Casper

A high-level general-purpose programming language

Final Report

Michael Makris

UNI: mm3443

COMS W4115 Programming Languages and Translators

Fall 2018

Contents

Tables

Figures

1. Introduction

Casper is a limited general-purpose imperative programming language developed for the educational purposes and project requirements of the Columbia University COMS W4115 Programming Languages and Translators course in the Fall of 2018[1]. It is an extension of the MicroC language provided by Professor Edwards[2] with the addition of some C language[3] features, style and syntax, and the exposure to some C standard library functions[4].

Specifically, Casper carries over from MicroC the syntax, the integer, float and Boolean data types, the variable and function declarations, the statement block {}, the if-else decision making, the while and for loops, the assignment =, the arithmetic operations +, -, *, /, the arithmetic comparisons ==, !=, <=, <, >, >=, the Boolean operations &&, ||, !, and, the comments //, /**/. From C, Casper adds the char data type, the do-while loop, the break and continue control-flow, and the % operator. It also exposes many functions from the Math, String, CType and Stdlib libraries.

Casper deviates from these two languages with the addition of the String data type and the operators _ and ? that allow for string manipulation; _ concatenates any type to a string and ? queries for a character in a string by position. Casper also allows for nested comments. There is an explicit boolean data type with true, false keywords, an additional loop structure do-until, a casting operator ~ for int to float exchange, expanded comparison operators for all data types, an exponent operator ^ and the assignment operators +=, -=, _=,. The C ++ and -- are replaced by +=1 and -=1 and instead are used as lookaheads without changing the value of the variable.

Despite these additions, Casper is still a very limited language and more interesting as an exercise in building a compiler than in its power as a programming language. There was simply not enough time to build in a reliable array construct as originally proposed.

[1] http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/index.html
[2] http://www.cs.columbia.edu/~sedwards/
[3] Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Second Edition, Prentice Hall, Inc., 1988
[4] Please review the microc.pdf and C documentation for the two languages, see Section 9. References, Resources and Bibliography

2. Language Tutorial

2.1 Environment Setup

To be able to use Casper, we first need to set up the proper environment. Described here is the minimum environment needed to compile the compiler and to compile a Casper program.

The simplest way to set up the environment is to follow the MicroC instructions in the README. This contains instructions for several operating systems. Download MicroC from:

http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/microc.tar.gz

Using Ubuntu 16.04 as an example, Casper requires OCaml 4.07.0 and LLVM 3.8.0 bindings:

```
$ sudo apt install ocaml llvm llvm-runtime m4 opam
$ opam init
$ opam install llvm.3.8
$ eval `opam config env`
```

Some useful commands:

`$ lsb_release -a`	Ubuntu version
`$ ocaml -version`	OCaml version
`$ llc --version`	LLVM version

Download casper.tar.gz from http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/ and expand:

```
$ tar -xvf casper.tar.gz
```

In the /Casper directory:

`$ make help` provides the possible options for make.

`$ make` will compile the compiler in ./_build and provide a link ./casper.native.

`$ ./casper.sh <program filename with no extension>` will compile and run the program <filename>. Do not include the file extension .goo with the filename.

`$ ./casperc.sh <program filename with no extension>` will compile the program <filename>.

2.2 Tutorial

Having set up the environment, let's review how to program in Casper:

- The file extension for Casper files is ".goo". However, it must not be included when compiling, e.g. `$ ./casper.sh ./helloworld`
- A program can have global variables at the beginning, followed by functions. It must have a main function which is the starting point of statement execution.
- Variables are declared as <type> varname;
- Functions are declared as <type> functionname(*opt. <type> arg1,* …){*vars statements*} and contain local variables at the beginning followed by statements. They end with a return statement except in the case of a void function.
- Statement blocks are surrounded by {}
- Statements end in ; unless they have a statement block such as a for loop.
- It's best to use {} to avoid confusion in loop or conditional statements.
- It's best to use () to disambiguate operations.
- It's best to initialize variables, especially strings. This must be a separate statement and not part of the variable declaration.
- Use // for comments to end of line or /* */ for comment blocks. There can be nested comment blocks.

Now let’s look at some simple programs.

Example program **helloworld.goo**

```
/* hello world */
int main() {
    printinl(42);
    printfnl(3.14);
    prints("Hello world!
");
    printcnl(`Q`);
    printbnl(true);
    return 0;
}
```

Output
```
42
3.1400000000
Hello world!
Q
true
```

This demonstrates a simple main function as well as several print statements for the different data types. Strings can be enclosed in ‘ or “ while characters are enclosed in `. Notice that prints() does not add a line feed character but “print” functions ending in “nl” do. However, Casper does not use escape characters but instead allows unescaped special characters in strings. So, the effect of closing the string on a new line is to capture that line feed character in the string and print it.
Use “printsnl” if you prefer otherwise.

Example program **temperaturetable.goo**

```
/* print Fahrenheit-Celcius table
   for fahr = 0, 20, ..., 300   */

void main() {
    int fahr; int celsius;
    int lower; int upper; int step;

    lower = 0;
    upper = 300;
    step = 20;

    fahr = lower;
    while(fahr <= upper){
        celsius = 5 * (fahr -32) / 9;
        printsnl("      " _ fahr _ "      " _ celsius);
        fahr += step;
    }
}
```

Output
```
        0       -17
        20      -6
        40      4
        60      15
        80      26
        100     37
        120     48
        140     60
        160     71
        180     82
        200     93
        220     104
        240     115
        260     126
        280     137
        300     148
```

In this program we have declaration of variables at the top of the main function, their initialization as a separate step, and then the use of a while loop with a predicate comparison in () and statement body in {}. We also have some arithmetic operations, the string concatenation operator _ that converts other types to string first, and a += assignment which adds the rvalue to the lvalue.

Example program **fib.goo**

```
/* print the first 50 numbers in the Fibonacci series */
float fib(float n) {
  if (n < 2.0) return 1.0;
  return fib(n-1.0) + fib(n-2.0);
}

void main() {
    int i;
    for (i = 0; i < 50; i+=1) {
        printfnl(fib(~i));
    }
}
```

Output
```
1.0000000000
1.0000000000
2.0000000000
3.0000000000
5.0000000000
8.0000000000
…
12586269025.0000000000
```

This small program presents some very interesting features of Casper. First, it has function `float fib(float n)` which is called from `main()` and returns the n^th^ Fibonacci number. Second, `fib` is a recursive function, calling itself to calculate the answer. Typically, the Fibonacci function operates on integers. However, integers are 32-bit in Casper and to allow for a larger set of the sequence, `fib` uses float which are equivalent to double in C. Also notice the ~ operator in `main()`which converts the integer i to a float.

3. Language Reference Manual

3.1 Introduction

Casper is a rather limited in scope general-purpose imperative language that resembles the C language, but with emphasis on the high level than the traditional C low level capabilities. For example, it includes a String data type and library functions to manipulate strings. In this respect, the language should be able to implement some of the usual algorithms for applications that are programmed in C, Java, and Python. See Section 1 for an overview.

3.2 Lexical Conventions

3.2.1 Comments

Comments are ignored by the language translator.

a) Line comments are applied from // to the end of the line.

b) Block comments are enclosed by /* some comment */ and can span multiple lines.

c) Block comments allow nested blocks like /* /* */ /* */ */.

3.2.2 Whitespace

Newline '\n', carriage return '\r', horizontal tab '\t', and space ' ' are considered whitespace and as with the comments they are ignored by the language translator.

3.2.3 Identifiers

An identifier is a sequence of letters, uppercase or lowercase, including the underscore _, and digits, with the first character always a letter. Identifiers are used as tokens to identify variables and functions.

3.2.4 Reserved words

The following strings are reserved for use as keywords:

```
int    float     str    chr  bool  void  true  false  null  if  else  for  while  do  until
break  continue  return
```

main is not a reserved word but it must be used in every program to identify the starting function.

3.2.5 Literals

3.2.5.1 Integer literals

Sequence of digits 0 … 9, optionally signed (prefixed with + or -).

3.2.5.2 Floating point literals

A floating-point literal consists of an integer part, optionally signed (prefixed with + or -), a decimal point, a fraction part, an e or an E, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part, or the fraction part (not both) may be missing; either the decimal point or the e/E and the exponent (not both) may be missing.

3.2.5.3 String literals

A sequence of characters surrounded by double quotes as in "abc" or single quotes as in 'abc'. This allows for one type of quote to be included in a string defined by the other type. Any ASCII character can be included in a string.

3.2.5.4 Character literals

A single character surrounded by ` as in `a`.

3.2.5.5 Boolean literals

true and false are reserved words used in Boolean expressions and assignments.

3.3 Operators

Operator	On Types	Return Type	Description
_	ltype String rtype Integer, Float, String, Boolean	String	binary string concatenation
?	ltype String rtype Integer	String	binary character at position
+	Integer, Float	Integer, Float	binary arithmetic addition
-	Integer, Float	Integer, Float	binary arithmetic subtraction
*	Integer, Float	Integer, Float	binary arithmetic multiplication
/	Integer, Float	Integer, Float	binary arithmetic float division
%	Integer, Float	Integer, Float	binary arithmetic modulus
^	Float	Float	binary arithmetic exponentiation
>	Integer, Float, String	Boolean	binary relational greater than
>=	Integer, Float, String	Boolean	binary relational greater than or equal
<	Integer, Float, String	Boolean	binary relational less than
<=	Integer, Float, String	Boolean	binary relational less than or equal
==	Integer, Float, String, Boolean, Char	Boolean	binary relational equal
!=	Integer, Float, String, Boolean, Char	Boolean	binary relational not equal
-	Integer, Float	Integer, Float	unary negation
++	Integer	Integer	unary return integer + 1, don't change rvalue
--	Integer	Integer	unary return integer - 1, don't change rvalue
=	Integer, Float, String, Boolean	Integer, Float, String, Boolean	assignment of right-hand expression to left-hand side
_=	ltype String rtype Integer, Float, String, Boolean	String	assignment of the concatenation of the two sides to the left-hand side
+=	Integer, Float	Integer, Float	assignment of the sum of the two sides to the left-hand side
-=	Integer, Float	Integer, Float	assignment of the difference of the two sides to the left-hand side
~	Integer, Float	Float, Integer	Converts from Integer to Float and vice-versa
&&	Boolean	Boolean	binary logical AND
||	Boolean	Boolean	binary logical OR
!	Boolean	Boolean	unary logical NOT

Table 3-1. Operators

Precedence and Associativity

Operator	Associativity
()	
- !	right to left
++ --	right to left
~	right to left
?	right to left
^	right to left
* / %	left to right
+ -	left to right
_	left to right
> >= < <=	left to right
== !=	left to right
&&	left to right
||	left to right
= _= += -=	right to left

Table 3-2. Precedence and Associativity

3.4 Expressions

casperExpression:

Epsilon
Integer Literal
Float Literal
String Literal
Character Literal
Boolean Literal
Void Literal
Identifier
casperExpression casperBinaryOperator casperExpression
casperUnaryOperator casperExpression
Identifier casperAssignment casperExpression
Identifier (casperExpression-list$_{opt}$)
(casperExpression)

casperExpression-list*:*

casperExpression
casperExpression-list , casperExpression

The Casper expressions include literal values of different types, identifers for variable and function names, function calls, expressions in () to disambiguate operation precedence, assignments of rvalues to lvalues and unary and binary operations using operators from Section 3.3.

3.5 Statements

casperStatement:

```
{ casperStatement-list }
casperExpression ;
if ( casperExpression ) casperStatement
if ( casperExpression ) casperStatement else casperStatement
for ( casperExpression_opt ; casperExpression ; casperExpression_opt ) casperStatement
while (casperExpression) casperStatement
do casperStatement until ( casperExpression) ;
do casperStatement while ( casperExpression ) ;
break ;
continue ;
return ;
return casperExpression_opt ;
;
```

casperStatement-list:

```
casperStatement
casperStatement casperStatement-list
```

Apart from expressions, the Casper statements are related to control-flow. There is the return from a function (void functions do not have one or have one with no expression), the conditional if-else, and the while loop with three additional variations, for, do-while and do-until. break and continue are used in loops to either exist the statement block or jump to the loop predicate.

Note: break and continue were not implemented exactly as shown in the grammar. Currently only one of them is allowed within a function. This is a known bug. They must be implemented as am environment that follows the entry and exist of while loops.

3.6 Data Types

casperType:

Type	**Keyword**	**Description**
Integer	int	32-bit integer
Floating Point	float	double
String	str	Pointer to immutable space
Character	chr	8-bit integer
Boolean	bool	1-bit integer
Void	void	Used to signify a function does not return a value

Table 3-3. Data Types

3.7 Variables

casperVariable:

```
casperType Identifier ;
```

There are global and local variables. They must be declared either before functions, or at the top of the function statement block respectively. Also, there is no initialization assignment with the declaration. It must be done separately.

Scope Variable scope is static and either global or local within a statement block {}.

3.8 Functions

casperFunction:

casperType Identifier (formals-list$_{opt}$) { locals-list$_{opt}$ casperStatement-list$_{opt}$ }

formals-list:

casperType Identifier
formals-list , casperType Identifier

locals-list:

casperVariable
locals-list casperVariable

The main() function is required in every Casper program.

3.9 Casper Program

declarations:

declarations casperVariable$_{opt}$
declarations casperFunction$_{opt}$

Casper:

declarations EOF

The Casper program file extension is .goo and should be in UTF-8 or ASCII text format. See Section 2 or 5 on how to compile.

3.10 Casper Standard Library

As of now there is no separate Casper Standard Library. Any functions must be copied to each Casper program.

3.11 Casper C Standard Library

This is made up of the `./stdlib.c` file which is compiled and linked by the shell scripts. See Section 5 for details. The library currently has some utility functions to implement string operations and it also exposes C standard libraries.

3.12 I/O

Currently there are multiple print functions to output all types of values: prints() for strings, printi() for integers, printf() for floats, printb() for Booleans, printbasi() for Booleans as 1 or 0, printc() for characters. Adding nl to the name, e.g. prinsnl(), adds a \n character at the end. Using _ converts any type to string. The printbig() function is also available from MicroC.

scanf() was not implemented as proposed.

3.13 TODO’s

Due to time limitations, there were features that either were proposed and not implemented or need further work. Upon retirement, I will:

- Implement arrays, as proposed.
- Implement I/O, including file I/O, as proposed.
- Add break and continue to a list to allow multiples within a loop and in several loops.
- Work on the Char type and expand operations and library functions.
- Create a Casper Standard Library that can be linked.
- Make Casper the greatest programming language in the world.

4. Project Plan

4.1 Process

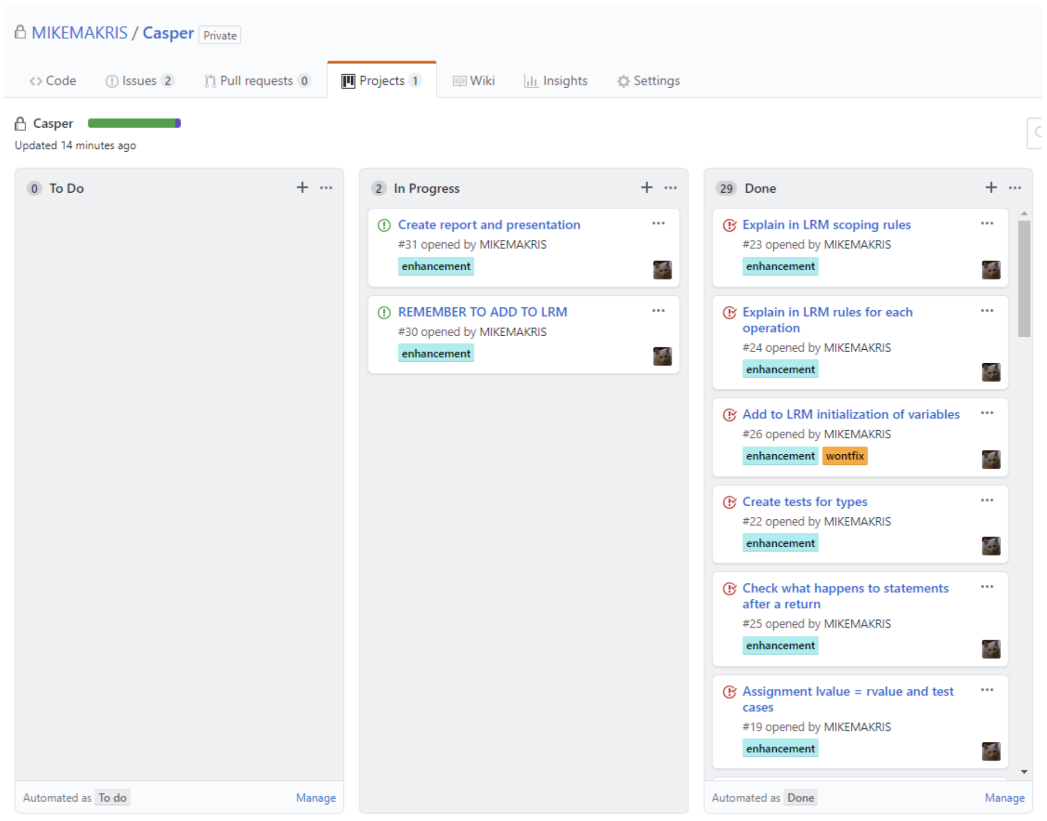
Casper was from the beginning proposed to be an extension of MicroC with additional features from the C language. The proposal in conjunction with the C language specification provided in effect the specification for Casper, with some minor variations as they became evident. In terms of delivery, the goal was not to strictly meet the original proposal or original language reference manual, but to understand how to implement the features and cover as much as possible in the limited time (time-boxed iterative process).

The proposal also provided the process of implementing the language, which was to implement the features as listed: the data types, name format, comment syntax, reserved words, operators, expressions, statements and control-flow. Even though these milestones sometimes overlapped and sometimes required revisions, the development process followed these units through the layers of the translator: AST, Scanner, Parser, Semantic checker and Code generator.

There were three types of testing. In the beginning, there was unit testing of the Scanner with Ocamllex and the Parser with Ocamlyacc and Menhir. Then there was integration testing of all the layers using the tools provided by MicroC, the pretty-printing and the top-level. Finally, for every new feature added to the language, there was a test suite built to maintain quality. Clang was also used to investigate and confirm implementations.

Finally, in terms of investigating a feature, there were regular meetings with Professor Edwards before class in addition to the lectures on OCaml and MicroC, and the resources listed in Section 9.

The project was managed using the GitHub Project tool, effectively a task board, Figure 4-1. As the project took on a new task, whether research, architectural, development, integration, or testing, a new issue was added to the project and task board. The issues were labeled as “enhancement”, “question”, “help wanted”, “bug” and “wontfix”. The issues were assigned to the developer and added to one of the milestones listed in Table 4-1.



Casper
Updated 14 minutes ago

0 To Do

2 In Progress

Create report and presentation
#31 opened by MIKEMAKRIS
enhancement

REMEMBER TO ADD TO LRM
#30 opened by MIKEMAKRIS
enhancement

29 Done

Explain in LRM scoping rules
#23 opened by MIKEMAKRIS
enhancement

Explain in LRM rules for each operation
#24 opened by MIKEMAKRIS
enhancement

Add to LRM initialization of variables
#26 opened by MIKEMAKRIS
enhancement wontfix

Create tests for types
#22 opened by MIKEMAKRIS
enhancement

Check what happens to statements after a return
#25 opened by MIKEMAKRIS
enhancement

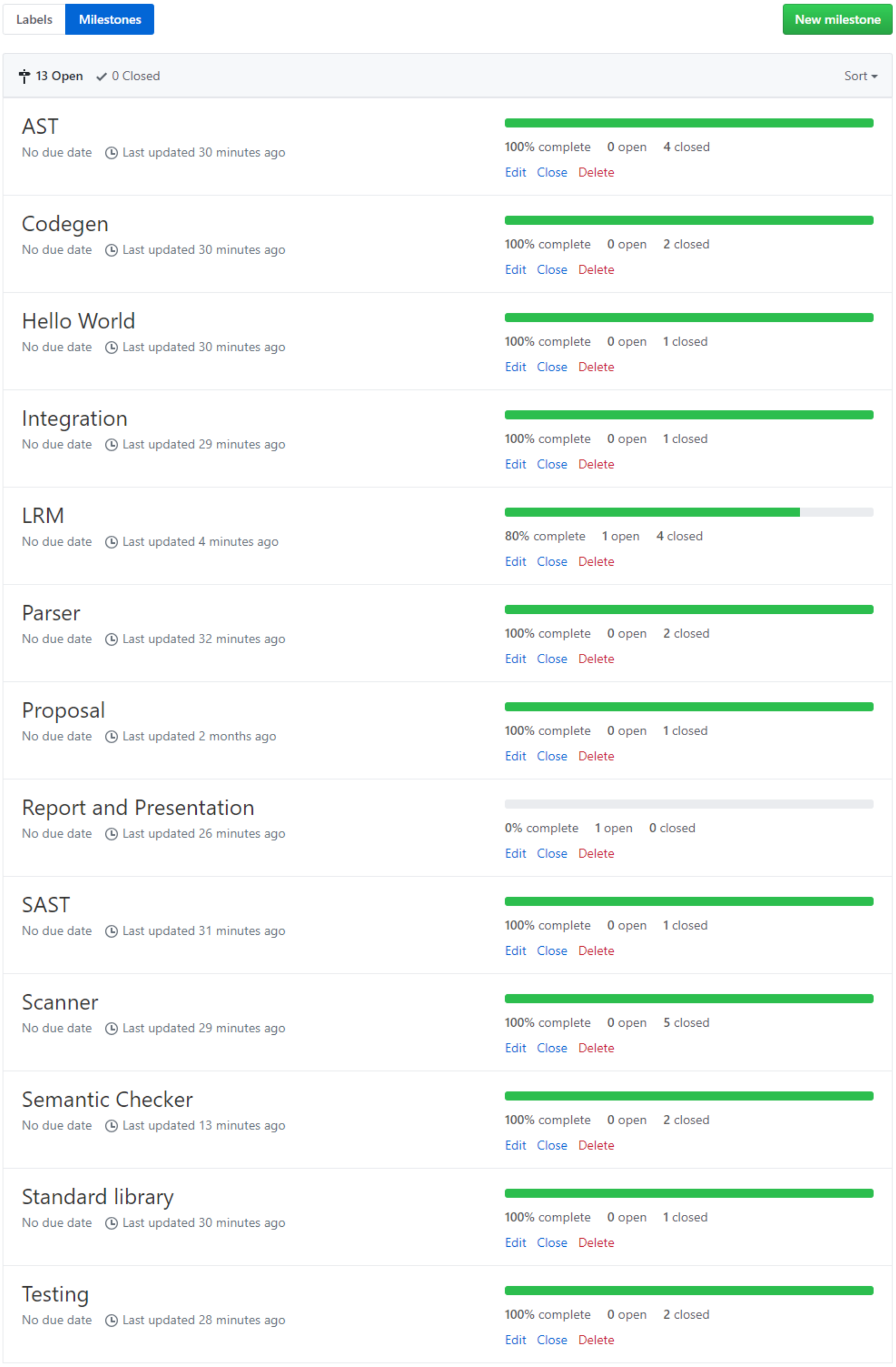
Assignment lvalue = rvalue and test cases
#19 opened by MIKEMAKRIS
enhancement

Automated as To do Manage

Automated as Done Manage

Figure 4-1. Project Task Board

Table 4-1. Project Milestones



Labels Milestones New milestone

13 Open ✓ 0 Closed Sort

AST	No due date	Last updated 30 minutes ago	100% complete	0 open 4 closed
Codegen	No due date	Last updated 30 minutes ago	100% complete	0 open 2 closed
Hello World	No due date	Last updated 30 minutes ago	100% complete	0 open 1 closed
Integration	No due date	Last updated 29 minutes ago	100% complete	0 open 1 closed
LRM	No due date	Last updated 4 minutes ago	80% complete	1 open 4 closed
Parser	No due date	Last updated 32 minutes ago	100% complete	0 open 2 closed
Proposal	No due date	Last updated 2 months ago	100% complete	0 open 1 closed
Report and Presentation	No due date	Last updated 26 minutes ago	0% complete	1 open 0 closed
SAST	No due date	Last updated 31 minutes ago	100% complete	0 open 1 closed
Scanner	No due date	Last updated 29 minutes ago	100% complete	0 open 5 closed
Semantic Checker	No due date	Last updated 13 minutes ago	100% complete	0 open 2 closed
Standard library	No due date	Last updated 30 minutes ago	100% complete	0 open 1 closed
Testing	No due date	Last updated 28 minutes ago	100% complete	0 open 2 closed

4.2 Programming Style Guide

- Indentation: Mostly four spaces, or two for OCaml “denser” code. No tabs used anywhere for compatibility, except as required in Makefile.
- Comments: At the top of each file to explain what it is about. Otherwise minimal comments at the beginning of each main block to annotate, provide a signature, or explain functionality.
- Naming conventions: Several different conventions used to distinguish between different items across different levels of the translation. For example,
    - OCaml names have lowercase_lowercase… in line with MicroC.
    - AST types start with “casper” in camelCase.
    - AST type values start with uppercase and continue in camelCase.
    - SAST types start with “sCasper” in camelCase.
    - SAST type values start with “S” and continue with AST type values.
    - Scanner tokens are in uppercase.
    - File names in lowercase except for “Makefile”. Test files start with “fail-“ or “test-“ as in MicroC.
- For text editor I use Notepad++ with UTF-8 encoding and Unix LF line ending.

4.3 Project Timeline

I originally had joined the Grape team but on 9/24/2018 talked to Professor Edwards and decided to work independently on a new project because of the complexities of being a CVN student. So, my project timeline starts on the fourth week. The period up to completing the front end and “Hello World” was pretty much on target. However, it became apparent towards the end that there was not enough time and especially not enough stamina to complete all of the proposed features.

Table 4-2. Project Timeline

Date	**Goals**
9/24/18	Start project independently
9/28/18	Submit Proposal
10/7/18	Set up environment, project
10/15/18	Submit Language Reference Manual, Scanner, Parser
11/14/18	Submit Hello World
11/25/18	Have MicroC test suite working with Casper
12/15/18	Finish development
12/19/18	Present and submit Final Report

4.4 Roles and Responsibilities

I originally was the manager for the Grape project. But since I decide to work independently, I had to assume all the roles and responsibilities of Manager, Language Guru, System Architect and Tester.

4.5 Software Development Environment

The software development environment consisted of:

Hardware	Lenovo® ThinkPad™ P71 Signature Edition Intel® Core™i7-7820HQ CPU @2.90GHz, 64 GB RAM
Operating System	Microsoft® Windows 10 Pro™ Windows Subsystem for Linux, Ubuntu® 16.04.3 LTS Xenial™
Languages and Tools	OCaml: The OCaml toplevel, version 4.07.0 OCamllex: The OCaml lexer generator, version 4.07.0 OCamlyacc: The OCaml parser generator, version 4.07.0 menhir, version 20180905 LLVM version 3.8.0 gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.10) clang version 3.8.0-2ubuntu4 dot - graphviz version 2.38.0
Version Control	GitHub-hosted Git repository at https://github.com/MIKEMAKRIS/Casper
Text Editor	Notepad++ v7.5.9
Mouse	Microsoft® IntelliMouse® Explorer 3.0
Type of caffeine	Coke Zero™ *and lots of carbs*

Table 4-3. Software Development Environment

4.6 Project Log

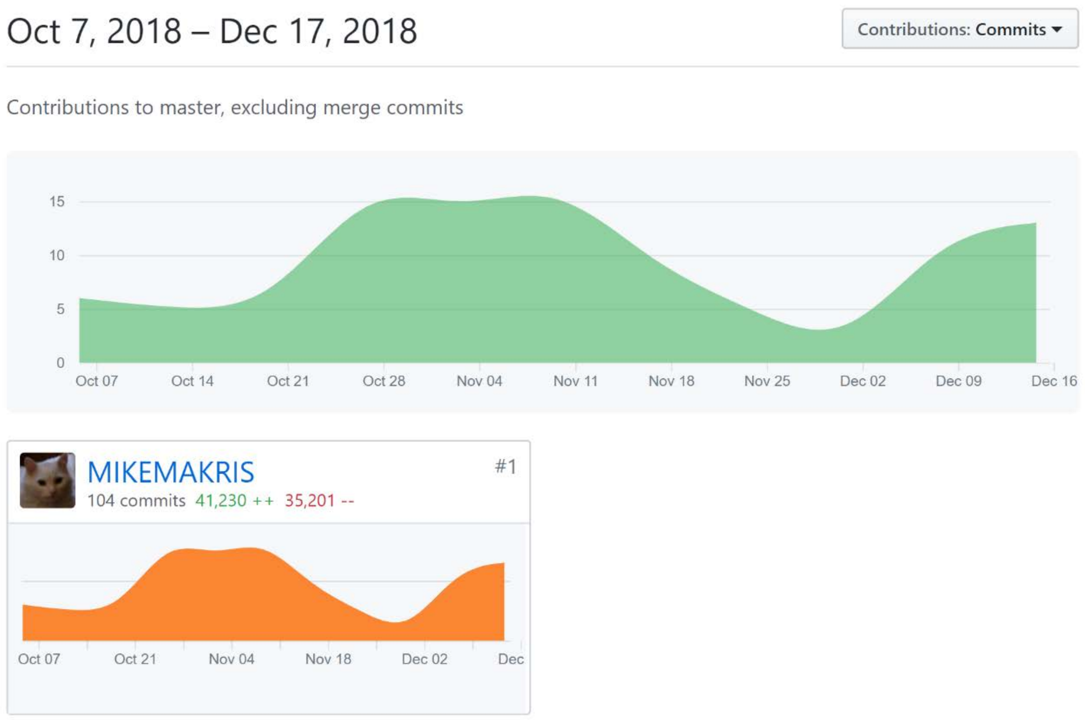


Figure 4-2. Commits to GitHub Repository

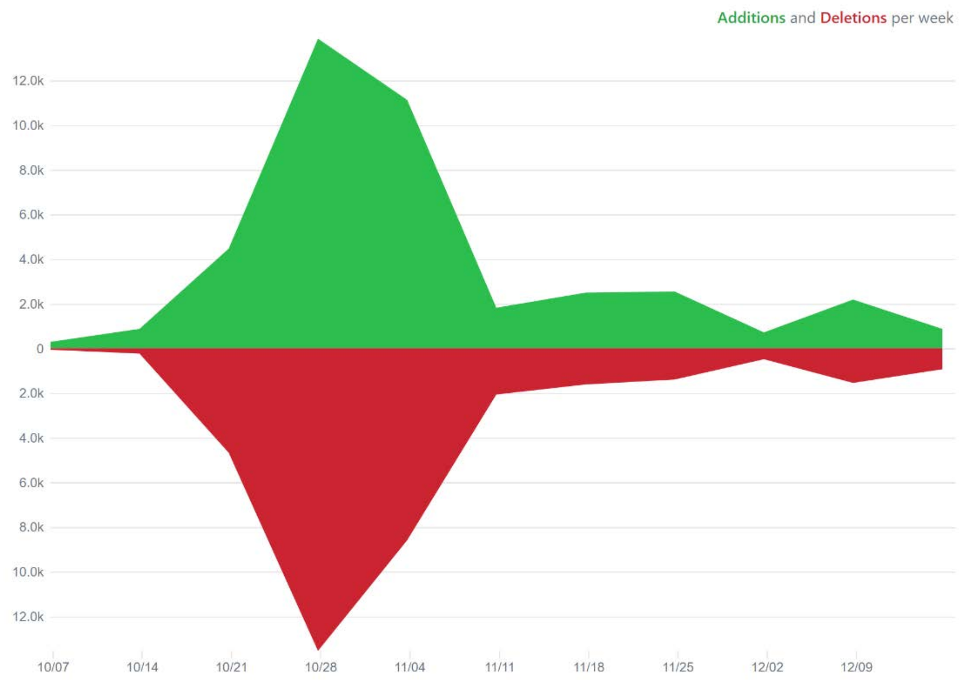


Figure 4-3. Code Frequency to GitHub Repository

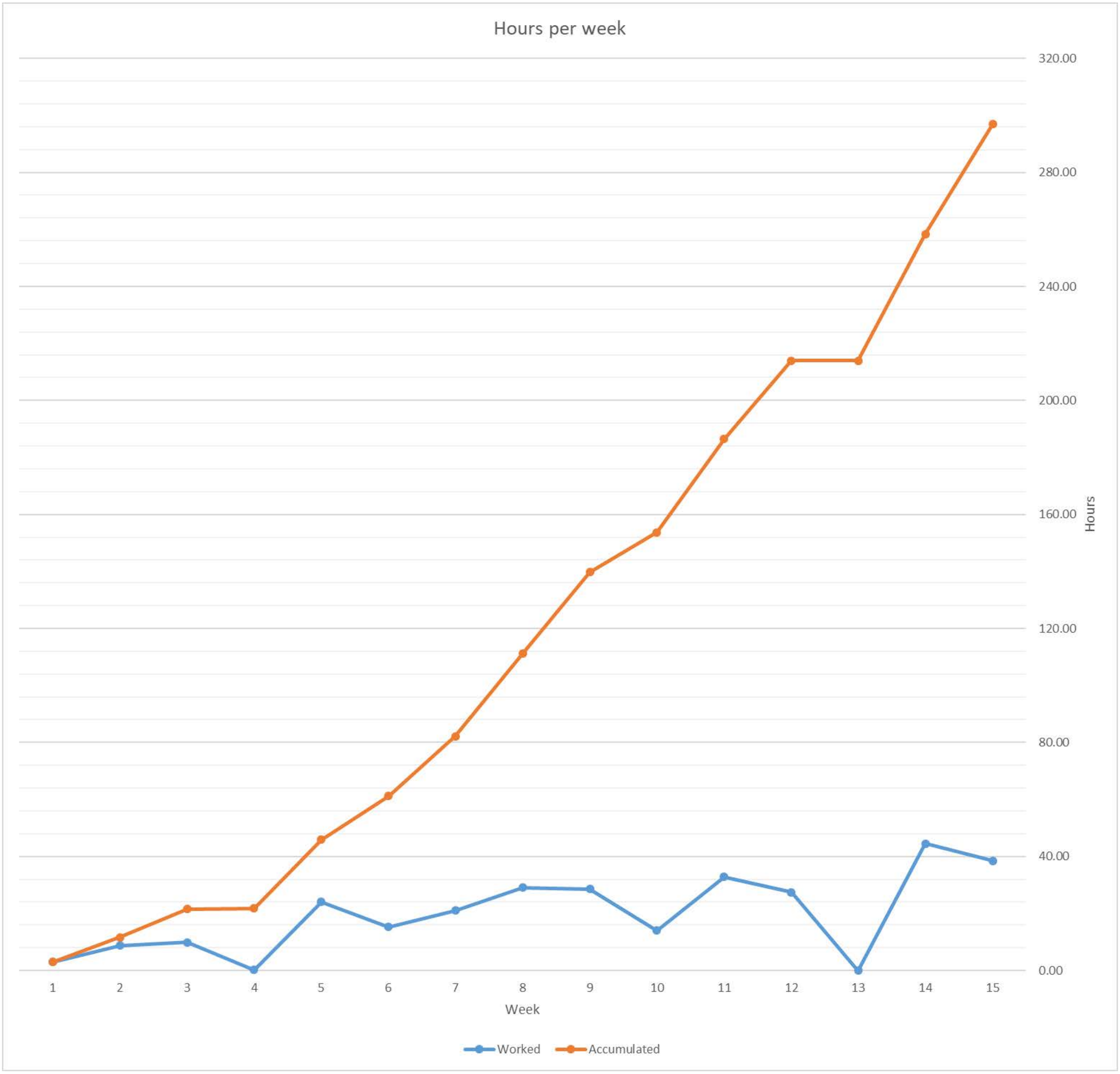


Figure 4-4. Hours worked per week and accumulated

Table 4-4. Project Log

	Date	In	Out	In	Out	In	Out	In	Out	Hours	Location	Description
Week 1	Mon 09/03/18									0:00		Labor Day Holiday
09-Sep	Tue 09/04/18									0:00		
	Wed 09/05/18	5:30 PM	6:00 PM							0:30	Campus	Formed team in class
YWeek 36	Thu 09/06/18									0:00	Home	PLT Team info
	Fri 09/07/18									0:00		
	Sat 09/08/18									0:00		
	Sun 09/09/18									0:00		
										0.5		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 2	Mon 09/10/18	5:30 PM	7:00 PM							1:30	Campus	Set up team/member roles. Started talking about language options, leaning towards a graph language.
16-Sep	Tue 09/11/18									0:00		
	Wed 09/12/18	5:30 PM	7:00 PM							1:30	Campus	Discussed language options, talked to Prof. Edwards about graph language.
YWeek 37	Thu 09/13/18									0:00		
	Fri 09/14/18									0:00		
	Sat 09/15/18									0:00		
	Sun 09/16/18									0:00		
										3		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 3	Mon 09/17/18	5:30 PM	11:00 PM							5:30	Campus	Worked on defining language.
23-Sep	Tue 09/18/18	6:00 PM	9:00 PM							3:00	Campus	Worked on defining language, writing proposal. Proposal was submitted by Timmy.
	Wed 09/19/18	5:30 PM	5:40 PM							0:10	Campus	Quick stand-up after class today.
YWeek 38	Thu 09/20/18									0:00		
	Fri 09/21/18									0:00		
	Sat 09/22/18									0:00		
	Sun 09/23/18									0:00		
										8.67		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 4	Mon 09/24/18	3:30 PM	3:45 PM							0:15	Campus	Talked to Prof. Edwards to do project on my own. It was ok'ed. Informed team.
30-Sep	Tue 09/25/18									0:00	Home	
	Wed 09/26/18									0:00	Home	
YWeek 39	Thu 09/27/18	5:00 PM	11:59 PM							6:59	Home	Worked on proposal, submitted.
	Fri 09/28/18	12:00 AM	2:41 AM							2:41	Home	Worked on proposal, submitted.
	Sat 09/29/18									0:00	Home	
	Sun 09/30/18									0:00		
										9.92		

Table 4-4. Project Log

	Date	In	Out	In	Out	In	Out	In	Out	Hours	Location	Description
Week 5	Mon 10/01/18	3:30 PM	3:45 PM							0:15	Home	Talked to Prof. Edwards about Casper. No garbage collection, too hard. Custom types. No dictionary. Easy to implement ' and " for strings. Have arrays and objects.
07-Oct	Tue 10/02/18									0:00	Home	
	Wed 10/03/18									0:00	Home	
YWeek 40	Thu 10/04/18									0:00	Home	
	Fri 10/05/18									0:00	Home	
	Sat 10/06/18									0:00	Home	
	Sun 10/07/18									0:00		
										0.25		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 6	Mon 10/08/18	3:30 PM	3:45 PM							0:15	Campus	Talked to Prof. Edwards about Casper. Focus on strings: first class objects. Mutable or immutable strings? Enable user to develop library for strings eg toLower(), CRLF <--> LF
14-Oct	Tue 10/09/18									0:00	Home	
	Wed 10/10/18	3:30 PM	3:45 PM							0:15	Campus	Talked to Prof. Edwards about Casper.
YWeek 41	Thu 10/11/18									0:00	Home	
	Fri 10/12/18	9:00 PM	11:50 PM							2:50	Home	Set up github. Set up project. Started working on parser.
	Sat 10/13/18	9:30 AM	11:45 AM	6:00 PM	11:59 PM					8:14	Home	Scanner. Split + to + for addition and _ for string concatenation. Added -=. Added reserved words null and continue.
	Sun 10/14/18	12:00 AM	1:00 AM	10:00 AM	3:00 PM	5:30 PM	11:59 PM			12:29	Home	Added exception to scanner, cleaned up types and added negatives. Working on parser.
										24.05		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 7	Mon 10/15/18	8:00 PM	11:45 PM							3:45	Home	Talked to Prof. Edwards. Main can be validated in later logic. Same with types so convert expressions to non-type. Wrote LRM and submitted. Parser not finished, continue after midterm.
21-Oct	Tue 10/16/18									0:00	Home	
	Wed 10/17/18									0:00	Home	
YWeek 42	Thu 10/18/18									0:00	Home	
	Fri 10/19/18									0:00	Home	
	Sat 10/20/18									0:00	Home	
	Sun 10/21/18	1:45 PM	6:00 PM	7:30 PM	11:59 PM	12:00 AM	2:48 AM			11:32	Home	
										15.29		

