop($1, CompLt, $3)    }
| expr COMPLEQ expr       { Binop($1, CompLeq, $3)   }
| expr COMPGT expr        { Binop($1, CompGt, $3)    }
| expr COMPGEQ expr       { Binop($1, CompEq, $3)    }
| expr COMPNEQ expr       { Binop($1, CompNeq, $3)   }
| expr RSHIFT expr        { Binop($1, RShift, $3)    }
| expr LSHIFT expr        { Binop($1, LShift, $3)    }
| expr BITAND expr        { Binop($1, BitAnd, $3)    }
| expr BITOR expr         { Binop($1, BitOr, $3)     }
| expr XOR    expr        { Binop($1, Xor, $3)       }
/* assignment expressions (special case of binary)    */
| expr ASSIGN expr        { Assign($1, Equal, $3)    }
| expr ADDEQUAL expr      { Assign($1, PlusEqual,$3)  }
| expr MINUSEQUAL expr    { Assign($1, MinusEqual, $3) }
| expr TIMESEQUAL expr    { Assign($1, TimesEqual, $3) }
| expr DIVEQUAL expr      { Assign($1, DivEqual, $3)  }
/* unary expressions                                  */
| MINUS expr                            { Unop(Neg, $2)          } /* what is our syntax for negative numbers? */
| BITNOT expr                           { Unop(BitNot, $2)       }
| NOT expr                              { Unop(Not, $2)          }
| expr NOT                              { Unop(Bang, $1)         }
| expr OCTOTHORPE                             { Unop(Octothorpe, $1)    }
| LPAREN typ_decl RPAREN expr   %prec CAST       { Cast($2, $4)           } /* added this in for now. Hoping precedence is correct*/
/* literal expressions                                */
| INT_LIT                               { Int_lit ($1)          }
| BOOL_LIT                              { Bool_lit($1)          }
| STRING_LIT                            { String_lit($1)        }
| FLOAT_LIT                             { Float_lit ($1)        }
| CHAR_LIT                              { Char_lit($1)          }
/* identifiers                                        */
| id                               { $1                  }
/* List expressions                                   */
| LBRACKET expr_list_opt RBRACKET              { List_lit(List.rev $2) } /* list literal */
| id ADDHEAD ASSIGN expr                 { ListAddHead($1,$4)      } /* add head    */
| id ADDTAIL ASSIGN expr                 { ListAddTail($1,$4)      } /* add tail    */
| expr CONCAT expr                    { Binop($1, Concat, $3)    }
| expr LENGTH                         { Length($1)           } /* also for prob types */
/* prob type expressions                                     */
| expr PROBCOLON expr                 { ProbColon($1,$3)       }
/*function call expression                                   */
| ID LPAREN args_opt RPAREN           { FunCall ($1,$3)        }
/* parentheses around an expression                          */
| LPAREN expr RPAREN                  { $2                     }

/* identifier is either an indexed list element or regular text identifier  */
id:
  ID                                  { Id($1)             }
| list_elt                            { $1                 }

/* indexing into a list  */
list_elt: ID index index_list_opt  { ListElement($1,$2,$3) }  /* indexed elt (for ex a[0][0][1])      */
index: LBRACKET expr RBRACKET      { Index($2)           }  /* a single index (for ex. [0])         */
index_list_opt:
                                   { []                  }  /* 0 or more indices (for ex. [1][2])    */
| index index_list_opt             { $1::$2              }

/* recursive expressions to handle plurals  */
expr_opt:                       /* zero or one expressions */
                   { Noexpr     }
| expr             { $1         }

expr_list_opt:                  /* zero or more expressions */
                   { []         }
| expr_list        { $1         }

expr_list:                      /* one or more expressions */
  expr             { [$1]       }
| expr_list COMMA expr  { $3 :: $1    }

args_opt:
                   { []         } /* zero or more arguments */
| args_list        { List.rev $1 }
```

```

args_list:
  expr                { [$1]       } /* one argument          */
| args_list COMMA expr  { $3 :: $1    } /* more than one argument */

/* STATEMENTS *****************************************************************************************/

stmt: /* statements end with either a semicolon or a block */
  expr SEMICOLON                                                    { Expr($1)         }
| BREAK SEMICOLON                                                   { Break            }
| CONTINUE SEMICOLON                                                { Continue         }
| RETURN expr_opt SEMICOLON                                         { Return($2)       }
| if_stmt                                                   { $1               }
| if_stmt ELSE block                                                { If_Else($1,$3)    }
| if_stmt elif elif_list                                          { If_Elif($1,$2,$3) }
| if_stmt elif elif_list ELSE block                                 { If_Elif_Else($1,$2,$3,$5)}
| FOR LPAREN expr_opt SEMICOLON expr_opt SEMICOLON expr_opt RPAREN block          { For($3,$5,$7,$9) }
| WHILE LPAREN expr RPAREN block                                    { While($3,$5)      }

if_stmt: IF LPAREN expr RPAREN block                                { If($3,$5)        }
elif:    ELIF LPAREN expr RPAREN  block                             { Elif($3,$5)      }
block:   LBRACE stmt_list RBRACE                                    { Block(List.rev $2)}

stmt_list:
  /* nothing */      { []     }          /* zero or more statements */
| stmt_list stmt      { $2::$1 }

elif_list:
| /*nothing*/          { []     }        /* zero or more elif statements*/
| elif elif_list       { $1::$2 }


program: decls EOF { $1 }                 /* a program consists of declarations and the EOF token*/
decls:
  /* nothing */ { ([], []) }
| decls vdecl { (($2 :: fst $1), snd $1) } /* global variable declarations */
| decls fdecl { (fst $1, ($2 :: snd $1)) } /* function definitions         */


/* FUNCTION DECLARATIONS ************************************************************************************/
fdecl: typ_decl ID LPAREN formals_opt RPAREN     /* are we going to use the function keyword??*/
LBRACE vdecl_list stmt_list RBRACE
{{ typ_name = $1;
   fname = $2;
   formals = List.rev $4;
   locals = List.rev $7;
   body = List.rev $8;
  }}

formals_opt:
/* nothing */ { [] }                          /* zero or more formal parameters */
| formal_list { $1 }

formal_list:
  typ_decl ID                 { [($1,$2)]     } /* formal param consists of type declaration and identifer */
| formal_list COMMA typ_decl ID { ($3,$4) :: $1 } /* a list of formal params are separted by commas          */

vdecl_list:
/* nothing */      { []       }                          /*      zero or more variable declarations  */
| vdecl_list vdecl { $2 :: $1 }

```

```
vdecl:                                  /*    a variable declaration consists of   */
  typ_decl ID SEMICOLON { Vdecl (($1, $2),Noexpr) }          /*    a type declaration and identifier,   */
| typ_decl ID ASSIGN expr SEMICOLON { Vdecl (($1, $2),$4)}  /*    and optional initializer          */


/* TYPE DECLARATIONS ***************************************************************************************/
typ_decl:                     /* A type decl is composed of a list of types */
| spec_qual_list    {$1}

typ_spec:
  INT    { Int    }
| BOOL   { Bool   }
| FLOAT  { Float  }
| VOID   { Void   }
| CHAR   { Char   }
| STRING  { String }

typ_qual:
| PROB   { Prob }
| LIST   { List }

spec_qual_list:
| typ_spec spec_qual_list_opt {$1::$2}  /* int/bool/float/void/char/string, followed by zero or more types */
| typ_qual spec_qual_list {$1::$2}      /* list/prob, followed by at least one type */

spec_qual_list_opt:                  /* zero or more types */
| /*nothing*/    {[]}
| spec_qual_list {$1}
```

sast.ml

