ALBS

Programming Languages and Translators
Columbia University - Spring 2016

Suhani Singhal [ss4925]
Brennan Wallace [bgw2119]

Table of Contents

I. Introduction 4

II. Language Tutorial 5

- Running an ALBS Program
- Writing an ALBS Program

III. Language Manual 6

- Introduction
- Lexical Conventions
  - Tokens
  - Comments
  - Identifiers
  - Keywords
  - Constants
    - Integer literal
    - Float literal
    - Character literal
    - Boolean literal
  - String literal
- Expressions
  - Infix Expressions
  - Operators
    - Unary Operators
    - Logical Binary Operators
    - Comparison Operators
    - Other Operators
- Data Types
  - Overview
  - Integer
  - Character
  - Float
  - Array
  - Boolean
  - Void
  - Structs
  - Function
- Scope
- Control Flow

Program Entry and Exit
Expression statements and blocks
If-Else
Else-If
Loops - While and For
Punctuation
Semicolon
Colon
Curly Braces
White Space
Parentheses
Square Brackets
Standard Library
I/O
Array Operations
References
IV. Project Plan 24
A. Process
1. Planning
2. Specification
3. Development
4. Testing
B. ALBS Programming Style Guide
1. Basic White Space Rules
2. Indentation
3. Brackets
4. Comments
5. Other rules
C. Project Timeline
D. Team Roles
E. Development Environment
F. Project Log
V. Architectural Design 42
A. Diagram
B. Interfaces Between Components
C. Implementation Credits for Components
VI. Test Plan 43
Example Program 1
B. Example Program 2

C. Test Suite
1. Error Tests
2. Array Tests
3. Char Tests
4. Struct Tests
5. Control Flow Tests
6. Other Tests
B. Rationale Behind Test Suites
C. Test Suite Automation
D. Implementation Credits for Testing
VII. Lessons Learned 65
A. Lessons Team Members Learned
Suhani
Brennan
B. Advice For Future Teams
Suhani
Brennan
VIII. Appendix 66

I. Introduction

ALBS is a procedural general purpose programming language that is compiled to LLVM. Compilation to such a low-level allows tighter control of how a given is executed allowing programmers to have better control over the efficiency of their creation.

ALBS has a broad range of data types: booleans, integers, floats, and characters. This range of operations allows a wide variety of computational tasks to be easily represented. It also has advanced structures in the form of arrays and user defined structs, so users can manipulate large and interesting collections of data.

ALBS is statically typed preventing possible runtime errors and making programs more straightforward to understand. Furthermore it uses a static scope making functions' data dependencies explicit and increasing their reusability.

II. Language Tutorial

Running an ALBS Program

Running an albs program is very simple and consists of 3 steps after make.

One: Converting the .sb file to llvm
Generalized Command: ./albs.native <[sb file name]> [.ll file]
Example: ./albs.native <test-if1.sb> test-if1.ll

Two: Use clang to create the executable
Generalized Command: clang [.ll file] -o [executable name]
Example: clang test-if1.ll -o program

Three: Run the executable
Generalized Command: ./[executable name]
Example: ./program

Writing an ALBS Program

The syntax is similar to C. A sample .sb file looks like this:

```
(* this is a comment *)
{ :int } main = { (* mandatory main function *)
      (* these are variable declarations *)
      int i;
      struct Location l;
      bln b;
      chr[] ch;
      ch = new int[10]; (* this is an array initialization *)
      l.longitude =  30.75;
      print(“Hello World”);
}
struct Location[ (* sample struct *)
      flt longitude;
      flt latitude;
]
```

III. Language Manual

1. Introduction

This manual describes the ALBS language - a procedural/imperative language with a C-like syntax, that is compiled down to LLVM.

Throughout this manual, sample code is written in monospace font, as this is.

Throughout the LRM, placeholders are represented as **sample_placeholder**

2. Lexical Conventions

2.1. Tokens

There are four types of tokens: keywords, identifiers, constants, operators, and punctuation. Blanks, horizontal and vertical tabs, newlines, form feeds and comments as described below (collectively know as whitespace) are ignored except as they separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.

2.2. Comments

Comments are multiple line blocks and the block must start with (* characters with no space between them though any amount of space can be before them as well as code that is evaluated normally. The comment is terminated by *) without any spaces between them. Anything inside a comment is ignored for the purposing of compilation and creating the abstract syntax tree.

To Illustrate:

```
rtn 0; (*this is a comment *)
this is not a comment
(* neither is this
```

2.3. Identifiers

Identifiers are sequences of alphanumeric characters and underscores that are largely used as variable names. A regular expression that can be used to define them is:
['a'-'z' 'A'-'Z'] (['a'-'z' 'A'-'Z'] | ['0'-'9'] | '_')* .

2.4. Keywords

ALBS has a set of reserved keywords and function names that cannot be used as identifiers for variables. This is a list of all reserved keywords:

Data Types	Control Flow	I/O functions	Other
bln	rtn	print	true
flt	if	getchar	false
int	else		new
chr	main		
void	for		
struct	while		

2.5. Constants

2.5.1. Integer literal

Integer literals are a sequence of numerals without decimal point. Not having a decimal point is crucial as this distinguishes them from doubles. A hyphen may precede the numerals to indicate that the value is negative. Note: -0 is will be converted to 0 during parsing. A regular expression for ints can be defined as follows:

-?[‘0’-’9’]+

To illustrate:

```
int zero = 0; (* zero is 0 *)
int positive_one = 1; (* positive_one is 1 *)
int negative_one = -1;(* negative_one is -1 *)
```

2.5.2. Float literal

Unlike integers, floats are required to have a single decimal point somewhere in the sequence of digits but are otherwise similar. A regular expression for floats can be defined as follows:
-?([‘0’-’9’]+ ’.’ [‘0’-’9’]* | [‘0’-’9’]* ’.’ [‘0’-’9’]+)

If one side of the decimal has no characters it is equivalent to a single 0 being there. A float literal with a zero before (to the left) of the first non-zero is equivalent to a float literal of the same sequence without the preceding 0.

To illustrate:

```
flt zero = 0.0; (* zero is 0 *)
flt positive_one_point_one = 1.100; (* equivalent to 1.1 *)
flt negative_one_point_one = -1.10; (* equivalent to -1.1 *)
```

2.5.3. Character literal

Character literals use a single quote before and after the character. Characters generally represent themselves (‘a’ is the literal for the character representing the letter a). However, the following table shows how some characters are defined using a backslash and another character. This table largely pulled from The C Programming Language 2nd Edition as we feel it is important to support most of the same “special” characters in the same way.

Character	Literal w/Quotes
newline	‘\n’
horizontal tab	‘\t’
vertical tab	‘\v’
backspace	‘\b’
carriage return	‘\r’
backslash	‘\\’
single quote	‘\’’
double quote	‘\”

To Illustrate:

```
chr a = ‘a’;
chr newline = ‘\n’;
(*both of these set the variables to their namesake values*)
rtn ‘c’; (*shows a non assignment usage*)
```

2.5.4. Boolean literal

Boolean literals come in the form of two reserved keywords true and false. To Illustrate:

```
rtn true == true;
rtn true != false;
rtn false == false;
rtn true;
(* all of the above return true*)
```

2.6. String literal

String literals are declared using a sequence of characters (without single quotes) that start and end with a double quote. Such literals can only be declared within a print statement, that can be used to output messages to the stdout.
To Illustrate:
print(“Hello World!”);

3. Expressions

3.1. Infix Expressions

Operators in expressions are evaluated in according to the follow precedence table where a lower number is evaluated before a higher number and on a tie are evaluated as according to their associativity. Operators higher on the table have a higher precedence.

Operator (Symbols)	**Operator (Names)**	**Associativity**
(**expression**)	Parentheses	Left to Right
(**type**) - !	Cast Negation Not	Left to Right Right to Left Right to Left
* / %	Multiplication Modulo Remainder	Left to Right Left to Right Left to Right
+ -	Addition Multiplication	Left to Right Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to Right Left to Right Left to Right Left to Right
== !=	Equal Not Equal	Left to Right Left to Right
&	And	Left to right
|	Or	Left to right
=	Assign	Right to Left

3.2. Operators

Operator tokens indicate operations perform an action that takes the value of one or two operands (a literal or variable) and returns a new value. There are two category types number of operators (unary or binary) and output types.

The following tables describe the operators. Capital letters represent appropriate variables or literals. Descriptions in curly brackets should not be taken literally.

3.2.1. Unary Operators

Name	Notation	Description
Negation	-**variable**	Can be applied to a float or an integer and returns a new float or integer that as a value equal to the original multiplied by -1.
Not	!**variable**	Can be applied to a boolean and returns the opposite value of that boolean.

3.2.2. Arithmetic Binary Operator

Name	Notation	Description
Addition	**var** + **var**	Can be applied to two floats or two integers and returns a new float or integer (output type is the same as the input types) that has a value approximately equal to mathematical sum of the two inputs. Output if overflow occurs is not guaranteed.
Subtraction	**var** - **var**	Can be applied to two floats or two integers and returns a new float or integer (output type is the same as the input types) that has a value approximately equal to mathematical difference of the two inputs. Output if overflow occurs is not guaranteed.
Multiplication	**var** * **var**	Can be applied to two floats or two integers and returns a new float or integer (output type is the same as the input types) that has a value approximately equal to mathematical product of the two inputs. Output if overflow occurs is not guaranteed.
Division	**var** / **var**	Can be applied to two floats or two integers and returns a new float or integer (output type is the same as the

		input types) that has a value approximately equal to mathematical quotient of dividing the two inputs (A divided by B). The value is truncated. Output if overflow occurs is not guaranteed.
Modulo	**var** % **var**	Can be applied to two integers and returns a new integer (output type is the same as the input types) that has a value approximately equal to mathematical remainder of dividing the two inputs (A divided by B). Output if overflow occurs is not guaranteed.

We support a special inline casting for Add, Sub, Multiply and Divide binops that takes in an integer identifier and a character literal, casts the character to its integer value, ands it to the second integer operand, and returns an integer.

For example:

```
int i;
int j;
i = 2;
j = i + ‘a’; (* this stores 97+2 i.e. 98 in j*)
```

3.2.3. Logical Binary Operators

Name	Notation	Description
And	**var** & **var**	Takes two booleans and returns true if both operands are true.
Or	**var** | **var**	Takes two booleans and returns true if at least one operand is true.

3.2.4. Comparison Operators

Name	Notation	Description
Equals	**var** == **var**	Takes two operands of the same type and returns true if they are identical in value and false otherwise.
Not Equals	**var** != **var**	Takes two operands of the same type and returns false if they are identical in value and true otherwise.

Less than	**var** < **var**	Takes two operands of the same type. Returns true if they are floats and integers and A is mathematically less than B. Returns true if they are characters and A precedes B in the ASCII character table. Otherwise false is returned.
.Greater than	**var** > **var**	Takes two operands of the same type. Returns true if they are floats and integers and A is mathematically greater than B. Returns true if they are characters and A follows B in the ASCII character table. Otherwise returns false.
Less than or equal to	**var** <= **var**	Takes two operands of the same type. Returns false if they are floats and integers and A is mathematically greater than B. Returns false if they are characters and A follows B in the ASCII character table. Otherwise returns true.
.Greater than or equal to	**var** >= **var**	Takes two operands of the same type. Returns false if they are floats and integers and A is mathematically less than B. Returns false if they are characters and A precedes B in the ASCII character table. Otherwise returns true.

3.2.5. Other Operators

Name	Notation	Description
Parentheses	(**Some Expression**)	Used for precedence control as what inside parentheses will be evaluated first.
Assign	A = **literal, variable, or an expression of the type of A**;	Assigns the value of the right side of the = sign to the variable on the left side.

4. Data Types

Type	Syntax	Details
integer	int	32-bit, 2's complement
character	chr	8-bit, Ascii
float	flt	32-bit, IEEE 754
array	type[] name	Single Type, fixed size
boolean	bln	True or False only
void	void	non-existent value
function	{type type... type : return_type} funcName = ID ID … ID [**body**]	-

4.1. Overview

There are four fairly standard data types that are similar to their equivalents in C. These types are integer, character, float and boolean.

4.2. Integer

Integers are represented with 32-bit memory chunks in 2's complement. This gives an maximum and minimum of 2,147,483,647 and -2,147,483,648 respectively. The standard arithmetic operators (+,-,/,*) can be applied to integers. Truncation is used to calculate the result of division if a non mathematical integer would be the mathematical result. For example 7/2 == 3 would return true.

```
int x;
int y;
x = 2;
y = 7/2; (* value of y is 3 *)
```

4.3. Character

We support the ASCII character set in using a byte of memory for each character. We use the system as ASCII where ASCII characters have a corresponding number and it is this number the the byte is set to (unsigned).

```
chr x;
chr y;
int z;
x = ‘a’;
y = ‘7’;
print x;(*prints a*)
print y;(*prints 7*)
z = x + y; (*adds 97 and 55. prints 152*)
```

4.4. Float

They are represented using 32 bits as according to the IEEE 754 standard. Each float must include a decimal point and have digits before, after, or both. In terms of regular expressions this can be represented as:
[‘0’-’9’]+ ‘.’ [‘0’-’9’]* | [‘0’-’9’]* ‘.’ [‘0’-’9’]+

```
flt x;
x = 0.1;
```

4.5. Array

A array can be of type int, chr, bln or flt. Every element in the array must be of the same type.
They are represented as:

```
type[] name;
```

Arrays can be declared as:

```
int[] array;
array = new int[2];
```

where the size of array is 2, and it can contain two elements of type int.

Arrays can then be initialized. For instance,

```
array[0] = 10; (*sets element at index 0 to 10*)
array[1] = 20; (*sets element at index 1 to 20*)
```

Elements can be accessed by using their index location.
For example,

```
print(array[0]);(*prints 10 *)
print(array[1]);(*prints 20 *)
```

A special type of character array can be declared inline when being passed to the print statement. The syntax is as follows:

```
print(“hello world”); //prints hello world
```

4.6. Boolean

A boolean can be true or false.

```
bln x;
bln y;
x = true;
y = false;
print(x&y); (*prints false*)
```

4.7. Void

This represents a non-existent value. If a function returns nothing, then it returns void by default.

4.8. Structs

Structs combine multiple other data types into an aggregate with named fields. These named fields can then be easily accessed using a simple notation.

Declaration:
struct **struct_name**[
      type **field_name**;
      type **field_name**;
      ...
]
Usage:
struct **struct_name** **variable_name**;
variable_name.**field_name**;

For example,

```
struct Person[
      chr gender;
      int age;
      flt weight;
]
{:int} main = {
      struct Person p;
      p.gender = ‘f’;
      p.age = 20;
      p.weight = 120.5;
}
```

4.9. Function

Functions take in zero or more parameters and return a value of a predetermined data-type. There is not a set keyword unlike the previous data types but instead the following convention is used:
{ paramType1 paramType2 . . . paramTypeK **:** returnType}
      functionName **=** paramName1 paramName2 . . . paramNameK [
      (*code*)
]

For example,

```
{int int : int} sampleFunction = x y [
      (*code*)
]
```

5. Scope

Scope is controlled by curly brackets and square brackets from control structures and function definitions respectively.