Table 4-4. Project Log

	Date	In	Out	In	Out	In	Out	In	Out	Hours	Location	Description
Week 8	Mon 10/22/18									0:00	Home	Talked to Prof. Edwards about variable declarations with assignment, array declarations, variable declaration as statement to allow mix with statements, assignment as expression/statement, i++, remove print/input keywords, add C LRM Grammar section equivalent to Casper.
28-Oct	Tue 10/23/18	12:30 AM	5:00 AM							4:30	Home	Set up LLVM on Windows Subsystem for Linux, added multilevel comments, cleanup based on discussion.
	Wed 10/24/18									0:00	Home	
YWeek 43	Thu 10/25/18									0:00	Home	
	Fri 10/26/18									0:00	Home	
	Sat 10/27/18	7:00 PM	11:59 PM							4:59	Home	scanner: created copy with no Parser and printf statements instead of tokens to validate
	Sun 10/28/18	12:00 AM	1:30 AM	10:00 AM	5:00 PM	9:00 PM	11:59 PM			11:29	Home	scanner: worked on line numbers, error messages, strings, removed print input keywords
										20.97		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 9	Mon 10/29/18	12:00 AM	3:00 AM							3:00	Home	Makefile, trying to compile everything together. Talked to Prof. Edwards about scanner, parser, ast. Need to change type to primitive to accommodate arrays as variables in ast. Change syntax of arrays in parser. Ok to remove print/input keywords. Merge assignments into Binop expression. Remove "of unit" for void, null, break and continue. Split boolean in scanner to two keywords. Consider bringing back the regular expression for strings in scanner instead of using the buffer. Will leave for later when I have better idea what is needed. Already tested the buffer and it handles escape characters. If it becomes a problem then switch. Thinking about removing Do loops.
04-Nov	Tue 10/30/18	10:00 PM	11:59 PM							1:59	Home	
	Wed 10/31/18	12:00 AM	3:30 AM							3:30	Home	
YWeek 44	Thu 11/01/18									0:00	Home	
	Fri 11/02/18									0:00	Home	
	Sat 11/03/18	10:00 AM	4:30 PM	7:00 PM	11:59 PM					11:29	Home	Applied ideas from discussion. Built up scanner, parser, ast and .ml.
	Sun 11/04/18	12:00 AM	2:50 AM	3:45 PM	5:30 PM	7:30 PM	11:59 PM			9:04	Home	Unable to compile everything. First removed the linecount. Then went back to microc and started building up from that.
										29.04		

	Date	In	Out	In	Out	In	Out	In	Out	Hours	Location	Description
Week 10	Mon 11/05/18	12:00 AM	2:00 AM	1:00 PM	3:30 PM	7:00 PM	11:50 PM			9:20	Home	Cleaning up parser using menhir. AST pretty print functions. Problems compiling everything to get ast print.
11-Nov	Tue 11/06/18	9:45 PM	11:59 PM	12:00 AM	2:30 AM					4:44	Home	Able to compile up to ast print.
	Wed 11/07/18									0:00	Home	
YWeek 45	Thu 11/08/18									0:00	Home	
	Fri 11/09/18									0:00	Home	
	Sat 11/10/18	10:30 AM	12:30 PM	4:30 PM	8:30 PM					6:00	Home	work on sast
	Sun 11/11/18	10:00 AM	2:00 PM	5:00 PM	6:00 PM	8:30 PM	11:59 PM			8:29	Home	Work on semant.ml. Had to remove arrays to move forward.
										28.55		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 11	Mon 11/12/18	12:00 AM	1:00 AM	10:00 PM	11:59 PM					2:59	Home	Added check_assigned to Return as per lecture comment.
18-Nov	Tue 11/13/18	10:00 AM	5:00 PM	8:00 PM	11:59 PM					10:59	Home	Minimal Codegen.ml, cleanup of makefile, casper.sh, hello world test. Submitted hello world. Created .dot files and control-flow graphs.
	Wed 11/14/18									0:00	Home	
YWeek 46	Thu 11/15/18									0:00	Home	
	Fri 11/16/18									0:00	Home	
	Sat 11/17/18									0:00	Home	
	Sun 11/18/18									0:00	Home	
										13.97		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 12	Mon 11/19/18									0:00	Home	Talked to Prof. Edwards about strings and arrays.
25-Nov	Tue 11/20/18									0:00	Home	
	Wed 11/21/18	6:00 PM	11:59 PM							5:59	Home	Set up clang and clangviz.sh to visualize llvm code. Try to understand break and continue.
YWeek 47	Thu 11/22/18	12:00 AM	1:20 AM	10:30 AM	1:30 PM	2:30 PM	4:30 PM	7:30 PM	11:59 PM	10:49	Home	Codegen statements and expressions
	Fri 11/23/18	12:00 AM	2:15 AM							2:15	Home	
	Sat 11/24/18	10:15 AM	6:00 PM	8:30 PM	11:59 PM					11:14	Home	Codegen statements and expressions, removed Exp - should be in standard library, test cases and debugging
	Sun 11/25/18	8:30 AM	11:00 AM							2:30	Home	All microc tests converted and debugged.
										32.79		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 13	Mon 11/26/18									0:00	Home	Talked to Prof. Edwards about standard library. Can implement in C and expand printbig.c
02-Dec	Tue 11/27/18	6:00 PM	11:59 PM							5:59	Home	Brought back exponent operator. Added math.h to stdlib.c and implemented exponent operator and math functions. Added float-to-int operation as ~ also.
	Wed 11/28/18	12:00 AM	1:30 AM	10:30 PM	11:59 PM					2:59	Home	Trying to get other built-in functions with no arguments, working on strings.
YWeek 48	Thu 11/29/18	12:00 AM	3:30 AM	11:00 AM	5:00 PM					9:30	Home	Built-in functions, strings
	Fri 11/30/18	9:30 AM	4:30 PM							7:00	Home	Built-in functions, strings
	Sat 12/01/18	9:00 PM	11:00 PM							2:00	Home	print bool functions
	Sun 12/02/18									0:00	Home	
										27.47		

Table 4-4. Project Log

	Date	In	Out	In	Out	In	Out	In	Out	Hours	Location	Description
Week 14	Mon 12/03/18									0:00	Home	
09-Dec	Tue 12/04/18									0:00	Home	
	Wed 12/05/18									0:00	Home	
YWeek 49	Thu 12/06/18									0:00	Home	
	Fri 12/07/18									0:00	Home	
	Sat 12/08/18									0:00	Home	
	Sun 12/09/18									0:00	Home	
										0		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 15	Mon 12/10/18									0:00	Home	
16-Dec	Tue 12/11/18									0:00	Home	
	Wed 12/12/18	12:00 PM	6:00 PM	10:30 PM	11:59 PM					7:29	Home	break, continue, Con, ConAsgn
YWeek 50	Thu 12/13/18	12:00 AM	1:00 AM	9:30 AM	1:00 PM	2:00 PM	5:00 PM	7:00 PM	11:00 PM	11:30	Home	break, continue, Con, ConAsgn, testing
	Fri 12/14/18	9:30 AM	5:15 PM	8:00 PM	11:00 PM					10:45	Home	Char type, ctype library, binary search algorithm, tried to add arrays
	Sat 12/15/18	12:00 PM	8:00 PM							8:00	Home	cleaned up, sieve, tried qsort - swap works
	Sun 12/16/18	9:00 AM	11:00 AM	6:00 PM	8:30 PM	9:45 PM	11:59 PM			6:44	Home	Report and cleanup
										44.47		
	Date	**In**	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**	**Hours**	**Location**	**Description**
Week 16	Mon 12/17/18	12:00 AM	2:00 AM	8:00 AM	1:30 PM	4:00 PM	11:59 PM			15:29	Home	Report and cleanup
23-Dec	Tue 12/18/18	12:00 AM	1:00 AM	8:00 AM	2:30 PM	5:00 PM	11:59 PM			14:29	Home	Report and cleanup
	Wed 12/19/18	12:00 AM	2:30 AM	9:00 AM	3:00 PM					8:30	Home	Report. Presentation, submit and present
YWeek 51	Thu 12/20/18									0:00	Home	
	Fri 12/21/18									0:00	Home	
	Sat 12/22/18									0:00	Home	
	Sun 12/23/18									0:00	Home	
										38.47		
							Total Hours:			**297.4**		

Table 4-5. Git log

```
commit
cc27318c3ecbbf299c102ad63aa378ab6328529e
Author: MM <mm3443@columbia.edu>
Date:   Mon Dec 17 18:52:32 2018 -0500
   Report

commit
c1278c89f0ff46096fdbd367351407dfa3d5d5e3
Author: MM <mm3443@columbia.edu>
Date:   Mon Dec 17 13:34:17 2018 -0500
   Report

commit
ce28aea61f49a62a0bd79236b432fbd0fb4e3475
Author: MM <mm3443@columbia.edu>
Date:   Mon Dec 17 12:13:34 2018 -0500
   Report

commit
5621ff842123eea0d85d2203a225eccb69682492
Author: MM <mm3443@columbia.edu>
Date:   Mon Dec 17 09:55:50 2018 -0500
   Report, cleanup, examples

commit
f2c1b0d25bad8e061f1c997dc26ea40a40f7160a
Author: MM <mm3443@columbia.edu>
Date:   Mon Dec 17 01:59:34 2018 -0500
   Report and cleanup

commit
74e1a5cfac0e791f7062b9eeb3263cab22ba92af
Author: MM <mm3443@columbia.edu>
Date:   Mon Dec 17 00:59:49 2018 -0500
   Report and cleanup

commit
8cecc9fa44736ca3bb5fa27d38ffc9f90e233d9f
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 16 20:33:44 2018 -0500
   Report

commit
6bf03ecba2176f69792ed9d4345b0a4e1a279ca9
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 16 11:57:05 2018 -0500
   Report

commit
91afae68fcdbd88bc683dd5680d148a0d62502a8
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 16 10:20:19 2018 -0500
   Cleanup

commit
58864f8f5be4ded5367f49b579c4924e2ab7481c
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 16 10:19:49 2018 -0500
   Cleanup

commit
71f1870ec10651a5730d01b0a452ba968b7edb17
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 16 10:01:06 2018 -0500
   Report

commit
084028d44bb1f9c88c72d89333ddbe4a18178103
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 16 09:41:48 2018 -0500
   cleanup

commit
53145352ee3348b490c8aca146ba8646f2b52197
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 16 09:34:56 2018 -0500
   cleanup

commit
c6ce76ffcf92f408ccc43f38f23dcd49ed21820c
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 16 08:56:09 2018 -0500
   qsort

commit
2441eb026b595b2f96b58e94083d9cf0799b6fca
Author: MM <mm3443@columbia.edu>
Date:   Sat Dec 15 20:52:17 2018 -0500
   x

commit
17b826ce35d507ade117b021f5b66c3e0b217942
Author: MM <mm3443@columbia.edu>
Date:   Sat Dec 15 18:03:33 2018 -0500
   sieve, cleanup
   sieve of eratosthenes

commit
bf78f9f8120f269ec6f8e79412a38bc92e1aa31f
Author: MM <mm3443@columbia.edu>
Date:   Sat Dec 15 12:56:11 2018 -0500
   bsearch

commit
ea7ebc92133f682e8d5d95565b2533219ac9e851
Author: MM <mm3443@columbia.edu>
Date:   Sat Dec 15 12:23:10 2018 -0500
   Cleanup

commit
0b5c5eda4bd90d1d62b02e14dba088788193b14c
Author: MM <mm3443@columbia.edu>
Date:   Sat Dec 15 09:59:30 2018 -0500
   x

commit
9ab3f8bb1a6387124504024b392438cbeedb70e5
Author: MM <mm3443@columbia.edu>
Date:   Sat Dec 15 09:51:09 2018 -0500
   Cleanup

commit
46300b3d03cb9e44a74eb43b7c82b48ff93ca1db
Author: MM <mm3443@columbia.edu>
Date:   Sat Dec 15 09:08:06 2018 -0500
   Char type, ctype library
   binary search algorithm on strings
   char type
   ctype library

commit
127664f8a4f5a4b7163802dade6b93c3a24db225
Author: MM <mm3443@columbia.edu>
Date:   Fri Dec 14 00:01:31 2018 -0500
   testing

commit
5c05105013325ee7c62af69434705fe91eb0dfa0
Author: MM <mm3443@columbia.edu>
Date:   Thu Dec 13 21:55:16 2018 -0500
   Inc / Dec cleanup

commit
d6e9bea10da85d64a938281ca9b9525cd8431c35
Author: MM <mm3443@columbia.edu>
Date:   Thu Dec 13 21:52:24 2018 -0500
   Switched ++ and -- to binops
   Switched ++ and -- to binops Add and Sub.
   These do not change value of variable (not
lvalue). Must use += amd -= for that,

commit
d3538494b577d8cb893c26a30c21de622c93bc52
Author: MM <mm3443@columbia.edu>
Date:   Thu Dec 13 16:49:34 2018 -0500
   Concat and ConAssign for all
   Append any type to string

commit
d77d07bc80ea758cf52023757c463f5c5f1ca520
Author: MM <mm3443@columbia.edu>
Date:   Thu Dec 13 01:08:38 2018 -0500
   Break, Continue
   Break, Continue. Do not work for nested loops

commit
e52a6a01d605129daf18456ddeab9883ec2d1848
Author: MM <mm3443@columbia.edu>
Date:   Sun Dec 2 11:56:13 2018 -0500
   bool print functions
   Converting printb to print true/false - Issues with
string comparison function

commit
564c92ef268a581953ba8935b4ff074a87dc2007
Author: MM <mm3443@columbia.edu>
Date:   Fri Nov 30 16:47:07 2018 -0500
   Strings and built-in functions
   _, ? operators
   comparisons for srtings
   = and _= for strings

commit
ac438faa04b44b5a356569c0f660fbe09adb6d9b
Author: MM <mm3443@columbia.edu>
Date:   Fri Nov 30 02:49:45 2018 -0500
   Built-in functions, strings
   Added Math.h built-in functions
   Added _ operator
   Need ? operator, comparisons for strings  and
string functions

commit
9258ec677d7283a7309ccdbc2c507beed3436535
Author: MM <mm3443@columbia.edu>
Date:   Wed Nov 28 01:29:01 2018 -0500
   Math operator, functions
   Brought back exponent operator.
   Added math.h to stdlib.c and implemented
exponent operator and math functions.
   Added float-to-int operation as ~ also.

commit
8a3c22f45533e0760d00be0e92957cbfb9aacb1a
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 25 11:08:39 2018 -0500
   All microc tests converted
   moved all microc tests in and debugged

commit
8ad83d26854fc4767862873e8ca134cf9bb568e3
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 25 00:10:35 2018 -0500
   Test cases and debugging
   Building test cases and debugging errors.
   Fixed List.rev  issues in parser statement block
and formals.
   Mod only for ints
   semant return validation switched to expression
to have better fail message
   Formatted testall.sh better

commit
9afb1507f97916e693aa97d0d45ca0b35545ddff
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 24 18:17:32 2018 -0500
   Codegen BinOp, Removed Exp
   Built BinOp code except Con, Cat
   Removed Exp, should be in standard library

commit
15ffeeee9173caea6c40a478463a9eea84fc96d3
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 24 15:31:26 2018 -0500
   do until and do while
   Added back do until and do while and
implemented in codegen

commit
b44110ad61126d4039e4576e2dae26c849309d32
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 24 11:16:48 2018 -0500
   instructions cleanup

commit
10809e7d288b5e157ab7ef6e259c9f0059b17283
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 24 11:04:17 2018 -0500
   instructions cleanup

commit
78e510a3aa11d0ec159c0b635ac5afa19fadaf2d
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 24 10:59:47 2018 -0500
   instructions cleanup

commit
1b85ee02cbcffea5ea909bc52dcd03525a28d4c5
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 24 10:55:48 2018 -0500
   instructions cleanup
```

```
commit
a6b04aee32e62a69361cdda31281ec9c7c18167e
Author: MM <mm3443@columbia.edu>
Date:   Fri Nov 23 02:20:01 2018 -0500
    Codegen and tests
    Worked on Unary and Assignment operators.
    Added integer to float operator.
    Added test cases for added operators.
    Missing ConAssign from assignments.

commit
f46d4880889cf69be10bf42ab36a79cc72e62144
Author: MM <mm3443@columbia.edu>
Date:   Thu Nov 22 16:22:02 2018 -0500
    Codegen expressions
    AddAssign, SubAssign, Not, Neg, Inc
    separated tests to add as I build code

commit
f4880e5e0f275efc727a229a548a16503285f096
Author: MM <mm3443@columbia.edu>
Date:   Thu Nov 22 11:29:14 2018 -0500
    Clang and llvm graphs
    visualize llvm code
    use clang to compare casper and c programs

commit
409f2afb73f766e9e104b3f004b5390aa85d44fe
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 13 23:30:17 2018 -0500
    At Hello World, added .dot files and control-flow
graphs

commit
787c90a149ed623e0f96f488d7eb53710bafac38
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 13 16:54:41 2018 -0500
    Hello world

commit
f78348a6a7710960d211766c046ddc066c2894be
Merge: 9015b97 5cfdb05
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 13 16:47:11 2018 -0500
    Merge branch 'master' of
https://github.com/MIKEMAKRIS/Casper

commit
9015b97aa74a3ab245027d9e180ae96497b936a2
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 13 16:47:06 2018 -0500
    readme

commit
5cfdb05b8e914686dac395ffcd4a59985e0c2e63
Author: MIKEMAKRIS <mm3443@columbia.edu>
Date:   Tue Nov 13 16:43:01 2018 -0500
    Update README.md

commit
2eb938108236e6b22e8ae8c74a935abe780788ea
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 13 16:34:16 2018 -0500
    Hello World first submission
    compiles with warnings
    cleaned up makefile, testall.sh casper.sh

commit
533fc6f1d987694c6b671553c0e9198e829363f0
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 13 13:12:03 2018 -0500
    Codegen.ml getting hellowrold first shot
    stripped down codegen.ml from microc compiles
with warnings for missing functionality but test-
helloworld.goo passes

commit
9242ab5de82b5af967712ae8c9b40321591aed69
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 13 10:23:07 2018 -0500
    At codegen.ml

commit
66ce61b2f39799a400237d452f4ae4656469dfac
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 13 02:06:50 2018 -0500
    Semant.ml added check_assign for return

commit
687503eac3c3905bcaaf99d118800c398359321c
Author: MM <mm3443@columbia.edu>
Date:   Mon Nov 12 00:56:27 2018 -0500
    Up to semant without arrays
    Took out array and reduced casperVariable to
bind. Took out array literal. It can print sast.

commit
824b97a8ecd1b9b5215366e20b3ac4dab0e39eb7
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 11 20:59:05 2018 -0500
    *** BEFORE REMOVING ARRAYS ***
    Save everything before simplifying casperVariable
to type * string. MUST REACH HELLO WORLD!

commit
6db677de7ef042fe175cbb2d2844d3b98a38eff4
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 11 20:55:41 2018 -0500
    BEFORE REMOVING ARRAYS

commit
2c04180d9cda17c78bec728ba5e44913024364e2
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 11 17:09:30 2018 -0500
    SEMANT to check_binds locals

commit
467965535f5b5ef34347e20ee504abf34431d259
Author: MIKEMAKRIS <mm3443@columbia.edu>
Date:   Sun Nov 11 16:57:21 2018 -0500
    Update README.md

commit
3fef49feadaf3322a06191aa37319d5335199fca
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 11 14:15:45 2018 -0500
    Before simplifying casperVariable
    renamed parser.mly

commit
202da486d823db71d6416ef39c5e3f6b4a452f8d
Author: MIKEMAKRIS <mm3443@columbia.edu>
Date:   Sun Nov 11 13:05:51 2018 -0500
    Update README.md

commit
20155d9f4db00a3d63baddfac56ff7d2a90737a5
Author: MIKEMAKRIS <mm3443@columbia.edu>
Date:   Sun Nov 11 13:05:17 2018 -0500
    Update README.md

commit
d8f7a007d398ef333a8bdaa23123c36d89e21536
Author: MIKEMAKRIS <mm3443@columbia.edu>
Date:   Sun Nov 11 13:02:40 2018 -0500
    Update README.md

commit
1ac56ebf9af1d2ce9e8122da4e34420ced7ec87b
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 10 18:30:38 2018 -0500
    SAST etc.
    test files to .goo, fix in Makefile, testfile.sh  and
casper.ml
    sast,ml cleaned up
    Array commented out in ast casperType. Should
be able to recognize simple variables from [-1].
    casper.sh for compiling and running

commit
31091877d0783054a5265d125c4953b7469d1a55
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 10 10:32:19 2018 -0500
    At AST pretty-print
    Scanner change to throw exception on ][ to
exclude multi-dimensional arrays
    compile.sh placeholder for testing llvm later

commit
a70eb1667cf716e257afe684b7775a4b805314aa
Author: MM <mm3443@columbia.edu>
Date:   Wed Nov 7 02:39:09 2018 -0500
    AST pretty print works, fixed formals

commit
96c359f3e1d6b97d9da93a585604a542595b93cb
Author: MM <mm3443@columbia.edu>
Date:   Wed Nov 7 01:52:58 2018 -0500
    AST print
    Able to compile up to ./casper.native -a ans see
ast printout. Issue with formals confused with
variable declaration.

commit
d9fc37f8b5196af6633a9169a091f730081f221b
Author: MM <mm3443@columbia.edu>
Date:   Tue Nov 6 00:10:26 2018 -0500
    CONVERSION: unable to complie

commit
1bc574a6944abe50effb473967b9b791d23a9036
Author: MM <mm3443@columbia.edu>
Date:   Mon Nov 5 22:14:32 2018 -0500
    CONVERSION: AST compiles
    Removed casperPrimitive. Need to come back to
figure out one-deimensional arrays

commit
71e253c3442c313d4ea2e6744320f9dfdd9fe320
Author: MM <mm3443@columbia.edu>
Date:   Mon Nov 5 20:25:19 2018 -0500
    CONVERSION: Makefile
    Added scanner and menhir tests

commit
4932817506aa458d008d0b585bd446c1da7a7e5c
Author: MM <mm3443@columbia.edu>
Date:   Mon Nov 5 15:23:56 2018 -0500
    CONVERSIONS: Parser, AST
    tested some code with menhir
    creatiing AST pretty-print functions

commit
a796afd475e944284ef580466194ee7800811680
Author: MM <mm3443@columbia.edu>
Date:   Mon Nov 5 01:44:35 2018 -0500
    CONVERSION: debugging parser with menhir

commit
37bfdc4f7abe723ae2887572ee69e70ab19dc601
Author: MM <mm3443@columbia.edu>
Date:   Mon Nov 5 00:31:09 2018 -0500
    CONVERSION Parser cleanup
    Makefile looks good, scanner looks good, parser
missing expressions. Using menhir to debug - see
file menhirTestInput.txt and menhir --interpret --
interpret-show-cst casperparse.mly <
./Test/menhirTestInput.txt

commit
3e9380e722d5d61e32d5997b142946e56d97c9bc
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 4 21:45:30 2018 -0500
    CONVERSION: Replaced with microc
    Changed microc names to casper, compiles good.

commit
f9b7c70a3d9fe6bf0318cb4fb4bb57b1fac9efe7
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 4 19:41:35 2018 -0500
    IMPORTANT Version before going to microc files
    Unable to compile, switch to microc and modify.

commit
5cb9c6ec145cf3a1b2f6154dd447b5191f817ecf
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 4 15:49:44 2018 -0500
    Scanner before removing line numbers

commit
3f9a58bb5f685792e9d18e5ea783db5483620bd0
Author: MM <mm3443@columbia.edu>
Date:   Sun Nov 4 01:47:01 2018 -0500
    Scanner Parser AST
    Compiling and debugging. Might have to remove
line numbers.

commit
0663517d8f9bcaa66d125bba92bdf8d06aab93bb
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 3 23:32:10 2018 -0400
    AST fix
```

```
commit
cd819d455e91b52054f85983149b5e7301816a83
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 3 22:58:08 2018 -0400
    LRM update
    no loops

commit
0818bb20558b9434f6ddbf848e4ba13a6e64c70c
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 3 22:18:43 2018 -0400
    Parser for John's review

commit
2f33cd61dd24907d69249d6f8be60943c3e0b622
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 3 21:42:57 2018 -0400
    Parser and AST
    Fixing arrays and other items from discussion of
20181029

commit
434af7447ebebd923fd3c7bc090b955045f450d2
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 3 13:47:25 2018 -0400
    Scanner/Parser after discussion 10/29
    Removed DO UNTIL because of decision to not
support Do Until and Do While loops.
    Removed PRINT INPUT keywords
    Modified array construct

commit
df45f1de5bfb5192299e330d1ccc28dc2c3097c9
Author: MM <mm3443@columbia.edu>
Date:   Sat Nov 3 13:08:14 2018 -0400
    Scanner, cleanup

commit
e5b3d69b702b07afe9d61d21e35f25a5fd987bf6
Author: MM <mm3443@columbia.edu>
Date:   Wed Oct 31 03:21:48 2018 -0400
    cleanup
    trying to validate string in scanner

commit
78b68293e917b32488d7cbb85b4aa878aab82224
Author: MM <mm3443@columbia.edu>
Date:   Tue Oct 30 22:33:13 2018 -0400
    Before changes of 20181029
    Good set before changes after talking to Prof.

commit
36455ea5e1449eb0090c0dcf4efe1f4b52904a05
Author: MM <mm3443@columbia.edu>
Date:   Mon Oct 29 03:05:52 2018 -0400
    Putting things together

commit
0ab75efc65483426071b2b6a87215c7a929fdd9f
Author: MM <mm3443@columbia.edu>
Date:   Mon Oct 29 01:56:45 2018 -0400
    Parser cleanup

commit
771591af7a7e18fa6c10bea8dfcfd144c0e20f35
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 28 23:35:14 2018 -0400
    Reviewed Scanner and AST

    Cleaned up AST. Added Con_Assignment.

commit
c3adfd6599b84a460d4beb738df412baca998fe9
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 28 17:04:02 2018 -0400
    cleaup

commit
6fc9d1f7334eddeb09ad6cd9081764bbfc767501
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 28 16:58:46 2018 -0400
    cleanup

commit
e2c82c4d61fe0ab66ae8565b5cb73bf3fae95c59
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 28 16:56:23 2018 -0400
    Scanner
    Validation with test version. Handle strings with
special characters. Improve error messages.

commit
5f754745db283ab6cd987ead0782fc068e51ce38
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 28 13:47:03 2018 -0400
    scanner celanup and validation

commit
ac64d89930917f77448c3136126d4ebe623e1711
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 28 10:55:17 2018 -0400
    Scanner version with line count ideas
    Two methods,
    use built-in lexbuf.lex_curr_p
    or build it with rule token lines words chars

commit
66145558e4fca51316b7c2cbdb95d918ce983bd9
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 28 01:27:36 2018 -0400
    Scanner
    validation and track position for error messages

commit
93d05a4f7e70db6532a304dc4c1f02d650e8b6b0
Author: MM <mm3443@columbia.edu>
Date:   Sat Oct 27 23:44:29 2018 -0400
    Working on parser,ast

commit
38f2794b33f4225547267de0fe0f2a7bb0db497d
Author: MM <mm3443@columbia.edu>
Date:   Tue Oct 23 05:04:34 2018 -0400
    Creating parser
    cleanup

commit
59383ee87206f53ee54ad5cbbb295da88c42324f
Author: MM <mm3443@columbia.edu>
Date:   Tue Oct 23 05:00:06 2018 -0400
    Creating parser
    see log comments

commit
767b521455b13541ee7e06e33d07dc6855605aba
Author: MM <mm3443@columbia.edu>
Date:   Mon Oct 22 02:52:20 2018 -0400
    Creating parser
    debugging parser.mly

commit
1d36aab3e4737a196d12a43530666278e6675626
Author: MM <mm3443@columbia.edu>
Date:   Mon Oct 22 01:46:22 2018 -0400
    Creating parser
    Worked on ast.mli, fixes scanner.mll and fixing
parser.mly based on microc.pdf.

commit
8b60a6a46519d57cc07ea28020f2c75158dc84e5
Author: MM <mm3443@columbia.edu>
Date:   Mon Oct 15 23:45:49 2018 -0400
    LRM submission
    LRM, scanner.mll, parser.mly

commit
fcd1b1b474418b49966b511d10f340e7aeeaa5dd
Author: MM <mm3443@columbia.edu>
Date:   Mon Oct 15 01:35:31 2018 -0400
    Creating parser

commit
1b2f132cd0e861ed8e1c9721f7e608ad3b33780a
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 14 21:25:06 2018 -0400
    Creating Parser
    working on parser

commit
4de9648f87c86bf265132aa8e15f027ad4859eac
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 14 15:01:08 2018 -0400
    Creating Parser
    Added exception to scanner, cleaned up types
and added negatives.

commit
01e5b24114d6039ce33c71825ea88671ff8d2a84
Author: MM <mm3443@columbia.edu>
Date:   Sun Oct 14 01:00:40 2018 -0400
    Creating Parser
    Split + to + for addition and _ for string
concatenation. Added -=. Added  reserved words
null and continue. Added one nested comment /* /*
*/ */ in scanner. Working on parser.

commit
35f0f0c772e02b915da1b6b4cde514da9a79e9ea
Author: MM <mm3443@columbia.edu>
Date:   Sat Oct 13 22:48:51 2018 -0400
    Creating Parser
    Scanner based on proposal

commit
b601f62a241b9f297818c072a81a68f8fb6ca23c
Author: MM <mm3443@columbia.edu>
Date:   Sat Oct 13 11:37:57 2018 -0400
    Added project log
    Created log based on my notes as manager of
previous team and after switching to Casper.

commit
6a188c95e647452cab1a80d1dff1b349204508bc
Author: MM <mm3443@columbia.edu>
Date:   Sat Oct 13 01:20:35 2018 -0400
    Creating Parser
    Adding types and operators to AST from Proposal
    Renamed calc to casper
    Modified Makefile

commit
ad9ebe0adac1b1634004991e26392670cc54f0dc
Author: MM <mm3443@columbia.edu>
Date:   Sat Oct 13 00:41:45 2018 -0400
    Creating Parser
    Starting from HW1 Q3 calculator example to
create the parsser

commit
142fd6388523ef2f9273aeae12bdcf0671160241
Author: MIKEMAKRIS <mm3443@columbia.edu>
Date:   Sat Oct 13 00:40:07 2018 -0400
    Casper language proposal
    20180928

commit
fee1781152afa41df7ac4e2f6b89d4df2a808933
Author: MIKEMAKRIS <mm3443@columbia.edu>
Date:   Sat Oct 13 00:15:23 2018 -0400
    Initial commit
```

5. Architectural Design

Please see the Appendix Section 8 for the code listing of each module and script in this section.

5.1 The Casper Translator

The Casper Translator, casper.native, comprises of a Scanner, a Parser, a Semantic Checker and an LLVM IR Code Generator. Figure 5-1 shows the components in terms of the code breakdown.

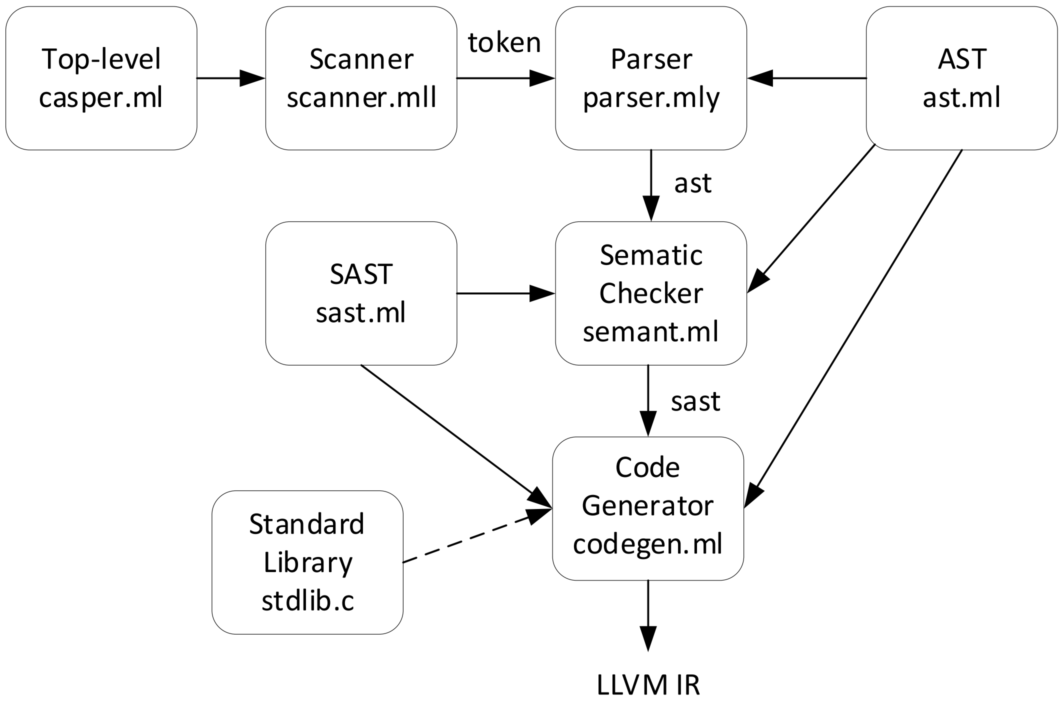


Figure 5-1. Architecture of the Casper Translator, casper.native

5.1.1 The Top-level

The top-level module orchestrates the process of creating the LLVM intermediate code for a Casper source file. It first reads from the source file the sequence of characters and feeds them to the Scanner "token" rule. This composes tokens which are passed to the Parser "casper" program data structure. This returns the Abstract Syntax Tree which is sent to the Semantic Checker. The result of that is to return a semantically-checked AST which is fed into the Code Generator to produce the LLVM intermediate code.

The module was effectively copied from MicroC and is written in OCaml.

5.1.2 The Scanner

The Scanner's job is to process a sequence of characters, eliminate comments and whitespace and build tokens that represent elements of the source code. Some interesting features in the Scanner are:

1. It allows for nested line comments.
2. It uses a buffer to read strings. This allows for any character to be added to a string literal which was an intentional feature of the language to avoid the use of escaped characters. The buffer in OCaml is self-adjusting in size.
3. It allows for strings in single or double quotes. Characters are in "`".
4. It performs syntax checking for some sequences of characters that are not valid.

The Scanner is written for processing with OCamlLex.

There was an effort to build line numbers into the token buffer so as to produce more meaningful error messages. That version of the Scanner exists in ./X/CasperScannerValidationWithLinenumbers.mll (Section 8.9.5). However, I was unable to carry it over to the Parser.

5.1.3 The Parser

The Parser uses the AST data structure and the tokens from the Scanner to build the Abstract Syntax Tree for the source code. This has no parenthesis, braces, commas or semicolons. The precedence and associative rules in the Parser resolve any conflicts.

The Parser applies patterns to the sequence of tokens to recognize what calls to make to the Semantic Checker and what values to pass. It is written for processing with OCamlYacc.

5.1.4 The AST

The AST module defines the structures for the Casper language as OCaml types. These are:

- The Casper Program
- The Casper Data Types:
    - Int
    - Float
    - String
    - Char
    - Bool
    - Void
- The Casper Global Variable
- The Casper Function
- The Casper Local Variable
- The Casper Statements:
    - Statement Block
    - Expression
    - If Condition
    - For Loop
    - While Loop
    - Do Until Loop
    - Do While Loop
    - Break
    - Continue
    - Return
- The Casper Expressions:
    - Epsilon
    - Integer Literal
    - Float Literal
    - String Literal
    - Character Literal
    - Boolean Literal
    - Void Literal
    - Null Literal
    - Identifier
    - Binary Operator
    - Unary Operator
    - Assignment Operator
    - Function Call

- The Casper Assignment Operations
    - Assign, =
    - Concatenate and Assign, _=
    - Add and Assign, +=
    - Subtract and Assign, -=
- The Casper Binary Operators:
    - Arithmetic +, -, *, / , %, ^
    - String Concatenate _, CharAt ?
    - Comparison >, >=, <, <=, ==, !=
    - Boolean And &&, Or ||
- The Casper Unary Operators:
    - Boolean Not !
    - Negation –
    - Integer<->Float ~

Using these structure definitions, the Parser and the Semantic Checker can perform their functions.

A second function of the AST module is to provide functions that pretty-print these structures. These functions are used both as a development tool, see Section 5.1.10 on how to use, and also in the code to translate the structures into strings.

This module was expanded from the MicroC version to implement the additional language features.

5.1.5 The SAST

The SAST module enhances the AST structures by appending their type to them. In doing so, it enables the Semantic Checker to verify the types of all elements of a Casper program.

A second function of the SAST module is to provide functions that pretty-print these structures. These functions are used both as a development tool, see Section 5.1.10 on how to use, and also in the code to translate the structures into strings.

This module was expanded from the MicroC version to implement the additional language features.

5.1.6 The Semantic Checker

The Semantic Checker performs sematic analysis on the abstract syntax tree created by the Parser.

1. It verifies that the elements of the source code have and accept the correct data types. This is done by building a semantically checked abstract syntax tree from the Parser AST, using the SAST module data structure definitions.
2. It builds the Symbol Table for the program.
    a. The Checker adds the names of standard library functions and the source code names to the Symbol Table and verifies there are no duplicate names for functions and global variables, or for local variables in a specific block.
    b. It checks for their declarations.
    c. It checks there is a “main” function in the program.
    d. It checks there are no “void” variables or function arguments.
    e. It checks that statement blocks are well-formed.

This module was expanded from the MicroC version to verify the additional language features. One modification from the original was the check for the Return statement which Professor Edwards mentioned in class that it should be a check on lvalue type = rvalue type also instead of just return type = function type.

5.1.7 The Code Generator

The Code Generator assumes it has a valid Casper program in the form of an AST. It traverses the tree and translates each node into LLVM code to build an LLVM module. In the Casper case, the SAST is a tuple of globals and functions, including external functions from the C library. For the functions, they are first defined as prototypes and then their bodies are constructed by the locals, formals (to stack) and statements. The statements are further constructed from expressions, with a return at the end if it is missing.

The Code Generator defines many external C functions. See Sections 5.1.8 and 5.2.2.

The output of the Code Generator can be processed by the LLVM Static Compiler into a three-address code with static-single assignment.

This module was expanded from the MicroC version to generate code for the additional language features.

5.1.8 The Casper C Standard Library

The Casper C Standard Library exposes C functions to the Code Generator, see Section 5.2.2. Here, it is important to say that the Code Generator declares the function signatures for several C functions either from the C standard libraries or from `./stdlib.c` itself. The function declarations are validated with the Semantic Checker and are available in the Code Generator to be translated into LLVM IR when used in the source code.

5.1.9 Compiling Casper

Please see Section 2 for setting up the language environment.

In the /Casper directory, `./Makefile` provides the following options:

`$ make`	- Compile the compiler casper.native (in ./_build and provide a link ./casper.native) using the OCaml Builder same as: make casper.native
`$ make all`	- Compile the compiler (as in `make`) and libraries (using gcc)
`$ make test`	- Make all and then run regression suite, see Section 6
`$ make testS`	- Load a version of the scanner that prints out the tokens instead of feeding to parser and then scans a program, see Section 6
`$ make testP`	- Use menhir to evaluate some token strings with the parser, see Section 6 Note: This requires the installation of menhir.
`$ make clean`	- Remove all generated files
`$ make cleang`	- Leave compiler, remove other generated files (to remove .ll, .s, .dot, .out, .png files)
`$ make casper.tar.gz`	- Create tarball of Casper

5.1.10 casper.native options

The Casper Translator, besides creating the LLVM IR (see Section 5.2.1), it provides some additional functionality that was useful during development.

`$ ./casper.native -help` provides:

```
Casper version 1.0.
usage: ./casper.native [-a|-s|-l|-c] [file.goo]
  -a Print the AST
  -s Print the SAST
  -l Print the generated LLVM IR
  -c Check and print the generated LLVM IR (default)
  -help  Display this list of options
  --help  Display this list of options
```

The -a option uses the pretty-print functions in `./as.ml` to print out the source code using the AST after the code was parsed and the AST was created. Similarly the -s option uses the SAST to print out the source – it includes the types with the symbols. Finally, -l prints the LLVM IR translation.