```
(* Semantically-checked Abstract Syntax Tree and functions for printing it *)


open Ast
```

```
type sexpr = typ_name * sx
and sx =
   SBinop of sexpr * binary_op * sexpr
|   SUnop of unary_op * sexpr
|   SAssign of sexpr * assign_op * sexpr
|   SInt_lit of int
|   SBool_lit of bool
|   SFloat_lit of float
|   SChar_lit of char
|   SString_lit of string
|   SList_lit of sexpr list
|   SId of string
|   SCast of typ_name * sexpr
|   SListElement of sexpr * sexpr * sexpr list
|   SListAddHead of sexpr * sexpr
|   SListAddTail of sexpr * sexpr
|   SLength of sexpr
|   SProbColon of sexpr * sexpr
|   SFunCall of string * sexpr list
|   SIndex of sexpr
|   SNoexpr

type svdecl =
 SVdecl of bind * sexpr

type sstmt =
| SBlock of sstmt list
| SExpr of sexpr
| SReturn of sexpr
| SIf of sexpr * sstmt
| SIf_Else of sstmt * sstmt
| SIf_Elif of sstmt * sstmt * sstmt list
| SIf_Elif_Else of sstmt * sstmt * sstmt list * sstmt
| SFor of sexpr * sexpr * sexpr * sstmt
| SWhile of sexpr * sstmt
| SBreak
| SContinue
| SElif of sexpr * sstmt (*only use inside an if statement, never alone!*)

type sfunc_decl = { styp_name : typ_name;
```

```
                  sfname : string;
                  sformals : bind list;
                  slocals : svdecl list;
                  sbody : sstmt list;
                }


let rec string_of_sexpr (t, e) =
 "(" ^ string_of_typ (List.hd t) ^ " : " ^ (match e with
   SInt_lit(l) -> string_of_int l
 | SFloat_lit(l) -> string_of_float l     (*check floats again*)
 | SBool_lit(true) -> "true"
 | SBool_lit(false) -> "false"
 | SId(s) -> s
 | SBinop(e1, o, e2) ->
     string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
 | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
 | SAssign(v, o, e) -> string_of_sexpr v ^ " " ^ string_of_assign o ^ " " ^
string_of_sexpr e
 | SFunCall(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
 | SNoexpr -> ""
         ) ^ ")"

let rec string_of_sstmt = function
   SBlock(stmts) ->
     "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
 | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
 | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
 (*| SIf(e, s, SBlock([])) ->
     "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
 | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
     string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2*)
 | SFor(e1, e2, e3, s) ->
     "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
     string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
 | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

type sprogram = svdecl list * sfunc_decl list

type sDummy = (sprogram * string) (* Dummy SAST node for throughline*)
```

```
(*****************************************************************)
(*
type sexpr = typ * sx
and sx =
   SLiteral of int
 | SFliteral of string
 | SBoolLit of bool
 | SId of string
 | SBinop of sexpr * binary_op * sexpr
 | SUnop of unary_op * sexpr
 | SAssign of string * sexpr
 | SCall of string * sexpr list
 | SNoexpr

type sstmt =
   SBlock of sstmt list
 | SExpr of sexpr
 | SReturn of sexpr
 | SIf of sexpr * sstmt * sstmt
 | SFor of sexpr * sexpr * sexpr * sstmt
 | SWhile of sexpr * sstmt

type sfunc_decl = {
   styp_name : typ_name;
   sfname : string;
   sformals : bind list;
   slocals : bind list;
   sbody : sstmt list;
 }

(*type sprogram = svdecl list * sfunc_decl list (* the start of our real SAST*)*)


(*

type sexpr = typ * sx
and sx =
   SLiteral of int
 | SFliteral of string
 | SBoolLit of bool
 | SId of string
 | SBinop of sexpr * op * sexpr
 | SUnop of uop * sexpr
 | SAssign of string * sexpr
 | SCall of string * sexpr list
 | SNoexpr

type sstmt =
   SBlock of sstmt list
 | SExpr of sexpr
 | SReturn of sexpr
 | SIf of sexpr * sstmt * sstmt
 | SFor of sexpr * sexpr * sexpr * sstmt
 | SWhile of sexpr * sstmt

type sfunc_decl = {
   styp : typ;
   sfname : string;
   sformals : bind list;
   slocals : bind list;
   sbody : sstmt list;
 }

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
 "(" ^ string_of_typ t ^ " : " ^ (match e with
   SLiteral(l) -> string_of_int l
 | SBoolLit(true) -> "true"
 | SBoolLit(false) -> "false"
 | SFliteral(l) -> l
 | SId(s) -> s
 | SBinop(e1, o, e2) ->
     string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
 | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
 | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
 | SCall(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
 | SNoexpr -> ""
         ) ^ ")"

let rec string_of_sstmt = function
```

```
   SBlock(stmts) ->
     "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
 | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
 | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
 | SIf(e, s, SBlock([])) ->
     "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
 | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
     string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
 | SFor(e1, e2, e3, s) ->
     "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
     string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
 | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

let string_of_sfdecl fdecl =
 string_of_typ fdecl.styp ^ " " ^
 fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
 ")\n{\n" ^
 String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
 String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
 "}\n"

let string_of_sprogram (vars, funcs) =
 String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
 String.concat "\n" (List.map string_of_sfdecl funcs)
*)
*)

