Democritus Language Final Report

Amy Xu
xx2152
Project Manager

Emily Pakulski
enp2111
Language Guru

Amarto Rajaram
aar2160
System Architect

Kyle Lee
kpl2111
Tester

May 11, 2016
# Contents

1. **Introduction** ........................................... 7
   1.1 Motivation and Product Goals .......................... 7
   Native Concurrency ........................................ 7
   Portability ............................................... 7
   Flexibility .............................................. 7

2. **Language Tutorial** .................................... 8
   2.1 Setup and Installation ................................ 8
   2.2 Compiling Your Code .................................. 8
   2.3 Writing Code ......................................... 9
   2.4 Getting Started ...................................... 9
       Declarations .......................................... 9
       Types .................................................. 9
       Primitives ............................................ 9
       Strings ............................................... 9
       Structs ............................................... 10
       Operators ............................................ 10
       Binary Operators: ................................... 10
       Unary Operators: .................................... 10
   2.5 Control Flow ......................................... 11
       Conditional Branching ............................... 11
       Loops and Iteration .................................. 11
   2.6 Multithreading ....................................... 12
   2.7 Miscellaneous ...................................... 12
       Malloc .............................................. 12
       Pointers ............................................. 12
       File I/O ............................................. 13
       Sockets API ......................................... 13

3. **Language Reference Manual** .......................... 15
   3.1 Introduction .......................................... 15
   3.2 Structure of a Democritus Program .................. 15
   3.3 Data types ........................................... 17
       Primitive Types ...................................... 17
       int ................................................... 17
       float ............................................... 17
       boolean ............................................ 17
       pointer ............................................. 17
       Complex Types ...................................... 17
       string .............................................. 17
3.9 Concurrency .......................................................... 27
Overview .................................................................. 27
Spawning Threads ..................................................... 27

4 Project Plan .............................................................. 28
4.1 Planning ................................................................ 28
4.2 Workflow .............................................................. 28
4.3 Team Member Responsibilities ................................. 29
4.4 Git Logs .................................................................. 30
master branch ......................................................... 30
threads-2 branch ....................................................... 40
nested-structs branch ................................................. 45
linkedlist-and-stack branch ....................................... 51
fix-malloc branch ..................................................... 62
build-malloc-attempt branch ..................................... 70
Final Report .............................................................. 80

5 Architecture Overview ............................................. 82
5.1 Compiler Overview ................................................ 83
The Scanner ............................................................. 83
The Parser ............................................................... 83
The Semantic Analyzer ............................................. 83
The Code Generator .................................................. 83

6 Testing ................................................................. 84
6.1 Integration Testing ............................................... 84
Development and Testing Process .............................. 84
Aside: Unit Testing .................................................. 85
6.2 The Test Suite and Automated Regression Testing ...... 85

7 Lessons Learned ..................................................... 86
7.1 Amy ..................................................................... 86
7.2 Emily ................................................................. 86
7.3 Amarto ............................................................... 86
7.4 Kyle .................................................................... 86

8 Code Listing ........................................................... 87
8.1 democritus.ml ....................................................... 87
8.2 scanner.ml .......................................................... 87
8.3 parser.mly ........................................................... 89
8.4 semant.ml ............................................................ 91
8.5 ast.ml ................................................................. 97
8.6 codegen.ml .......................................................... 100
8.7 bindings.c ............................................................ 109

9 Tests and Output ..................................................... 114
fail-assign1.dem ........................................................ 114
fail-assign1.err ........................................................ 114
fail-assign2.dem ........................................................ 114
fail-assign2.err ........................................................ 114
fail-assign3.dem ........................................................ 114
fail-assign3.err ........................................................ 115

3
<table>
<thead>
<tr>
<th>File Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fail-nomain.err</td>
<td>122</td>
</tr>
<tr>
<td>fail-return1.dem</td>
<td>122</td>
</tr>
<tr>
<td>fail-return1.err</td>
<td>122</td>
</tr>
<tr>
<td>fail-return2.dem</td>
<td>123</td>
</tr>
<tr>
<td>fail-return2.err</td>
<td>123</td>
</tr>
<tr>
<td>fail-struct-circular.dem</td>
<td>123</td>
</tr>
<tr>
<td>fail-struct-circular.err</td>
<td>123</td>
</tr>
<tr>
<td>test-arith1.dem</td>
<td>123</td>
</tr>
<tr>
<td>test-arith1.out</td>
<td>124</td>
</tr>
<tr>
<td>test-arith2.dem</td>
<td>124</td>
</tr>
<tr>
<td>test-arith2.out</td>
<td>124</td>
</tr>
<tr>
<td>test-arith3.dem</td>
<td>124</td>
</tr>
<tr>
<td>test-arith3.out</td>
<td>124</td>
</tr>
<tr>
<td>test-fib.dem</td>
<td>124</td>
</tr>
<tr>
<td>test-fib.out</td>
<td>125</td>
</tr>
<tr>
<td>test-fileops.dem</td>
<td>125</td>
</tr>
<tr>
<td>test-fileops.out</td>
<td>125</td>
</tr>
<tr>
<td>test-float.dem</td>
<td>125</td>
</tr>
<tr>
<td>test-float.out</td>
<td>125</td>
</tr>
<tr>
<td>test-for1.dem</td>
<td>126</td>
</tr>
<tr>
<td>test-for1.out</td>
<td>126</td>
</tr>
<tr>
<td>test-for2.dem</td>
<td>126</td>
</tr>
<tr>
<td>test-for2.out</td>
<td>126</td>
</tr>
<tr>
<td>test-for-as-while1.dem</td>
<td>126</td>
</tr>
<tr>
<td>test-for-as-while1.out</td>
<td>127</td>
</tr>
<tr>
<td>test-func1.dem</td>
<td>127</td>
</tr>
<tr>
<td>test-func1.out</td>
<td>127</td>
</tr>
<tr>
<td>test-func2.dem</td>
<td>127</td>
</tr>
<tr>
<td>test-func2.out</td>
<td>127</td>
</tr>
<tr>
<td>test-func3.dem</td>
<td>128</td>
</tr>
<tr>
<td>test-func3.out</td>
<td>128</td>
</tr>
<tr>
<td>test-func4.dem</td>
<td>128</td>
</tr>
<tr>
<td>test-func4.out</td>
<td>128</td>
</tr>
<tr>
<td>test-func5.dem</td>
<td>128</td>
</tr>
<tr>
<td>test-func5.out</td>
<td>129</td>
</tr>
<tr>
<td>test-gcd2.dem</td>
<td>129</td>
</tr>
<tr>
<td>test-gcd2.out</td>
<td>129</td>
</tr>
<tr>
<td>test-gcd.dem</td>
<td>129</td>
</tr>
<tr>
<td>test-gcd.out</td>
<td>129</td>
</tr>
<tr>
<td>test-global1.dem</td>
<td>129</td>
</tr>
<tr>
<td>test-global1.out</td>
<td>130</td>
</tr>
<tr>
<td>test-global2.dem</td>
<td>130</td>
</tr>
<tr>
<td>test-global2.out</td>
<td>130</td>
</tr>
<tr>
<td>test-hello.dem</td>
<td>130</td>
</tr>
<tr>
<td>test-hello.out</td>
<td>131</td>
</tr>
<tr>
<td>test-helloworld-assign.dem</td>
<td>131</td>
</tr>
<tr>
<td>test-helloworld-assign.out</td>
<td>131</td>
</tr>
<tr>
<td>test-helloworld.dem</td>
<td>131</td>
</tr>
<tr>
<td>test-helloworld.out</td>
<td>131</td>
</tr>
<tr>
<td>test-if1.dem</td>
<td>131</td>
</tr>
<tr>
<td>test-if1.out</td>
<td>131</td>
</tr>
</tbody>
</table>
test-if2.dem .................................................. 131
test-if2.out .................................................. 131
test-if3.dem .................................................. 132
test-if3.out .................................................. 132
test-if4.dem .................................................. 132
test-if4.out .................................................. 132
test-linkedlist-final.dem ...................................... 132
test-linkedlist-final.out ...................................... 132
test-linkedlist-malloc.dem .................................... 133
test-linkedlist-malloc.out .................................... 133
test-linkedlist-proof.dem ..................................... 133
test-linkedlist-proof.out ..................................... 135
test-local1.dem ............................................... 134
test-local1.out ............................................... 134
test-mod.dem .................................................. 135
test-mod.out .................................................. 135
test-ops1.dem .................................................. 135
test-ops1.out .................................................. 135
test-ops2.dem .................................................. 136
test-ops2.out .................................................. 137
test-pointer-bool.dem ........................................ 137
test-pointer-bool.out ........................................ 137
test-pointer-int.dem .......................................... 138
test-pointer-int.out .......................................... 138
test-pointer-malloc.dem ...................................... 138
test-pointer-malloc.out ...................................... 138
test-pointer-struct-onevl.dem ................................ 139
test-pointer-struct-onevl.out ................................ 139
test-pointer-struct-twelvl.dem ............................... 140
test-pointer-struct-twelvl.out ............................... 140
test-sleep.dem ................................................ 140
test-sleep.out ................................................ 140
test-struct1.dem .............................................. 140
test-struct1.out .............................................. 140
test-struct.dem ............................................... 141
test-struct.out ............................................... 141
test-structs-nested1.dem .................................... 141
test-structs-nested1.out .................................... 141
test-structs-nested.dem .................................... 142
test-structs-nested.out .................................... 142
test-threading1.dem .......................................... 142
test-threading1.out .......................................... 142
test-threading2.dem .......................................... 142
test-threading2.out .......................................... 142
test-var1.dem .................................................. 142
test-var1.out .................................................. 142
1. Introduction

Democritus is a programming language with a static type system and native support for concurrent programming, with facilities for both imperative and functional programming. Democritus is compiled to the LLVM (Low Level Virtual Machine) intermediate form, which can then be optimized to machine-specific assembly code. Democritus’ syntax draws inspiration from contemporary languages, aspiring to emulate Go and Python in terms of focusing on use cases familiar to the modern software engineer, emphasizing readability, and having “one – and preferably only one – obvious way to do it”\(^1\).

1.1 Motivation and Product Goals

Native Concurrency

Users should be able to easily and quickly thread their program. Their development process should not be hindered by the use of multithreading, nor should they have to define special threading classes as is common in some other languages.

Portability

Developed under the LLVM IR, code written in Democritus can be compiled and run on any machine that LLVM can run on. As an industrial-level compiler, LLVM offers robustness and portability as the compiler back-end of Democritus.

Flexibility

Though Democritus is not an object-oriented language, it seeks to grant users flexibility in functionality, supporting structures, standard primitive data types, native string support, and pointers for precise memory control.

\(^1\)http://c2.com/cgi/wiki?PythonPhilosophy
2. Language Tutorial

Democritus is a statically-typed, imperative language with standard methods for conditional blocks, iteration, variable assignment, and expression evaluation. In this chapter, we will cover environment configuration as well as utilizing both Democritus' basic and advanced features.

2.1 Setup and Installation

To set up the Democritus compiler, OCaml and LLVM must be installed. Testing and development was done in both native Ubuntu 15.04 and Ubuntu 14.04 running on a virtual machine.

For Ubuntu 15.04, we need the matching LLVM 3.6 OCaml Library.

```bash
sudo apt-get install m4 clang-3.7 clang clang-3.7-doc libclang-common-3.7-dev libclang-3.7-dev libclang1-3.7 libclang1-3.7-dbgl libllvm-3.7-ocaml-dev libllvm3.7 libllvm3.7-dbgl
sudo apt-get install runtime clang-modernize-3.7 clang-format-3.7 python-clang-3.7 llvdb-3.7-dev liblldb-3.7-dbg liblldb-3.7 dbg lldb
```

For Ubuntu 14.04:

```bash
sudo apt-get install m4 llvm software-properties-common
sudo add-apt-repository --yes ppa:avsm/ppa
sudo apt-get update –qq
sudo apt-get install --y opam
opam init
opam install llvm.3.4 ocamlfind
```

After setting up the environment, clone the git repository into your desired installation directory:

```
git clone https://github.com/DemocritusLang/Democritus.git
```

2.2 Compiling Your Code

To build the compiler, cd into the Democritus repository, and run make.
If building fails, try running `eval 'opam config env'`, which should update your local environment use OPAM packages and compilers. It's recommended to add the above command to your shell's configuration file if you plan on developing with Democritus.

To compile code, simply run

```bash
./Democritus < filename.dem > outfile.lli
```

To run compiled code, call `lli` on the output:

```bash
lli outfile.lli
```

## 2.3 Writing Code

Code can be written in any text file, but Democritus source files should have the `.dem` extension by convention. Democritus programs consist of global function, struct, and variable declarations. Only the code inside `main()` will be executed at runtime. At this time, linking is not included in the Democritus compiler; all code should be written and compiled from a single `.dem` source file.

## 2.4 Getting Started

### Declarations

Functions are declared with the `<function func_name(a type, b type) return_type> syntax. Variables are declared with the `<let var_name var_type;` syntax. Statements are terminated with the semicolon `;`. Note that all variable declarations must happen before statements (including assignments) in any given function.

```c
function triangle_area(base int, height int) int{
  return base*height/2;
}
```

### Types

#### Primitives

Primitive types in Democritus include booleans and integers. The `void` type is also used for functions.

#### Strings

Strings are built-in to Democritus. String literals are added to global static memory at runtime, and string variables point to the literals. These literals are automatically null-terminated.

```c
function main() int{
  let s string;
  let foo int;
  let bar bool;

  bar = true;
  s = "Hello, World!"
  foo = 55;
  bar = false;

  return 0;
}
```
**Structs**

Structs are declared at the global level with the `<struct struct_type { named fields }>` syntax. Struct declarations may also be nested.

```plaintext
1 struct Person{
2   let name string;
3   let age int;
4   let info struct Info;
5 }
6
7 struct Info{
8   let education string;
9   let salary int;
10 }
11
12 function main() int{
13    let p struct Person;
14    p.name = "Joe";
15    p.age = 30;
16    p.info.education = "Bachelor’s";
17    p.info.salary = 99999;
18    print(p.name);
19    print(" earns: ");
20    print_int(p.info.salary);
21    return 0;
22 }
```

**Operators**

Democritus includes the ‘standard’ set of operators, defined as follows:

**Binary Operators:**
- arithmetic: +, -, *, /, %
- logical: ==, !=, <, <=, >, >=, && (and), || (or)

**Unary Operators:**
- arithmetic: -
- logical: ! (not)
- addressing: & (reference), * (reference)

Logical expressions return a boolean value.

The expressions on each side of a binary operation must be of the same type. The &&, ||, and ! operators must be called on boolean expressions.

References can only be called on addressable fields (such as variables, or struct fields). Dereferences can only be called on pointer types.
2.5 Control Flow

As an imperative language, Democritus executes statements sequentially from the top of any given function to the bottom. Branching and iteration is done similarly to many other imperative languages.

Conditional Branching

Conditional branching is done with:

```plaintext
  if(boolean expression)
  {
    /* do something here */
  }
  else
  {
    /* do alternative here */
  }
```

Here is an example of conditional branching in Democritus:

```plaintext
struct Person{
  let education string;
  let name string;
  let age int;
  let working bool;
}

function main() int{
  let p Struct person;
  p.name = "Joe"
  p.education = "Bachelor’s"
  p.age = 25;
  p.working = false;

  if(p.working){
    print(p.name);
    print(" works.\n");
  }else{
    print(p.name);
    print(" is looking for work.\n");
  }

  return 0;
}
```

This program prints “Joe is looking for work.”

Loops and Iteration

Iteration can be done either via a for loop. A for(e1; e2; e3) loop may take three expressions; e1 is called prior to entering the loop, e2 is a boolean conditional statement for the loop, and e3 is called after each iteration. Both e1 and e3 are optional; omitting both converts the for loop into the conditional while used in other languages.
```plaintext
function main() int{
  let i int;
  for(i = 0; i<42; i=i+1){
    i = i+1;
  }
  for(;false;){
    // This block will never be reached
  }
  for(true){
    print_int(i);
  }
  return 0;
}
```

This program will print 42 forever.

### 2.6 Multithreading

Democritus supports threading with the `thread()` function call, which then calls the underlying `pthread` function in C. Any defined function can be called with multiple threads. The calling syntax is as follows:

```plaintext
thread("functionname", args, numthreads);
```

Multithreaded functions must take a `*void` type as input and return a `*void` to conform with C’s calling convention. An example of a multithreaded program:

```plaintext
function multiprint(noop *void) *void{
  let x *void;
  print("Hello, World!\n");
  return x;
}

function main() int{
  thread("multiprint", 0, 6); /* "Hello, World!" will be printed six times. */
  return 0;
}
```

### 2.7 Miscellaneous

Besides threading, a couple of other functions from C have been bound to Democritus.

**Malloc**

`malloc(size)` may be called, returning a pointer to a newly heap-allocated block of `size` bytes. These pointers may be bound to strings, which themselves are pointers to string literals. An example utilizing `malloc` will be included with file I/O.

**Pointers**

Democritus features a pointer type, a numerical value which points to an address in memory.

A pointer type is defined `<* type>`, and as such they can be declared as follows: `let a *int;`
Dereferencing is performed with * (e.g. *a), and may only be performed on pointer types. Referencing, which returns the memory address of a variable or addressable location in memory, is performed with &a.

Since malloc returns a *void type, it must be casted in order to be used or accessed as another type. To cast an a malloc’d pointer:

```c
let i *int;
let malloced = cast malloc(num_bytes) to *int;
```

With pointers and heap-allocated memory from malloc(), we are able to build up basic data structures.

**File I/O**

Files may be opened with open(). This call returns an integer, which may be then bound as a file descriptor. C functions such as write(), read(), or lseek() may then be called on the file descriptor.

- **open(filename, fd, fd2)**: opens a file. filename is a string referring to a file to be opened. Fd and fd2 are file descriptors used for open.
- **write(fd, text, length)**: writes to a file. fd refers to the file descriptor of an open file. text is a string representing text to be written. length is an integer specifying the number of bytes to be written.
- **read(fd, buf, length)**: reads from a file. fd refers to the file descriptor of an open file. buf is a pointer to malloc’d or allocated space. length is an integer representing amount of data to be read. Buffers should be malloc’d before reading.
- **lseek(fd, offset, whence)**: sets a file descriptors cursor position. fd is the file descriptor to an open file. offset is an integer describing how many bytes the cursor should be offset by, and whence is an integer describing how offset should be applied: as an absolute location, relative location to the current cursor, or relative location to the end of the file.

For more detailed information on these calls, run man function_name.

```c
function main() int{
  let fd int;
  let malloced string;

  fd = open("tests/HELLOOOOOO.txt", 66, 384);  /* Open this file */
  write(fd, "hellooo!\n", 10);           /* Write these 10 bytes */

  malloced = malloc(10);       /* Allocate space for the data and null terminator */
  lseek(fd, 0, 0);             /* Jump to the front of the file */
  read(fd, malloced, 10);     /* Read the data we just wrote into the buffer */

  print(malloced);            /* Prints "hellooo!\n" */
  return 0;
}
```

**Sockets API**

Democritus provides support for networking functionality through use of the C sockets API. A bound C function, request_from_server, allows a user to retrieve the contents of a webpage, written to a file, as follows:
function main() int
{
    let fd int; // the file descriptor
    request_from_server("www.xkcd.com/index.html"); // write the content of the page to "index.html"
    exec_prog("/bin/cat", "cat", "tests/index.html"); // cat the file to stdout, used for testing
    return 0;
}

3.1 Introduction

In this language reference manual, Democritus, its syntax, and underlying operating mechanisms will be documented. In the grammars shown in this reference manual, all terminals are expressed in uppercase and all nonterminals are kept lowercase. The Lexical Conventions section will detail terminals (also known as tokens).

3.2 Structure of a Democritus Program

A basic Democritus program reduces to a list of global variable, struct, and function declarations. Code ‘to be executed’ should be written in functions. These declarations are accessible and usable from any scope in a Democritus program. At runtime, the function main() will be executed.

The full grammar of a program is as follows:

```
program:
  decls EOF

decls:
  /* nothing */
  | decls vdecl
  | decls fdecl
  | decls sdecl

fdecl:
  FUNCTION ID LPAREN formals_opt RPAREN typ LBRACE vdecl_list stmt_list RBRACE

formals_opt:
  /* nothing */
  | formal_list

formal_list:
  ID typ
  | formal_list COMMA ID typ

typ:
  INT
  | FLOAT
  | BOOL
  | VOID
```
vdecl_list:
    /* nothing */
    | vdecl_list vdecl

vdecl:
    LET ID typ SEMI

sdecl:
    STRUCT ID LBRACE vdecl_list RBRACE

stmt_list:
    /* nothing */
    | stmt_list stmt

stmt:
    expr SEMI
    | RETURN SEMI
    | RETURN expr SEMI
    | LBRACE stmt_list RBRACE
    | IF LPAREN expr RPAREN stmt %prec NOELSE
    | IF LPAREN expr RPAREN stmt ELSE stmt
    | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
    | FOR LPAREN expr RPAREN stmt

eexpr_opt:
    /* nothing */
    | expr

expr:
    LITERAL
    | FLOATLITERAL
    | TRUE
    | FALSE
    | ID
    | STRING
    | expr PLUS expr
    | expr MINUS expr
    | expr STAR expr
    | expr DIVIDE expr
    | expr MOD expr
    | expr EQ expr
    | expr NEQ expr
    | expr LT expr
    | expr LEQ expr
    | expr GT expr
    | expr GEQ expr
3.3 Data types

Primitive Types

int
A standard 32-bit two’s-complement signed integer. It can take any value in the inclusive range (-2147483648, 2147483647).

float
A 64-bit floating precision number, represented in the IEEE 754 format.

boolean
A 1-bit true or false value.

pointer
A 64-bit pointer that holds the value to a location in memory; pointers may be passed and dereferenced.

Complex Types

string
An immutable array of characters, implemented as a native data type in Democritus. Pointers variables are 8-bit pointers to the location of the string literal in the global static memory.

struct
A struct is a simple user-defined data structure that holds various data types, such as primitives, other structs, or pointers.
3.4 Lexical Conventions

In this subsection, we will cover the standard lexical conventions for Democritus. Lexical elements are scanned as ‘tokens,’ which are then parsed into a valid Democritus program. Democritus is a free-format language, discarding all whitespace characters such as ‘ ’, \t, and \n.

Identifiers

Identifiers for Democritus will be defined as follows: any sequence of letters and numbers without whitespaces and not a keyword will be parsed as an identifier. Identifiers must start with a letter, but they may contain any lowercase or uppercase ASCII letter, numbers, and the underscore ‘_’. Identifiers are case-sensitive, so ‘var1’ and ‘Var1’ would be deemed separate and unique. Identifiers are used to identify named elements, such as variables, struct fields, and functions. Note that identifiers cannot begin with a number. The following is a regular expression for identifiers:

\[ \text{ID} = \"[a-zA-Z][a-zA-Z0-9_]\*\" \]

Reserved Keywords

The following is a list of reserved Democritus keywords:

```
if   else   for   return   int
bool  void  true   false  string
struct  *void  function   let
```

Literals

Literals are used to represent various values or constants within the language.