5.1.11 Clang

Another tool that was used during development was Clang. (To install: `$ sudo apt install clang`). This utility prints out the LLVM-IR from a C program in a similar way as casper.native. It can then be used with `opt` to generate control-flow graphs in `dot` format for each code block and to create images of them with `dot`. See the shell script `./clang/clangviz.sh` for its use.

5.2 The Casper Program

Please refer to Section 3 for the Casper language and the Casper Standard Library. Here we discuss additional elements of a Casper program and how it is compiled.

Based on the implementation of MicroC, Casper translates a source code into LLVM IR. To compile the source code into a Casper executable we have the following additional steps, as shown in Figure 5-2.

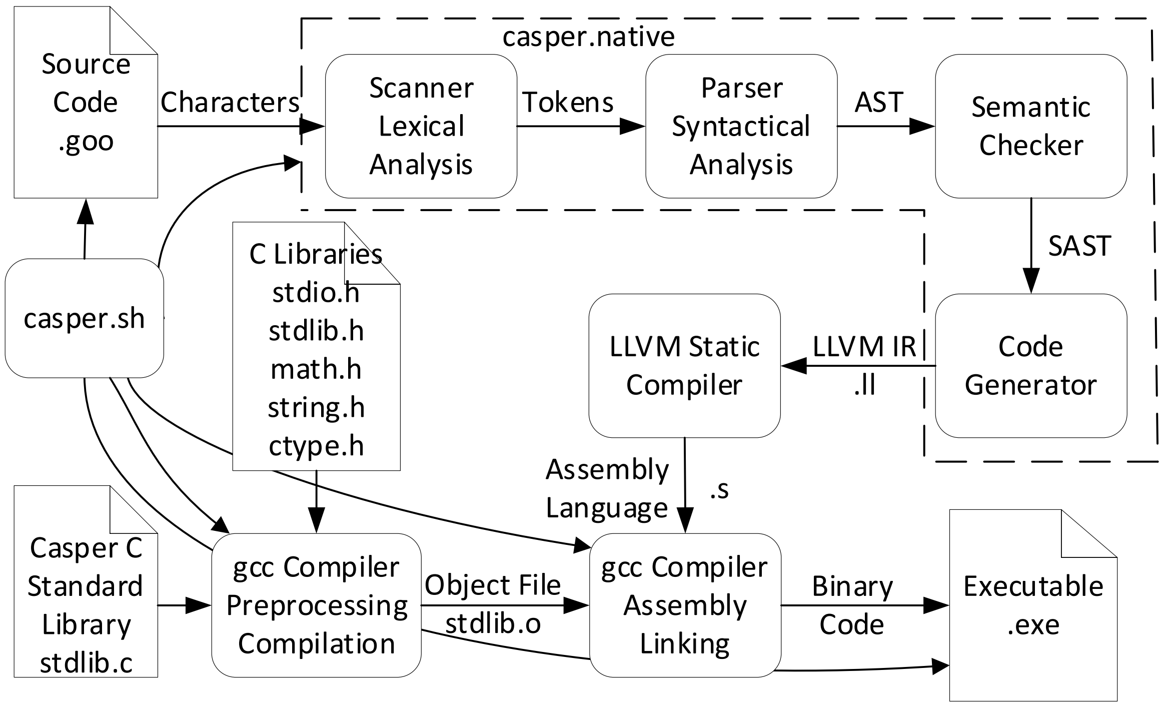


Figure 5-2. Compiling a Casper program

5.2.1 Compiling a Casper Program

Please refer to Section 2.1 and 5.1 for setting up the language environment. There are additional components required for `./casperll.sh` below.

There are three shell scripts available to compile a Casper program:

```
$ ./casper.sh <program filename with no extension>
```

This will (Figure 5-2):

a) Compile the program `<filename>.goo` using `./casper.native` to create the LLVM IR `<filename>.ll`.
b) Compile the `<filename>.ll` using the LLVM Static Compiler into the assembly language `<filename>.s`.
c) Compile the Casper C Standard Library `./stdlib.c` using the gcc compiler into `./stdlib.o`.

d) Assemble and Link `<filename>.s` and `./stdlib.o` using the gcc compiler into the final executable `<filename>.exe`.
e) Remove the intermediate files `<filename>.ll` and `<filename>.s`.
f) Run the executable.

```
$ ./casperc.sh <program filename with no extension>
```
Same as `./casper.sh` but without running the executable.

```
$ ./casperll.sh <program filename with no extension>
```
This will compile and run the executable in the same way as `./casper.sh` but will also:

a) Leave the intermediate `<filename>.ll` LLVM IR and assembly `<filename>.s` files for analysis.
b) Use `opt` to generate control-flow graphs in `dot` format for each code block.
c) Use `dot` to generate an image of the main() function.

This was used extensively during the code generation stage to debug and research how to implement the translation into LLVM.

Note: For `./casperll.sh` to work, it requires opt and dot which can be installed with:
```
$ sudo apt-get install opt
$ sudo apt-get install graphviz
```

It is best to compile a Casper program in the `/Casper` directory otherwise the shell scripts need to be modified.

5.2.2 The Casper C Standard Library

The Casper C Standard Library, `./stdlib.c`, has two purposes.

1. To load and expose the C standard libraries stdio.h, stdlib.h, ctype.h, string.h, math.h, to the Code Generator.
2. To provide additional functions to the Code Generator that were easier to implement is C. These include:
   a. `void printbig(int)`
      Originally from MicroC, it prints integers in an interesting format.
   b. `char *concats(char *, char *), char *concati(char *, int),`
      `char *concatf(char *, double), char *concatb(char *, int)`
      Enable concatenation of any type to string. For the "_" operator and "_=" assignment.
   c. `char *concati(char *, int)`
      For the Character At operator "?".
   d. `int scomp(char *, char *, int)`
      For String comparisons.

6. Test Plan

6.1 Source Code and Target Language Programs

6.1.1 Fibonacci, ./examples/fib.goo

Source Language Program (Casper)

```
/* print the first 50 numbers in the Fibonacci series */
float fib(float n) {
  if (n < 2.0) return 1.0;
  return fib(n-1.0) + fib(n-2.0);
}

void main() {
    int i;
    for (i = 0; i < 50; i+=1) {
        printfnl(fib(~i));
    }
}
```

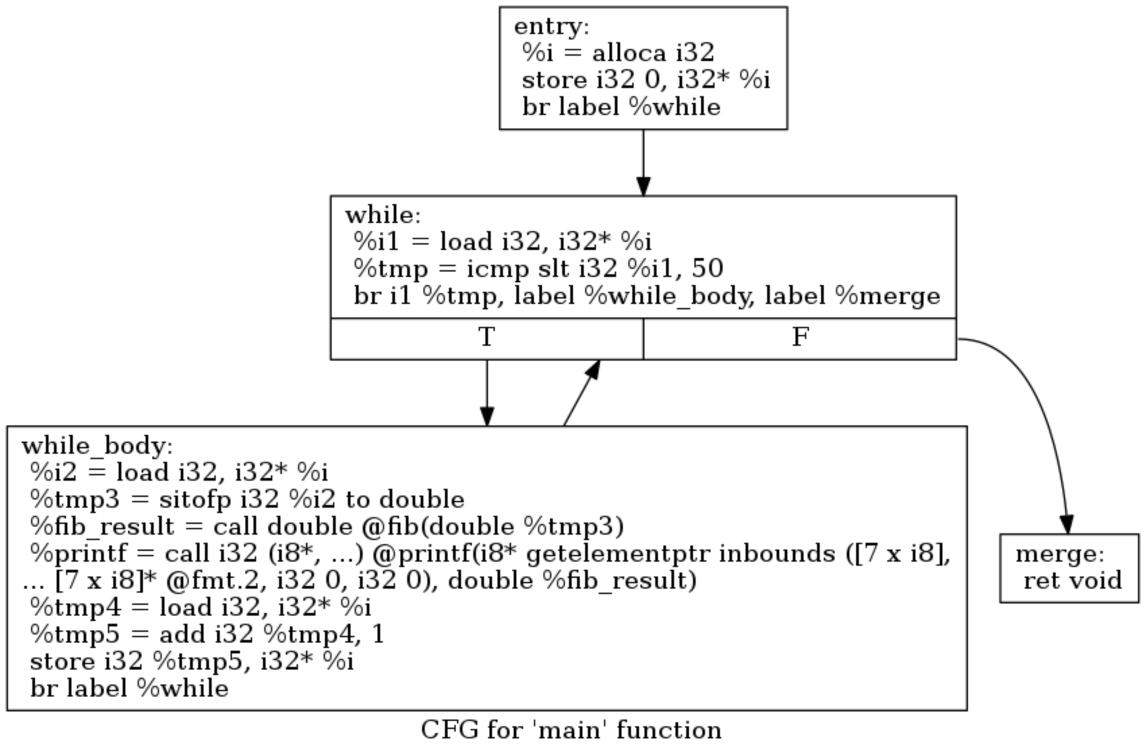
Output

```
1.0000000000
1.0000000000
2.0000000000
3.0000000000
5.0000000000
8.0000000000
…
12586269025.0000000000
```

Target Language Program (LLVM)

```
; ModuleID = 'Casper'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.2 = private unnamed_addr constant [7 x i8] c"%.10f\0A\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%g\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.5 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.7 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.10 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.11 = private unnamed_addr constant [7 x i8] c"%.10f\0A\00"
@fmt.12 = private unnamed_addr constant [3 x i8] c"%g\00"
@fmt.13 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.14 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.15 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.16 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.17 = private unnamed_addr constant [4 x i8] c"%d\0A\00"

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

declare i32 @printb(i1, i32)
declare i32 @printbig(i32)

declare i8* @concats(i8*, i8*)

declare i8* @concati(i8*, i32)

declare i8* @concatf(i8*, double)

declare i8* @concatb(i8*, i1)

declare i8* @charat(i8*, i32)

declare i1 @scomp(i8*, i8*, i32)

declare i32 @isalnum(i8)

declare i32 @isalpha(i8)

declare i32 @iscntrl(i8)

declare i32 @isdigit(i8)

declare i32 @isgraph(i8)

declare i32 @islower(i8)

declare i32 @isprint(i8)

declare i32 @ispunct(i8)

declare i32 @isspace(i8)

declare i32 @isupper(i8)

declare i32 @isxdigit(i8)

declare i32 @strlen(i8*)

declare i8* @strcpy(i8*, i8*)

declare i8* @strncpy(i8*, i8*, i32)

declare i8* @strcat(i8*, i8*)

declare i32 @strcmp(i8*, i8*)

declare i8* @strchr(i8*, i8*)

declare i8* @strrchr(i8*, i8*)

declare i32 @strspn(i8*, i8*)

declare i32 @strcspn(i8*, i8*)

declare i8* @strstr(i8*, i8*)

declare i8* @strerror(i32)

declare i8* @strtok(i8*, i8*)

declare double @pow(double, double)

declare double @sin(double)
declare double @cos(double)

declare double @tan(double)

declare double @asin(double)

declare double @acos(double)

declare double @atan(double)

declare double @sinh(double)

declare double @cosh(double)

declare double @tanh(double)

declare double @exp(double)

declare double @log(double)

declare double @log10(double)

declare double @sqrt(double)

declare double @floor(double)

declare double @ceil(double)

declare double @fabs(double)

declare i32 @srand(i32, ...)

declare i32 @rand()

declare double @fmod(double, double)

define void @main() {
entry:
  %i = alloca i32
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %while_body, %entry
  %i1 = load i32, i32* %i
  %tmp = icmp slt i32 %i1, 50
  br i1 %tmp, label %while_body, label %merge

while_body:                                       ; preds = %while
  %i2 = load i32, i32* %i
  %tmp3 = sitofp i32 %i2 to double
  %fib_result = call double @fib(double %tmp3)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([7 x i8], [7 x i8]* @fmt.2, i32
0, i32 0), double %fib_result)
  %tmp4 = load i32, i32* %i
  %tmp5 = add i32 %tmp4, 1
  store i32 %tmp5, i32* %i
  br label %while

merge:                                            ; preds = %while
  ret void
}

define double @fib(double %n) {
```

```
entry:
  %n1 = alloca double
  store double %n, double* %n1
  %n2 = load double, double* %n1
  %flst = fcmp olt double %n2, 2.000000e+00
  br i1 %flst, label %then, label %else

merge:                                            ; preds = %else
  %n3 = load double, double* %n1
  %fsub = fsub double %n3, 1.000000e+00
  %fib_result = call double @fib(double %fsub)
  %n4 = load double, double* %n1
  %fsub5 = fsub double %n4, 2.000000e+00
  %fib_result6 = call double @fib(double %fsub5)
  %fadd = fadd double %fib_result, %fib_result6
  ret double %fadd

then:                                             ; preds = %entry
  ret double 1.000000e+00

else:                                             ; preds = %entry
  br label %merge
}
```



```
entry:
 %i = alloca i32
 store i32 0, i32* %i
 br label %while
```

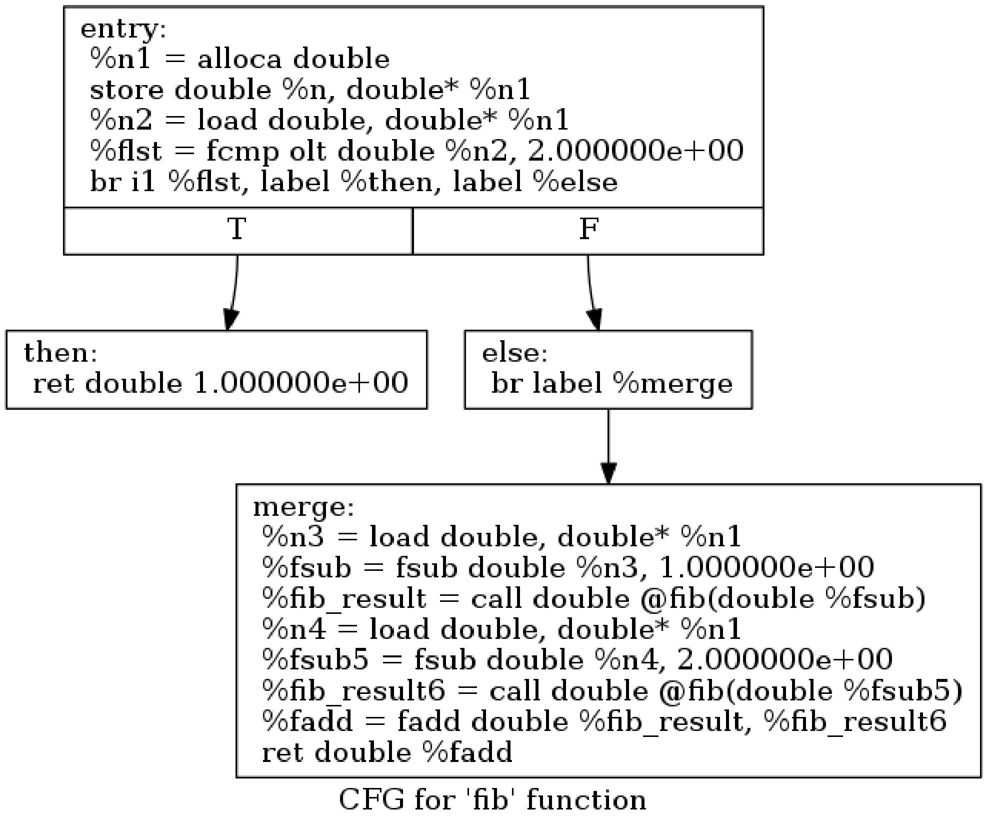
```
while:
 %i1 = load i32, i32* %i
 %tmp = icmp slt i32 %i1, 50
 br i1 %tmp, label %while_body, label %merge
```

```
while_body:
 %i2 = load i32, i32* %i
 %tmp3 = sitofp i32 %i2 to double
 %fib_result = call double @fib(double %tmp3)
 %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([7 x i8],
... [7 x i8]* @fmt.2, i32 0, i32 0), double %fib_result)
 %tmp4 = load i32, i32* %i
 %tmp5 = add i32 %tmp4, 1
 store i32 %tmp5, i32* %i
 br label %while
```

```
merge:
 ret void
```

Figure 6-1. Control-flow graph for ./examples/fib.goo main()

```
Generated with $ dot -Tpng cfg.main.dot > main.png
```



```
entry:
 %n1 = alloca double
 store double %n, double* %n1
 %n2 = load double, double* %n1
 %flst = fcmp olt double %n2, 2.000000e+00
 br i1 %flst, label %then, label %else
```

```
then:
 ret double 1.000000e+00
```

```
else:
 br label %merge
```

```
merge:
 %n3 = load double, double* %n1
 %fsub = fsub double %n3, 1.000000e+00
 %fib_result = call double @fib(double %fsub)
 %n4 = load double, double* %n1
 %fsub5 = fsub double %n4, 2.000000e+00
 %fib_result6 = call double @fib(double %fsub5)
 %fadd = fadd double %fib_result, %fib_result6
 ret double %fadd
```

Figure 6-2. Control-flow graph for ./examples/fib.goo fib()

6.1.2 Binary Search. ./bsearch.goo

Source Language Program (Casper)

```
int binarysearch(str s, str search) {
    str t;
    int first;
    int last;
    int middle;
    int answer;

    first = 0;
    last = strlen(s) - 1;
    middle = (first+last)/2;

    while (first <= last) {
      t = s ? middle;
      if ( t < search )
         first = middle + 1;
      else {
          if (t == search) {
             answer = middle;
             break;
          }
          else
            last = middle - 1;
      }
      middle = (first + last)/2;
    }
    if (first > last) answer = -1;

    return answer;
}

void main(){
    int answer;
    str search;

    search = "W";

    answer = binarysearch("Hello World", search);

    if(answer > -1)
        printsnl(search _ " found at position " _ answer);
    else
        printsnl("Not found! " _  search _ " isn't present in the list.");
}
```

Output

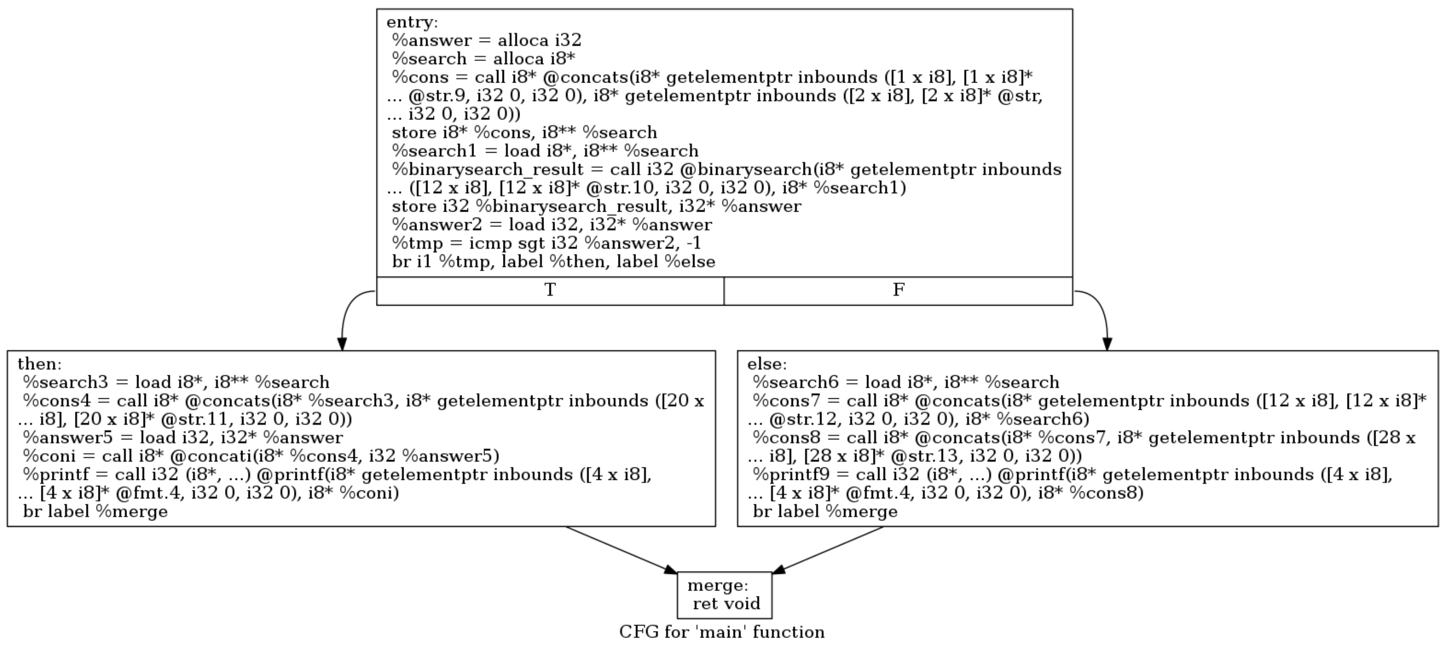
```
W found at position 6
```

Target Language Program (LLVM)

```
; ModuleID = 'Casper'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.2 = private unnamed_addr constant [7 x i8] c"%.10f\0A\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%g\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.5 = private unnamed_addr constant [3 x i8] c"%s\00"
```

```
@fmt.6 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.7 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@str = private unnamed_addr constant [2 x i8] c"W\00"
@str.9 = private unnamed_addr constant [1 x i8] zeroinitializer
@str.10 = private unnamed_addr constant [12 x i8] c"Hello World\00"
@str.11 = private unnamed_addr constant [20 x i8] c" found at position \00"
@str.12 = private unnamed_addr constant [12 x i8] c"Not found! \00"
@str.13 = private unnamed_addr constant [28 x i8] c" isn't present in the list.\00"
@fmt.14 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.15 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.16 = private unnamed_addr constant [7 x i8] c"%.10f\0A\00"
@fmt.17 = private unnamed_addr constant [3 x i8] c"%g\00"
@fmt.18 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.19 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.20 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.21 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.22 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@str.23 = private unnamed_addr constant [1 x i8] zeroinitializer

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

declare i32 @printb(i1, i32)

declare i32 @printbig(i32)

declare i8* @concats(i8*, i8*)

declare i8* @concati(i8*, i32)

declare i8* @concatf(i8*, double)

declare i8* @concatb(i8*, i1)

declare i8* @charat(i8*, i32)

declare i1 @scomp(i8*, i8*, i32)

declare i32 @isalnum(i8)

declare i32 @isalpha(i8)

declare i32 @iscntrl(i8)

declare i32 @isdigit(i8)

declare i32 @isgraph(i8)

declare i32 @islower(i8)

declare i32 @isprint(i8)

declare i32 @ispunct(i8)

declare i32 @isspace(i8)

declare i32 @isupper(i8)

declare i32 @isxdigit(i8)

declare i32 @strlen(i8*)

declare i8* @strcpy(i8*, i8*)
declare i8* @strncpy(i8*, i8*, i32)

declare i8* @strcat(i8*, i8*)

declare i32 @strcmp(i8*, i8*)

declare i8* @strchr(i8*, i8*)

declare i8* @strrchr(i8*, i8*)

declare i32 @strspn(i8*, i8*)

declare i32 @strcspn(i8*, i8*)

declare i8* @strstr(i8*, i8*)

declare i8* @strerror(i32)

declare i8* @strtok(i8*, i8*)

declare double @pow(double, double)

declare double @sin(double)

declare double @cos(double)

declare double @tan(double)

declare double @asin(double)

declare double @acos(double)

declare double @atan(double)

declare double @sinh(double)

declare double @cosh(double)

declare double @tanh(double)

declare double @exp(double)

declare double @log(double)

declare double @log10(double)

declare double @sqrt(double)

declare double @floor(double)

declare double @ceil(double)

declare double @fabs(double)

declare i32 @srand(i32, ...)

declare i32 @rand()

declare double @fmod(double, double)

define void @main() {
entry:
  %answer = alloca i32
  %search = alloca i8*
  %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.9, i32 0, i32 0),
i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str, i32 0, i32 0))
  store i8* %cons, i8** %search
  %search1 = load i8*, i8** %search
  %binarysearch_result = call i32 @binarysearch(i8* getelementptr inbounds ([12 x i8], [12 x i8]*
@str.10, i32 0, i32 0), i8* %search1)
  store i32 %binarysearch_result, i32* %answer
  %answer2 = load i32, i32* %answer
  %tmp = icmp sgt i32 %answer2, -1
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  ret void

then:                                             ; preds = %entry
  %search3 = load i8*, i8** %search
  %cons4 = call i8* @concats(i8* %search3, i8* getelementptr inbounds ([20 x i8], [20 x i8]*
@str.11, i32 0, i32 0))
  %answer5 = load i32, i32* %answer
  %coni = call i8* @concati(i8* %cons4, i32 %answer5)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.4, i32
0, i32 0), i8* %coni)
  br label %merge

else:                                             ; preds = %entry
  %search6 = load i8*, i8** %search
  %cons7 = call i8* @concats(i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str.12, i32 0, i32
0), i8* %search6)
  %cons8 = call i8* @concats(i8* %cons7, i8* getelementptr inbounds ([28 x i8], [28 x i8]* @str.13,
i32 0, i32 0))
  %printf9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.4,
i32 0, i32 0), i8* %cons8)
  br label %merge
}

define i32 @binarysearch(i8* %s, i8* %search) {
entry:
  %s1 = alloca i8*
  store i8* %s, i8** %s1
  %search2 = alloca i8*
  store i8* %search, i8** %search2
  %t = alloca i8*
  %first = alloca i32
  %last = alloca i32
  %middle = alloca i32
  %answer = alloca i32
  store i32 0, i32* %first
  %s3 = load i8*, i8** %s1
  %strlen = call i32 @strlen(i8* %s3)
  %tmp = sub i32 %strlen, 1
  store i32 %tmp, i32* %last
  %first4 = load i32, i32* %first
  %last5 = load i32, i32* %last
  %tmp6 = add i32 %first4, %last5
  %tmp7 = sdiv i32 %tmp6, 2
  store i32 %tmp7, i32* %middle
  br label %while

while:                                            ; preds = %merge15, %entry
  %first8 = load i32, i32* %first
  %last9 = load i32, i32* %last
  %tmp10 = icmp sle i32 %first8, %last9
  br i1 %tmp10, label %while_body, label %merge
```

```
while_body:                                       ; preds = %while
  %s11 = load i8*, i8** %s1
  %middle12 = load i32, i32* %middle
  %cat = call i8* @charat(i8* %s11, i32 %middle12)
  %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.23, i32 0, i32 0),
i8* %cat)
  store i8* %cons, i8** %t
  %t13 = load i8*, i8** %t
  %search14 = load i8*, i8** %search2
  %scomp = call i1 @scomp(i8* %t13, i8* %search14, i32 3)
  br i1 %scomp, label %then, label %else

merge:                                            ; preds = %then22, %while
  %first31 = load i32, i32* %first
  %last32 = load i32, i32* %last
  %tmp33 = icmp sgt i32 %first31, %last32
  br i1 %tmp33, label %then35, label %else36

merge15:                                          ; preds = %merge21, %then
  %first27 = load i32, i32* %first
  %last28 = load i32, i32* %last
  %tmp29 = add i32 %first27, %last28
  %tmp30 = sdiv i32 %tmp29, 2
  store i32 %tmp30, i32* %middle
  br label %while

then:                                             ; preds = %while_body
  %middle16 = load i32, i32* %middle
  %tmp17 = add i32 %middle16, 1
  store i32 %tmp17, i32* %first
  br label %merge15

else:                                             ; preds = %while_body
  %t18 = load i8*, i8** %t
  %search19 = load i8*, i8** %search2
  %scomp20 = call i1 @scomp(i8* %t18, i8* %search19, i32 1)
  br i1 %scomp20, label %then22, label %else24

merge21:                                          ; preds = %else24
  br label %merge15

then22:                                           ; preds = %else
  %middle23 = load i32, i32* %middle
  store i32 %middle23, i32* %answer
  br label %merge

else24:                                           ; preds = %else
  %middle25 = load i32, i32* %middle
  %tmp26 = sub i32 %middle25, 1
  store i32 %tmp26, i32* %last
  br label %merge21

merge34:                                          ; preds = %else36, %then35
  %answer37 = load i32, i32* %answer
  ret i32 %answer37

then35:                                           ; preds = %merge
  store i32 -1, i32* %answer
  br label %merge34

else36:                                           ; preds = %merge
  br label %merge34
}
```



```
entry:
 %answer = alloca i32
 %search = alloca i8*
 %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]*
... @str.9, i32 0, i32 0), i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str,
... i32 0, i32 0))
 store i8* %cons, i8** %search
 %search1 = load i8*, i8** %search
 %binarysearch_result = call i32 @binarysearch(i8* getelementptr inbounds
... ([12 x i8], [12 x i8]* @str.10, i32 0, i32 0), i8* %search1)
 store i32 %binarysearch_result, i32* %answer
 %answer2 = load i32, i32* %answer
 %tmp = icmp sgt i32 %answer2, -1
 br i1 %tmp, label %then, label %else
```

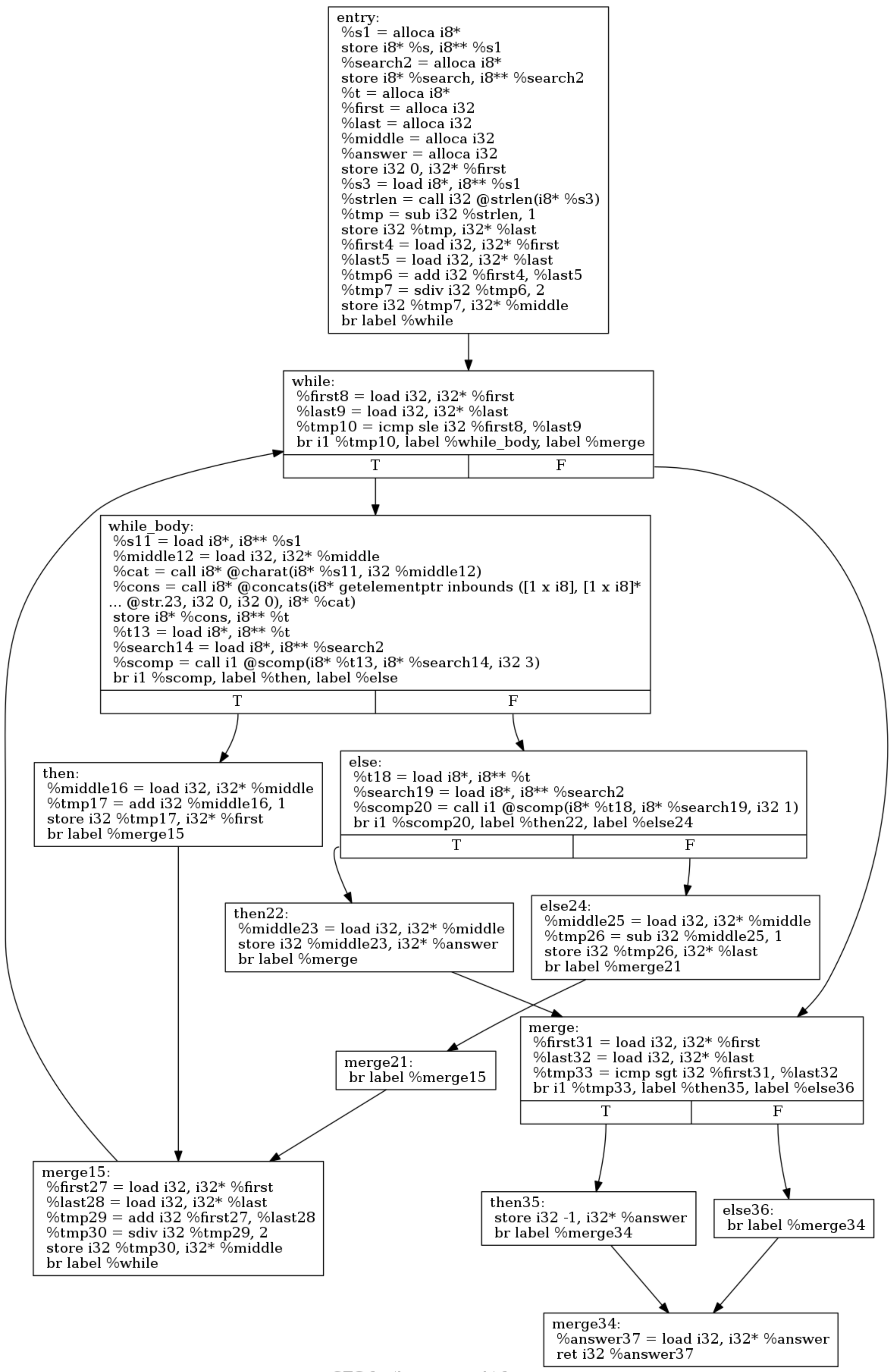
```
then:
 %search3 = load i8*, i8** %search
 %cons4 = call i8* @concats(i8* %search3, i8* getelementptr inbounds ([20 x
... i8], [20 x i8]* @str.11, i32 0, i32 0))
 %answer5 = load i32, i32* %answer
 %coni = call i8* @concati(i8* %cons4, i32 %answer5)
 %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
... [4 x i8]* @fmt.4, i32 0, i32 0), i8* %coni)
 br label %merge
```

```
else:
 %search6 = load i8*, i8** %search
 %cons7 = call i8* @concats(i8* getelementptr inbounds ([12 x i8], [12 x i8]*
... @str.12, i32 0, i32 0), i8* %search6)
 %cons8 = call i8* @concats(i8* %cons7, i8* getelementptr inbounds ([28 x
... i8], [28 x i8]* @str.13, i32 0, i32 0))
 %printf9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
... [4 x i8]* @fmt.4, i32 0, i32 0), i8* %cons8)
 br label %merge
```

```
merge:
 ret void
```

CFG for 'main' function

Figure 6-3. Control-flow graph for ./examples/bsearch.goo main()



```
entry:
 %s1 = alloca i8*
 store i8* %s, i8** %s1
 %search2 = alloca i8*
 store i8* %search, i8** %search2
 %t = alloca i8*
 %first = alloca i32
 %last = alloca i32
 %middle = alloca i32
 %answer = alloca i32
 store i32 0, i32* %first
 %s3 = load i8*, i8** %s1
 %strlen = call i32 @strlen(i8* %s3)
 %tmp = sub i32 %strlen, 1
 store i32 %tmp, i32* %last
 %first4 = load i32, i32* %first
 %last5 = load i32, i32* %last
 %tmp6 = add i32 %first4, %last5
 %tmp7 = sdiv i32 %tmp6, 2
 store i32 %tmp7, i32* %middle
 br label %while
```

```
while:
 %first8 = load i32, i32* %first
 %last9 = load i32, i32* %last
 %tmp10 = icmp sle i32 %first8, %last9
 br i1 %tmp10, label %while_body, label %merge
```

```
while_body:
 %s11 = load i8*, i8** %s1
 %middle12 = load i32, i32* %middle
 %cat = call i8* @charat(i8* %s11, i32 %middle12)
 %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]*
... @str.23, i32 0, i32 0), i8* %cat)
 store i8* %cons, i8** %t
 %t13 = load i8*, i8** %t
 %search14 = load i8*, i8** %search2
 %scomp = call i1 @scomp(i8* %t13, i8* %search14, i32 3)
 br i1 %scomp, label %then, label %else
```

```
then:
 %middle16 = load i32, i32* %middle
 %tmp17 = add i32 %middle16, 1
 store i32 %tmp17, i32* %first
 br label %merge15
```

```
else:
 %t18 = load i8*, i8** %t
 %search19 = load i8*, i8** %search2
 %scomp20 = call i1 @scomp(i8* %t18, i8* %search19, i32 1)
 br i1 %scomp20, label %then22, label %else24
```

```
then22:
 %middle23 = load i32, i32* %middle
 store i32 %middle23, i32* %answer
 br label %merge
```

```
else24:
 %middle25 = load i32, i32* %middle
 %tmp26 = sub i32 %middle25, 1
 store i32 %tmp26, i32* %last
 br label %merge21
```

```
merge:
 %first31 = load i32, i32* %first
 %last32 = load i32, i32* %last
 %tmp33 = icmp sgt i32 %first31, %last32
 br i1 %tmp33, label %then35, label %else36
```

```
merge21:
 br label %merge15
```

```
merge15:
 %first27 = load i32, i32* %first
 %last28 = load i32, i32* %last
 %tmp29 = add i32 %first27, %last28
 %tmp30 = sdiv i32 %tmp29, 2
 store i32 %tmp30, i32* %middle
 br label %while
```

```
then35:
 store i32 -1, i32* %answer
 br label %merge34
```

```
else36:
 br label %merge34
```

```
merge34:
 %answer37 = load i32, i32* %answer
 ret i32 %answer37
```

CFG for 'binarysearch' function

Figure 6-4. Control-flow graph for ./examples/bsearch.goo binarysearch()

6.1.3 Sieve Of Eratosthenes, ./examples/sieve.goo

Source Language Program (Casper)

```
/* get the left len characters from string s */
str left(str s, int len){
    int i;
    str answer;

    answer = "";
    i = strlen(s);
    if (len > i) len = i;
    if (len < 0) len = i;

    for(i = 0; i < len; i +=1) {
        answer _= s?i;
    }

    return answer;
}

/* get the middle len characters from string s
   starting at position start                 */
str mid(str s, int start, int len){
    int i;
    str answer;

    answer = "";
    i = strlen(s);
    if (len > i) len = i;
    if (len < 0) len = i;
    if (start > i) start = 0;
    if (start < 0) start = 0;

    for(i = start; i < start + len; i +=1) {
        answer _= s?i;
    }

    return answer;
}

/* make a sequence of T’s for true
   change the positions in the sequence
   to F if it is a multiple of a prime */
void SieveOfEratosthenes(int n)
{
    int i; int p;
    str prime;

    prime = "T";

    for(i=0; i<n; i+=1)
        prime _= "T";

    for (p=2; p*p<=n; p+=1) {
        if(prime?p == "T")
            for (i=p*p; i<=n; i+= p)
                prime = left(prime, i) _ "F" _ mid(prime, ++i, -1);
    }

    for(p=2; p<=n; p+=1)
        if(prime?p == "T")
          printinl(p);
}
```

```
int main() {
    int n;
    n = 100;

    prints("
Sift the Two's and Sift the Three's,
The Sieve of Eratosthenes.
When the multiples sublime,
The numbers that remain are Prime.

Anonymous

");

    printsnl("Following are the prime numbers smaller than or equal to " _ n);
    SieveOfEratosthenes(n);
    return 0;
}
```

Output

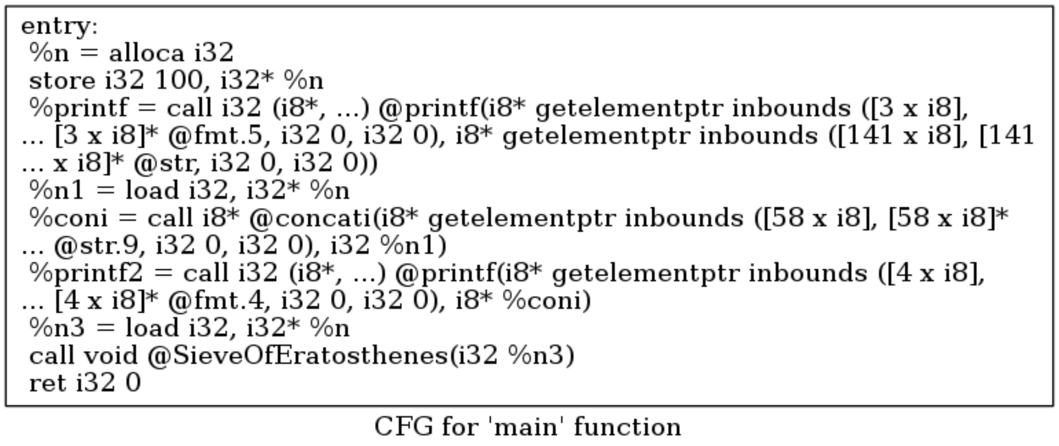
```
Sift the Two's and Sift the Three's,
The Sieve of Eratosthenes.
When the multiples sublime,
The numbers that remain are Prime.

Anonymous

Following are the prime numbers smaller than or equal to 100
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
```