(* dummy pretty printer*)
(*                 (vars, funcs)                              *)
let string_of_sprogram (_,_) = "someday I'll pretty print the SAST! \n"
```

codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html
Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)
(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
 let context = L.global_context () in

 (* Create the LLVM compilation module into which we will generate code *)
 let the_module = L.create_module context "Smap" in

 (*
 For reference - delete before committing!!!
 typedef struct list{
   void **data;
   int capacity;
   int size;
 } list;

 typedef struct prob {
   list probs;
   list vals;
   int length;
} prob;
*)

 let i32_t = L.i32_type context (* 32 bit integer *)
 and i8_t = L.i8_type context   (* 8 bit integer *)
 and i1_t = L.i1_type context   (* 1 bit integer *)
 and str = L.pointer_type (L.i8_type context)
 and float_t = L.double_type context
 and void_t = L.void_type context
 in
 let dummy_t = L.struct_type context [|str; i32_t;|]
```

```
in
let list_t : L.lltype = L.named_struct_type context "list"
in
let prob_t : L.lltype = L.named_struct_type context "prob"
in
(* Return the LLVM type for a MicroC type *)
let ltype_of_typ = function
  A.Int -> i32_t
| A.Char -> i8_t
| A.Bool -> i1_t
| A.Float -> float_t
| A.Void -> void_t
| A.String -> L.pointer_type list_t
| A.Prob -> L.pointer_type prob_t (*need to change to prob_t eventually*)
| A.List -> L.pointer_type list_t
| _ -> void_t (*add in prob, string, and list types later! *)
in

(*make sure to include struct definitions for the llvm struct types named earlier*)
let _ =
  L.struct_set_body list_t
    [| L.pointer_type (L.pointer_type i8_t)
    ; i32_t
    ; i32_t |] false;
L.struct_set_body prob_t
  [| list_t
  ; list_t
  ; i32_t |] false;
in

(* for reference
SFunCall ("printint", [e])
  -> L.build_call printint_func [| (expr builder e) |]
     "printint" builder
*)
let genGlobalListLit (typlist,exp) n m =
  let defaultZero =
    let zero = L.const_int i32_t 0 in
    L.const_named_struct list_t [|zero; zero; zero|]
  in
  let wholeThing = (match exp with
    (SList_lit(inits))
```

```
     -> (match inits with
         ([A.Int],SInt_lit(v))::rest -> let one = L.const_int i32_t 1 in
                                     L.const_named_struct list_t [|one; one; one|]
         | _ -> let zero = L.const_int i32_t 0 in
                 L.const_named_struct list_t [|zero; zero; zero|])
     | _ -> defaultZero)
   in StringMap.add n (L.define_global n wholeThing the_module) m
 in

 (* Create a map of global variables after creating each *)
 let global_vars : L.llvalue StringMap.t =
 let global_var m (SVdecl((t, n),exp)) =
 let simpleType = List.hd t in (*just use head of type list for now *)
 match simpleType with
    A.List
    -> genGlobalListLit exp n m
   | _
    -> let init = (match simpleType with (* add more types in this pattern match! *)
                      A.Float
                      -> (match exp with
                            (_,SFloat_lit v)
                            -> L.const_float (ltype_of_typ simpleType) v
                            | _
                            ->  L.const_float (ltype_of_typ simpleType) 0.0)
                      | A.Int
                      -> (match exp with
                            (_, SInt_lit v)
                            -> L.const_int (ltype_of_typ simpleType) v
                            |_
                            ->  L.const_int(ltype_of_typ simpleType) 0)
                      (*TODO: change this to actually work with prob type!!!*)
                      | A.Prob
                      -> L.const_int i32_t 0
                      | _
                      -> L.const_int i32_t 0)
       in StringMap.add n (L.define_global n init the_module) m in
 List.fold_left global_var StringMap.empty globals in

 (* declare built in C functions *)
 let printstr_t: L.lltype = L.function_type i32_t [| L.pointer_type i8_t |] in
 let printstr_func : L.llvalue = L.declare_function "printstr" printstr_t the_module
in
```

```
 let corresponding_char_t: L.lltype = L.function_type i32_t [| i32_t |] in
 let corresponding_char_func : L.llvalue = L.declare_function "corresponding_char"
corresponding_char_t the_module in
 let printchar_t : L.lltype = L.function_type i32_t [| i8_t |] in
 let printchar_func : L.llvalue = L.declare_function "printchar" printchar_t
the_module in

 let print_list_char_t : L.lltype = L.function_type i32_t [| L.pointer_type list_t |]
in
 let print_list_char_func : L.llvalue = L.declare_function "print_list_char"
print_list_char_t the_module in

 let printb_t: L.lltype = L.function_type i32_t [| i1_t |] in
 let printb_func : L.llvalue = L.declare_function "printb" printb_t the_module in

 let stringLength_t: L.lltype = L.function_type i32_t [| L.pointer_type list_t |] in
 let stringLength_func: L.llvalue = L.declare_function "stringLength" stringLength_t
the_module in

 let stringReverse_t: L.lltype = L.function_type i32_t [| L.pointer_type list_t |] in
 let stringReverse_func: L.llvalue = L.declare_function "stringReverse"
stringReverse_t the_module in

 let absolute_t: L.lltype = L.function_type i32_t [| i32_t |] in
 let absolute_func : L.llvalue = L.declare_function "absolute" absolute_t the_module
in

 let key_test_t: L.lltype = L.function_type i32_t [| i32_t;i32_t |] in
 let key_test_func : L.llvalue = L.declare_function "key_test" key_test_t the_module
in

 let characterLocation_t: L.lltype = L.function_type i32_t [| L.pointer_type
list_t;L.pointer_type list_t |] in
 let characterLocation_func : L.llvalue = L.declare_function "characterLocation"
characterLocation_t the_module in

 let isCompDivisible_t: L.lltype = L.function_type i32_t [| i32_t |] in
 let isCompDivisible_func : L.llvalue = L.declare_function "isCompDivisible"
isCompDivisible_t the_module in

 let divisible_t: L.lltype = L.function_type i32_t [| i32_t;i32_t |] in
 let divisible_func : L.llvalue = L.declare_function "divisible" divisible_t
the_module in

 let power_t: L.lltype = L.function_type i32_t [| i32_t;i32_t |] in
 let power_func : L.llvalue = L.declare_function "power" power_t the_module in

 let ceilFloat_t: L.lltype = L.function_type i32_t [| float_t |] in
 let ceilFloat_func : L.llvalue = L.declare_function "ceilFloat" ceilFloat_t
the_module in

 let ascii_t: L.lltype = L.function_type i32_t [| L.pointer_type list_t |] in
 let ascii_func : L.llvalue = L.declare_function "ascii" ascii_t the_module in

 let int_to_char_t: L.lltype = L.function_type (L.pointer_type list_t) [| i32_t|] in
 let int_to_char_func : L.llvalue = L.declare_function "int_to_char" int_to_char_t
the_module in

 let corresponding_int_t: L.lltype = L.function_type i32_t [| L.pointer_type
list_t;i32_t |] in
 let corresponding_int_func: L.llvalue = L.declare_function "corresponding_int"
corresponding_int_t the_module in

 let testMakeStruct_t: L.lltype = L.function_type dummy_t [| L.pointer_type i8_t;i32_t
|] in
 let testMakeStruct_func : L.llvalue = L.declare_function "testMakeStruct"
testMakeStruct_t the_module in

 let printint_t: L.lltype = L.function_type i32_t [| i32_t |] in
 let printint_func : L.llvalue = L.declare_function "printint" printint_t the_module
in

 let printf_t : L.lltype = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
 let printf_func : L.llvalue = L.declare_function "printf" printf_t the_module in

 let push_front_t : L.lltype = L.var_arg_function_type i32_t [|L.pointer_type list_t;
L.pointer_type i8_t|] in
 let push_front_func : L.llvalue = L.declare_function "push_front" push_front_t
the_module in

 let push_back_t : L.lltype = L.var_arg_function_type i32_t [|L.pointer_type list_t;
L.pointer_type i8_t|] in
```

```
 let push_back_func : L.llvalue = L.declare_function "push_back" push_back_t
the_module in

 let init_list_t : L.lltype = L.var_arg_function_type i32_t [|L.pointer_type list_t|]
in
 let init_list_func : L.llvalue = L.declare_function "init_list" init_list_t
the_module in

 let init_prob_t : L.lltype = L.var_arg_function_type i32_t [|L.pointer_type prob_t;
                                                              L.pointer_type list_t;
                                                              L.pointer_type list_t|]
in
 let init_prob_func : L.llvalue = L.declare_function "init_prob" init_prob_t
the_module in

 let peek_t : L.lltype = L.var_arg_function_type (L.pointer_type i8_t)
[|L.pointer_type prob_t|] in
 let peek_func : L.llvalue = L.declare_function "peek" peek_t the_module in
  let list_length_t : L.lltype = L.var_arg_function_type i32_t [|L.pointer_type
list_t|] in
 let list_length_func : L.llvalue = L.declare_function "list_length" list_length_t
the_module in

 let modulus_operation_t: L.lltype = L.var_arg_function_type i32_t [|i32_t;i32_t|] in
 let modulus_operation_func : L.llvalue = L.declare_function "modulus_operation"
modulus_operation_t the_module in

 let get_length_t :L.lltype = L.var_arg_function_type i32_t [|L.pointer_type prob_t|]
in
 let get_length_func : L.llvalue = L.declare_function "get_length" get_length_t
the_module in

 let get_at_t : L.lltype = L.var_arg_function_type (L.pointer_type i8_t)
[|L.pointer_type list_t; i32_t|] in
 let get_at_func : L.llvalue = L.declare_function "get_at" get_at_t the_module in

 let set_at_t : L.lltype = L.var_arg_function_type (L.pointer_type list_t)
[|L.pointer_type list_t; i32_t; L.pointer_type i8_t|] in
 let set_at_func : L.llvalue = L.declare_function "set_at" set_at_t the_module in

 let print_list_int_t : L.lltype = L.var_arg_function_type i32_t [|L.pointer_type
list_t|] in
```

```
 let print_list_int_func : L.llvalue = L.declare_function "print_list_int"
print_list_int_t the_module in

 let print_list_float_t : L.lltype = L.var_arg_function_type i32_t [|L.pointer_type
list_t|] in
 let print_list_float_func : L.llvalue = L.declare_function "print_list_float"
print_list_float_t the_module in

 let get_vals_t : L.lltype = L.var_arg_function_type (L.pointer_type list_t)
[|L.pointer_type prob_t|] in
 let get_vals_func : L.llvalue = L.declare_function "get_vals" get_vals_t the_module
in

 let get_probs_t : L.lltype = L.var_arg_function_type (L.pointer_type list_t)
[|L.pointer_type prob_t|] in
 let get_probs_func : L.llvalue = L.declare_function "get_probs" get_probs_t
the_module in

 let print_prob_int_debug_t : L.lltype = L.var_arg_function_type i32_t
[|L.pointer_type prob_t|] in
 let print_prob_int_debug_func : L.llvalue = L.declare_function "print_prob_int_debug"
print_prob_int_debug_t the_module in

 let bad_add_head_t : L.lltype = L.var_arg_function_type i32_t [| L.pointer_type
list_t; L.pointer_type i8_t |] in
 let bad_add_head_func : L.llvalue = L.declare_function "bad_add_head" bad_add_head_t
the_module in

 let very_bad_get_head_t : L.lltype = L.var_arg_function_type i32_t [| L.pointer_type
list_t|] in
 let very_bad_get_head_func : L.llvalue = L.declare_function "very_bad_get_head"
very_bad_get_head_t the_module in

 (* Define each function (arguments and return type) so we can
 call it even before we've created its body *)
 let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
 let function_decl m fdecl =
 let name = fdecl.sfname
 and formal_types = Array.of_list (List.map (fun (t,_) -> ltype_of_typ (List.hd t))
fdecl.sformals) in
 let simpleReturnType = List.hd fdecl.styp_name in
 let ftype = L.function_type (ltype_of_typ simpleReturnType) formal_types in
 StringMap.add name (L.define_function name ftype the_module, fdecl) m in
 List.fold_left function_decl StringMap.empty functions in
 (* ... *)
 (* Fill in the body of the given function *)
 let build_function_body fdecl =
   let (the_function, _) =
   StringMap.find fdecl.sfname function_decls in
   let builder =
   L.builder_at_end context (L.entry_block the_function) in
   let test_str =
     L.build_global_stringptr "test test test!\n" "test" builder in
   let newLine_str =
     L.build_global_stringptr "\n" "newLine" builder in
   let int_format_str =
   L.build_global_stringptr "%d" "fmt" builder
   and float_format_str =
   L.build_global_stringptr "%f" "fmt" builder in

 let local_vars =
   let add_formal m (t, n) p = (* add formal parameters to scope *)
   L.set_value_name n p;
   let simpleType = List.hd t in
   let local = L.build_alloca (ltype_of_typ simpleType) n builder in
   ignore (L.build_store p local builder);
   StringMap.add n local m
   and add_local m (SVdecl((t, n),exp)) = (* add locals to scope  *)
   let simpleType2 = List.hd t in
   let local_var = L.build_alloca (ltype_of_typ simpleType2) n builder in
   StringMap.add n local_var m in
   let formals = List.fold_left2 add_formal StringMap.empty fdecl.sformals
(Array.to_list (L.params the_function)) in
   List.fold_left add_local formals fdecl.slocals in

 (* Return the value for a variable or formal argument. Check local names first, then