On a function level scope is managed in the following way. Upon entering the function the parameters are at the outermost level of scope. Anything declared outside any control statements or inner function bodies are also at this level.

Each interior control structure creates a new scope one level lower than the level the interior function of control structure resides in and for each of these new levels of scope upon entering the function the parameters are at the outermost level of scope and, again, anything declared outside any control statements are also at this level. This recursive nature can continue ad infinitum.

Variables residing in a scope level are not in any scope levels above the level the variable was declared in and are in every scope that is based of the current scope. These results in variables coming into into scope when they are declared and going out of scope when the matching closing parenthesis is found for opening parentheses most immediately preceding the variables declaration. When a function calls a function the scope of the caller is not inherited by the callee. All function not in another function are in all scopes.

Sample Code:

```
{int int: int} bar = a b [
      int z;
      z = a % b;
      (*ok because the only variables in this scope are a and b*)
      rtn z;
]

{int int: int} foo = x y[(*x and y are immediately in the scope*)

      int z = 0; (*z at the same level of scope as x and y*)
      int w = bar x y; (*w also at this level*)

      if (w == 0) [
            int v = 200 + z;
            (*legal because z is inherited from the outer scope.*)
      ]
      int v = w + bar x y;
      (*ok because the v in the if will be removed from scope*)
      rtn v;
]
```

6. Control Flow

6.1. Program Entry and Exit

Entry into the program is accomplished by starting at the main function. This means that the program also starts with any command line arguments entered into the program when the program is executed. The main function is free to call other functions which can also call other functions or themselves. No function can call the main function. Functions can be declared outside the main (these are global functions), inside the main, or inside each other.

Here is a sample code:

```
{ : int} hello = [
      print “hello!”; (*  prints hello! *)
      rtn 0;
]
{ : int} main = args [
      hello();
      rtn 0;
]
```

This would result in hello world being printed.

6.2. Expression statements and blocks

Expression statements are assignments or function calls. Our language supports only expression statements, i.e. those statements that evaluate to a value. Statements are terminated by a semicolon.

Square braces [ ] are used to group statements together into blocks. There is no semicolon after a block ends with a square brace.

Sample code:

```
int x; (*this is an expression*)
x = 10;
print(x); (*this returns void*)

if(x==10) [ (*this is a block*)
      print(“equal”);(*prints a string “equal”*)
]
```

6.3. If-Else

The if-else statement is used to express decisions.

Syntax:

```
if(condition1) [
      statement1;
]
else [
      statement2;
]
```

Here, the else part is optional.

If the condition1 evaluates to true, then the statement1 is executed. If expression is false, and if there is an else part, statement2 is executed instead.

The square brackets are mandatory and help avoid ambiguity.

6.4. Else-If

This is useful for writing a multi-way decision. For example:

```
if(a==b)[
      d=10;
]
else if(b==c)[
      d=20;
]
else[
      d=30;
]
```

The expressions are evaluated in order. If a==b is true, then the statements associated with it are executed, and this terminates the whole chain. Otherwise b==c is checked and if true the statements inside the brackets corresponding to the else if are executed. The last else statement is useful for checking the default case, however it can be omitted.

6.5. Loops - While and For

While loops have the following syntax:

```
while (expression) [
      statement1;
      statement2;
]
```

First, the expression in the round parentheses is evaluated as true (or nonzero) or false.

If true, then the statements within the square brackets are evaluated. Then the expression is tested again, and if true, the statements in the body of the loop are executed again.

If the expression is evaluated at false, the control skip to the end of the loop, marked by the closing square bracket].

If there is only one statement enclosed in a while loop, this statement can be specified without using braces. For instance,

```
while (expression)
      statement;
```

The for loop has the syntax:

```
for (expr1; expr2; expr3)[
      Statement;
]
```

which is equivalent to

```
expr1;
while (expr2) [
      statement;
      expr3;
]
```

There are three expression within a for loop. Usually, expr1 and expr3 are assignments or function calls and expr2 is an expression evaluating to true (nonzero) or false.

These expressions are optional, though the semicolons are mandatory. If expr2 is absent, it is always considered as true. For instance,

```
for (;;) [
]
```

is a loop that executes forever, unless the program is stopped.

7. Punctuation

7.1. Semicolon

A semicolon is used for termination of a line of code. It is also used to separate expressions in a for loop, as shown in the example.

```
int x; (*This is a variable declaration in ALBS. *)
x = 25; (*This is a variable initialization in ALBS. *)

for(x=1; x<4; x++)[
      (* code *)
]
```

7.2. Colon

The colon symbol is used to separate the parameter types and the return type in the function declaration.

```
{int int : int} add = x y [ rtn x + y; ]
(*The colon separates the two int parameters*)
(*from the int return type of the add function.*)
```

7.3. Curly Braces

Curly braces are used for the enclosing of the parameter and return types in a function declaration.

```
{int int : int} add = x y [ rtn x + y; ]
(*The curly braces enclose the int parameters and return type.*)
```

7.4. White Space

White space is defined as spaces, tabs, newlines, and form feeds. It is ignored except to separate tokens.

To Illustrate:

```
if(a == 0) [
      rtn 0;
]
```

Is equivalent to:

```
if(a == 0) [ rtn 0; ]
```

But not to:

```
ifa==0 [
     rtn0;
]
```

which would result in a parsing error.

7.5. Parentheses

Parenthese are used control the order of evaluation.

```
x = 5 / 5 + 1; (*evaluates to 2*)
y = 5/(5 + 1); (*evaluates to 0*)
```

7.6. Square Brackets

The square brackets must be used to enclose the function definition. They are also used to in if/if else/else constructs.

```
{int int: int} multiply = a b [
      int c;
      if (a == 0) [
            rtn 0;
      ](*if definition is enclosed with square brackets.*)
      else [
            c = a - 1;
            rtn b + multiply(c, b);
      ](*else definition is enclosed with square brackets*)
](*multiply definition is enclosed with square brackets.*)
```

8. Standard Library

8.1. I/O

Function name	Sample Code	Description
print	`list[0] = 5.4;` `print(list[0]);`	Prints the argument passed to it.
getchar	`chr c;` `c = getchar();`	Gets a character value from stdin and stores it in c

8.2. Array Operations

Function/Feature name	Example	Description
Get element at index	`value = myArray[1];`	Returns the value at the index of the array myArray as an value of the same type as the array. Indices start from 0.
Set element	`myArray[1] = 10;`	Sets the value at the index of myArray as the value specified.

9. References

“C Operator Precedence." *Cppreference.com*., 10 July 2015. Web.

Kernighan, Brian W., and Dennis M. Ritchie. *The C programming language*. Vol. 2. Englewood Cliffs: prentice-Hall, 1988.

Singh, Uday, et. al. “Superscript Language Reference Manual.” 2015. http://www.cs.columbia.edu/~sedwards/classes/2015/4115-fall/index.html

IV. Project Plan

A. Process

1. Planning

For planning we first figured out on a high level what we wanted our language to do and large-scale characteristics like static scoping and what data-types were important to support. We also made a list of features we would like to support and divided them into features we felt were absolutely necessary and features we would add if we had the time to do so. We then planned out a timeline (as shown in the timeline section) of when we would complete certain milestones.

2. Specification

Creating and maintaining the LRM really helped with keeping track of exactly how our language would operate and what features it would support. Indeed, we changed our language from a functional to a procedural language and the first thing we did was rewrite out LRM to reflect what exactly our language would be like given this shift in programming paradigm.

3. Development

After our group was reduced to two members the remaining members decided to use the pair-programing style of development to implement most of the key features. Essentially we would meet one to three times a week and work on the development/testing item we were tackling at the moment. The greatly simplified the organizational work and had other benefits as well (see Brennan's entry in lessons learned about this).

4. Testing

For testing out idea was to test the successful implementation of each feature by creating one or two simple programs to test the feature using only features we also knew worked correctly. Then we created larger programs (which can be seen in the "Test Plan" section of this document) to test the features working together as part of a far more complicated system.

B. ALBS Programming Style Guide

1. Basic White Space Rules

- Align text to the left.
- Other than indentation never use more than one space when a single space will do.
- Function declarations and parts of control flow should be on their own line.
- No more than 3 empty lines should separate any two lines with text on them.

2. Indentation

- Indent one tab or 4 spaces every time scope increases in depth.
- A line of code should be indented at least one more level than its function declaration line and one more most immediate control flow parent.

3. Brackets

- Opening brackets should be on the same line and one space away from the the code the brackets are for. This does not include brackets enclosing the parameter types of a function.
- Closing brackets should be at the same level of indentation the opening bracket is one with no other text (other than comments) on the line with the closing brackets.
- Using ifs without brackets is fine for ifs with only one line in the body.

4. Comments

- Comment indentation is up to the user.
- Keep the opening and closing on their own lines.

5. Other rules

- No code to the left of a semi-colons.
- Don’t split control flow code, expressions, statements or function headers across more than one line.
- Ascii only.

C. Project Timeline

Planned Date	Actual Date	Item
March 6th	March 4th	LRM
March 25th	March 27th	First Program (Printing an Int)
March 30th	March 27th	Control Flow & Non-Main Functions (Largely from MicroC)
April 4th	April 4th	Hello World! (Strings Completed)
April 10th	April 4th	Get test suite running
April 20th	May 6th	Integers,Floats,Booleans,Strings
April 25th	May 7th	Arrays
Not Planned	May 7th	Structs
May 8th	May 10th	Report and Presentation

You can see we missed a lot of our planned dates but not by much. The reason integers, floats, booleans and strings took so long was we got integers, strings and booleans done but it took a long time for us to set up floats. We were also able to do structs which we did not plan for.

D. Team Roles

Because our team pair programmed we split most of the responsibilities evenly, handling them together. There was a slight divide in Suhani doing a little more of the compiler creation at times and being more of the language guru, while Brennan handled slightly more of the infrastructure, testing and product management. This split was slight though and occurred more towards the end of the semester.

E. Development Environment

The development environment was fairly simple we used sublime text editor to edit files and view results. Sublime also has a nice display to see and open the contents of a specified directory. Towards the end of the semester we switched to Atom which had a slightly nicer UI but was very similar in terms of functionality.

We used a bash script for testing and a the compiler itself was written in ocaml. The bash script’s testing results were displayed via terminal (where the script was called from). This lead us naturally to using a lot of terminal commands and tools in our workflow. cat, vim. nano were all used. We also navigated a lot by terminal (using cd and ls).