Integer Literals

Integer literals are simply a sequence of ASCII digits, represented in decimal.

\[ \text{INT} = \"[0-9]+\" \]

Boolean Literals

Boolean literals represent the two possible values that boolean variables can take, \texttt{true} or \texttt{false}. These literals are represented in lowercase.

\[ \text{BOOLEAN} = \"true|false\" \]

String Literals

String literals represent strings of characters, including escaped characters. String literals are automatically null-terminated. Strings are opened and closed with double quotations. A special OCaml \texttt{lexbuf} was used to parse string literals.


\[ \text{read_string buf = parse} \]
All Democritus Tokens

The list of tokens used in Democritus are as follows:

```
"//" { comment lexbuf }
"/*" { multicomment lexbuf }
'( ' { LPAREN }
') ' { RPAREN }
'{ ' { LBRACE }
'} ' { RBRACE }
'; ' { SEMI }
', ' { COMMA }
'+ ' { PLUS }
'- ' { MINUS }
'* ' { STAR }
'& ' { REF }
'. ' { DOT }
'/ ' { DIVIDE }
'= ' { ASSIGN }
"==" { EQ }
"!=" { NEQ }
"<" { LT }
"<=" { LEQ }
">" { GT }
">=" { GEQ }
"&&" { AND }
"||" { OR }
"!" { NOT }
"if" { IF }
"else" { ELSE }
"for" { FOR }
"return" { RETURN }
"int" { INT }
"bool" { BOOL }
"void" { VOID }
"true" { TRUE }
"string" { STRTYPE }
"struct" { STRUCT }
"*void" { VOIDSTAR }
```
Punctuation

Semicolon

The semicolon ‘;’ is required to terminate any statement in Democritus.

statement SEMI

Curly Brackets

In order to keep the language free-format, curly braces are used to delineate separate and nested blocks. These braces are required even for single-statement conditional and iteration loops.

LBRACE statements RBRACE

Parentheses

To assert precedence, expressions may be encapsulated within parentheses to guarantee order of operations.

LPAREN expression RPAREN

Comments

Comments may either be single-line, initialized with two backslashes, or multi-line, enclosed by \* and *\.

COMMENT = ("// \[^'\n\]* \n") | ("/* \[^*/\]* */")

3.5 Variable Declarations

In Democritus, local variables must be declared at the top of each function, before being later assigned.

Variable Declaration

Democritus requires all named variables to be declared with its type at the top of each function. Named variables are declared with the let [ID] type syntax. Assignment to these variables may then be done with =.

The grammar for variable declarations is as follows:

vdecl:
    LET ID typ SEMI

typ:
    INT
Struct Declaration

Structs are defined at the global scope, and can then be declared as variables. The global definitions are as follows:

```
sdecl:
   STRUCT ID LBRACE vdecl_list RBRACE

vdecl_list:
   | vdecl_list vdecl
```

3.6 Expressions and Operators

Expressions

Expressions may be any of the following:

Literal

A literal of any type, as detailed in the lexical conventions section.

Identifier

An identifier for a variable.

Binary Operation

A binary operation between an expression and another expression.

Unary Operation

A unary operation acting on the expression appearing on the immediate right of the operator.

Struct Access

An expression of a struct type accessing an identifier field with the dot (.) operator.

Struct Assignment

An expression of a struct type assigning a value to one of its fields (accessed with the dot (.) operator) using the = operator.

Function Call

A call to a function along with its formal arguments.
Variable Assignment

An identifier being assigned a value with the = operator.

Parenthisization

Another expression nested within parentheses.

The grammar for expressions is as follows:

```
expr:
    LITERAL
    | FLOATLITERAL
    | TRUE
    | FALSE
    | ID
    | STRING
    | expr PLUS expr  (* expr TERMINAL expr are binary operations *)
    | expr MINUS expr
    | expr STAR expr
    | expr DIVIDE expr
    | expr MOD expr
    | expr EQ expr
    | expr NEQ expr
    | expr LT expr
    | expr LEQ expr
    | expr GT expr
    | expr GEQ expr
    | expr AND expr
    | expr OR expr
    | expr DOT ID  (* struct access *)
    | expr DOT ID ASSIGN expr (* struct assign *)
    | MINUS expr %prec NEG (* unary arith negate *)
    | STAR expr %prec DEREF (* unary deref *)
    | REF expr (* unary ref *)
    | NOT expr (* unary log negate *)
    | ID ASSIGN expr
    | ID LPAREN actuals_opt RPAREN (* function call *)
    | LPAREN expr RPAREN (* paren’d expr *)
```

Binary and Unary Operations

A binary operation operates on the two expressions on the left and right side of the operator. Binary operations may be:

- an addition, subtraction, mult., division, or modulo on two arithmetic expressions (+, -, *, /, %).
  Modulo only works on integer types.
- equality or inequality expression between boolean expressions (==, !=, <, <=, >, >=, &, |, ||)

A unary operation operates on the expression on the operator’s right side:

- a negation of an arithmetic expression (-)
• a dereference of a pointer type (*)
• an address reference of a variable or field within a struct (&)
• a negation of a boolean expression (!)

**Arithmetic Operations**
Democritus supports all the arithmetic operations standard to most general-purpose languages, documented below. Automatic casting has not been included in the language, and the compiler will throw an error in the case that arithmetic operations are performed between the same types of expressions.

**Addition and Subtraction**
Addition works with the + character, behaving as expected. Subtraction is called with -.  

**Multiplication and Division**
Multiplication is called with *, and division with /. Division between integers discards the fractional part of the division.

**Modulo**
The remainder of an integer division operation can be computed via the modulo % operator.

**Boolean Expressions**
Democritus features all standard logical operators, utilizing ! for negation, and && and || for and and or, respectively. Each expression will return a boolean value of true or false.

**Equality**
Equality is tested with the == operator. Inequality is tested with !=. Equality may be tested on both boolean and arithmetic expressions.

**Negation**
Negation is done with !, a unary operation for boolean expressions.

**Comparison**
Democritus also features the <, <=, >, and >= operators. These represent less than, less than or equal to, greater than, and greater than or equal to, respectively. These operators are called on arithmetic expressions and return a boolean value.

**Chained Expressions**
Boolean expressions can be chained with && and ||, representing and and or. These operators have lower precedence than any of the other boolean operators described above. The and operator has a higher precedence than or.

**Parentheses**
Parentheses are used to group expressions together, since they have the highest order of precedence. Using parentheses will ensure that whatever is encapsulated within will be evaluated first.
Function Calls

Function calls are treated as expressions with a type equal to their return type. As an applicative-order language, Democritus evaluates function arguments first before passing them to the function. The grammar for function calls is as follows:

expr:
   
   | ID LPAREN actuals_opt RPAREN (* Function call *)

actuals_opt:
   /* nothing */?
   | actuals_list

actuals_list:
   expr
   | actuals_list COMMA expr

Pointers and References

Referencing and dereferencing operations are used to manage memory and addressing in Democritus. The unary operator & gives a variable or struct field’s address in memory, and the operator * dereferences a pointer type.

Operator Precedence and Associativity

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>()</td>
<td>Parenthesis</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>2</td>
<td>()</td>
<td>Function call</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>Dereference</td>
<td>Right-to-left</td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>Address-of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>!</td>
<td>Negation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~</td>
<td>Unary minus</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td>Multiplication</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>Division</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%</td>
<td>Modulo</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>+</td>
<td>Addition</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>Subtraction</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>&gt;&gt; =</td>
<td>For relational &gt; and ≥ respectively</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;=</td>
<td>For relational &lt; and ≤ respectively</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>== !=</td>
<td>For relational = and ≠ respectively</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>8</td>
<td>&amp;&amp;</td>
<td>Logical and</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>=</td>
<td>Assignment</td>
<td>Right-to-left</td>
</tr>
</tbody>
</table>

3.7 Statements

Statements written in a Democritus program are run from the top to bottom, sequentially. Statements can reduce to the following:
Expressions
An expression statement consists of an expression followed by a semicolon. Expressions in expression statements will be evaluated, with their values calculated.

Return Statements
A return statement is either a RETURN SEMI or RETURN expr SEMI. They are used as endpoints of a function, and control from a function returns to the original caller when a return statement is executed. Returns may be empty or return a type, though non-void functions must return an expression of their type.

Nested Blocks
A nested block is another statement list encapsulated within braces {}.

Conditional Statements
Conditional statements follow the IF (boolean expr) stmt1 ELSE stmt2 format. When the expr evaluates to true, stmt1 is run. Otherwise, if an ELSE and stmt2 have been specified, stmt2 is run.

Conditional Loops
A conditional loop is similar to a conditional statement, except in that it will loop or run repeatedly until its given boolean expression evaluates to false. In the case that the expression never evaluates to false, an infinite loop will occur.

Democritus eliminates the while keyword sometimes used in conditional iteration. Conditional loops follow the FOR (expr1;boolean expr2;expr3) stmt format where expr1 and expr3 are optional expressions to be evaluated prior to entering the for loop and upon each loop completion, respectively. Prior to entering or re-entering the loop, expr2 is evaluated; control only transfers to stmt if this evaluation returns true. If both expr1 and expr3 are omitted, a simpler for loop can be written of the form FOR (boolean expr) stmt.

The full grammar for statements is as follows:

stmt_list:
    /* nothing */
    | stmt_list stmt
stmt:
    expr SEMI
    | RETURN SEMI
    | RETURN expr SEMI
    | LBRACE stmt_list RBRACE
    | IF LPAREN expr RPAREN stmt %prec NOELSE
    | IF LPAREN expr RPAREN stmt ELSE stmt
    | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
    | FOR LPAREN expr RPAREN stmt
### 3.8 Functions

#### Overview and Grammar

Functions can be defined in Democritus to return one or no data type. Functions are evaluated via eager (applicative-order) evaluation and the function implementation must directly follow the function header. The grammar for function declarations is as follows:

```
fdecl:
    FUNCTION ID LPAREN formals_opt RPAREN typ LBRACE vdecl_list stmt_list RBRACE
formals_opt:
    /* nothing */
    | formal_list
formal_list:
    ID typ
    | formal_list COMMA ID typ

All functions require return statements at the end, and must return an expression of the same type as the function. **void** functions may simply terminate with an empty return statement.
```

```plaintext
} function function_name([formal_arg type, ... ]) type_r {
    [function implementation]
    return [variable of type type_r]
}
```

#### Calling and Recursion

Functions may be recursive and call themselves:

```plaintext
function recursive_func(i int) void {
    if (i < 0) {
        return;
    } else {
        print( hi );
        recursive_func(i-1); // Call ourselves again.
    }
}
```

Functions may be called within other functions:

```plaintext
function main() void{
    recursive_func(3);
    return; // Return nothing for void.
}
```
3.9 Concurrency

Overview
Democritus intends to cater to modern software engineering use cases. Developments in the field are steering us more and more towards highly concurrent programming as the scale at which software is used trends upward.

Spawning Threads
To spawn threads, Democritus uses a wrapper around the C-language pthread family of functions.

The thread_t data type wraps pthread_t.

To spawn a thread, the thread function takes a variable number of arguments where the first argument is a function and the remaining optional arguments are the arguments for that function. It returns an error code.

The detach boolean determines whether or not the parent thread will be able to join on the thread or not.

```c
{ 
    function thread(f function, arg *void, arg, nthreads int) *void;
}
```
4. Project Plan

4.1 Planning

Much of the planning was facilitated by our weekly meetings with David Watkins, our TA. He very clearly explained what the requirements of each milestone entailed, and helped keep expectations transparent. Since Professor Edwards had emphasized the need for vertical development of features instead of horizontal building of each compiler layer, we quickly identified the key features that would be required to enable the key functionality of our language. Two of the most important components were structs and threads.

Our initial plan, which we largely followed, was to complete the Language Reference Manual, experiment with the layers of the compiler and get Hello, World! working, and then use what we had learned to begin implementing the more crucial aspects of the language.

4.2 Workflow

Workflow was facilitated by Git and GitHub, which allowed for the team to easily work on multiple features simultaneously and (usually) merge together features without overlapping conflicts. The Git workflow reached an optimal point by the conclusion of the project; new features would be developed, tested, and finalized in separate branches, and the commits for that feature would be squashed down until a single commit representing the new feature would be merged into the master branch.

Features were developed individually or through paired programming, depending on the scope and complexity of the feature. GitHub and division of labor allowed for many team members to work independently, at different times of the day per their own schedule. Branching allowed for one person to quickly deploy buggy code to another in hopes of resolving the issue, without any modification or bad commits to the master branch. GroupMe was used extensively for inter-team communication.
### 4.3 Team Member Responsibilities

<table>
<thead>
<tr>
<th>Team Member</th>
<th>Responsibilities</th>
<th>GitHub Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy Xu</td>
<td>structs, nested structs, pointers and casting</td>
<td>axxu</td>
</tr>
<tr>
<td></td>
<td>linkedlist implementation</td>
<td></td>
</tr>
<tr>
<td>Emily Pakulski</td>
<td>C-bindings help, language direction, git workflow</td>
<td>ohEmily</td>
</tr>
<tr>
<td></td>
<td>shell-scripting</td>
<td></td>
</tr>
<tr>
<td>Amarto Rajaram</td>
<td>C-bindings, threads, sockets, file I/O, malloc</td>
<td>Amarto</td>
</tr>
<tr>
<td></td>
<td>sleep, string operations</td>
<td></td>
</tr>
<tr>
<td>Kyle Lee</td>
<td>structs, LRM, Final Report, debugging</td>
<td>kyle–lee</td>
</tr>
<tr>
<td></td>
<td>assisting Amy</td>
<td></td>
</tr>
</tbody>
</table>
4.4 Git Logs

Note that ‘PLT Student’ was team member Amy Xu.

master branch

commit 4ecbc629057cc19c839d5e0f5f9224b88710ac8e
Merge: d0a23f0 cb6b509
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Wed May 11 18:04:44 2016 -0400

Merge pull request #24 from DemocritusLang/linkedlist_and_stack

Final linked list demo

commit cb6b5091e6812c3f3a60869bdb39bdd6900c201e
Author: PLT Student <axxu3795@gmail.com>
Date: Wed May 11 17:35:32 2016 -0400

Final linked list demo

Fixed merge accident

Fixed malloc size

Added to demo folder

commit d0a23f079f4a9828c6d2724141c5b7de1a126958
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 11 17:52:08 2016 -0400

Added simple threads test.

commit e113aa58cc30af12cad615d07ed57b99905e0edf
Author: Emily <ohEmily@users.noreply.github.com>
Date: Wed May 11 16:26:44 2016 -0400

Multithreading and networking working together. Added concurrent comic download. (#23)

* Added multithreaded test—getting parse error. Added strcat and int to string wrappers

* fixed parse error.

* Added memset to fix test—multithreaded—sockets.

* Fix memset bug

* Updated test file so sockets test passes

* Fixed thread function signature

* Added failing test.
* Fixed bug in init_thread() and added test that passes string into thread().
* possibly fixed request bug.
* Fixed binary file reading bugs.
* Moved code into demo directory.
* Fixed merge conflicts after rebase.

commit d4972e0ff8b5ba850bf39528cc0b113bc1912ee5
Merge: dbb06cc 232be1f
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Wed May 11 16:10:39 2016 -0400

Merge pull request #22 from DemocritusLang/build_malloc_attempt

commit 232be1f27fb7ffafa7110e5184c6016b81e2da94ff
Author: PLT Student <axxu3795@gmail.com>
Date: Wed May 11 03:45:50 2016 -0400

Malloc and simple linked lists working

commit dbb06cc9f0f27fb7ffafa7110e5184c6016b81e2da94ff
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 11 13:30:05 2016 -0400

Added check in tester for whether code was already compiled.

commit 36aee48817bdeeec1553b1c16963e5a8cc7aaaa6b
Merge: 0371f67 2402f74
Author: Amarto <aar2160@columbia.edu>
Date: Wed May 11 06:10:49 2016 -0400

Merge pull request #21 from DemocritusLang/sockets_finished

Sockets finished
commit 2402f740bec59d6ed4912b2ee302d9b9b7bb5480
Author: Amarto <aar2160@columbia.edu>
Date: Wed May 11 05:45:05 2016 -0400

Added free(), execl wrapper, and corrected output reference for socket test.
Changed tests to use free() after malloc. Refactored the weblink downloading
method to only take one param so it matches the signature for a thread function

commit 170645297d4578f9551f690b08b82718beeef033
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 11 01:44:23 2016 -0400

Changed up get request impl a bit.

commit 5226ad21d850b6622e09388a4337d0b515123f76
Author: Amarto <aar2160@columbia.edu>
Date: Tue May 10 17:46:11 2016 -0400

Added basic socket impl and loading files. Need to handle tests

commit 0371f67bac394c08ae72cb372491fd3264839f65
Merge: a6ce096 a784099
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Wed May 11 02:57:57 2016 -0400

Merge pull request #20 from DemocritusLang/add_float_and_mod

commit a78409901632a6f70e889fee8f5ccff2bfe12989
Author: PLT Student <axxu3795@gmail.com>
Date: Tue May 10 23:26:55 2016 -0400

Added modulo and floats

commit a6ce096863beb92e5b5251b0802613edab31cc76
Author: Emily <ohEmily@users.noreply.github.com>
Date: Tue May 10 23:09:23 2016 -0400

Added sleep function and test. (#18)

commit ff330840be2c13aeed6f2a6d87a69ce153f29421
Merge: a63b40f fdca4df4
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue May 10 15:28:57 2016 -0400
Merge pull request #17 from DemocritusLang/add_pointers

Authors done

commit fdcdaf4fa032354c8a9ec96f41cecb76b94e66f0
Author: PLT Student <axxu3795@gmail.com>
Date: Tue May 10 02:02:43 2016 -0400

Pointers done

Dereference syntax there, need to clean warning

Added ref op and semantic checking

Working pointers for ints, need to test rest

Modified test-pointer-int.dem for clarity and wrote test-pointer-bool, passing

Added single and multilevel struct tests, passing

Linked list test not working

Added hypothetical linked list tests (not working

Linked list proof of concept

Changed type* to *type to reflect Go syntax

commit a63b40fb618149ec65e57ea3ac986bdccf9f4ac4
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue May 10 01:11:44 2016 -0400

Added single line comments

commit 901814668aa2d7513fe74b45fa4390d82635ac01
Author: Amarto <aar2160@columbia.edu>
Date: Tue May 10 00:46:58 2016 -0400

Fixed void pointer notation to match Go syntax, fixed test

commit 0ed94930f8362cb6eb322ec6a9570043660aabb5
Merge: b662f67 d9b467b
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue May 10 00:43:18 2016 -0400

Merge pull request #15 from DemocritusLang/clean_nested_structs

Working nested structs

commit d9b467b80ce152696875d6ec1d3d2f1ec6ea77e6
Author: PLT Student <axxu3795@gmail.com>
Date: Mon May 9 22:47:07 2016 -0400

Working nested structs

Added nested struct test
Fixed mistyped identifier

nested structs working

Fixed typo in test-structs-nested.out and added another test

Edited test to be more informative of functionality

test-struct-nested

Commit b662f676ae12fbb27eedaf5af6ae990d76f423bc
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Mon May 9 20:34:05 2016 -0400

Finished file I/O. lseek also implemented.

Commit 84c1fc11bc2a2e59b8fec9d68937db8205f1b5d9
Author: Amarto <aar2160@columbia.edu>
Date: Sun May 8 20:30:58 2016 -0400

Added malloc and started file I/O.

Commit 8b394d051cfde07be958214aae56bf47988fb803
Author: Emily <ohEmily@users.noreply.github.com>
Date: Mon May 9 11:15:23 2016 -0400

Updated all instances of MicroC to Democritus and added 'make all' target (#12)

* Changed MicroC -> Democritus and added make all target.

* Changed file extension for democrituslang files from .mc to .dem.

Commit ed27ce5f8a31a740f3eb0e5ad3ff3cfc7a838f9
Author: Amarto <aar2160@columbia.edu>
Date: Sun May 8 19:31:43 2016 -0400

Fixed warnings resulting from merge

Commit 6cbdf15f8854a02fb695fa9aa41b50966431a7
Author: Amarto <aar2160@columbia.edu>
Date: Sat Apr 30 15:16:57 2016 -0400

Added multithreading and void pointers, and added calling bound C functions
Added declaration of thread() function to codegen. Everything compiles

Added basic threads checking to semant.ml. Need to wait until arguments for
pthread are passed in

Working on codegen.ml, but getting compiler warning. Working on threading test,
but need NULL keyword?

Added tests for threading and modified codegen and semant

Baby steps. Still not working. (temp commit).
Oops. But still not working.

Fixed some things in test case. Pretty sure function name should be passed in as a string. (temp commit.)

Temp commit. More debug info. Maybe fixed some bugs but same error.

temp commit – fixed compiler warning but old tests are failing

Fixed old tests, fixed compiler warning

Added correct (?) invocation of args in thread_init. Still not_found exception

It was failing to match on [e]. Changed to e, and now it’s giving a broken module error: params don’t match

Still not working (broken module) but now using lookup_function and pattern matching to remove option

Added a void ptr type for thread (kinda hacky for now but it’s for testing threads ). Also it’s now finding the function from the string name

Added thread testing script

THREADS NOW WORK IN SCRIPT!!!