global names *)
 let lookup n = try StringMap.find n local_vars with
                   Not_found -> StringMap.find n global_vars in

 let rec expr builder ((theTyp, e) : sexpr) =
 (*
 allocInitElt helper function
 Description:
```

```
Allocate and intitalize a list elt, then cast to ( char * )
For ex: int * tmp = malloc(sizeOf(int));
*tmp = 5;
char * tmp2 = ( char* ) tmp;
After this step, ready to pass vals to the push_front function!
*)
let allocInitElt v =
(* save type of element *)
let vtype = fst v in
(* 5 *)
let value = expr builder v in
(* int * tmp *)
let tmp = match (vtype) with
             [A.Int] -> L.build_alloca (L.pointer_type i32_t) "tmp" builder
           | [A.Float] -> L.build_alloca (L.pointer_type float_t) "tmp" builder
           | [A.Char] -> L.build_alloca (L.pointer_type i8_t) "tmp" builder
           | [A.Bool] -> raise(Failure("fill in for bool"))
           | [A.Prob] -> raise(Failure("fill in for prob someday... >.<\""))
in
(* malloc(sizeOf(int)) *)
let heapAddr = match (vtype) with
                     [A.Int] -> L.build_malloc i32_t "tmpAddr" builder
                   | [A.Float] -> L.build_malloc float_t "tmpAddr" builder
                   | [A.Char] -> L.build_malloc i8_t "tmpAddr" builder
                   | [A.Bool] -> raise(Failure("fill in for bool"))
                   | [A.Prob] -> raise(Failure("fill in for prob someday... >.<\""))
in
(* tmp = malloc(sizeOf(int)); *)
let tmp' = ignore(L.build_store heapAddr tmp builder); tmp in
(* *tmp *)
let tmp'' = L.build_load tmp' "tmpWithVal" builder in
(* *tmp = 5 *)
let tmp''' = ignore (L.build_store value tmp'' builder); tmp'' in
(* char * tmp2 = ( char* ) tmp;  *)
let asChar = L.build_bitcast tmp''' (L.pointer_type i8_t) "asChar" builder
in asChar
in
(* multidimensional indexing for lists helper function *)
let rec deRef ((typs,i)::rest) handle =
  let elt = L.build_call get_at_func [| handle; expr builder (typs,i)|]
      "get_at" builder in
      match typs with
```

```
         [A.Int]
         -> let eltAsPtr = L.build_bitcast elt (L.pointer_type i32_t) "eltAsPtr"
builder  in
         L.build_load eltAsPtr "eltDeref" builder
       | [A.Float]
       ->  let eltAsPtr = L.build_bitcast elt (L.pointer_type float_t) "eltAsPtr"
builder in
       L.build_load eltAsPtr "eltDeref" builder
       | [A.Char]
       -> let eltAsPtr = L.build_bitcast elt (L.pointer_type i8_t) "eltAsPtr" builder
in
           L.build_load eltAsPtr "eltDeref" builder
       | [A.String]
       -> let eltAsPtr = L.build_bitcast elt (L.pointer_type list_t) "eltAsPtr"
builder in
       L.build_load eltAsPtr "eltDeref" builder
       | A.List::t
       ->
       if (List.length rest) = 0
       then L.build_bitcast elt (L.pointer_type list_t) "eltAsPtr" builder
       else deRef rest (L.build_bitcast elt (L.pointer_type list_t) "eltAsPtr"
builder)
       | [A.Bool]
       -> raise(Failure("fill in for bool"))
       | A.Prob::tl
       -> raise(Failure("fill in for prob someday... >.<\""))
   in
   match e with
   SInt_lit i
   -> L.const_int i32_t i
   | SIndex i
   -> expr builder i
   | SString_lit s
   -> let len = String.length s in
      let rec explode i m str acc =
       if i = m then acc
       else let c = (String.get str i) in
       c::(explode (i+1) m str acc) in
      let chars = explode 0 len s [] in
      let toCharNode c = ([A.Char],SChar_lit c) in
      let asList = ([A.List;A.Char], SList_lit(List.map toCharNode chars)) in
      (* L.build_global_stringptr s "the_str" builder *)
      expr builder asList
   | SBool_lit b
   -> L.const_int i1_t (if b then 1 else 0)
   | SFloat_lit l
   -> L.const_float float_t l
   | SChar_lit l
   -> L.const_int i8_t (Char.code l)
   | SNoexpr
   -> L.const_int i32_t 0
   | SId s
   -> L.build_load (lookup s) s builder
   | SLength(p)
   -> let p' = expr builder p in
     (match List.hd (fst p) with
        A.Prob
        -> L.build_call get_length_func [| p' |]
        "get_length" builder
      | A.List
      -> L.build_call list_length_func [| p' |]
      "list_length" builder
      | A.String
      -> L.build_call list_length_func [| p' |]
      "list_length" builder
      |_
      -> raise (Failure("semant should have caught this misuse of .length!
"^A.string_of_typ_name (fst p))) )
   | SListElement (listId,index,rest)
   -> let eltType = fst index in
      (match eltType with
        A.List::tl
        -> let theList = expr builder listId in


           deRef (index::rest) theList
        |_
        -> let theList = expr builder listId in
           let elt = L.build_call get_at_func [| theList; expr builder index|]
                "get_at" builder in
           let eltAsPtr = match eltType with
                      [A.Int] -> L.build_bitcast elt (L.pointer_type i32_t) "eltAsPtr"
builder
                      | [A.Float] -> L.build_bitcast elt (L.pointer_type float_t)
"eltAsPtr" builder
```

```
                      | [A.Char] -> L.build_bitcast elt (L.pointer_type i8_t)
"eltAsPtr" builder
                      | [A.Bool] -> raise(Failure("fill in for bool"))
                      | A.Prob::tl -> raise(Failure("fill in for prob someday...
>.<\""))
                     in
           let eltDeRef = L.build_load eltAsPtr "eltDeref" builder in
           eltDeRef)
   | SListAddHead (e1,e2)
   ->  let e1' = expr builder e1
       and toAdd = match fst e2 with
                    (A.List::t)
                    -> expr builder e2
                    | _
                    -> allocInitElt e2
       in let _ = L.build_call push_front_func [| e1'; toAdd |]
          "push_front" builder in e1'
   | SListAddTail (e1,e2)
   ->  let e1' = expr builder e1
       and toAdd = match fst e2 with
                    (A.List::t)
                    -> expr builder e2
                    | _
                    -> allocInitElt e2
       in let _ = L.build_call push_back_func [| e1'; toAdd |]
          "push_back" builder in e1'
   | SList_lit elts
   -> (match theTyp with
       [A.List;A.List;u]
       -> (*create the sublists*)
       let sublists = List.map (expr builder) elts in
       (* cast the list pointers to char pointers*)
       let toChar l = L.build_bitcast l (L.pointer_type i8_t) "toChar" builder in
       let charPtrs = List.map toChar sublists in
       let addElt lst c =
         L.build_call push_back_func [| lst; c |]
         "push_back" builder
       in
       (* allocate list struct *)
       let aList = L.build_alloca (L.pointer_type list_t) "theList" in
       (* get pointer to list struct *)
       let anAllocatedList = L.build_malloc list_t "listAddr" builder in
```

```
    (* initialize the list with list_init *)
    let _ =  L.build_call init_list_func [| anAllocatedList|]
    "init_list" builder in
    let _ = List.map (addElt anAllocatedList) charPtrs
    (* return the pointer to the now-intialized list*)
    in anAllocatedList
    | [A.List;u] (*simple list of one type - (no lists of lists here)*)
    -> (match elts with
        [] (*CASE: EMPTY LIST *)
        ->
        (* get pointer to list struct *)
        let anAllocatedList = L.build_malloc list_t "listAddr" builder in
        (* initialize the list with list_init *)
        let _ =  L.build_call init_list_func [| anAllocatedList|]
        "init_list" builder
        in anAllocatedList
      | (elt::rest) (*CASE: NON-EMPTY LIST *)
      ->
      let addElt lst c =
        L.build_call push_back_func [| lst; c |]
        "push_back" builder
      in
      (* allocate list struct *)
      let aList = L.build_alloca (L.pointer_type list_t) "theList" in
      (* get pointer to list struct *)
      let anAllocatedList = L.build_malloc list_t "listAddr" builder in
      (* initialize the list with list_init *)
      let _ =  L.build_call init_list_func [| anAllocatedList|]
      "init_list" builder in
      (* add any starting elts to the list *)
      let values = List.map allocInitElt (elt::rest)
      in
      let _ = List.map (addElt anAllocatedList) values
      (* return the pointer to the now-intialized list*)
      in anAllocatedList)
    |_
    -> raise (Failure "Only support simple lists at the moment >.<\""))
| SProbColon (lhs,rhs)
-> let lhs' = expr builder lhs
   and rhs' = expr builder rhs in
    (* get pointer to prob struct *)
    let anAllocatedProb = L.build_malloc prob_t "probAddr" builder in
```

```
       (* initialize the prob with prob_init *)
       let _ =  L.build_call init_prob_func [| anAllocatedProb; lhs'; rhs'|]
       "init_prob" builder in
       (* let vs = L.build_call get_vals_func [| anAllocatedProb|]
       "get_vals" builder in
       let _ = L.build_call print_list_int_func [|  (vs)  |]
       "print_list_int" builder in *)
       anAllocatedProb
   | SUnop(op, ((t, _) as e))
   -> let e' = expr builder e in (match op with
       A.Neg when t = [A.Float] -> L.build_fneg  e' "tmp" builder
       | A.Neg
       -> L.build_neg e' "tmp" builder
       | A.BitNot
       -> L.build_not e' "tmp" builder
       | A.Bang
       -> let cPtr = L.build_call peek_func [| e'|] "peek" builder in
          (* let vs = L.build_call get_vals_func [| e'|] "get_vals" builder in
          let cPtr = L.build_call get_at_func [|vs; L.const_int i32_t 0|] "get_at"
builder in  *)
          let eltAsPtr = match theTyp with
                        [A.Int] -> L.build_bitcast cPtr (L.pointer_type i32_t)
"eltAsPtr" builder
                      | [A.Float] -> L.build_bitcast cPtr (L.pointer_type float_t)
"eltAsPtr" builder
                      | [A.Char] -> L.build_bitcast cPtr (L.pointer_type i8_t)
"eltAsPtr" builder
                      | [A.Bool] -> L.build_bitcast cPtr (L.pointer_type i1_t)
"eltAsPtr" builder
                      | [A.List] -> L.build_bitcast cPtr (L.pointer_type list_t)
"eltAsPtr" builder
                     in
          let eltDeRef = L.build_load eltAsPtr "eltDeref" builder in
          eltDeRef
       | A.Octothorpe
       -> L.build_call get_probs_func [| e'|] "get_probs" builder
       | A.Not
       -> L.build_not e' "tmp" builder)
   | SBinop (([Float],_ ) as e1, op, e2) ->
   let e1' = expr builder e1
   and e2' = expr builder e2 in
   (match op with
     A.Add     -> L.build_fadd
   | A.Sub     -> L.build_fsub
   | A.Mul     -> L.build_fmul
   | A.Div     -> L.build_fdiv
   | A.CompNeq -> L.build_fcmp L.Fcmp.One
   | A.CompLt  -> L.build_fcmp L.Fcmp.Olt
   | A.CompLeq -> L.build_fcmp L.Fcmp.Ole
   | A.CompGt  -> L.build_fcmp L.Fcmp.Ogt
   | A.CompGeq -> L.build_fcmp L.Fcmp.Oge
   | A.BitAnd | A.BitOr ->
       raise (Failure "internal error: semant should have rejected and/or on float")
   ) e1' e2' "tmp" builder
     | SBinop (e1, op, e2) ->
   let e1' = expr builder e1
   and e2' = expr builder e2 in
   (match op with
     A.Add     -> L.build_add
   | A.Sub     -> L.build_sub
   | A.Mul     -> L.build_mul
   | A.Div     -> L.build_sdiv
   | A.BitAnd  -> L.build_and
   | A.BitOr   -> L.build_or
   | A.CompNeq -> L.build_icmp L.Icmp.Ne
   | A.CompLt  -> L.build_icmp L.Icmp.Slt
   | A.CompLeq -> L.build_icmp L.Icmp.Sle
   | A.CompGt  -> L.build_icmp L.Icmp.Sgt
   | A.CompGeq -> L.build_icmp L.Icmp.Sge
   | A.CompEq  -> L.build_icmp L.Icmp.Eq
   | A.RShift  -> L.build_ashr
   | A.LShift  -> L.build_lshr
   | A.Xor     -> L.build_xor
   ) e1' e2' "tmp" builder
   | SAssign (e1, op, e2)
   -> (match e1 with
       (_,SId nm)
       -> let e1' = lookup nm
          and e2' = expr builder e2 in
         (match op with (*PlusEqual | MinusEqual | TimesEqual | DivEqual | Equal*)
            PlusEqual -> raise (Failure("special assignments need binops to be able
to work"))
          | MinusEqual -> raise (Failure("special assignments need binops to be able
to work"))
```

```
           | TimesEqual -> raise (Failure("special assignments need binops to be able
to work"))
           | DivEqual -> raise (Failure("special assignments need binops to be able to
work"))
           | Equal -> ignore(L.build_store e2' e1' builder); e1')
       | (ty, SListElement (listId,index,rest))
       -> match rest with
            []
            -> let theList = expr builder listId
               and i = expr builder index
               and e2' = allocInitElt e2 in
               let _ = L.build_call set_at_func [| theList; i; e2'|] "set_at" builder
               in theList
            | _
            -> let theList = expr builder listId in

               let lastIndex = List.hd (List.rev rest)
               and allbutLastIndex = index::(List.rev (List.tl (List.rev rest))) in
               let listToIndexInto = deRef allbutLastIndex theList
               and i = expr builder lastIndex
               and e2' = allocInitElt e2 in
               let _ = L.build_call set_at_func [| listToIndexInto; i; e2'|] "set_at"
builder
               in listToIndexInto )