F. Project Log

Figure: Github Commit History

The git repository is available at: https://github.com/suhanisinghal/SB

Commit History:

```
ommit f1c552cdd651456831b4a43facbaeb1a5e994ab6
Author: Suhani <suhanisinghal@gmail.com>
Date:   Wed May 11 14:48:27 2016 -0400

    Demo

commit 0e48c63bd588a58386fffb0a9c2a14050f89b5ba
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed May 11 11:59:20 2016 -0400

    histogram cleaned up

commit 72891963ec4f42f3bda799ce7ab2c95d0fb26f85
Merge: d912c59 f3c1e3c
Author: Suhani <suhanisinghal@gmail.com>
Date:   Wed May 11 11:44:07 2016 -0400

    Merge branch 'master' of https://github.com/suhanisinghal/SB

commit d912c5917a1033a5681a68c50578f4c3471aa745
Author: Suhani <suhanisinghal@gmail.com>
```

```
Date:   Wed May 11 11:43:45 2016 -0400

    Added demo for structs

commit f3c1e3c45c01a58abb9a8cba5c0128172296ea95
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed May 11 11:13:04 2016 -0400

    fixed array demo

commit 2b64b82489fcaa01bc762fa0e470ff0523383f15
Author: Suhani <suhanisinghal@gmail.com>
Date:   Wed May 11 11:00:04 2016 -0400

    Demo folder - modified test script

commit 8a8e13db986328d3098437981d08b285ecabb674
Author: Suhani <suhanisinghal@gmail.com>
Date:   Wed May 11 10:42:22 2016 -0400

    fixed some tests

commit df772630d7676a685cd6cfe886b1b4d9a78ec564
Author: Suhani <suhanisinghal@gmail.com>
Date:   Wed May 11 10:35:34 2016 -0400

    demo special chars

commit 0abc78e496d5214653760385c4c8d1c141e61355
Author: Suhani <suhanisinghal@gmail.com>
Date:   Wed May 11 10:18:48 2016 -0400

    implicit casting of chr to int - demo of arrays added

commit 38a084323b545959c9f29f0bc2ab87d3eecdd648
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue May 10 23:54:27 2016 -0400

    added tests

commit d045b385ecc15cc67126835d696bde132ec90cdf
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 23:07:21 2016 -0400
```

```
    Input char in char array

commit 97cd8490b37906f87e80d2939c3a6b080c6e84a3
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 22:11:52 2016 -0400

    still trying

commit c06f629ca3cdaddc294bdeb103c9f4ade1926abc
Merge: 021d286 4021f3c
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 22:08:46 2016 -0400

    Merge branch 'master' of https://github.com/suhanisinghal/SB

commit 021d286f80726c3fa995218147685d3a75859079
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 22:08:23 2016 -0400

    A single character input just works

commit 4021f3cedc48d9fbe25d695c2f253b4941eb6ea8
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue May 10 20:42:51 2016 -0400

    changed comment formating

commit 8fa5ddb3fe887c36738f4b4c013423f5e44239e4
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue May 10 20:16:31 2016 -0400

    negation test fixed

commit e776cb6d44f0df2c57cfda404e324c9571647648
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 20:15:32 2016 -0400

    Struct unops now work

commit c2ba248e1514c8dff74c26355da2df341bce9034
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue May 10 20:15:10 2016 -0400
```

```
    negation tests

commit 11d486129b6a2abd2f0affca1a1b7dfd36373f04
Merge: f6f15c9 9565a8e
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 19:54:27 2016 -0400

    Merge branch 'master' of https://github.com/suhanisinghal/SB

commit f6f15c9985ee74f54efe40dd93bb0492e5455955
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 19:54:02 2016 -0400

    Struct binops now work

commit 9565a8eb80a6a864cdba227fb14d47a0d702046b
Merge: c9e1cf7 bab98e3
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue May 10 19:50:55 2016 -0400

    Merge branch 'master' of https://github.com/suhanisinghal/SB

commit c9e1cf7acdcc44cb020e00f90d7e50e1d8cd0644
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue May 10 19:49:22 2016 -0400

    negation tests

commit bab98e3382c13206a76b7b2665b792b4a21c58c9
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 19:40:10 2016 -0400

    Added tests for binops

commit 4874852fe9474bf19eae642c9f28f58620420cd9
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 19:15:28 2016 -0400

    Refactoring binops

commit ddafe0e5b5f77501ad99a4e85c4915b6e8eef647
Author: Brennan Wallace <brennangoffwallace@gmail.com>
```

```
Date:   Tue May 10 18:57:24 2016 -0400

    for loops work now

commit f768332ad2a00cdbd0e43dee1f732a94f336dc78
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 18:09:13 2016 -0400

    Added test cases for structs

commit 11f433f4be9a25315df4e978bf474553b86ddbe2
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 17:46:11 2016 -0400

    Int, Float and Char can now be struct fields

commit 2b3114971f484f2a7885ddab3b2b35819ad106ce
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 02:24:08 2016 -0400

     Structs with int fields can now be printed - need to print other type of
fields too

commit de477bf67d7059d878d3116e9f4e6ec97589f412
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 01:07:42 2016 -0400

    bool arrays now work

commit 2e5c377d29acb7b7d3b5491ae57ab1a22863a0f2
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 00:09:55 2016 -0400

    Added test case for assigning array val on RHS

commit 86df82d81e39c8c3e03364591f7b3bcb9c74a3db
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue May 10 00:02:55 2016 -0400

    Fixed char arrays

commit 7eba241f860a1898ea33a4a5b50baa4d6bc20a83
Author: Suhani <suhanisinghal@gmail.com>
```

```
Date:   Mon May 9 23:42:07 2016 -0400

    Fixed float arrays

commit 7b778a8a18530ff7ac828914c435b0220a3e366c
Author: Suhani <suhanisinghal@gmail.com>
Date:   Mon May 9 23:38:12 2016 -0400

    Fixed assignment to variable on rhs

commit 4dd0acdc9b163fae7a060dd065f76c1468d94543
Author: Suhani <suhanisinghal@gmail.com>
Date:   Mon May 9 22:50:32 2016 -0400

    Fixed char test

commit 3ea393063195a5276e8056e70ee688eb36aac314
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 22:15:40 2016 -0400

    fixed gitingore again i hope

commit fabc5a7550ba56b577240deaec4cf2fb3704bc25
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 22:14:19 2016 -0400

    added tests

commit 8802d5bd99b723c0531d7e6b991720fc73691ca1
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 15:02:54 2016 -0400

      fixed characters and made a test for them isolated the error with char
arrays

commit f7ec601fbe70fdf8866e33afcf907a9f2a218ea6
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 14:57:45 2016 -0400

    fixed an issue causing array tests to fail

commit 19f96ef32d7f9a6dfe3023d98096fb39fe9bc7e6
Author: Brennan Wallace <brennangoffwallace@gmail.com>
```

```
Date:   Mon May 9 14:48:24 2016 -0400

    err with chars

commit 374352e76ce70449aceefa8fa963f73eab50ba63
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 13:51:22 2016 -0400

    floats work for array printing

commit f7e8ec8699cdf6f4cd6e9a0793915a2ad95bd862
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 04:28:41 2016 -0400

    i think arrays are the symbol table wrong (as ints)

commit 58ee3edddeab5d81060daf253aa68bde3ffb5281
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 04:13:35 2016 -0400

    getting there with prints i think there is an issue with codegen loading for
diffrent types

commit abf62c5a9ac19a7abcf0ba909829e34a9e2894df
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 04:02:49 2016 -0400

    something werid going on with array printing

commit 8cdd8c01f1dfddea0842e1943cee5a4f5c1ebb8d
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 03:32:11 2016 -0400

     fixed float arrays BUT there is an issue with printing floats from them and
from neagtive floats. Flip the _ print back to print as int to see

commit cad2c5bc76f9b6b2d60ac6d5beaf76dd37f24bc9
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 03:18:42 2016 -0400

    prob with floats which may be the issue behind the float arrays right now

commit 4aadfcb1a1240cfdb9e995623557fa694481a3c4
```

```
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 03:09:41 2016 -0400

    commit progress on float arrays still broken though

commit ac4b4ca9736ea45f80575b5f086a6aab8a744907
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 02:25:24 2016 -0400

    took care of semant issues with arrays

commit 66c597aed2bb21f493d94e086f145e11dc63a45b
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:48:52 2016 -0400

    now testing int arrays

commit 24237598f09748dabf20a929fbca102164bdada8
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:31:12 2016 -0400

    fixed fail assign 3 test

commit 9ff86ecbac860cb9a3a2ca6e13ee72ba0b83584e
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:28:57 2016 -0400

    fixed fail assign 1 and 2 tests

commit 77fe2056c4e6095a1ecfc9e6030a82a0769b3d3b
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:25:42 2016 -0400

    fixed semant assgn checking

commit ab59b4d3ce26acabd9e2e9c9c5b7da5adc26354c
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:15:42 2016 -0400

    removed .out and .log files for the last time

commit c7cc05ab3aad23c4fa79be3813c09380fcd6b3e0
Author: Brennan Wallace <brennangoffwallace@gmail.com>
```

```
Date:   Mon May 9 01:14:03 2016 -0400

    altered .gitignore

commit ef11131329c24919d4253a29ff402e5a654fbcfc
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:12:12 2016 -0400

    test-var2 altered

commit aef17539ba6e3a9318dc2a528e260ab686d7ab16
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:07:16 2016 -0400

    missed a trash file its gone now

commit 481de636e1b3e60841317a3654652c1141e1bb1c
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:06:10 2016 -0400

    removed a bunch of rubbish

commit 9e2c397034a4329f8de2e302084f8208e77ad957
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 01:02:17 2016 -0400

     fixed last while test and fixed gitignore file to ignore resutls of failed
tests:

commit e3b9ddfb1ae0c93867d0d129258b96c75b126c84
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 00:54:35 2016 -0400

    Saved while and fixed some tests

commit e459d9e6f5e0bcc767faee3e61cdef52dc6c0629
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Mon May 9 00:34:21 2016 -0400

     fixed binops for bools

commit 2cdaa201632e3c597750d1866cb426774045171c
Author: Brennan Wallace <brennangoffwallace@gmail.com>
```

```
Date:   Sun May 8 23:45:02 2016 -0400

    fixed ops tests

commit 85bfb071fd0c2ad893c8c3bcb3dcc57757843ab8
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Sun May 8 23:35:31 2016 -0400

    fixed some tests

commit 35a326b1f94fdc6a974a34ba9c4bf762c0ac03f6
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Sun May 8 23:24:13 2016 -0400

      at least partially solved function input issue and boolean useage in if
statements

commit dc26e733aa4f8b4c598241499d17a9748fb2e04b
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Sun May 8 22:40:28 2016 -0400

    clean up extra files

commit 74893a476962d5d6d2770504cfaf4c68c1b1eb3f
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Sun May 8 22:39:14 2016 -0400

    cleaned up semant and made error with passing values to functions very clear
in test-local1

commit 890dc029f9cf51613f79a6d2b766f98319d4548f
Author: Suhani <suhanisinghal@gmail.com>
Date:   Sun May 8 21:42:08 2016 -0400

    fixed random OCaml bug

commit ea13b38f50b4b137a6a14481fd4f8160d69fbea3
Author: Suhani <suhanisinghal@gmail.com>
Date:   Sun May 8 04:21:27 2016 -0400

    Structs added, can load values too - printing is pending

commit 55a85ae04597578e0d00a80842d81f428597303a
```

```
Author: Suhani <suhanisinghal@gmail.com>
Date:   Sat May 7 21:45:41 2016 -0400

    Structs in progress

commit d73d7523782efeff041748b0b8649e4e71b40fa7
Author: Suhani <suhanisinghal@gmail.com>
Date:   Sat May 7 01:32:30 2016 -0400

    Int Arrays now workgit add *git add *git add * :) :) :) :)

commit eaa4557da98b9e64cb481b44a72116335e02b80c
Author: Suhani <suhanisinghal@gmail.com>
Date:   Sat May 7 01:23:35 2016 -0400

    Datatype not DataTypegit add *git add * -Code compiles now

commit 9b998573fbf9ee5e3ef2a78defb99faca4d51e75
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Sat May 7 00:34:10 2016 -0400

    broken arrays

commit d8c0771779540ea5cf489b89aeffa53f7b079d1b
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Fri May 6 21:58:35 2016 -0400

    booleans work

commit 8ba7302ef3a1f66f3b1795d91bb2fa5d47fb46d9
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Fri May 6 21:54:41 2016 -0400

    floats can print and unops and binops work for floats

commit 22d9c90c526ba0b5545932cfce615e137f48440b
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Fri May 6 21:39:48 2016 -0400

    floats have binops isnt that just fabs

commit 268a1393f423ce9d5c3c2e8c083443b4d914d3cc
Author: Brennan Wallace <brennangoffwallace@gmail.com>
```

```
Date:   Fri May 6 21:19:16 2016 -0400

    can print chars

commit 05244cd0d666305234dd2352612e8bc40dab6561
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Fri May 6 20:03:25 2016 -0400

    floats are sorta done (no ops)

commit 3c181505aeeb01678fcf8657e882a327bfb43f1d
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Fri May 6 18:38:21 2016 -0400

    **crosses fingers**

commit 50054186592256e52b93dd77e767c151e17c39ca
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue May 3 21:29:49 2016 -0400

    broken things: binops, arrays, floats

commit 2f7d6a9f5dbe52926810906a7284c175db15c6c6
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue May 3 20:23:27 2016 -0400

    floats are getting there

commit 5a135bc8a0c3dde084696e11ce080842709f7c74
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue Apr 12 20:20:24 2016 -0400

    Fixed the fail tests

commit 0ff20a2d45e5650ea246e2423a51a8d421247f21
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue Apr 12 19:58:52 2016 -0400

    Fixed test cases

commit 8bd11e6c69958b7ae9fb196089fedcdf557e6388
Author: Suhani <suhanisinghal@gmail.com>
Date:   Tue Apr 12 19:46:56 2016 -0400
```

```
    Functions inside prints and test cases

commit b027d5c95164156d93239f0ffc9fd269583e840b
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 17:32:21 2016 -0400

    removed microc.ml

commit 8d1c949b76fb7d49ea4473cecd59299bc1ac36d8
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 17:30:33 2016 -0400

    removed a microc realted string

commit acae24f122b646befbe2eb08edebc6c453cdc6cd
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 17:28:47 2016 -0400

    saving print progress

commit 4ecc1648f4ee601d4bdcecfa570ca19f43745255
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 17:19:48 2016 -0400

    cleaned up print some more

commit 9b19c8d0e9d6d2dfb25009e9bd4de3839786bfe7
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 17:06:37 2016 -0400

    tests working

commit 4683b36c82d1e88a7daba427902e85c05e9fb7b0
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 16:55:39 2016 -0400

    making progress on print

commit 6b0bbb253c8164b67693420d157e0f05f2a4f2ce
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 13:23:00 2016 -0400
```

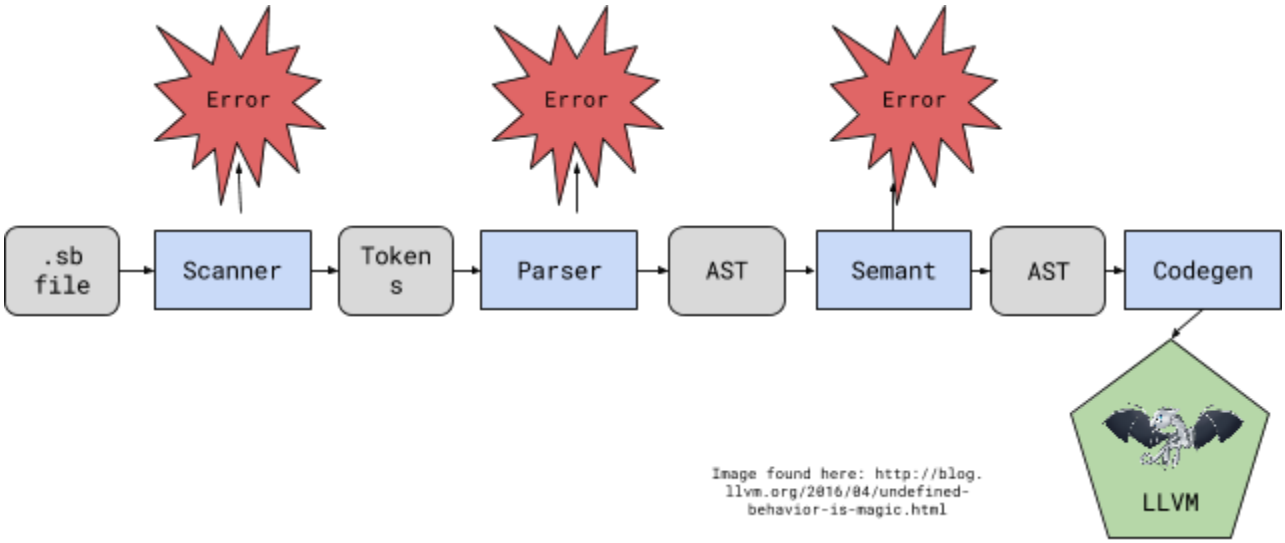
```
    changed tests etensions

commit fe390902713ae6fe047044550dfd153b28f77367
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 13:22:13 2016 -0400

    changed extension to .sb and names to albs in various files

commit 809e58d819b22096eac94e653e022e7ad3825937
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Apr 6 12:19:13 2016 -0400

    cleaned up print

commit 9ac77bd288327e21a0e75dd094c3ec4480b986ac
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue Apr 5 22:29:47 2016 -0400

    added tests

commit 4c66671ae4b98be58ba3b458a643ce9af86dc7d0
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue Apr 5 21:19:01 2016 -0400

    special characters now working

commit 38dc53e19f9a28c1e0b6ec74e4763067c94239b1
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue Apr 5 21:02:48 2016 -0400

    hello world(no newline)

commit dbf187a5c26c5d03c185fdf806f56db468d29259
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Sun Mar 27 14:45:21 2016 -0400

    main function works with a print call with an int

commit 39aad17e3275768d8f81f0e5c5fb5fa216a97958
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Sun Mar 27 12:41:13 2016 -0400

    try again
```

```
commit 5613094618b62a502d9dd6fe8751e92913e90c09
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Sun Mar 27 12:33:55 2016 -0400

    moved tests

commit 08f581ce749b8626d12b598f0c60967b0101ad51
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Wed Mar 23 17:17:47 2016 -0400

    Created a folder for out code

commit 68eba09d912189fe30285a3fc118cefe2e663e09
Merge: 975a462 1bd84f6
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue Mar 22 21:04:03 2016 -0400

    Merge branch 'master' of https://github.com/brennangw/ALBS

commit 975a462e8757432b7a50b4bfd8e98b1db8c772b8
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue Mar 22 21:03:32 2016 -0400

        added microc testing compliation then cleaned lets remember to not add an
uncleaned directory after a make

commit 1bd84f6085c607734d17734f4e3f33dcd00b2edc
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue Mar 22 20:51:07 2016 -0400

    Initial commit

commit bf8a1d64428d1935f92b2dfc461277e58f8b5e4b
Author: Brennan Wallace <brennangoffwallace@gmail.com>
Date:   Tue Mar 22 20:15:48 2016 -0400

    Initial commit
```

V. Architectural Design

A. Diagram

Below shows the flow of information through the compiler. Grey shapes are the program representation and blue shapes are the compiler components.