Passing threading test

Fixed compiler warnings from pattern matching in codegen

commit bca9388f1d5b7011fde7461b2f1055562f1c7561
Author: PLT Student <axxu3795@gmail.com>
Date: Thu May 5 00:59:11 2016 −0400

Clean compilation without warnings

commit ecf06799e7b2a68a08ef9603a4b9eacfdfc7b3ce
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 13:28:00 2016 −0400

Removed codegen warnings, and some semant warnings

commit 08a4e105a2267891a38e76b4f280a4631bbe3413
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 00:54:52 2016 −0400

fixed struct tests for let format

commit 152ab95f7e0c087cc914c0a5c2b176951b77a1d3
Merge: b4f812b 116094b
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 00:36:07 2016 −0400

Merge add_structs

commit 116094b8ee508fd191c6d793cb14b9b5d6955c2a
Semantic checking to disallow circularly dependent structs

commit 490aa96cacf6e93d9a5b981f37244cc5b0cb6c6
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 17:03:42 2016 -0400

Fixed the stack overflow problem and updated tests

commit 41cb475f79c1d6baf22baf68b02219be8a9a49b2
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:19:54 2016 -0400

struct access works (messy)

commit eef6eb9fc000a844957f73dde0a56980a3b44ee0
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:14:10 2016 -0400

Structs reach llvm failure point. need to clean up exception catching and matches.

commit b4f812b37b0788d6f4a6d09495f41bc1515488ec
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sat Apr 30 13:29:07 2016 -0400

Flattened built-in function declarations so we don’t need extra variables.

commit 2c0b9cb13e157863e8dcd7fb2f3346426c5b1
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 02:06:26 2016 -0400

changed to named structs

commit aa095775c0b41e87763e8dcd7fb2f3346426c5b1
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Fri Apr 29 22:21:28 2016 -0400

added semant for struct field assignment

commit c90388e0f0d00ed30323be990eb29ec089fefa474
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 21:19:46 2016 -0400

Created struct field index list

commit ef7a1054b5250ff47bd60f9fbd2a6e13396e1796
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 03:07:09 2016 -0400

1type_of_type now includes struct types so structs can be allocated

commit e1b6f98760055b9f39c4f0a3606f06d94c2fc8b
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 26 18:48:27 2016 -0400
Cleaned up some warnings, still not sure what 42 is

commit 24ec2afbb38e4bad5f7684ef663aa6d6993116dce
Author: PLT Student <axxu3795@gmail.com>
Date: Mon Apr 25 13:38:02 2016 -0400

Working error checking for struct

commit 95a5222e09300e7f5037e22c78bd4282cba9929f
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 25 13:01:23 2016 -0400

Added struct tests

commit 995258d61bd5f0db54369a7fc65dd6f188e6d415
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 25 12:57:47 2016 -0400

Working struct semant (throws not found exception)

commit 037886b737edcbe231b780897b7b31e67e36046
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 19:10:56 2016 -0400

match struct compiles

commit 6fa5581d225d28b23d83efadf00b0a868b96740
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 18:49:31 2016 -0400

added broken struct accessor method

commit e91042b38c3dea600325c9239ddd42b7cbfebf6a
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 23 17:47:47 2016 -0400

Adding check_access, still need to match inside

commit 3940c80078342f00879360619d0d5f5ad0ba1c57
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 17:09:42 2016 -0400

Prepared to start adding structs to semant.

commit 450a12b335d46566822e314cbe3030fde240a17c
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 23 16:26:47 2016 -0400

Gave struct types a string to hold for struct type name

commit e870131767f7edd529e0a3fbb2b1e9a3ff366bdc
Merge: a37ba16 38d78d3
Author: Amarto <aar2160@columbia.edu>
Date: Tue Apr 19 23:40:52 2016 -0400
Merge pull request #7 from DemocritusLang/change_syntax_order

Change syntax order with tests

commit ca8356e47677421467fad358a65b6bd16809b4b37
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 19 21:49:46 2016 -0400

Added dot operator syntax as a binop

commit 38d78d3708f6dd5058345b5de776ae035f123240
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Mon Apr 18 00:23:08 2016 -0400

Fixed bad string tests.

commit 9523521d6768e94f504ff983a1de4738870f897
Author: PLT Student <axxu3795@gmail.com>
Date: Mon Apr 18 00:01:30 2016 -0400

I forgot to make clean the last commit b/c i’m dumb

commit b699bb84863eb88e006f95ef82f687f3e367586de
Author: PLT Student <axxu3795@gmail.com>
Date: Sun Apr 17 23:58:44 2016 -0400

Compiles with the third struct list

commit 34076bdcc2f6a519691555482261913623bfd97d
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sun Apr 17 23:36:54 2016 -0400

Initial addition of struct to parsing

commit d221587f8c81155a1ca8f9e2ba50b0a83a89684
Author: Amarto <aar2160@columbia.edu>
Date: Sun Apr 17 23:36:06 2016 -0400

Fixed test-helloworld-assign declaration order

commit 12301820c5bd32b55b29dfbd5a99068e62ee6b5
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 17 23:14:02 2016 -0400

Changed tests to add let keyword.

commit 3a626ec31e042cfa3bcb8f5410dd666fae12bea
Author: Amarto <aar2160@columbia.edu>
Date: Wed Apr 13 01:45:56 2016 -0400

Changed parser and scanner with LET keyword. Still working on tests

commit c6ecb302808b192e6e6f51360537556119a867ec
Author: Amarto <aar2160@columbia.edu>
Date: Tue Mar 15 00:33:01 2016 -0400
Temp commit — tried to change variable order but got SR error.

commit a37ba16c593be6be0d9980aec71b1d2b93eaf69e
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 11 13:05:53 2016 -0400

Added tentative install instructions (needs testing on Ubuntu 14.x and before)

commit 605b8bd1f6b6a1612d588ce0c5a52f107292d609
Merge: caa0380 0f67850
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue Apr 5 18:36:38 2016 -0400

Merge pull request #6 from DemocritusLang/strings_2
HelloWorld checkpoint!
:pizza:

commit 0f678507385ebacba0c05a41eac72ada0d9df015
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:29:24 2016 -0400

fixed failing function call test (semant only checks for print())

commit 22a7da1415ce11b8dbb75ea0a8766e50a40bac38
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 5 18:26:25 2016 -0400

Fixed missing printb

commit cb55b59f5981f69e28a347525e35ef1071020447
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:12:31 2016 -0400

Fixed tarball makefile builder for helloworld

commit 67ddab48071f203f447caf80f6906870af14e510
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:02:03 2016 -0400

Added test case for string assignment and printing.

commit daceabeb68dbd482000334162a0f797788554616
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 13:49:14 2016 -0400

Print hello world working. Tests that use printb() are failing, because i had to remove it from semant.ml temporarily.

commit f121b87bdcb8a6afac89d6374a9eb2705529d123
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 02:22:46 2016 -0400

Compiling, but not passing tests.

commit 93833564906d28a356e6cd8303241e69a764ac2f0
Tried moving strings to types...

commit de2ba3fd50270ec7894abbb7f8c9a1bb0efd8ca3
Author: Emily Pakulski <enp2111@columbia.edu>
Date:  Sun Apr 3 17:56:06 2016 -0400

Added partial implementation of string literal.

commit 2fd8d9408fc9ceb78e03d3ebdc2195ae4ad7403
Author: Emily Pakulski <enp2111@columbia.edu>
Date:  Sun Apr 3 16:53:52 2016 -0400

Added test and function for helloworld. Need string literal implementation.

commit 0471926a19d88626bab3140b6a12abcff588288620
Author: Emily Pakulski <enp2111@columbia.edu>
Date:  Sun Apr 3 16:27:19 2016 -0400

Changed Edwards’ print to be print_int to avoid confusion with our print implementation.

commit caa0380101c0ac9f657eb626b1930c5ca72bfa5
Author: Emily Pakulski <enp2111@columbia.edu>
Date:  Mon Mar 14 22:47:26 2016 -0400

Added function keyword to function declarations.

commit de3f696465ef9a95a737282d1affa7d1d812ca0
Author: Amarto Rajaram <amarto.rajaram@gmail.com>
Date:  Mon Mar 14 21:58:20 2016 -0400

Removed while keyword; replaced functionality with for.

commit d0829835a72243f858b3e2426ab81b04792ed883
Author: Amarto Rajaram <amarto.rajaram@gmail.com>
Date:  Mon Mar 14 21:03:09 2016 -0400

Added Edwards’ tests back in.

commit 798f67d953e965daee1282e67bf6b0e5373982058
Author: Emily Pakulski <enp2111@columbia.edu>
Date:  Fri Feb 26 11:45:05 2016 -0500

Edwards’ MicroC code with our README.

threads-2 branch

commit c298c8ae9920595dd8f6cb25aa76b5b3a25275a
Author: Amarto <aar2160@columbia.edu>
Date:  Sun May 8 17:35:31 2016 -0400

THREADS NOW WORK IN SCRIPT!”git status!
commit f131bac05e4816370e8892d53bd5e57ea996fe3a
Author: Amarto <aar2160@columbia.edu>
Date: Sun May 8 14:41:25 2016 -0400
Added thread testing script

commit c6f48061c066b532f9cfcf0d87ed0218fd44d231a
Author: Amarto <aar2160@columbia.edu>
Date: Sun May 8 02:52:26 2016 -0400
Added a void ptr type for thread (kinda hacky for now but it’s for testing threads)
Also it’s now finding the function from the string name

commit 2a17bc340b7fc0a03844fe5db10c2f1b2f635ed41c
Author: Amarto <aar2160@columbia.edu>
Date: Sat May 7 12:54:29 2016 -0400
Still not working (broken module) but now using lookup_function and pattern
matching to remove option

commit 97ee5e9b35c053cd5bf306e57fc037a41b318f211d6
Author: Amarto <aar2160@columbia.edu>
Date: Sat May 7 03:20:03 2016 -0400
It was failing to match on [e]. Changed to e, and now it’s giving a broken module
error: params don’t match

commit 81792eee3a686912c0116005af65341b927e69db
Author: Amarto <aar2160@columbia.edu>
Date: Sat May 7 00:04:54 2016 -0400
Added correct(?) invocation of args in thread_init. Still not_found exception

commit 1faeb7d59482ecda134d22f208d88b914b8bd2e4
Author: Amarto <aar2160@columbia.edu>
Date: Thu May 5 02:23:59 2016 -0400
Fixed old tests, fixed compiler warning

commit 012414249fc2a0873d38f67cb2f2a68fb06793
Author: Amarto <aar2160@columbia.edu>
Date: Thu May 5 02:01:45 2016 -0400
temp commit – fixed compiler warning but old tests are failing

commit 4adb64624bacd11b5d94d9be9a0ac5a6eb5b8d9da
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 4 22:06:33 2016 -0400
Temp commit. More debug info. Maybe fixed some bugs but same error.

commit 7e18b4917c27092a34390e173a013fe4c87a06b0
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 4 21:32:25 2016 -0400
Fixed some things in test case. Pretty sure function name should be passed in as a
string. (temp commit.)

commit f8b057052c0cba41d1b22f2a4c26bf56a4b0de53
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 4 20:58:01 2016 -0400

Oops. But still not working.

commit ec286074732024096574ccbb3fb704e4569049325
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 4 19:36:05 2016 -0400

Baby steps. Still not working. (temp commit).

commit 583818a84f8f7097075417620fbebe1a31cdd911
Author: Amarto <aar2160@columbia.edu>
Date: Wed May 4 18:38:40 2016 -0400

Added tests for threading and modified codegen and semant

commit f5157bee06030e89dbaabdfc842678a67a90e683
Author: Amarto <aar2160@columbia.edu>
Date: Wed May 4 17:10:34 2016 -0400

Working on codegen.ml, but getting compiler warning. Working on threading test, but need NULL keyword?

commit 2e76483930fddba4b216c6d109da6e70bc44b7
Author: Amarto <aar2160@columbia.edu>
Date: Sat Apr 30 15:47:54 2016 -0400

Added basic threads checking to semant.ml. Need to wait until arguments for pthread are passed in

commit c8d806e00f51e5ca4a8ced271e70e2c46461e68a
Author: Amarto <aar2160@columbia.edu>
Date: Sat Apr 30 15:16:57 2016 -0400

Added declaration of thread() function to codegen. Everything compiles

commit b4f812b37b0788d6f4a6d09495f41bc1515488ec
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sat Apr 30 13:29:07 2016 -0400

Flattened built-in function declarations so we don’t need extra variables.

commit e870131767f7edd529e0a3fbb2b1e9a3ff366bdc
Merge: a37ba16 38d78d3
Author: Amarto <aar2160@columbia.edu>
Date: Tue Apr 19 23:40:52 2016 -0400

Merge pull request #7 from DemocritusLang/change_syntax_order

Change syntax order with tests

commit 38d78d3708f6dd5058345b5de776ae035f123240
Fixed bad string tests.

commit d221587f8c81155a1ca8f9e2ba50b0a83a89684
Author: Amarto <aar2160@columbia.edu>
Date: Sun Apr 17 23:36:06 2016 -0400
Fixed test-helloworld-assign declaration order

commit 12301820c5bbd32b55b29dfbd5a99068e62ee6b5
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 17 23:14:02 2016 -0400
Changed tests to add let keyword.

commit 3a626ec31e042cfa3bcb8fc5410dd66fdae12bea
Author: Amarto <aar2160@columbia.edu>
Date: Wed Apr 13 01:45:56 2016 -0400
 Changed parser and scanner with LET keyword. Still working on tests

commit c6ecb302808b192e6e6f51360537556119a867ec
Author: Amarto <aar2160@columbia.edu>
Date: Tue Mar 15 00:33:01 2016 -0400
Temp commit —— tried to change variable order but got SR error.

commit a37ba16c593be6be0d9980aec71b1d2b93eaf69e
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 11 13:05:53 2016 -0400
Added tentative install instructions (needs testing on Ubuntu 14.x and before)

commit 605b8bd1f6b6a1612d588ce0c5a52f107292d609
Merge: c6ecb302808b192e6e6f51360537556119a867ec
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue Apr 5 18:36:38 2016 -0400
Merge pull request #6 from DemocritusLang/strings

HelloWorld checkpoint!
:pizza:

commit 0f678507385ebacba0c05a41eac72ada0d9df015
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:29:24 2016 -0400
fixed failing function call test (semant only checks for print())

commit 22a7da1415c11b8dbb75ea0a8766e50a48bac38
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 5 18:26:25 2016 -0400
Fixed missing printb
commit cb55b59f5981f69e28a347525e35ef1071020447
Author: <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:12:31 2016 -0400

Fixed tarball makefile builder for helloworld

commit 67ddab48071f203f447caf80f6906870af14e510
Author: <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:02:03 2016 -0400

Added test case for string assignment and printing.

commit daceabeb68dbd482000334162a0f797788554616
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 13:49:14 2016 -0400

Print hello world working. Tests that use printb() are failing, because i had to remove it from semant.ml temporarily.

commit f121b87bdcb8a6afac89d6374a9eb2705529d123
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 02:22:46 2016 -0400

Compiling, but not passing tests.

commit 93833564906d28a36e6cd8303241e69a764ac2f0
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 18:11:16 2016 -0400

Tried moving strings to types...

commit de2ba3fd50270ec7894abbb7f8c9a1bb0efd8ca3
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 17:56:06 2016 -0400

Added partial implementation of string literal.

commit 2fd8d9408fc9ceb78e03d3ebdc2195aee4ad7403
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:53:52 2016 -0400

Added test and function for helloworld. Need string literal implementation.

commit 0471926a19d8626bab3140b6a12abcf588288620
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:27:19 2016 -0400

Changed Edwards’ print to be print_int to avoid confusion with our print implementation.

commit caa0380101c0ac9f657eb626b1930c5ca72b5d85e
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Mon Mar 14 22:47:26 2016 -0400

Added function keyword to function declarations.
commit de3f696465ef9a95a737282d1affa7d1d812cad0  
Author: Amarto Rajaram <aar2160@columbia.edu>  
Date:  Mon Mar 14 21:58:20 2016 -0400  
Removed while keyword; replaced functionality with for.

commit d0829835a72243f858b8e2426ab81b04792ed883  
Author: Amarto Rajaram <amarto.rajaram@gmail.com>  
Date:  Mon Mar 14 21:03:09 2016 -0400  
Added Edwards’ tests back in.

commit 798f67d953e965dae128e679f6b0e5373982058  
Author: Emily Pakulski <enp2111@columbia.edu>  
Date:  Fri Feb 26 11:45:05 2016 -0500  
Edwards’ MicroC code with our README.

nested-structs branch

commit 2545a29f6e4a106506f61cd21d00e68024c2d23e  
Author: PLT Student <axxu37958@gmail.com>  
Date:  Mon May 9 01:06:25 2016 -0400  
IT WORKS

commit b91958665b48ea59c0cd3b74fc531007c9a59e98  
Author: PLT Student <axxu37958@gmail.com>  
Date:  Mon May 9 00:21:41 2016 -0400  
IT WORKS... v messy

commit b39e230a6670c3e04f95b08d17fd7e8b054004f9  
Author: PLT Student <axxu37958@gmail.com>  
Date:  Sun May 8 23:50:39 2016 -0400  
Throwing error for comparison

commit a67b43313123361da7e0d7b533ad6a2f025dedf5  
Author: PLT Student <axxu37958@gmail.com>  
Date:  Thu May 5 17:28:12 2016 -0400  
Access working, not assigns

commit ba05136aa87a6c755ab85df7aaf95010aaccbf86b  
Author: PLT Student <axxu37958@gmail.com>  
Date:  Thu May 5 04:17:04 2016 -0400  
.lii finally stores

commit 394caf29ae741057ab618ea52c8a773c23e222c0  
Author: PLT Student <axxu37958@gmail.com>  
Date:  Thu May 5 00:48:16 2016 -0400  
fixed segfault but need to move to new branch
still stuck on structs

How to make pointer to value...

I forgot to clean again...

Working on struct type problem still

Why is assigning Color to Color failing semant

nested structs so far

fixed struct tests for let format

Semantic checking to disallow circularly dependent structs
Fixed the stack overflow problem and updated tests

Structs reach llvm failure point. Need to clean up exception catching and matches.

Flattened built-in function declarations so we don’t need extra variables.

changed to named structs

added semant for struct field assignment

Created struct field index list

ltype of_type now includes struct types so structs can be allocated

Cleaned up some warnings, still not sure what 42 is

Commit: 490aa96caf6e93d9a5b981f37244cc5b0cb6c6
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 17:03:42 2016 -0400

Commit: 41cb475f79c1d6baf22bab68b02219be8a9a49b2
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:19:54 2016 -0400

Commit: eef6eb9fc00a844957f73d60a56980a3b44ee0
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:14:10 2016 -0400

Commit: b4f812b37b0788d6f4a609495f41bc1515488ec
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sat Apr 30 13:29:07 2016 -0400

Commit: 2c0b9c6a3e157863e8dccc7f2c2346a6262c5b1
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 02:06:26 2016 -0400

Commit: aa095775c0b41e8776214758f2af8d31e142d1ea
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Fri Apr 29 22:21:28 2016 -0400

Commit: c90388e0f0d000e30323be990eb29ec089f4f74
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 21:19:46 2016 -0400

Commit: ef7a105b5250ff47bd60f9fbdb2a6e13396e1796
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 03:07:09 2016 -0400

Commit: 24ec2af384e4bad5f7684ef663aa6d6993116dce
Author: PLT Student <axxu3795@gmail.com>
Working error checking for struct

commit 95a5222e09300e7f5037e22c78bd4282c892f9
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 25 13:01:23 2016 -0400

Added struct tests

commit 995258d61bd5f0db54369a7fc65dd6f188e6d415
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 25 13:01:23 2016 -0400

Working struct semant (throws not found exception)

commit 03786b77edcbe231b780897b7b31e67e36046
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 19:10:56 2016 -0400

match struct compiles

commit 6fa5581d2255d28b23d83efaf0b0a868b96740
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 18:49:31 2016 -0400

added broken struct accessor method

commit e91042b38c3dea600325c9239dd42b7cb7f6a
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 23 17:47:47 2016 -0400

Adding check access, still need to match inside

commit 3940c8007342f00879360619d0d5fad0bal057
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 17:09:42 2016 -0400

Prepared to start adding structs to semant.

commit 450a12b335d46566822e31d43cbe3030f0dc240a17c
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 23 16:26:47 2016 -0400

Gave struct types a string to hold for struct type name

commit e870131767f7edd529e0a3fbb2b1e9a3ff366bd
Merge: a37ba16 38d78d3
Author: Amarto <aar2160@columbia.edu>
Date: Tue Apr 19 23:40:52 2016 -0400

Merge pull request #7 from DemocritusLang/change_syntax_order

Change syntax order with tests

commit ca8356e47677421467fad358a65bbd16809b4b37
Added dot operator syntax as a binop.

Fixed bad string tests.

I forgot to make clean the last commit b/c I’m dumb

Compiles with the third struct list

Initial addition of struct to parsing

Fixed test-helloworld-assign declaration order

Changed tests to add let keyword.

Changed parser and scanner with LET keyword. Still working on tests

Temp commit — tried to change variable order but got SR error.

Added tentative install instructions (needs testing on Ubuntu 14.x and before)

commit 605b8bd1f6b6a1612d588ce0c5a52f107292d609
Merge: caa0380 0f67850
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue Apr 5 18:36:38 2016 -0400

  Merge pull request #6 from DemocritusLang/strings_2

HelloWorld checkpoint!
:pizza:

commit 0f678507385ebacba0c05a41eac72ada0d9df015
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:29:24 2016 -0400

  fixed failing function call test (semant only checks for print())

commit 22a7da1415cellb8d2bb75ea0a8766e50a48bac38
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 5 18:26:25 2016 -0400

  Fixed missing printb

commit cb55b59f5981f69e28a347525e35ef1071020447
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:12:31 2016 -0400

  Fixed tarball makefile builder for helloworld

commit 67ddab48071f203f447caf80f6906870af14e510
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:02:03 2016 -0400

  Added test case for string assignment and printing.

commit daceabeb68bd482000334162a0f797788554616
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 13:49:14 2016 -0400

  Print hello world working. Tests that use printb() are failing, because i had to
  remove it from semant.ml temporarily.

commit f121b87bdcb8a6afac89d6374a9eb2705529d123
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 02:22:46 2016 -0400

  Compiling, but not passing tests.

commit 93833564906d28a36e6cd8303241e69a764ac2f0
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 18:11:16 2016 -0400

  Tried moving strings to types...
commit de2ba3fd50270ec7894abb7f8c9a1bb0efd8ca3
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 17:56:06 2016 -0400

  Added partial implementation of string literal.

commit 2fd8d9408fc9ceb78e03d3ebdc2195ae4ad7403
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:53:52 2016 -0400

  Added test and function for helloworld. Need string literal implementation.

commit 0471926a19d8626bab3140b6a12abcdf588288620
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:27:19 2016 -0400

  Changed Edwards’ print to be print_int to avoid confusion with our print implementation.

commit caa0380101c0ac9f657eb626b1930c5ca72b7d5e
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Mon Mar 14 22:47:26 2016 -0400

  Added function keyword to function declarations.

commit de3f696465ef9a95a737282d1affa7d1d812cad0
Author: Amarto Rajaram <aar2160@columbia.edu>
Date: Mon Mar 14 21:58:20 2016 -0400

  Removed while keyword; replaced functionality with for.

commit d0829835a72243f858bbe2426ab81b04792ed883
Author: Amarto Rajaram <amarto.rajaram@gmail.com>
Date: Mon Mar 14 21:03:09 2016 -0400

  Added Edwards’ tests back in.

commit 798f67d953e965dael287e679f6b0e5373982058
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Fri Feb 26 11:45:05 2016 -0500

  Edwards’ MicroC code with our README.

linkedlist-and-stack branch

commit cb6b5091e6812c3f3a60869b0db39b0d6900c201e
Author: PLT Student <axxu3795@gmail.com>
Date: Wed May 11 17:35:32 2016 -0400

  Final linked list demo
  Fixed merge accident
  Fixed malloc size
  Added to demo folder
commit d0a23f079f4a9828c6d2724141c5b7del1a26958
Author: Emily Pakulski <enp21118@columbia.edu>
Date: Wed May 11 17:52:08 2016 -0400

Added simple threads test.

commit e113aa58cc30af12cad615d07ed57b99905e0edf
Author: Emily <ohEmily@users.noreply.github.com>
Date: Wed May 11 16:26:44 2016 -0400

Multithreading and networking working together. Added concurrent comic download. (#23)
* Added multithreaded test—getting parse error. Added strcat and int to string wrappers
* fixed parse error.
* Added memset to fix test-mtithreaded-sockets.
* Fix memset bug
* Updated test file so sockets test passes
* Fixed thread function signature
* Added failing test.
* Fixed bug in init_thread() and added test that passes string into thread().
* possibly fixed request bug.
* Fixed binary file reading bugs.
* Moved code into demo directory.
* Fixed merge conflicts after rebase.

commit d4972e0ff8b5ba850bf39528cc0b113bc1912ee5
Merge: dbb06cc 232be1f
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Wed May 11 16:10:39 2016 -0400

Merge pull request #22 from DemocritusLang/build_malloc_attempt
Malloc and simple linked lists working

commit 232be1f27fb7ffafa7110e5184c6016b81e2da94ff
Author: PLT Student <axxu3795@gmail.com>
Date: Wed May 11 03:45:50 2016 -0400

Malloc and simple linked lists working
Fixed shift reduce errors
temp commit

CASTING AND MALLOC WORK

Cleaned up warnings

Cleaned up codegen warnings

Fixed (\ast a). dotops and halfassed addnode

Half way linked lists

Linked lists with add and print list functions

commit dbb06cc9f0f2f608583946158d48d8d841d8dc62
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 11 13:30:05 2016 −0400

Added check in tester for whether code was already compiled.

commit 36ae48817bdeec1553b1c16963e5abcc7aaaaa6b
Merge: 0371f67 2402f74
Author: Amarto <aar2160@columbia.edu>
Date: Wed May 11 06:10:49 2016 −0400

Merge pull request #21 from DemocritusLang/sockets finished

Sockets finished

commit 2402f740bec59d6ed4912b2ee302d9b9b7bb5480
Author: Amarto <aar2160@columbia.edu>
Date: Wed May 11 05:45:05 2016 −0400

Added free(), execl wrapper, and corrected output reference for socket test.