    | SFunCall ("printint", [e])
    -> L.build_call printint_func [| (expr builder e) |]
       "printint" builder
    | SFunCall ("stringLength", [e]) ->
       L.build_call stringLength_func [| (expr builder e) |]
       "stringLength" builder
    | SFunCall ("stringReverse", [e]) ->
       L.build_call stringReverse_func [| (expr builder e) |]
       "stringReverse" builder
    | SFunCall ("corresponding_char", [e]) ->
       L.build_call corresponding_char_func [| (expr builder e) |]
       "corresponding_char" builder
    | SFunCall ("corresponding_int", [e;e2]) ->
    L.build_call corresponding_int_func [| (expr builder e);(expr builder e2) |]
    "corresponding_int" builder
    | SFunCall ("modulus_operation", [e;e2]) ->
    L.build_call modulus_operation_func [| (expr builder e);(expr builder e2) |]
    "modulus_operation" builder
```

```
| SFunCall ("key_test", [e;e2]) ->
L.build_call key_test_func [| (expr builder e);(expr builder e2) |]
"key_test" builder

| SFunCall ("absolute", [e]) ->
L.build_call absolute_func [| (expr builder e) |]
"absolute" builder
| SFunCall ("ceilFloat", [e]) ->
L.build_call ceilFloat_func [| (expr builder e) |]
"ceilFloat" builder
| SFunCall ("int_to_char", [e]) ->
L.build_call int_to_char_func [| (expr builder e) |]
"int_to_char" builder
| SFunCall ("characterLocation", [e;e2]) ->
L.build_call characterLocation_func [| (expr builder e); (expr builder e2) |]
"characterLocation" builder
| SFunCall ("isCompDivisible", [e]) ->
L.build_call isCompDivisible_func [| (expr builder e) |]
"isCompDivisible" builder
| SFunCall ("divisible", [e;e2]) ->
L.build_call divisible_func [| (expr builder e); (expr builder e2) |]
"divisible" builder
| SFunCall ("power", [e;e2])->
L.build_call power_func [| (expr builder e); (expr builder e2) |]
"power" builder
| SFunCall ("ascii", [e]) ->
L.build_call ascii_func [| (expr builder e) |]
"ascii" builder
| SFunCall ("very_bad_get_head", [e])
-> let arg = (expr builder e) in
   L.build_call very_bad_get_head_func [| arg |]
   "very_bad_get_head" builder
| SFunCall ("testMakeStruct",[(_,theSExpr) as arg])
-> (match theSExpr with
    SInt_lit _
    -> L.build_call testMakeStruct_func [| test_str; expr builder arg |]
        "testMakeStruct" builder
    | _
    -> L.build_call printb_func [| (expr builder arg) |]
        "printb" builder)
| SFunCall ("println",arg)
-> let _ = expr builder (theTyp,SFunCall("print",arg)) in
```

```
      L.build_call printstr_func [| newLine_str |] "printstr" builder
   | SFunCall ("print",[(typeList,_) as arg])
   -> (match typeList with
         [A.Int]
       -> L.build_call printint_func [| (expr builder arg) |]
         "printint" builder
       | [A.Char] ->
         L.build_call printchar_func [| (expr builder arg) |]
         "printchar" builder
       | [A.Bool]
       -> L.build_call printb_func [| (expr builder arg) |]
          "printb" builder
       | [A.Float]
       -> L.build_call printf_func [| float_format_str ; (expr builder arg) |]
       "printf" builder
       | [A.String]
       -> L.build_call print_list_char_func [| (expr builder arg) |]
       "print_list_char" builder
       | [A.List; A.Int]
       -> L.build_call print_list_int_func [| (expr builder arg) |]
       "print_list_int" builder
       | [A.List; A.Float]
       -> L.build_call print_list_float_func [| (expr builder arg) |]
       "print_list_float" builder
       | [A.List; A.Char]
       -> L.build_call print_list_char_func [| (expr builder arg) |]
       "print_list_char" builder
       | _
       -> raise(Failure("printing type "^A.string_of_typ_name typeList ^ "not yet
supported")))
   | SFunCall ("print_list_int", [e]) ->
       L.build_call print_list_int_func [| (expr builder e) |]
       "print_list_int" builder
   | SFunCall (f, args) ->
      let (fdef, fdecl) = StringMap.find f function_decls in
      let llargs = List.rev (List.map (expr builder) (List.rev args)) in
      let result = (match (List.hd fdecl.styp_name) with
      A.Void -> ""
      | _ -> f ^ "_result") in
      L.build_call fdef (Array.of_list llargs) result builder
   in
```

```
let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
  Some _ -> ()
  | None -> ignore (f builder) in

let rec stmt builder = function
  SBlock sl -> List.fold_left stmt builder sl
  | SExpr e -> ignore(expr builder e); builder
  | SReturn e -> ignore(match (List.hd fdecl.styp_name) with
                                         (* Special "return nothing" instr *)
                                         A.Void -> L.build_ret_void builder
                                         (* Build return statement *)
                                         | _ -> L.build_ret (expr builder e) builder
); builder
  | SWhile (predicate, body) ->
  let pred_bb = L.append_block context "while" the_function in
  ignore(L.build_br pred_bb builder);

  let body_bb = L.append_block context "while_body" the_function in
  add_terminal (stmt (L.builder_at_end context body_bb) body)
    (L.build_br pred_bb);

  let pred_builder = L.builder_at_end context pred_bb in
  let bool_val = expr pred_builder predicate in

  let merge_bb = L.append_block context "merge" the_function in
  ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
  L.builder_at_end context merge_bb

  (* Implement for loops as while loops *)
  | SFor (e1, e2, e3, body) -> stmt builder
    ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )

  | SIf (p, b) ->
      let pred_bb = L.append_block context "if" the_function in
      ignore(L.build_br pred_bb builder);

      let body_bb = L.append_block context "if_body" the_function in
        let merge_bb = L.append_block context "merge" the_function in
      add_terminal (stmt (L.builder_at_end context body_bb) b)
        (L.build_br merge_bb);
      let pred_builder = L.builder_at_end context pred_bb in
      let bool_val = expr pred_builder p in

    ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
    L.builder_at_end context merge_bb

  | SIf_Else (p, b) ->
    let if_node = (match p with SIf(lhs, rhs) -> (lhs, rhs)
                | _ -> raise(Failure("If_Else SIf block matching error"))) in

    let pred_bb = L.append_block context "if" the_function in
      ignore(L.build_br pred_bb builder);

    let body_bb = L.append_block context "if_body" the_function in
        let else_bb = L.append_block context "else_body" the_function in
          let merge_bb = L.append_block context "merge" the_function in
            add_terminal (stmt (L.builder_at_end context body_bb) (snd if_node))
              (L.build_br merge_bb);

            add_terminal (stmt (L.builder_at_end context else_bb) b)
            (L.build_br merge_bb);

      let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = expr pred_builder (fst if_node) in
          ignore(L.build_cond_br bool_val body_bb else_bb pred_builder);
          L.builder_at_end context merge_bb

  | SIf_Elif(if_stmt, fst_elif, elif_lst) ->
    let if_node = (match if_stmt with
        SIf(lhs, rhs) -> (lhs, rhs)
      | _ -> raise(Failure("If_Elif SIf block matching error"))) in
    let elif_node = (match fst_elif with
        SElif(lhs, rhs) -> (lhs, rhs)
      | _ -> raise(Failure("If_Elif SElif block matching error"))) in

    let pred_bb = L.append_block context "if" the_function in
      ignore(L.build_br pred_bb builder);

    let body_bb = L.append_block context "if_body" the_function in

      let elif_pred_bb = L.append_block context "elif" the_function in
```

```
       let elif_bb = L.append_block context "elif_body" the_function in

           let merge_bb = L.append_block context "merge" the_function in

           (* if stmt body *)
             add_terminal (stmt (L.builder_at_end context body_bb) (snd if_node))
               (L.build_br merge_bb);
           (* elif conditional *)
             let pred_builder = L.builder_at_end context elif_pred_bb in
             let elif_val = expr pred_builder (fst elif_node) in
               ignore(L.build_cond_br elif_val elif_bb merge_bb pred_builder);
           (* elif stmt body *)
             add_terminal (stmt (L.builder_at_end context elif_bb) (snd elif_node))
             (L.build_br merge_bb);

     (* let rec elif_builder elifs = function
     | s :: s ->
     | [] -> None *)

 (* if conditional *)
     let pred_builder = L.builder_at_end context pred_bb in
       let bool_val = expr pred_builder (fst if_node) in
         ignore(L.build_cond_br bool_val body_bb elif_pred_bb pred_builder);

     L.builder_at_end context merge_bb

(* SIf (predicate, then_stmt, else_stmt) ->
        let bool_val = expr builder predicate in
  let merge_bb = L.append_block context "merge" the_function in
        let build_br_merge = L.build_br merge_bb in (* partial function *)
  let then_bb = L.append_block context "then" the_function in
  add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
    build_br_merge;
  let else_bb = L.append_block context "else" the_function in
  add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
    build_br_merge;
  ignore(L.build_cond_br bool_val then_bb else_bb builder);
  L.builder_at_end context merge_bb *)

 in

 (* Build the code for each statement in the function *)
 let builder = stmt builder (SBlock fdecl.sbody) in
 (* Add a return if the last block falls off the end *)
 add_terminal builder (match (List.hd fdecl.styp_name) with
 A.Void -> L.build_ret_void
 | A.Float -> L.build_ret (L.const_float float_t 0.0)
 | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
 (* ... *)

 in
 List.iter build_function_body functions;
 the_module



(*(* <= un comment this line to see the old microc semantic analyzer with nice
colors*)
(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
 let context    = L.global_context () in
 (* Create the LLVM compilation module into which
    we will generate code *)
 let the_module = L.create_module context "MicroC" in
 (* Get types from the context *)
 let i32_t      = L.i32_type    context
 and i8_t       = L.i8_type     context
 and i1_t       = L.i1_type     context
 and float_t    = L.double_type context
 and void_t     = L.void_type   context in
 (* Return the LLVM type for a MicroC type *)
 let ltype_of_typ = function
     A.Int   -> i32_t
   | A.Bool  -> i1_t
   | A.Float -> float_t
   | A.Void  -> void_t
 in
 (* Create a map of global variables after creating each *)
 let global_vars : L.llvalue StringMap.t =
   let global_var m (t, n) =
     let init = match t with
         A.Float -> L.const_float (ltype_of_typ t) 0.0
       | _ -> L.const_int (ltype_of_typ t) 0
     in StringMap.add n (L.define_global n init the_module) m in
```

```
  List.fold_left global_var StringMap.empty globals in
let printf_t : L.lltype =
    L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func : L.llvalue =
    L.declare_function "printf" printf_t the_module in
let printbig_t : L.lltype =
    L.function_type i32_t [| i32_t |] in
let printbig_func : L.llvalue =
    L.declare_function "printbig" printbig_t the_module in
(* Define each function (arguments and return type) so we can
   call it even before we've created its body *)
let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
  let function_decl m fdecl =
    let name = fdecl.sfname
    and formal_types =
Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
    in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
  List.fold_left function_decl StringMap.empty functions in
(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.sfname function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in
  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
  and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder in
  (* Construct the function's "locals": formal arguments and locally
     declared variables.  Allocate each on the stack, initialize their
     value, if appropriate, and remember their values in the "locals" map *)
  let local_vars =
    let add_formal m (t, n) p =
      L.set_value_name n p;
let local = L.build_alloca (ltype_of_typ t) n builder in
      ignore (L.build_store p local builder);
StringMap.add n local m
    (* Allocate space for any locally declared variables and add the
     * resulting registers to our map *)
    and add_local m (t, n) =
let local_var = L.build_alloca (ltype_of_typ t) n builder
in StringMap.add n local_var m
    in
    let formals = List.fold_left2 add_formal StringMap.empty fdecl.sformals
        (Array.to_list (L.params the_function)) in
     List.fold_left add_local formals fdecl.slocals
   in
   (* Return the value for a variable or formal argument.
      Check local names first, then global names *)
   let lookup n = try StringMap.find n local_vars
                  with Not_found -> StringMap.find n global_vars
   in
   (* Construct code for an expression; return its value *)
*)
```