Compiler Components

albs.ml: top level control of compiler, sets thing in motion and conrtols if llvm is generated, if it is checked or if only the ast is produced.

codegen.ml: creates the actual LLVM code based on the abstrat syntax tree it is evaulating.

parser.mly: takes the tokenized input and creates the abstract syntax tree.

scanner.mll: tokenizes the input program file

semant.ml: semantically checks the AST given to it.

B. Interfaces Between Components

Tokens defined in the scanner provide a clean and consistent representation of the program without unnecessary details like whitespace and comments.

The next interface is the abstract syntax tree defined in the ast.ml file, which allows the structure of the program to be passed to semantic checking and to the codegen.

C. Implementation Credits for Components

Since we pair-programmed our group made the majority of components as a team effort. Some language features if done individually (which we did on occasion) would usually require the individual to alter all or at least a majority of the files comprising the compiler pipeline.

VI. Test Plan

A. Example Program 1

```
(* Defines a struct for complex number and adds to complex numbers *)
{ : int} main = [
    struct complex c1;
    struct complex c2;
    struct complex c3;

    c1.real = 1.0;
    c1.imag = -1.0;
    c2.real = 41.0;
    c2.imag = 1.0;

    c3.real = c1.real + c2.real;
    c3.imag = c1.imag + c2.imag;

    print("\n real: \n");
    print(c3.real);
    print("\n imaginary: \n");
    print(c3.imag);

    rtn 0;
]
struct complex[
    flt real;
    flt imag;
    int x;
    chr name;
]
```

LLVM generated (Target code):

```
%complex = type <{ i32, double, double, i32, i8 }>

@fmt = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@0 = private unnamed_addr constant [10 x i8] c"\0A real: \0A\00"
@1 = private unnamed_addr constant [15 x i8] c"\0A imaginary: \0A\00"

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

define i32 @main() {
entry:
  %c1 = alloca %complex
  %c2 = alloca %complex
  %c3 = alloca %complex
  %real = getelementptr inbounds %complex, %complex* %c1, i32 0, i32 1
  store double 1.000000e+00, double* %real
  %imag = getelementptr inbounds %complex, %complex* %c1, i32 0, i32 2
  store double -1.000000e+00, double* %imag
  %real1 = getelementptr inbounds %complex, %complex* %c2, i32 0, i32 1
  store double 4.100000e+01, double* %real1
  %imag2 = getelementptr inbounds %complex, %complex* %c2, i32 0, i32 2
  store double 1.000000e+00, double* %imag2
  %real3 = getelementptr inbounds %complex, %complex* %c3, i32 0, i32 1
  %real4 = getelementptr inbounds %complex, %complex* %c1, i32 0, i32 1
  %real5 = load double, double* %real4
  %real6 = getelementptr inbounds %complex, %complex* %c2, i32 0, i32 1
  %real7 = load double, double* %real6
  %tmp = fadd double %real5, %real7
  store double %tmp, double* %real3
  %imag8 = getelementptr inbounds %complex, %complex* %c3, i32 0, i32 2
  %imag9 = getelementptr inbounds %complex, %complex* %c1, i32 0, i32 2
  %imag10 = load double, double* %imag9
  %imag11 = getelementptr inbounds %complex, %complex* %c2, i32 0, i32 2
  %imag12 = load double, double* %imag11
  %tmp13 = fadd double %imag10, %imag12
  store double %tmp13, double* %imag8
```

```
  %string_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @0, i32 0, i32 0))
  %real14 = getelementptr inbounds %complex, %complex* %c3, i32 0, i32 1
  %real15 = load double, double* %real14
  %abcd = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), double %real15)
  %string_printf16 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([15 x i8], [15 x i8]* @1, i32 0, i32 0))
  %imag17 = getelementptr inbounds %complex, %complex* %c3, i32 0, i32 2
  %imag18 = load double, double* %imag17
  %abcd19 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), double %imag18)
  ret i32 0
}

declare i8 @getchar()
```

Test run:

```
real:
42.000000
imaginary:
0.000000
```

B. Example Program 2

```
(* Takes user input, stores it as a character array, counts the frequency of
every letter by casting char to an int, and displays a histogram *)
{ : int} main = [

    chr c;
    int[] charTotals;
    int i;
    int numberOfLines;
    int len;
    int numberOfChars;

    numberOfChars = 27;
    i = 0;
    numberOfLines = 0;
    charTotals = new int[numberOfChars];
    zeroOutArray(charTotals, numberOfChars);
    countFrequency(charTotals, numberOfChars);
    numberOfLines = highestIntOfArray(charTotals, numberOfChars);
    printHistogram(charTotals, numberOfChars, numberOfLines);
    rtn 0;
]

{ int[] int : void } zeroOutArray = intArr sizeOfArray [
  int i;
  i = 0;
  while (i < sizeOfArray) {
    intArr[i] = 0;
    i = i + 1;
  }
]

{ int[] int int: void } printHistogram = countArr sizeOfArray maxSize [

  int outerCount;
  int innerCount;
```

```
  int i;
  int currentValue;

  print("\nHistogram follows:\n");

  outerCount = maxSize;
  while(outerCount > 0) {
    innerCount = 0;
    while(innerCount < sizeOfArray) {
      currentValue = countArr[innerCount];
      if (currentValue >= outerCount) {
        print("x ");
      } else {
        print("  ");
      }
      innerCount = innerCount + 1;
    }
    print("\n");
    outerCount = outerCount - 1;
  }
  print("a b c d e f g h i j k l m n o p q r s t u v w x y z other\n");

]

{ int[] int : void } printIntArray = intArr sizeOfArray [
  int i;
  int s;
  i = 0;
  while (i < sizeOfArray) {
    s = intArr[i];
    print(s);
    i = i + 1;
  }
  print("\n");
]
{int[] int : int } highestIntOfArray = intArr sizeOfArray [
  int highestInt;
  int i;
  int current;
```

```
  highestInt = 0;
  i = 0;
  while (i < sizeOfArray) {
      current = intArr[i];
      if (current > highestInt) {
          highestInt = current;
        }
      i = i + 1;
  }
  rtn highestInt;
]
{int[] int : void } countFrequency = intArr sizeOfArray [
  int i;
  int h;
  int cAsInt;
  int charNum;
  chr c;
  int temp;
  int j;
  while(c != '\r'){
        c = getchar();
        j = addCharToInt(c,-97); (*adds as -97 +'ch_val')
        if(j<0){
          j = 26;
       }
       if(j>25){
           j = 26;
       }
      temp = intArr[j]; (* index of char *)
      temp = temp + 1; (* increment by 1 *)
      intArr[j]= temp;
      i = i + 1;
    }
  rtn;
]
{chr int : int} addChrToInt = c offset [
      (* int = int + char *)
    if (c == 'a' ) {
        offset = offset + 'a';
```

```
   }
   if (c == 'b' ) {
       offset = offset + 'b';
   }
 if (c == 'c' ) {
       offset = offset + 'c';
  }
 if (c == 'd' ) {
       offset = offset + 'd';
  }
 if (c == 'e' ) {
       offset = offset + 'e';
  }
 if (c == 'f' ) {
       offset = offset + 'f';
  }
 if (c == 'g' ) {
       offset = offset + 'g';
  }
 if (c == 'h' ) {
       offset = offset + 'h';
  }
 if (c == 'i' ) {
       offset = offset + 'i';
  }
 if (c == 'j' ) {
       offset = offset + 'j';
  }
 if (c == 'k' ) {
       offset = offset + 'k';
  }
 if (c == 'l' ) {
       offset = offset + 'l';
  }
 if (c == 'm' ) {
       offset = offset + 'm';
  }
 if (c == 'n' ) {
       offset = offset + 'n';
```

```
  }
 if (c == 'o' ) {
       offset = offset + 'o';
  }
 if (c == 'p' ) {
       offset = offset + 'p';
  }
 if (c == 'q' ) {
       offset = offset + 'q';
  }
 if (c == 'r' ) {
       offset = offset + 'r';
  }
 if (c == 's' ) {
       offset = offset + 's';
  }
 if (c == 't' ) {
       offset = offset + 't';
  }
 if (c == 'u' ) {
       offset = offset + 'u';
  }
 if (c == 'v' ) {
       offset = offset + 'v';
  }
 if (c == 'w' ) {
       offset = offset + 'w';
  }
 if (c == 'x' ) {
       offset = offset + 'x';
  }
 if (c == 'y' ) {
       offset = offset + 'y';
  }
 if (c == 'z' ) {
       offset = offset + 'z';
  }
  rtn offset;
]
```