Changed tests to use free() after malloc. Refactored the weblink downloading
method to only take one param so it matches the signature for a thread function

commit 170645297d4578f9551f690b08b82718beebf033
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Wed May 11 01:44:23 2016 −0400

Changed up get request impl a bit.

commit 5226ad21d850b6622e09388a4337d0b515123f76
Author: Amarto <aar2160@columbia.edu>
Date: Tue May 10 17:46:11 2016 −0400

Added basic socket impl and loading files. Need to handle tests

commit 0371f67bac9f0f2f608583946158d48d8d841d8dc62
Merge: a6ce096 a784099
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Wed May 11 02:57:57 2016 −0400

Merge pull request #20 from DemocritusLang/add float and mod
Added modulo and floats

commit a78409901632a6f70e889fee8f5ccff2bfe12989
Author: PLT Student <axxu3795@gmail.com>
Date: Tue May 10 23:26:55 2016 -0400

Added modulo and floats

Mod done

Working on floats

fixed floating print issue

Working floats

Added floats in struct test

commit a6ce096863beb92e5b5251b0802613edab31cc76
Author: Emily <ohEmily@users.noreply.github.com>
Date: Tue May 10 23:09:23 2016 -0400

Added sleep function and test. (#18)

commit ff330840be2c13aeed6f2a6d87a69ce153f29421
Merge: a63b40f fdcdaf4
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue May 10 15:28:57 2016 -0400

Merge pull request #17 from DemocritusLang/add_pointers

Pointers done

commit fdcdaf4fa032354c8a9ec96f41cecb76b94e66f0
Author: PLT Student <axxu3795@gmail.com>
Date: Tue May 10 02:02:43 2016 -0400

Pointers done

Dereference syntax there, need to clean warning

Added ref op and semantic checking

Working pointers for ints, need to test rest

Modified test-pointer-int.dem for clarity and wrote test-pointer-bool, passing

Added single and multilevel struct tests, passing

Linkedlist test not working

Added hypothetical linkedlist tests (not working

Linked of list proof of concept

Changed type* to *type to reflect Go syntax
commit a63b40fb618149ec65e57ea3ac986bdccf9f4ac4
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue May 10 01:11:44 2016 -0400
Added singleline comments

commit 901814668aa2d7513fe74b45fa4390d82635ac01
Author: Amarto <aar2160@columbia.edu>
Date: Tue May 10 00:46:58 2016 -0400
Fixed void pointer notation to match Go syntax, fixed test

commit 0ed94930f8362cb6eb322ec6a9570043660aabb5
Merge: b662f67 d9b467b
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue May 10 00:43:18 2016 -0400
Merge pull request #15 from DemocritusLang/clean_nested_structs
Working nested structs

commit d9b467b80ce152696875d6ec1d3d2f1ec6ea77e6
Author: PLT Student <axxu3795@gmail.com>
Date: Mon May 9 22:47:07 2016 -0400
Working nested structs
Added nested struct test
Fixed mistyped identifier
nested structs working
Fixed typo in test-structs-nested.out and added another test
Edited test to be more informative of functionality
test-struct-nested1

commit b662f676ae12fbb27eedaf5af6e990d76f423bc
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Mon May 9 20:34:05 2016 -0400
Finished file I/O. lseek also implemented.

commit 84c1fc11bc2a2e59b8fec9d68937db8205f1b5d9
Author: Amarto <aar2160@columbia.edu>
Date: Sun May 8 20:30:58 2016 -0400
Added malloc and started file I/O.

commit 8b3944051cfde07be958214aae56bf47988fb803
Author: Emily <ohEmily@users.noreply.github.com>
Date: Mon May 9 11:15:23 2016 -0400
Updated all instances of MicroC to Democritus and added ‘make all’ target (#12)

* Changed MicroC -> Democritus and added make all target.

* Changed file extension for democrituslang files from .mc to .dem.

commit ed27ce5f8a31a740f3eb0e5ad3ff3cfcf7a838f9
Author: Amarto <aar2160@columbia.edu>
Date: Sun May 8 19:31:43 2016 -0400

Fixed warnings resulting from merge

commit c6cbdf15fd8854a02fb695fa9aa41b50966431a7
Author: Amarto <aar2160@columbia.edu>
Date: Sat Apr 30 15:16:57 2016 -0400

Added multithreading and void pointers, and added calling bound C functions
Added declaration of thread() function to codegen. Everything compiles

Added basic threads checking to semant.ml. Need to wait until arguments for
pthread are passed in

Working on codegen.ml, but getting compiler warning. Working on threading test,
but need NULL keyword?

Added tests for threading and modified codegen and semant

Baby steps. Still not working. (temp commit).

Oops. But still not working.

Fixed some things in test case. Pretty sure function name should be passed in as a
string. (temp commit.)

Temp commit. More debug info. Maybe fixed some bugs but same error.

temp commit – fixed compiler warning but old tests are failing

Fixed old tests, fixed compiler warning

Added correct(?) invocation of args in thread_init. Still not_found exception

It was failing to match on [e]. Changed to e, and now it’s giving a broken module
error: params don’t match

Still not working (broken module) but now using lookup_function and pattern
matching to remove option

Added a void ptr type for thread (kinda hacky for now but it’s for testing threads).
Also it’s now finding the function from the string name

Added thread testing script

THREADS NOW WORK IN SCRIPT!!!

Passing threading test
Fixed compiler warnings from pattern matching in codegen

commit bca9388f1d5b7011fde7461b2f1055562f1c7561
Author: PLT Student <axxu3795@gmail.com>
Date: Thu May 5 00:59:11 2016 -0400

Clean compilation without warnings

commit ecf06799e7b2a68a08ef9603a4b9eacfdfe7b3ce
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 13:28:00 2016 -0400

Removed codegen warnings, and some semant warnings

commit 08a4e105a2267891a38e76b4f280a4631bbee3413
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 00:54:52 2016 -0400

fixed struct tests for let format

commit 152ab95f7e0c087cc914c0a5c2b176951b77a1d3
Merge: b4f812b 116094b
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 00:36:07 2016 -0400

Merge add_structs

commit 116094b8ee508fd191c6d793cb14b9b5d6955c2a
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 20:57:04 2016 -0400

Semantic checking to disallow circularly dependent structs

commit 490aa96cafc6e93d9a5b981f37244cc5b0cb6c6
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 17:03:42 2016 -0400

Fixed the stack overflow problem and updated tests

commit 41cb475f79c1d6baf22baf68b02219be8a9a49b2
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:19:54 2016 -0400

struct access works (messy)

commit eef6eb9fc000a844957f73ddee0a56980a3b44ee0
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:14:10 2016 -0400

Structs reach llvm failure point. need to clean up exception catching and matches.

commit b4f812b37b0788d6f4a6d09495f41bc1515488ec
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sat Apr 30 13:29:07 2016 -0400
Flattened built-in function declarations so we don’t need extra variables.

commit 2c0b9cba13e157863e8dcc7fb2f3346a6262c5b1
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 02:06:26 2016 -0400

changed to named structs

commit aa095775c0b41e8f8d14758f2af8d31e42d1ea
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Fri Apr 29 22:21:28 2016 -0400

added semant for struct field assignment

commit c90388e0f0d0eed30323be990eb29ee89feef474
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 21:19:46 2016 -0400

Created struct field index list

commit ef7a1054b5250ff47bd60f9fbd2a6e13396e1796
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 03:07:09 2016 -0400

ltype_of_type now includes struct types so structs can be allocated

commit e1b6f98760055b9f39c4f0a03606f06d94c2fc8b
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 26 18:48:27 2016 -0400

Cleaned up some warnings, still not sure what 42 is

commit 24ec2af838e4bad5f7684ef663aa6d6993116dce
Author: PLT Student <axxu3795@gmail.com>
Date: Mon Apr 25 13:38:02 2016 -0400

Working error checking for struct

commit 037886b737edccbe231b780897b7b31e67e36046
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 19:10:56 2016 -0400

match struct compiles
394 commit 6fa5581d2255d28b23d83efaf0d00b0a868b96740
395 Author: Kyle Lee <kylelee.contact@gmail.com>
396 Date: Sat Apr 23 18:49:31 2016 -0400
397
398 added broken struct accessor method
399
400 commit e91042b38c3dea600325c9239dd42b7cbfebf6a
401 Author: PLT Student <axxu3795@gmail.com>
402 Date: Sat Apr 23 17:47:47 2016 -0400
403
404 Adding check_access, still need to match inside
405
406 commit 3940c80078342f00879360619d0d5f5ad0b1c57
407 Author: Kyle Lee <kylelee.contact@gmail.com>
408 Date: Sat Apr 23 17:09:42 2016 -0400
409
410 Prepared to start adding structs to semant.
411
412 commit 450a12b335d46566822e31d4b30f2c240a17c
413 Author: PLT Student <axxu3795@gmail.com>
414 Date: Sat Apr 23 16:26:47 2016 -0400
415
416 Gave struct types a string to hold for struct type name
417
418 commit e8701317767f7edd529e0a3fbb2b0e9a3ff36bdc
419 Merge: a37ba16 38d78d3
420 Author: Amarto <aar2160@columbia.edu>
421 Date: Tue Apr 19 23:40:52 2016 -0400
422
423 Merge pull request #7 from DemocritusLang/change_syntax_order
424
425 Change syntax order with tests
426
427 commit ca8356e47677421467fad358a65bbd16809b37
428 Author: PLT Student <axxu3795@gmail.com>
429 Date: Tue Apr 19 21:49:46 2016 -0400
430
431 Added dot operator syntax as a binop
432
433 commit 38d78d3708f6dd505345b5c776ae035f123240
434 Author: Emily Pakulski <enp2111@columbia.edu>
435 Date: Mon Apr 18 00:23:08 2016 -0400
436
437 Fixed bad string tests.
438
439 commit 9523521d6768e94f504ff983aldeb4738870f897
440 Author: PLT Student <axxu3795@gmail.com>
441 Date: Mon Apr 18 00:01:30 2016 -0400
442
443 I forgot to make clean the last commit b/c i'm dumb
444
445 commit b699bb8486e3eb86e0066f95e82f687f3e367586de
446 Author: PLT Student <axxu3795@gmail.com>
447 Date: Sun Apr 17 23:58:44 2016 -0400
448
449 Compiles with the third struct list
commit 34076bdcc2f6a519691555482261913623bffd97d
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sun Apr 17 23:36:54 2016 -0400

Initial addition of struct to parsing

commit d2221587f8c81155a1ca8f9e2ba50b0a83a89684
Author: Amarto <aar2160@columbia.edu>
Date: Sun Apr 17 23:36:06 2016 -0400

Fixed test—helloworld—assign declaration order

commit 12301820c5bbd32b55b29dfbd5a99068e62ee6b5
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 17 23:14:02 2016 -0400

Changed tests to add let keyword.

commit c6ecb302808b192e6e6f51360537556119a867ec
Author: Amarto <aar2160@columbia.edu>
Date: Wed Apr 13 01:45:56 2016 -0400

Changed parser and scanner with LET keyword. Still working on tests

commit a37ba16c593be6be0d9980aec71b1d2b93eaf69e
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 11 13:05:53 2016 -0400

Added tentative install instructions (needs testing on Ubuntu 14.x and before)

commit 605b8bd1f6b6a1612d588ce0c5a52f107292d609
Merge: caa0380 0f67850
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue Apr 5 18:36:38 2016 -0400

Merge pull request #6 from DemocritusLang/strings_2

HelloWorld checkpoint!

:pizza:

commit 0f678507385ebacba0c05a41eac72ada0d9df015
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:29:24 2016 -0400

fixed failing function call test (semant only checks for print())

commit 22a7da1415ce11b8dbb75ea0a8766e50a48bac38
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 5 18:26:25 2016 -0400
Fixed missing printb

commit cb55b59f5981f69e28a347525e35ef1071020447
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:12:31 2016 -0400

Fixed tarball makefile builder for helloworld

commit 67ddab48071f203f447caf80f6906870af14e510
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:02:03 2016 -0400

   Added test case for string assignment and printing.

commit daceabeb68dbd482000334162a0f797788554616
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 13:49:14 2016 -0400

   Print hello world working. Tests that use printb() are failing, because i had to
   remove it from semant.ml temporarily.

commit f121b87bdc8b8a6afac89d6374a9eb2705529d123
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 02:22:46 2016 -0400

   Compiling, but not passing tests.

commit 93833564906d28a36e6cd830324162a0f797788554616
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 18:11:16 2016 -0400

   Tried moving strings to types...

commit de2ba3fd50270ec7894abbb7f8c9a1bb0efd8ca3
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 17:56:06 2016 -0400

   Added partial implementation of string literal.

commit 2fd8d9408fc9ceb78e03d3ebdc2195aee4ad7403
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:53:52 2016 -0400

   Added test and function for helloworld. Need string literal implementation.

commit 0471926a19d8626bab3140b6a2abc588288620
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:27:19 2016 -0400

   Changed Edwards’ print to be print_int to avoid confusion with our print
   implementation.
Added function keyword to function declarations.

commit de3f696465ef9a95a737282d1affa7d1d812cad0
Author: Amarto Rajaram <aar2160@columbia.edu>
Date: Mon Mar 14 21:58:20 2016 −0400

Removed while keyword; replaced functionality with for.

commit d0829835a72243f858bbe2426ab81b04792ed883
Author: Amarto Rajaram <amarto.rajaram@gmail.com>
Date: Mon Mar 14 21:03:09 2016 −0400

Added Edwards’ tests back in.

commit 798f67d953e956dae1e1282e679f6b8e537982058
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Fri Feb 26 11:45:05 2016 −0500

Edwards’ MicroC code with our README.

**fix-malloc branch**

commit c7f317b4dd9d4de04a2152b83e4a65feadd128
Author: PLT Student <axxu3795@gmail.com>
Date: Wed May 11 01:28:24 2016 −0400

commented out check_assign, test-pointer-malloc replicating malloc problem in codegen

commit a6ce096863beb92e5b5251b0802613edab31cc76
Author: Emily <ohEmily@users.noreply.github.com>
Date: Tue May 10 23:09:23 2016 −0400

Added sleep function and test. (#18)

commit ff330840be2c13aead6eaf2a6d87a69ce153f29421
Merge: a63b40f fdcadf4
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue May 10 15:28:57 2016 −0400

Merge pull request #17 from DemocritusLang/add_pointers

Pointers done

commit fdcadf4fa032354ca8a9ec96f41cecb76b94e66f0
Author: PLT Student <axxu3795@gmail.com>
Date: Tue May 10 02:02:43 2016 −0400

Pointers done

Dereference syntax there, need to clean warning

Added ref op and semantic checking

Working pointers for ints, need to test rest
Modified test-pointer-int_dem for clarity and wrote test-pointer-bool, passing

Added single and multilevel struct tests, passing

Linkedlist test not working

Added hypothetical linkedlist tests (not working

Linked of list proof of concept

Changed type* to *type to reflect Go syntax

commit a63b40f618149ec65e57ea3ac986bdccf9f4ac4
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue May 10 01:11:44 2016 -0400

Added singleline comments

commit 901814668aa2d7513fe74b45fa4390d82635ac01
Author: Amarto <aar2160@columbia.edu>
Date: Tue May 10 00:46:58 2016 -0400

Fixed void pointer notation to match Go syntax, fixed test

commit 0ed94930f8362cb6eb322ec6a9570043660aabb5
Merge: b662f67 d9b467b
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue May 10 00:43:18 2016 -0400

Working nested structs

commit d9b467b80ce152696875d6ec1d3d2f1ec6ea77e6
Author: PLT Student <axxu3795@gmail.com>
Date: Mon May 9 22:47:07 2016 -0400

Working nested structs

commit b662f676ae12fbb27eedaf5af6ae990d76f423bc
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Mon May 9 20:34:05 2016 -0400
Finished file I/O. lseek also implemented.

commit 84c1fc11bc2a2e59b8fec9d68937db8205f1b5d9
Author: Amarto <aar2160@columbia.edu>
Date: Sun May 8 20:30:58 2016 -0400
Added malloc and started file I/O.

commit 8b3944051cfde07be958214aae56bf47988fb803
Author: Emily <ohEmily@users.noreply.github.com>
Date: Mon May 9 11:15:23 2016 -0400
Updated all instances of MicroC to Democritus and added 'make all' target (#12)
  * Changed MicroC -> Democritus and added make all target.
  * Changed file extension for democrituslang files from .mc to .dem.

commit ed27ce5f8a31a740f3eb0e5ad3ff3cfcf7a838f9
Author: Amarto <aar2160@columbia.edu>
Date: Sun May 8 19:31:43 2016 -0400
Fixed warnings resulting from merge

commit c6cbdf15fd8854a02fb695fa9aa41b50966431a7
Author: Amarto <aar2160@columbia.edu>
Date: Sat Apr 30 15:16:57 2016 -0400
Added multithreading and void pointers, and added calling bound C functions
Added declaration of thread() function to codegen. Everything compiles
Added basic threads checking to semant.ml. Need to wait until arguments for
pthread are passed in

Working on codegen.ml, but getting compiler warning. Working on threading test,
but need NULL keyword?

Added tests for threading and modified codegen and semant

Baby steps. Still not working. (temp commit).

Oops. But still not working.

Fixed some things in test case. Pretty sure function name should be passed in as a
string. (temp commit.)

Temp commit. More debug info. Maybe fixed some bugs but same error.

temp commit – fixed compiler warning but old tests are failing

Fixed old tests, fixed compiler warning

Added correct(?) invocation of args in thread_init. Still not_found exception

It was failing to match on [e]. Changed to e, and now it’s giving a broken module
error: params don’t match
Still not working (broken module) but now using lookup_function and pattern matching to remove option

Added a void ptr type for thread (kinda hacky for now but it’s for testing threads). Also it’s now finding the function from the string name

Added thread testing script

THREADS NOW WORK IN SCRIPT!!!

Passing threading test

Fixed compiler warnings from pattern matching in codegen

commit bca9388f1d5b7011fde7461b2f1055562f1c7561
Author: PLT Student <axxu3795@gmail.com>
Date: Thu May 5 00:59:11 2016 -0400

Clean compilation without warnings

commit ecf06799e7b2a68a08ef9603a4b9eacfd6c7b3ce
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 13:28:00 2016 -0400

Removed codegen warnings, and some semant warnings

commit 08a4e105a2267891a38e76b4f280a4631bbe3413
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 00:54:52 2016 -0400

fixed struct tests for let format

commit 152ab95f7e0c087cc914c0a5c2b176951b77a1d3
Merge: b4f812b 116094b
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 00:36:07 2016 -0400

Merge add structs

commit 116094b8ee508fd191c6d793cb14b9b5d6955c2a
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 20:57:04 2016 -0400

Semantic checking to disallow circularly dependent structs

commit 490aa96cacf6e93d9a5b981f37244cc5b0cb6c6
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 17:03:42 2016 -0400

Fixed the stack overflow problem and updated tests

commit 41cb475f79c1d6baf22baf68b02219be8a9a49b2
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:19:54 2016 -0400
struct access works (messy)

commit eef6eb9fc000a844957f73dde0a56980a3b44ee0
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:14:10 2016 -0400

Structs reach llvm failure point. need to clean up exception catching and matches.

commit b4f812b37b0788d6f4a6d09495f41bc1515488ec
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sat Apr 30 13:29:07 2016 -0400

Flattened built-in function declarations so we don’t need extra variables.

commit 2c0b9cba13e157863e8dcc7fb2f3346a6262c5b1
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 02:06:26 2016 -0400

changed to named structs

commit aa095775c0b41e8776214758f2af8d31e142d1ea
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Fri Apr 29 22:21:28 2016 -0400

added semant for struct field assignment

commit c90388e0f0d0eed30323be990eb29ec089efef474
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 21:19:46 2016 -0400

Created struct field index list

commit ef7a1054b5250ff47bd60f9bf2f63e246a6262c5b1
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 03:07:09 2016 -0400

ltype_of_type now includes struct types so structs can be allocated

commit e1b6f98760055b9f39c4f0a03606f06d94c2fc8b
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 26 18:48:27 2016 -0400

Cleaned up some warnings, still not sure what 42 is

commit 24ec2af838e4bad5f7684ef663aa6d6993116dce
Author: PLT Student <axxu3795@gmail.com>
Date: Mon Apr 25 13:38:02 2016 -0400

Working error checking for struct

commit 95a5222e09300e7f5037e22c78bd4282cba9929f
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 25 13:01:23 2016 -0400

Added struct tests
commit 995258d61bd5f0db54369a7fc65dd6f188e6d415
Author: Kyle Lee <kylelee.contact@gmail.com>
Date:   Mon Apr 25 12:57:47 2016 −0400

   Working struct semant (throws not found exception)

commit 037886b737edccbe231b780897b7b31e67e36046
Author: Kyle Lee <kylelee.contact@gmail.com>
Date:   Sat Apr 23 19:10:56 2016 −0400

   match struct compiles

commit 6fa5581d2255d28b23d83efaf00b0a868b96740
Author: Kyle Lee <kylelee.contact@gmail.com>
Date:   Sat Apr 23 18:49:31 2016 −0400

   added broken struct accessor method

commit e91042b38c3dea600325c9239dd42b7cebfeb6a
Author: PLT Student <axxu3795@gmail.com>
Date:   Sat Apr 23 17:47:47 2016 −0400

   Adding check_access, still need to match inside

commit 3940c80078342f00879360619d0d5f5ad0bac57
Author: Kyle Lee <kylelee.contact@gmail.com>
Date:   Sat Apr 23 17:09:42 2016 −0400

   Prepared to start adding structs to semant.

commit 450a12b335d46566822e314cbe303fde240a17c
Author: PLT Student <axxu3795@gmail.com>
Date:   Sat Apr 23 16:26:47 2016 −0400

   Gave struct types a string to hold for struct type name

commit e870131767f7edd529e0a3fbb2b1e9a3ff366bdc
Merge: a37ba16 38d78d3
Author: Amarto <aar2160@columbia.edu>
Date:   Tue Apr 19 23:40:52 2016 −0400

   Merge pull request #7 from DemocritusLang/change_syntax_order

commit 38d78d3708f6dd5058345b5de776ae035f123240
Author: Emily Pakulski <enp2111@columbia.edu>
Date:   Mon Apr 18 00:23:08 2016 −0400

   Added dot operator syntax as a binop

commit ca8356e47677421467fad358a65b0d16809b4b37
Author: PLT Student <axxu3795@gmail.com>
Date:   Tue Apr 19 21:49:46 2016 −0400

   Change syntax order with tests

commit 38d78d3708f6dd5058345b5de776ae035f123240
Author: Emily Pakulski <enp2111@columbia.edu>
Date:   Mon Apr 18 00:23:08 2016 −0400