list.c

```c
#include <stdlib.h>
#include <stdio.h>
#include "list.h"

int init_list(list *l) {
   l->capacity = INITIAL_LIST_CAPACITY;
   l->size = 0;

   l->data = (char **) malloc(INITIAL_LIST_CAPACITY * sizeof(char *));
   if (!l->data)
       return -1; // failure

   return 0; // success
}

//return length of list
int list_length (list * l){
    return l->size;
}

// checks if list is at capacity
// ret 1 if needs resizing, ret 0 if doesn't
int check_resizing(list *l) {
   if (l->size == l->capacity) {return 1;} else {return 0;}
}

// ret 1 if empty
// ret 0 if non-empty
int check_empty(list *l) {
    if (l->size == 0) {return 1;} else {return 0;}
}

// ret 0 if successful
// ret -1 on failure
int resize(list *l) {
    l->data = (char **) realloc(l->data, l->capacity * 2 * sizeof(char *));
    if (l->data == NULL)
        return -1;
    l->capacity *= 2;
    return 0;
}

void push_back(list *l, char *item) {
    if (check_resizing(l))
        resize(l);
    // set item
    l->data[l->size] = item;
    l->size++;
}

void push_front(list *l, char *item) {
    if (check_resizing(l))
        resize(l);

    // shift everything over by 1
    for (int i=l->size; i > 0; i--) {
        l->data[i] = l->data[i-1];
    }

    l->data[0] = item;
    l->size++;
}

// ret 0 on success
// ret -1 on failure
// WARNING: WILL CAUSE MEMORY LEAKS! DATA DOES NOT GET FREE'D
int del_back(list *l) {
    if (check_empty(l))
        return -1;

    l->size--;
```

```
   return 0;
}

// ret 0 on success
// ret -1 on failure
// WARNING: WILL CAUSE MEMORY LEAKS! DATA DOES NOT GET FREE'D
int del_front(list *l) {
   if (check_empty(l))
       return -1;

   // shift over data
   for (int i=0; i < (l->size - 1); i++)
       l->data[i] = l->data[i+1];

   l->size--;
   return 0;
}

// ret 0 on success
// ret -1 on failure
// WARNING: WILL CAUSE MEMORY LEAKS! DATA DOES NOT GET FREE'D
int del_at(list *l, int i) {
   if (check_empty(l))
       return -1;

   // shift over data
   for (int j=i; j < (l->size - 1); j++)
       l->data[j] = l->data[j+1];

   l->size--;
   return 0;
}

char *get_back(list *l) {
   if (l->size == 0)
       return NULL; // list is empty

   // offset data pointer by size of array
   return l->data[l->size - 1];
}

char *get_front(list *l) {
   if (l->size == 0)
       return NULL; // list is empty

   // get index 0, ie. first element
   return l->data[0];
}

char *get_at(list *l, int i) {
   if (check_empty(l))
       return NULL;

   // check if index is within bounds of array
   if (i < l->size && i >= 0)
       return l->data[i];

   return NULL;
}

list* set_at(list *l, int i, char* val) {
   if (check_empty(l))
       return NULL;

   // check if index is within bounds of array
   if (i < l->size && i >= 0){
       l->data[i] = val;
       return l;
   }
   return NULL;
}

void print_list_int(list *l) {
   printf("[");
   for (int i=0; i<(l->size)-1; i++){
       printf("%d, ", *(int *)l->data[i]);
   }
   printf("%d]", *(int *)l->data[(l->size)-1]);
}

void print_list_float(list *l) {
   printf("[");
   for (int i=0; i<(l->size)-1; i++){
       printf("%f, ", *(double *)l->data[i]);
```

```
    }
    printf("%f]", *(double*)l->data[(l->size)-1]);
}

void print_list_char(list *l) {
    for (int i=0; i<(l->size)-1; i++){
        printf("%c", *l->data[i]);
    }
    printf("%c", *l->data[(l->size)-1]);
}

#ifdef BUILD_TEST
int main() {
    list l;
    init_list(&l);

    int a = 0;
    int *p = &a;

    for (int i=0; i<10; i++) {
        p = (int *) malloc(sizeof(int));
        *p = a;
        printf("pushing: %d\n", *p);
        push_back(&l, (char *) p);
        a++;
    }

    del_front(&l);
    del_front(&l);
    del_front(&l);
    printf("%d\n", *((int *) get_front(&l)));

    return 0;
}
#endif
```