Target Language Program (LLVM)

```
; ModuleID = 'Casper'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.2 = private unnamed_addr constant [7 x i8] c"%.10f\0A\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%g\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.5 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.7 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@str = private unnamed_addr constant [141 x i8] c"\0ASift the Two's and Sift the Three's,\0AThe
Sieve of Eratosthenes.\0AWhen the multiples sublime,\0AThe numbers that remain are
Prime.\0A\0AAnonymous\0A\0A\00"
@str.9 = private unnamed_addr constant [58 x i8] c"Following are the prime numbers smaller than or
equal to \00"
@fmt.10 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.11 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.12 = private unnamed_addr constant [7 x i8] c"%.10f\0A\00"
@fmt.13 = private unnamed_addr constant [3 x i8] c"%g\00"
@fmt.14 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.15 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.16 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.17 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.18 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@str.19 = private unnamed_addr constant [2 x i8] c"T\00"
@str.20 = private unnamed_addr constant [1 x i8] zeroinitializer
@str.21 = private unnamed_addr constant [2 x i8] c"T\00"
@str.22 = private unnamed_addr constant [2 x i8] c"T\00"
@str.23 = private unnamed_addr constant [2 x i8] c"F\00"
@str.24 = private unnamed_addr constant [1 x i8] zeroinitializer
@str.25 = private unnamed_addr constant [2 x i8] c"T\00"
@fmt.26 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.27 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.28 = private unnamed_addr constant [7 x i8] c"%.10f\0A\00"
@fmt.29 = private unnamed_addr constant [3 x i8] c"%g\00"
@fmt.30 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.31 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.32 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.33 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.34 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@str.35 = private unnamed_addr constant [1 x i8] zeroinitializer
@str.36 = private unnamed_addr constant [1 x i8] zeroinitializer
@fmt.37 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.38 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.39 = private unnamed_addr constant [7 x i8] c"%.10f\0A\00"
@fmt.40 = private unnamed_addr constant [3 x i8] c"%g\00"
@fmt.41 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.42 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.43 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.44 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.45 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@str.46 = private unnamed_addr constant [1 x i8] zeroinitializer
@str.47 = private unnamed_addr constant [1 x i8] zeroinitializer

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

declare i32 @printb(i1, i32)

declare i32 @printbig(i32)

declare i8* @concats(i8*, i8*)
declare i8* @concati(i8*, i32)

declare i8* @concatf(i8*, double)

declare i8* @concatb(i8*, i1)

declare i8* @charat(i8*, i32)

declare i1 @scomp(i8*, i8*, i32)

declare i32 @isalnum(i8)

declare i32 @isalpha(i8)

declare i32 @iscntrl(i8)

declare i32 @isdigit(i8)

declare i32 @isgraph(i8)

declare i32 @islower(i8)

declare i32 @isprint(i8)

declare i32 @ispunct(i8)

declare i32 @isspace(i8)

declare i32 @isupper(i8)

declare i32 @isxdigit(i8)

declare i32 @strlen(i8*)

declare i8* @strcpy(i8*, i8*)

declare i8* @strncpy(i8*, i8*, i32)

declare i8* @strcat(i8*, i8*)

declare i32 @strcmp(i8*, i8*)

declare i8* @strchr(i8*, i8*)

declare i8* @strrchr(i8*, i8*)

declare i32 @strspn(i8*, i8*)

declare i32 @strcspn(i8*, i8*)

declare i8* @strstr(i8*, i8*)

declare i8* @strerror(i32)

declare i8* @strtok(i8*, i8*)

declare double @pow(double, double)

declare double @sin(double)

declare double @cos(double)

declare double @tan(double)
declare double @asin(double)

declare double @acos(double)

declare double @atan(double)

declare double @sinh(double)

declare double @cosh(double)

declare double @tanh(double)

declare double @exp(double)

declare double @log(double)

declare double @log10(double)

declare double @sqrt(double)

declare double @floor(double)

declare double @ceil(double)

declare double @fabs(double)

declare i32 @srand(i32, ...)

declare i32 @rand()

declare double @fmod(double, double)

define i32 @main() {
entry:
  %n = alloca i32
  store i32 100, i32* %n
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.5, i32
0, i32 0), i8* getelementptr inbounds ([141 x i8], [141 x i8]* @str, i32 0, i32 0))
  %n1 = load i32, i32* %n
  %coni = call i8* @concati(i8* getelementptr inbounds ([58 x i8], [58 x i8]* @str.9, i32 0, i32
0), i32 %n1)
  %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.4,
i32 0, i32 0), i8* %coni)
  %n3 = load i32, i32* %n
  call void @SieveOfEratosthenes(i32 %n3)
  ret i32 0
}

define void @SieveOfEratosthenes(i32 %n) {
entry:
  %n1 = alloca i32
  store i32 %n, i32* %n1
  %i = alloca i32
  %p = alloca i32
  %prime = alloca i8*
  %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.20, i32 0, i32 0),
i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.19, i32 0, i32 0))
  store i8* %cons, i8** %prime
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %while_body, %entry
  %i2 = load i32, i32* %i
  %n3 = load i32, i32* %n1
  %tmp = icmp slt i32 %i2, %n3
  br i1 %tmp, label %while_body, label %merge

while_body:                                       ; preds = %while
  %tmp4 = load i8*, i8** %prime
  %cons5 = call i8* @concats(i8* %tmp4, i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.21,
i32 0, i32 0))
  store i8* %cons5, i8** %prime
  %tmp6 = load i32, i32* %i
  %tmp7 = add i32 %tmp6, 1
  store i32 %tmp7, i32* %i
  br label %while

merge:                                            ; preds = %while
  store i32 2, i32* %p
  br label %while8

while8:                                           ; preds = %merge18, %merge
  %p11 = load i32, i32* %p
  %p12 = load i32, i32* %p
  %tmp13 = mul i32 %p11, %p12
  %n14 = load i32, i32* %n1
  %tmp15 = icmp sle i32 %tmp13, %n14
  br i1 %tmp15, label %while_body9, label %merge10

while_body9:                                      ; preds = %while8
  %prime16 = load i8*, i8** %prime
  %p17 = load i32, i32* %p
  %cat = call i8* @charat(i8* %prime16, i32 %p17)
  %scomp = call i1 @scomp(i8* %cat, i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.22, i32 0,
i32 0), i32 1)
  br i1 %scomp, label %then, label %else

merge10:                                          ; preds = %while8
  store i32 2, i32* %p
  br label %while41

merge18:                                          ; preds = %else, %merge24
  %tmp39 = load i32, i32* %p
  %tmp40 = add i32 %tmp39, 1
  store i32 %tmp40, i32* %p
  br label %while8

then:                                             ; preds = %while_body9
  %p19 = load i32, i32* %p
  %p20 = load i32, i32* %p
  %tmp21 = mul i32 %p19, %p20
  store i32 %tmp21, i32* %i
  br label %while22

while22:                                          ; preds = %while_body23, %then
  %i25 = load i32, i32* %i
  %n26 = load i32, i32* %n1
  %tmp27 = icmp sle i32 %i25, %n26
  br i1 %tmp27, label %while_body23, label %merge24

while_body23:                                     ; preds = %while22
  %i28 = load i32, i32* %i
  %prime29 = load i8*, i8** %prime
  %left_result = call i8* @left(i8* %prime29, i32 %i28)
  %cons30 = call i8* @concats(i8* %left_result, i8* getelementptr inbounds ([2 x i8], [2 x i8]*
@str.23, i32 0, i32 0))
  %i31 = load i32, i32* %i
  %tmp32 = add i32 %i31, 1
  %prime33 = load i8*, i8** %prime
  %mid_result = call i8* @mid(i8* %prime33, i32 %tmp32, i32 -1)
  %cons34 = call i8* @concats(i8* %cons30, i8* %mid_result)
  %cons35 = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.24, i32 0, i32
0), i8* %cons34)
  store i8* %cons35, i8** %prime
  %p36 = load i32, i32* %p
  %tmp37 = load i32, i32* %i
  %tmp38 = add i32 %tmp37, %p36
  store i32 %tmp38, i32* %i
  br label %while22

merge24:                                          ; preds = %while22
  br label %merge18

else:                                             ; preds = %while_body9
  br label %merge18

while41:                                          ; preds = %merge51, %merge10
  %p44 = load i32, i32* %p
  %n45 = load i32, i32* %n1
  %tmp46 = icmp sle i32 %p44, %n45
  br i1 %tmp46, label %while_body42, label %merge43

while_body42:                                     ; preds = %while41
  %prime47 = load i8*, i8** %prime
  %p48 = load i32, i32* %p
  %cat49 = call i8* @charat(i8* %prime47, i32 %p48)
  %scomp50 = call i1 @scomp(i8* %cat49, i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.25,
i32 0, i32 0), i32 1)
  br i1 %scomp50, label %then52, label %else54

merge43:                                          ; preds = %while41
  ret void

merge51:                                          ; preds = %else54, %then52
  %tmp55 = load i32, i32* %p
  %tmp56 = add i32 %tmp55, 1
  store i32 %tmp56, i32* %p
  br label %while41

then52:                                           ; preds = %while_body42
  %p53 = load i32, i32* %p
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.10,
i32 0, i32 0), i32 %p53)
  br label %merge51

else54:                                           ; preds = %while_body42
  br label %merge51
}

define i8* @mid(i8* %s, i32 %start, i32 %len) {
entry:
  %s1 = alloca i8*
  store i8* %s, i8** %s1
  %start2 = alloca i32
  store i32 %start, i32* %start2
  %len3 = alloca i32
  store i32 %len, i32* %len3
  %i = alloca i32
  %answer = alloca i8*
  %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.36, i32 0, i32 0),
i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.35, i32 0, i32 0))
```

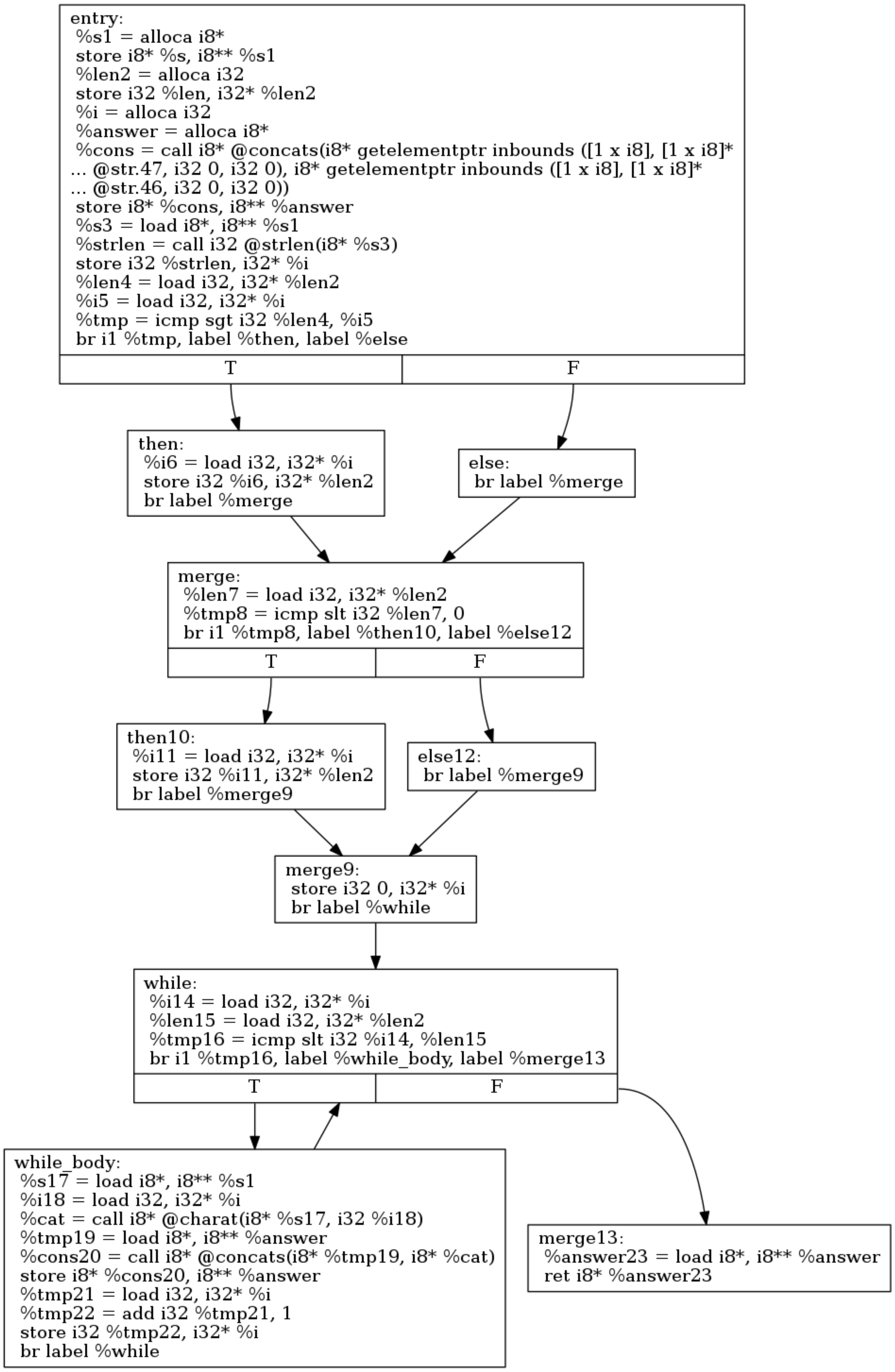
```
  store i8* %cons, i8** %answer
  %s4 = load i8*, i8** %s1
  %strlen = call i32 @strlen(i8* %s4)
  store i32 %strlen, i32* %i
  %len5 = load i32, i32* %len3
  %i6 = load i32, i32* %i
  %tmp = icmp sgt i32 %len5, %i6
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  %len8 = load i32, i32* %len3
  %tmp9 = icmp slt i32 %len8, 0
  br i1 %tmp9, label %then11, label %else13

then:                                             ; preds = %entry
  %i7 = load i32, i32* %i
  store i32 %i7, i32* %len3
  br label %merge

else:                                             ; preds = %entry
  br label %merge

merge10:                                          ; preds = %else13, %then11
  %start14 = load i32, i32* %start2
  %i15 = load i32, i32* %i
  %tmp16 = icmp sgt i32 %start14, %i15
  br i1 %tmp16, label %then18, label %else19

then11:                                           ; preds = %merge
  %i12 = load i32, i32* %i
  store i32 %i12, i32* %len3
  br label %merge10

else13:                                           ; preds = %merge
  br label %merge10

merge17:                                          ; preds = %else19, %then18
  %start20 = load i32, i32* %start2
  %tmp21 = icmp slt i32 %start20, 0
  br i1 %tmp21, label %then23, label %else24

then18:                                           ; preds = %merge10
  store i32 0, i32* %start2
  br label %merge17

else19:                                           ; preds = %merge10
  br label %merge17

merge22:                                          ; preds = %else24, %then23
  %start25 = load i32, i32* %start2
  store i32 %start25, i32* %i
  br label %while

then23:                                           ; preds = %merge17
  store i32 0, i32* %start2
  br label %merge22

else24:                                           ; preds = %merge17
  br label %merge22

while:                                            ; preds = %while_body, %merge22
  %i27 = load i32, i32* %i
  %start28 = load i32, i32* %start2
  %len29 = load i32, i32* %len3
  %tmp30 = add i32 %start28, %len29
  %tmp31 = icmp slt i32 %i27, %tmp30
  br i1 %tmp31, label %while_body, label %merge26

while_body:                                       ; preds = %while
  %s32 = load i8*, i8** %s1
  %i33 = load i32, i32* %i
  %cat = call i8* @charat(i8* %s32, i32 %i33)
  %tmp34 = load i8*, i8** %answer
  %cons35 = call i8* @concats(i8* %tmp34, i8* %cat)
  store i8* %cons35, i8** %answer
  %tmp36 = load i32, i32* %i
  %tmp37 = add i32 %tmp36, 1
  store i32 %tmp37, i32* %i
  br label %while

merge26:                                          ; preds = %while
  %answer38 = load i8*, i8** %answer
  ret i8* %answer38
}

define i8* @left(i8* %s, i32 %len) {
entry:
  %s1 = alloca i8*
  store i8* %s, i8** %s1
  %len2 = alloca i32
  store i32 %len, i32* %len2
  %i = alloca i32
  %answer = alloca i8*
  %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.47, i32 0, i32 0),
i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.46, i32 0, i32 0))
  store i8* %cons, i8** %answer
  %s3 = load i8*, i8** %s1
  %strlen = call i32 @strlen(i8* %s3)
  store i32 %strlen, i32* %i
  %len4 = load i32, i32* %len2
  %i5 = load i32, i32* %i
  %tmp = icmp sgt i32 %len4, %i5
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  %len7 = load i32, i32* %len2
  %tmp8 = icmp slt i32 %len7, 0
  br i1 %tmp8, label %then10, label %else12

then:                                             ; preds = %entry
  %i6 = load i32, i32* %i
  store i32 %i6, i32* %len2
  br label %merge

else:                                             ; preds = %entry
  br label %merge

merge9:                                           ; preds = %else12, %then10
  store i32 0, i32* %i
  br label %while

then10:                                           ; preds = %merge
  %i11 = load i32, i32* %i
  store i32 %i11, i32* %len2
  br label %merge9

else12:                                           ; preds = %merge
  br label %merge9
```

```
while:                                              ; preds = %while_body, %merge9
  %i14 = load i32, i32* %i
  %len15 = load i32, i32* %len2
  %tmp16 = icmp slt i32 %i14, %len15
  br i1 %tmp16, label %while_body, label %merge13

while_body:                                         ; preds = %while
  %s17 = load i8*, i8** %s1
  %i18 = load i32, i32* %i
  %cat = call i8* @charat(i8* %s17, i32 %i18)
  %tmp19 = load i8*, i8** %answer
  %cons20 = call i8* @concats(i8* %tmp19, i8* %cat)
  store i8* %cons20, i8** %answer
  %tmp21 = load i32, i32* %i
  %tmp22 = add i32 %tmp21, 1
  store i32 %tmp22, i32* %i
  br label %while

merge13:                                            ; preds = %while
  %answer23 = load i8*, i8** %answer
  ret i8* %answer23
}
```



```
entry:
 %n = alloca i32
 store i32 100, i32* %n
 %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
... [3 x i8]* @fmt.5, i32 0, i32 0), i8* getelementptr inbounds ([141 x i8], [141
... x i8]* @str, i32 0, i32 0))
 %n1 = load i32, i32* %n
 %coni = call i8* @concati(i8* getelementptr inbounds ([58 x i8], [58 x i8]*
... @str.9, i32 0, i32 0), i32 %n1)
 %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
... [4 x i8]* @fmt.4, i32 0, i32 0), i8* %coni)
 %n3 = load i32, i32* %n
 call void @SieveOfEratosthenes(i32 %n3)
 ret i32 0
```

Figure 6-5. Control-flow graph for ./examples/sieve.goo main()



```
entry:
 %s1 = alloca i8*
 store i8* %s, i8** %s1
 %len2 = alloca i32
 store i32 %len, i32* %len2
 %i = alloca i32
 %answer = alloca i8*
 %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]*
... @str.47, i32 0, i32 0), i8* getelementptr inbounds ([1 x i8], [1 x i8]*
... @str.46, i32 0, i32 0))
 store i8* %cons, i8** %answer
 %s3 = load i8*, i8** %s1
 %strlen = call i32 @strlen(i8* %s3)
 store i32 %strlen, i32* %i
 %len4 = load i32, i32* %len2
 %i5 = load i32, i32* %i
 %tmp = icmp sgt i32 %len4, %i5
 br i1 %tmp, label %then, label %else
```

```
then:
 %i6 = load i32, i32* %i
 store i32 %i6, i32* %len2
 br label %merge
```

```
else:
 br label %merge
```

```
merge:
 %len7 = load i32, i32* %len2
 %tmp8 = icmp slt i32 %len7, 0
 br i1 %tmp8, label %then10, label %else12
```

```
then10:
 %i11 = load i32, i32* %i
 store i32 %i11, i32* %len2
 br label %merge9
```

```
else12:
 br label %merge9
```

```
merge9:
 store i32 0, i32* %i
 br label %while
```

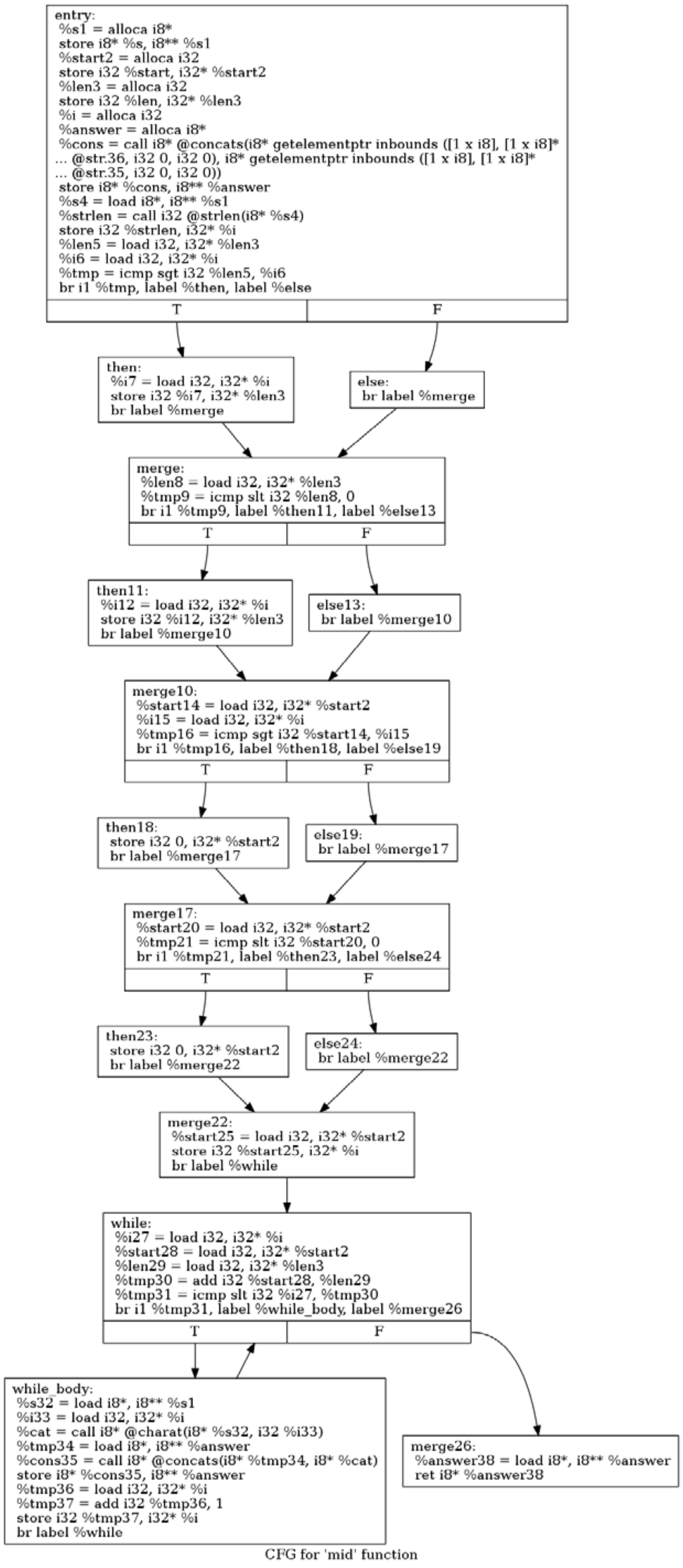
```
while:
 %i14 = load i32, i32* %i
 %len15 = load i32, i32* %len2
 %tmp16 = icmp slt i32 %i14, %len15
 br i1 %tmp16, label %while_body, label %merge13
```

```
while_body:
 %s17 = load i8*, i8** %s1
 %i18 = load i32, i32* %i
 %cat = call i8* @charat(i8* %s17, i32 %i18)
 %tmp19 = load i8*, i8** %answer
 %cons20 = call i8* @concats(i8* %tmp19, i8* %cat)
 store i8* %cons20, i8** %answer
 %tmp21 = load i32, i32* %i
 %tmp22 = add i32 %tmp21, 1
 store i32 %tmp22, i32* %i
 br label %while
```

```
merge13:
 %answer23 = load i8*, i8** %answer
 ret i8* %answer23
```

CFG for 'left' function

Figure 6-6. Control-flow graph for ./examples/sieve.goo left()



```
entry:
 %s1 = alloca i8*
 store i8* %s, i8** %s1
 %start2 = alloca i32
 store i32 %start, i32* %start2
 %len3 = alloca i32
 store i32 %len, i32* %len3
 %i = alloca i32
 %answer = alloca i8*
 %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]*
... @str.36, i32 0, i32 0), i8* getelementptr inbounds ([1 x i8], [1 x i8]*
... @str.35, i32 0, i32 0))
 store i8* %cons, i8** %answer
 %s4 = load i8*, i8** %s1
 %strlen = call i32 @strlen(i8* %s4)
 store i32 %strlen, i32* %i
 %len5 = load i32, i32* %len3
 %i6 = load i32, i32* %i
 %tmp = icmp sgt i32 %len5, %i6
 br i1 %tmp, label %then, label %else
```

```
then:
 %i7 = load i32, i32* %i
 store i32 %i7, i32* %len3
 br label %merge
```

```
else:
 br label %merge
```

```
merge:
 %len8 = load i32, i32* %len3
 %tmp9 = icmp slt i32 %len8, 0
 br i1 %tmp9, label %then11, label %else13
```

```
then11:
 %i12 = load i32, i32* %i
 store i32 %i12, i32* %len3
 br label %merge10
```

```
else13:
 br label %merge10
```

```
merge10:
 %start14 = load i32, i32* %start2
 %i15 = load i32, i32* %i
 %tmp16 = icmp sgt i32 %start14, %i15
 br i1 %tmp16, label %then18, label %else19
```

```
then18:
 store i32 0, i32* %start2
 br label %merge17
```

```
else19:
 br label %merge17
```

```
merge17:
 %start20 = load i32, i32* %start2
 %tmp21 = icmp slt i32 %start20, 0
 br i1 %tmp21, label %then23, label %else24
```

```
then23:
 store i32 0, i32* %start2
 br label %merge22
```

```
else24:
 br label %merge22
```

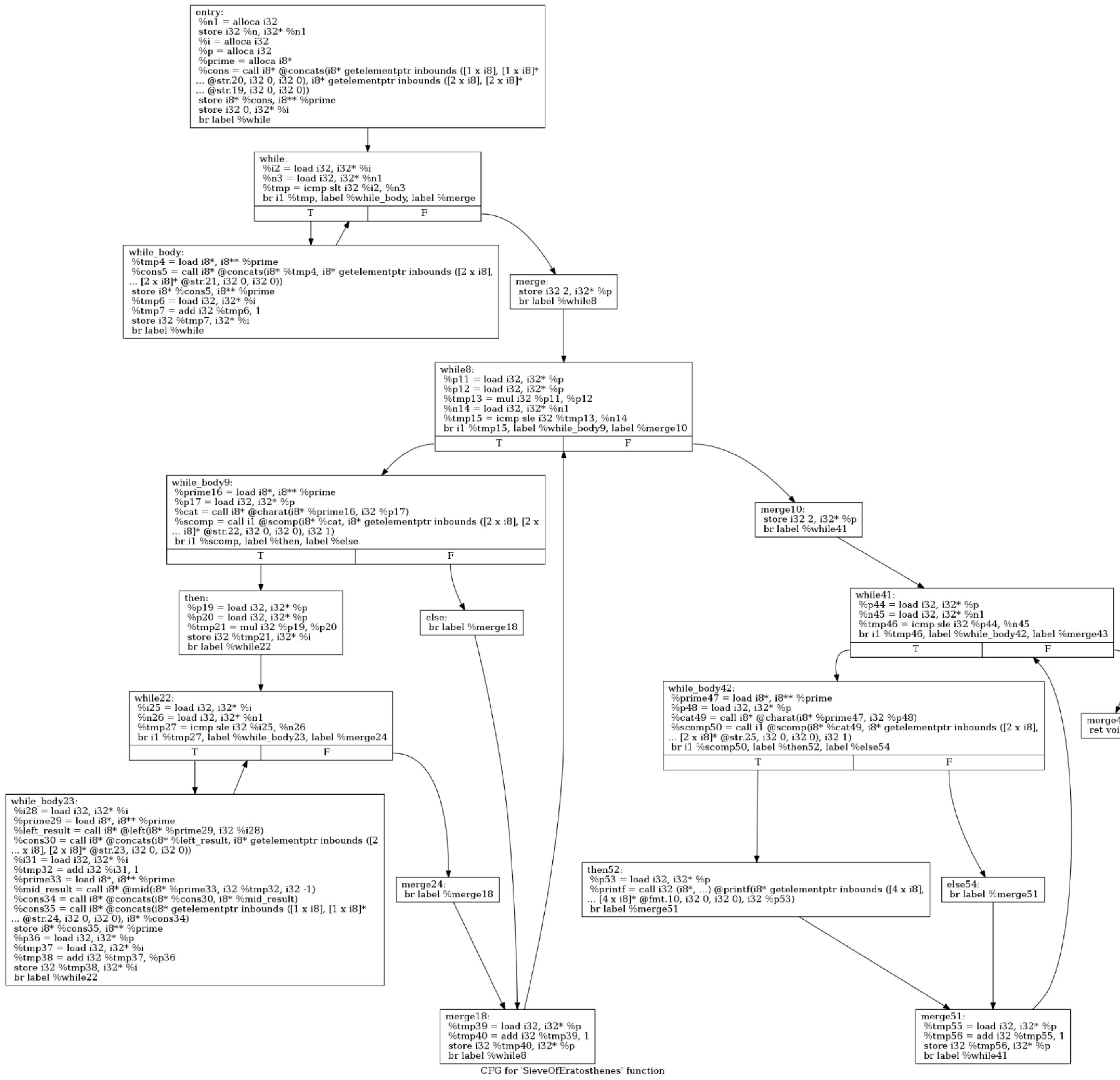
```
merge22:
 %start25 = load i32, i32* %start2
 store i32 %start25, i32* %i
 br label %while
```

```
while:
 %i27 = load i32, i32* %i
 %start28 = load i32, i32* %start2
 %len29 = load i32, i32* %len3
 %tmp30 = add i32 %start28, %len29
 %tmp31 = icmp slt i32 %i27, %tmp30
 br i1 %tmp31, label %while_body, label %merge26
```

```
while_body:
 %s32 = load i8*, i8** %s1
 %i33 = load i32, i32* %i
 %cat = call i8* @charat(i8* %s32, i32 %i33)
 %tmp34 = load i8*, i8** %answer
 %cons35 = call i8* @concats(i8* %tmp34, i8* %cat)
 store i8* %cons35, i8** %answer
 %tmp36 = load i32, i32* %i
 %tmp37 = add i32 %tmp36, 1
 store i32 %tmp37, i32* %i
 br label %while
```

```
merge26:
 %answer38 = load i8*, i8** %answer
 ret i8* %answer38
```

Figure 6-7. Control-flow graph for ./examples/sieve.goo mid()



```
entry:
 %n1 = alloca i32
 store i32 %n, i32* %n1
 %i = alloca i32
 %p = alloca i32
 %prime = alloca i8*
 %cons = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]*
... @str.20, i32 0, i32 0), i8* getelementptr inbounds ([2 x i8], [2 x i8]*
... @str.19, i32 0, i32 0))
 store i8* %cons, i8** %prime
 store i32 0, i32* %i
 br label %while
```

```
while:
 %i2 = load i32, i32* %i
 %n3 = load i32, i32* %n1
 %tmp = icmp slt i32 %i2, %n3
 br i1 %tmp, label %while_body, label %merge
```

```
while_body:
 %tmp4 = load i8*, i8** %prime
 %cons5 = call i8* @concats(i8* %tmp4, i8* getelementptr inbounds ([2 x i8],
... [2 x i8]* @str.21, i32 0, i32 0))
 store i8* %cons5, i8** %prime
 %tmp6 = load i32, i32* %i
 %tmp7 = add i32 %tmp6, 1
 store i32 %tmp7, i32* %i
 br label %while
```

```
merge:
 store i32 2, i32* %p
 br label %while8
```

```
while8:
 %p11 = load i32, i32* %p
 %p12 = load i32, i32* %p
 %tmp13 = mul i32 %p11, %p12
 %n14 = load i32, i32* %n1
 %tmp15 = icmp sle i32 %tmp13, %n14
 br i1 %tmp15, label %while_body9, label %merge10
```

```
while_body9:
 %prime16 = load i8*, i8** %prime
 %p17 = load i32, i32* %p
 %cat = call i8* @charat(i8* %prime16, i32 %p17)
 %scomp = call i1 @scomp(i8* %cat, i8* getelementptr inbounds ([2 x i8], [2 x
... i8]* @str.22, i32 0, i32 0), i32 1)
 br i1 %scomp, label %then, label %else
```

```
then:
 %p19 = load i32, i32* %p
 %p20 = load i32, i32* %p
 %tmp21 = mul i32 %p19, %p20
 store i32 %tmp21, i32* %i
 br label %while22
```

```
else:
 br label %merge18
```

```
while22:
 %i25 = load i32, i32* %i
 %n26 = load i32, i32* %n1
 %tmp27 = icmp sle i32 %i25, %n26
 br i1 %tmp27, label %while_body23, label %merge24
```

```
while_body23:
 %i28 = load i32, i32* %i
 %prime29 = load i8*, i8** %prime
 %left_result = call i8* @left(i8* %prime29, i32 %i28)
 %cons30 = call i8* @concats(i8* %left_result, i8* getelementptr inbounds ([2
... x i8], [2 x i8]* @str.23, i32 0, i32 0))
 %i31 = load i32, i32* %i
 %tmp32 = add i32 %i31, 1
 %prime33 = load i8*, i8** %prime
 %mid_result = call i8* @mid(i8* %prime33, i32 %tmp32, i32 -1)
 %cons34 = call i8* @concats(i8* %cons30, i8* %mid_result)
 %cons35 = call i8* @concats(i8* getelementptr inbounds ([1 x i8], [1 x i8]*
... @str.24, i32 0, i32 0), i8* %cons34)
 store i8* %cons35, i8** %prime
 %p36 = load i32, i32* %p
 %tmp37 = load i32, i32* %i
 %tmp38 = add i32 %tmp37, %p36
 store i32 %tmp38, i32* %i
 br label %while22
```

```
merge24:
 br label %merge18
```

```
merge18:
 %tmp39 = load i32, i32* %p
 %tmp40 = add i32 %tmp39, 1
 store i32 %tmp40, i32* %p
 br label %while8
```

```
merge10:
 store i32 2, i32* %p
 br label %while41
```

```
while41:
 %p44 = load i32, i32* %p
 %n45 = load i32, i32* %n1
 %tmp46 = icmp sle i32 %p44, %n45
 br i1 %tmp46, label %while_body42, label %merge43
```

```
while_body42:
 %prime47 = load i8*, i8** %prime
 %p48 = load i32, i32* %p
 %cat49 = call i8* @charat(i8* %prime47, i32 %p48)
 %scomp50 = call i1 @scomp(i8* %cat49, i8* getelementptr inbounds ([2 x i8],
... [2 x i8]* @str.25, i32 0, i32 0), i32 1)
 br i1 %scomp50, label %then52, label %else54
```

```
merge43:
 ret void
```

```
then52:
 %p53 = load i32, i32* %p
 %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
... [4 x i8]* @fmt.10, i32 0, i32 0), i32 %p53)
 br label %merge51
```

```
else54:
 br label %merge51
```

```
merge51:
 %tmp55 = load i32, i32* %p
 %tmp56 = add i32 %tmp55, 1
 store i32 %tmp56, i32* %p
 br label %while41
```

Figure 6-8. Control-flow graph for ./examples/sieve.goo SieveOfEratosthenes()

6.2 Test Automation

Built into the `./Makefile` is the option to run the test suite:

```
$ make test
```

This calls the `./makesupport/testall.sh` script which was adapted from MicroC and which compiles, runs and compares the output of all the programs in the `./tests` directory to their predefined output.

There are two types of tests, the program filenames that start with:

1. `./tests/test-` These are test cases that compile and produce an output. They were chosen while implementing a specific language feature to test it, ranging from variable type declarations to assignments, control-flow and calling library functions.
2. `./tests/fail-` These are test cases that fail to compile and produce an error message. They were chosen when a new error message was added to the translator to verify it is properly produced at compile time.

There are 183 test programs in the `./tests` directory and are listed in the Appendix 8.12.

6.3 Other Testing

Besides the automated test suite, there were also unit tests for the Scanner and the Parser. In addition, the functionality was added to the compiling scripts to produce the dot/png files of the LLVM as shown in Section 6.1 to verify the code produced. The pretty-printing options of casper.native were used during testing as explained in Section 5.1.10. Clang was also added, as explained in Section 5.1.11 to compare code to a C program equivalent, since Casper is closely related to MicroC and C.