   Fixed bad string tests.
commit 9523521d6768e94f504ff983aldeb4738870f897
Author: PLT Student <axxu3795@gmail.com>
Date: Mon Apr 18 00:01:30 2016 -0400
I forgot to make clean the last commit b/c i’m dumb

commit b69bb84863eb86e006f95e82f687f3e367586de
Author: PLT Student <axxu3795@gmail.com>
Date: Sun Apr 17 23:58:44 2016 -0400
Compiles with the third struct list

commit 34076bdcc2f6a51969155482261913623bf97d
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sun Apr 17 23:36:54 2016 -0400
Initial addition of struct to parsing

commit d221587f8c811555a1ca8f9e2ba50b0a3a89684
Author: Amarto <aar2160@columbia.edu>
Date: Sun Apr 17 23:36:06 2016 -0400
Fixed test-helloworld-assign declaration order

commit 12301820c5bbd32b55b29dfbd5a99068e62ee6b5
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 17 23:14:02 2016 -0400
Changed tests to add let keyword.

commit 3a626ec31e042cfa3bcb8fc5410dd666fae12bea
Author: Amarto <aar2160@columbia.edu>
Date: Wed Apr 13 01:45:56 2016 -0400
Changed parser and scanner with LET keyword. Still working on tests

commit c6ecb302808b192e6e6f5136053756119a867ec
Author: Amarto <aar2160@columbia.edu>
Date: Tue Mar 15 00:33:01 2016 -0400
Temp commit — tried to change variable order but got SR error.

commit a37ba16c593be6be0d9980aec71b1d2b93eaf69e
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 11 13:05:53 2016 -0400
Added tentative install instructions (needs testing on Ubuntu 14.x and before)

commit 605b8bd1f66b6a1612d588ce0c5a52f107292d609
Merge: caa0380 f0f7850
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue Apr 5 18:36:38 2016 -0400
Merge pull request #6 from DemocritusLang/strings_2
HelloWorld checkpoint!
:pizza:

commit 0f678507385ebacba0c05a41eac72ada0d9df015
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:29:24 2016 -0400

fixed failing function call test (semant only checks for print())

commit 22a7da1415ce11b8dbb75ea0a8766e50a48bac38
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 5 18:26:25 2016 -0400

Fixed missing printb

commit cb55b59f5981f69e28a347525e35ef1071020447
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:12:31 2016 -0400

Fixed tarball makefile builder for helloworld

commit 67ddab48071f203f447caf80f6906870af14e510
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:02:03 2016 -0400

Added test case for string assignment and printing.

commit daceabeb68db48200334162a0f797788554616
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 13:49:14 2016 -0400

Print hello world working. Tests that use printb() are failing, because i had to remove it from semant.ml temporarily.

commit f121b87bdcb8a6afac89d6374a9eb2705529d123
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 02:22:46 2016 -0400

Compiling, but not passing tests.

commit 93833564906d28a36e6cd8303241e69a764ac2f0
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 18:11:16 2016 -0400

Tried moving strings to types...

commit de2ba3fd502700ec7894abb7f8c9albb0efd8ca3
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 17:56:06 2016 -0400

Added partial implementation of string literal.

commit 2fd8d9408fc9ceb78e03d3ebdc2195ae4ad7403
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:53:52 2016 -0400
Added test and function for helloworld. Need string literal implementation.

commit 0471926a19d8626bab3140b6a12abcf588288620
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:27:19 2016 -0400

Changed Edwards’ print to be print_int to avoid confusion with our print implementation.

commit caa0380101c0380101c00a95a737282d1a7d1d812cad0
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Mon Mar 14 22:47:26 2016 -0400

Added function keyword to function declarations.

commit de3f694b65ef9a95a737282d1a7d1d812cad0
Author: Amarto Rajaram <aar2160@columbia.edu>
Date: Mon Mar 14 21:58:20 2016 -0400

Removed while keyword; replaced functionality with for.

commit d0829835a72243f8588b8e2426ab81b04792ed883
Author: Amarto Rajaram <amarto.rajaram@gmail.com>
Date: Mon Mar 14 21:03:09 2016 -0400

Added Edwards’ tests back in.

commit 798f67d953e965d1e1282e679f6b8e5373982058
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Fri Feb 26 11:45:05 2016 -0500

Edwards’ MicroC code with our README.

build-malloc-attempt branch

commit 232be1f27fb7ffa7110e5184c6016b81e2da94ff
Author: PLT Student <axxu3795@gmail.com>
Date: Wed May 11 03:45:50 2016 -0400

Malloc and simple linked lists working

Fixed shift reduce errors

temp commit

CASTING AND MALLOC WORK

Cleaned up warnings

Cleaned up codegen warnings

Fixed (*a). dotops and halfassed addnode

Half way linked lists

Linked lists with add and print_list functions

70
Added check in tester for whether code was already compiled.

Merge pull request #21 from DemocritusLang/sockets.finished

Sockets finished

Added free(), execl wrapper, and corrected output reference for socket test. Changed tests to use free() after malloc. Refactored the weblink downloading method to only take one param so it matches the signature for a thread function

Changed up get request impl a bit.

Added basic socket impl and loading files. Need to handle tests

Fixed floating print issue
Working floats
Added floats in struct test

commit a6ce096863beb92e5b5251b0802613edab31cc76
Author: Emily <ohEmily@users.noreply.github.com>
Date: Tue May 10 23:09:23 2016 -0400

Added sleep function and test. (#18)

commit ff330840be2c13aeed6f2a6d87a69ce153f29421
Merge: a63b40f fdcafd4
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue May 10 15:28:57 2016 -0400

Merge pull request #17 from DemocritusLang/add_pointers

Pointers done

commit fdcafd4fa032354c8a9ec96f41cecb76b94e66f0
Author: PLT Student <axxu3795@gmail.com>
Date: Tue May 10 02:02:43 2016 -0400

Pointers done

Dereference syntax there, need to clean warning
Added ref op and semantic checking
Working pointers for ints, need to test rest
Modified test-pointer-int.dem for clarity and wrote test-pointer-bool, passing
Added single and multilevel struct tests, passing
Linkedlist test not working
Added hypothetical linkedlist tests (not working
Linked of list proof of concept
Changed type∗ to ∗type to reflect Go syntax

commit a63b40fb618149ec65e57ea3ac986bdccf9f4ac4
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue May 10 01:11:44 2016 -0400

Added singleline comments

commit 901814668aa2d7513fe74b45fa4390d82635ac01
Author: Amarto <aar2160@columbia.edu>
Date: Tue May 10 00:46:58 2016 -0400

Fixed void pointer notation to match Go syntax, fixed test
132  commit 0ed94930f8362cb6eb322ec6a9570043660aabb5
133  Merge: b662f67 d9b467b
134  Author: Amy Xin Xu <axxu3795@gmail.com>
135  Date: Tue May 10 00:43:18 2016 −0400
136
137  Merge pull request #15 from DemocritusLang/clean_nested_structs
138
139  Working nested structs
140
141  commit d9b467b80ce152696875d6ec1d3d2flec6ea77e6
142  Author: PLT Student <axxu3795@gmail.com>
143  Date: Mon May 9 22:47:07 2016 −0400
144
145  Working nested structs
146
147  Added nested struct test
148
149  Fixed mistyped identifier
150
151  nested structs working
152
153  Fixed typo in test-structs-nested.out and added another test
154
155  Edited test to be more informative of functionality
156
157  test-struct-nested1
158
159  commit b662f676ae12fbb27eedaf5af6ae990d76f423bc
160  Author: Emily Pakulski <enp2111@columbia.edu>
161  Date: Mon May 9 20:34:05 2016 −0400
162
163  Finished file I/O. lseek also implemented.
164
165  commit 84c1fc11bc2a2e59b8f8ec9d68937db8205f1b5d9
166  Author: Amarto <aar2160@columbia.edu>
167  Date: Sun May 8 20:30:58 2016 −0400
168
169  Added malloc and started file I/O.
170
171  commit 8b3944051cfde07be958214aae56bf47988fb803
172  Author: Emily <ohEmily@users.noreply.github.com>
173  Date: Mon May 9 11:15:23 2016 −0400
174
175  Updated all instances of MicroC to Democritus and added 'make all' target (#12)
176
177  Changed MicroC -> Democritus and added make all target.
178
179  Changed file extension for democrituslang files from .mc to .dem.
180
181  commit ed27ce5f8a31a740f3eb0e5ad3ff3cfc7a838f9
182  Author: Amarto <aar2160@columbia.edu>
183  Date: Sun May 8 19:31:43 2016 −0400
184
185  Fixed warnings resulting from merge
186
187  commit c6c5bf15fd8854a02fb695fa9a41b50966431a7
Added multithreading and void pointers, and added calling bound C functions
Added declaration of thread() function to codegen. Everything compiles
Added basic threads checking to semant.ml. Need to wait until arguments for
pthread are passed in
Working on codegen.ml, but getting compiler warning. Working on threading test,
but need NULL keyword?
Added tests for threading and modified codegen and semant
Baby steps. Still not working. (temp commit).
Oops. But still not working.
Fixed some things in test case. Pretty sure function name should be passed in as a
string. (temp commit.)
Temp commit. More debug info. Maybe fixed some bugs but same error.
temp commit – fixed compiler warning but old tests are failing
Fixed old tests, fixed compiler warning
Added correct (?) invocation of args in thread_init. Still not found exception
It was failing to match on [e]. Changed to e, and now it’s giving a broken module
error: params don’t match
Still not working (broken module) but now using lookup_function and pattern
matching to remove option
Added a void ptr type for thread (kinda hacky for now but it’s for testing threads
). Also it’s now finding the function from the string name
Added thread testing script
THREADS NOW WORK IN SCRIPT!!!
Passing threading test
Fixed compiler warnings from pattern matching in codegen
Clean compilation without warnings
Removed codegen warnings, and some semant warnings

commit 08a4e105a2267891a38e76b4f280a4631bbe3413
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 00:54:52 2016 -0400

  fixed struct tests for let format

commit 152ab95f7e0c087cc914c0a5c2b176951b77a1d3
Merge: b4f812b 116094b
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Wed May 4 00:36:07 2016 -0400

  Merge add_structs

commit 116094b8ee508fd191c6d793cb14b9b5d6955c2a
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 10:57:04 2016 -0400

  Semantic checking to disallow circularly dependent structs

commit 490aa96cafcf6e93d9a5b981f37244cc5b0cb6c6
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 17:03:42 2016 -0400

  Fixed the stack overflow problem and updated tests

commit 41cb475f791cd6baf22baf68b02219be8a9a49b2
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:19:54 2016 -0400

  struct access works (messy)

commit eef6eb9fc000a844957f73dde0a56980a3b44ee0
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 30 14:14:10 2016 -0400

  Structs reach llvm failure point. need to clean up exception catching and matches.

commit b4f812b37b0788d6f4a6d09495f41bc1515488ec
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sat Apr 30 13:29:07 2016 -0400

  Flattened built-in function declarations so we don’t need extra variables.

commit 2c0b9c8b13e157863e8dca7f2b2f3346a6262c5b1
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 30 02:06:26 2016 -0400

  changed to named structs

commit aa095775c0b41e8776214758f2af8d3e142d1ea
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Fri Apr 29 22:21:28 2016 -0400

  added semant for struct field assignment
commit c90388e0f0d0eed30323be990eb29ec089fef474
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 21:19:46 2016 −0400
Created struct field index list

commit ef7a1054b5250ff47bd60f9fbd2a6e13396e1796
Author: PLT Student <axxu3795@gmail.com>
Date: Wed Apr 27 03:07:09 2016 −0400
ltype_of_type now includes struct types so structs can be allocated

commit e1b6f98760055b9f39c4f0a03606f06d94c2fc8b
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 26 18:48:27 2016 −0400
Cleaned up some warnings, still not sure what 42 is

commit 24ec2afb38e4bad5f7684ef663aa6d6993116dce
Author: PLT Student <axxu3795@gmail.com>
Date: Mon Apr 25 13:38:02 2016 −0400
Working error checking for struct

commit 95a5222e09300e7f5037e22c78bd4282cba9929f
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 25 13:01:23 2016 −0400
Added struct tests

commit 995258d61bd5f0db54369a7fc65dd6f188e6d415
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 25 12:57:47 2016 −0400
Working struct semant (throws not found exception)

commit 037886b737edccbe231b780897b7b31e67e36046
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 19:10:56 2016 −0400
match struct compiles

commit 6fa5581d2255d28b23d83efafdd0d0a868b96740
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sat Apr 23 18:49:31 2016 −0400
added broken struct accessor method

commit e91042b38c3dea600325c9239dd42b7cbfebf6a
Author: PLT Student <axxu3795@gmail.com>
Date: Sat Apr 23 17:47:47 2016 −0400
Adding check_access, still need to match inside

commit 3940c80078342f00879360619d0d5f5ad0bade57
Prepared to start adding structs to semant.

Gave struct types a string to hold for struct type name

Merge pull request #7 from DemocritusLang/change_syntax_order

Change syntax order with tests

Added dot operator syntax as a binop

Fixed bad string tests.

I forgot to make clean the last commit b/c I’m dumb

Compiles with the third struct list

Initial addition of struct to parsing

Fixed test-helloworld-assign declaration order
commit 12301820c5bbd32b55b29dfbd5a99068e62ee6b5
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 17 23:14:02 2016 -0400

Changed tests to add let keyword.

commit 3a626ec31e042cfa3cbcb8fc5410dd666fae12bea
Author: Amarto <aar2160@columbia.edu>
Date: Wed Apr 13 01:45:56 2016 -0400

Changed parser and scanner with LET keyword. Still working on tests

commit c6ecb302808b192e6e6f51360537556119a867ec
Author: Amarto <aar2160@columbia.edu>
Date: Tue Mar 15 00:33:01 2016 -0400

Temp commit — tried to change variable order but got SR error.

commit a37ba16c593be6be0d9980aec71b1d2b93eaf69e
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon Apr 11 13:05:53 2016 -0400

Added tentative install instructions (needs testing on Ubuntu 14.x and before)

commit 605b8bd1f66a1612d588ce0c5a52f107292d609
Merge: caa0380 0f67850
Author: Amy Xin Xu <axxu3795@gmail.com>
Date: Tue Apr 5 18:36:38 2016 -0400

Merge pull request #6 from DemocritusLang/strings_2

HelloWorld checkpoint!

: pizza:

commit 0f678507385ebacba0c05a41eac72ada0d9df015
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:29:24 2016 -0400

fixed failing function call test (semant only checks for print())

commit 22a7da1415c0b8dbb75e0a8766e50a48bac38
Author: PLT Student <axxu3795@gmail.com>
Date: Tue Apr 5 18:26:25 2016 -0400

Fixed missing printb

commit cb55b59f5981f69e28a347525e35ef1071020447
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:12:31 2016 -0400

Fixed tarball makefile builder for helloworld

commit 67ddab48071f203f447caf80f6906870af14e510
Author: = <kylelee.contact@gmail.com>
Date: Tue Apr 5 18:02:03 2016 -0400
Added test case for string assignment and printing.

commit daceabeb68dbd482000334162a0f797788554616
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 13:49:14 2016 -0400

Print hello world working. Tests that use printb() are failing, because I had to remove it from semant.ml temporarily.

commit f121b87bdcb8a6afac89d6374a9eb2705529d123
Author: Amarto <aar2160@columbia.edu>
Date: Mon Apr 4 02:22:46 2016 -0400

Compiling, but not passing tests.

commit 93833564906d28a36e6cd8303241e69a764ac2f0
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 18:11:16 2016 -0400

Tried moving strings to types...

commit d2ba3fd50270ec87f8c91a8bb0efd8ca3
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 17:56:06 2016 -0400

Added partial implementation of string literal.

commit 2fd8d9048fc9c7eb78e03d3ebdc2195ae4ad7403
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:53:52 2016 -0400

Added test and function for helloworld. Need string literal implementation.

commit 0471926a19d8626bab3140b6a2abcf588288620
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Sun Apr 3 16:27:19 2016 -0400

Changed Edwards’ print to be print_int to avoid confusion with our print implementation.

commit cca0380101c0ac9f657eb626b1930c5ca72b5af5e
Author: Emily Pakulski <enp2111@columbia.edu>
Date: Mon Mar 14 22:47:26 2016 -0400

Added function keyword to function declarations.

commit de3f696465ef9a95a737282d1affa7d1d812cad0
Author: Amarto Rajaram <aar2160@columbia.edu>
Date: Mon Mar 14 21:58:20 2016 -0400

Removed while keyword; replaced functionality with for.

commit d0829835a72243f858bbe2426ab81b04792ed883
Author: Amarto Rajaram <amarto.rajaram@gmail.com>
Date: Mon Mar 14 21:03:09 2016 -0400
516 Added Edwards’ tests back in.
517
518 commit 798f67d953e965dael282e679f6b0e5373982058
519 Author: Emily Pakulski <enp2111@columbia.edu>
520 Date: Fri Feb 26 11:45:05 2016 −0500
521
522 Edwards’ MicroC code with our README.

Final Report

1 commit 7c0f7a5ed8540cf9cb43f628a03db1c82e626dee
2 Author: Emily Pakulski <enp2111@columbia.edu>
3 Date: Wed May 11 21:48:51 2016 −0400
4
5 Making including easier.
6
7 commit 197d46b63145a1aef3b3675c19fb9231549ee74b
8 Author: Emily Pakulski <enp2111@columbia.edu>
9 Date: Wed May 11 21:41:01 2016 −0400
10
11 Added script that should allow us to dynamically generate source code files.
12
13 commit f62eb166d6f1faa62b2dd1ba9eebc15d378b29af
14 Author: Emily Pakulski <enp2111@columbia.edu>
15 Date: Wed May 11 20:51:45 2016 −0400
16
17 Tiny changes.
18
19 commit 89bb5c883803ea47bf5c1ac32ba671f92c4a260c
20 Author: Kyle Lee <kylelee.contact@gmail.com>
21 Date: Wed May 11 16:28:39 2016 −0400
22
23 Did testing chapter
24
25 commit 73839a25ced0849181664738054bcb79ff67230e
26 Author: Kyle Lee <kylelee.contact@gmail.com>
27 Date: Wed May 11 15:23:16 2016 −0400
28
29 finished up LRM
30
31 commit 1bf1b7f84d1a54e1c9fa34a4dd9df3de5afec2d7
32 Author: Kyle Lee <kylelee.contact@gmail.com>
33 Date: Wed May 11 14:48:34 2016 −0400
34
35 LRM basically fully done
36
37 commit d1d930bc3c77ab377132ddc3c75b37bc1d3425d7
38 Author: Kyle Lee <kylelee.contact@gmail.com>
39 Date: Wed May 11 03:46:38 2016 −0400
40
41 finished expressions and operators
42
43 commit a946ec6f6af1b9782ba5c1c895cb4f1c0d74b1d
44 Author: Kyle Lee <kylelee.contact@gmail.com>
45 Date: Wed May 11 01:16:41 2016 −0400

80
started updating LRM; fixed grammars and such

commit 6aa26ab70e7e8f52bf3579f373638796750759b9
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue May 10 03:24:54 2016 -0400

added plan, fixed some rendering issues

commit 95add631e41738269f470110aa807994286e1586
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Tue May 10 00:48:11 2016 -0400

Added more tutorial stuff, started architecture and other sections

commit 4f72336d4ef7f06f536d4f9c1febcde3dc55d56c
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Mon May 9 03:25:32 2016 -0400

added more introduction page info

commit 7468774d0ac3345921d545057fcf6214433824ef
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Sun May 8 20:04:33 2016 -0400

Added tutorial section; added sections to chapters

commit e016dac39ce5ac68ff4f7862eee25514a637a78
Author: Kyle Lee <kylelee.contact@gmail.com>
Date: Fri May 6 19:43:37 2016 -0400

initial commit for final report; added more chapters

commit 57517095899f2a1a4a0bb39c94339d5f5e5cfc6
Author: kyle—lee <kylelee.contact@gmail.com>
Date: Fri May 6 18:41:55 2016 -0400

Initial commit
5. Architecture Overview

Democritus’ compiler is built off of Professor Stephen Edwards’ MicroC compiler.
5.1 Compiler Overview

Several files make up the source code of the compiler. These include:

- `scanner.mll`: the OCamllex scanner.
- `ast.ml`: the abstract syntax tree, summarizing the overall structure of a Democritus program.
- `parser.mly`: the Ocamlyacc parser. Tokens from the scanner are parsed into the abstract syntax tree in the parser.
- `semant.ml`: the semantic analyzer.
- `codegen.ml`: the LLVM IR code generator.
- `democritus.ml`: the overarching OCaml program that calls the four main steps of the compiler.
- `bindings.c`: a C file that provides facilitates low-level operations that interact with the OS through C functions, such as for threads, which is then compiled to LLVM bytecode.

The Scanner

The scanner is simply a text scanner that parses text into various tokens, to then be interpreted by the parser. It is at this stage that irrelevant details (whitespace and comments) are discarded and some incorrect programs (untokenizable) are caught. The regular expressions used by the scanner are listed in the language reference chapter.

The Parser

The parser is a token scanner that converts the tokens read into a valid abstract syntax tree of to better represent the structure of the program. Here it discards irrelevant information, such as parenthesis and punctuation. If the program follows valid syntax, it will be parsed accordingly. Otherwise, compilation of code will yield a parse error.

The Semantic Analyzer

The semantic analyzer checks the correctness of user programs. It establishes that a program is well typed by building a symbol table and checking for consistency. For example, it will check whether variables are defined within a scope, whether we are dereferencing from a pointer, whether types of expressions match their uses in definitions and function calls, and whether structs are used correctly (a large modification we made was semantic checking for circular struct definitions).

The Code Generator

The code generator then takes in a definition of a program and builds the equivalent LLVM IR. It is in the code generator that we specify instructions regarding allocating memory and storing information, as well as accessing and manipulating said information. After the IR is generated, LLVM can optimize the code for specific platforms.
6. Testing

As with any software project, extensive testing was required to verify that all the features being implemented were working properly.

6.1 Integration Testing

Development and Testing Process

Development of new features required making them pass through the scanner, parser, semantic analyzer, and then code generation, in that order. When envisioning or developing a new feature, the testing process would proceed as follows:

1. Write example code implementing and utilizing the desired feature. (E.g. writing a struct definition in a new test file).

2. Modify the scanner (if needed) to read new tokens required by the new feature.

3. Modify the parser (usually needed) to change the grammar of the program to accept the new feature and pass necessary information (E.g. struct field names) to the semantic analyzer.

4. Modify the example code and test it so that only the ‘correct’ implementation of the feature passes the parser. Modify the scanner and parser until this step passes.

5. Modify the semantic analyzer so that it detects possible semantic issues that could arise from utilization of the new feature (E.g. accessing an undefined field in a struct or an undefined struct).

6. Modify the example code and test it so that only the ‘correct’ implementation passes the semantic analyzer; try testing multiple cases that should cause the analyzer to raise an error. Modify the semantic analyzer until this step passes.

7. Modify the code generator so that it generates the appropriate LLVM IR representing your new feature (E.g., allocating the correct amount of memory for new structs, building a map of struct field indexes, calling LLVM.build_struct_gep, etc.).

8. Modify your example code to utilize your feature and produce some visible effect or output (E.g. assigning a struct field, doing arithmetic on it, then printing it).

9. Test the code and ensure that running the program produces the expected output or effect; continue working on code generation until it does.