prob.c

```
// implementation of the probability data structure
// WARNING: data does not get freed --> memory leaks galore
// NOTE: underlying probability buckets have 5 points of precision
//       --> this value is chosen to suit most needs and is used to have
//        --> proper rounding and so sum(probs) == 1.0
//        --> allows to thousandth percent: .001% or .00001

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "list.h"
#include "prob.h"

double precision(double num, int precision) {
   return floor(pow(10,precision)*num)/pow(10,precision);
}

// return -1 on error
// return 0 on success
int init_prob(prob *p, list *gprobs, list *gdata) {
   // min len of both lists
   int minlen = min(gprobs->size, gdata->size);

   // malloc both arrays in prob --> these are copies
   p->probs.data = (char **) malloc(sizeof(char *) * minlen);
   if (p->probs.data == NULL)
       return -1;
   p->vals.data = (char **) malloc(sizeof(char *) * minlen);
   if (p->vals.data == NULL)
       return -1;
   // set size and capacity to minlen
   p->probs.size = minlen; p->vals.size = minlen;
   p->probs.capacity = minlen; p->vals.capacity = minlen;

   // store this in struct so easy to pull out later
   p->length = minlen;

   // zip and copy
   for (int i = 0; i < minlen; i++) {
       p->probs.data[i] = gprobs->data[i];
       p->vals.data[i] = gdata->data[i];
   }

   // check for negative values
   for (int i = 0; i < minlen; i++) {
       if (prob_at(p, i) < 0)
           return -1;
   }

   // normalize
   normalize(p);

   // seed rng
   srand48(time(NULL));

   return 0;
}

void normalize(prob *p) {
   // cannot normalize 0 vector
   if (check_nonzero(p) == 0)
       return;

   // calculate magnitude
   double sum = 0;
   for (int i=0; i < p->length; i++) {
       sum += prob_at(p, i);
   }

   // divide each element by sum and round
   for (int i=0; i< p->length; i++)
       prob_at(p, i) = precision(prob_at(p, i)/sum, PROB_PRECISION);

   // last element will be automatically calculated to ensure sum(n) == 1.0
   // WARNING: slight error is allowed
   double sum_of_probs = 0;
   for (int i=0; i < p->length-1; i++)
       sum_of_probs += prob_at(p, i);
   prob_at(p, (p->length - 1)) = (1.0 - sum_of_probs);
}

// checks that probability bucket vector is non-zero
// ret 0 if zero
// ret 1 if non-zero
int check_nonzero(prob *p) {
   for (int i = 0; i < p->length; i++) {
       if (prob_at(p, i) != 0)
```

```
            return 1;
    }
    return 0;
}

list *get_probs(prob *p) { return &p->probs; }

list *get_vals(prob *p) { return &p->vals; }

int get_length(prob *p) { return p->length; }

// returns a value based on the probability distribution
char *peek(prob *p) {
    double rnd = precision(drand48(), PROB_PRECISION);

    #ifdef debug
    // printf("rnd: %f\n", rnd);
    #endif

    // given: [.25, .5, .25]
    // intervals: -->edge case w/ 0 : [0, .25], (.25, .75], (.75, 1.0]
    // which interval is rnd in?

    for (int i = 0; i < p->length; i++) {
        double min_interval = 0;
        double max_interval = 0;

        // for min_interval
        for (int j=0; j < i; j++)
            min_interval += prob_at(p, j);

        // for max_interval
        for (int j=0; j < (i+1); j++)
            max_interval += prob_at(p,j);

        #ifdef debug
        // printf("i=%d\n(%f, %f]\n\n", i, min_interval, max_interval);
        #endif

        // edge case for i==0
        if (i==0) {
            if (rnd >= min_interval && rnd <= max_interval)
```

```
                return p->vals.data[i];
        }
        else{
            if (rnd > min_interval && rnd <= max_interval)
                return p->vals.data[i];
        }
    }

    // error
    fprintf(stderr, "probability peek error: no matching value found.\n");
    int val_0 = *(int*)p->vals.data[0];
    double probs_0 = *(double*)p->probs.data[0];
    fprintf(stderr, "Rand is %f. Length is %d. \nprobs[0] is %f. vals[0] is %d.",rnd,p-
>length, probs_0,val_0);
    return NULL;
}

int print_prob_int_debug(prob* p){
    printf("Prob's length is %d\n",p->length);
    for(int i =0;i<3;i++){
        printf("probs[%d] = %f\n",i, *(double*)p->probs.data[i]);
        printf("vals[%d] = %d\n",i, *(int*)p->vals.data[i]);
    }
    return 0;
}

// changes the probabilities of the target argument
void add_probs(prob *target, prob *p2) {
    int minlen = min(target->length, p2->length);
    for (int i = 0; i < minlen; i++)
        prob_at(target, i) = prob_at(target, i) + prob_at(p2, i);

    normalize(target);
}

// changes the probabilities of the target argument
void sub_probs(prob *target, prob *p2) {
    int minlen = min(target->length, p2->length);
    for (int i = 0; i < minlen; i++)
        prob_at(target, i) = prob_at(target, i) - prob_at(p2, i);

    normalize(target);
```

```
}

// changes the probabilities of the target argument
void times_probs(prob *target, prob *p2) {
   int minlen = min(target->length, p2->length);
   for (int i = 0; i < minlen; i++)
       prob_at(target, i) = prob_at(target, i) * prob_at(p2, i);

   normalize(target);
}

// changes the probabilities of the target argument
void div_probs(prob *target, prob *p2) {
   int minlen = min(target->length, p2->length);
   for (int i = 0; i < minlen; i++)
       prob_at(target, i) = prob_at(target, i) / prob_at(p2, i);

   normalize(target);
}

// TODO: return pointer to malloced list of the values that
// satisfy the condition
char *get_values_by_range(prob *p, int min, int max) {
   return NULL;
}

#ifdef BUILD_TEST
int main() {
   list l1;
   init_list(&l1);
   list l2;
   init_list(&l2);

   prob prb;
   int a=5;
   int *p;
   double *d;

   puts("probabilities");

   d = (double *) malloc(sizeof(double));
   *d = .25;
```

```
printf("pushing: %f\n", *d);
push_back(&l1, (char *) d);

d = (double *) malloc(sizeof(double));
*d = .5;
printf("pushing: %f\n", *d);
push_back(&l1, (char *) d);

d = (double *) malloc(sizeof(double));
*d = .25;
printf("pushing: %f\n", *d);
push_back(&l1, (char *) d);

puts("values");

for (int i=0; i<3; i++) {
    p = (int *) malloc(sizeof(int));
    *p = a;
    printf("pushing: %d\n", *p);
    push_back(&l2, (char *) p);
    a++;
}

init_prob(&prb, &l1, &l2);
// print_list_int(&prb.probs);
// print_list_int(&prb.vals);

if(check_nonzero(&prb))
    printf("non zero!\n");

int x; int y; int z;
x = y = z = 0;
for(int i=0; i< 100000; i++) {
    int num = *(int *) peek(&prb);
    if (num == 5)
        x++;
    else if (num == 6)
        y++;
    else if (num == 7)
        z++;
}
```

```
   printf("probs: ");
   print_probs(&prb);

   printf("values: ");
   print_list_int(&prb.vals);

   printf("results: 5: %d // 6: %d // 7: %d\n", x, y, z);

   // printf("%d\n", *(int *) peek(&prb));

   return 0;
}
#endif
```