6.3.1 Scanner Unit Test, ./makesupport/testscanner.sh and ./X/CasperScannerValidationInput.txt

Using the option `$ make testS` one can run the ./makesupport/testscanner.sh script on source code to evaluate the Scanner behavior:

./makesupport/testscanner.sh

```
ocamllex ./X/CasperScannerValidation.mll
ocaml ./X/CasperScannerValidation.ml < ./X/CasperScannerValidationInput.txt
rm -f ./X/CasperScannerValidation.ml

ocamllex ./X/CasperScannerValidationWithLinenumbers.mll
ocaml ./X/CasperScannerValidationWithLinenumbers.ml < ./X/CasperScannerValidationInput.txt
rm -f ./X/CasperScannerValidationWithLinenumbers.ml
```

./X/CasperScannerValidationInput.txt

```
/*
    CasperScannerValidationInput.txt
    Michael Makris, mm3443
    PLT Fall 2018

    Purpose: to validate CasperScanner.mll. Removed Parser dependency and added printf statements
to identify current token read.

    Use:
        ocamllex CasperScannerValidation.mll
        ocaml CasperScannerValidation.ml < CasperScannerValidationInput.txt
*/

int gcd(int a, int b) {
   // gcd function
    while (a != b)
    {
        if (a > b)
            a = a - b;
        else
```

```
            b = b -a;
    }
    return a;
}
```

Output

```
138 states, 4502 transitions, table size 18836 bytes
Comments level 0 start
Comments level 0 end
keyword     int
identifier  gcd
operator    (
keyword     int
identifier  a
operator    ,
keyword     int
identifier  b
operator    )
operator    {
Line comment Start
Line comment End
keyword     while
operator    (
identifier  a
operator    !=
identifier  b
operator    )
operator    {
keyword     if
operator    (
identifier  a
operator    >
identifier  b
operator    )
identifier  a
operator    =
identifier  a
operator    -
identifier  b
operator    ;
keyword     else
identifier  b
operator    =
identifier  b
operator    -
identifier  a
operator    ;
operator    }
keyword     return
identifier  a
operator    ;
operator    }
```

6.3.2 Parser Unit Test, ./makesupport/testparser.sh and ./makesupport/menhirTestInput.txt

Similarly, using the option `$ make testP` one can run the ./makesupport/testparser.sh script on tokens to evaluate the Parser behavior:

./makesupport/testparser.sh

```
#!/bin/sh
menhir --interpret --interpret-show-cst parser.mly < ./X/menhirTestInput.txt
menhir --interpret --interpret-show-cst parser.mly < ./X/menhirTestInput.txt | grep -n "REJECT"
```

./makesupport/menhirTestInput.txt

```
INT
FLOAT
BOOL
STRING
VOID
LBRACKET INTLITERAL RBRACKET
LPAREN INT IDENTIFIER COMMA FLOAT IDENTIFIER LBRACKET INTLITERAL RBRACKET RPAREN
INT IDENTIFIER LBRACKET RBRACKET SEMICOLON
INT IDENTIFIER SEMICOLON
FLOAT IDENTIFIER SEMICOLON
STRING IDENTIFIER SEMICOLON
BOOL IDENTIFIER SEMICOLON
VOID IDENTIFIER SEMICOLON
INT IDENTIFIER LBRACKET INTLITERAL RBRACKET SEMICOLON
FLOAT IDENTIFIER LBRACKET INTLITERAL RBRACKET SEMICOLON
STRING IDENTIFIER LBRACKET INTLITERAL RBRACKET SEMICOLON
BOOL IDENTIFIER LBRACKET INTLITERAL RBRACKET SEMICOLON
VOID IDENTIFIER LBRACKET INTLITERAL RBRACKET SEMICOLON
INT IDENTIFIER LPAREN RPAREN LBRACE RBRACE
INT IDENTIFIER LPAREN INT IDENTIFIER COMMA INT IDENTIFIER RPAREN LBRACE RBRACE
VOID IDENTIFIER LPAREN STRING IDENTIFIER COMMA BOOL IDENTIFIER RPAREN LBRACE INTLITERAL SEMICOLON
RBRACE
VOID IDENTIFIER LPAREN STRING IDENTIFIER COMMA BOOL IDENTIFIER RPAREN LBRACE INTLITERAL SEMICOLON
RBRACE
VOID IDENTIFIER LPAREN STRING IDENTIFIER COMMA BOOL IDENTIFIER RPAREN LBRACE INTLITERAL SEMICOLON
FLTLITERAL SEMICOLON RBRACE
BOOL IDENTIFIER LPAREN STRING IDENTIFIER COMMA BOOL IDENTIFIER RPAREN LBRACE INT IDENTIFIER
SEMICOLON FLTLITERAL SEMICOLON RBRACE
BOOL IDENTIFIER LPAREN STRING IDENTIFIER COMMA BOOL IDENTIFIER RPAREN LBRACE INT IDENTIFIER
SEMICOLON BREAK SEMICOLON CONTINUE SEMICOLON RETURN SEMICOLON BOOL IDENTIFIER LBRACKET INTLITERAL
RBRACKET SEMICOLON RBRACE
INT IDENTIFIER LPAREN INT IDENTIFIER COMMA INT IDENTIFIER LBRACKET INTLITERAL RBRACKET RPAREN
LBRACE IF LPAREN NOT BOOLLITERAL RPAREN LBRACE BREAK SEMICOLON RBRACE ELSE LBRACE IF LPAREN
BOOLLITERAL RPAREN LBRACE BREAK SEMICOLON RBRACE ELSE LBRACE RBRACE RETURN SEMICOLON RBRACE RETURN
SEMICOLON RBRACE INT IDENTIFIER LPAREN INT IDENTIFIER COMMA INT IDENTIFIER RPAREN LBRACE IF LPAREN
BOOLLITERAL RPAREN LBRACE BREAK SEMICOLON RBRACE ELSE LBRACE IF LPAREN BOOLLITERAL RPAREN LBRACE
BREAK SEMICOLON RBRACE ELSE LBRACE RBRACE DEC LBRACKET IDENTIFIER PLUS INTLITERAL MINUS FLTLITERAL
CONCAT STRLITERAL RBRACKET SEMICOLON RETURN SEMICOLON RBRACE INC MINUS IDENTIFIER SEMICOLON
IDENTIFIER ASSIGN LPAREN STRLITERAL CHARAT INTLITERAL RPAREN SEMICOLON RETURN IDENTIFIER LPAREN
IDENTIFIER COMMA IDENTIFIER COMMA BOOLLITERAL RPAREN SEMICOLON IDENTIFIER ASSIGN IDENTIFIER
CON_ASSIGN STRLITERAL SEMICOLON FOR LPAREN SEMICOLON BOOLLITERAL SEMICOLON RPAREN LBRACE RBRACE
WHILE LPAREN LPAREN INC IDENTIFIER COMMA MINUS IDENTIFIER RPAREN NEQ IDENTIFIER RPAREN LBRACE
RBRACE INTLITERAL SEMICOLON INT IDENTIFIER LBRACKET INTLITERAL RBRACKET SEMICOLON IDENTIFIER ASSIGN
LBRACKET STRLITERAL COMMA STRLITERAL COMMA STRLITERAL RBRACKET SEMICOLON RBRACE
```

Output

```
REJECT
REJECT
REJECT
REJECT
REJECT
REJECT
REJECT
REJECT
ACCEPT
[casper:
  [declarations:
    [declarations:]
    [casperVariable: [casperType: INT] IDENTIFIER SEMICOLON]
  ]
  EOF
]
ACCEPT
[casper:
  [declarations:
    [declarations:]
    [casperVariable: [casperType: FLOAT] IDENTIFIER SEMICOLON]
  ]
  EOF
]
ACCEPT
[casper:
  [declarations:
    [declarations:]
    [casperVariable: [casperType: STRING] IDENTIFIER SEMICOLON]
  ]
  EOF
]
ACCEPT
[casper:
  [declarations:
    [declarations:]
    [casperVariable: [casperType: BOOL] IDENTIFIER SEMICOLON]
  ]
  EOF
]
ACCEPT
[casper:
  [declarations:
    [declarations:]
    [casperVariable: [casperType: VOID] IDENTIFIER SEMICOLON]
  ]
  EOF
]
REJECT
REJECT
REJECT
REJECT
REJECT
ACCEPT
[casper:
  [declarations:
    [declarations:]
    [casperFunction:
      [casperType: INT]
      IDENTIFIER
      [formalsBlock: LPAREN [optFormals:] RPAREN]
      LBRACE
      [localsBlock:]
      [statementsList:]
      RBRACE
    ]
  ]
  EOF
]
ACCEPT
…
```

7. Lessons Learned

Having done this project on my own, as a CVN student, I was spared the complexities of working within a group project. But I was exposed to different challenges which I can share here.

The first was that I could not easily bounce ideas off others and get their opinion, advice or knowledge. In this project with a new language, OCaml, and paradigm, functional programming, to learn, I would have greatly benefited from interacting extensively with others.

The second challenge was to cover all aspects of the project. This was both hard and very rewarding. On one hand I was not able to build a more extensive and interesting language because I had to take care of everything else. On the other hand, I had to understand and implement every piece of the project because there was no one else to do it. And I think the last part made me appreciate the project and the course a lot more because I got deeper into them in my effort to complete the project.

As an advice, I would say that this project is very time consuming and never ending. Don't leave anything for later because there won't be any time later. Keep the language features to a reasonable level and ask for advice. From the very beginning, think of a specific algorithm you want to implement with your language and make that happen. Listen to every word from Professor Edwards because it will become true and you will probably need to repeat it somewhere in the project or exams.

8. Appendix

Attached here are the code listings of all the modules used in this project.

8.1 Top-level, ./casper.ml

```
(*
    Top-level of the compiler for Casper
    Scan & parse the input,
    check the resulting AST and generate an SAST from it, generate LLVM IR,
    and dump the module
    Based on MicroC
    File: casper.ml
    Michael Makris, mm3443
    PLT Fall 2018
*)

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast),     "Print the AST");
    ("-s", Arg.Unit (set_action Sast),    "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile), "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "Casper version 1.0.\nusage: ./casper.native [-a|-s|-l|-c] [file.goo]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.casper Scanner.token lexbuf in
  match !action with
    Ast -> print_string (Ast.string_of_casperProgram ast)
  | _ -> let sast = Semant.check ast in
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sCasperProgram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
                   Llvm_analysis.assert_valid_module m;
                   print_string (Llvm.string_of_llmodule m)
```

8.2 Scanner, ./scanner.mll

```
(*
    OCamlLex scanner for Casper
    File: scanner.mll
    Michael Makris, mm3443
    PLT Fall 2018
*)

(* header *)
{
   open Parser

   let getChr c =
       let c = (Scanf.unescaped c) in c .[1]
}

(* definitions *)
let whitespace = [' ' '\t']+
let newline = '\n' | '\r' | "\r\n"

let digit = ['0'-'9']
let integer = digit+ (* '-'?digit+ *)
let float = (digit+ '.' digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ '.' ['e' 'E'] ['+' '-']? digit+)
            | ('.' digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ '.' digit+) | (digit+ '.')
            | ('.' digit+)

let letter = ['a'-'z' 'A'-'Z' '_']
let ascii = (('`')(( [' ' - '~' ] ) as mychar)('`'))
let id = letter(letter|digit)*

let badbraces = (['{'](whitespace | newline | '{')*['{']) | (['}'](whitespace | newline)*['{'])
let badbrackets = (['['](whitespace | newline | '[')*['['])
                  | ([']'](whitespace | newline | ']')*[']']) | ([']'](whitespace | newline)*['['])
let badend = [';'](whitespace | newline)*['{'] | ['}'](whitespace | newline)*[';']
let badcommas = [','](whitespace | newline)*[',']
let badsemis =  [';'](whitespace | newline)*[';']
let badcomments = "*/" | "**"
let badcombs = badbraces | badbrackets | badend | badcommas | badsemis | badcomments

(* rules *)
rule token = parse
  (* whitespace *)
    whitespace { token lexbuf } | newline { token lexbuf }

  (* comments *)
  | "//" { linecomment lexbuf } | "/*" { blockcomment 0 lexbuf }

  (* blocks, delimiters, terminators *)
  | '(' { LPAREN }    | ')' { RPAREN }
  | '[' { LBRACKET }  | ']' { RBRACKET }
  | '{' { LBRACE }    | '}' { RBRACE }
  | ';' { SEMICOLON } | ',' { COMMA }

  (* string operators *)
  | '_' { CONCAT }    | '?' { CHARAT }

  (* inc/decrement operators *)
  | "++" { INC }  | "--" { DEC }
  (* cast operators *)
  | "~" { ITOF }

  (* arithmetic operators *)
  | '+' { PLUS }      | '-' { MINUS }
  | '*' { TIMES }     | '/' { DIVIDE }
  | '%' { MODULUS }   | '^' { EXPONENT }

  (* relational operators *)
  | '>'  { GT }   | ">=" { GTE }
  | '<'  { LT }   | "<=" { LTE }
  | "==" { EQ }   | "!=" { NEQ }

  (* assignment operators *)
  | '='  { ASSIGN } | "_=" { CON_ASSIGN } | "+=" { ADD_ASSIGN } | "-=" { SUB_ASSIGN }

  (* logical operators *)
  | "&&" { AND } | "||" { OR } | "!" { NOT }

  (* conditional keywords *)
  | "if" { IF } | "else" { ELSE }

  (* loop keywords *)
  | "for" { FOR } | "while" { WHILE } | "do" { DO } | "until" { UNTIL }
  | "break" { BREAK } | "continue" { CONTINUE }

  (* function keywords *)
  | "return" { RETURN }

  (* type keywords *)
  | "int" { INT }     | "float" { FLOAT } | "str" { STRING }
  | "chr" { CHAR }  | "bool"  { BOOL }  | "void" { VOID }

  (* literals *)
  | integer as lexeme { INTLITERAL(int_of_string lexeme) }
  | float as lexeme   { FLTLITERAL(lexeme) }
  | '''               { STRLITERAL (stringSQ (Buffer.create 100) lexbuf) }
  | '"'               { STRLITERAL (stringDQ (Buffer.create 100) lexbuf) }
  | ascii             { CHRLITERAL(mychar) }
  | "true"            { BOOLLITERAL (true) }
  | "false"           { BOOLLITERAL (false) }
  | "null"            { NULL }

  (* indentifier *)
  |  id as lexeme     { IDENTIFIER(lexeme) }

  (* end of file *)
  | eof               { EOF }

  (* errors *)
  | badcombs as lexeme { raise (Failure("Bad syntax: " ^ lexeme)) }
  | _ as character     { raise (Failure("Bad character: " ^ Char.escaped character)) }

and stringSQ tempbuffer = parse
    '''                 { Buffer.contents tempbuffer }
  | newline     { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringSQ tempbuffer lexbuf }
  | [^ '''  '\n' '\r']+ { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringSQ tempbuffer
                           lexbuf }
  | eof                 { raise (Failure("Non-terminated single quotes")) }

and stringDQ tempbuffer = parse
    '"'                 { Buffer.contents tempbuffer }
  | newline             { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringDQ tempbuffer
                           lexbuf }
  | [^ '"'  '\n' '\r']+ { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringDQ tempbuffer
                           lexbuf }
  | eof                 { raise (Failure("Non-terminated double quotes")) }

and linecomment = parse
    newline { token lexbuf }
  | eof     { EOF }
  | _       { linecomment lexbuf }

and blockcomment level = parse
    "*/"    { if level = 0 then token lexbuf else blockcomment (level-1) lexbuf }
  | "/*"    { blockcomment (level+1) lexbuf }
  | newline { blockcomment level lexbuf }
  | eof     { raise (Failure("Non-terminated comments")) }
  | _       { blockcomment level lexbuf }

(* trailer *)
```

8.3 Parser, ./parser.mly

```
/*  OCamlYacc parser for Casper
    Based on MicroC
    File: parser.mly
    Michael Makris, mm3443
    PLT Fall 2018
*/
%{ open Ast %}

/* blocks, delimiters, terminators */
%token LPAREN RPAREN LBRACKET RBRACKET LBRACE RBRACE SEMICOLON COMMA EOF
/* string operators */
%token CONCAT CHARAT
/* arithmetic operators */
%token PLUS MINUS TIMES DIVIDE MODULUS EXPONENT
/* relational operators */
%token GT GTE LT LTE EQ NEQ
/* inc/decrement operators */
%token INC DEC
/* assignment operators */
%token ASSIGN CON_ASSIGN ADD_ASSIGN SUB_ASSIGN
/* cast operators */
%token ITOF
/* logical operators */
%token AND OR NOT
/* conditional keywords */
%token IF ELSE
/* loop keywords */
%token FOR WHILE DO UNTIL BREAK CONTINUE
/* function keywords */
%token RETURN
/* type keywords */
%token INT FLOAT STRING CHAR BOOL VOID
/* null keyword */
%token NULL

/* literals */
%token <int> INTLITERAL
%token <string> FLTLITERAL
%token <string> STRLITERAL
%token <char> CHRLITERAL
%token <bool> BOOLLITERAL
%token VOIDLITERAL

/* identifier for variable and function names */
%token <string> IDENTIFIER

/* precedence */
%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN CON_ASSIGN ADD_ASSIGN SUB_ASSIGN
%left OR
%left AND
%left EQ NEQ
%left GT GTE LT LTE
%left CONCAT
%left PLUS MINUS
%left TIMES DIVIDE MODULUS
%right EXPONENT
%right CHARAT
%right ITOF
%right INC DEC
%right NOT NEG
```

```
%start casper
%type <Ast.casperProgram> casper

%%

/****** program declaration ******/
casper: declarations EOF { $1 }

declarations:
    /* epsilon */               { ([], []) }
  | declarations casperVariable { (($2 :: fst $1), snd $1) }
  | declarations casperFunction { (fst $1, ($2 :: snd $1)) }

/****** variable declaration ******/
/* variable declaration: <type> <variable_name>; */
casperVariable: casperType IDENTIFIER SEMICOLON {$1, $2}

/****** type declaration ******/
casperType:
    INT     { Int }
  | FLOAT   { Float }
  | STRING  { String }
  | CHAR    { Char }
  | BOOL    { Bool }
  | VOID    { Void }

/****** function declaration ******/
/* <type> <function_name> (<arg1>, <arg2>, ...) { <statements> } */
casperFunction: casperType IDENTIFIER formalsBlock
                LBRACE localsBlock statementsList RBRACE {{
                    functionType       = $1;
                    functionName       = $2;
                    functionFormals    = List.rev $3;
                    functionLocals     = List.rev $5;
                    functionStatements = List.rev $6;
                }}

/* formals Block*/
formalsBlock: LPAREN optFormals RPAREN          {$2}

/* optional formals list */
optFormals: /* epsilon */  { [] }
            | formalsList { $1 }

/* formals list with elements */
formalsList: casperType IDENTIFIER                      { [($1, $2)] }
             | formalsList COMMA casperType IDENTIFIER  { ($3,$4) :: $1 }

/* optional locals list */
localsBlock:  /* epsilon */              { [] }
            | localsBlock casperVariable { $2::$1 }

/* optional statement list */
statementsList: /* epsilon */                   { [] }
                | statementsList casperStatement { $2::$1 }

casperStatement:
  /* { <statement>; ... } */
    LBRACE statementsList RBRACE            { StatementBlock(List.rev $2) }
  /* <expression>; */
  | casperExpression SEMICOLON              { Expression $1 }
  /* if (<expression>) { <statements> } */
  | IF LPAREN casperExpression RPAREN casperStatement %prec NOELSE
```

```
                                            { IfCondition($3, $5, StatementBlock([])) }
  /* if (<expression>) { <statements> } else { <statements> } */
  | IF LPAREN casperExpression RPAREN casperStatement
    ELSE casperStatement                    { IfCondition($3, $5, $7) }
  /* for (<expression>; <expression>; <expression>) { <statements> } */
  | FOR LPAREN optExpression SEMICOLON casperExpression SEMICOLON optExpression RPAREN
    casperStatement                         { ForLoop($3, $5, $7, $9) }
  /* while (<expression>) { <statements> } */
  | WHILE LPAREN casperExpression RPAREN casperStatement
                                            { WhileLoop($3, $5) }
  /* do { <statements> } until (<expression>);*/
  | DO casperStatement UNTIL LPAREN casperExpression RPAREN SEMICOLON
                                            { DoUntilLoop($2, $5) }
  /* do { <statements> } while (<expression>);*/
  | DO casperStatement WHILE LPAREN casperExpression RPAREN SEMICOLON
                                            { DoWhileLoop($2, $5) }
  /* break; */
  | BREAK SEMICOLON                         { Break }
  /* continue; */
  | CONTINUE SEMICOLON                      { Continue }
  /* return <expression>; */
  | RETURN optExpression SEMICOLON          { Return $2 }

/****** expression declaration ******/
/* optional expression */
optExpression: { Epsilon }
               | casperExpression { $1 }

casperExpression:
    INTLITERAL                                   { IntLIT  ($1) }
  | FLTLITERAL                                   { FltLIT  ($1) }
  | STRLITERAL                                   { StrLIT  ($1) }
  | CHRLITERAL                                   { ChrLIT  ($1) }
  | BOOLLITERAL                                  { BoolLIT ($1) }
  | VOIDLITERAL                                  { VoidLIT }
  | NULL                                         { NullLIT }
  | IDENTIFIER                                   { Identifier ($1) }
  | casperExpression  PLUS      casperExpression { BinOP($1, Add, $3) }
  | casperExpression  MINUS     casperExpression { BinOP($1, Sub, $3) }
  | casperExpression  TIMES     casperExpression { BinOP($1, Mul, $3) }
  | casperExpression  DIVIDE    casperExpression { BinOP($1, Div, $3) }
  | casperExpression  MODULUS   casperExpression { BinOP($1, Mod, $3) }
  | casperExpression  EXPONENT  casperExpression { BinOP($1, Exp, $3) }
  | casperExpression  CONCAT    casperExpression { BinOP($1, Con, $3) }
  | casperExpression  CHARAT    casperExpression { BinOP($1, Cat, $3) }
  | casperExpression  GT        casperExpression { BinOP($1, Grt, $3) }
  | casperExpression  GTE       casperExpression { BinOP($1, Gre, $3) }
  | casperExpression  LT        casperExpression { BinOP($1, Lst, $3) }
  | casperExpression  LTE       casperExpression { BinOP($1, Lse, $3) }
  | casperExpression  EQ        casperExpression { BinOP($1, Eql, $3) }
  | casperExpression  NEQ       casperExpression { BinOP($1, Neq, $3) }
  | casperExpression  AND       casperExpression { BinOP($1, And, $3) }
  | casperExpression  OR        casperExpression { BinOP($1, Or , $3) }
  | INC casperExpression                         { BinOP($2, Add, IntLIT(1)) }
  | DEC casperExpression                         { BinOP($2, Sub, IntLIT(1)) }
  | NOT casperExpression                         { UnrOP(Not, $2) }
  | MINUS casperExpression %prec NEG             { UnrOP(Neg, $2) }
  | ITOF casperExpression                        { UnrOP(ItoF, $2) }
  | IDENTIFIER  ASSIGN      casperExpression     { AsgnOP($1, Asgn, $3) }
  | IDENTIFIER  CON_ASSIGN  casperExpression     { AsgnOP($1, ConAsgn, $3) }
  | IDENTIFIER  ADD_ASSIGN  casperExpression     { AsgnOP($1, AddAsgn, $3) }
  | IDENTIFIER  SUB_ASSIGN  casperExpression     { AsgnOP($1, SubAsgn, $3) }
  | IDENTIFIER LPAREN elements RPAREN            { FunctionCall($1, $3) }
  | LPAREN casperExpression RPAREN               { $2 }
```

```
/* optional element list */
elements: /* epsilon */   { [] }
          | elementsList  { List.rev $1 }

/* argument list with elements */
elementsList: casperExpression                      { [$1] }
              | elementsList COMMA casperExpression { $3::$1 }
```

8.4 AST, ./ast.ml

```
(*
    Abstract Syntax Tree for Casper
    with functions for printing it
    Based on MicroC
    File: ast.ml
    Michael Makris, mm3443
    PLT Fall 2018
*)

type casperBinaryOperator =
      Add | Sub | Mul | Div | Mod | Exp
    | Con | Cat
    | Grt | Gre | Lst | Lse | Eql | Neq
    | And | Or

type casperUnaryOperator = Not | Neg | ItoF

type casperAssignment = Asgn | ConAsgn | AddAsgn | SubAsgn

type casperExpression =
      Epsilon
    | IntLIT of int
    | FltLIT of string
    | StrLIT of string
    | ChrLIT of char
    | BoolLIT of bool
    | VoidLIT
    | NullLIT
    | Identifier of string
    | BinOP of casperExpression * casperBinaryOperator * casperExpression
    | UnrOP of casperUnaryOperator * casperExpression
    | AsgnOP of string * casperAssignment * casperExpression
    | FunctionCall of string * casperExpression list

type casperStatement =
      StatementBlock of casperStatement list
    | Expression of casperExpression
    | IfCondition of casperExpression * casperStatement * casperStatement
    | ForLoop of casperExpression * casperExpression * casperExpression * casperStatement
    | WhileLoop of casperExpression * casperStatement
    | DoUntilLoop of casperStatement * casperExpression
    | DoWhileLoop of casperStatement * casperExpression
    | Break
    | Continue
    | Return of casperExpression

type casperType = Int | Float| String | Char | Bool | Void

type casperVariable = casperType * string

type casperFunction = {
    functionType       : casperType;
    functionName       : string;
    functionFormals    : casperVariable list;
    functionLocals     : casperVariable list;
    functionStatements : casperStatement list;
}

type casperProgram = casperVariable list * casperFunction list
```

```
(* Pretty-printing functions *)
let string_of_casperBinaryOperator = function
      Add -> "+"
    | Sub -> "-"
    | Mul -> "*"
    | Div -> "/"
    | Mod -> "%"
    | Exp -> "^"
    | Con -> "_"
    | Cat -> "?"
    | Grt -> ">"
    | Gre -> ">="
    | Lst -> "<"
    | Lse -> "<="
    | Eql -> "=="
    | Neq -> "!="
    | And -> "&&"
    | Or  -> "||"

let string_of_casperUnaryOperator = function
      Not -> "!"
    | Neg -> "-"
    | ItoF -> "~"

let string_of_casperAssignment = function
      Asgn     -> "="
    | ConAsgn  -> "_="
    | AddAsgn  -> "+="
    | SubAsgn  -> "-="

let rec string_of_casperExpression = function
      Epsilon -> ""
    | IntLIT(i) -> string_of_int i
    | FltLIT(s) -> s
    | StrLIT(s) -> s
    | ChrLIT(_) -> "" (* TODO fix Core module to print char c*)
    | BoolLIT(true) -> "true"
    | BoolLIT(false) -> "false"
    | VoidLIT -> ""
    | NullLIT -> ""
    | Identifier(s) -> s
    | BinOP(e1, o, e2) -> string_of_casperExpression e1 ^ " " ^ string_of_casperBinaryOperator o ^
                          " " ^ string_of_casperExpression e2
    | UnrOP(o, e) -> string_of_casperUnaryOperator o ^ string_of_casperExpression e
    | AsgnOP(s, o, e) -> s ^ " " ^ string_of_casperAssignment o ^ " " ^
                          string_of_casperExpression e
    | FunctionCall(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_casperExpression el)
                              ^ ")"

let string_of_casperType = function
      Int    -> "int"
    | Float  -> "float"
    | String -> "str"
    | Char   -> "chr"
    | Bool   -> "bool"
    | Void   -> "void"

let string_of_casperVariable (variableType, variableName) = string_of_casperType variableType ^
                                                            " " ^ variableName
```

```
let rec string_of_casperStatement = function
      StatementBlock (s) -> "{\n" ^ String.concat "" (List.map string_of_casperStatement s) ^ "}\n"
    | Expression(e) -> string_of_casperExpression e ^ ";\n";
    | IfCondition(e, s, StatementBlock([])) -> "if (" ^ string_of_casperExpression e ^ ") " ^
                                              string_of_casperStatement s
    | IfCondition(e, s1, s2) ->  "if (" ^ string_of_casperExpression e ^ ")\n" ^
                                         string_of_casperStatement s1
                                 ^ "else \n" ^ string_of_casperStatement s2
    | ForLoop(e1, e2, e3, s) -> "for (" ^ string_of_casperExpression e1 ^ "; " ^
                                         string_of_casperExpression e2 ^ "; " ^
                                         string_of_casperExpression e3 ^ ") " ^
                                         string_of_casperStatement s
    | WhileLoop(e, s) -> "while (" ^ string_of_casperExpression e ^ ") " ^
                                    string_of_casperStatement s
    | DoUntilLoop(s, e) -> "do " ^ string_of_casperStatement s ^
                           "    until (" ^ string_of_casperExpression e ^ ");\n"
    | DoWhileLoop(s, e) -> "do " ^ string_of_casperStatement s ^
                           "    while (" ^ string_of_casperExpression e ^ ");\n"
    | Break     -> "break;\n";
    | Continue  -> "continue;\n";
    | Return(e) -> "return " ^ string_of_casperExpression e ^ ";\n"

let string_of_casperFunction casperFunction =
    string_of_casperType casperFunction.functionType ^ " " ^ casperFunction.functionName ^
    "(" ^ String.concat ", " (List.map string_of_casperVariable casperFunction.functionFormals) ^
    ") {\n" ^
    String.concat ", " (List.map string_of_casperVariable casperFunction.functionLocals) ^
    String.concat "" (List.map string_of_casperStatement casperFunction.functionStatements) ^ "}\n"

let string_of_casperProgram (programGlobals, programFunctions) =
    String.concat ";\n" (List.map string_of_casperVariable programGlobals) ^ "\n" ^
    String.concat "\n" (List.map string_of_casperFunction programFunctions)

(* END Pretty-printing functions*)
```

8.5 SAST, ./sast.ml

```
(*
    Semantically-checked Abstract Syntax Tree for Casper
    with functions for printing it
    Based on MicroC
    File: sast.ml
    Michael Makris, mm3443
    PLT Fall 2018
*)

open Ast

type sCasperExpression = casperType * sx
and sx =
      SEpsilon
    | SIntLIT of int
    | SFltLIT of string
    | SStrLIT of string
    | SChrLIT of char
    | SBoolLIT of bool
    | SVoidLIT
    | SNullLIT
    | SIdentifier of string
    | SBinOP of sCasperExpression * casperBinaryOperator * sCasperExpression
    | SUnrOP of casperUnaryOperator * sCasperExpression
    | SAsgnOP of string * casperAssignment * sCasperExpression
    | SFunctionCall of string * sCasperExpression list

type sCasperStatement =
      SStatementBlock of sCasperStatement list
    | SExpression of sCasperExpression
    | SIfCondition of sCasperExpression * sCasperStatement * sCasperStatement
    | SForLoop of sCasperExpression * sCasperExpression * sCasperExpression * sCasperStatement
    | SWhileLoop of sCasperExpression * sCasperStatement
    | SDoUntilLoop of sCasperStatement * sCasperExpression
    | SDoWhileLoop of sCasperStatement * sCasperExpression
    | SBreak
    | SContinue
    | SReturn of sCasperExpression

type sCasperFunction = {
    sFunctionType       : casperType;
    sFunctionName       : string;
    sFunctionFormals    : casperVariable list;
    sFunctionLocals     : casperVariable list;
    sFunctionStatements : sCasperStatement list;
  }

type sCasperProgram = casperVariable list * sCasperFunction list
```

```
(* Pretty-printing functions *)

let rec string_of_sCasperExpression (t, e) =
  "(" ^ string_of_casperType t ^ " : " ^ (match e with
      SEpsilon -> ""
    | SIntLIT(i) -> string_of_int i
    | SFltLIT(s) -> s
    | SStrLIT(s) -> s
    | SChrLIT(_) -> "" (* TODO fix Core module to print char c*)
    | SBoolLIT(true) -> "true"
    | SBoolLIT(false) -> "false"
    | SVoidLIT -> ""
    | SNullLIT -> ""
    | SIdentifier(s) -> s
    | SBinOP(e1, o, e2) -> string_of_sCasperExpression e1 ^ " " ^ string_of_casperBinaryOperator o
                   ^ " " ^ string_of_sCasperExpression e2
    | SUnrOP(o, e) -> string_of_casperUnaryOperator o ^ string_of_sCasperExpression e
    | SAsgnOP(s, o, e) -> s ^ " " ^ string_of_casperAssignment o ^ " " ^
                                    string_of_sCasperExpression e
    | SFunctionCall(f, el) -> f ^ "(" ^
                              String.concat ", " (List.map string_of_sCasperExpression el) ^ ")"
) ^ ")"

let rec string_of_sCasperStatement = function
      SStatementBlock (s) -> "{\n" ^ String.concat "" (List.map string_of_sCasperStatement s) ^
                              "}\n"
    | SExpression(e) -> string_of_sCasperExpression e ^ ";\n";
    | SIfCondition(e, s, SStatementBlock([])) -> "if (" ^ string_of_sCasperExpression e ^ ") " ^
                                                          string_of_sCasperStatement s
    | SIfCondition(e, s1, s2) ->  "if (" ^ string_of_sCasperExpression e ^ ")\n" ^
                                           string_of_sCasperStatement s1 ^
                                  "else \n" ^ string_of_sCasperStatement s2
    | SForLoop(e1, e2, e3, s) -> "for (" ^ string_of_sCasperExpression e1 ^ "; " ^
                                           string_of_sCasperExpression e2 ^ "; " ^
                                           string_of_sCasperExpression e3 ^ ")" ^
                                           string_of_sCasperStatement s
    | SWhileLoop(e, s) -> "while (" ^ string_of_sCasperExpression e ^ ") " ^
                                      string_of_sCasperStatement s
    | SDoUntilLoop(s, e) -> "do " ^ string_of_sCasperStatement s  ^
                            "    until (" ^ string_of_sCasperExpression e ^ ");\n"
    | SDoWhileLoop(s, e) -> "do " ^ string_of_sCasperStatement s ^
                            "    while (" ^ string_of_sCasperExpression e ^ ");\n"
    | SBreak     -> "break;\n";
    | SContinue  -> "continue;\n";
    | SReturn(e) -> "return " ^ string_of_sCasperExpression e ^ ";\n"

let string_of_sCasperFunction casperFunction =
    string_of_casperType casperFunction.sFunctionType ^ " " ^ casperFunction.sFunctionName ^
    "(" ^ String.concat ", " (List.map string_of_casperVariable casperFunction.sFunctionFormals) ^
    ") {\n" ^
       String.concat "" (List.map string_of_casperVariable casperFunction.sFunctionLocals) ^
       String.concat "" (List.map string_of_sCasperStatement casperFunction.sFunctionStatements) ^
    "}\n"

let string_of_sCasperProgram (programGlobals, programFunctions) =
    String.concat ";\n" (List.map string_of_casperVariable programGlobals) ^ "\n" ^
    String.concat "\n" (List.map string_of_sCasperFunction programFunctions)

(* END Pretty-printing functions*)
```

8.6 Semantic Checker, ./semant.ml

```
(*
    Semantic checking for Casper
    Based on MicroC
    File: semant.ml
    Michael Makris, mm3443
    PLT Fall 2018
*)

open Ast
open Sast

module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions) =

  (* Verify a list of bindings has no void types or duplicate names *)
  let check_binds (kind : string) (binds : casperVariable list) =
    List.iter (function
    (Void, b) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
      | _ -> ()) binds;
    let rec dups = function
        [] -> ()
      | ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
    raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in

  (**** Check global variables ****)

  check_binds "global" globals;

  (**** Check functions ****)

  (* Collect function declarations for built-in functions: no bodies *)
  let built_in_decls =
    let add_bind map (name, ty, fty) = StringMap.add name {
      functionType = fty;
      functionName = name;
      functionFormals = [(ty, "x")];
      functionLocals = []; functionStatements = [] } map
    in List.fold_left add_bind StringMap.empty [ ("printi", Int, Void);
    ("printinl", Int, Void); ("printfnl", Float, Void); ("printf", Float, Void);
    ("prints", String, Void); ("printsnl", String, Void); ("printc", Char, Void);
    ("printcnl", Char, Void);("printbig", Int, Void);
    ("printbasi", Bool, Void); ("printb", Bool, Void); ("printbnl", Bool, Void);
    ("sin", Float, Float); ("cos", Float, Float); ("tan", Float, Float);
    ("asin", Float, Float); ("acos", Float, Float); ("atan", Float, Float);
    ("sinh", Float, Float); ("cosh", Float, Float); ("tanh", Float, Float);
    ("exp", Float, Float); ("log", Float, Float); ("log10", Float, Float);
    ("sqrt", Float, Float); ("floor", Float, Float); ("ceil", Float, Float);
    ("abs", Float, Float); ("srand", Int, Void); ("strlen", String, Int);
    ("strerror", Int, String); ("isalnum", Char, Int); ("isalpha", Char, Int);
    ("iscntrl", Char, Int); ("isdigit", Char, Int); ("isgraph", Char, Int);
    ("islower", Char, Int); ("isprint", Char, Int); ("ispunct", Char, Int);
    ("isspace", Char, Int); ("isupper", Char, Int); ("isxdigit", Char, Int); ]
  in

  let built_in_decls =
    let add_bind map (name, fty) = StringMap.add name {
      functionType = fty;
      functionName = name;
      functionFormals = [];
      functionLocals = []; functionStatements = [] } map
    in List.fold_left add_bind built_in_decls [ ("rand", Int)]
  in

  let built_in_decls =
    let add_bind map (name, ty1, ty2, fty) = StringMap.add name {
      functionType = fty;
      functionName = name;
      functionFormals = [(ty1, "x"); (ty2, "x")];
      functionLocals = []; functionStatements = [] } map
    in List.fold_left add_bind built_in_decls [ ("fmod", Float, Float, Float);
       ("strcpy", String, String, String); ("strcat", String, String, String);
       ("strcmp", String, String, Int); ("strchr", String, String, String);
       ("strrchr", String, String, String); ("strspn", String, String, Int);
       ("strcspn", String, String, Int); ("strpbrk", String, String, String);
       ("strstr", String, String, String); ("strtok", String, String, String)]
  in

  let built_in_decls =
    let add_bind map (name, ty1, ty2, ty3, fty) = StringMap.add name {
      functionType = fty;
      functionName = name;
      functionFormals = [(ty1, "x"); (ty2, "x"); (ty3, "x")];
      functionLocals = []; functionStatements = [] } map
    in List.fold_left add_bind built_in_decls [
      ("strncpy", String, String, Int, String);
      ("strncat", String, String, Int, String);
      ("strncmp", String, String, Int, Int) ]
  in

  (* Add function name to symbol table *)
  let add_func map fd =
    let built_in_err = "function " ^ fd.functionName ^ " may not be defined"
    and dup_err = "duplicate function " ^ fd.functionName
    and make_err er = raise (Failure er)
    and n = fd.functionName (* Name of the function *)
    in match fd with (* No duplicate functions or redefinitions of built-ins *)
         _ when StringMap.mem n built_in_decls -> make_err built_in_err
       | _ when StringMap.mem n map -> make_err dup_err
       | _ ->  StringMap.add n fd map
  in

  (* Collect all function names into one symbol table *)
  let function_decls = List.fold_left add_func built_in_decls functions
  in

  (* Return a function from our symbol table *)
  let find_func s =
    try StringMap.find s function_decls
    with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in

  let _ = find_func "main" in (* Ensure "main" is defined *)
  let check_function func =
    (* Make sure no formals or locals are void or duplicates *)
    check_binds "formal" func.functionFormals;
    check_binds "local" func.functionLocals;

    (* Raise an exception if the given rvalue type cannot be assigned to the given lvalue type *)
    let check_assign lvaluet rvaluet err =
       if lvaluet = rvaluet then lvaluet else raise (Failure err)
    in

    (* Build local symbol table of variables for this function *)
    let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
                StringMap.empty (globals @ func.functionFormals @ func.functionLocals )
    in

    (* Return a variable from our local symbol table *)
    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
    in

    (* Return a semantically-checked expression, i.e., with a type *)
    let rec casperExpression = function
        Epsilon       -> (Void, SEpsilon)
      | IntLIT i      -> (Int, SIntLIT i)
      | FltLIT f      -> (Float, SFltLIT f)
      | StrLIT s      -> (String, SStrLIT s)
      | ChrLIT c      -> (Char, SChrLIT c)
      | BoolLIT b     -> (Bool, SBoolLIT b)
      | VoidLIT       -> (Void, SVoidLIT)
      | NullLIT       -> (Void, SNullLIT)
      | Identifier s  -> (type_of_identifier s, SIdentifier s)
      | BinOP(e1, op, e2) as e ->
          let (t1, e1') = casperExpression e1
          and (t2, e2') = casperExpression e2 in
          (* All binary operators require operands of the same type *)
          let same = t1 = t2 in
          (* Determine expression type based on operator and operand types *)
          let ty = match op with
            Add | Sub | Mul | Div | Mod when same && t1 = Int -> Int
          | Add | Sub | Mul | Div | Exp | Mod when same && t1 = Float -> Float
          | Con when same && t1 = String -> String
          | Con when t1 = String && t2 = Int -> String
          | Con when t1 = String && t2 = Float -> String
          | Con when t1 = String && t2 = Bool -> String
          | Cat when t1 = String && t2 = Int -> String
          | Eql | Neq when same -> Bool
          | Grt | Gre | Lst | Lse when same && (t1 = Int || t1 = Float || t1 = String) -> Bool
          | And | Or when same && t1 = Bool -> Bool
          | _ -> raise (Failure ("illegal binary operator " ^ string_of_casperType t1 ^ " " ^
                        string_of_casperBinaryOperator op ^ " " ^ string_of_casperType t2 ^ " in "
                        ^ string_of_casperExpression e))
          in (ty, SBinOP((t1, e1'), op, (t2, e2')))
      | UnrOP(op, e) as ex ->
          let (t, e') = casperExpression e in
          let ty = match op with
            Neg when t = Int || t = Float -> t
          | Not when t = Bool -> Bool
          | ItoF when t = Int -> Float
          | ItoF when t = Float -> Int
          | _ -> raise (Failure ("illegal unary operator " ^ string_of_casperUnaryOperator op ^
                                 string_of_casperType t ^ " in " ^ string_of_casperExpression ex))
          in (ty, SUnrOP(op, (t, e')))
      | AsgnOP(var, op, e) as ex ->
          let lt = type_of_identifier var
          and (rt, e') = casperExpression e in
          let same = lt = rt in
          let ty = match op with
            ConAsgn when same && lt = String -> String
          | ConAsgn when lt = String && rt = Int -> String
          | ConAsgn when lt = String && rt = Float -> String
          | ConAsgn when lt = String && rt = Bool -> String
          | Asgn when same -> lt
          | AddAsgn when same && (lt = Int || lt = Float) -> lt
          | SubAsgn when same && (lt = Int || lt = Float) -> lt
          | _ -> raise (Failure ("illegal assignment " ^ string_of_casperType lt ^
                                  string_of_casperAssignment op ^ string_of_casperType rt ^
                                  " in " ^ string_of_casperExpression ex))
          in (ty, SAsgnOP(var, op, (rt, e')))
      | FunctionCall(fname, args) as call ->
          let fd = find_func fname in
          let param_length = List.length fd.functionFormals in
          if List.length args != param_length then
            raise (Failure ("expecting " ^ string_of_int param_length ^ " arguments in " ^
                            string_of_casperExpression call))
          else let check_call (ft, _) e =
            let (et, e') = casperExpression e in
            let err = "illegal argument found " ^ string_of_casperType et ^ " expected " ^
                      string_of_casperType ft ^ " in " ^ string_of_casperExpression e
            in (check_assign ft et err, e')
          in
          let args' = List.map2 check_call fd.functionFormals args
          in (fd.functionType, SFunctionCall(fname, args'))
    in

    let check_bool_expr e =
      let (t', e') = casperExpression e
      and err = "expected Boolean expression in " ^ string_of_casperExpression e
      in if t' != Bool then raise (Failure err) else (t', e')
    in

    (* Return a semantically-checked statement i.e. containing sexprs *)
    let rec check_stmt = function
        Expression e -> SExpression (casperExpression e)
      | IfCondition(p, b1, b2) -> SIfCondition(check_bool_expr p, check_stmt b1, check_stmt b2)
      | ForLoop(e1, e2, e3, st) -> SForLoop(casperExpression e1, check_bool_expr e2,
                                     casperExpression e3, check_stmt st)
      | WhileLoop(p, s) -> SWhileLoop(check_bool_expr p, check_stmt s)
      | DoUntilLoop(s, p) -> SDoUntilLoop(check_stmt s, check_bool_expr p)
      | DoWhileLoop(s, p) -> SDoWhileLoop(check_stmt s, check_bool_expr p)
      | Break -> SBreak
      | Continue -> SContinue
      | Return e -> let lt = func.functionType and (rt, e') = casperExpression e in
            let err = "return gives " ^ string_of_casperType rt ^ " expected " ^
                      string_of_casperType func.functionType ^ " in " ^
                      string_of_casperExpression e
            in (SReturn (check_assign lt rt err, e') )
        (* A block is correct if each statement is correct and nothing
        follows any Return statement.  Nested blocks are flattened. *)
      | StatementBlock sl ->
          let rec check_stmt_list = function
              [Break as s] -> [check_stmt s]
            | Break :: _   -> raise (Failure "nothing may follow a break")
            | [Continue as s] -> [check_stmt s]
            | Continue :: _   -> raise (Failure "nothing may follow a continue")
            | [Return _ as s] -> [check_stmt s]
            | Return _ :: _   -> raise (Failure "nothing may follow a return")
            | StatementBlock sl :: ss  -> check_stmt_list (sl @ ss) (* Flatten blocks *)
            | s :: ss         -> check_stmt s :: check_stmt_list ss
            | []              -> []
          in SStatementBlock(check_stmt_list sl)

    in (* body of check_function *)
    { sFunctionType = func.functionType;
      sFunctionName = func.functionName;
      sFunctionFormals = func.functionFormals;
      sFunctionLocals  = func.functionLocals;
      sFunctionStatements = match check_stmt (StatementBlock func.functionStatements) with
      SStatementBlock(sl) -> sl
      | _ -> raise (Failure ("internal error: block didn't become a block?"))
    }
  in (globals, List.map check_function functions)
```