The process of writing test code, compiling it, and observing its output after being run as LLVM IR was the integration testing method that the Democritus team utilized throughout development. It helped ensure that whole features were working properly, and that the language, built up from multiple features, was still functioning correctly. Integration testing was done on all new features added to the language, as well as the existing ones from MicroC (such as basic variable assignment, conditional iteration, etc).
Aside: Unit Testing

Unit testing was not overly utilized in this development process, besides for testing to ensure that new features could pass certain layers of the compiler while working towards a passing integration test. This is because unit tests can still pass, while whole features lose vertical integration in the process of building up a compiler. This is because new features may often conflict with each other and the successful introduction of one feature could very well mean the breaking of another. This leads us to the test suite and automated regression testing.

6.2 The Test Suite and Automated Regression Testing

Democritus’ test suite was built upon MicroC’s automated regression testing package. Within the tests directory, there are dozens of integration test files for various language features as well as their expected stdout output. Additionally, there are several ‘fail’ tests used for showing invalid Democritus code as well as their expected error outputs.

The automated regression testing suite was used to quickly test all major language features by compiling each test, writing the error thrown by compilation (if it was a failure) or output of running the LLVM file (if compilation was a success) to a temporary file, and comparing that output to the expected output of each test with diff. The automated test was a shell script, invoked with ./testall.sh in the Democritus root directory.

The test suite was used frequently throughout development; while developing new features, team members would utilize the test suite to ensure that all major features of the language were still working. If a certain test in the suite failed, more verbose information about the test’s failure could be accessed in the testall.log file generated by the testing suite. The automated regression testing was crucial in ensuring that the language stayed consistent and working, and that our master branch remained ‘updated’ and error-free.
7. Lessons Learned

7.1 Amy

Trying to force new code to match legacy code can be more effort than it’s worth. It’s always okay to branch and attempt a larger rewrite if it will make everyone’s lives easier. Also, be sure to understand your own syntax when writing tests.

7.2 Emily

Remote teamwork can be tough. Writing tests that guarantee no regressions is surprisingly difficult, especially when testing against remote files.

7.3 Amarto

Debugging a compiler is like playing whack-a-mole – it’s much easier to write a script to isolate the action you’re trying to debug, and then gradually build it back into the compiler.

7.4 Kyle

In a team, try to play your strengths and figure out where you can help most effectively. If you think you can do something well or more efficiently than someone else, try to do it and save time – same thing works the other way (if pressed for time, let someone who knows how to do it manage it)
8. Code Listing

8.1 democritus.ml

```ocaml
let action = if Array.length Sys.argv > 1 then
  List.assoc Sys.argv.(1) [("-a", Ast); (* Print the AST only *)
    ("-l", LLVM_IR); (* Generate LLVM, don’t check *)
    ("-c", Compile) (* Generate, check LLVM IR *)
else Compile in
let lexbuf = Lexing.from_channel stdin in
let ast = Parser.program Scanner.token lexbuf in
Semant.check ast;
match action with
  Ast -> print_string (Ast.string_of_program ast)
| LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
| Compile -> let m = Codegen.translate ast in
  Llvm_analysis.assert_valid_module m;
  print_string (Llvm.string_of_llmodule m)
```

8.2 scanner.mll

```ocaml
rule token = parse

[' ' \t ' \r ' \n'] { token lexbuf } (* Whitespace *)

"//" { comment lexbuf } (* Comments *)

"/*" { multicomment lexbuf } (* Multiline comments *)

'|' { LPAREN }

'|)'} { RPAREN }

'|{'} { LBRACE }

'|'} { RBRACE }

'|;' { SEMI }

'|:' { COLON }

'|,' { COMMA }

'|'+' { PLUS }
```
and comment = parse
    | ''' { comment lexbuf }
    | _ { comment lexbuf }

and multicomment = parse
    | "/* { token lexbuf }
    | _ { multicomment lexbuf }

and read_string buf =
    parse
    | ''' { STRING (Buffer.contents buf) }
    | ''' '/* { Buffer.add_char buf '/'; read_string buf lexbuf }
    | ''' '/ { Buffer.add_char buf '/'; read_string buf lexbuf }
    | ''' 'b' { Buffer.add_char buf 'b'; read_string buf lexbuf }

8.3 parser.mly

/* Democritus, adapted from MicroC by Stephen Edwards Columbia University */
/* Ocamlyacc parser */

let first (a,_,_)=a;;
let second (_,b,_)=b;;
let third (_,_,c)=c;;

let first (a,_,_)=a;;
let second (_,b,_)=b;;
let third (_,_,c)=c;;

let first (a,_,_)=a;;
let second (_,b,_)=b;;
let third (_,_,c)=c;;

%token COLON SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS STAR DIVIDE MOD ASSIGN NOT DOT DEREF REF
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token LET RETURN IF ELSE FOR INT FLOAT BOOL VOID STRTYPE FUNCTION STRUCT VOIDSTAR
CAST TO SET
%token <string> STRING
%token <float> FLOATLITERAL
%token <int> LITERAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%nonassoc POINTER
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left STAR DIVIDE MOD
%right NOT NEG DEREF REF
%left DOT
%start program
%type <Ast.program> program

program:
decls EOF {$1}
decls:
    /* nothing */ { [], [], [] }
decls vdecl { ($2 :: first $1), second $1, third $1 }
decls fdecl { first $1, ($2 :: second $1), third $1 }
decls sdecl { first $1, second $1, ($2 :: third $1 ) }
fdecl:
    FUNCTION ID LPAREN formals_opt RPAREN typ LBRACE vdecl_list stmt_list RBRACE
    { { typ = $6; 
      fname = $2; 
      formals = $4; 
      locals = List.rev $8; 
      body = List.rev $9 } } 
formals_opt:
    /* nothing */ { [] } 
formal_list { List.rev $1 }
formal_list:
    ID typ 
    { (($2,$1]) } 
    | formal_list COMMA ID typ { ($4,$3 :: $1 } 
typ:
    INT { Int } 
    | FLOAT { Float } 
    | BOOL { Bool } 
    | VOID { Void } 
    | STRTYPE { MyString } 
    | STRUCT ID { StructType ($2) } 
    | VOIDSTAR { Voidstar } 
    | STAR %prec POINTER typ { PointerType ($2) } 
vdecl_list:
    /* nothing */ { [] } 
    | vdecl_list vdecl { $2 :: $1 } 
vdecl:
    LET ID typ SEMI { ($3, $2) } 
sdecl:
    STRUCT ID LBRACE vdecl_list RBRACE 
    { { sname = $2; 
      sformals = $4; 
    } } 
stmt_list:
    /* nothing */ { [] } 
    | stmt_list stmt { $2 :: $1 } 
stmt:
    expr SEMI { Expr $1 } 
    | RETURN SEMI { Return Noexpr } 
    | RETURN expr SEMI { Return $2 } 
    | LBRACE stmt_list RBRACE { Block(List.rev $2) } 
    | IF LPAREN expr RPAREN stmt $prec NOELSE { If($3, $5, Block([])) }
8.4 semant.ml

(* Democritus, adapted from MicroC by Stephen Edwards Columbia University *)
(* Semantic checking for compiler *)
open Ast
module StringMap = Map.Make(String)
module StringSet = Set.Make(String)

(* Semantic checking of a program. Returns void if successful, throws an exception if something is wrong.

Check each global variable, then check each function *)

let check (globals, functions, structs) =

(* Raise an exception if the given list has a duplicate *)
let report_duplicate exceptf list =
  let rec helper = function
  | n1 :: n2 :: t when n1 = n2 -> raise (Failure (exceptf n1))
  | _ :: t -> helper t
  | [] -> ()
  in helper (List.sort compare list)
in

(* Raise an exception if there is a recursive struct dependency*)
let find_sdecl_from_name struct_type_name =
  try List.find (fun s -> s.sname= struct_type_name) structs
  with Not_found -> raise (Failure("Struct of name " ^ struct_type_name ^ "not found."))
in
let rec check_recursive_struct_helper sdecl seen_set =
  let check if repeat struct_type_name =
    let found = StringSet.mem struct_type_name seen_set in
    if found then raise (Failure ("recursive struct definition"))
    else check_recursive_struct_helper (find_sdecl_from_name struct_type_name) (StringSet.add struct_type_name seen_set)
in
  let is_struct_field = function
    (StructType s, _) -> check if repeat s
    | _ -> ()
in
  List.iter (is_struct_field) sdecl.sformals
in
let check_recursive_struct sdecl =
  check_recursive_struct_helper sdecl StringSet.empty
in
let _ = List.map check_recursive_struct structs
in

(* Raise an exception if a given binding is to a void type *)
let check_not_void exceptf = function
  (Void, n) -> raise (Failure (exceptf n))
  | _ -> ()
in

(* Raise an exception of the given rvalue type cannot be assigned to the given lvalue type *)
let check_assign lvaluet rvaluet err =
  if (String.compare (string_of_typ lvaluet) (string_of_typ rvaluet)) == 0
  then lvaluet
  else raise err
(*if lvaluet == rvaluet then lvaluet else raise err*)

in

let match_struct_to_accessor a b =
    let sl = try List.find (fun s -> s.sname=a) structs
    with Not_found -> raise (Failure("Struct of name " ^ a ^ " not found.")) in
    try fst( List.find (fun s -> snd(s)=b) sl.sformals) with
    Not_found -> raise (Failure("Struct " ^ a ^ " does not have field " ^ b))
    in

let check_access lvaluet rvalues =
    match lvaluet with
    | StructType s ->
        match struct.to_accessor s rvalues
        | _ -> raise (Failure(string of_typ lvaluet ^ " is not a struct"))

in

(**** Checking Global Variables ****)
List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;
report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);
(**** Checking Functions ****)
if List.mem "append_strings" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function append_strings may not be defined")) else ();
if List.mem "int_to_string" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function int_to_string may not be defined")) else ();
if List.mem "print" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function print may not be defined")) else ();
if List.mem "thread" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function thread may not be defined")) else ();
if List.mem "exec_prog" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function exec_prog may not be defined")) else ();
if List.mem "free" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function free may not be defined")) else ();
if List.mem "malloc" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function malloc may not be defined")) else ();
if List.mem "open" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function open may not be defined")) else ();
if List.mem "close" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function close may not be defined")) else ();
if List.mem "read" (List.map (fun fd -> fd.fname) functions)
then raise (Failure("function read may not be defined")) else ();
if List.mem "write" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function write may not be defined")) else ();

if List.mem "lseek" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function lseek may not be defined")) else ();

if List.mem "sleep" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function sleep may not be defined")) else ();

if List.mem "request_from_server" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function request_from_server may not be defined")) else ();

if List.mem "memset" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function memset may not be defined")) else ();

report_duplicate (fun n -> "duplicate function " ^ n)
(List.map (fun fd -> fd.fname) functions);

(* Function declaration for a named function *)

let built_in_decls_funcs = [
  { typ = Void; fname = "print_int"; formals = [(Int, "x")];
    locals = []; body = []};

  { typ = Void; fname = "printb"; formals = [(Bool, "x")];
    locals = []; body = []};

  { typ = Void; fname = "print_float"; formals = [(Float, "x")];
    locals = []; body = []};

  { typ = Void; fname = "thread"; formals = [(MyString, "func");
        (MyString, "arg"); (Int, "nthreads")];
    locals = []; body = []};

  { typ = MyString; fname = "malloc"; formals = [(Int, "size")];
    locals = []; body = []};

  { typ = Int; fname = "open"; formals = [(MyString, "name");
        (Int, "flags"); (Int, "mode")];
    locals = []; body = []};

  { typ = Int; fname = "close"; formals = [(Int, "fd")];
    locals = []; body = []};

  { typ = Int; fname = "read"; formals = [(Int, "fd");
        (MyString, "buf"); (Int, "count")];
    locals = []; body = []};

  { typ = Int; fname = "write"; formals = [(Int, "fd");
        (MyString, "buf"); (Int, "count")];
    locals = []; body = []};

  { typ = Int; fname = "lseek"; formals = [(Int, "fd");
        (Int, "offset"); (Int, "whence")];
    locals = []; body = []};

  { typ = Int; fname = "sleep"; formals = [(Int, "seconds")];
    locals = []; body = []};

  { typ = Int; fname = "memset"; formals = [(MyString, "s");
        (Int, "val"); (Int, "size")];
    locals = []; body = []};
{ typ = MyString; fname = "request_from_server"; formals = [(MyString, "link")]; locals = []; body = [] }

{ typ = Int; fname = "exec_prog"; formals = [(MyString, "arg1"); (MyString, "arg2"); (MyString, "arg3")]; locals = []; body = [] }

{ typ = Void; fname = "free"; formals = [(MyString, "tofree")]; locals = []; body = [] }

{ typ = Void; fname = "append_strings"; formals = [(MyString, "str1"); (MyString, "str2")]; locals = []; body = [] }

{ typ = Void; fname = "int_to_string"; formals = [(Int, "n"); (MyString, "buf")]; locals = []; body = [] }

let built_in_decls_names = [ "print_int"; "printb"; "print_float"; "thread"; "malloc"; "open"; "close"; "read"; "write"; "lseek"; "sleep"; "memset"; "request_from_server"; "exec_prog"; "free"; "append_strings"; "int_to_string" ]

let built_in_decls = List.fold_right2 (StringMap.add) built_in_decls_names built_in_decls_funcs (StringMap.singleton "print"
  { typ = Void; fname = "print"; formals = [(MyString, "x")];
   locals = []; body = [] })

let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m) built_in_decls functions

let function_decl s = try StringMap.find s function_decls with Not_found -> raise (Failure ("unrecognized function " ^ s))

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

let check_function func =
  List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^ " in " ^ func.fname)) func.formals;

report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname) (List.map snd func.formals);
List.iter (check_not_void (fun n -> "illegal void local " ^ n ^ " in " ^ func.fname)) func.locals;

report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname) (List.map snd func.locals);

(* Type of each variable (global, formal, or local *)
let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m) StringMap.empty (globals @ func.formals @ func.locals )
in

let type_of_identifier s =
  try StringMap.find s symbols
  with Not_found -> raise (Failure ("undeclared identifier " ^ s))
in

(* Return the type of an expression or throw an exception *)
let rec expr = function
  | Literal _ -> Int
  | FloatLiteral _ -> Float
  | BoolLit _ -> Bool
  | MyStringLit _ -> MyString
  | Id s -> type_of_identifier s
  | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
    (match op with
      | Add | Sub | Mult | Div when t1 = Int && t2 = Int -> Int
      | Add | Sub | Mult | Div when t1 = Float && t2 = Float -> Float
      | Mod when t1 = Int && t2 = Int -> Int
      | Equal | Neq when t1 = t2 = Int -> Bool
      | Less | Leq | Greater | Geq when t1 = Int && t2 = Int -> Bool
      | And | Or when t1 = Bool && t2 = Bool -> Bool
      | . -> raise (Failure ("illegal binary operator " ^
        string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
        string_of_typ t2 ^ " in " ^ string_of_expr e))
      | Dotop(e1, field) -> let lt = expr e1 in
        check_access (lt) (field)
      | Castop(t, _) -> (*check later*) t
      | Unop(op, e) as ex -> let t = expr e in
        (match op with
          | Neg when t = Int -> Int
          | Not when t = Bool -> Bool
          | Deref -> (match t with
            | PointerType s -> s
            | . -> raise (Failure("cannot dereference a " ^ string_of_typ t))
          | -> PointerType(t)
          | . -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
            string_of_typ t ^ " in " ^ string_of_expr ex)))
          | Noexpr -> Void
          | Call(fname, actuals) as call -> let fd = function_decl fname in
            if List.length actuals != List.length fd.formals then
              raise (Failure ("expecting " ^ string_of_int
              (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
            else
        )
      | . -> raise (Failure ("illegal binary operator " ^
        string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
        string_of_typ t2 ^ " in " ^ string_of_expr e))
      )
  | Dotop(e1, field) -> let lt = expr e1 in
    check_access (lt) (field)
  | Castop(t, _) -> (*check later*) t
  | Unop(op, e) as ex -> let t = expr e in
    (match op with
      | Neg when t = Int -> Int
      | Not when t = Bool -> Bool
      | Deref -> (match t with
        | PointerType s -> s
        | . -> raise (Failure("cannot dereference a " ^ string_of_typ t))
      | -> PointerType(t)
      | . -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
        string_of_typ t ^ " in " ^ string_of_expr ex)))
      | Noexpr -> Void
      | Call(fname, actuals) as call -> let fd = function_decl fname in
        if List.length actuals != List.length fd.formals then
          raise (Failure ("expecting " ^ string_of_int
          (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
        else

      )
  | . -> raise (Failure ("illegal binary operator " ^
        string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
        string_of_typ t2 ^ " in " ^ string_of_expr e))
  )

96
let check assign ft et = Check ("illegal actual argument found " ^ string_of_typ et ^ "; expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e)))

fd.formals actuals;
fd.typ

| Assign(e1, e2) as ex ->

  mk_assign fd
  | Assign(e1, e2) as ex ->

  mk_assign fd

  let lt = type_of_identifier s and rt = expr e2 in
  check_assign (lt) (rt) (Failure ("illegal assignment " ^ string_of_typ lt ^ "; = " ^ string_of_typ rt ^ "; in " ^ string_of_expr ex))

| Unop(op, _) ->

  (match op with
    | Deref -> expr e2
    | _ -> raise (Failure ("whatever"))
  )

| Dotop (_, _) -> expr e2
| _ -> raise (Failure ("whatever"))

let check_bool_expr e = if expr e != Bool
  then raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
  else () in

(*) Verify a statement or throw an exception *)

let rec stmt = function

  Block sl
    | Return _ as s -> stmt s
    | Return _ :: _ -> raise (Failure ("nothing may follow a return"))
    | Block sl :: ss -> check_block (sl @ ss)
    | s :: ss -> stmt s ; check_block ss
    | [] -> ()
  in check_block sl

  Expr e -> ignore (expr e)
  | Return e -> let t = expr e in if t = func.typ then () else
  raise (Failure ("return gives " ^ string_of_typ t ^ "; expected " ^ string_of_typ func.typ ^ " in " ^ string_of_expr e))

  | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
  | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
    ignore (expr e3); stmt st
  | While(p, s) -> check_bool_expr p; stmt s
  in

  stmt (Block func.body)

  in

List.iter check_function functions

8.5 ast.ml
type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq | Greater | Geq | And | Or

type uop = Neg | Not | Deref | Ref

type typ = Int | Float | Bool | Void | MyString | StructType of string | Voidstar | PointerType of typ

type bind = typ * string

type expr =
  | Literal of int
  | FloatLiteral of float
  | BoolLit of bool
  | MyStringLit of string
  | Id of string
  | Binop of expr * op * expr
  | Dotop of expr * string
  | Castop of typ * expr
  | Unop of uop * expr
  (* | SAssign of expr * string * expr *)
  | Assign of expr * expr
  | Call of string * expr list
  | Noexpr

type stmt =
  | Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
  typ: typ;
  fname: string;
  formals: bind list;
  locals: bind list;
  body: stmt list;
}
type struct_decl = {
  sname: string;
  sformals: bind list;
}
type program = bind list * func_decl list * struct_decl list

(* Democritus, adapted from MicroC by Stephen Edwards Columbia University *)

(* Abstract Syntax Tree and functions for printing it *)

let string_of_op = function
  Add -> "+"
let string_of_uop = function
  Neg -> "-"
  Not -> "!
  Deref -> "*
  Ref -> "&

let rec string_of_typ = function
  Int -> "int"
  Float -> "float"
  Bool -> "bool"
  Void -> "void"
  MyString -> "string"
  StructType(s) -> "struct ^ s
  Voidstar -> "voidstar"
  PointerType(s) -> "pointerof ^ (string_of_typ s)

let rec string_of_expr = function
  Literal(l) -> string_of_int l
  FloatLiteral(l) -> string_of_float l
  BoolLit(true) -> "true"
  BoolLit(false) -> "false"
  MyStringLit(s) -> s
  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
  Dotop(e1, e2) -> string_of_expr e1 ^ ". " ^ e2
  Castop(t, e) -> "(" ^ string_of_typ t ^ ")" ^ string_of_expr e
  SAssign(e1, v, e2) -> string_of_expr(e1) ^ ". " ^ v ^ " = " ^ string_of_expr e2
  Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  Call(f, el) ->
    f ^ "(" ^ String.concat ", " ^ (List.map string_of_expr el) ^ ")"
  Noexpr -> ""

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
8.6 codegen.ml
A. StructType s -> Hashtbl.find struct_types s
A. MyString -> ptr t
A. Voidstar -> ptr t
A. PointerType t -> L.pointer_type ltype_of_typ t in
let populate_struct_type sdecl =
  let struct_t = Hashtbl.find struct_types sdecl.A.sname in
  let type_list = Array.of_list(List.map (fun (t, _) -> ltype_of_typ t) sdecl.A.
    sformals) in
  L.struct_set_body struct_t type_list true
  in
  ignore(List.map populate_struct_type structs);
let string.option_to_string = function
  None -> ""
  | Some(s) -> s
in
(* struct_field_index is a map where key is struct name and value is another map*)
(* in the second map, the key is the field name and the value is the index number*)
let struct_field_index_list =
  let handle_list m individual_struct =
    (* list of all field names for that struct*)
    let struct_field_name_list = List.map snd individual_struct.A.sformals in
    let increment n = n + 1 in
    let add_field_and_index (m, i) field_name =
      (* add each field and index to the second map*)
      (StringMap.add field_name (increment i) m, increment i) in
    (* struct_field_map is the second map, with key = field name and value = index*)
    let struct_field_map =
      List.fold_left add_field_and_index (StringMap.empty, -1) struct_field_name_list
      in
    (* add field map (the first part of the tuple) to the main map *)
    StringMap.add individual_struct.A.sname (fst struct_field_map) m
    in
    List.fold_left handle_list StringMap.empty structs
    in
    (* Declare each global variable; remember its value in a map *)
    let global_vars =
      let global_var m (t, n) =
        let init = 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
    List.fold_left global_vars StringMap.empty globals in
    let append_strings_t = L.function_type void_t [ [ ptr_t; ptr_t ] ] in
    let append_strings_func = L.declare_function "append_strings" append_strings_t
      the_module in
    let int_to_string_t = L.function_type void_t [ [ i32_t; ptr_t ] ] in
    let int_to_string_func = L.declare_function "int_to_string" int_to_string_t
      the_module in
    let printf_t = L.var_arg.function_type i32_t [ [ ptr_t ] ] in
    let printf_func = L.declare_function "printf" printf_t the_module in
    let execl_t = L.var_arg.function_type i32_t [ [ ptr_t ] ] in
    let execl_func = L.declare_function "exec_prog" execl_t the_module in