LLVM generated (Target code):

```
@fmt = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.7 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.10 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.11 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.12 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.13 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.14 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.15 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@0 = private unnamed_addr constant [2 x i8] c"\0A\00"
@fmt.16 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.17 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.18 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.19 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@1 = private unnamed_addr constant [21 x i8] c"\0AHistogram follows:\0A\00"
@2 = private unnamed_addr constant [3 x i8] c"x \00"
@3 = private unnamed_addr constant [3 x i8] c"  \00"
@4 = private unnamed_addr constant [2 x i8] c"\0A\00"
@5 = private unnamed_addr constant [53 x i8] c"a b c d e f g h i j k l m n o p q r s t u v w x y z other\0A\00"
@fmt.20 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.21 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.22 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.23 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.24 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.25 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.26 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.27 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
declare i32 @printf(i8*, ...)
define i32 @addChrToInt(i8 %c, i32 %offset) {
entry:
  %c1 = alloca i8
  store i8 %c, i8* %c1
  %offset2 = alloca i32
  store i32 %offset, i32* %offset2
  %c3 = load i8, i8* %c1
  %tmp = icmp eq i8 %c3, 97
  br i1 %tmp, label %then, label %else
merge:                                            ; preds = %else, %then
  %c6 = load i8, i8* %c1
  %tmp7 = icmp eq i8 %c6, 98
  br i1 %tmp7, label %then9, label %else12
then:                                             ; preds = %entry
  %offset4 = load i32, i32* %offset2
  %tmp5 = add i32 %offset4, 97
  store i32 %tmp5, i32* %offset2
  br label %merge
else:                                             ; preds = %entry
  br label %merge
merge8:                                           ; preds = %else12, %then9
  %c13 = load i8, i8* %c1
  %tmp14 = icmp eq i8 %c13, 99
  br i1 %tmp14, label %then16, label %else19
```

```
then9:                                            ; preds = %merge
  %offset10 = load i32, i32* %offset2
  %tmp11 = add i32 %offset10, 98
  store i32 %tmp11, i32* %offset2
  br label %merge8
else12:                                           ; preds = %merge
  br label %merge8
merge15:                                          ; preds = %else19, %then16
  %c20 = load i8, i8* %c1
  %tmp21 = icmp eq i8 %c20, 100
  br i1 %tmp21, label %then23, label %else26
then16:                                           ; preds = %merge8
  %offset17 = load i32, i32* %offset2
  %tmp18 = add i32 %offset17, 99
  store i32 %tmp18, i32* %offset2
  br label %merge15
else19:                                           ; preds = %merge8
  br label %merge15

merge22:                                          ; preds = %else26, %then23
  %c27 = load i8, i8* %c1
  %tmp28 = icmp eq i8 %c27, 101
  br i1 %tmp28, label %then30, label %else33
then23:                                           ; preds = %merge15
  %offset24 = load i32, i32* %offset2
  %tmp25 = add i32 %offset24, 100
  store i32 %tmp25, i32* %offset2
  br label %merge22
else26:                                           ; preds = %merge15
  br label %merge22
merge29:                                          ; preds = %else33, %then30
  %c34 = load i8, i8* %c1
  %tmp35 = icmp eq i8 %c34, 102
  br i1 %tmp35, label %then37, label %else40
then30:                                           ; preds = %merge22
  %offset31 = load i32, i32* %offset2
  %tmp32 = add i32 %offset31, 101
  store i32 %tmp32, i32* %offset2
  br label %merge29
else33:                                           ; preds = %merge22
  br label %merge29
merge36:                                          ; preds = %else40, %then37
  %c41 = load i8, i8* %c1
  %tmp42 = icmp eq i8 %c41, 103
  br i1 %tmp42, label %then44, label %else47
then37:                                           ; preds = %merge29
  %offset38 = load i32, i32* %offset2
  %tmp39 = add i32 %offset38, 102
  store i32 %tmp39, i32* %offset2
  br label %merge36
else40:                                           ; preds = %merge29
  br label %merge36
merge43:                                          ; preds = %else47, %then44
  %c48 = load i8, i8* %c1
  %tmp49 = icmp eq i8 %c48, 104
  br i1 %tmp49, label %then51, label %else54
then44:                                           ; preds = %merge36
  %offset45 = load i32, i32* %offset2
  %tmp46 = add i32 %offset45, 103
  store i32 %tmp46, i32* %offset2
  br label %merge43
else47:                                           ; preds = %merge36
```

```
  br label %merge43
merge50:                                          ; preds = %else54, %then51
  %c55 = load i8, i8* %c1
  %tmp56 = icmp eq i8 %c55, 105
  br i1 %tmp56, label %then58, label %else61
then51:                                           ; preds = %merge43
  %offset52 = load i32, i32* %offset2
  %tmp53 = add i32 %offset52, 104
  store i32 %tmp53, i32* %offset2
  br label %merge50
else54:                                           ; preds = %merge43
  br label %merge50
merge57:                                          ; preds = %else61, %then58
  %c62 = load i8, i8* %c1
  %tmp63 = icmp eq i8 %c62, 106
  br i1 %tmp63, label %then65, label %else68
then58:                                           ; preds = %merge50
  %offset59 = load i32, i32* %offset2
  %tmp60 = add i32 %offset59, 105
  store i32 %tmp60, i32* %offset2
  br label %merge57
else61:                                           ; preds = %merge50
  br label %merge57
merge64:                                          ; preds = %else68, %then65
  %c69 = load i8, i8* %c1
  %tmp70 = icmp eq i8 %c69, 107
  br i1 %tmp70, label %then72, label %else75
then65:                                           ; preds = %merge57
  %offset66 = load i32, i32* %offset2
  %tmp67 = add i32 %offset66, 106
  store i32 %tmp67, i32* %offset2
  br label %merge64
else68:                                           ; preds = %merge57
  br label %merge64
merge71:                                          ; preds = %else75, %then72
  %c76 = load i8, i8* %c1
  %tmp77 = icmp eq i8 %c76, 108
  br i1 %tmp77, label %then79, label %else82
then72:                                           ; preds = %merge64
  %offset73 = load i32, i32* %offset2
  %tmp74 = add i32 %offset73, 107
  store i32 %tmp74, i32* %offset2
  br label %merge71
else75:                                           ; preds = %merge64
  br label %merge71
merge78:                                          ; preds = %else82, %then79
  %c83 = load i8, i8* %c1
  %tmp84 = icmp eq i8 %c83, 109
  br i1 %tmp84, label %then86, label %else89
then79:                                           ; preds = %merge71
  %offset80 = load i32, i32* %offset2
  %tmp81 = add i32 %offset80, 108
  store i32 %tmp81, i32* %offset2
  br label %merge78
else82:                                           ; preds = %merge71
  br label %merge78
merge85:                                          ; preds = %else89, %then86
  %c90 = load i8, i8* %c1
  %tmp91 = icmp eq i8 %c90, 110
  br i1 %tmp91, label %then93, label %else96
then86:                                           ; preds = %merge78
  %offset87 = load i32, i32* %offset2
```

```
  %tmp88 = add i32 %offset87, 109
  store i32 %tmp88, i32* %offset2
  br label %merge85
else89:                                           ; preds = %merge78
  br label %merge85
merge92:                                          ; preds = %else96, %then93
  %c97 = load i8, i8* %c1
  %tmp98 = icmp eq i8 %c97, 111
  br i1 %tmp98, label %then100, label %else103
then93:                                           ; preds = %merge85
  %offset94 = load i32, i32* %offset2
  %tmp95 = add i32 %offset94, 110
  store i32 %tmp95, i32* %offset2
  br label %merge92
else96:                                           ; preds = %merge85
  br label %merge92
merge99:                                          ; preds = %else103, %then100
  %c104 = load i8, i8* %c1
  %tmp105 = icmp eq i8 %c104, 112
  br i1 %tmp105, label %then107, label %else110
then100:                                          ; preds = %merge92
  %offset101 = load i32, i32* %offset2
  %tmp102 = add i32 %offset101, 111
  store i32 %tmp102, i32* %offset2
  br label %merge99
else103:                                          ; preds = %merge92
  br label %merge99
merge106:                                         ; preds = %else110, %then107
  %c111 = load i8, i8* %c1
  %tmp112 = icmp eq i8 %c111, 113
  br i1 %tmp112, label %then114, label %else117
then107:                                          ; preds = %merge99
  %offset108 = load i32, i32* %offset2
  %tmp109 = add i32 %offset108, 112
  store i32 %tmp109, i32* %offset2
  br label %merge106
else110:                                          ; preds = %merge99
  br label %merge106
merge113:                                         ; preds = %else117, %then114
  %c118 = load i8, i8* %c1
  %tmp119 = icmp eq i8 %c118, 114
  br i1 %tmp119, label %then121, label %else124
then114:                                          ; preds = %merge106
  %offset115 = load i32, i32* %offset2
  %tmp116 = add i32 %offset115, 113
  store i32 %tmp116, i32* %offset2
  br label %merge113
else117:                                          ; preds = %merge106
  br label %merge113
merge120:                                         ; preds = %else124, %then121
  %c125 = load i8, i8* %c1
  %tmp126 = icmp eq i8 %c125, 115
  br i1 %tmp126, label %then128, label %else131
then121:                                          ; preds = %merge113
  %offset122 = load i32, i32* %offset2
  %tmp123 = add i32 %offset122, 114
  store i32 %tmp123, i32* %offset2
  br label %merge120
else124:                                          ; preds = %merge113
  br label %merge120
merge127:                                         ; preds = %else131, %then128
  %c132 = load i8, i8* %c1
```

```
  %tmp133 = icmp eq i8 %c132, 116
  br i1 %tmp133, label %then135, label %else138
then128:                                          ; preds = %merge120
  %offset129 = load i32, i32* %offset2
  %tmp130 = add i32 %offset129, 115
  store i32 %tmp130, i32* %offset2
  br label %merge127
else131:                                          ; preds = %merge120
  br label %merge127
merge134:                                         ; preds = %else138, %then135
  %c139 = load i8, i8* %c1
  %tmp140 = icmp eq i8 %c139, 117
  br i1 %tmp140, label %then142, label %else145
then135:                                          ; preds = %merge127
  %offset136 = load i32, i32* %offset2
  %tmp137 = add i32 %offset136, 116
  store i32 %tmp137, i32* %offset2
  br label %merge134
else138:                                          ; preds = %merge127
  br label %merge134
merge141:                                         ; preds = %else145, %then142
  %c146 = load i8, i8* %c1
  %tmp147 = icmp eq i8 %c146, 118
  br i1 %tmp147, label %then149, label %else152
then142:                                          ; preds = %merge134
  %offset143 = load i32, i32* %offset2
  %tmp144 = add i32 %offset143, 117
  store i32 %tmp144, i32* %offset2
  br label %merge141
else145:                                          ; preds = %merge134
  br label %merge141
merge148:                                         ; preds = %else152, %then149
  %c153 = load i8, i8* %c1
  %tmp154 = icmp eq i8 %c153, 119
  br i1 %tmp154, label %then156, label %else159
then149:                                          ; preds = %merge141
  %offset150 = load i32, i32* %offset2
  %tmp151 = add i32 %offset150, 118
  store i32 %tmp151, i32* %offset2
  br label %merge148
else152:                                          ; preds = %merge141
  br label %merge148
merge155:                                         ; preds = %else159, %then156
  %c160 = load i8, i8* %c1
  %tmp161 = icmp eq i8 %c160, 120
  br i1 %tmp161, label %then163, label %else166
then156:                                          ; preds = %merge148
  %offset157 = load i32, i32* %offset2
  %tmp158 = add i32 %offset157, 119
  store i32 %tmp158, i32* %offset2
  br label %merge155
else159:                                          ; preds = %merge148
  br label %merge155
merge162:                                         ; preds = %else166, %then163
  %c167 = load i8, i8* %c1
  %tmp168 = icmp eq i8 %c167, 121
  br i1 %tmp168, label %then170, label %else173
then163:                                          ; preds = %merge155
  %offset164 = load i32, i32* %offset2
  %tmp165 = add i32 %offset164, 120
  store i32 %tmp165, i32* %offset2
  br label %merge162
```

```
else166:                                          ; preds = %merge155
  br label %merge162
merge169:                                         ; preds = %else173, %then170
  %c174 = load i8, i8* %c1
  %tmp175 = icmp eq i8 %c174, 122
  br i1 %tmp175, label %then177, label %else180
then170:                                          ; preds = %merge162
  %offset171 = load i32, i32* %offset2
  %tmp172 = add i32 %offset171, 121
  store i32 %tmp172, i32* %offset2
  br label %merge169
else173:                                          ; preds = %merge162
  br label %merge169
merge176:                                         ; preds = %else180, %then177
  %offset181 = load i32, i32* %offset2
  ret i32 %offset181
then177:                                          ; preds = %merge169
  %offset178 = load i32, i32* %offset2
  %tmp179 = add i32 %offset178, 122
  store i32 %tmp179, i32* %offset2
  br label %merge176
else180:                                          ; preds = %merge169
  br label %merge176
}
define void @countFrequency(i32* %intArr, i32 %sizeOfArray) {
entry:
  %intArr1 = alloca i32*
  store i32* %intArr, i32** %intArr1
  %sizeOfArray2 = alloca i32
  store i32 %sizeOfArray, i32* %sizeOfArray2
  %i = alloca i32
  %h = alloca i32
  %cAsInt = alloca i32
  %charNum = alloca i32
  %c = alloca i8
  %temp = alloca i32
  %j = alloca i32
  br label %while
while:                                            ; preds = %merge8, %entry
  %c25 = load i8, i8* %c
  %tmp26 = icmp ne i8 %c25, 92
  br i1 %tmp26, label %while_body, label %merge27
while_body:                                       ; preds = %while
  %tmp = call i8 @getchar()
  store i8 %tmp, i8* %c
  %c3 = load i8, i8* %c
  %addChrToInt_result = call i32 @addChrToInt(i8 %c3, i32 -97)
  store i32 %addChrToInt_result, i32* %j
  %j4 = load i32, i32* %j
  %tmp5 = icmp slt i32 %j4, 0
  br i1 %tmp5, label %then, label %else
merge:                                            ; preds = %else, %then
  %j6 = load i32, i32* %j
  %tmp7 = icmp sgt i32 %j6, 25
  br i1 %tmp7, label %then9, label %else10
then:                                             ; preds = %while_body
  store i32 26, i32* %j
  br label %merge
```

```
else:                                             ; preds = %while_body
  br label %merge
merge8:                                           ; preds = %else10, %then9
  %j11 = load i32, i32* %j
  %tmp12 = add i32 %j11, 1
  %intArr13 = load i32*, i32** %intArr1
  %tmp14 = getelementptr i32, i32* %intArr13, i32 %tmp12
  %tmp15 = load i32, i32* %tmp14
  store i32 %tmp15, i32* %temp
  %temp16 = load i32, i32* %temp
  %tmp17 = add i32 %temp16, 1
  store i32 %tmp17, i32* %temp
  %j18 = load i32, i32* %j
  %tmp19 = add i32 %j18, 1
  %intArr20 = load i32*, i32** %intArr1
  %tmp21 = getelementptr i32, i32* %intArr20, i32 %tmp19
  %temp22 = load i32, i32* %temp
  store i32 %temp22, i32* %tmp21
  %i23 = load i32, i32* %i
  %tmp24 = add i32 %i23, 1
  store i32 %tmp24, i32* %i
  br label %while
then9:                                            ; preds = %merge
  store i32 26, i32* %j
  br label %merge8
else10:                                           ; preds = %merge
  br label %merge8
merge27:                                          ; preds = %while
  ret void
}
define i32 @highestIntOfArray(i32* %intArr, i32 %sizeOfArray) {
entry:
  %intArr1 = alloca i32*
  store i32* %intArr, i32** %intArr1
  %sizeOfArray2 = alloca i32
  store i32 %sizeOfArray, i32* %sizeOfArray2
  %highestInt = alloca i32
  %i = alloca i32
  %current = alloca i32
  store i32 0, i32* %highestInt
  store i32 0, i32* %i
  br label %while
while:                                            ; preds = %merge, %entry
  %i13 = load i32, i32* %i
  %sizeOfArray14 = load i32, i32* %sizeOfArray2
  %tmp15 = icmp slt i32 %i13, %sizeOfArray14
  br i1 %tmp15, label %while_body, label %merge16
while_body:                                       ; preds = %while
  %i3 = load i32, i32* %i
  %tmp = add i32 %i3, 1
  %intArr4 = load i32*, i32** %intArr1
  %tmp5 = getelementptr i32, i32* %intArr4, i32 %tmp
  %tmp6 = load i32, i32* %tmp5
  store i32 %tmp6, i32* %current
  %current7 = load i32, i32* %current
```

```
  %highestInt8 = load i32, i32* %highestInt
  %tmp9 = icmp sgt i32 %current7, %highestInt8
  br i1 %tmp9, label %then, label %else
merge:                                            ; preds = %else, %then
  %i11 = load i32, i32* %i
  %tmp12 = add i32 %i11, 1
  store i32 %tmp12, i32* %i
  br label %while
then:                                             ; preds = %while_body
  %current10 = load i32, i32* %current
  store i32 %current10, i32* %highestInt
  br label %merge
else:                                             ; preds = %while_body
  br label %merge
merge16:                                          ; preds = %while
  %highestInt17 = load i32, i32* %highestInt
  ret i32 %highestInt17
}
define void @printIntArray(i32* %intArr, i32 %sizeOfArray) {
entry:
  %intArr1 = alloca i32*
  store i32* %intArr, i32** %intArr1
  %sizeOfArray2 = alloca i32
  store i32 %sizeOfArray, i32* %sizeOfArray2
  %i = alloca i32
  %s = alloca i32
  store i32 0, i32* %i
  br label %while
while:                                            ; preds = %while_body, %entry
  %i10 = load i32, i32* %i
  %sizeOfArray11 = load i32, i32* %sizeOfArray2
  %tmp12 = icmp slt i32 %i10, %sizeOfArray11
  br i1 %tmp12, label %while_body, label %merge
while_body:                                       ; preds = %while
  %i3 = load i32, i32* %i
  %tmp = add i32 %i3, 1
  %intArr4 = load i32*, i32** %intArr1
  %tmp5 = getelementptr i32, i32* %intArr4, i32 %tmp
  %tmp6 = load i32, i32* %tmp5
  store i32 %tmp6, i32* %s
  %s7 = load i32, i32* %s
  %int_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.13, i32 0, i32 0), i32 %s7)
  %i8 = load i32, i32* %i
  %tmp9 = add i32 %i8, 1
  store i32 %tmp9, i32* %i
  br label %while
merge:                                            ; preds = %while
  %string_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x i8]* @0, i32 0, i32 0))
  ret void
}
define void @printHistogram(i32* %countArr, i32 %sizeOfArray, i32 %maxSize) {
entry:
  %countArr1 = alloca i32*
  store i32* %countArr, i32** %countArr1
  %sizeOfArray2 = alloca i32
```

```
  store i32 %sizeOfArray, i32* %sizeOfArray2
  %maxSize3 = alloca i32
  store i32 %maxSize, i32* %maxSize3
  %outerCount = alloca i32
  %innerCount = alloca i32
  %i = alloca i32
  %currentValue = alloca i32
  %string_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([21 x i8], [21 x i8]* @1, i32 0, i32 0))
  %maxSize4 = load i32, i32* %maxSize3
  store i32 %maxSize4, i32* %outerCount
  br label %while
while:                                            ; preds = %merge21, %entry
  %outerCount25 = load i32, i32* %outerCount
  %tmp26 = icmp sgt i32 %outerCount25, 0
  br i1 %tmp26, label %while_body, label %merge27
while_body:                                       ; preds = %while
  store i32 0, i32* %innerCount
  br label %while5
while5:                                           ; preds = %merge, %while_body
  %innerCount18 = load i32, i32* %innerCount
  %sizeOfArray19 = load i32, i32* %sizeOfArray2
  %tmp20 = icmp slt i32 %innerCount18, %sizeOfArray19
  br i1 %tmp20, label %while_body6, label %merge21
while_body6:                                      ; preds = %while5
  %innerCount7 = load i32, i32* %innerCount
  %tmp = add i32 %innerCount7, 1
  %countArr8 = load i32*, i32** %countArr1
  %tmp9 = getelementptr i32, i32* %countArr8, i32 %tmp
  %tmp10 = load i32, i32* %tmp9
  store i32 %tmp10, i32* %currentValue
  %currentValue11 = load i32, i32* %currentValue
  %outerCount12 = load i32, i32* %outerCount
  %tmp13 = icmp sge i32 %currentValue11, %outerCount12
  br i1 %tmp13, label %then, label %else
merge:                                            ; preds = %else, %then
  %innerCount16 = load i32, i32* %innerCount
  %tmp17 = add i32 %innerCount16, 1
  store i32 %tmp17, i32* %innerCount
  br label %while5
then:                                             ; preds = %while_body6
  %string_printf14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @2, i32 0, i32 0))
  br label %merge
else:                                             ; preds = %while_body6
  %string_printf15 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @3, i32 0, i32 0))
  br label %merge
merge21:                                          ; preds = %while5
  %string_printf22 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x i8]* @4, i32 0, i32 0))
  %outerCount23 = load i32, i32* %outerCount
  %tmp24 = sub i32 %outerCount23, 1
  store i32 %tmp24, i32* %outerCount
  br label %while
merge27:                                          ; preds = %while
  %string_printf28 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([53 x i8], [53 x i8]* @5, i32 0, i32 0))
  ret void
}
```

```
define void @zeroOutArray(i32* %intArr, i32 %sizeOfArray) {
entry:
  %intArr1 = alloca i32*
  store i32* %intArr, i32** %intArr1
  %sizeOfArray2 = alloca i32
  store i32 %sizeOfArray, i32* %sizeOfArray2
  %i = alloca i32
  store i32 0, i32* %i
  br label %while
while:                                            ; preds = %while_body, %entry
  %i8 = load i32, i32* %i
  %sizeOfArray9 = load i32, i32* %sizeOfArray2
  %tmp10 = icmp slt i32 %i8, %sizeOfArray9
  br i1 %tmp10, label %while_body, label %merge
while_body:                                       ; preds = %while
  %i3 = load i32, i32* %i
  %tmp = add i32 %i3, 1
  %intArr4 = load i32*, i32** %intArr1
  %tmp5 = getelementptr i32, i32* %intArr4, i32 %tmp
  store i32 0, i32* %tmp5
  %i6 = load i32, i32* %i
  %tmp7 = add i32 %i6, 1
  store i32 %tmp7, i32* %i
  br label %while
merge:                                            ; preds = %while
  ret void
}
define i32 @main() {
entry:
  %c = alloca i8
  %charTotals = alloca i32*
  %i = alloca i32
  %numberOfLines = alloca i32
  %len = alloca i32
  %numberOfChars = alloca i32
  store i32 27, i32* %numberOfChars
  store i32 0, i32* %i
  store i32 0, i32* %numberOfLines
  %numberOfChars1 = load i32, i32* %numberOfChars
  %tmp = mul i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32), %numberOfChars1
  %arr_size = add i32 %tmp, 1
  %mallocsize = mul i32 %arr_size, ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %tmp2 = bitcast i8* %malloccall to i32*
  store i32 %arr_size, i32* %tmp2
  br label %array.cond
array.cond:                                       ; preds = %array.init, %entry
  %counter = phi i32 [ 0, %entry ], [ %tmp3, %array.init ]
  %tmp3 = add i32 %counter, 1
  %tmp4 = icmp slt i32 %counter, %arr_size
  br i1 %tmp4, label %array.init, label %array.done
array.init:                                       ; preds = %array.cond
  %tmp5 = getelementptr i32, i32* %tmp2, i32 %counter
  store i32 0, i32* %tmp5
  br label %array.cond
```

```
array.done:                                       ; preds = %array.cond
  store i32* %tmp2, i32** %charTotals
  %numberOfChars6 = load i32, i32* %numberOfChars
  %charTotals7 = load i32*, i32** %charTotals
  call void @zeroOutArray(i32* %charTotals7, i32 %numberOfChars6)
  %numberOfChars8 = load i32, i32* %numberOfChars
  %charTotals9 = load i32*, i32** %charTotals
  call void @countFrequency(i32* %charTotals9, i32 %numberOfChars8)
  %numberOfChars10 = load i32, i32* %numberOfChars
  %charTotals11 = load i32*, i32** %charTotals
  %highestIntOfArray_result = call i32 @highestIntOfArray(i32* %charTotals11, i32 %numberOfChars10)
  store i32 %highestIntOfArray_result, i32* %numberOfLines
  %numberOfLines12 = load i32, i32* %numberOfLines
  %numberOfChars13 = load i32, i32* %numberOfChars
  %charTotals14 = load i32*, i32** %charTotals
  call void @printHistogram(i32* %charTotals14, i32 %numberOfChars13, i32 %numberOfLines12)
  ret i32 0
}
declare i8 @getchar()
declare noalias i8* @malloc(i32)
```