8.7 Code Generator, ./codegen.ml

```
(*
    Code generation for Casper
    Translation takes a semantically checked AST and produces LLVM IR
    Based on MicroC
    File: codegen.ml
    Michael Makris, mm3443
    PLT Fall 2018
*)
(*
LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

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

*)

module C = Char
module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

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

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

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and float_t    = L.double_type context
  and str_t      = L.pointer_type (L.i8_type context)
  and i1_t       = L.i1_type     context
  and void_t     = L.void_type   context
  and cnull      = L.const_null (L.i32_type context)
  in
```

```
(* Return the LLVM type for a Casper type *)
let ltype_of_casperType = function
    A.Int    -> i32_t
  | A.Float  -> float_t
  | A.String -> str_t
  | A.Char   -> i8_t
  | A.Bool   -> i1_t
  | A.Void   -> void_t
in

(* pointer to while loop merge_bb, pred_bb for break and continue statement branch *)
(* TODO: expand to list of pointers for nested loops *)
let break_block = ref (cnull) in
let continue_block = ref (cnull) in

(* Create a map of global variables after creating each *)
let global_vars : L.llvalue StringMap.t =
  let global_var m (t, n) =
    if (ltype_of_casperType t = str_t)
    then (
      let init = L.const_null str_t in
      StringMap.add n (L.define_global n init the_module) m
    )
    else (
      let init = match t with
        A.Float -> L.const_float (ltype_of_casperType t) 0.0
        | _ -> L.const_int (ltype_of_casperType t) 0
        in StringMap.add n (L.define_global n init the_module) m
    ) in
  List.fold_left global_var StringMap.empty globals in

let printf_t : L.lltype = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func : L.llvalue = L.declare_function "printf" printf_t the_module in
let printb_t : L.lltype = L.function_type i32_t [| i1_t; i32_t |] in
let printb_func : L.llvalue = L.declare_function "printb" printb_t the_module in
let printbig_t : L.lltype = L.function_type i32_t [| i32_t |] in
let printbig_func : L.llvalue = L.declare_function "printbig" printbig_t the_module in

(* expose string.h strcat function for the _ operator *)
let concats_t : L.lltype = L.function_type str_t [| str_t; str_t |] in
let concats_func : L.llvalue = L.declare_function "concats" concats_t the_module in
let concati_t : L.lltype = L.function_type str_t [| str_t; i32_t |] in
let concati_func : L.llvalue = L.declare_function "concati" concati_t the_module in
let concatf_t : L.lltype = L.function_type str_t [| str_t; float_t |] in
let concatf_func : L.llvalue = L.declare_function "concatf" concatf_t the_module in
let concatb_t : L.lltype = L.function_type str_t [| str_t; i1_t |] in
let concatb_func : L.llvalue = L.declare_function "concatb" concatb_t the_module in
```

```
(* function in stdlib.c for ? operator *)
let charat_t : L.lltype = L.function_type str_t [| str_t; i32_t |] in
let charat_func : L.llvalue = L.declare_function "charat" charat_t the_module in
(* function in stdlib.c for string comparison operators *)
let scomp_t : L.lltype = L.function_type i1_t [| str_t; str_t; i32_t |] in
let scomp_func : L.llvalue = L.declare_function "scomp" scomp_t the_module in
(* expose ctype.h functions as build-in functions *)
let isalnum_t : L.lltype = L.function_type i32_t [| i8_t |] in
let isalnum_func : L.llvalue = L.declare_function "isalnum" isalnum_t the_module in
let isalpha_t : L.lltype = L.function_type i32_t [| i8_t |] in
let isalpha_func : L.llvalue = L.declare_function "isalpha" isalpha_t the_module in
let iscntrl_t : L.lltype = L.function_type i32_t [| i8_t |] in
let iscntrl_func : L.llvalue = L.declare_function "iscntrl" iscntrl_t the_module in
let isdigit_t : L.lltype = L.function_type i32_t [| i8_t |] in
let isdigit_func : L.llvalue = L.declare_function "isdigit" isdigit_t the_module in
let isgraph_t : L.lltype = L.function_type i32_t [| i8_t |] in
let isgraph_func : L.llvalue = L.declare_function "isgraph" isgraph_t the_module in
let islower_t : L.lltype = L.function_type i32_t [| i8_t |] in
let islower_func : L.llvalue = L.declare_function "islower" islower_t the_module in
let isprint_t : L.lltype = L.function_type i32_t [| i8_t |] in
let isprint_func : L.llvalue = L.declare_function "isprint" isprint_t the_module in
let ispunct_t : L.lltype = L.function_type i32_t [| i8_t |] in
let ispunct_func : L.llvalue = L.declare_function "ispunct" ispunct_t the_module in
let isspace_t : L.lltype = L.function_type i32_t [| i8_t |] in
let isspace_func : L.llvalue = L.declare_function "isspace" isspace_t the_module in
let isupper_t : L.lltype = L.function_type i32_t [| i8_t |] in
let isupper_func : L.llvalue = L.declare_function "isupper" isupper_t the_module in
let isxdigit_t : L.lltype = L.function_type i32_t [| i8_t |] in
let isxdigit_func : L.llvalue = L.declare_function "isxdigit" isxdigit_t the_module in
(* expose string.h functions as build-in functions *)
let strlen_t : L.lltype = L.function_type i32_t [| str_t |] in
let strlen_func : L.llvalue = L.declare_function "strlen" strlen_t the_module in
let strcpy_t : L.lltype = L.function_type str_t [| str_t; str_t |] in
let strcpy_func : L.llvalue = L.declare_function "strcpy" strcpy_t the_module in
let strncpy_t : L.lltype = L.function_type str_t [| str_t; str_t; i32_t |] in
let strncpy_func : L.llvalue = L.declare_function "strncpy" strncpy_t the_module in
let strcat_t : L.lltype = L.function_type str_t [| str_t; str_t |] in
let strcat_func : L.llvalue = L.declare_function "strcat" strcat_t the_module in
let strncat_t : L.lltype = L.function_type str_t [| str_t; str_t; i32_t |] in
let strncat_func : L.llvalue = L.declare_function "strcat" strncat_t the_module in
let strcmp_t : L.lltype = L.function_type i32_t [| str_t; str_t |] in
let strcmp_func : L.llvalue = L.declare_function "strcmp" strcmp_t the_module in
let strncmp_t : L.lltype = L.function_type i32_t [| str_t; str_t; i32_t |] in
let strncmp_func : L.llvalue = L.declare_function "strcmp" strncmp_t the_module in
let strchr_t : L.lltype = L.function_type str_t [| str_t; str_t |] in
let strchr_func : L.llvalue = L.declare_function "strchr" strchr_t the_module in
let strrchr_t : L.lltype = L.function_type str_t [| str_t; str_t |] in
let strrchr_func : L.llvalue = L.declare_function "strrchr" strrchr_t the_module in
let strspn_t : L.lltype = L.function_type i32_t [| str_t; str_t |] in
let strspn_func : L.llvalue = L.declare_function "strspn" strspn_t the_module in
let strcspn_t : L.lltype = L.function_type i32_t [| str_t; str_t |] in
let strcspn_func : L.llvalue = L.declare_function "strcspn" strcspn_t the_module in
let strpbrk_t : L.lltype = L.function_type str_t [| str_t; str_t |] in
let strpbrk_func : L.llvalue = L.declare_function "strrchr" strpbrk_t the_module in
let strstr_t : L.lltype = L.function_type str_t [| str_t; str_t |] in
let strstr_func : L.llvalue = L.declare_function "strstr" strstr_t the_module in
let strerror_t : L.lltype = L.function_type str_t [| i32_t |] in
let strerror_func : L.llvalue = L.declare_function "strerror" strerror_t the_module in
let strtok_t : L.lltype = L.function_type str_t [| str_t; str_t |] in
let strtok_func : L.llvalue = L.declare_function "strtok" strtok_t the_module in

(* expose math.h pow function for the ^ operator *)
let pow_t : L.lltype = L.function_type float_t [| float_t; float_t |] in
let pow_func : L.llvalue = L.declare_function "pow" pow_t the_module in
(* expose math.h functions as build-in functions *)
let sin_t : L.lltype = L.function_type float_t [| float_t |] in
let sin_func : L.llvalue = L.declare_function "sin" sin_t the_module in
let cos_t : L.lltype = L.function_type float_t [| float_t |] in
let cos_func : L.llvalue = L.declare_function "cos" cos_t the_module in
let tan_t : L.lltype = L.function_type float_t [| float_t |] in
let tan_func : L.llvalue = L.declare_function "tan" tan_t the_module in
let asin_t : L.lltype = L.function_type float_t [| float_t |] in
let asin_func : L.llvalue = L.declare_function "asin" asin_t the_module in
let acos_t : L.lltype = L.function_type float_t [| float_t |] in
let acos_func : L.llvalue = L.declare_function "acos" acos_t the_module in
let atan_t : L.lltype = L.function_type float_t [| float_t |] in
let atan_func : L.llvalue = L.declare_function "atan" atan_t the_module in
let sinh_t : L.lltype = L.function_type float_t [| float_t |] in
let sinh_func : L.llvalue = L.declare_function "sinh" sinh_t the_module in
let cosh_t : L.lltype = L.function_type float_t [| float_t |] in
let cosh_func : L.llvalue = L.declare_function "cosh" cosh_t the_module in
let tanh_t : L.lltype = L.function_type float_t [| float_t |] in
let tanh_func : L.llvalue = L.declare_function "tanh" tanh_t the_module in
let exp_t : L.lltype = L.function_type float_t [| float_t |] in
let exp_func : L.llvalue = L.declare_function "exp" exp_t the_module in
let log_t : L.lltype = L.function_type float_t [| float_t |] in
let log_func : L.llvalue = L.declare_function "log" log_t the_module in
let log10_t : L.lltype = L.function_type float_t [| float_t |] in
let log10_func : L.llvalue = L.declare_function "log10" log10_t the_module in
let sqrt_t : L.lltype = L.function_type float_t [| float_t |] in
let sqrt_func : L.llvalue = L.declare_function "sqrt" sqrt_t the_module in
let floor_t : L.lltype = L.function_type float_t [| float_t |] in
let floor_func : L.llvalue = L.declare_function "floor" floor_t the_module in
let ceil_t : L.lltype = L.function_type float_t [| float_t |] in
let ceil_func : L.llvalue = L.declare_function "ceil" ceil_t the_module in
let fabs_t : L.lltype = L.function_type float_t [| float_t |] in
let fabs_func : L.llvalue = L.declare_function "fabs" fabs_t the_module in
let srand_t : L.lltype = L.var_arg_function_type i32_t [| i32_t |] in
let srand_func : L.llvalue = L.declare_function "srand" srand_t the_module in
let rand_t : L.lltype = L.function_type i32_t [|  |] in
let rand_func : L.llvalue = L.declare_function "rand" rand_t the_module in
let fmod_t : L.lltype = L.function_type float_t [| float_t; float_t |] in
let fmod_func : L.llvalue = L.declare_function "fmod" fmod_t the_module in

(* Define each function (arguments and return type) so we can
   call it even before we've created its body *)
let function_decls : (L.llvalue * sCasperFunction) StringMap.t =
  let function_decl m fdecl =
    let name = fdecl.sFunctionName
    and formal_types =
  Array.of_list (List.map (fun (t,_) -> ltype_of_casperType t) fdecl.sFunctionFormals)
    in let ftype = L.function_type (ltype_of_casperType fdecl.sFunctionType) formal_types in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
  List.fold_left function_decl StringMap.empty functions in

(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.sFunctionName function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in

  let int_format_str_nl = L.build_global_stringptr "%d\n" "fmt" builder
  and int_format_str = L.build_global_stringptr "%d" "fmt" builder
  and float_format_str_nl = L.build_global_stringptr "%.10f\n" "fmt" builder
  and float_format_str = L.build_global_stringptr "%g" "fmt" builder
  and str_format_str_nl = L.build_global_stringptr "%s\n" "fmt" builder
  and str_format_str = L.build_global_stringptr "%s" "fmt" builder
  and chr_format_str_nl = L.build_global_stringptr "%c\n" "fmt" builder
  and chr_format_str = L.build_global_stringptr "%c" "fmt" builder
  and bool_format_str_ln = L.build_global_stringptr "%d\n" "fmt" builder in

  (* Construct the function's "locals": formal arguments and locally
     declared variables.  Allocate each on the stack, initialize their
     value, if appropriate, and remember their values in the "locals" map *)
  let local_vars =
    let add_formal m (t, n) p = L.set_value_name n p;
  let local = L.build_alloca (ltype_of_casperType t) n builder in
      ignore (L.build_store p local builder);
  StringMap.add n local m

    (* Allocate space for any locally declared variables and add the
     * resulting registers to our map *)
    and add_local m (t, n) =
  let local_var = L.build_alloca (ltype_of_casperType t) n builder
  in StringMap.add n local_var m
```

```
    in

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

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

  (* Construct code for an expression; return its value *)
  let rec expr builder ((_, e) : sCasperExpression) = match e with
      SEpsilon    -> L.const_int i32_t 0
  | SIntLIT i   -> L.const_int i32_t i
    | SFltLIT f   -> L.const_float_of_string float_t f
    | SStrLIT s   -> L.build_global_stringptr s "str" builder
    | SChrLIT c   -> L.const_int i8_t (C.code c)
    | SBoolLIT b  -> L.const_int i1_t (if b then 1 else 0)
    | SVoidLIT    -> L.const_int i32_t 0
    | SNullLIT    -> L.const_int i32_t 0
    | SIdentifier s -> L.build_load (lookup s) s builder
    | SBinOP (((A.String,_ ) as e1), op, ((t, _) as e2)) ->
      let e1' = expr builder e1
      and e2' = expr builder e2
      in
      (match op with
          A.Con when t = A.Int   -> L.build_call concati_func [| e1' ; e2' |] "coni" builder
        | A.Con when t = A.Float -> L.build_call concatf_func [| e1' ; e2' |] "conf" builder
        | A.Con when t = A.Bool  -> L.build_call concatb_func [| e1' ; e2' |] "conb" builder
        | A.Con                  -> L.build_call concats_func [| e1' ; e2' |] "cons" builder
        | A.Cat -> L.build_call charat_func [| e1' ; e2' |] "cat" builder
        | A.Eql -> L.build_call scomp_func [| e1' ; e2' ; (expr builder (A.Int, SIntLIT 1)) |] "scomp" builder
        | A.Neq -> L.build_call scomp_func [| e1' ; e2' ; (expr builder (A.Int, SIntLIT 2)) |] "scomp" builder
        | A.Lst -> L.build_call scomp_func [| e1' ; e2' ; (expr builder (A.Int, SIntLIT 3)) |] "scomp" builder
        | A.Lse -> L.build_call scomp_func [| e1' ; e2' ; (expr builder (A.Int, SIntLIT 4)) |] "scomp" builder
        | A.Grt -> L.build_call scomp_func [| e1' ; e2' ; (expr builder (A.Int, SIntLIT 5)) |] "scomp" builder
        | A.Gre -> L.build_call scomp_func [| e1' ; e2' ; (expr builder (A.Int, SIntLIT 6)) |] "scomp" builder
        | A.Add | A.Sub | A.Mul | A.Div | A.Mod | A.Exp | A.And | A.Or ->
          raise (Failure "internal error: semant should have rejected these ops on string")
      )
```

```
| SBinOP ((A.Float,_ ) as e1, op, e2) ->
  let e1' = expr builder e1
  and e2' = expr builder e2 in
  (match op with
      A.Add     -> L.build_fadd e1' e2' "fadd"
    | A.Sub     -> L.build_fsub e1' e2' "fsub"
    | A.Mul     -> L.build_fmul e1' e2' "fmul"
    | A.Div     -> L.build_fdiv  e1' e2' "fdiv"
    | A.Mod     -> L.build_call fmod_func [| e1' ; e2' |] "fmod"
    | A.Exp     -> L.build_call pow_func [| e1' ; e2' |] "fpow"
    | A.Eql     -> L.build_fcmp L.Fcmp.Oeq e1' e2' "feq"
    | A.Neq     -> L.build_fcmp L.Fcmp.One e1' e2' "fneq"
    | A.Lst     -> L.build_fcmp L.Fcmp.Olt e1' e2' "flst"
    | A.Lse     -> L.build_fcmp L.Fcmp.Ole e1' e2' "flse"
    | A.Grt     -> L.build_fcmp L.Fcmp.Ogt e1' e2' "fgrt"
    | A.Gre     -> L.build_fcmp L.Fcmp.Oge e1' e2' "fgre"
    | A.And | A.Or | A.Con | A.Cat ->
      raise (Failure "internal error: semant should have rejected and/or/con/cat on float")
  ) builder
| SBinOP (e1, op, e2) ->
  let e1' = expr builder e1
  and e2' = expr builder e2 in
  (match op with
      A.Add     -> L.build_add e1' e2' "tmp"
    | A.Sub     -> L.build_sub e1' e2' "tmp"
    | A.Mul     -> L.build_mul e1' e2' "tmp"
    | A.Div     -> L.build_sdiv e1' e2' "tmp"
    | A.Mod     -> L.build_srem e1' e2' "tmp"
    | A.And     -> L.build_and e1' e2' "tmp"
    | A.Or      -> L.build_or e1' e2' "tmp"
    | A.Eql     -> L.build_icmp L.Icmp.Eq e1' e2' "tmp"
    | A.Neq     -> L.build_icmp L.Icmp.Ne e1' e2' "tmp"
    | A.Lst     -> L.build_icmp L.Icmp.Slt e1' e2' "tmp"
    | A.Lse     -> L.build_icmp L.Icmp.Sle e1' e2' "tmp"
    | A.Grt     -> L.build_icmp L.Icmp.Sgt e1' e2' "tmp"
    | A.Gre     -> L.build_icmp L.Icmp.Sge e1' e2' "tmp"
    | A.Exp | A.Con | A.Cat ->
        raise (Failure "internal error: semant should have rejected pow/con/cat on int or bool")
  ) builder
| SUnrOP(op, ((t, _) as e)) -> let e' = expr builder e in
    (match op with
        A.Neg when t = A.Float -> L.build_fneg   e'           "tmp" builder
      | A.Neg                  -> L.build_neg    e'           "tmp" builder
      | A.Not                  -> L.build_not    e'           "tmp" builder
      | A.ItoF when t = A.Float -> L.build_fptosi e' i32_t    "tmp" builder
      | A.ItoF                  -> L.build_sitofp e' float_t  "tmp" builder
    )
```

```
| SAsgnOP (s, op, ((t, _) as e)) -> let e' = expr builder e in
    (match op with
        A.Asgn when t = A.String   -> ignore(L.build_store (
                                            L.build_call concats_func [| (expr builder (A.String, SStrLIT "")) ; e' |]
                                                          "cons" builder)
                                            (lookup s) builder); e'
      | A.ConAsgn when t = A.Int   -> ignore(L.build_store (
                                            L.build_call concati_func [| (L.build_load (lookup s) "tmp" builder) ; e' |]
                                                          "coni" builder)
                                            (lookup s) builder); e'
      | A.ConAsgn when t = A.Float -> ignore(L.build_store (
                                            L.build_call concatf_func [| (L.build_load (lookup s) "tmp" builder) ; e' |]
                                                          "conf" builder)
                                            (lookup s) builder); e'
      | A.ConAsgn when t = A.Bool  -> ignore(L.build_store (
                                            L.build_call concatb_func [| (L.build_load (lookup s) "tmp" builder) ; e' |]
                                                          "conb" builder)
                                            (lookup s) builder); e'
      | A.ConAsgn                  -> ignore(L.build_store (
                                            L.build_call concats_func
                                                   [| (L.build_load (lookup s) "tmp" builder) ; e' |]
                                                          "cons" builder)
                                            (lookup s) builder); e'
      | A.Asgn                     -> ignore(L.build_store e' (lookup s) builder); e'
      | A.AddAsgn when t = A.Float -> ignore(L.build_store (L.build_fadd (L.build_load (lookup s) "tmp" builder) e'
                                                                    "tmp" builder) (lookup s) builder); e'
      | A.AddAsgn                  -> ignore(L.build_store (L.build_add (L.build_load (lookup s) "tmp" builder) e'
                                                                    "tmp" builder) (lookup s) builder); e'
      | A.SubAsgn when t = A.Float -> ignore(L.build_store (L.build_fsub (L.build_load (lookup s) "tmp" builder) e'
                                                                    "tmp" builder) (lookup s) builder); e'
      | A.SubAsgn                  -> ignore(L.build_store (L.build_sub (L.build_load (lookup s) "tmp" builder) e'
                                                                    "tmp" builder) (lookup s) builder); e'

     )
| SFunctionCall ("printi",   [e]) -> L.build_call printf_func [| int_format_str      ; (expr builder e) |] "printf" builder
| SFunctionCall ("printinl", [e]) -> L.build_call printf_func [| int_format_str_nl   ; (expr builder e) |] "printf" builder
| SFunctionCall ("printf",   [e]) -> L.build_call printf_func [| float_format_str    ; (expr builder e) |] "printf" builder
| SFunctionCall ("printfnl", [e]) -> L.build_call printf_func [| float_format_str_nl ; (expr builder e) |] "printf" builder
| SFunctionCall ("prints",   [e]) -> L.build_call printf_func [| str_format_str      ; (expr builder e) |] "printf" builder
| SFunctionCall ("printsnl", [e]) -> L.build_call printf_func [| str_format_str_nl   ; (expr builder e) |] "printf" builder
| SFunctionCall ("printc",   [e]) -> L.build_call printf_func [| chr_format_str      ; (expr builder e) |] "printf" builder
| SFunctionCall ("printcnl", [e]) -> L.build_call printf_func [| chr_format_str_nl   ; (expr builder e) |] "printf" builder
| SFunctionCall ("printbasi", [e])-> L.build_call printf_func [| bool_format_str_ln  ; (expr builder e) |] "printf" builder
| SFunctionCall ("printb",   [e]) -> L.build_call printb_func [| (expr builder e) ; (expr builder (A.Int, SIntLIT 0)) |]
                                                  "printb" builder
| SFunctionCall ("printbnl", [e]) -> L.build_call printb_func [| (expr builder e) ; (expr builder (A.Int, SIntLIT 1)) |]
                                                 "printb" builder
| SFunctionCall ("printbig", [e]) -> L.build_call printbig_func [| (expr builder e) |] "printbig" builder
| SFunctionCall ("sin",   [e]) -> L.build_call sin_func   [| (expr builder e) |] "sin"   builder
| SFunctionCall ("cos",   [e]) -> L.build_call cos_func   [| (expr builder e) |] "cos"   builder
| SFunctionCall ("tan",   [e]) -> L.build_call tan_func   [| (expr builder e) |] "tan"   builder
| SFunctionCall ("asin",  [e]) -> L.build_call asin_func  [| (expr builder e) |] "asin"  builder
| SFunctionCall ("acos",  [e]) -> L.build_call acos_func  [| (expr builder e) |] "acos"  builder
| SFunctionCall ("atan",  [e]) -> L.build_call atan_func  [| (expr builder e) |] "atan"  builder
| SFunctionCall ("sinh",  [e]) -> L.build_call sinh_func  [| (expr builder e) |] "sinh"  builder
| SFunctionCall ("cosh",  [e]) -> L.build_call cosh_func  [| (expr builder e) |] "cosh"  builder
| SFunctionCall ("tanh",  [e]) -> L.build_call tanh_func  [| (expr builder e) |] "tanh"  builder
| SFunctionCall ("exp",   [e]) -> L.build_call exp_func   [| (expr builder e) |] "exp"   builder
| SFunctionCall ("log",   [e]) -> L.build_call log_func   [| (expr builder e) |] "log"   builder
| SFunctionCall ("log10", [e]) -> L.build_call log10_func [| (expr builder e) |] "log10" builder
| SFunctionCall ("sqrt",  [e]) -> L.build_call sqrt_func  [| (expr builder e) |] "sqrt"  builder
| SFunctionCall ("floor", [e]) -> L.build_call floor_func [| (expr builder e) |] "floor" builder
| SFunctionCall ("ceil",  [e]) -> L.build_call ceil_func  [| (expr builder e) |] "ceil"  builder
| SFunctionCall ("abs",   [e]) -> L.build_call fabs_func  [| (expr builder e) |] "abs"   builder
| SFunctionCall ("srand", [e]) -> L.build_call srand_func [| (expr builder e) |] "srand" builder
| SFunctionCall ("rand",  []) -> L.build_call rand_func   [|  |] "rand"  builder
| SFunctionCall ("fmod",  [e1; e2])   -> L.build_call fmod_func [| (expr builder e1); (expr builder e2) |]   "fmod"  builder
| SFunctionCall ("strlen", [e1])     -> L.build_call strlen_func  [| (expr builder e1) |]                    "strlen" builder
| SFunctionCall ("strcpy", [e1; e2]) -> L.build_call strcpy_func  [| (expr builder e1); (expr builder e2) |] "strcpy" builder
| SFunctionCall ("strncpy",[e1; e2; e3]) -> L.build_call strncpy_func [| (expr builder e1); (expr builder e2);
                                                                      (expr builder e3) |] "strncpy" builder
| SFunctionCall ("strcat", [e1; e2]) -> L.build_call strcat_func  [| (expr builder e1); (expr builder e2) |] "strcat" builder
| SFunctionCall ("strncat",[e1; e2; e3]) -> L.build_call strncat_func [| (expr builder e1); (expr builder e2);
                                                                      (expr builder e3) |] "strncat" builder
| SFunctionCall ("strcmp", [e1; e2]) -> L.build_call strcmp_func  [| (expr builder e1); (expr builder e2) |] "strcmp" builder
| SFunctionCall ("strncmp",[e1; e2; e3]) -> L.build_call strncmp_func [| (expr builder e1); (expr builder e2);
                                                                      (expr builder e3) |] "strncmp" builder
| SFunctionCall ("strchr", [e1; e2]) -> L.build_call strchr_func  [| (expr builder e1); (expr builder e2) |] "strchr" builder
| SFunctionCall ("strrchr",[e1; e2]) -> L.build_call strrchr_func [| (expr builder e1); (expr builder e2) |] "strrchr" builder
| SFunctionCall ("strspn", [e1; e2]) -> L.build_call strspn_func  [| (expr builder e1); (expr builder e2) |] "strspn" builder
| SFunctionCall ("strcspn",[e1; e2]) -> L.build_call strcspn_func [| (expr builder e1); (expr builder e2) |] "strcspn" builder
| SFunctionCall ("strpbrk",[e1; e2]) -> L.build_call strpbrk_func [| (expr builder e1); (expr builder e2) |] "strpbrk" builder
| SFunctionCall ("strstr", [e1; e2]) -> L.build_call strstr_func  [| (expr builder e1); (expr builder e2) |] "strstr" builder
| SFunctionCall ("strerror",[e1])    -> L.build_call strerror_func [| (expr builder e1) |]                   "strerror" builder
| SFunctionCall ("strtok",  [e1; e2])-> L.build_call strtok_func  [| (expr builder e1); (expr builder e2) |] "strtok" builder

| SFunctionCall ("isalnum",  [e]) -> L.build_call isalnum_func  [| (expr builder e) |] "isalnum"  builder
| SFunctionCall ("isalpha",  [e]) -> L.build_call isalpha_func  [| (expr builder e) |] "isalpha"  builder
| SFunctionCall ("iscntrl",  [e]) -> L.build_call iscntrl_func  [| (expr builder e) |] "iscntrl"  builder
| SFunctionCall ("isdigit",  [e]) -> L.build_call isdigit_func  [| (expr builder e) |] "isdigit"  builder
| SFunctionCall ("isgraph",  [e]) -> L.build_call isgraph_func  [| (expr builder e) |] "isgraph"  builder
| SFunctionCall ("islower",  [e]) -> L.build_call islower_func  [| (expr builder e) |] "islower"  builder
| SFunctionCall ("isprint",  [e]) -> L.build_call isprint_func  [| (expr builder e) |] "isprint"  builder
| SFunctionCall ("ispunct",  [e]) -> L.build_call ispunct_func  [| (expr builder e) |] "ispunct"  builder
| SFunctionCall ("isspace",  [e]) -> L.build_call isspace_func  [| (expr builder e) |] "isspace"  builder
| SFunctionCall ("isupper",  [e]) -> L.build_call isupper_func  [| (expr builder e) |] "isupper"  builder
| SFunctionCall ("isxdigit", [e]) -> L.build_call isxdigit_func [| (expr builder e) |] "isxdigit" builder
```

```
  | SFunctionCall (f, args) -> let (fdef, fdecl) = StringMap.find f function_decls in

let llargs = List.rev (List.map (expr builder) (List.rev args)) in
let result = (match fdecl.sFunctionType with
                    A.Void -> ""
                  | _ -> f ^ "_result") in
     L.build_call fdef (Array.of_list llargs) result builder
in

(* LLVM insists each basic block end with exactly one "terminator"
   instruction that transfers control.  This function runs "instr builder"
   if the current block does not already have a terminator.  Used,
   e.g., to handle the "fall off the end of the function" case. *)
let add_terminal builder instr =
  match L.block_terminator (L.insertion_block builder) with
    Some _ -> ()
  | None -> ignore (instr builder) in

(* Build the code for the given statement; return the builder for
   the statement's successor (i.e., the next instruction will be built
   after the one generated by this call) *)

let rec stmt builder = function
    SStatementBlock sl -> List.fold_left stmt builder sl
  | SExpression e -> ignore(expr builder e); builder
  | SBreak ->    ignore(L.build_br (L.block_of_value !break_block) builder); builder
  | SContinue -> ignore(L.build_br (L.block_of_value !continue_block) builder); builder
  | SReturn e -> ignore(match fdecl.sFunctionType with
                 (* Special "return nothing" instr *)
                 A.Void -> L.build_ret_void builder
                 (* Build return statement *)
                 | _ -> L.build_ret (expr builder e) builder ); builder
  | SIfCondition (predicate, then_stmt, else_stmt) ->
     let bool_val = expr builder predicate in
    let merge_bb = L.append_block context "merge" the_function in
         let build_br_merge = L.build_br merge_bb in (* partial function *)

            let then_bb = L.append_block context "then" the_function in
               add_terminal (stmt (L.builder_at_end context then_bb) then_stmt) build_br_merge;

            let else_bb = L.append_block context "else" the_function in
               add_terminal (stmt (L.builder_at_end context else_bb) else_stmt) build_br_merge;

            ignore(L.build_cond_br bool_val then_bb else_bb builder);
            L.builder_at_end context merge_bb
    | SWhileLoop (predicate, body) ->
        let pred_bb = L.append_block context "while" the_function in
          let body_bb = L.append_block context "while_body" the_function in
            let merge_bb = L.append_block context "merge" the_function in
              let _ = L.build_br pred_bb builder in
                let _ = break_block := L.value_of_block merge_bb in
                let _ = continue_block := L.value_of_block pred_bb in
                  let pred_builder = L.builder_at_end context pred_bb in
                    let bool_val = expr pred_builder predicate in
                      let _ = L.build_cond_br bool_val body_bb merge_bb pred_builder in
                        add_terminal (stmt (L.builder_at_end context body_bb) body) (L.build_br pred_bb);
                        L.builder_at_end context merge_bb

    (* Implement for loops as while loops *)
    | SForLoop (e1, e2, e3, body) -> stmt builder
      ( SStatementBlock [SExpression e1 ; SWhileLoop (e2, SStatementBlock [body ; SExpression e3]) ] )
    (* Implement do until loops as while loops *)
    | SDoUntilLoop (body, (t, e)) -> stmt builder
      ( SStatementBlock [body ; SWhileLoop ( ( t, SUnrOP(A.Not, (t,e)) ) , body) ] )
    (* Implement do while loops as while loops *)
    | SDoWhileLoop (body, predicate) -> stmt builder
      ( SStatementBlock [body ; SWhileLoop (predicate, body) ] )
  in

  (* Build the code for each statement in the function *)
  let builder = stmt builder (SStatementBlock fdecl.sFunctionStatements) in

  (* Add a return if the last block falls off the end *)
  add_terminal builder (match fdecl.sFunctionType with
      A.Void -> L.build_ret_void
    | A.Float -> L.build_ret (L.const_float float_t 0.0)
    | t -> L.build_ret (L.const_int (ltype_of_casperType t) 0))
in

List.iter build_function_body functions;
the_module
```

8.8 Casper C Standard Library, ./stdlib.c

```
/*
    Standard Library for Casper
    Based on MicroC printbig.c
    File: stdlib.c
    Michael Makris, mm3443
    PLT Fall 2018
*/
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <math.h>

/* String functions for operators */
char *concats(char *s1, char *s2) { /* String _ String */
    char *answer = (char *) malloc(strlen(s1) + strlen(s2) + 1);
    strcpy(answer, s1);
    strcat(answer, s2);
    return answer;
}

char *concati(char *s1, int x) { /* String _ int */
    int length = snprintf( NULL, 0, "%d", x );
    char* s2 = malloc( length + 1 );
    snprintf(s2, length + 1, "%d", x);

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

char *concatf(char *s1, double f) { /* String _ float */
    int length = snprintf( NULL, 0, "%g", f );
    char* s2 = malloc( length + 1 );
    snprintf(s2, length + 1, "%g", f);

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

char *concatb(char *s1, int x) { /* String _ bool */
    char *s2 = "false";

    if (x == 1) s2 = "true";

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

char *charat(char *s, int i) {
    char *answer = (char *) malloc(2);
    answer[0] = s[i]; answer[1] = '\0';
    return answer;
}
```

```
int scomp(char *s1, char *s2, int i) {
    int answer = 0;
    int comp = strcmp(s1, s2);
    switch (i) {
        case 1:   /* Eql */
            answer = (comp == 0);
            break;
        case 2:   /* Neq */
            answer = (comp != 0);
            break;
        case 3:   /* Lst */
            answer = (comp < 0);
            break;
        case 4:   /* Lse */
            answer = (comp <= 0);
            break;
        case 5:   /* Grt */
            answer = (comp > 0);
            break;
        case 6:   /* Gre */
            answer = (comp >= 0);
            break;
    }
    if (answer != 0) answer = 1;

    return answer;
}

void printb(int b, int nl){
    if(b){
      if(nl) printf("%s\n", "true");
      else printf("true");
    } else {
      if(nl) printf("%s\n", "false");
      else printf("false");
    }
}

/*
 *  A function illustrating how to link C code to code generated from LLVM
 * Font information: one byte per row, 8 rows per character
 * In order, space, 0-9, A-Z
 */
static const char font[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x1c, 0x3e, 0x61, 0x41, 0x43, 0x3e, 0x1c, 0x00,
  0x00, 0x40, 0x42, 0x7f, 0x7f, 0x40, 0x40, 0x00,
  0x62, 0x73, 0x79, 0x59, 0x5d, 0x4f, 0x46, 0x00,
  0x20, 0x61, 0x49, 0x4d, 0x4f, 0x7b, 0x31, 0x00,
  0x18, 0x1c, 0x16, 0x13, 0x7f, 0x7f, 0x10, 0x00,
  0x27, 0x67, 0x45, 0x45, 0x45, 0x7d, 0x38, 0x00,
  0x3c, 0x7e, 0x4b, 0x49, 0x49, 0x79, 0x30, 0x00,
  0x03, 0x03, 0x71, 0x79, 0x0d, 0x07, 0x03, 0x00,
  0x36, 0x4f, 0x4d, 0x59, 0x59, 0x76, 0x30, 0x00,
  0x06, 0x4f, 0x49, 0x49, 0x69, 0x3f, 0x1e, 0x00,
  0x7c, 0x7e, 0x13, 0x11, 0x13, 0x7e, 0x7c, 0x00,
  0x7f, 0x7f, 0x49, 0x49, 0x49, 0x7f, 0x36, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00,
  0x7f, 0x7f, 0x41, 0x41, 0x63, 0x3e, 0x1c, 0x00,
  0x00, 0x7f, 0x7f, 0x49, 0x49, 0x49, 0x41, 0x00,
  0x7f, 0x7f, 0x09, 0x09, 0x09, 0x09, 0x01, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x49, 0x79, 0x79, 0x00,
  0x7f, 0x7f, 0x08, 0x08, 0x08, 0x7f, 0x7f, 0x00,
  0x00, 0x41, 0x41, 0x7f, 0x7f, 0x41, 0x41, 0x00,
  0x20, 0x60, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x7f, 0x7f, 0x18, 0x3c, 0x76, 0x63, 0x41, 0x00,
  0x00, 0x7f, 0x7f, 0x40, 0x40, 0x40, 0x40, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x0e, 0x7f, 0x7f, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x38, 0x7f, 0x7f, 0x00,
  0x3e, 0x7f, 0x41, 0x41, 0x41, 0x7f, 0x3e, 0x00,
  0x7f, 0x7f, 0x11, 0x11, 0x11, 0x1f, 0x0e, 0x00,
  0x3e, 0x7f, 0x41, 0x51, 0x71, 0x3f, 0x5e, 0x00,
  0x7f, 0x7f, 0x11, 0x31, 0x79, 0x6f, 0x4e, 0x00,
  0x26, 0x6f, 0x49, 0x49, 0x4b, 0x7a, 0x30, 0x00,
  0x00, 0x01, 0x01, 0x7f, 0x7f, 0x01, 0x01, 0x00,
  0x3f, 0x7f, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x0f, 0x1f, 0x38, 0x70, 0x38, 0x1f, 0x0f, 0x00,
  0x1f, 0x7f, 0x38, 0x1c, 0x38, 0x7f, 0x1f, 0x00,
  0x63, 0x77, 0x3e, 0x1c, 0x3e, 0x77, 0x63, 0x00,
  0x00, 0x03, 0x0f, 0x78, 0x78, 0x0f, 0x03, 0x00,
  0x61, 0x71, 0x79, 0x5d, 0x4f, 0x47, 0x43, 0x00
};

void printbig(int c)
{
  int index = 0;
  int col, data;
  if (c >= '0' && c <= '9') index = 8 + (c - '0') * 8;
  else if (c >= 'A' && c <= 'Z') index = 88 + (c - 'A') * 8;
  do {
    data = font[index++];
    for (col = 0 ; col < 8 ; data <<= 1, col++) {
      char d = data & 0x80 ? 'X' : ' ';
      putchar(d); putchar(d);
    }
    putchar('\n');
  } while (index & 0x7);
}

/* #define BUILD_TEST */
#ifdef BUILD_TEST
int main()
{
  char s[] = "HELLO WORLD09AZ";
  char *c;
  for ( c = s ; *c ; c++) printbig(*c);
}
#endif
```