polymorphicPrint.c

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
// void printstr(const char *str)
// {
//  puts(str);
// }
int all[17576][3];
int first = 0;
int l = 0;

void printb(int x)
{
   if(x){
       printf("true");
   }
   else{
       printf("false");
   }

}
```

```
void printint(int x)
{
   printf("%d\n", x);
}
void printstr(char* x)
{
   printf("%s", x);
}
```

```
//Methods for enigma
int corresponding_int(list* x, int index) {
   char* current_elem = get_at(x,index);
   int ascii_val = (int) (*current_elem);
   return ascii_val - 65;
}
void corresponding_char(int number) {
   char element = (number + 65);
   printf("%c", element);
}
char current_char(int number) {
   char element = (number + 65);
   return element;
}
void initialize_potential_settings() {
   for(int i=0;i<26;i++){
       for(int j=0;j<26;j++){
           for(int k=0;k<26;k++){
               all[l][0] = i;
               all[l][1] = j;
               all[l][2] = k;
               l++;
           }
       }
   }
}
int key_test(int number, int index) {
   if(first==0) {
       initialize_potential_settings();
       first=first+1;

   }

   return all[number][index];


}
int modulus_operation(int x, int y) {
   return x%y;
}
//Methods defined in LRM
int stringLength(list* x) {
 return x->size;

}
int absolute(int x) {
   if (x < 0) {
       return -x;
   }
   return x;
}
void stringReverse(list* x) {
   int len = stringLength(x);
   for (int j = len-1; j > -1; j--) { //Not efficient but my code with recurstion was
failing idk why :-(
       char* temp = get_at(x,j);
       char res = *(temp);
       printf("%c", res);
   }
}
int ascii(list* x) {
   char* current_elem = (get_at(x,0));
   int ascii_val = (int) *current_elem;

   return ascii_val;
}
list* int_to_char(int x) {

   char* z = malloc(1);
   *z = x;
   list *str = malloc(sizeof(struct list));
   push_back(str,z);
   return str;
}
int ceilFloat(double x) {

   int y = (int) x;
   float z = x - y;
   if(z >= 0.5) {
       return y+1;
   }
   return y;
}
int characterLocation(list* x, list* y) {
   int length_x = stringLength(x);
   char* first_element;
   char* second_element = get_at(y,0);
   for(int i=0; i < length_x;i++) {
       first_element = get_at(x,i);
       if(*first_element == *second_element) {
           return i;
       }
   }
   return -1;
}
int isCompDivisible(int x) {
   if(x%2 == 0) {
       return 1;
   }
   return 0;
}
int divisible(int x, int y) {
   if(x < y) {
       return 0;
   }
   if(x%y == 0) {
       return 1;
   }
   return 0;
}
int power(int x, int y) { //Exponentiation by squaring
   int squared = 1;
   for (;;)
   {
       if (y & 1)
           squared *= x;
       y >>= 1;
       if (!y)
           break;
       x *= x;
   }
   return squared;
}
void printchar(char c){
   printf("%c",c);
}
#ifdef BUILD_TEST
int main()
{
   printstr("42");
   return 0;
}
#endif
```

---

testall.sh

```
#!/bin/sh

# Regression testing script for SMAP
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the microc compiler.  Usually "./microc.native"
# Try "_build/microc.native" if ocamlbuild was unable to create a symbolic link.
SMAP="./smap.native"
#SMAP="_build/smap.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
   echo "Usage: testall.sh [options] [.smap files]"
   echo "-k    Keep intermediate files"
   echo "-h    Print this help"
   exit 1
}

SignalError() {
   if [ $error -eq 0 ] ; then
   echo "FAILED"
   error=1
   fi
   echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
   generatedfiles="$generatedfiles $3"
   echo diff -b $1 $2 ">" $3 1>&2
   diff -b "$1" "$2" > "$3" 2>&1 || {
   SignalError "$1 differs"
   echo "FAILED $1 differs from $2" 1>&2
   }
}

# Run <args>
# Report the command, run it, and report any errors
```

```
Run() {
   echo $* 1>&2
   eval $* || {
   SignalError "$1 failed on $*"
   return 1
   }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
   echo $* 1>&2
   eval $* && {
   SignalError "failed: $* did not report an error"
   return 1
   }
   return 0
}

Check() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                            s/.mc//'`
   reffile=`echo $1 | sed 's/.mc$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""

   generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
   Run "$SMAP" "$1" ">" "${basename}.ll" &&
   Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
   Run "$CC" "-o" "${basename}.exe" "${basename}.s" "polymorphicPrint.o" "prob.o"
"list.o" "testMakeStruct.o" "-lm" "-lncurses" &&
   Run "./${basename}.exe" > "${basename}.out" &&
   Compare ${basename}.out ${reffile}.out ${basename}.diff
```

```
    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}

CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.mc//'`
    reffile=`echo $1 | sed 's/.mc$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$SMAP" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}

while getopts kdpsh c; do
    case $c in
    k) # Keep intermediate files
        keep=1
        ;;
    h) # Help
        Usage
        ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
 echo "Could not find the LLVM interpreter \"$LLI\"."
 echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
 exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ $# -ge 1 ]
then
    files=$@
else
    files="test/test-*.smap test/fail-*.smap"
fi

for file in $files
do
    case $file in
    *test-*)
        Check $file 2>> $globallog
        ;;
    *fail-*)
        CheckFail $file 2>> $globallog
        ;;
    *)
```

```
       echo "unknown file type $file"
       globalerror=1
       ;;
   esac
done

exit $globalerror
```