Run:

```
./test-histogram
sdcvrthbgnh4757rf4wawsxcswerhtvAAAerdcsaaaag\r

Histogram follows:
                                                  x
                                                  x
                                                  x
                                                  x
x                                                 x
x                                 x x             x
x   x         x                   x x       x     x
x   x x x   x x                   x x x   x x     x
x x x x x x x           x         x x x   x x x   x
a b c d e f g h i j k l m n o p q r s t u v w x y z other
```

C. Test Suite

Each test in the test suite is named after the feature that they test. Thus it is easy to tell what the test corresponds to. The output and the .sb files are paired by name so example.sb corresponds to example.out.

1. Error Tests

fail-assign1

fail-assign2

fail-assign3

fail-dead1

fail-dead2

fail-expr1

fail-expr2

2. Array Tests

test-array1

test-array2

test-array3

test-array4

test-array5

Test-array6

3. Char Tests

test-chars1

4. Struct Tests

test-structs1

test-structs2

test-structs3

test-struct-binop

test-struct-unop

5. Control Flow Tests

test-if1

test-if2

test-if3

test-if4

test-if5

test-while1

test-while2

6. Other Tests

test-local1

test-local2

test-ops1

test-ops2

test-var1

test-var2

test-var3

B. Rationale Behind Test Suites

Test cases were added to test new features as we created them. Therefore a test's success depends only on the feature we want to test (what the test is named after) and functionality that was already tested. Of course at the very beginning this wasn't true because you do have to have some minimum of working components to even have the most basic tests (ints, main function and printing for example). So essentially we tried to have at least one test for each feature and then a few larger tests (the example programs) that tested large scale and complicated integration between those features.

C. Test Suite Automation

A high-level of automation for testing was key for an efficient workflow. We used a bash script file that would test all of the tests written so far (and placed in the correct directory) using a single command. The script would then run our compiler against all of the input files of the tests, run the resulting code, and then compare the output of the resulting code to the corresponding file for each test (a test was two files the ALBS program and the expected output) that contained what the test's output "should" be. For error tests the compiler error was determined before hand and compared to the result of actually running the compiler.

The script also displayed the output of each test, labeled by test, in the terminal for easy inspection with a conclusion of the test passing or not. If tests failed a text file was automatically generated with either the output that was generated of the compiler or the failure message along with what the test should have either outputted or caused as an error.

D. Implementation Credits for Testing

Since our group did pair programming, tests were largely co-written and Prof. Edward's microC test framework was invaluable for setting up the testing framework. However, if we did development on our own, we would write tests for the new features to show they were really complete. Both of us added a few tests after the feature to bolster our testing and these would sometimes reveal errors and bugs, so it was a very good idea to do so.

VII. Lessons Learned

A. Lessons Team Members Learned

Suhani

This project was the best way to comprehend how a compiler works. As a novice to functional and LLVM - the learning curve was steep - however, time and effort paid off well. Introduction to this new programming paradigm was delightful - I can't imagine using OOP to build this. Playing God to my own language helped me appreciate the forethought that goes into structuring your data at every single stage of the compiler. This project made PLT one of the best classes I have ever taken.

Brennan

Pair programming is a very effective paradigm when working with the right people on the right project. Doing pair programming on a project with many new conceptual components was really useful as I had someone to dispel inaccurate mental models. Also from an efficiency angle pair programming is really effective when working with a programming paradigm and/or language that is novel to both people as the pair can learn from each other and are less likely to get stuck for long periods as one of them will have an idea to get out of whatever the current issue is.

B. Advice For Future Teams

Suhani

You will detest OCaml - expect to be bad at it, invest time and don't expect to be good at it anytime soon. You will regret not starting your *codegen* early - start it, rather than fiddling around with language syntax. You might, like me, loop through 10 different team members - before ending up with only one - friends and strangers drop this class and slack off! Form a group and pray for the best. Compiling to LLVM (that emulates assembly) is the best way to *learn* PLT.

Brennan

Go to the teaching assistants early and often and not just your required meetings with your mentor! Especially because the LLVM documentation is lackluster (to put it nicely) and the OCaml documentation is not that much better the TAs are an invaluable resource for getting not just a better understanding of the conceptual and architectural features, but also help with smaller issues like the tricky OCaml programming issues you will run into. It might be a good idea to just have your weekly meetings during TA hours (though do try to really solve the problem and understand the issue before you ask the TAs for help).

VIII. Appendix

albs.ml

```
(* Top-level of the ALBS compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | LLVM_IR | Compile

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

testall.sh

```
#!/bin/sh

# Regression testing script for ALBS
# 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="/usr/local/opt/llvm/bin/lli"

# Path to the albs compiler.  Usually "./albs.native"
```

```
# Try "_build/albs.native" if ocamlbuild was unable to create a symbolic link.
ALBS="./albs.native"
#ALBS="_build/albs.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] [.sb 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
}

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
    Run "$ALBS" "<" $1 ">" "${basename}.ll" &&
    Run "$LLI" "${basename}.ll" ">" "${basename}.out" &&
```

```
    Run "cat" "${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
}

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
    Run "$ALBS" "<" $1 ">" "${basename}.ll" &&
    Run "$LLI" "${basename}.ll" ">" "${basename}.out" &&
    # Run "cat" "${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/.sb//'`
    reffile=`echo $1 | sed 's/.sb$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$ALBS" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
    # "echo was:" &&
    # "cat" "${basename}.err" &&
    # "echo should be:" &&
    # "cat" "${reffile}.err" &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
      if [ $keep -eq 0 ] ; then
```

```
          rm -f $generatedfiles
      fi
      echo "OK"
      echo "###### SUCCESS" 1>&2
    else
      echo "###### FAILED" 1>&2
      globalerror=$error
    fi
}

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

shift `expr $OPTIND - 1`

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

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


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

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

exit $globalerror
```

scanner.mll

```
(* Ocamllex scanner for ALBS *)
{ open Parser
    let unescape s =
      Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}
let digit = ['0'-'9']
let float = '-'?(digit+) ['.'] digit+
let bool = "true" | "false"
let escaped_char = '\\' ['\\' ''' '"' 'n' 'r' 't']
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let char = ''' (ascii | digit) ''' | ''' escaped_char '''
let string = '"' ( (ascii | escaped_char)* as s) '"'

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "(*"     { comment lexbuf }           (* Comments *)
```

```
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LSBRACE }
| ']'      { RSBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| float as lxm { FLOAT_LITERAL(float_of_string lxm) }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| ':'      { COLON }
| "while"  { WHILE }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&"      { AND }
| "|"      { OR }
| "!"      { NOT }
| "if"     { IF }
| "for"    { FOR }
| "else"   { ELSE }
| "rtn"    { RETURN }
| "int"    { INT }
| "flt"    { FLOAT }
| "bln"    { BOOL }
| "chr"    { CHAR }
| "void"   { VOID }
| "new"    { NEW }
| "struct" { STRUCT }
| "." { DOT }
| bool as lxm { BOOLEAN_LITERAL (bool_of_string lxm) }
| char as lxm  { CHAR_LITERAL( String.get lxm 1 ) }
```

```
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof       { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
| string    { STRING_LITERAL(unescape s) }
and comment = parse
  "*)" { token lexbuf }
| _    { comment lexbuf }
```

parser.mly

```
/* Ocamlyacc parser for ALBS */
%{
open Ast
let unescape s =
  Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
%}

%token SEMI LPAREN RPAREN LSBRACE RSBRACE LBRACE RBRACE COMMA COLON NEW STRUCT DOT
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE INT FLOAT BOOL VOID CHAR
%token <int> LITERAL
%token <bool> BOOLEAN_LITERAL
%token <float> FLOAT_LITERAL
%token <char> CHAR_LITERAL
%token <string> STRING_LITERAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
```

```
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
 | vdecl_list fdecl_list sdecl_list { $1, $2, $3}
 | /* nothing */ { [], [], [] }

sdecl_list:
    /* nothing */    { [] }
  | sdecl_list sdecl { $2 :: $1 }

sdecl:
  STRUCT ID LSBRACE vdecl_list RSBRACE
  { {
   sname = $2;
   svar_decl_list = List.rev $4;
  } }

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

fdecl:
    LBRACE param_types_list COLON datatype RBRACE ID ASSIGN param_ids_list LSBRACE
vdecl_list stmt_list RSBRACE
     { { datatype = $4;
       fname = $6;
   formals = List.combine $2 $8;
       locals = List.rev $10;
       body = List.rev $11 } }

param_types_list:
```

```
    /* nothing */ { [] }
  | param_types_list_r   { List.rev $1 }

param_types_list_r:
    datatype                    { [($1)] }
  | param_types_list_r datatype { ($2) :: $1 }

param_ids_list:
    /* nothing */ { [] }
  | param_ids_list_r   { List.rev $1 }

param_ids_list_r:
    ID                   { [($1)] }
  | param_ids_list_r ID { ($2) :: $1 }

typ:
  | INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | FLOAT { Float }
  | CHAR { Char }

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
  datatype ID SEMI { ($1, $2) }

obj_type:
  | STRUCT ID { Objecttype($2) }

singular_type:
  | typ {$1}
  | obj_type {$1}

array_type:
  typ LSBRACE brackets RSBRACE { Arraytype($1, $3) }
```

```
datatype:
  | singular_type { Datatype($1) }
  | array_type { $1 }

brackets:
      |     /* nothing */                    { 1 }
      |     brackets RSBRACE LSBRACE { $1 + 1 }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
  | STRING_LITERAL   { StringLit(unescape $1) }
  | LITERAL          { Literal($1) }
  | FLOAT_LITERAL    { FloatLit($1) }
  | CHAR_LITERAL         { CharLit($1) }
  | BOOLEAN_LITERAL  { BoolLit($1) }
  | ID               { Id($1) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
```