8.9 Makefile and related scripts

8.9.1 Makefile

```
# "make casper.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild,
# e.g., by including packages, enabling warnings
# See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc
casper.native :
	opam config exec -- \
	ocamlbuild -use-ocamlfind casper.native

# Compile the standard library
stdlib : stdlib.c
	cc -o stdlib -DBUILD_TEST stdlib.c

.PHONY : help
help:
	./makesupport/help.sh

# "make testS" loads a version of the scanner that prints out the tokens instead of
# feeding to parser and then scans a program
.PHONY : testS
testS:
	./makesupport/testscanner.sh

# "make testP" uses menhir to evaluate some token strings with the parser
.PHONY: testP
testP:
	./makesupport/testparser.sh

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

# "make all" builds the executable and standard library
.PHONY : all
all : casper.native stdlib.o

# "make clean" removes all generated files
.PHONY : clean
clean :
	ocamlbuild -clean
	./makesupport/clean.sh

# "make cleang" leaves compiler removes other generated files
.PHONY : cleang
cleang :
	./makesupport/clean.sh

# Building the tarball
TESTS = \
  add1 addassign arith1 arith2 arith3 assign break1 break2 conassign1 conassign2 \
  cont1 dec dountil1 dountil2 dountil3 dowhile1 dowhile2 dowhile3 dup fib float1 \
  float2 float3 for1 for2 for3 func1 func2 func3 func4 func5 func6 func7 func8 func9 \
  gcd2 gcd global1 global2 global3 global5 global6 hello helloworld if1 if2 if3 \
  if4 if5 if6 if7 inc1 inc2 int1 int2 int3 itof local1 local2 local3 negation \
  not ops1 ops2 pow printb1 printb printbig print stdlib str1 str3 str4 str5 \
  subassign var1 var2 while1 while2 while3 while4
```

```
FAILS = \
  addassign1 addassign1 addassign2 addassign3 addassign4 addassign5 addassign6 assign10 \
  assign11 assign12 assign13 assign14 assign1 assign2 assign3 assign4 assign5 assign6 \
  assign7 assign8 assign9 break1 continue1 dead1 dead2 dead3 dec1 dec2 dec3 dec4 dec5 \
  dec6 dup1 dup2 dup3 dup4 expr1 expr2 expr3 float1 float2 float3 fmod for1 for2 for3 \
  for4 for5 func10 func1 func2 func3 func4 func5 func6 func7 func8 func9 global1 \
  global2 if1 if2 if3 inc1 inc2 inc3 inc4 inc5 inc6 local1 negation1 negation2 \
  negation3 negation4 nomain not1 not2 not3 not4 not5 not6 printb printc printbig \
  printf printi prints printsnl rand return1 return2 str1 str3 str4 str7 subassign1 \
  subassign2 subassign3 subassign4 subassign5 subassign6 while1 while2

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

CLANG = clangviz.sh xxx.c
DOCS = Proposal.pdf LRM.pdf Casper.pdf Log.xlsx
EXAMPLES = bsearch.goo sieve.goo temperaturetable.goo
MAKESUPPORT = clean.sh help.sh testall.sh testparser.sh testscanner.sh
TESTFRONT = menhirTestInput.txt CasperScannerValidationWithLinenumbers.mll \
            CasperScannerValidationInput.txt CasperScannerValidation.mll

TARFILES = casper.ml scanner.mll parser.mly ast.ml sast.ml semant.ml codegen.ml \
	Makefile _tags casper.sh casperc.sh casperll.sh stdlib.c README README.md \
	$(CLANG:%=clang/%) \
	$(DOCS:%=docs/%) \
	$(EXAMPLES:%=examples/%) \
	$(MAKESUPPORT:%=makesupport/%) \
	$(TESTFILES:%=tests/%) \
	$(TESTFRONT:%=X/%)

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

8.9.2 ./makesupport/testsall.sh

```
#!/bin/sh

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

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

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

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

# Set time limit for all operations
ulimit -t 30

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

keep=0

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

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

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

```
# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
       SignalError "$1 failed on $*"
       return 1
    }
}

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

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

    echo -n "$basename\t\t\t"

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe ${basename}.out"
&&
    Run "$CASPER" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "stdlib.o" "-lm" &&
    Run "./${basename}.exe" > "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

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

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

    echo -n "$basename\t\t\t"

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

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

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

shift `expr $OPTIND - 1`

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

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

if [ ! -f stdlib.o ]
then
    echo "Could not find stdlib.o"
    echo "Try \"make stdlib.o\""
    exit 1
fi
```

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

for file in $files
do
    case $file in
       *test-*)
           Check $file 2>> $globallog
           ;;
       *fail-*)
           CheckFail $file 2>> $globallog
           ;;
       *)
           echo "unknown file type $file"
           globalerror=1
           ;;
    esac
done

exit $globalerror
```

8.9.3 ./makesupport/testscanner.sh

```
ocamllex ./X/CasperScannerValidation.mll
ocaml ./X/CasperScannerValidation.ml < ./X/CasperScannerValidationInput.txt
rm -f ./X/CasperScannerValidation.ml

ocamllex ./X/CasperScannerValidationWithLinenumbers.mll
ocaml ./X/CasperScannerValidationWithLinenumbers.ml < ./X/CasperScannerValidationInput.txt
rm -f ./X/CasperScannerValidationWithLinenumbers.ml
```

8.9.4 ./X/CasperScannerValidation.mll

```
(*
    CasperScannerValidation.mll
    Michael Makris, mm3443
    PLT Fall 2018

    Purpose: to validate CasperScanner.mll. Removed Parser dependency and added printf statements
to identify current token read.

    Use:
        ocamllex CasperScannerValidation.mll
        ocaml CasperScannerValidation.ml < CasperScannerValidationInput.txt
*)

{
open Printf
}

(* definitions *)
let whitespace = [' ' '\t']+
let newline = '\n' | '\r' | "\r\n"

let digit = ['0'-'9']
let integer = digit+ (* '-'?digit+ *)
let float = (digit+ '.' digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ '.' ['e' 'E'] ['+' '-']? digit+)
            | ('.' digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ '.' digit+) | (digit+ '.')
            | ('.' digit+)

let letter = ['a'-'z' 'A'-'Z' '_']
let id = letter(letter|digit)*

let badbraces = (['{'](whitespace | newline | '{')*['{']) | (['}'](whitespace | newline |
'}')*['}']) | (['}'](whitespace | newline)*['{'])
let badbrackets = (['['](whitespace | newline | '[')*['[']) | ([']'](whitespace | newline |
']')*[']'])
let badend = [';'](whitespace | newline)*['{'] | ['}'](whitespace | newline)*[';']
let badcommas = [','](whitespace | newline)*[',']
let badsemis =  [';'](whitespace | newline)*[';']
let badcomments = "*/" | "**"
let badcombs = badbraces | badbrackets | badend | badcommas | badsemis | badcomments


(* rules *)
rule token = parse
  (* whitespace *)
    whitespace { token lexbuf } | newline { token lexbuf }

  (* comments *)
  | "//" { printf "Line comment Start\n"; linecomment lexbuf }
  | "/*" { printf "Comments level 0 start\n"; blockcomment 0 lexbuf }

  (* operators *)
  | '(' | ')' | '[' | ']' | '{' | '}' | ';' | ','
  | '_' | '?' | '+' | '-' | '*' | '/' | '%' | '^'
  | '>'  | ">=" | '<'  | "<=" | "==" | "!=" | "++" | "--"
  | '=' | "_=" | "+=" | "-="
  | "&&" | "||" | "!"
  as op { printf "operator    %s\n" op; token lexbuf }

  (* keywords *)
  | "if" | "else" | "for" | "while" | "break" | "continue" | "return"
  | "int" | "float" | "str" | "bool" | "void"
  | "true" | "false" | "null"
  as word {printf "keyword     %s\n" word; token lexbuf}

  (* literals *)
  | integer as lexeme { printf "integer     %s (%d)\n" lexeme (int_of_string lexeme);   token
lexbuf }
  | float as lexeme   { printf "float       %s (%f)\n" lexeme (float_of_string lexeme); token
lexbuf }
  | '''               { printf "string      %s\n" (stringSQ (Buffer.create 100) lexbuf); token
lexbuf }
  | '"'               { printf "string      %s\n" (stringDQ (Buffer.create 100) lexbuf); token
lexbuf }

  (* indentifier *)
  |  id as lexeme     { printf "identifier  %s\n" lexeme; token lexbuf }

  (* end of file *)
  | eof               { printf "EOF\n" }

  (* errors *)
  | badcombs as lexeme { raise (Failure("Bad syntax: " ^ lexeme)) }
  | _ as character     { raise (Failure("Bad character: " ^ Char.escaped character)) }

and stringSQ tempbuffer = parse
    '''                 { Buffer.contents tempbuffer }
  | newline             { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringSQ tempbuffer
lexbuf }
  | [^ '''  '\n' '\r']+ { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringSQ tempbuffer
lexbuf }
  | eof                 { raise (Failure("Non-terminated single quotes")) }

and stringDQ tempbuffer = parse
    '"'                 { Buffer.contents tempbuffer }
  | newline             { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringDQ tempbuffer
lexbuf }
  | [^ '"'  '\n' '\r']+ { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringDQ tempbuffer
lexbuf }
  | eof                 { raise (Failure("Non-terminated double quotes")) }

and linecomment = parse
    newline { printf "Line comment End\n"; token lexbuf }
  | eof     { printf "EOF \n" }
  | _       { linecomment lexbuf }

and blockcomment level = parse
    "*/"    { if level = 0 then (printf "Comments level %d end\n" level; token lexbuf)
              else (printf "Comments level %d end \n" level; blockcomment (level-1) lexbuf) }
  | "/*"    { printf "Comments level %d start\n" (level+1); blockcomment (level+1) lexbuf }
  | newline { blockcomment level lexbuf }
  | eof     { raise (Failure("Non-terminated comments")) }
  | _       { blockcomment level lexbuf }
```

```
(* trailer *)

{
let main () =
    let cin =
        if Array.length Sys.argv > 1
        then open_in Sys.argv.(1)
        else stdin
    in
        let lexbuf = Lexing.from_channel cin in
            token lexbuf
let _ = Printexc.print main ()
}
```

8.9.5 ./X/CasperScannerValidationWithLinenumbers.mll

```
(*
    CasperScannerValidationWithLinenumbers.mll
    Michael Makris, mm3443
    PLT Fall 2018

    Purpose: to validate CasperScanner.mll. Removed Parser dependency and added printf statements
to identify current token read.

    Use:
        ocamllex CasperScannerValidationWithLinenumbers.mll
        ocaml CasperScannerValidationWithLinenumbers.ml < CasperScannerValidationInput.txt
*)

{
open Printf
}

(* definitions *)
let whitespace = [' ' '\t']+
let newline = '\n' | '\r' | "\r\n"

let digit = ['0'-'9']
let integer = digit+ (* '-'?digit+ *)
let float = (digit+ '.' digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ '.' ['e' 'E'] ['+' '-']? digit+)
            | ('.' digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ ['e' 'E'] ['+' '-']? digit+)
            | (digit+ '.' digit+) | (digit+ '.')
            | ('.' digit+)

let letter = ['a'-'z' 'A'-'Z' '_']
let id = letter(letter|digit)*

let badbraces = (['{'](whitespace | newline | '{')*['{']) | (['}'](whitespace | newline |
'}')*['}']) | (['}'](whitespace | newline)*['{'])
let badbrackets = (['['](whitespace | newline | '[')*['[']) | ([']'](whitespace | newline |
']')*[']'])
let badend = [';'](whitespace | newline)*['{'] | ['}'](whitespace | newline)*[';']
let badcommas = [','](whitespace | newline)*[',']
let badsemis =  [';'](whitespace | newline)*[';']
let badcomments = "*/" | "**"
let badcombs = badbraces | badbrackets | badend | badcommas | badsemis | badcomments

(* rules *)
rule token linecount = parse
  (* whitespace *)
```

```
    whitespace { token linecount lexbuf } | newline { token (linecount+1) lexbuf }

  (* comments *)
  | "//" { printf "Line comment starts at line %d\n" linecount; linecomment linecount lexbuf  }
  | "/*" {  printf "Comments level 0 start at line %d\n" linecount; blockcomment 0 linecount
linecount lexbuf }

  (* operators *)
  | '(' | ')'   | '[' | ']'   | '{' | '}' | ';' | ',' as op  { printf "delimiter at line %d: %c\n"
linecount op; token linecount lexbuf }
  | '_' | '?' | '+' | '-' | '*' | '/' | '%' | '^' | '>' | ">=" | '<' | "<=" | "==" | "!=" | "++" |
"--" | '=' | "_=" | "+=" | "-=" | "&&" | "||" | "!" as op { printf "operator at line %d: %s\n"
linecount op; token linecount lexbuf }

  (* keywords *)
  | "true" | "false" | "null" | "if" | "else" | "for" | "while" | "break" | "continue" | "return" |
"int" | "float" | "str" | "bool" | "void" as word {printf "keyword at line %d: %s\n" linecount
word; token linecount lexbuf}

  (* literals *)
  | integer as inum { printf "integer at line %d: %s (%d)\n" linecount inum (int_of_string inum);
token linecount lexbuf }
  | float  as fnum  { printf "float at line %d: %s (%f)\n" linecount fnum (float_of_string fnum);
token linecount lexbuf }
  | '''             { stringSQ (Buffer.create 1000) linecount linecount lexbuf }
  | '"'             { stringDQ (Buffer.create 1000) linecount linecount lexbuf }

  (* identifiers *)
  | id as lexeme    { printf "identifier at line %d: %s\n" linecount lexeme; token linecount lexbuf
}

  (* end of file *)
  | eof             { printf "EOF at line %d\n" linecount }

  (* errors *)
  | badcombs as lexeme { raise (Failure("Bad syntax: " ^ lexeme ^ " at line " ^ (string_of_int
linecount))) }
  | _ as character     { raise (Failure("Bad character: " ^ Char.escaped character ^ " at line " ^
(string_of_int linecount))) }


and stringSQ tempbuffer linecount origline = parse
    '''                 { printf "String at line %d:\n%s\n\n" origline (Buffer.contents
tempbuffer); token linecount lexbuf }
  | newline             { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringSQ tempbuffer
(linecount+1) origline lexbuf }
  | [^ '''  '\n' '\r']+ { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringSQ tempbuffer
linecount origline lexbuf }
  | eof                 { raise (Failure("Unterminated single quotes at line " ^ (string_of_int
origline))) }  (* ^ ": " ^ Buffer.contents tempbuffer *)
(*  | _ as character      { Buffer.add_string tempbuffer (Char.escaped character); stringSQ
tempbuffer linecount origline lexbuf } *)

and stringDQ tempbuffer linecount origline = parse
    '"'                 { printf "String at line %d:\n%s\n\n" origline (Buffer.contents
tempbuffer); token linecount lexbuf }
  | newline             { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringDQ tempbuffer
(linecount+1) origline lexbuf }
  | [^ '"'  '\n' '\r']+ { Buffer.add_string tempbuffer (Lexing.lexeme lexbuf); stringDQ tempbuffer
linecount origline lexbuf }
  | eof                 { raise (Failure("Unterminated double quotes at line " ^ (string_of_int
origline))) }  (* ^ ": " ^ Buffer.contents tempbuffer *)
(*  | _ as character      { Buffer.add_string tempbuffer (Char.escaped character); stringDQ
tempbuffer linecount origline lexbuf } *)
```

```
and linecomment linecount = parse
    newline { printf "End of line comment at line %d\n" linecount; token (linecount+1) lexbuf }
  | eof     { printf "EOF at line %d\n" linecount }
  | _       { linecomment linecount lexbuf }

and blockcomment level linecount origline = parse
    "*/"    { if level = 0 then (printf "Comments level %d end at line %d\n" level linecount; token
linecount lexbuf)
              else (printf "Comments level %d end at line %d\n" (level) linecount; blockcomment
(level-1) linecount origline lexbuf) }
  | "/*"    { printf "Comments level %d start at line %d\n" (level+1) linecount; blockcomment
(level+1) linecount origline lexbuf }
  | newline { blockcomment level (linecount+1) origline lexbuf }
  | eof     { raise (Failure("Unterminated comments at line " ^ (string_of_int origline))) }
  | _       { blockcomment level linecount origline lexbuf }

(* trailer *)
{
let main () =
    let cin =
        if Array.length Sys.argv > 1
        then open_in Sys.argv.(1)
        else stdin
    in
        let lexbuf = Lexing.from_channel cin in
            token 1 lexbuf
let _ = Printexc.print main ()
}
```

8.9.6 ./makesupport/testparser.sh

```
#!/bin/sh
menhir --interpret --interpret-show-cst parser.mly < ./X/menhirTestInput.txt
menhir --interpret --interpret-show-cst parser.mly < ./X/menhirTestInput.txt | grep -n "REJECT"
```

8.9.7 ./makesupport/clean.sh

```
#!/bin/bash
# Remove intermediate files except Casper ocamlbuild compilation
# File: clean.sh
# Use: ./clean.sh
# Michael Makris, mm3443
# PLT Fall 2018

rm -rf scanner.ml parser.ml parser.mli testall.log ocamlllvm *.diff
rm -f *.cmi *.cmo *.out *.o *.output *.err *.exe *.ll *.s *.dot *.png
rm -f ./tests/*.s ./tests/*.ll ./tests/*.dot ./tests/*.png ./tests/*.exe
rm -f ./examples/*.s ./examples/*.ll ./examples/*.dot ./examples/*.png ./examples/*.exe
rm -f ./llvm/*.s  ./llvm/*.ll  ./llvm/*.dot
```

8.9.8 ./makesupport/help.sh

```
#!/bin/sh
echo
echo "make options for Casper:"
echo "     make         - Compile the compiler casper.native"
echo "                    same as: make casper.native"
echo "     make all     - Compile the compiler and libraries"
echo "     make test    - Make all and then run regression suite"
echo "     make testS   - Load a version of the scanner that"
echo "                    prints out the tokens instead of feeding"
echo "                    to parser and then scans a program"
echo "     make testP   - Use menhir to evaluate some token"
echo "                    strings with the parser"
echo "     make clean   - Remove all generated files"
echo "     make cleang  - Leave compiler, remove other generated files"
echo "     make casper.tar.gz  - Create tarball"
echo "     make help    - Display this list of options"
echo
```

8.10 Casper Program Compiling Scripts

8.10.1 ./casper.sh

```
#!/bin/bash
# Compile and run programs for Casper
# File: casper.sh
# Michael Makris, mm3443
# PLT Fall 2018

echo "usage: ./casper.sh <program filename with no extension>"
echo
echo "---> running casper.native           --->"
echo "---> COMPILE ERRORS COME NEXT        --->"
./casper.native < $1.goo > $1.ll
echo
echo "-------------------------------------->"
echo "---> llc command                      --->"
llc -relocation-model=pic $1.ll > $1.s

echo "---> compiling stdlib object file    --->"
cc -c -Wall stdlib.c
echo "---> linking files                    --->"
cc -o $1.exe $1.s stdlib.o -lm

echo "---> removing intermediate files     --->"
rm -f ./$1.s ./$1.ll

echo "---> running executable" $1.exe " --->"
echo
./$1.exe
```

8.10.2 ./casperc.sh

```
#!/bin/bash
# Compile programs for Casper
# File: casperc.sh
# Michael Makris, mm3443
# PLT Fall 2018

echo "usage: ./casper.sh <program filename with no extension>"
echo
echo "---> running casper.native           --->"
echo "---> COMPILE ERRORS COME NEXT        --->"
./casper.native < $1.goo > $1.ll
echo
echo "-------------------------------------->"
echo "---> llc command                      --->"
llc -relocation-model=pic $1.ll > $1.s

echo "---> compiling stdlib object file    --->"
cc -c -Wall stdlib.c
echo "---> linking files                    --->"
cc -o $1.exe $1.s stdlib.o -lm

echo "---> removing intermediate files     --->"
rm -f ./$1.s ./$1.ll

echo "---> executable" $1.exe " has been created if there were no errors above"
echo
```

8.10.3 ./casperll.sh

```
#!/bin/bash
# Compile and run programs for Casper
# create png of main and other ll dot files for debugging
# File: casperll.sh
# Michael Makris, mm3443
# PLT Fall 2018

echo "usage: ./casperll.sh <program filename with no extension>"
echo
echo "---> running casper.native           --->"
echo "---> COMPILE ERRORS COME NEXT        --->"
./casper.native < $1.goo > $1.ll
echo
echo "--------------------------------------->"
echo "---> llc command                     --->"
llc -relocation-model=pic $1.ll > $1.s

echo "---> compiling stdlib object file    --->"
cc -c -Wall stdlib.c
echo "---> linking files                   --->"
cc -o $1.exe $1.s stdlib.o -lm

echo "---> main() control-flow graph       --->"
opt -dot-cfg $1.ll
dot -Tpng cfg.main.dot > $1.png
echo "---> see" $1.png "--->"
echo "---> running executable" $1.exe " --->"
echo
./$1.exe
```

8.11 Clang Script

8.11.1 ./clang/clangviz.sh

```
#!/bin/bash
# View clang llvm output
# File: clangviz.sh
# Michael Makris, mm3443
# PLT Fall 2018

echo "usage: ./clangviz.sh <C program filename with no extension>"

echo "---> running clang  --->"
clang -emit-llvm -S -c $1.c -o - > $1.ll

echo "---> running opt  --->"
opt -dot-cfg $1.ll

echo "---> main() control-flow graph --->"
dot -Tpng cfg.main.dot > $1.png

echo "---> cleanup --->"
rm cfg.main.dot $1.ll

echo "---> see" $1.png
```

8.12 Test Suite

This is the set of test programs for the test suite described in Section 6.2.

```
./tests/ fail-addassign1.goo

int main()
{
  int i;
  bool b;

  i = 42;
  b = true;
  i += b; /* Fail: add-assigning a bool to an integer */
}

Output
Fatal error: exception Failure("illegal assignment int+=bool in i += b")


./tests/ fail-addassign2.goo

int main()
{
  int i;
  str s;

  i = 42;
  s = "hello";
  i += s; /* Fail: add-assigning a string to an integer */
}

Output
Fatal error: exception Failure("illegal assignment int+=str in i += s")


./tests/ fail-addassign3.goo

int main()
{
  int i;
  float f;

  i = 42;
  f = 3.14;
  i += f; /* Fail: add-assigning a float to an integer */
}

Output
Fatal error: exception Failure("illegal assignment int+=float in i += f")


./tests/ fail-addassign4.goo

int main()
{
  float f;
  bool b;

  f = 3.14;
  b = true;
  f += b; /* Fail: add-assigning a bool to an float */
```

```
}

Output
Fatal error: exception Failure("illegal assignment float+=bool in f += b")


./tests/ fail-addassign5.goo

int main()
{
  float f;
  str s;

  f = 3.14;
  s = "hello";
  f += s; /* Fail: add-assigning a string to a float */
}

Output
Fatal error: exception Failure("illegal assignment float+=str in f += s")


./tests/ fail-addassign6.goo

int main()
{
  float f;
  int i;

  f = 3.14;
  i = 42;
  f += i; /* Fail: add-assigning an integer to a float */
}

Output
Fatal error: exception Failure("illegal assignment float+=int in f += i")


./tests/ fail-assign10.goo

int main()
{
  bool b;
  float f;

  b = true;
  f = 3.14;
  b = f; /* Fail: assigning a float to a bool */
}

Output
Fatal error: exception Failure("illegal assignment bool=float in b = f")


./tests/ fail-assign11.goo

int main()
{
  bool b;
  str s;

  b = true;
  s = "hello";
  b = s; /* Fail: assigning a string to a bool */
```

```
}

Output
Fatal error: exception Failure("illegal assignment bool=str in b = s")


./tests/ fail-assign12.goo

int main()
{
  bool b;
  int i;

  b = true;
  i = 42;
  b = i; /* Fail: assigning an integer to bool */
}

Output
Fatal error: exception Failure("illegal assignment bool=int in b = i")


./tests/ fail-assign13.goo

int main()
{
  int i;
  bool b;

  b = 48; /* Fail: assigning an integer to a bool */
}

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


./tests/ fail-assign14.goo

void myvoid()
{
  return;
}

int main()
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}

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



./tests/ fail-assign1.goo

int main()
{
  int i;
  bool b;

  i = 42;
  b = true;
  i = b; /* Fail: assigning a bool to an integer */
}

Output
Fatal error: exception Failure("illegal assignment int=bool in i = b")


./tests/ fail-assign2.goo

int main()
{
  int i;
  str s;

  i = 42;
  s = "hello";
  i = s; /* Fail: assigning a string to an integer */
}

Output
Fatal error: exception Failure("illegal assignment int=str in i = s")


./tests/ fail-assign3.goo

int main()
{
  int i;
  float f;

  i = 42;
  f = 3.14;
  i = f; /* Fail: assigning a float to an integer */
}

Output
Fatal error: exception Failure("illegal assignment int=float in i = f")


./tests/ fail-assign4.goo

int main()
{
  float f;
  bool b;

  f = 3.14;
  b = true;
  f = b; /* Fail: assigning a bool to a float */
}

Output
Fatal error: exception Failure("illegal assignment float=bool in f = b")


./tests/ fail-assign5.goo

int main()
{
  float f;
  str s;

  f = 3.14;
  s = "hello";
  f = s; /* Fail: assigning a string to a float */
}

Output
Fatal error: exception Failure("illegal assignment float=str in f = s")


./tests/ fail-assign6.goo

int main()
{
  float f;
  int i;

  f = 3.14;
  i = 42;
  f = i; /* Fail: assigning an integer to a float */
}

Output
Fatal error: exception Failure("illegal assignment float=int in f = i")


./tests/ fail-assign7.goo

int main()
{
  str s;
  int i;

  s = 'hello';
  i = 42;
  s = i; /* Fail: assigning an integer to a string */
}

Output
Fatal error: exception Failure("illegal assignment str=int in s = i")


./tests/ fail-assign8.goo

int main()
{
  str s;
  float f;

  s = "hello";
  f = 3.14;
  s = f; /* Fail: assigning a float to a string */
}

Output
Fatal error: exception Failure("illegal assignment str=float in s = f")


./tests/ fail-assign9.goo

int main()
{
  str s;
  bool b;

  s = "hello";
  b = true;
  s = b; /* Fail: assigning a bool to a string */
}

Output
Fatal error: exception Failure("illegal assignment str=bool in s = b")


./tests/ fail-break1.goo

int main()
{
  while(true) {
    break;
    continue;
  }
  return 0;
}

Output
Fatal error: exception Failure("nothing may follow a break")


./tests/ fail-continue1.goo

int main()
{
  while(true) {
    continue;
    break;
  }
  return 0;
}

Output
Fatal error: exception Failure("nothing may follow a continue")


./tests/ fail-dead1.goo

int main()
{
  int i;

  i = 15;
  return i;
  i = 32; /* Error: code after a return */
}

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



./tests/ fail-dead2.goo

int main()
{
  int i;

  {
    i = 15;
    return i;
  }
  i = 32; /* Error: code after a return */
```

```
}

Output
Fatal error: exception Failure("Bad syntax: ;

  {")



./tests/ fail-dead3.goo

int main()
{
  int i;

  i = 15;
  return i;

  i = 32; /* Error: code after a return */
}

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



./tests/ fail-dec1.goo

int main()
{
  float f;

  f = --1.1; /* Fail: DEC a float */
}

Output
Fatal error: exception Failure("illegal binary operator float - int in 1.1 - 1")


./tests/ fail-dec2.goo

int main()
{
  float f;

  f = 1.1;
  f = --f; /* Fail: DEC a float */
}

Output
Fatal error: exception Failure("illegal binary operator float - int in f - 1")


./tests/ fail-dec3.goo

int main()
{
  str s;

  s = --"hello"; /* Fail: DEC a string */
}

Output
Fatal error: exception Failure("illegal binary operator str - int in hello - 1")
```

./tests/ fail-dec4.goo

```
int main()
{
  str s;

  s = 'hello';
  s = --s; /* Fail: DEC a string */
}
```

Output

```
Fatal error: exception Failure("illegal binary operator str - int in s - 1")
```

./tests/ fail-dec5.goo

```
int main()
{
  bool b;

  b = --true; /* Fail: DEC a bool */
}
```

Output

```
Fatal error: exception Failure("illegal binary operator bool - int in true - 1")
```

./tests/ fail-dec6.goo

```
int main()
{
  bool b;

  b = --b; /* Fail: DEC a bool */
}
```

Output

```
Fatal error: exception Failure("illegal binary operator bool - int in b - 1")
```

./tests/ fail-dup1.goo

```
int a;
bool b;

void foo(int c, bool c) {
  printi(c);
  printb(c);
}

int main() {
  a = 1;
  b = true;
  foo(a, b);
  return 0;
}
```

Output

```
Fatal error: exception Failure("duplicate formal c")
```

./tests/ fail-dup2.goo

```
int a;
bool a;

void foo(int c, bool d) {
  printi(c);
  printb(d);
}

int main() {
  a = 1;
  b = true;
  foo(a, b);
  return 0;
}

Output
Fatal error: exception Failure("duplicate global a")
```

```
./tests/ fail-dup3.goo

int a;
bool b;

void a(int c, bool d) {
  printi(c);
  printb(d);
}

int main() {
  int a;
  int a;

  a = 1;
  b = true;
  a(a, b);
  return 0;
}

Output
Fatal error: exception Failure("duplicate local a")
```

```
./tests/ fail-dup4.goo

int a;
bool b;

void a(int c, bool d) {
  printi(c);
  printb(d);
}

int main() {
  int a;
  bool a;

  a = 1;
  b = true;
  a(a, b);
  return 0;
}
```

```
Output
Fatal error: exception Failure("duplicate local a")
```

```
./tests/ fail-expr1.goo

int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}

int main()
{
  return 0;
}

Output
Fatal error: exception Failure("illegal binary operator bool + int in d + a")
```

```
./tests/ fail-expr2.goo

int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
}

int main()
{
  return 0;
}

Output
Fatal error: exception Failure("illegal binary operator bool + int in b + a")
```

```
./tests/ fail-expr3.goo

int a;
float b;

void foo(int c, float d)
{
  int d;
  float e;
  b + a; /* Error: float + int */
}
```

```
int main()
{
  return 0;
}

Output
Fatal error: exception Failure("illegal binary operator float + int in b + a")
```

```
./tests/ fail-float1.goo

int main()
{
  -3.5 && 1; /* Float with AND? */
  return 0;
}

Output
Fatal error: exception Failure("illegal binary operator float && int in -3.5 && 1")
```

```
./tests/ fail-float2.goo

int main()
{
  -3.5 && 2.5; /* Float with AND? */
  return 0;
}

Output
Fatal error: exception Failure("illegal binary operator float && float in -3.5 && 2.5")
```

```
./tests/ fail-float3.goo

void testfloat(float a, float b)
{
  printf(a + b);
  printf(a - b);
  printf(a * b);
  printf(a / b);
  printb(a == b);
  printb(a == a);
  printb(a != b);
  printb(a != a);
  printb(a > b);
  printb(a >= b);
  printb(a < b);
  printb(a <= b);


  printf(a - (10.0 * b));
  printf(a - (1e1 * b));
  printf(a - (1.0e1 * b));
  printf(a - (1.0e1.0 * b));  /* e1.0 invalid */
}

int main()
{
  float c;
  float d;
  c = 42.0;
  d = 3.14159;

  testfloat(c, d);

  testfloat(d, d);

  return 0;
}

Output
Fatal error: exception Stdlib.Parsing.Parse_error


./tests/ fail-fmod.goo

/* Should be illegal to redefine */
void fmod() {}

Output
Fatal error: exception Failure("function fmod may not be defined")



./tests/ fail-for1.goo

int main()
{
  int i;
  for ( ; true ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

  for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}

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



./tests/ fail-for2.goo

int main()
{
  int i;

  for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}

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



./tests/ fail-for3.goo
```

```
int main()
{
  int i;

  for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */

  return 0;
}

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



./tests/ fail-for4.goo

int main()
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}

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



./tests/ fail-for5.goo

int main()
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
  }

  return 0;
}

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



./tests/ fail-func10.goo

int a;
bool b;

void a(int c, bool d) {
  printi(c);
  printb(d);
}

int main() {
  a = 1;
  b = true;
  foo(a, b);
  return 0;
}
```

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


./tests/ fail-func1.goo

int foo() {}

int bar() {}

int baz() {}

void bar() {} /* Error: duplicate function bar */

int main()
{
  return 0;
}

Output
Fatal error: exception Failure("duplicate function bar")



./tests/ fail-func2.goo

int foo(int a, bool b, int c) { }

void bar(int a, bool b, int a) {} /* Error: duplicate formal a in bar */

int main()
{
  return 0;
}

Output
Fatal error: exception Failure("duplicate formal a")



./tests/ fail-func3.goo

int foo(int a, bool b, int c) { }

void bar(int a, void b, int c) {} /* Error: illegal void formal b */

int main()
{
  return 0;
}

Output
Fatal error: exception Failure("illegal void formal b")



./tests/ fail-func4.goo

int foo() {}

void bar() {}

int prints() {} /* Should not be able to define prints */
```

```
void baz() {}

int main()
{
  return 0;
}

Output
Fatal error: exception Failure("function prints may not be defined")



./tests/ fail-func5.goo

int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;

  return 0;
}

int main()
{
  return 0;
}

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



./tests/ fail-func6.goo

void foo(int a, bool b)
{
}

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

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



./tests/ fail-func7.goo

void foo(int a, bool b)
{
}

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

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

```
./tests/ fail-func8.goo

void foo(int a, bool b)
{
}

void bar()
{
}

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

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

```
./tests/ fail-func9.goo

void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, 42); /* Fail: int, not bool */
}

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

```
./tests/ fail-global1.goo

int c;
bool b;
void a; /* global variables should not be void */


int main()
{
  return 0;
}

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

```
./tests/ fail-global2.goo

int b;
bool c;
```

```
int a;
int b; /* Duplicate global variable */

int main()
{
  return 0;
}

Output
Fatal error: exception Failure("duplicate global b")
```

```
./tests/ fail-if1.goo

int main()
{
  if (true) {}
  if (false) {} else {}
  if (42) {} /* Error: non-bool predicate */
}

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

```
./tests/ fail-if2.goo

int main()
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}

Output
Fatal error: exception Failure("undeclared identifier foo")
```

```
./tests/ fail-if3.goo

int main()
{
  if (true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}

Output
Fatal error: exception Failure("undeclared identifier bar")
```

```
./tests/ fail-inc1.goo

int main()
{
  float f;

  f = ++1.1; /* Fail: INC a float */
```

```
}

Output
Fatal error: exception Failure("illegal binary operator float + int in 1.1 + 1")


./tests/ fail-inc2.goo

int main()
{
  float f;

  f = 1.1;
  f = ++f; /* Fail: INC a float */
}

Output
Fatal error: exception Failure("illegal binary operator float + int in f + 1")


./tests/ fail-inc3.goo

int main()
{
  str s;

  s = ++"hello"; /* Fail: INC a string */
}

Output
Fatal error: exception Failure("illegal binary operator str + int in hello + 1")


./tests/ fail-inc4.goo

int main()
{
  str s;

  s = 'hello';
  s = ++s; /* Fail: INC a string */
}

Output
Fatal error: exception Failure("illegal binary operator str + int in s + 1")


./tests/ fail-inc5.goo

int main()
{
  bool b;

  b = ++true; /* Fail: INC a bool */
}

Output
Fatal error: exception Failure("illegal binary operator bool + int in true + 1")


./tests/ fail-inc6.goo

int main()
{
  bool b;

  b = ++b; /* Fail: INC a bool */
}

Output
Fatal error: exception Failure("illegal binary operator bool + int in b + 1")


./tests/ fail-local1.goo

void a(int c, bool d) {
  printi(c);
  printb(d);
}

int main() {
  int a;
  bool c;

  a = 1;
  b = true;
  a(a, b);
  return 0;
}

Output
Fatal error: exception Failure("undeclared identifier b")


./tests/ fail-negation1.goo

int main()
{
  bool b;

  b = -true; /* Fail: negating a bool */
}

Output
Fatal error: exception Failure("illegal unary operator -bool in -true")


./tests/ fail-negation2.goo

int main()
{
  bool b;
  bool nb;

  b = true;
  nb = -b;  /* Fail: negating a bool */
}

Output
Fatal error: exception Failure("illegal unary operator -bool in -b")


./tests/ fail-negation3.goo

int main()
{
  str s;
  s = -"hello"; /* Fail: negating a string */
}
```

```
Output
Fatal error: exception Failure("illegal unary operator -str in -hello")
```

```
./tests/ fail-negation4.goo

int main()
{
  str s;
  str ns;

  s = "hello";
  ns = -s;  /* Fail: negating a string */
}
```

```
Output
Fatal error: exception Failure("illegal unary operator -str in -s")
```

```
./tests/ fail-nomain.goo
```

```
Output
Fatal error: exception Failure("unrecognized function main")
```

```
./tests/ fail-not1.goo

int main()
{
  int i;

  i = !1; /* Fail: NOT an int */
}
```

```
Output
Fatal error: exception Failure("illegal unary operator !int in !1")
```

```
./tests/ fail-not2.goo

int main()
{
  int i;
  int ni;

  i = 1;
  ni = !i;  /* Fail: NOT an int */
}
```

```
Output
Fatal error: exception Failure("illegal unary operator !int in !i")
```

```
./tests/ fail-not3.goo

int main()
{
  str s;
  s = !"hello"; /* Fail: NOT a string */
}
```

```
Output
Fatal error: exception Failure("illegal unary operator !str in !hello")
```

./tests/ fail-not4.goo

```
int main()
{
  str s;
  str ns;

  s = "hello";
  ns = !s;  /* Fail: NOT a string */
}
```

```
Output
Fatal error: exception Failure("illegal unary operator !str in !s")
```

./tests/ fail-not5.goo

```
int main()
{
  float f;

  f = !3.14e-1;   /* Fail: NOT a float */
}
```

```
Output
Fatal error: exception Failure("illegal unary operator !float in !3.14e-1")
```

./tests/ fail-not6.goo

```
int main()
{
  float f;
  float nf;

  f = 3.14e-1;
  nf = !f;  /* Fail: NOT a float */
}
```

```
Output
Fatal error: exception Failure("illegal unary operator !float in !f")
```

./tests/ fail-printb.goo

```
/* Should be illegal to redefine */
void printb() {}
```

```
Output
Fatal error: exception Failure("function printb may not be defined")
```

./tests/ fail-printbig.goo

```
/* Should be illegal to redefine */
void printbig() {}
```

```
Output
Fatal error: exception Failure("function printbig may not be defined")
```

./tests/ fail-printc.goo

```
/* Should be illegal to redefine */
void printc() {}
```

```
Output
Fatal error: exception Failure("function printc may not be defined")
```

./tests/ fail-printf.goo

```
/* Should be illegal to redefine */
void printf() {}
```

```
Output
Fatal error: exception Failure("function printf may not be defined")
```

./tests/ fail-printi.goo

```
/* Should be illegal to redefine */
void printi() {}
```

```
Output
Fatal error: exception Failure("function printi may not be defined")
```

./tests/ fail-prints.goo

```
/* Should be illegal to redefine */
void prints() {}
```

```
Output
Fatal error: exception Failure("function prints may not be defined")
```

./tests/ fail-printsnl.goo

```
/* Should be illegal to redefine */
void printsnl() {}
```

```
Output
Fatal error: exception Failure("function printsnl may not be defined")
```

./tests/ fail-rand.goo