101
let free_t = L.function_type void_t [ | ptr_t | ] in
let free_func = L.declare_function "free" free_t the_module in
let malloc_t = L.function_type ptr_t [ | i32_t | ] in
let malloc_func = L.declare_function "malloc" malloc_t the_module in
let request_from_server_t = L.function_type ptr_t [ | ptr_t | ] in
let request_from_server_func = L.declare_function "request_from_server"
                    request_from_server_t the_module in
let memset_t = L.function_type ptr_t [ | ptr_t; i32_t; i32_t | ] in
let memset_func = L.declare_function "memset" memset_t the_module in
(* File I/O functions *)
let open_t = L.function_type i32_t [ | ptr_t; i32_t; i32_t | ] in
let open_func = L.declare_function "open" open_t the_module in
let close_t = L.function_type i32_t [ | i32_t | ] in
let close_func = L.declare_function "close" close_t the_module in
let read_t = L.function_type i32_t [ | i32_t; ptr_t; i32_t | ] in
let read_func = L.declare_function "read" read_t the_module in
let write_t = L.function_type i32_t [ | i32_t; ptr_t; i32_t | ] in
let write_func = L.declare_function "write" write_t the_module in
let lseek_t = L.function_type i32_t [ | i32_t; i32_t; i32_t | ] in
let lseek_func = L.declare_function "lseek" lseek_t the_module in
let sleep_t = L.function_type i32_t [ | i32_t | ] in
let sleep_func = L.declare_function "sleep" sleep_t the_module in
let default_t = L.function_type ptr_t [ | ptr_t | ] in
let default_func = L.declare_function "default_start_routine" default_t the_module in
let param_ty = L.function_type ptr_t [ | ptr_t | ] in (* a function that returns
                void* and takes as argument void* *)
let param_ptr = L.pointer_type param_ty in
let thread_t = L.function_type void_t [ | param_ptr; ptr_t; i32_t | ] in (*a function
                that returns void and takes (above) and a voidstar and an int *)
let thread_func = L.declare_function "init_thread" thread_t the_module in
(* Define each function (arguments and return type) so we can call it *)
let functiondecls =
    let functiondecl m fdecl =
        let name = fdecl.A.fname
        and formal_types =
            Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals)
        in let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in
            StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left functiondecl StringMap.empty functions in
(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.A.fname 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 in
  let float_format_str = L.build_global_stringptr "%f\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.buildalloca (ltype of typ t) n builder in
    ignore (L.build_store p local builder);
    StringMap.add n local m in

    let add_local m (t, n) =
      let local_var = L.buildalloca (ltype of typ t) n builder in
      StringMap.add n local_var m in

    let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
      (Array.to_list (L.params the_function)) in
    List.fold_left add_local formals fdecl.A.locals in

  (* Return the value for a variable or formal argument *)
  let lookup n = try StringMap.find n local_vars
    with Not_found -> try StringMap.find n global_vars
    with Not_found -> raise (Failure("undeclared variable " ^ n))
  in

  (* Construct code for an expression; return its value *)
  let rec llvalue expr getter builder = function
    A.Id s -> lookup s
    | A.Dotop(e1, field) ->
      (match e1 with
        A.Id s -> let etype = fst(
          try List.find (fun t->snd(t)=s) fdecl.A.locals
          with Not_found -> raise (Failure("Unable to find" ^ s ^ "in dotop")))
        in
          (try match etype with
            A.StructType t->
            let index_number_list = StringMap.find t struct_field_index_list in
            let index_number = StringMap.find field index_number_list in
            let struct_llvalue = lookup s in
            let access_llvalue = L.build_struct_gep struct_llvalue index_number "
              dotop_terminal" builder in
            access_llvalue
            | _ -> raise (Failure("No structype."))
            with Not_found -> raise (Failure("unable to find" ^ s))) )
      | _ as el_expr -> let el’_llvalue = llvalue_expr_getter builder el_expr in
      let loaded_el’ = expr builder el_expr in
      let el’_lltype = L.typeof loaded_el’ in
      let el’_struct_name_string_option = L.struct_name el’_lltype in
      let el’_struct_name_string = string_option_to_string el’
        _struct_name_string_option in
  in
let index_number_list = StringMap.find el'_struct_name_string
    struct_field_index_list in
let index_number = StringMap.find field index_number_list in
let access_llvalue = L.build_struct_gep el'_llvalue index_number "gep_in_dotop"
    builder in
    access_llvalue )

| A.Unop(op, e) ->
  (match op with
  A.Deref ->
    let e1' = expr builder e in
    let e1loaded = L.build_load e1'value "loaded_deref" builder in
    e1loaded
    | . -> raise (Failure("nooo"))
  )
  | _ -> raise (Failure("in llvalue_expr_getter but not a dotop!"))

and
expr builder = function
A.Literal i -> L.const_int i32_t i
(* |
  | A.MyStringLit str -> L.const_stringz context str *)
| A.FloatLiteral f -> L.const_float f f
| A.MyStringLit str -> L.build_global_stringptr str "tmp" builder
| A.BoolLit b -> L.const_int 1l_t (if b then 1 else 0)
| A.Noexpr -> L.const_int i32_t 0
| A.Id s -> L.build_load (lookup s) s builder
| A.Binop (el, op, e2) ->
  let el' = expr builder el
  and e2' = expr builder e2 in
  (match op with
  A.Add -> (let el_type_string = L.string_of_lltype (L.type_of el') in
    (match el_type_string with
      "double" -> L.build_fadd
      "i32" -> L.build_add
      | . -> raise(Failure("Can only add ints or floats")) ))
  | A.Sub -> (let el_type_string = L.string_of_lltype (L.type_of el') in
    (match el_type_string with
      "double" -> L.build_fsub
      "i32" -> L.build_sub
      | . -> raise(Failure("Can only subtract ints or floats")) ))
  | A.Mult -> (let el_type_string = L.string_of_lltype (L.type_of el') in
    (match el_type_string with
      "double" -> L.build_fmul
      "i32" -> L.build_mul
      | . -> raise(Failure("Can only multiply ints or floats")) ))
  | A.Div -> (let el_type_string = L.string_of_lltype (L.type_of el') in
    (match el_type_string with
      "double" -> L.build_fdiv
      "i32" -> L.build_sdiv
      | . -> raise(Failure("Can only divide ints or floats")) ))
  | A.Mod -> L.build_srem
  | A.And -> L.build_and
  | A.Or -> L.build_or
  | A.Equal -> L.build_icmp L.Icmp.Eq
  | A.Neq -> L.build_icmp L.Icmp.Ne
  | A.Less -> L.build_icmp L.Icmp.Slt
| A.Leq    -> L.build_icmp L.Icmp.Sle |
| A.Greater -> L.build_icmp L.Icmp.Sgt |
| A.Geq    -> L.build_icmp L.Icmp.Sge |
| e1' e2' "tmp" builder |
| A.Dotop(e1, field) -> let _ = expr builder e1 in |
| (match e1 with |
| A.Id s -> let etype = fst( |
| try List.find (fun t->snd(t)=s) fdecl.A.locals |
| with Not_found -> raise (Failure("Unable to find" ^ s ^ "in dotop"))) |
| in |
| (try match etype with |
| A.StructType t-> |
| let index.number.list = StringMap.find t struct.field.index.list in |
| let index.number = StringMap.find field index.number.list in |
| let struct.llvalue = lookup s in |
| let access.llvalue = L.build_struct_gep struct.llvalue index.number "gep_in_dotop" builder in |
| let loaded_access = L.build_load access.llvalue "loaded_dotop_terminal" builder in |
| loaded_access |
| | _ -> raise (Failure("No structype.")) |
| with Not_found -> raise (Failure("unable to find" ^ s)) } |
| | _ as el_expr -> let el'.llvalue = llvalue_expr_getter builder el_expr in |
| let loaded_el' = expr builder el_expr in |
| let el'.lltype = L.type_of loaded_el' in |
| let el'.struct_name_string_option = L.struct_name el'.lltype in |
| let el'.struct_name_string = string_option_to_string el'.struct_name_string_option in |
| let index.number_list = StringMap.find el'.struct_name_string |
| struct.field_index_list in |
| let index.number = StringMap.find field index.number_list in |
| let access.llvalue = L.build_struct_gep el'.llvalue index.number "gep_in_dotop" |
| builder in |
| L.build_load access.llvalue "loaded_dotop" builder |
| ) |
| | A.Unop(op, e) -> |
| let e' = expr builder e in |
| (match op with |
| A.Neg    -> L.build_neg e' "tmp" builder |
| | A.Not    -> L.build_not e' "temp" builder |
| | A.Deref  -> let e_loaded = L.build_load e' "loaded_deref" builder in |
| e_loaded |
| | A.Ref    -> let e.llvalue = (llvalue_expr_getter builder e) in |
| e.llvalue |
| ) |
| | A.Castop(ast_cast_type, e) -> |
| let cast.lltype = ltype_of_typ ast_cast_type in |
| let e.llvalue = expr builder e in |
| L.build_pointercast e.llvalue cast.lltype "plz" builder |
| | A.Assign (lhs, e2) -> let e2' = expr builder e2 in |
| (match lhs with |
| A.Id s ->ignore (L.build_store e2' (lookup s) builder); e2' |
| A.Dotop (e1, field) -> |
| (match e1 with |
| A.Id s -> let eltyp = fst( |

105
try List.find (fun t -> snd(t) = s) fdecl.A.locals
  with Not_found -> raise(Failure("unable to find" ^ s ^ " in Sassign"))

  (match eltyp with
    A.StructType t -> (try
      let index_number_list = StringMap.find t struct_field_index_list in
      let index_number = StringMap.find field index_number_list in
      let struct llvalue = lookup s in
      let index_number = StringMap.find field index_number_list in
      let access llvalue = L.build_struct_gep struct_llvalue index_number
        field builder in
        (try (ignore(L.build_store e2' access llvalue builder);e2')
          with Not_found -> raise (Failure("unable to store " ^ t )) )
      with Not_found -> raise (Failure("unable to find" ^ s)) )
    | _ -> raise (Failure("StructType not found."))) )
   let llvalue = (llvalue_expr_getter builder e) in
   let loaded = L.build_load e llvalue "loaded deref" builder in
   let _ = L.build_store e2' loaded builder in
   e2'

| A.Unop(op, e) ->
  (match op with
   A.Deref ->
     let e.llvalue = (llvalue_expr_getter builder e) in
     let e.loaded = L.build_load e.llvalue "loaded deref" builder in
     let _ = L.build_store e2' e.loaded builder in
     e2'
     | _ -> raise (Failure("nooo"))

   | A.Call ("print_int", [e]) | A.Call ("printb", [e]) ->
     L.build_call printf_func [] int_format_str ; (expr builder e) [] "printf" builder
   | A.Call ("print_float", [e]) ->
     L.build_call printf_func [] float_format_str; (expr builder e) [] "printf" builder
   | A.Call ("print", [e]) ->
     L.build_call printf_func [] (expr builder e) [] "printf" builder
   | A.Call ("append_strings", e) ->
     let evald_expr = List.map (expr builder)e in
     let evald_expr_arr = Array.of_list evald_expr in
     L.build_call append_strings_func evald_expr_arr "" builder
   | A.Call ("int_to_string", e) ->
     let evald_expr = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call int_to_string.func evald_expr_arr "" builder

| A.Call ("exec_prog", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call execl.func evald_expr_arr "exec_prog" builder

| A.Call("free", e) ->
L.build.call free.func (Array.of_list (List.map (expr builder) e)) "" builder

| A.Call ("malloc", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call malloc.func evald_expr_arr "malloc" builder

| A.Call ("memset", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call memset.func evald_expr_arr "memset" builder

(* File I/O functions *)
| A.Call("open", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call open.func evald_expr_arr "open" builder

| A.Call("close", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call close.func evald_expr_arr "close" builder

| A.Call("read", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call read.func evald_expr_arr "read" builder

| A.Call("write", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call write.func evald_expr_arr "write" builder

| A.Call("lseek", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call lseek.func evald_expr_arr "lseek" builder

| A.Call("sleep", e) ->
let evald_expr_list = List.map (expr builder)e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build.call sleep.func evald_expr_arr "sleep" builder

| A.Call ("thread", e) ->
(* L.build.call printf.func [| int_format_str ; l.const.int i32.t 8 |] "printf"
let evald_expr_list = List.map (expr builder) e in
let target_func_strptr = List.hd evald_expr_list in
let target_func_str = L.string_of_llvalue target_func_strptr in
let get_string v = match v with
  | A.MyStringLit i -> i
  | _ -> "" in
let target_func_str = get_string (List.hd e) in
(let target_func_str = Option.default "" Some(target_func_str_opt) in *)
let target_func_llvalue_opt = L.lookup_function target_func_str the_module in
let deopt x = match x with
  | Some f -> f
  | None -> default_func in
let target_func_llvalue = deopt target_func_llvalue_opt in
let remaining_list = List.tl evald_expr_list in
let new_arg_list = Array.of list new_arg_list in
L.build_call thread_func
new_arg_arr "" builder

| A.Call ("request_from_server", e) ->
let evald_expr_list = List.map (expr builder) e in
let evald_expr_arr = Array.of_list evald_expr_list in
L.build_call request_from_server_func evald_expr_arr "request_from_server" builder

| A.Call (f, act) ->
  let (fdef, fdecl) = StringMap.find f function_decls in
  let actuals = List.rev (List.map (expr builder) (List.rev act)) in
  let result = (match fdecl.A.typ with
    A.Void -> L.build_ret void builder
    | _ -> f "result") in
L.build_call fdef (Array.of_list actuals) result builder

(| SOMEInvoke "f builder" if the current block doesn’t already
have a terminal (e.g., a branch). *)
let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
  | Some _ -> ()
  | None -> ignore (f builder) in

(* Build the code for the given statement; return the builder for
the statement’s successor *)
let rec stmt builder = function
  A.Block sl -> List.fold_left stmt builder sl
  | A.Expr e -> ignore (expr builder e); builder
  | A.Return e -> ignore (match fdecl.A.typ with
    A.Void -> L.build_ret void builder
    | _ -> L.build_ret (expr builder e) builder);
    A.If (predicate, then_stmt, else_stmt) ->
    let bool_val = expr builder predicate in
    let merge_bb = L.append_block context "merge" the_function in
    add_terminal (stmt (L.builder at_end context then_bb) then_stmt)
let else_bb = L.append_block context "else" the_function in
add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
(L.build_br merge_bb);
ignore (L.build_cond_br bool_val then_bb else_bb builder);
L.builder_at_end context merge_bb

| A.While (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

| A.For (e1, e2, e3, body) -> stmt builder
  ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )
in

(* Build the code for each statement in the function *)
let builder = stmt builder (A.Block fdecl.A.body) in

(* Add a return if the last block falls off the end *)
add_terminal builder (match fdecl.A.typ with
  A.Void -> L.build_ret_void
  | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in
List.iter build_function_body functions;

let llmem = Llvm.MemoryBuffer.of_file "bindings.bc" in
let llm = Llvm_bitreader.parse_bitcode context llmem in
ignore(Llvm_linker.link_modules the_module llm Llvm_linker.Mode.PreserveSource);
the_module

8.7 bindings.c

#include <pthread.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <errno.h>
#include <netinet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUFSIZE 4096

void append_strings(void *str1, void *str2)
{
    strcat((char *)str1, (char *)str2);
}

void int_to_string(int n, void *buf)
{
    sprintf(buf, "%d", n);
}

int exec_prog(void *str1, void *str2, void *str3)
{
    execl((char *)str1, (char *)str2, (char *)str3, NULL);
    return 0;
}

/*
 * Given a URL, send a GET request.
*/

//void *get_request(void *url, void *filePath)
//void *request_from_server(void *urlVoid)
{
    // www.xkcd.com/index.html
    char *urlStr = (char *)urlVoid;
    int idxslash = strchr(urlStr, '/') - urlStr;
    char *url = malloc(idxslash + 1);
    char *filePath = malloc(strlen(urlStr) - (idxslash) + 1);
    memset(url, 0, idxslash - 1);
    memset(filePath, 0, strlen(urlStr) - (idxslash));

    strncat(url, urlStr, idxslash);
    strncat(filePath, urlStr + idxslash, strlen(urlStr) - (idxslash));
    char *fileName = strrchr(urlStr, '/') + 1;

    char *serverIP;
    int sock; // socket we connect to remote on
    struct sockaddr_in serverAddr;
    struct hostent *he;
    char recvbuf[BUFSIZE];

    if ((he = gethostbyname((char *)url)) == NULL) {
        fprintf(stderr, "gethostbyname() failed.");
        exit(1);
    }

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        fprintf(stderr, "socket() failed.");
        exit(1);
    }
serverIP = inet_ntoa(*(struct in_addr *)&h_addr);
memset(&serverAddr, 0, sizeof(serverAddr));
serverAddr.sin_addr.s_addr = inet_addr(serverIP);
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(80);

int connected = connect(sock, (struct sockaddr *)&serverAddr, sizeof(serverAddr));
if (connected < 0) {
    fprintf(stderr, "connect() failed.");
    exit(1);
}

// send HTTP request
if (((char *) url)[strlen((char *) url) - 1] == '/') {
    strcat(url, "index.html");
}

snprintf(recvbuf, sizeof(recvbuf),
    "GET %s HTTP/1.0\r\n"
    "Host: %s:%s\r\n"
    "\r\n",
    filePath, url, "80");
if (send(sock, recvbuf, strlen(recvbuf), 0) != strlen(recvbuf)) {
    fprintf(stderr, "send() failed.");
    exit(1);
}

// wrap the socket with a FILE* so that we can read the socket using fgets()
FILE *fd;
if ((fd = fdopen(sock, "rb")) == NULL) {
    fprintf(stderr, "fdopen() failed.");
    exit(1);
}

/* check header for valid protocol and status code */
if (fgets(recvbuf, sizeof(recvbuf), fd) == NULL) {
    fprintf(stderr, "server terminated connection without response.");
    exit(1);
}
if (strncmp("HTTP/1.0 ", recvbuf, 9) != 0 && strncmp("HTTP/1.1 ", recvbuf, 9) != 0) {
    fprintf(stderr, "unknown protocol response: %s.", recvbuf);
    exit(1);
}
if (strncmp("200", recvbuf + 9, 3) != 0) {
    fprintf(stderr, "request failed with status code %s.", recvbuf);
    exit(1);
}
/* ignore remaining header lines */
do {
    memset(recvbuf, 0, BUFSIZE);
    if (fgets(recvbuf, sizeof(recvbuf), fd) == NULL) {
        fprintf(stderr, "server terminated connection without sending file.");
        exit(1);
    }
} while (strcmp("\r\n", recvbuf) != 0);
char *filePathName = malloc(100);
memset(filePathName, 0, 100);
char *last_slash;
if ((last_slash = strrchr(filePath, '/')) != NULL) {
    if (strlen(last_slash) == 1) {
        strcpy(filePathName, "index.html");
    } else {
        strcpy(filePathName, last_slash + 1);
    }
}

/* open and read into file */
printf("%s\n", filePathName);
FILE *outputFile = fopen(filePathName, "wb");
if (outputFile == NULL) {
    fprintf(stderr, "fopen() failed.");
    exit(1);
}

size_t n;
int total = 0;
memset(recvbuf, 0, BUFSIZE);
printf("buffer contents: %s\n", recvbuf);
while ((n = fread(recvbuf, 1, BUFSIZE, fd)) > 0) {
    if (fwrite(recvbuf, 1, n, outputFile) != n) {
        fprintf(stderr, "fwrite() failed.");
        exit(1);
    }
    memset(recvbuf, 0, BUFSIZE);
    total += n;
}
fprintf(stderr, "total bytes written: %d\n", total);
if (ferror(fd)) {
    fprintf(stderr, "fread() failed.");
    exit(1);
}
fclose(outputFile);
fclose(fd);
return NULL;
}

void *default_start_routine(void *arg)
{
    return arg;
}

void init_thread(void *(*start_routine) (void *, void *, int nthreads))
{
    pthread_t thread[nthreads];
    int i;
for (i = 0; i < nthreads; i++) {
    pthread_create(&thread[i], NULL, start_routine, arg);
}

for (i = 0; i < nthreads; i++) {
    pthread_join(thread[i], NULL);
}
9. Tests and Output

fail-assign1.dem

```plaintext
function main() int
{
    let i int;
    let b bool;
    i = 42;
    i = 10;
    b = true;
    b = false;
    i = false; /* Fail: assigning a to bool an integer */
}
```

fail-assign1.err

```
Fatal error: exception Failure("illegal assignment int = bool in i = false")
```

fail-assign2.dem

```plaintext
function main() int
{
    let i int;
    let b bool;
    b = 48; /* Fail: assigning an integer to a bool */
}
```

fail-assign2.err

```
Fatal error: exception Failure("illegal assignment bool = int in b = 48")
```

fail-assign3.dem

```plaintext
function myvoid() void
{
    return;
}

function main() int
{
    let i int;
    i = myvoid(); /* Fail: assigning a to void an integer */
}
```
fail-assign3.err

1 Fatal error: exception Failure("illegal assignment int = void in i = myvoid()")

fail-dead1.dem

1 function main() int
2 {
3   let i int;
4   i = 15;
5   return i;
6   i = 32; /* Error: code after a return */
7 }

fail-dead1.err

1 Fatal error: exception Failure("nothing may follow a return")

fail-dead2.dem

1 function main() int
2 {
3   let i int;
4   {
5     i = 15;
6     return i;
7   }
8   i = 32; /* Error: code after a return */
9 }

fail-dead2.err

1 Fatal error: exception Failure("nothing may follow a return")

fail-expr1.dem

1 let a int;
2 let b bool;
3
4 function foo(c int, d bool) void
5 {
6   let dd int;
7   let e bool;
8   a + c;
9   c - a;
10  a * 3;
11  c / 2;
12  d + a; /* Error: bool + int */
13 }
14
15 function main() int
16 {
17   return 0;
18 }
fail-expr1.err
1 Fatal error: exception Failure("illegal binary operator bool + int in \texttt{d + a}\")

fail-expr2.dem
1 let \texttt{a int};
2 let \texttt{b bool};
3 4 function \texttt{foo(c int, d bool) void}
5 { 6 let \texttt{d int};
7 let \texttt{e bool};
8 \texttt{b + a}; /\texttt{* Error: bool + int */}
9 }
10
11 function \texttt{main() int}
12 {
13 \texttt{return 0;}
14 }

fail-expr2.err
1 Fatal error: exception Failure("illegal binary operator bool + int in \texttt{b + a}\")

fail-for1.dem
1 function \texttt{main() int}
2 {
3 let \texttt{i int};
4 \texttt{for ( ; true ; ) {} /* OK: Forever */}
5 \texttt{for (i = 0 ; i < 10 ; i = i + 1) { 6 \texttt{if (i == 3) return 42;}
7 \texttt{}} 8 \texttt{for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */}
9 \texttt{return 0;}
10 }

fail-for1.err
1 Fatal error: exception Failure("undeclared identifier \texttt{j}\")

fail-for2.dem
1 function \texttt{main() int}
2 {
3 let \texttt{i int};
4 \texttt{for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */}
5 \texttt{return 0;}
6 }

fail-for2.err
1 Fatal error: exception Failure("undeclared identifier \texttt{j}\")
fail-for2.err