```
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | expr ASSIGN expr { Assign($1, $3) }
  | ID DOT ID { StructAccess($1,$3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | NEW typ bracket_args RSBRACE  { ArrayCreate(Datatype($2), List.rev $3) }
  | NEW ID { StructCreate($2) } /*stuct_name*/
  | expr bracket_args RSBRACE    { ArrayAccess($1, List.rev $2) }

bracket_args:
  |   LSBRACE expr            { [$2] }
  |   bracket_args RSBRACE LSBRACE expr { $4 :: $1 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

semant.ml

```
(* Semantic checking for the ALBS compiler *)

open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
```

```
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions, structs) =

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

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

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

  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

  (**** Checking Functions ****)

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

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

  (* Function declaration for a named function *)
  let built_in_decls =

  StringMap.add "print" {
                            datatype  =  Datatype(Void);  fname  =  "print";  formals  =
[(Datatype(Float), "x")];
                locals = []; body = [] }

  (StringMap.singleton "getchar"  {
              datatype = Datatype(Char); fname = "getchar"; formals = [];
              locals = []; body = [] }
            ) in

  let  function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                          built_in_decls functions
  in
  let function_decl s = try StringMap.find s function_decls
       with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in

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

  let check_function func =

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

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

    List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
      " in " ^ func.fname)) func.locals;

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

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

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

    (* Return the type of an expression or throw an exception *)
    let rec expr = function
          | Literal _ -> Datatype(Int)
      | FloatLit _ -> Datatype(Float)
      | CharLit _ -> Datatype(Char)
      | StringLit _ -> Datatype(Int) (* ADDED *)
      | BoolLit _ -> Datatype(Bool)
      | Id s -> type_of_identifier s

      | ArrayCreate (t, n) -> t
      | ArrayAccess (e , el) -> expr(e)

           | StructAccess (n, f) -> Datatype(Objecttype("")) (*struct_var_name,
struct_field_name*)
      | StructCreate (n) ->  Datatype(Objecttype(""))(*struct name*)

      | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in

      if compare (string_of_datatype t1) "struct" = 0 (*don't compare structs*)
      then  t1
      else if compare (string_of_datatype t2) "struct" = 0 (*don't compare structs*)
      then t2
      else (match op with

          | Add | Sub | Mult | Div when t1 = Datatype(Int) && t2 = Datatype(Int) ->
Datatype(Int)
          | Add | Sub | Mult | Div when t1 = Datatype(Int) && t2 = Datatype(Char) ->
Datatype(Int)
```

```
         | Add | Sub | Mult | Div when t1 = Datatype(Char) && t2 = Datatype(Char) ->
Datatype(Char)


               | And | Or | Equal | Neq | Less | Leq | Greater | Geq when t1 =
Datatype(Int) && t2 = Datatype(Int) -> Datatype(Bool)
             | And | Or | Equal | Neq | Less | Leq | Greater | Geq when t1 =
Datatype(Char) && t2 = Datatype(Char) -> Datatype(Bool)


        | Equal | Neq when t1 = t2 -> Datatype(Bool)
                 | And | Or when t1 = Datatype(Bool) && t2 = Datatype(Bool) ->
Datatype(Bool)


         | Add | Sub | Mult | Div when t1 = Datatype(Float) && t2 = Datatype(Float) ->
Datatype(Float)
               | Less | Leq | Greater | Geq when t1 = Datatype(Float) && t2 =
Datatype(Float) -> Datatype(Bool)



        | _ -> raise (Failure ("illegal binary operator " ^
              string_of_datatype t1 ^ " " ^ string_of_op op ^ " " ^
              string_of_datatype t2 ^ " in " ^ string_of_expr e))
        )
      | Unop(op, e) as ex -> let t = expr e in

       if compare (string_of_datatype t) "struct" = 0 (*don't compare structs*)
      then  t
      else
       (match op with
         Neg when t = Datatype(Int) -> Datatype(Int)
   | Neg when t = Datatype(Float) -> Datatype(Float)
       | Not when t = Datatype(Bool) -> Datatype(Bool)
         | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                     string_of_datatype t ^ " in " ^ string_of_expr ex)))
      | Noexpr -> Datatype(Void)
      | Assign(var, e) as ex -> let lt = expr var
                                and rt = expr e in


                                     if compare (string_of_datatype lt) "struct" != 0
(*don't compare structs*)
                                  then if compare (string_of_datatype rt) "struct" != 0
(*don't compare structs*)
```

```
                                then ignore (check_assign lt rt
                                   (Failure ("illegal assignment: types dont match
left: " ^ string_of_datatype lt ^ " right: " ^ string_of_datatype rt )));
       lt;

       (* check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
                           " = " ^ string_of_typ rt ^ " in " ^
                           string_of_expr ex)) *)
     | Call(fname, actuals) as call -> let fd = function_decl fname in
        if List.length actuals != List.length fd.formals then
          raise (Failure ("expecting " ^ string_of_int
            (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
        else

          if fname <> "print"
          then List.iter2 (fun (ft, _) e -> let et = expr e in
             ignore (check_assign ft et
                        (Failure ("illegal actual argument found of type " ^
string_of_datatype et ^ " for variable/literal " ^ string_of_expr e ^
             ", function " ^ fname ^ " expected " ^ string_of_datatype ft ^ "."))))


            fd.formals actuals;

          fd.datatype
    in

    let check_bool_expr e =
    (*ignore(print_endline (string_of_datatype ( e)) ;); *)
    (*    ignore(print_endline (string_of_datatype (Datatype(Bool)));)*)
    if (string_of_datatype (expr e)) <> "bln"
      then raise (Failure ("expected Boolean expression as type of " ^ string_of_expr
e ^
     " it was " ^ string_of_datatype (expr e)))
     else () in

    (* Verify a statement or throw an exception *)
    let rec stmt = function
      Block sl -> let rec check_block = function
```

```
          [Return _ as s] -> stmt s
        | Return _ :: _ -> raise (Failure "nothing may follow a return")
        | Block sl :: ss -> check_block (sl @ ss)
        | s :: ss -> stmt s ; check_block ss
        | [] -> ()
       in check_block sl
     | Expr e -> ignore (expr e)
     | Return e -> let t = expr e in if t = func.datatype then () else
        raise (Failure ("return gives " ^ string_of_datatype t ^ " expected " ^
                                string_of_datatype func.datatype ^ " as type for " ^
string_of_expr e))

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

   stmt (Block func.body)

 in
 List.iter check_function functions
```

codegen.ml

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

module L = Llvm
module A = Ast

open Llvm
open Ast

module SymbolsMap = Map.Make(String)
module StringMap = Map.Make(String)
let struct_types:(string, lltype) Hashtbl.t = Hashtbl.create 10
let struct_datatypes:(string, string) Hashtbl.t = Hashtbl.create 10
let struct_field_indexes:(string, int) Hashtbl.t = Hashtbl.create 50
let struct_field_datatypes:(string, datatype) Hashtbl.t = Hashtbl.create 50

let translate (globals, functions, structs) =
     let context = L.global_context () in
     let the_module = L.create_module context "ALBS"
     and i32_t  = L.i32_type  context
```

```
      and f_t   = L.double_type   context
      and i8_t   = L.i8_type   context
      and i1_t   = L.i1_type   context
      and void_t = L.void_type context in


      let temp_ltype_of_typ (datatype:A.datatype) = match datatype with
      Datatype(A.Int) -> i32_t
      | Datatype(A.Bool) -> i1_t
      | Datatype(A.Float) -> f_t
      | Datatype(A.Char) -> i8_t
      | Datatype(A.Void) -> void_t
      | _ -> void_t in

      let find_struct name =
      try Hashtbl.find struct_types name
with | Not_found ->  raise (Failure ("Struct not found")) in


let rec get_ptr_type datatype = match datatype with
| Arraytype(t, 0) -> temp_ltype_of_typ (Datatype(t))
| Arraytype(t, 1) -> L.pointer_type (temp_ltype_of_typ (Datatype(t)))
| Arraytype(t, i) -> L.pointer_type (get_ptr_type (Arraytype(t, (i-1))))
|   _ -> void_t in

let ltype_of_typ = function
Datatype(A.Int) -> i32_t
| Datatype(A.Bool) -> i1_t
| Datatype(A.Float) -> f_t
| Datatype(A.Char) -> i8_t
| Datatype(A.Void) -> void_t
| Arraytype(t, i) -> get_ptr_type (Arraytype(t, (i)))
| Datatype(A.Objecttype(struct_name)) -> L.pointer_type(find_struct struct_name)
(*gives llvm for this struct type*)

in



(* Declare each global variable; remember its value in a map *)
let global_vars =
let global_var m (t, n) =
      let init = L.const_int (ltype_of_typ t) 0
in StringMap.add n (L.define_global n init the_module) m in
List.fold_left global_var StringMap.empty globals in

(* Declare printf(), which the print built-in function will call *)
let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module in




(* Define each struct stub (arguments) so we can create it *)
let struct_decl_stub sdecl =

      let struct_t = L.named_struct_type context sdecl.A.sname in (*make llvm for this
struct type*)
```

```
      print_endline ";struct_decl_stub called";
      Hashtbl.add struct_types sdecl.sname struct_t  (* add to map name vs
llvm_stuct_type *)
in


(* Add var_types for each struct so we can create it *)
let struct_decl sdecl =

      let struct_t = Hashtbl.find struct_types sdecl.sname in (*get llvm struct_t code
for it*)

      let type_list = List.map (fun (t,_) -> ltype_of_typ t) sdecl.A.svar_decl_list in
(*map the datatypes*)
      let name_list = List.map (fun (_,n) -> n) sdecl.A.svar_decl_list in (*map the
names*)

      let type_list = i32_t :: type_list in
      let name_list = ".k" :: name_list in

      let type_array = (Array.of_list type_list) in

      List.iteri (fun i f ->
            let n = sdecl.sname ^ "." ^ f in

            Hashtbl.add struct_field_indexes n i; (*add to name struct_field_indices*)
      ) name_list;


      L.struct_set_body struct_t type_array true

in

(* Add var_types for each struct so we can create it *)
let struct_decl_field_datatypes sdecl =


      let type_list = List.map (fun (t,_) -> t) sdecl.A.svar_decl_list in (*map the
datatypes*)
      let name_list = List.map (fun (_,n) -> n) sdecl.A.svar_decl_list in (*map the
names*)

      (* Add key all fields in the struct *)
      let type_list = (Datatype(A.Int)):: type_list in
      let name_list = ".key" :: name_list in

(*
      ignore(Hashtbl.add struct_field_datatypes "location.a" (Datatype(A.Int)));
      ignore(Hashtbl.add struct_field_datatypes "location.b" (Datatype(A.Float)));
      ignore(Hashtbl.add struct_field_datatypes "location.c" (Datatype(A.Bool)));
      ignore(Hashtbl.add struct_field_datatypes "loc.d" (Datatype(A.Char)));
      ignore(Hashtbl.add struct_field_datatypes "loc.e" (Datatype(A.Float)));
      ignore(Hashtbl.add struct_field_datatypes "loc.x" (Datatype(A.Int)));
 *)
      ignore(

            List.map2 (fun f t ->
```

```
                let n = sdecl.sname ^ "." ^ f in
                Hashtbl.add struct_field_datatypes n t; (*add name, datatype*)
            ) name_list type_list;

      )


in

let _ = List.map (fun s -> struct_decl_stub s) structs in
let _ = List.map (fun s -> struct_decl s) structs in

let struct1 = structs in
let x = List.map (fun s -> struct_decl_field_datatypes s) struct1 in



(* Define each function (arguments and return type) so we can call it *)
let function_decls =
let function_decl m fdecl =
      let name = fdecl.A.fname
      and formal_types =
      Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals)
in let fdecl_datatype = fdecl.A.datatype (*return type*)
in let llvm_fdecl = (ltype_of_typ fdecl_datatype)
in let ftype = L.function_type llvm_fdecl 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.A.fname function_decls in
      let builder = L.builder_at_end context (L.entry_block the_function) in
      let float_format_str = L.build_global_stringptr "%f\n" "fmt" builder in
      let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
      let bool_format_str = L.build_global_stringptr "%s\n" "fmt" builder in
      let char_format_str = L.build_global_stringptr "%c\n" "fmt" builder in

  (* Construct the function's "locals": formal arguments and locally
  declared variables.  Allocate each on the stack, initialize their
  value, if appropriate, and remember their values in the "locals" map *)
let local_vars =

let add_formal m (t, n) p = L.set_value_name n p;
let local = L.build_alloca (ltype_of_typ t) n builder in
ignore (L.build_store p local builder);
StringMap.add n local m in

let add_local m (var_type, var_name) = (* map, datatype, name *)

let t = match var_type with
Datatype(Objecttype(n)) ->

ignore(Hashtbl.add struct_datatypes var_name n);  (* add to map name vs type *)

find_struct n
|   _ -> ltype_of_typ var_type
in
```

```
let llvm_for_allocation = L.build_alloca t var_name builder in


StringMap.add var_name llvm_for_allocation m;


in


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

List.fold_left add_local formals fdecl.A.locals in



(*name vs type*)
let symbol_vars =

let add_to_symbol_table m (t, n) =
      SymbolsMap.add n t m in

      let symbolmap = List.fold_left add_to_symbol_table SymbolsMap.empty
fdecl.A.formals in

      List.fold_left add_to_symbol_table symbolmap fdecl.A.locals in



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


(* Return the type for a variable or formal argument *)
let lookup_datatype n = try SymbolsMap.find n symbol_vars
with Not_found -> SymbolsMap.find n symbol_vars
in


(* Return the datatype for a struct *)
let lookup_struct_datatype(id, field) =

      let struct_name = Hashtbl.find struct_datatypes id in (*gets name of struct*)

      let search_term = ( struct_name ^ "." ^ field) in (*builds struct_name.field*)

      let my_datatype = Hashtbl.find struct_field_datatypes search_term in (*get
datatype*)

      my_datatype

in
```

```
let int_binops op =  (

                        match op with
                              | A.Add     -> L.build_add
                              | A.Sub     -> L.build_sub
                              | A.Mult    -> L.build_mul
                              | A.Div     -> L.build_sdiv
                              | A.Equal   -> L.build_icmp L.Icmp.Eq
                              | A.Neq     -> L.build_icmp L.Icmp.Ne
                              | A.Less    -> L.build_icmp L.Icmp.Slt
                              | A.Leq     -> L.build_icmp L.Icmp.Sle
                              | A.Greater -> L.build_icmp L.Icmp.Sgt
                              | A.Geq     -> L.build_icmp L.Icmp.Sge
                              | _ -> raise (Failure "Invalid Int Binop")
                        )
in

let float_binops op =  (

            match op with
            A.Add     -> L.build_fadd
            | A.Sub     -> L.build_fsub
            | A.Mult    -> L.build_fmul
            | A.Div     -> L.build_fdiv
            | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
            | A.Neq     -> L.build_fcmp L.Fcmp.One
            | A.Less    -> L.build_fcmp L.Fcmp.Ult
            | A.Leq     -> L.build_fcmp L.Fcmp.Ole
            | A.Greater -> L.build_fcmp L.Fcmp.Ogt
            | A.Geq     -> L.build_fcmp L.Fcmp.Oge
            | _ -> raise (Failure "Invalid")
      )
in

let bool_binops op =  (

            match op with
            | A.And     -> L.build_and
            | A.Or      -> L.build_or
            | A.Equal   -> L.build_icmp L.Icmp.Eq
            | A.Neq     -> L.build_icmp L.Icmp.Ne
            | _ -> raise (Failure "Unsupported bool binop")
      )
in