```
/* Should be illegal to redefine */
void rand() {}
```

```
Output
Fatal error: exception Failure("function rand may not be defined")
```

```
./tests/ fail-return1.goo

int main()
{
  return true; /* Should return int */
}

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


./tests/ fail-return2.goo

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

int main()
{
  return 42;
}

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



./tests/ fail-str1.goo

int main()
{
  str s;
  float f;

  s = "hello";
  f = 3.14;
  s _= b;
}

Output
Fatal error: exception Failure("undeclared identifier b")


./tests/ fail-str3.goo

int main()
{
  str s;
  bool b;

  s = "hello";
  b = true;
  s = s ? b; /* Fail: charat a bool to a string */
}

Output
Fatal error: exception Failure("illegal binary operator str ? bool in s ? b")


./tests/ fail-str4.goo
```

```
int main()
{
  str s;
  float f;

  s = "hello";
  f = 1.0;
  s = s ? f; /* Fail: ccharat a float to a string */
}
```

```
Output
Fatal error: exception Failure("illegal binary operator str ? float in s ? f")
```

./tests/ fail-str7.goo

```
int main()
{
  str s;
  int i;
  int j;

  s = "hello";
  i = 1;
  j = 0;
  s = s ? i ? j; /* Fail: ccharat a float to a string */
}
```

```
Output
Fatal error: exception Failure("illegal binary operator int ? int in i ? j")
```

./tests/ fail-subassign1.goo

```
int main()
{
  int i;
  bool b;

  i = 42;
  b = true;
  i -= b; /* Fail: sub-assigning a bool to an integer */
}
```

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

./tests/ fail-subassign2.goo

```
int main()
{
  int i;
  str s;

  i = 42;
  s = "hello";
  i -= s; /* Fail: sub-assigning a string to an integer */
}
```

```
Output
Fatal error: exception Failure("illegal assignment int-=str in i -= s")
```

./tests/ fail-subassign3.goo

```
int main()
{
  int i;
  float f;

  i = 42;
  f = 3.14;
  i -= f; /* Fail: sub-assigning a float to an integer */
}
```

Output

```
Fatal error: exception Failure("illegal assignment int-=float in i -= f")
```

./tests/ fail-subassign4.goo

```
int main()
{
  float f;
  bool b;

  f = 3.14;
  b = true;
  f -= b; /* Fail: sub-assigning a bool to an float */
}
```

Output

```
Fatal error: exception Failure("illegal assignment float-=bool in f -= b")
```

./tests/ fail-subassign5.goo

```
int main()
{
  float f;
  str s;

  f = 3.14;
  s = "hello";
  f -= s; /* Fail: sub-assigning a string to a float */
}
```

Output

```
Fatal error: exception Failure("illegal assignment float-=str in f -= s")
```

./tests/ fail-subassign6.goo

```
int main()
{
  float f;
  int i;

  f = 3.14;
  i = 42;
  f -= i; /* Fail: sub-assigning an integer to a float */
}
```

Output

```
Fatal error: exception Failure("illegal assignment float-=int in f -= i")
```

./tests/ fail-while1.goo

```
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (42) { /* Should be boolean */
    i = i + 1;
  }

}
```

Output

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

./tests/ fail-while2.goo

```
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (true) {
    foo(); /* foo undefined */
  }

}
```

Output

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

./tests/ test-add1.goo

```
int add(int x, int y)
{
  return x + y;
}

int main()
{
  printinl( add(17, 25) );
  return 0;
}
```

Output

```
42
```

./tests/ test-addassign.goo

```
int main()
```

```
{
    int y;
    int z;
    float x;
    float w;

    y=2; z=3;
    printinl(y);
    printinl(z);
    y+=5;
    printinl(y);
    y+=z;
    printinl(y);

    x=3.14e-5;
    w=5.1e5;
    printfnl(x);
    printfnl(w);
    x+=1.1;
    printfnl(x);
    x+=w;
    printfnl(x);
    x+=-10e10;
    printfnl(x);
    return 0;
}
```

```
Output
2
3
7
10
0.0000314000
510000.0000000000
1.1000314000
510001.1000314000
-99999489998.8999633789
```

./tests/ test-arith1.goo

```
int main()
{
  printinl(39 + 3);
  return 0;
}
```

```
Output
42
```

./tests/ test-arith2.goo

```
int main()
{
  printinl(1 + 2 * 3 + 4);
  return 0;
}
```

```
Output
11
```

```
./tests/ test-arith3.goo

int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a + 5;
  printinl(a);
  return 0;
}

Output
47
```

```
./tests/ test-assign.goo

int main()
{
    int i;
    int ii;
    float f;
    float ff;
    bool b;
    bool bb;
    str s;
    str ss;

    i = 1;
    ii = i;
    printinl(i);
    printinl(ii);

    f = 3.14;
    ff = f;
    printfnl(f);
    printfnl(ff);

    s = "hello

    X's
";
    ss = s;
    prints(s);
    prints(ss);

    s = 'hello
';
    ss = s;
    prints(s);
    prints(ss);

    b = true;
    bb = b;
    printbnl(b);
    printbnl(bb);
    return 0;
}

Output
1
1
3.1400000000
3.1400000000
hello

    X's
hello

    X's
hello
hello
true
true
```

```
./tests/ test-break1.goo

int main()
{   printinl(1);
    while(true){
        if(true)
           break;
        else
           printinl(2);
    }
    printinl(3);
    return 0;
}

Output
1
3
```

```
./tests/ test-break2.goo

int main()
{
   int i;
   for(i = 0; i < 3; i=++i) {
       while(true){
          printinl(1);
          break;
       }
       printinl(2);
   }
   printinl(3);
   return 0;
}
Output
1
2
1
2
1
2
```

```
3
```

./tests/ test-conassign1.goo

```
int main()
{
   str s;
   str x;

   s = "hello";
   s _= " world";
   printsnl(s);

   s _= "!";
   printsnl(s);

   x = s;
   printsnl(x);

   s _= x;
   printsnl(s);

   s = x;
   printsnl(s);


   return 0;
}
```

Output

```
hello world
hello world!
hello world!
hello world!hello world!
hello world!
```

./tests/ test-conassign2.goo

```
int main()
{
    str s;
    str a;
    int i;
    float f;
    bool b;
    bool c;

    s = "a";
    a = "b";

    i=1;f=3.14;b=true;c=false;

    s = s _ a;
    s = s _ i;
    s = s _ b;
    s = s _ c;
    s = s _ f;

    s _= a;
    s _= i;
    s _= b;
    s _= c;
    s _= f;

    printsnl(s _ "
");
    return 0;
}
Output
ab1truefalse3.14b1truefalse3.14
```

```
./tests/ test-cont1.goo

int main()
{
    bool b;
    b = true;
    while(b) {
        printinl(1);
        b = false;
        if(true) continue;
        printinl(2);
    }
    printinl(3);
    return 0;
}
Output
1
3
```

```
./tests/ test-dec.goo

int main()
{
    int i;

    printinl(--1); /* 0 */
    printinl(--0); /* -1 */

    i = -1;
    printinl(i);   /* -1 */
    printinl(--i); /* -2 */
    printinl(--i); /* -2 */

    i = --10;
    printinl(i);   /* 9 */
    i=--i;
    printinl(i);   /* 8 */
    --i;
    printinl(i);   /* 8 */

    return 0;
}

Output
0
-1
-1
-2
-2
```

```
9
8
8
```

```
./tests/ test-dountil1.goo

int main()
{
  int i;
  i = 5;
  do{
    printinl(i);
    i = i - 1;
  } until (i == 0) ;
  printinl(42);
  return 0;
}

Output
5
4
3
2
1
42
```

```
./tests/ test-dountil2.goo

int foo(int a)
{
  int j;
  j = 0;
  do {
    j = j + 2;
    a = a - 1;
  } until (a == 0);
  return j;
}

int main()
{
  printinl(foo(7));
  return 0;
}

Output
14
```

```
./tests/ test-dountil3.goo

int main()
{
  bool i;
  i = true;
  do {
    printbnl(i);
    i = false;
  } until (!i);
  printinl(42);
  return 0;
}

Output
true
42
```

./tests/ test-dowhile1.goo

```
int main()
{
  int i;
  i = 5;
  do {
    printinl(i);
    i = i - 1;
  } while (i > 0);
  printinl(42);
  return 0;
}

Output
5
4
3
2
1
42
```

./tests/ test-dowhile2.goo

```
int foo(int a)
{
  int j;
  j = 0;
  do {
    j = j + 2;
    a = a - 1;
  } while (a > 0);
  return j;
}

int main()
{
  printinl(foo(7));
  return 0;
}

Output
14
```

./tests/ test-dowhile3.goo

```
int main()
{
  bool i;
  i = true;
  do {
    printbnl(i);
    i = false;
  } while (i);
  printinl(42);
  return 0;
}

Output
true
42
```

./tests/ test-dup.goo

```
int a;
bool b;

void a(int c, bool d) {
  printinl(c);
  printbnl(d);
}

int main() {
  a = 1;
  b = true;
  a(a, b);
  return 0;
}

Output
1
true
```

./tests/ test-fib.goo

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

int main()
{
  printinl(fib(0));
  printinl(fib(1));
  printinl(fib(2));
  printinl(fib(3));
  printinl(fib(4));
  printinl(fib(5));
  return 0;
}

Output
1
1
2
3
5
8
```

./tests/ test-float1.goo

```
int main()
{
  float a;
  a = 3.14159267;
  printfnl(a);
  return 0;
}
```

Output

```
3.1415926700
```

./tests/ test-float2.goo

```
int main()
{
  float a;
  float b;
  float c;
  a = 3.14159267;
  b = -2.71828;
  c = a + b;
  printfnl(c);
  return 0;
}
```

Output

```
0.4233126700
```

./tests/ test-float3.goo

```
void testfloat(float a, float b)
{
  printfnl(a + b);
  printfnl(a - b);
  printfnl(a * b);
  printfnl(a / b);
  printbnl(a == b);
  printbnl(a == a);
  printbnl(a != b);
  printbnl(a != a);
  printbnl(a > b);
  printbnl(a >= b);
  printbnl(a < b);
  printbnl(a <= b);


  printfnl(a - (10.0 * b));
  printfnl(a - (1e1 * b));
  printfnl(a - (1.0e1 * b));
  printfnl(a - (1.0e1 * b));
  printfnl(a - (10.0e-1 * b));
}

int main()
{
  float c;
  float d;

  c = 42.0;
  d = 3.14159;

  testfloat(c, d);

  testfloat(d, d);

  return 0;
}

Output
45.1415900000
38.8584100000
131.9467800000
13.3690265121
false
true
true
false
true
true
false
false
10.5841000000
10.5841000000
10.5841000000
10.5841000000
38.8584100000
6.2831800000
0.0000000000
9.8695877281
1.0000000000
true
true
false
false
false
true
false
true
-28.2743100000
-28.2743100000
-28.2743100000
-28.2743100000
0.0000000000


./tests/ test-for1.goo

int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    printinl(i);
  }
  printinl(42);
  return 0;
}

Output
0
1
2
3
4
```

```
42
```

./tests/ test-for2.goo

```
int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    printinl(i);
    i = i + 1;
  }
  printinl(42);
  return 0;
}
```

Output

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

./tests/ test-for3.goo

```
int main()
{
  int i;
  for (i=0; i<5; i+=1) {
    printinl(i);
  }
  printinl(42);
  return 0;
}
```

Output

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

./tests/ test-func1.goo

```
int add(int a, int b)
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  printinl(a);
  return 0;
}
```

```
Output
42
```

./tests/ test-func2.goo

```
/* Bug noticed by Pin-Chin Huang */
/* Expressions in formals execute right to left */

int fun(int x, int y)
{
  return 0;
}

int main()
{
  int i;
  i = 1;

  fun( i = i+1, i = 4);

  printinl(i);  /* 4 + 1 */
  return 0;
}
```

```
Output
5
```

./tests/ test-func3.goo

```
void printem(int a, int b, int c, int d)
{
  printinl(a);
  printinl(b);
  printinl(c);
  printinl(d);
}

int main()
{
  printem(42,17,192,8);
  return 0;
}
```

```
Output
42
17
192
8
```

./tests/ test-func4.goo

```
int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

int main()
{
  int d;
  d = add(52, 10);
  printinl(d);
  return 0;
}

Output
62
```

```
./tests/ test-func5.goo

int foo(int a)
{
  return a;
}

int main()
{
  return 0;
}

Output
```

```
./tests/ test-func6.goo

void foo() {}

int bar(int a, bool b, int c) { return a + c; }

int main()
{
  printinl(bar(17, false, 25));
  return 0;
}

Output
42
```

```
./tests/ test-func7.goo

int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  printinl(a);
  return 0;
}
```

```
Output
115
```

./tests/ test-func8.goo

```
void foo(int a)
{
  printinl(a + 3);
}

int main()
{
  foo(40);
  return 0;
}
```

```
Output
43
```

./tests/ test-func9.goo

```
void foo(int a)
{
  printinl(a + 3);
  return;
}

int main()
{
  foo(40);
  return 0;
}
```

```
Output
43
```

./tests/ test-gcd2.goo

```
int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}

int main()
{
  printinl(gcd(14,21));
  printinl(gcd(8,36));
  printinl(gcd(99,121));
  return 0;
}
```

```
Output
7
4
11
```

```
./tests/ test-gcd.goo

int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

int main()
{
  printinl(gcd(2,14));
  printinl(gcd(3,15));
  printinl(gcd(99,121));
  return 0;
}

Output
2
3
11
```

```
./tests/ test-global1.goo

int a;
int b;

void printa()
{
  printinl(a);
}

void printbnlb()
{
  printinl(b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  printa();
  printbnlb();
  incab();
  printa();
  printbnlb();
  return 0;
}

Output
42
21
43
```

```
22
```

./tests/ test-global2.goo

```
bool i;

int main()
{
  int i; /* Should hide the global i */

  i = 42;
  printinl(i + i);
  return 0;
}
```

Output

```
84
```

./tests/ test-global3.goo

```
int i;
bool b;
int j;

int main()
{
  i = 42;
  j = 10;
  printinl(i + j);
  return 0;
}
```

Output

```
52
```

./tests/ test-global5.goo

```
int b;
bool f;
float s;
str i;

int main() {
  /* Should hide the globals */
  int i;
  bool b;
  float f;
  str s;

  i = 42;
  printinl(i + i);
  b = true;
  printbnl(b && b);
  f = 3.14;
  printfnl(f/f);
  s = "hello
";
  prints(s);
  return 0;
}

Output
84
true
1.0000000000
hello



./tests/ test-global6.goo

int i;
bool b;
int j;
float f1;
float f2;
str s;

int main()
{
  i = 42;
  j = 10;
  printinl(i + j);
  f1 = 3.14;
  f2 = f1;
  printfnl(f1 + f2);
  s = "hello
";
  prints(s);

  return 0;
}

Output
52
6.2800000000
hello


./tests/ test-hello.goo

int main()
{
  printinl(42);
  printinl(71);
  printinl(1);
  return 0;
}

Output
42
71
1



./tests/ test-helloworld.goo

int main() {
    printinl(42);
    printfnl(3.14);
    prints("Hello world!
");
    printcnl(`Q`);
    printbnl(true);
    return 0;
}
Output
42
3.1400000000
Hello world!
Q
true
```

./tests/ test-if1.goo

```
int main()
{
  if (true) printinl(42);
  printinl(17);
  return 0;
}
```

Output

```
42
17
```

./tests/ test-if2.goo

```
int main()
{
  if (true) printinl(42); else printinl(8);
  printinl(17);
  if (false) {printinl(42);} else {printinl(8);}
  printinl(17);
  return 0;
}
```

Output

```
42
17
8
17
```

./tests/ test-if3.goo

```
int main()
{
  if (false) printinl(42);
  printinl(17);
  return 0;
}
```

Output

```
17
```

./tests/ test-if4.goo

```
int main()
```

```
{
  if (false) printinl(42); else printinl(8);
  printinl(17);
  return 0;
}

Output
8
17
```

./tests/ test-if5.goo

```
int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 printinl(cond(true));
 printinl(cond(false));
 return 0;
}

Output
42
17
```

./tests/ test-if6.goo

```
int cond(bool b)
{
  int x;
  x = 10;
  if (b)
    if (x == 10)
        x = 42;
    else
        x = 17;
  return x;
}

int main()
{
 printinl(cond(true));
 printinl(cond(false));
 return 0;
}

Output
42
10
```

```
./tests/ test-if7.goo

int cond(bool b)
{
  int x;
  x = 10;
  if (b) {
    if (x == 10)
      x = 42;
  }
  else
    x = 17;
  return x;
}

int main()
{
 printinl(cond(true));
 printinl(cond(false));
 return 0;
}

Output
42
17
```

```
./tests/ test-inc1.goo

int main()
{
    int i;

    i = 1;
    printinl(i);
    printinl(++i);
    i = ++1;
    printinl(i);
    i=++i;
    printinl(i);
    ++i;
    printinl(i);

    return 0;
}

Output
1
2
2
3
3
```

```
./tests/ test-inc2.goo

int main()
{
    int i;

    i = 1;
    printinl(++i);
    i+=1;
    printinl(i);

    if(i > 1) {
        for(i = 0; i < 5; i+=1){
          printinl(i);
        }
    }

    return 0;
}

Output
2
2
0
1
2
3
4


./tests/ test-int1.goo

int main()
{
  int a;
  a = 3;
  printinl(a);
  return 0;
}

Output
3


./tests/ test-int2.goo

int main()
{
  int a;
  int b;
  int c;
  a = 3;
  b = -2;
  c = a + b;
  printinl(c);
  return 0;
}

Output
1



./tests/ test-int3.goo

void testint(int a, int b)
{
  printinl(a + b);
  printinl(a - b);
  printinl(a * b);
  printinl(a / b);
  printinl(a % b);
  printbnl(a == b);
  printbnl(a == a);
  printbnl(a != b);
  printbnl(a != a);
  printbnl(a > b);
  printbnl(a >= b);
  printbnl(a < b);
  printbnl(a <= b);
}

int main()
{
  int c;
  int d;

  c = 42;
  d = 3;

  testint(c, d);

  testint(d, d);

  testint(c, d-1);
  testint(c-1, d-1);

  return 0;
}
```

Output

```
45
39
126
14
0
false
true
true
false
true
true
false
false
6
0
9
1
0
true
true
false
false
false
true
false
true
44
40
84
21
0
false
true
true
false
```

```
true
true
false
false
43
39
82
20
1
false
true
true
false
true
true
false
false
```

./tests/ test-itof.goo

```
int main()
{
    int x;
    float y;

    x = 2;
    y = 3.14;

    printinl(x);
    printfnl(y);

    printinl(~y);
    printfnl(~x);

    y = ~x;
    printinl(x);
    printfnl(y);

    y += ~x;
    printfnl(y);

    return 0;
}
```

Output

```
2
3.1400000000
3
2.0000000000
2
2.0000000000
4.0000000000
```

./tests/ test-local1.goo

```
void foo(bool i)
{
  int i; /* Should hide the formal i */

  i = 42;
  printinl(i + i);
}
```

```
int main()
{
  foo(true);
  return 0;
}

Output
84



./tests/ test-local2.goo

int foo(int a, bool b)
{
  int c;
  bool d;

  c = a;

  return c + 10;
}

int main() {
 printinl(foo(37, false));
 return 0;
}

Output
47



./tests/ test-local3.goo

void foo(bool i, int f, float s, str b)
{
  /* Should hide the formals */
  int i;
  float f;
  str s;
  bool b;

  i = 42;
  f = 3.14;
  s = "hello";
  b = false;

  printinl(i + i);
  printfnl(f + f);
  prints(s);
  printbnl(b || b);
}

int main()
{
  foo(true, 10, 1.1e-10, "hi");
  return 0;
}

Output
84
6.2800000000
```

```
hellofalse
```

./tests/ test-negation.goo

```
int main()
{
    int y;
    int z;
    float x;
    float w;

    x=-3.14e-2; y=-2;

    z=-y;
    printinl(y);
    printinl(z);

    w=-x;
    printfnl(x);
    printfnl(w);

    x=3.14e-2; y=2;

    z=-y;
    printinl(y);
    printinl(z);

    w=-x;
    printfnl(x);
    printfnl(w);

    return 0;
}
```

Output

```
-2
2
-0.0314000000
0.0314000000
2
-2
0.0314000000
-0.0314000000
```

./tests/ test-not.goo

```
int main()
{
    bool b;
    bool nb;

    b = true;
    nb = !b;
    printbnl(b);
    printbnl(nb);

    nb = !nb;
    printbnl(nb);

    nb = !true;
    printbnl(nb);
    b = false;
    nb = !b;
    printbnl(b);
    printbnl(nb);

    nb = !nb;
    printbnl(nb);

    nb = !false;
    printbnl(nb);

    return 0;
}
```

```
Output
true
false
true
false
false
true
false
true
```

```
./tests/ test-ops1.goo

int main()
{
  printinl(1 + 2);
  printinl(1 - 2);
  printinl(1 * 2);
  printinl(100 / 2);
  printinl(99);
  printbnl(1 == 2);
  printbnl(1 == 1);
  printinl(99);
  printbnl(1 != 2);
  printbnl(1 != 1);
  printinl(99);
  printbnl(1 < 2);
  printbnl(2 < 1);
  printinl(99);
  printbnl(1 <= 2);
  printbnl(1 <= 1);
  printbnl(2 <= 1);
  printinl(99);
  printbnl(1 > 2);
  printbnl(2 > 1);
  printinl(99);
  printbnl(1 >= 2);
  printbnl(1 >= 1);
  printbnl(2 >= 1);
  return 0;
}
```

```
Output
3
-1
2
50
99
false
```

```
true
99
true
false
99
true
false
99
true
true
false
99
false
true
99
false
true
true
```

./tests/ test-ops2.goo

```
int main()
{
  printbnl(true);
  printbnl(false);
  printbnl(true && true);
  printbnl(true && false);
  printbnl(false && true);
  printbnl(false && false);
  printbnl(true || true);
  printbnl(true || false);
  printbnl(false || true);
  printbnl(false || false);
  printbnl(!false);
  printbnl(!true);
  printinl(-10);
  printinl(--42);

  printfnl(1.0 + 2.0);
  printfnl(1.0 - 2.0);
  printfnl(1.0 * 2.0);
  printfnl(1.0 / 2.0);
  printfnl(1.0 % 2.0);
  printfnl(2.0 ^ 2.0);
  printfnl(-1.0 + 2.0);
  printfnl(-1.0 - 2.0);
  printfnl(-1.0 * 2.0);
  printfnl(-1.0 / 2.0);
  printfnl(-1.0 % 2.0);
  printfnl(-2.0 ^ 2.0);
  printfnl(-1.0 + -2.0);
  printfnl(-1.0 - -2.0);
  printfnl(-1.0 * -2.0);
  printfnl(-1.0 / -2.0);
  printfnl(-1.0 % -2.0);
  printfnl(-2.0 ^ -2.0);

  printinl(~1.0 + ~2.0);
  printinl(~1.0 - ~2.0);
  printinl(~1.0 * ~2.0);
  printinl(~1.0 / ~2.0);
  printinl(~1.0 % ~2.0);
  printinl(~(2.0 ^ 2.0));
  printinl(~-1.0 + ~2.0);
  printinl(~-1.0 - ~2.0);
  printinl(~-1.0 * ~2.0);
  printinl(~-1.0 / ~2.0);
  printinl(~-1.0 % ~2.0);
  printinl(~(-2.0 ^ 2.0));
  printinl(~-1.0 + -~2.0);
  printinl(~-1.0 - -~2.0);
  printinl(~-1.0 * -~2.0);
  printinl(~-1.0 / -~2.0);
  printinl(~-1.0 % -~2.0);
  printinl(~(-2.0 ^ -2.0));

  printinl(1 + 2);
  printinl(1 - 2);
  printinl(1 * 2);
  printinl(1 / 2);
  printinl(1 % 2);
  printinl(~(~2 ^ ~2));
  printinl(-1 + 2);
  printinl(-1 - 2);
  printinl(-1 * 2);
  printinl(-1 / 2);
  printinl(-1 % 2);
  printinl(~(-~2 ^ ~2));
  printinl(-1 + -2);
  printinl(-1 - -2);
  printinl(-1 * -2);
  printinl(-1 / -2);
  printinl(-1 % -2);
  printinl(~(~-2 ^ ~-2));

  printfnl(~1 + ~2);
  printfnl(~1 - ~2);
  printfnl(~1 * ~2);
  printfnl(~1 / ~2);
  printfnl(~1 % ~2);
  printfnl(~2 ^ ~2);
  printfnl(~-1 + ~2);
  printfnl(~-1 - ~2);
  printfnl(~-1 * ~2);
  printfnl(~-1 / ~2);
  printfnl(~-1 % ~2);
  printfnl(~-2 ^ ~2);
  printfnl(~-1 + -~2);
  printfnl(~-1 - -~2);
  printfnl(~-1 * -~2);
  printfnl(~-1 / -~2);
  printfnl(~-1 % -~2);
  printfnl(~-2 ^ -~2);

  printsnl("hello" _ ' "world"' _ '!');
  printsnl("world" ? 5);
  printsnl("hello" _ ' "world"' ? 5);
  printsnl(("hello" _ " world") ? 0);
  printsnl(("hello" _ " world") ? 4);
  printsnl((("hello" _ " world") ? 4) ? 0);
  printsnl((("hello" _ " world") ? 4) ? 0 _ "world");
}
```

```
Output
true
false
```

```
true
false
false
false
true
true
true
false
true
false
-10
41
3.0000000000
-1.0000000000
2.0000000000
0.5000000000
1.0000000000
4.0000000000
1.0000000000
-3.0000000000
-2.0000000000
-0.5000000000
-1.0000000000
4.0000000000
-3.0000000000
1.0000000000
2.0000000000
0.5000000000
-1.0000000000
0.2500000000
3
-1
2
0
1
4
1
-3
-2
0
-1
4
-3
1
2
0
-1
0
3
-1
2
0
1
4
1
-3
-2
0
-1
4
-3
1
2
0
```

```
-1
0
3.0000000000
-1.0000000000
2.0000000000
0.5000000000
1.0000000000
4.0000000000
1.0000000000
-3.0000000000
-2.0000000000
-0.5000000000
-1.0000000000
4.0000000000
-3.0000000000
1.0000000000
2.0000000000
0.5000000000
-1.0000000000
0.2500000000
hello "world"!

hellol
h
o
o
oworld
```

./tests/ test-pow.goo

```
/*
 * Test for linking external C functions to LLVM-generated code
 *
 * ^ uses the math.h pow function
 * The C compiler generates stdlib.o
 * The LLVM compiler, llc, translates the .ll to an assembly .s file
 * The C compiler assembles the .s file and links the .o file with -lm to generate an executable
 */

int main()
{
  printfnl(~2 ^ -~3);
  printfnl(-~2 ^ -~3);
  printfnl(~-2 ^ ~3);
  printfnl(2.0 ^ 3.);
  printfnl(4.0 ^ 0.5);
  printfnl(4.0 ^ 0.3);
  return 0;
}
```

Output
```
0.1250000000
-0.1250000000
-8.0000000000
8.0000000000
2.0000000000
1.5157165665
```

./tests/ test-printb1.goo

```
int main()
```

```
{
    printbasi("a" == "a");
    printbasi("a" == "b");

    printbasi("a" != "a");
    printbasi("a" != "b");

    printbasi("b" < "c");
    printbasi("b" < "b");
    printbasi("b" < "a");

    printbasi("b" <= "c");
    printbasi("b" <= "b");
    printbasi("b" <= "a");

    printbasi("b" > "a");
    printbasi("b" > "b");
    printbasi("b" > "c");

    printbasi("b" >= "a");
    printbasi("b" >= "b");
    printbasi("b" >= "c");

  return 0;
}
```

```
Output
1
0
0
1
1
0
0
1
1
0
1
0
0
1
1
0
```

./tests/ test-printb.goo

```
int main()
{
   bool b;
   b = false;

   printb(b);
   printbnl(b);

   b = true;
   printb(b);
   printbnl(b);

   return 0;
}
```

```
Output
falsefalse
```

```
truetrue
```

./tests/ test-printbig.goo

```
/*
 * Test for linking external C functions to LLVM-generated code
 *
 * printbig is defined as an external function, much like printf
 * The C compiler generates printbig.o
 * The LLVM compiler, llc, translates the .ll to an assembly .s file
 * The C compiler assembles the .s file and links the .o file to generate
 * an executable
 */

int main()
{
  printbig(72); /* H */
  printbig(69); /* E */
  printbig(76); /* L */
  printbig(76); /* L */
  printbig(79); /* O */
  printbig(32); /*   */
  printbig(87); /* W */
  printbig(79); /* O */
  printbig(82); /* R */
  printbig(76); /* L */
  printbig(68); /* D */
  return 0;
}
```

Output

```
  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
        XX
        XX
        XX
  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX


  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX    XX    XX
  XX    XX    XX
  XX    XX    XX
  XX          XX


  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX
  XX
  XX
  XX


  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX
  XX
  XX
  XX
```

```
    XXXXXXXXXX
  XXXXXXXXXXXXXX
  XX          XX
  XX          XX
  XX          XX
  XXXXXXXXXXXXXX
    XXXXXXXXXX









      XXXXXXXXXX
  XXXXXXXXXXXXXX
    XXXXXX
      XXXXXX
    XXXXXX
  XXXXXXXXXXXXXX
      XXXXXXXXXX

    XXXXXXXXXX
  XXXXXXXXXXXXXX
  XX          XX
  XX          XX
  XX          XX
  XXXXXXXXXXXXXX
    XXXXXXXXXX

  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
      XX      XX
    XXXX      XX
  XXXXXXXX    XX
  XXXX  XXXXXXXX
  XX    XXXXXX


  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX
  XX
  XX
  XX

  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX          XX
  XX          XX
  XXXX      XXXX
    XXXXXXXXXX
      XXXXXX
```

```
./tests/ test-print.goo

int main()
{  int i;
   float f;
```

```
str s;
bool b;

i = 42;
f = 3.14;
s = "hello";
b = true;

printi(42);
printinl(42);
printf(3.14);
printfnl(3.14);
prints("hi");
printsnl("hi");
printbasi(true);
printb(true);
printbnl(true);

printi(42 + 1);
printinl(42 + 2);
printf(3.14 +1.0);
printfnl(3.14 * ~3);
printfnl(3.14 + 2.0);
prints("hi" _ " world");
printsnl("hi" _ " world");

printbasi(true && true);
printb(true && false);
printbnl(true || false);
printbnl(42 == 1);
printbnl(42 == 42);
printbnl(3.14 != 1.0);
printbnl(3.14 != 3.14);
printbnl(true == true);
printbnl(true != false);

printb(42 == 1);
printb(42 == 42);
printb(3.14 != 1.0);
printb(3.14 != 3.14);
printb(true == true);
printb(true != false);
printb(42 == 1);
printb(42 == 42);
printb(3.14 != 1.0);
printb(3.14 != 3.14);
printb(true == true);
printb(true != false);
printbasi(42 == 1);
printbasi(42 == 42);
printbasi(3.14 != 1.0);
printbasi(3.14 != 3.14);
printbasi(true == true);
printbasi(true == !true);
printbasi(true != false);
printi(i);
printf(f);
prints(s);
printb(b);
printbasi(b);
printinl(i);
printfnl(f);
printsnl(s);
printbnl(b);
```

```
   printinl(i + i);
   printfnl(f - f);
   printsnl(s _ s);
   printbnl(b && b);
   printbasi(b || b);

   printbasi("hi" == " world");
   printbasi("hi" != " world");
   printsnl("" _ true);
   printsnl("" _ 3.14);
   printsnl("" _ 42);

   printc(`a`);
   printcnl(`b`);

   return 0;
}
Output
4242
3.143.1400000000
hihi
1
truetrue
4344
4.149.4200000000
5.1400000000
hi worldhi world
1
falsetrue
false
true
true
false
true
true
falsetruetruefalsetruetruefalsetruetruefalsetruetrue0
1
1
0
1
0
1
423.14hellotrue1
42
3.1400000000
hello
true
84
0.0000000000
hellohello
true
1
0
1
true
3.14
42
ab



./tests/ test-stdlib.goo

int main()
```

```
{
  printfnl(sin(1.0));
  printfnl(cos(1.0));
  printfnl(tan(1.0));
  printfnl(asin(1.0));
  printfnl(acos(1.0));
  printfnl(atan(1.0));
  printfnl(sinh(1.0));
  printfnl(cosh(1.0));
  printfnl(tanh(1.0));
  printfnl(exp(1.0));
  printfnl(log(1.0));
  printfnl(log10(1.0));
  printfnl(sqrt(1.0));
  printfnl(sqrt(0.0));
  printfnl(sqrt(~2));
  printfnl(sqrt(-~2));
  printfnl(floor(1.5));
  printfnl(ceil(1.5));
  printfnl(abs(-1.5));
  printfnl(fmod(1.0,2.0));
  printfnl(fmod(2.0,2.0));

  srand(8);
  printinl(rand());
  printinl(strlen("hello" _ "world" _ '!'));

  printinl(strspn("hello", "hell"));
  printinl(strcspn("hello", "whel"));
  printinl(strcmp("hello", "world"));
  printinl(strncmp("hello", "hexx", 2));
  printsnl(strstr("hello", "l"));


  printsnl(strchr("hello", "l"));
  printsnl(strrchr("hello", "hello"));
  printsnl(strpbrk("hello", "hell"));

  printsnl(strerror(1));
  printsnl(strerror(2));
  printsnl(strerror(3));


  /*
  printsnl(strcpy("hello", "world"));
  printsnl(strncpy("hello", "world", 3));
  printsnl(strcat("hello", "world"));
  printsnl(strncat("hello", "world", 3));

  printsnl(strtok("hello", "el"));
  */



printbnl(isalnum(`a`) > 0);
printbnl(isalnum(`!`) > 0);
printbnl(isalpha(`a`) > 0);
printbnl(isalpha(`#`) > 0);
printbnl(iscntrl(`a`) > 0);
printbnl(isdigit(`0`) > 0);
printbnl(isdigit(`x`) > 0);
printbnl(isgraph(`a`) > 0);
printbnl(isgraph(`2`) > 0);
printbnl(isgraph(` `) > 0);
printbnl(islower(`a`) > 0);
printbnl(islower(`A`) > 0);
printbnl(isprint(` `) > 0);
printbnl(ispunct(`&`) > 0);
printbnl(ispunct(`a`) > 0);
printbnl(isspace(` `) > 0);
printbnl(isspace(`_`) > 0);
printbnl(isupper(`B`) > 0);
printbnl(isupper(`a`) > 0);
printbnl(isxdigit(`a`) > 0);

  return 0;
}
```

```
Output
0.8414709848
0.5403023059
1.5574077247
1.5707963268
0.0000000000
0.7853981634
1.1752011936
1.5430806348
0.7615941560
2.7182818285
0.0000000000
0.0000000000
1.0000000000
0.0000000000
1.4142135624
-nan
1.0000000000
2.0000000000
1.5000000000
1.0000000000
0.0000000000
757547896
11
4
0
-15
-12
llo
(null)
(null)
(null)
Operation not permitted
No such file or directory
No such process
true
false
true
false
false
true
false
true
true
false
true
false
true
true
false
```

```
true
false
true
false
true
```

./tests/ test-str1.goo

```
int main()
{
  str a;
  a = "hello
  world
";
  prints(a);
  a = 'hello world!';
  printsnl(a);
  return 0;
}
```

Output

```
hello
  world
hello world!
```

./tests/ test-str3.goo

```
void testsrt(str s, int i)
{
  printsnl(s _ s);
  printsnl(s ? i);
}

int main()
{
  str s1;
  int i1;

  s1 = "hello";
  i1 = 1;

  testsrt(s1, i1);

  return 0;
}
```

Output

```
hellohello
e
```

./tests/ test-str4.goo

```
int main()
{
    printbasi("a" == "a");
    printbasi("a" == "b");

    printbasi("a" != "a");
    printbasi("a" != "b");

    printbasi("b" < "c");
    printbasi("b" < "b");
    printbasi("b" < "a");

    printbasi("b" <= "c");
    printbasi("b" <= "b");
    printbasi("b" <= "a");

    printbasi("b" > "a");
    printbasi("b" > "b");
    printbasi("b" > "c");

    printbasi("b" >= "a");
    printbasi("b" >= "b");
    printbasi("b" >= "c");

  return 0;
}
```

```
Output
1
0
0
1
1
0
0
1
1
0
1
0
0
1
1
0
```

./tests/ test-str5.goo

```
int main()
{
  str s;
  bool b;

  s = "hello";
  b = true;
  s = s _ b;
  printsnl(s);

  b = false;
  s _= b;
  printsnl(s);
}
```

```
Output
hellotrue
hellotruefalse
```

./tests/ test-subassign.goo

```
int main()
{
    int y;
    int z;
    float x;
    float w;

    y=2; z=3;
    printinl(y);
    printinl(z);
    y-=5;
    printinl(y);
    y-=z;
    printinl(y);

    x=3.14e-5;
    w=5.1e5;
    printfnl(x);
    printfnl(w);
    x-=1.1;
    printfnl(x);
    x-=w;
    printfnl(x);
    x-=-10e10;
    printfnl(x);
    return 0;
}
```

```
Output
2
3
-3
-6
0.0000314000
510000.0000000000
-1.0999686000
-510001.0999686000
99999489998.9000244141
```

./tests/ test-var1.goo

```
int main()
{
  int a;
  a = 42;
  printinl(a);
  return 0;
}
```

```
Output
42
```

./tests/ test-var2.goo

```
int a;

void foo(int c)
{
  a = c + 42;
}
```

```
int main()
{
  foo(73);
  printinl(a);
  return 0;
}

Output
115
```

./tests/ test-while1.goo

```
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    printinl(i);
    i = i - 1;
  }
  printinl(42);
  return 0;
}

Output
5
4
3
2
1
42
```

./tests/ test-while2.goo

```
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

int main()
{
  printinl(foo(7));
  return 0;
}

Output
14
```

./tests/ test-while3.goo

```
int main()
```

```
{
  bool i;
  i = true;
  while (i) {
    printbnl(i);
    i = false;
  }
  printinl(42);
  return 0;
}

Output
true
42
```

./tests/ test-while4.goo

```
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    printinl(i);
    i -= 1;
  }
  printinl(42);
  return 0;
}

Output
5
4
3
2
1
42
```

9. References, Resources and Bibliography

Stephen A. Edwards
The MicroC Compiler
Columbia University, Fall 2018
http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/microc.pdf
http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/index.html

OCaml source and test cases for the MicroC language, which generates LLVM IR:
http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/microc.tar.gz

Brian W. Kernighan and Dennis M. Ritchie
The C Programming Language, Second Edition
Prentice Hall, Inc., 1988
http://s3-us-west-2.amazonaws.com/belllabs-microsite-dritchie/cbook/index.html

Dennis M. Ritchie
C Reference Manual
Bell Telephone Laboratories, May 1975
https://www.bell-labs.com/usr/dmr/www/

OCaml
https://caml.inria.fr, https://ocaml.org/

Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy and Jérôme Vouillon
The OCaml system release 4.07 Documentation and user's manual
https://caml.inria.fr/pub/docs/manual-ocaml/
July 13, 2018

Emmanuel Chailloux, Pascal Manoury and Bruno Pagano
Developing Applications with Objective Caml
http://caml.inria.fr/pub/docs/oreilly-book/
O'Reilly France

Yaron Minsky, Anil Madhavapeddy & Jason Hickey
Real World OCaml
O'Reilly Media, 2013
https://v1.realworldocaml.org/

OCamlLex tutorial
https://courses.softlab.ntua.gr/compilers/2015a/ocamllex-tutorial.pdf

OCamlYacc tutorial
https://courses.softlab.ntua.gr/compilers/2015a/ocamlyacc-tutorial.pdf

LLVM
OCaml bindings documentation, https://llvm.moe/ocaml/Llvm.html
https://llvm.org/docs/tutorial/OCamlLangImpl7.html

Clang
https://clang.llvm.org/get_started.html

Alfred V. Aho, Monica Lam, Ravi Sethi, and Jeffrey D. Ullman
Compilers: Principles, Techniques, and Tools
Addison-Wesley, 2006. Second Edition

For all kinds of programming questions,
https://stackoverflow.com

For all kinds of questions,
https://en.wikipedia.org