1 Fatal error: exception Failure("undeclared identifier j")

fail-for3.dem

1 function main() int
2 {
3     let i int;
4
5     for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */
6
7     return 0;
8 }

fail-for3.err

1 Fatal error: exception Failure("expected Boolean expression in i")

fail-for4.dem

1 function main() int
2 {
3     let i int;
4
5     for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */
6
7     return 0;
8 }

fail-for4.err

1 Fatal error: exception Failure("undeclared identifier j")

fail-for5.dem

1 function main() int
2 {
3     let i int;
4
5     for (i = 0; i < 10 ; i = i + 1) {
6         foo(); /* Error: no function foo */
7     }
8
9     return 0;
10 }

fail-for5.err

1 Fatal error: exception Failure("unrecognized function foo")

fail-for-as-while1.dem
function main() int
{
    let i int;
    for (true) {
        i = i + 1;
    }
    for (42) {
        // Should be boolean */
        i = i + 1;
    }
}

fail-for-as-while1.err

Fatal error: exception Failure("expected Boolean expression in 42")

fail-for-as-while2.err

Fatal error: exception Failure("unrecognized function foo")

fail-func1.err

Fatal error: exception Failure("duplicate function bar")
fail-func2.dem

```plaintext
1  function foo(a int, b bool, c int) int { }
2
3  function bar(a int, b bool, a int) void {} /* Error: duplicate formal a in bar */
4
5  function main() int
6  {  return 0;
7  }
```

fail-func2.err

```plaintext
1  Fatal error: exception Failure("duplicate formal a in bar")
```

fail-func3.dem

```plaintext
1  function foo(a int, b bool, c int) int { }
2
3  function bar(a int, b void, c int) void {} /* Error: illegal formal void b */
4
5  function main() int
6  {  return 0;
7  }
```

fail-func3.err

```plaintext
1  Fatal error: exception Failure("illegal void formal b in bar")
```

fail-func4.dem

```plaintext
1  function foo() int {} 
2
3  function bar() void {}
4
5  function print() int {} /* Should not be able to define print */
6
7  function baz() void {}
8
9  function main() int
10  {  return 0;
11  }
```

fail-func4.err

```plaintext
1  Fatal error: exception Failure("function print may not be defined")
```

fail-func5.dem

```plaintext
1  function foo() int {}
2
3  function bar() int {}
4   let a int;
```
let b void; /* Error: illegal local void b */
let c bool;

return 0;
}

function main() int
{
  return 0;
}

fail-func5.err

Fatal error: exception Failure("illegal void local b in bar")

fail-func6.err

function foo(a int, b bool) void
{
}

function main() int
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}

fail-func6.err

Fatal error: exception Failure("expecting 2 arguments in foo(42)")

fail-func7.err

function foo(a int, b bool) void
{
}

function main() int
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}

fail-func7.err

Fatal error: exception Failure("expecting 2 arguments in foo(42, true, false)")

fail-func8.err

function foo(a int, b bool) void
{
}

function bar() void
{
}
function main() int
{
    foo(42, true);
    foo(42, bar()); /* and int void, not and int bool */
}

fail-func8.err

1 Fatal error: exception Failure("illegal actual argument found void expected bool in bar()")

fail-func9.dem

1 function foo(a int, b bool) void
2 {
3 }
4 function main() int
5 {
6    foo(42, true);
7    foo(42, 42); /* Fail: int, not bool */
8 }

fail-func9.err

1 Fatal error: exception Failure("illegal actual argument found int expected bool in 42")

fail-global1.dem

1 let c int;
2 let b bool;
3 let a void; /* global variables should not be void */
4
5 function main() int
6 {
7    return 0;
8 }

fail-global1.err

1 Fatal error: exception Failure("illegal void global a")

fail-global2.dem

1 let b int;
2 let c bool;
3 let a int;
4 let b int; /* Duplicate global variable */
5 function main() int
6 {
7    return 0;
8 }
9}
fail-global2.err

1 Fatal error: exception Failure("duplicate global b")

fail-if1.dem

1 function main() int
2 {
3   if (true) {}
4   if (false) {} else {}
5   if (42) {} /* Error: non-predicate bool */
6 }

fail-if1.err

1 Fatal error: exception Failure("expected Boolean expression in 42")

fail-if2.dem

1 function main() int
2 {
3   if (true) {
4     foo; /* Error: undeclared variable */
5   }
6 }

fail-if2.err

1 Fatal error: exception Failure("undeclared identifier foo")

fail-if3.dem

1 function main() int
2 {
3   if (true) {
4     42;
5   } else {
6     bar; /* Error: undeclared variable */
7   }
8 }

fail-if3.err

1 Fatal error: exception Failure("undeclared identifier bar")

fail-nomain.dem

fail-nomain.err

fail-nomain.err

1 Fatal error: exception Failure("unrecognized function main")

fail-return1.dem
function main() int {
    return true; /* Should return int */
}

fail-return1.err

Fatal error: exception Failure("return gives bool expected int in true")

fail-return2.dem

function foo() void {
    if (true) return 42; /* Should return void */
    else return;
}

function main() int {
    return 42;
}

fail-return2.err

Fatal error: exception Failure("return gives int expected void in 42")

fail-struct-circular.dem

struct A {
    let b struct B;
}

struct B {
    let c struct C;
}

struct C {
    let a struct A;
}

function main() int {
    let b bool;
    let x struct A;
    print("hello world\n");
    return 0;
}

fail-struct-circular.err

Fatal error: exception Failure("recursive struct definition")

test-arith1.dem
function main() int
{
  print_int(39 + 3);
  return 0;
}

test-arith1.out

42

test-arith2.dem

function main() int
{
  print_int(1 + 2 * 3 + 4);
  return 0;
}

test-arith2.out

11

test-arith3.dem

function foo(a int) int
{
  return a;
}

function main() int
{
  let a int;
  a = 42;
  a = a + 5;
  print_int(a);
  return 0;
}

test-arith3.out

47

test-fib.dem

function fib(x int) int
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

function main() int
{
  print_int(fib(0));
  print_int(fib(1));
  print_int(fib(2));
  print_int(fib(3));
}
```python
print_int(fib(4));
print_int(fib(5));
return 0;
}

test-fib.out

1 1
2 1
3 2
4 3
5 5
6 8

test-fileops.dem

function main() int
{
    let fd int;
    let malloced string;
    fd = open("tests/HELLOOOOOO.txt", 66, 384);
    write(fd, "hellooo!\n", 9);
    malloced = malloc(10);
    lseek(fd, 0, 0);
    read(fd, malloced, 10);
    print(malloced);
    free(malloced);
    return 0;
}

test-fileops.out

hellooo!

test-float.dem

function main() int
{
    let a float;
    let b float;
    let c float;
    a = 10.0;
    b = 0.5;
    print_float(a + b);
    print_float(a - b);
    print_float(a * b);
    print_float(a / b);
    return 0;
}

test-float.out
test-for1.dem

```plaintext
def function main() int
{
    let i int;
    for (i = 0; i < 5; i = i + 1) {
        print int(i);
    }
    print int(42);
    return 0;
}
```

`.test-for1.out`

```
0
1
2
3
4
42
```

test-for2.dem

```plaintext
def function main() int
{
    let i int;
    i = 0;
    for (; i < 5; ) {
        print int(i);
        i = i + 1;
    }
    print int(42);
    return 0;
}
```

`.test-for2.out`

```
0
1
2
3
4
42
```

test-for-as-while1.dem

```plaintext
def function main() int
{
    let i int;
    i = 5;
    for (i = 0) {
```
print_int(i);
i = i - 1;
}
print_int(42);
return 0;
}

```c
int(i);
i = i - 1;
}
print_int(42);
return 0;
}
```

**test-for-as-while1.out**

5
4
3
2
1
42

```c
int a int, b int) int
{
    return a + b;
}

function main() int
{
    let a int;
    a = add(39, 3);
    print_int(a);
    return 0;
}
```

**test-func1.dem**

```c
int add(a int, b int) int
{
    return a + b;
}

function main() int
{
    let a int;
    a = add(39, 3);
    print_int(a);
    return 0;
}
```

**test-func1.out**

42

```c
/* Bug noticed by Pin-Chin Huang */

function fun(x int, y int) int
{
    return 0;
}

function main() int
{
    let i int;
    i = 1;
    fun(i = 2, i = i+1);
    print_int(i);
    return 0;
}
```

**test-func2.dem**

```c
typeof a int, b int) int
{
    return a + b;
}

function main() int
{
    let a int;
    a = add(39, 3);
    print_int(a);
    return 0;
}
```

**test-func2.out**

42

```c
/* Bug noticed by Pin-Chin Huang */

function fun(x int, y int) int
{
    return 0;
}

function main() int
{
    let i int;
    i = 1;
    fun(i = 2, i = i+1);
    print_int(i);
    return 0;
}
```
test-func3.dem

```c
function print_intem(a int, b int, c int, d int) void
{
  print_int(a);
  print_int(b);
  print_int(c);
  print_int(d);
}

function main() int
{
  print_intem(42, 17, 192, 8);
  return 0;
}
```

test-func3.out

```text
42
17
192
8
```

test-func4.dem

```c
function add(a int, b int) int
{
  let c int;
  c = a + b;
  return c;
}

function main() int
{
  let d int;
  d = add(52, 10);
  print_int(d);
  return 0;
}
```

test-func4.out

```text
62
```

test-func5.dem

```c
function foo(a int) int
{
  return a;
}

function main() int
{
  return 0;
}
```
1 function gcd(a int, b int) int {
2     for (a != b) {
3         if (a > b) a = a - b;
4         else b = b - a;
5     } return a;
6 }
7 
8 function main() int {
9     print_int(gcd(14, 21));
10    print_int(gcd(8, 36));
11    print_int(gcd(99, 121));
12    return 0;
13 }

test-gcd2.out

1 7
2 4
3 11

1 function gcd(a int, b int) int {
2     for (a != b) {
3         if (a > b) a = a - b;
4         else b = b - a;
5     }
6     return a;
7 }
8 
9 function main() int {
10     print_int(gcd(2, 14));
11     print_int(gcd(2, 15));
12     print_int(gcd(99, 121));
13     return 0;
14 }

test-gcd.out

1 2
2 3
3 11

1 function print_int(a) void

129
function print_int(a) void
{
    print_int(a);
}

function print_intb() void
{
    print_int(b);
}

function incab() void
{
    a = a + 1;
    b = b + 1;
}

function main() int
{
    a = 42;
    b = 21;
    print_inta();
    print_intb();
    incab();
    print_inta();
    print_intb();
    return 0;
}

let i bool;

function main() int
{
    let i int; /* Should hide the global i */
    i = 42;
    print_int(i + i);
    return 0;
}

function main() int
{
    print_int(42);
}

130
```plaintext
4 print_int(71);
5 print_int(1);
6 return 0;
7 }

test-hello.out

1 42
2 71
3 1

test-helloworld-assign.dem

1 function main() int
2 {
3   let x string;
4   x = "hello world\n";
5   print(x);
6   return 0;
7 }

test-helloworld-assign.out

1 hello world

test-helloworld.dem

1 function main() int
2 {
3   print("hello world\n");
4   return 0;
5 }

test-helloworld.out

1 hello world

test-if1.dem

1 function main() int
2 {
3   if (true) print_int(42);
4   print_int(17);
5   return 0;
6 }

test-if1.out

1 42
2 17

test-if2.dem
```
```plaintext
function main() int
{
    if (true) print.int(42); else print.int(8);
    print.int(17);
    return 0;
}

test-if2.out

42
17

test-if3.dem

function main() int
{
    if (false) print.int(42);
    print.int(17);
    return 0;
}

test-if3.out

17

function main() int
{
    if (false) print.int(42); else print.int(8);
    print.int(17);
    return 0;
}

test-if4.out

8
17

struct LLNode
{
    let val int;
    let next *struct LLNode;
    let end bool;
}

function newLLNode(data int) *struct LLNode
{
    let a *struct LLNode;
    a = cast malloc(13) to *struct LLNode;
    (*a).val = data;
    (*a).end = false;
    return a;
}
```

132
function add_front(temp *struct LLNode, head *struct LLNode) *struct LLNode
{
    (*temp).next = head;
    head = temp;
    return head;
}

function add_tail(temp *struct LLNode, head *struct LLNode) *struct LLNode
{
    let temp *struct LLNode;
    let struct_holder_pointer *struct LLNode;
    let struct_holder struct LLNode;
    let templ.val int;
    templ.val = (*templ).val;
    struct_holder_pointer = (head);
    struct_holder = *struct_holder_pointer;
    temp = newLLNode(0);
    (*temp).end = true;
    for (!(struct_holder.end)) {
        struct_holder_pointer = struct_holder.next;
        struct_holder = *struct_holder_pointer;
    }
    (*struct_holder_pointer).val = templ.val;
    (*struct_holder_pointer).end = false;
    (*struct_holder_pointer).next = temp;
    return head;
}

function delete(delete_val int, head *struct LLNode) *struct LLNode
{
    let struct_holder_pointer *struct LLNode;
    let struct_holder struct LLNode;
    let struct_holder_pointer.prev *struct LLNode;

    struct_holder_pointer = (head);
    struct_holder = *struct_holder_pointer;
    for ((struct_holder.val) != delete_val) {
        struct_holder_pointer.prev = struct_holder_pointer;
        struct_holder_pointer = struct_holder.next;
        struct_holder = *struct_holder_pointer;
    }
    (*struct_holder_pointer.prev).next = struct_holder.next;
    return head;
}
function print_list(struct_holder_pointer *struct LLNode) void
{
    let struct_holder struct LLNode;
    struct_holder = *(struct_holder_pointer);

    for (!(struct_holder.end)){
        print_int(struct_holder.val);
        struct_holder = *(struct_holder.next);
    }
    return;
}

function main() int
{
    let temp *struct LLNode;
    let head *struct LLNode;
    let i int;

    //init the tail node;
    let init struct LLNode;

    init.end = true;
    head = &init;

    //add to front, 0 to 10
    for (i = 0 ; i < 6 ; i = i + 1) {
        temp = newLLNode(i);
        head = add_front(temp, head);
    }

    print_list(head);
    print("====\n");

    temp = newLLNode(42);
    head = add_tail(temp, head);

    print_list(head);
    print("====\n");

    head = delete(2, head);

    print_list(head);

    return 0;
}

test-linkedlist-final.out
struct LLNode
{
    let val int;
    let next *struct LLNode;
    let end bool;
}

function add(data int) *struct LLNode{
    let a *struct LLNode;
    a = cast malloc(4) to *struct LLNode;
    (*a).val = data;
    (*a).end = false;
    return a;
}

function print_list(struct_holder_pointer *struct LLNode) void
{
    let struct_holder struct LLNode;
    struct_holder = *(struct_holder_pointer);
    for (!{struct_holder.end}){
        print int(struct_holder.val);
        struct_holder = *(struct_holder.next);
    }
    return;
}

function main() int
{
    let templ *struct LLNode;
    let head *struct LLNode;
    let i int;
    let struct_holder struct LLNode;
    let print_num int;
    let test struct LLNode;
    test.end = true;
    head = &test;
for (i = 0 ; i < 5 ; i = i + 1) {
    temp1 = add(i);
    (*temp1).next = head;
    head = temp1;
}
print_list(head);
return 0;
}

test-linkedlist-malloc.out
1 4
2 3
3 2
4 1
5 0

test-linkedlist-proof.dem
1 struct LLNode
2 {
3     let data struct Rectangle;
4     let next *struct LLNode;
5 }
6
7 struct Rectangle
8 {
9     let width int;
10 }
11
12 function main() int
13 {
14     let head *struct LLNode;
15     let node1 struct LLNode;
16     let node2 struct LLNode;
17     let node3 struct LLNode;
18     let struct_holder struct LLNode;
19     let struct_pointer_holder *struct LLNode;
20     let print_num int;
21
22     //build up list
23     node1.data.width = 1;
24     head = &node1;
25     node2.data.width = 2;
26     node2.next = head;
27     head = &node2;
28     node3.data.width = 3;
29     node3.next = head;
30     head = &node3;
31
32     //print list head to tail
33     print_list(head);
34 }
struct
holder = *(head);
print_num = struct_holder.data.width;
print_int(print_num);

struct_pointer_holder = struct_holder.next;
struct_holder = *struct_pointer_holder;
print_num = struct_holder.data.width;
print_int(print_num);

struct_pointer_holder = struct_holder.next;
struct_holder = *struct_pointer_holder;
print_num = struct_holder.data.width;
print_int(print_num);

return 0;
}

test-linkedlist-proof.out

function foo(i bool) void
{
let i int; /* Should hide the formal i */
i = 42;
print_int(i + i);
}

function main() int
{
foo(true);
return 0;
}

test-local1.out

function main() int
{
print_int(38 % 3);
return 0;
}

test-mod.out
test-ops1.dem

1 function main() int
2 {
3    print_int(1 + 2);
4    print_int(1 - 2);
5    print_int(1 * 2);
6    print_int(100 / 2);
7    print_int(99);
8    printb(1 == 2);
9    printb(1 == 1);
10   print_int(99);
11   printb(1 != 2);
12   printb(1 != 1);
13   print_int(99);
14   printb(1 < 2);
15   printb(2 < 1);
16   print_int(99);
17   printb(1 <= 2);
18   printb(1 <= 1);
19   printb(2 <= 1);
20   print_int(99);
21   printb(1 > 2);
22   printb(2 > 1);
23   print_int(99);
24   printb(1 >= 2);
25   printb(1 >= 1);
26   printb(2 >= 1);
27   return 0;
28 }

test-ops1.out

1  3
2 -1
3  2
4  50
5  99
6  0
7  1
8  99
9  1
10  0
11  99
12  1
13  0
14  99
15  1
16  1
17  0
18  99
19  0
20  1
21  99
22  0
23  1
```plaintext
test-ops2.dem

function main() int
{
    printb(true);
    printb(false);
    printb(true && true);
    printb(true && false);
    printb(false && true);
    printb(false && false);
    printb(true || true);
    printb(true || false);
    printb(false || true);
    printb(false || false);
    printb(!false);
    printb(!true);
    print_int(-10);
    print_int(-42);
}

test-ops2.out

1 1
2 0
3 1
4 0
5 0
6 0
7 1
8 1
9 1
10 0
11 1
12 0
13 -10
14 42

test-pointer-bool.dem

function main() int
{
    let a bool;
    let b *bool;
    let c bool;
    a = true;
    b = &a;
    printb(a);
    c = *b;
    printb(c);
    a = !a;
}
c = *b;
printb(c);
return 0;
}

test-pointer-bool.out
1 1
2 1
3 0

test-pointer-int.dem
1 function main() int
2 { 
3  let a int;
4  let b *int;
5  let c int;
6
7  a = 1;
8  b = &a;
9
10  print_int(a);
11
12  c = *b;
13  print_int(c);
14
15  a = a + 1;
16
17  c = *b;
18  print_int(c);
19
20  return 0;
21 }

test-pointer-int.out
1 1
2 1
3 2

test-pointer-malloc.dem
1 function getpointer() *int{
2  let a *int;
3  a = cast malloc(4) to *int;
4  *a = 42;
5  return a;
6 }
7
8 function main() int
9 {
10  let a *int;
let b int;
a = getpointer();
b = *a;
print_int(b);
return 0;
}

test-pointer-malloc.out

42
test-pointer-struct-onelvl.dem

struct Rectangle
{
    let width int;
    let height int;
}

function main() int
{
    let a struct Rectangle;
    let b struct Rectangle;
    let c struct Rectangle;
    let d int;
    a.width = 10;
    b = &a;
    c = *b;
    d = c.width;
    print_int(d);
    return 0;
}

test-pointer-struct-onelvl.out

10
test-pointer-struct-twolvl.dem

struct Rectangle
{
    let width int;
    let height int;
    let color struct Color;
}

struct Color
{
    let red bool;
}

function main() int
```rust
struct Rectangle {
    width: int,
    height: int
}

struct Circle {
    radius: int,
    r: Rectangle
}

let a = Rectangle { width: 10, height: 20 }
let b = &a;
let c = *b;
d = c.radius;
print(d);
c.radius = 15;
b = &c;
a = *b;
d = a.radius;
print(d);
return 0;
```

test-pointer-struct-twolvl.out

```
1 1
2 0
```

test-sleep.dem

```rust
function main() int {
    print("hello...
    
");
sleep(1);
    print("goodbye...
    
");
    return 0;
}
```

test-sleep.out

```
1 hello...
2 goodbye...
```

test-struct1.dem

```rust
struct Rectangle {
    width: int,
    height: int
}

struct Circle {
    radius: int,
    r: Rectangle
}

let a = Rectangle { width: 10, height: 20 }
let b = &a;
let c = *b;
d = c.radius;
print(d);
c.radius = 15;
b = &c;
a = *b;
d = a.radius;
print(d);
return 0;
```
```c
function main() int
{
    let b bool;
    let x struct Circle;
    print("hello world\n");
    return 0;
}
```

**test-struct1.out**

```console
hello world
```

**test-struct.dem**

```c
struct Rectangle
{
    let width int;
    let height int;
}

struct Circle
{
    let radius int;
}

function main() int
{
    let b bool;
    let y int;
    let x struct Circle;
    x.radius=4;
    y = x.radius + 6;
    print_int(y);
    print("hello world\n");
    return 0;
}
```

**test-struct-float.dem**

```c
struct Rectangle
{
    let width float;
    let height float;
}

struct Circle
{
    let radius float;
    let rectangle struct Rectangle;
}

function main() int
{
    let r struct Rectangle;
    let c struct Circle;
```
17 r.width = 0.5;
18 c.rectangle.width = r.width;
19
20 print_float(c.rectangle.width);
21
22 return 0;
23 }

**test-struct-float.out**

1 0.500000

**test-struct.out**

1 10
2 hello world

**test-structs-nested1.dem**

1 struct Circle
2 {
3    let radius int;
4    let extra_struct struct TestStruct;
5 }
6
7 struct TestStruct
8 {
9    let number int;
10    let color struct Color;
11 }
12
13 struct Color
14 {
15    let color_number int;
16 }
17
18 function main() int
19 {
20    let a int;
21    let b int;
22
23    let circle struct Circle;
24
25    circle.extra_struct.number = 10;
26    circle.extra_struct.color.color_number = circle.extra_struct.number;
27
28    a = circle.extra_struct.color.color_number;
29    b = circle.extra_struct.number;
30
31    print_int(a);
32    print_int(b);
33 }

**test-structs-nested1.out**
```plaintext
test-structs-nested.dem

struct Circle
{
    let radius int;
    let extra struct TestStruct;
}

struct TestStruct
{
    let number int;
    let color struct Color;
}

struct Color
{
    let color_number int;
}

function main() int
{
    let a int;
    let b int;
    let c int;
    let d int;

    let circle struct Circle;
    let test struct struct TestStruct;
    let test color struct Color;

    test.color.color_number = 696969;
    test_struct.number = 100000;
    test_struct.color = test.color;

    circle.extra_struct = test_struct;
    circle.extra_struct.color.color_number = 69;
    circle.extra_struct.number = 10;

    a = test_struct.color.color_number;
    b = circle.extra_struct.color.color_number;
    c = test_struct.number;
    d = circle.extra_struct.number;

    print_int(a);
    print_int(b);
    print_int(c);
    print_int(d);
}

test-structs-nested.out
```
test-threading1.dem

```plaintext
1 function sayhello(noop *void) *void
2 {
3    let x *void;
4    print("hello!\n");
5    return x;
6 }
7
8
9 function main() int
10 {
11    thread("sayhello", ",", 5);
12    return 0;
13 }
```

test-threading1.out

```plaintext
1 hello!
2 hello!
3 hello!
4 hello!
5 hello!
```

test-threading2.dem

```plaintext
1 function sayhello(str string) *void
2 {
3    let x *void;
4    print(str);
5    return x;
6 }
7
8
9 function main() int
10 {
11    thread("sayhello", "hello!\n", 5);
12    return 0;
13 }
```

test-threading2.out

```plaintext
1 hello!
2 hello!
3 hello!
4 hello!
5 hello!
```

test-var1.dem
function main() int
{
    let a int;
    a = 42;
    print_int(a);
    return 0;
}

test-var1.out

1 42