(*Array functions*)
let initialise_array arr arr_len init_val start_pos builder =
      let new_block label =
      let f = block_parent (insertion_block builder) in
      append_block (global_context ()) label f
in
let bbcurr = insertion_block builder in
let bbcond = new_block "array.cond" in
let bbbody = new_block "array.init" in
let bbdone = new_block "array.done" in
ignore (build_br bbcond builder);
position_at_end bbcond builder;
```

```
(* Counter into the length of the array *)
let counter = build_phi [const_int i32_t start_pos, bbcurr] "counter" builder in
add_incoming ((build_add counter (const_int i32_t 1) "tmp" builder), bbbody) counter;
let cmp = build_icmp Icmp.Slt counter arr_len "tmp" builder in
ignore (build_cond_br cmp bbbody bbdone builder);
position_at_end bbbody builder;

(* Assign array position to init_val *)
let arr_ptr = build_gep arr [| counter |] "tmp" builder in
ignore (build_store init_val arr_ptr builder);
ignore (build_br bbcond builder);
position_at_end bbdone builder in



let getchar_ty = (L.function_type i8_t [||]) in
let getchar_func = L.declare_function "getchar" getchar_ty the_module in

(*Array functions*)
let struct_access struct_id rhs isAssign builder = (*id field*)


      let struct_name = Hashtbl.find struct_datatypes struct_id in

      let search_term = (struct_name ^ "." ^ rhs) in

      let field_index = Hashtbl.find struct_field_indexes search_term in

      let _val = L.build_struct_gep (lookup struct_id) field_index rhs builder in

      let _val =
            if isAssign then
                  build_load _val rhs builder
            else
                  _val

      in
      _val

in


(* Construct code for an expression; return its value *)
let rec expr builder = function
| A.FloatLit f  -> L.const_float f_t f
| A.StringLit s -> L.build_global_stringptr s "" builder
| A.CharLit c   -> L.const_int i8_t (Char.code c)
| A.Literal i   -> L.const_int i32_t i
| A.BoolLit b   -> L.const_int i1_t (if b then 1 else 0)
| A.Noexpr      -> L.const_int i32_t 0



| A.StructAccess(lhs,rhs) -> (

      struct_access lhs rhs true builder
)
```

```
| A.StructCreate(struct_name)  ->
(
      raise (Failure ("Structs still in progress 2"))
)

(*integer literals*)
| A.Call ("getchar", el) ->

      let f = getchar_func in
      let params = List.map (expr builder) el in
      print_endline ";hi"; build_call f (Array.of_list params) "tmp" builder


| A.Call ("print", [e]) ->

      (match List.hd [e] with


      | A.StringLit s ->
      let head = expr builder (List.hd [e]) in
      let llvm_val = L.build_in_bounds_gep head [| L.const_int i32_t 0 |]
"string_printf" builder in

      print_endline ";a.string print called";
      L.build_call printf_func [| llvm_val |] "string_printf" builder

      | A.FloatLit f -> print_endline ";a.floatlit print called"; L.build_call
printf_func [| float_format_str ; (expr builder e) |] "float_printf" builder


      | A.Id my_id ->
      (
            let my_typ = lookup_datatype my_id in
            ignore(print_endline("; my_typ: " ^ string_of_datatype my_typ));
            (match my_typ with

                  | Datatype(A.Int) | Datatype(A.Bool) ->
                  print_endline ";a.int or a.bool print called";L.build_call
printf_func [| int_format_str ; (expr builder e) |] "int_printf" builder

                  | Datatype(A.Float) ->
                  print_endline ";a.float print called";L.build_call printf_func [|
float_format_str ; (expr builder e) |] "float_printf" builder

                  | Datatype(A.Char) ->
                  print_endline ";a.char print called";L.build_call printf_func [|
char_format_str ; (expr builder e) |] "char_printf" builder

                  | _ ->
                  print_endline ";a.string print called";L.build_call printf_func [|
int_format_str ; (expr builder e) |] "string_printf" builder
            )
      )

      | A.StructAccess(var,field) ->
      (
            let my_datatype = lookup_struct_datatype(var,field) in (*get datatype*)
```

```
            match my_datatype with

            | Datatype(A.Bool) | Datatype(A.Int)  -> print_endline ";struct print
int/bool called"; L.build_call printf_func [| int_format_str ; (expr builder e) |]
"abcd" builder

            | Datatype(A.Char) -> print_endline ";struct print char called";
L.build_call printf_func [| char_format_str ; (expr builder e) |] "abcd" builder

            | Datatype(A.Float) -> print_endline ";struct print float called";
L.build_call printf_func [| float_format_str ; (expr builder e) |] "abcd" builder

            | _ ->      print_endline "struct print!!!!!!!";print_endline
(string_of_datatype my_datatype);print_endline "struct print!!!!!!!"; L.build_call
printf_func [| int_format_str ; (expr builder e) |] "abcd" builder

      )
      | A.ArrayAccess(e2,i2) -> (match e2 with
            | A.FloatLit f -> print_endline ";ArrayAccess float lit print called";
L.build_call printf_func [| int_format_str ; (expr builder e) |] "abcd" builder
            | A.Id my_id -> ignore(print_endline(";my_id: " ^ my_id));
            (
                  let my_typ = lookup_datatype my_id in
                  ignore(print_endline(";typ: "^ (string_of_datatype my_typ)));
                  (match (string_of_datatype my_typ) with
                        | "int" ->
                        print_endline ";mytype int_ print called";L.build_call
printf_func [| int_format_str ; (expr builder e) |] "int_printf" builder
                        | "flt" ->
                        print_endline ";mytype float_ print called";L.build_call
printf_func [| float_format_str ; (expr builder e) |] "float_printf" builder
                        | "chr" ->
                        print_endline ";mytype char_ print called"; L.build_call
printf_func [| char_format_str ; (expr builder e) |] "char_printf" builder
                        | _ ->
                        print_endline (";a._ in mytype match print called" ^
(string_of_datatype my_typ));L.build_call printf_func [| int_format_str ; (expr
builder e) |] "string_printf" builder
                  )
            )
            | _ -> print_endline (";ArrayAccess print called " ^ (string_of_expr e));
L.build_call printf_func [| int_format_str ; (expr builder e) |] "abcd" builder
      )
| _ -> print_endline ";_ print called"; L.build_call printf_func [| int_format_str ;
(expr builder e) |] "abcd" builder

)




(*Arrays*)
|  A.ArrayCreate(t, el)      ->
(
      if(List.length el > 1) (*list of dimension sizes*)
      then raise (Failure ("Only 1D arrays are allowed")) (*should throw an exception,
not return*)
```

```
else
      match t with
      | Datatype(Char) ->

            let e = List.hd el in
            let size = (expr builder e) in
            let t = ltype_of_typ t in
            let arr = build_array_malloc t size "tmp" builder in
            let arr = build_pointercast arr (pointer_type t) "tmp" builder in
            print_endline "; char[] called"; arr

      |   _ ->

      let e = List.hd el in
      let t = ltype_of_typ t in

      let size = (expr builder e) in
      let size_t = build_intcast (size_of t) i32_t "tmp" builder in
      let size = build_mul size_t size "tmp" builder in
      let size_real = build_add size (const_int i32_t 1) "arr_size" builder in

      let arr = build_array_malloc t size_real "tmp" builder in
      let arr = build_pointercast arr (pointer_type t) "tmp" builder in

      let arr_len_ptr = build_pointercast arr (pointer_type i32_t) "tmp" builder in

      (* Store length at this position *)
      ignore(build_store size_real arr_len_ptr builder);
      initialise_array arr_len_ptr size_real (const_int i32_t 0) 0 builder;
      print_endline "; _[] called"; arr
)


(*ID and index*)
| A.ArrayAccess(e, el)   ->
(
      let index = expr builder (List.hd el) in
      let index = (match e with
            | A.FloatLit f ->   build_add
            index (
                  const_float f_t
            1.0)
            "tmp" builder
            | _ ->   build_add index (const_int i32_t 1) "tmp" builder
      )

      in
      let arr = expr builder e in
      let _val = build_gep arr [| index |] "tmp" builder in

      if false
      then _val
      else build_load _val "tmp" builder
)

| A.Call (f, act) ->
      let (fdef, fdecl) = StringMap.find f function_decls in
```

```
      let actuals = List.rev (List.map (expr builder) (List.rev act)) in
      let result = (match fdecl.A.datatype with
            Datatype(Void) -> ""
            | _ -> f ^ "_result") in
      L.build_call fdef (Array.of_list actuals) result builder

| A.Id s -> L.build_load (lookup s) s builder

| A.Binop (e1, op, e2) ->
      let e1' = expr builder e1
      and e2' = expr builder e2 in

      ignore(print_endline ";in binop");
      (match e1 with

            | A.BoolLit b | A.Bool b -> (bool_binops op) e1' e2' "tmp" builder

            | A.FloatLit f -> (float_binops op) e1' e2' "tmp" builder

            | A.CharLit c -> (int_binops op) e1' e2' "tmp" builder

            | A.Literal i -> (int_binops op) e1' e2' "tmp" builder


            | A.Id my_id -> (

                  let my_typ = lookup_datatype my_id in
                  (
                        match my_typ with
                        | Datatype(A.Int) -> (
                                    ignore(print_endline ";matched int ID here");

                                     match op with
                                    | A.Add
                                    | A.Sub
                                    | A.Mult
                                    | A.Div     ->
                                    (      match e2 with
                                          | A.CharLit c -> (
                                                ignore(print_endline ";matched op");
                                                ignore(print_endline ";matched char
here");

                                                let temp_int_val = Char.code c in


                                                let temp = L.const_int i32_t
temp_int_val in
                                                let e2' = (temp) in
                                                (int_binops op) e1' e2' "tmp" builder

                                          )
                                          | _ -> (int_binops op) e1' e2' "tmp" builder

                                    )
                                    | A.Equal
                                    | A.Neq
                                    | A.Less
```

```
                                    | A.Leq
                                    | A.Greater
                                    | A.Geq     -> ignore(print_endline ";normal int
op");(int_binops op) e1' e2' "tmp" builder

                                    | _ -> raise (Failure "Invalid Int Binop")
                        )

                        |  Datatype(A.Char)  -> (int_binops op) e1' e2' "tmp" builder

                        | Datatype(A.Bool) -> (bool_binops op) e1' e2' "tmp" builder

                        | Datatype(A.Float) ->  (float_binops op) e1' e2' "tmp"
builder

                        | _ -> raise (Failure "Invalid Types of ID binop")

                  )
            )
            | A.StructAccess(id,field) ->
            (
                  let my_datatype = lookup_struct_datatype(id,field) in (*get
datatype*)

                  match my_datatype with

                  | Datatype(A.Bool) -> (bool_binops op) e1' e2' "tmp" builder

                  | Datatype(A.Int) | Datatype(A.Char) -> (int_binops op) e1' e2'
"tmp" builder

                  | Datatype(A.Float) -> (float_binops op) e1' e2' "tmp" builder

                  | _ ->      raise (Failure "Invalid Types of Struct binop")

            )

            | _ -> raise (Failure "Invalid Types")

      )


| A.Unop(op, e) ->
let e' = expr builder e in
(match op with
      | A.Neg     ->

      (match e with

            | A.FloatLit f -> L.build_fneg

            | A.Literal i -> L.build_neg

            | A.Id my_id ->
            (
                  let my_typ = lookup_datatype my_id in
                  (match my_typ with
                        | Datatype(A.Int) -> L.build_neg
```

```
                        | Datatype(A.Float) -> L.build_fneg
                        | _ -> raise (Failure "Invalid Unop")
                  )
            )
            | A.StructAccess(id,field) ->
            (
                  let my_datatype = lookup_struct_datatype(id,field) in (*get
datatype*)

                  match my_datatype with

                  | Datatype(A.Int) | Datatype(A.Char) -> L.build_neg

                  | Datatype(A.Float) -> L.build_fneg

                  | _ ->      raise (Failure "Invalid Types of Struct binop")

            )
            | _ ->  raise (Failure "Invalid Unop")

      )

| A.Not -> L.build_not) e' "tmp" builder

| A.Assign (lhs, rhs) ->
(
      let lhs =
      (
            print_endline ";assign";
            match (lhs) with

            | A.Id id -> lookup id


            (*ID and index*)
            | A.ArrayAccess(e, el)   ->
            (
                  let my_val =
                  match (e) with
                  | A.Id myaid ->
                  lookup myaid
                  | _ -> raise (Failure "Invalid2") in


                  let index = expr builder (List.hd el) in

                  let index = build_add index (const_int i32_t 1) "tmp" builder in
                  (
                        match (e) with
                        | A.Id myaid ->
                        (
                              let name = List.hd [e] in
                              let arr = L.build_load my_val myaid builder in
                              let _val = build_gep arr [| index |] "tmp" builder in
                              _val
                        )
                        | _ -> raise (Failure "Invalid3")
                  )
```

```
            )

            | A.StructAccess(lhs,rhs) -> (

                  struct_access lhs rhs false builder
            )

      )

in

let rhs = match rhs with
| A.Id id -> print_endline ";lookup RHS id"; expr builder rhs

| _ -> expr builder rhs

in
ignore (L.build_store rhs lhs builder);
rhs
)

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

(* Build the code for the given statement; return the builder for the statement's
successor *)
let rec stmt builder = function
A.Block sl -> List.fold_left stmt builder sl
| A.Expr e -> ignore (expr builder e); builder
| A.Return e -> ignore (match fdecl.A.datatype with
      Datatype(Void) -> L.build_ret_void builder
      | _ -> L.build_ret (expr builder e) builder); builder
| A.If (predicate, then_stmt, else_stmt) ->
let bool_val = expr builder predicate in
let merge_bb = L.append_block context "merge" the_function in

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

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

ignore (L.build_cond_br bool_val then_bb else_bb builder);
L.builder_at_end context merge_bb

| A.While (predicate, body) ->
let pred_bb = L.append_block context "while" the_function in
ignore (L.build_br pred_bb builder);

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

```
(L.build_br pred_bb);

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

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

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

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

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

List.iter build_function_body functions;
the_module
```