Venture

Start Yours

James Sands | js4597
Naina Sahrawat | ns3001
Zach Adler | zpa2001
Ben Carlin | bc2620

Professor Stephen Edwards
COMS 4115: Programming Languages and Translators
Spring 2017

Contents

Introduction

Motivation

"Adventure" was one of the original computer games. The player types instructions to their avatar in the game, moves through the world, and interacts with objects and characters. Because it is so modular, every version of the game is different. This variety of play experience is what makes text based adventure games fun to this day. With Venture we aimed to create a programming language that can be used to easily and intuitively create text based adventure games.

Language

Venture is a simple C-like imperative language. The language compiler was developed in OCaml and compiles into LLVM. A standard library written in C is linked to Venture code at compile time to generate a game executable.

In Venture, we aimed to provide a game designer tools to build such a fantasy world, as well as a standard library to handle tricky details of I/O. Functionality is provided to accept user input commands, compare them to pre-existing commands within the code of the game, and print appropriate responses. Control flow statements are provided to guide the user through the game based on their choices.

Tutorial

Data Types and Syntax

Venture supports the primitive data types 'int', 'string', 'bool', 'char', and 'void'. Venture uses C-like syntax, which curly braces indicating scope, parentheses indicating precedence, expressions, and argument passing. Each statement is ended with a semicolon.

Comments

Comments in Venture are indicated at the beginning and the end with "##". Nested comments are not supported.

Control Statements

Venture supports **if else** statements and **while** statements. The syntax for these statement is as follows:

```
if (<bool expr>){
      <venture code>
} else {
      < venture code>
}

while (<bool expr>){
      <venture code>
}
```

User Defined Functions and Program Structure

Aside from the Venture standard library, a user can create a function using the syntax illustrated in the example code provided below:

```
<return type> <fname>(<arg1 type> <arg1 name>, …) {
      ##Venture code here##
      return <expr of return type>;
}
```

A Venture program has the following structure:

```
<Global declarations>
<User defined functions>
int main(){
      <main code>
      return <expr int>;
}
```

Only code within the main function will be executed.

Variable Declarations

Global variables must be declared at the beginning of a Venture program, before function declarations. This is true within each scope (eg. in a function, the variables must be declared at the beginning of the body). Variables must be declared and defined separately.

Standard Library

Venture code supports **print(arg)** of any primitive data type, **input()** to accept user input, and **scompare(str1, str2)**.

Below is a venture program illustrating the basics of the language as explained above.

```
void foo(int a) {
        int i;
        string foo;

        foo = "hello";
        i = 0;
        while( i < a){
                print(foo);
                i = i + 1;
        }
}

int bar(bool b, int a, int c) {
        if (b) {
                return a + c;
        } else {
                return 0;
        }
}

int main()
{
        bool d;
        int e;

        d = true;
        e = 5;

        if(bar(true, 10, 4) > e){
                foo(1);
        }
        ##should print "hello"##
        return 0;
```

```
}
```

When executed, the code above will output:

```
hello
```

Compiling and Running

Run
`$ make`
to compile the compiler code, then simply run

`./venture.sh <filename>.vtr`

to compile the code and run the executable.

Language Reference Manual

Section 1. Lexical Elements

1.1 Identifiers

An identifier in Venture is a sequence of letters, uppercase A-Z, and lowercase a-z, a integer 0-9, and/or the underscore. The identifiers are case sensitive and are used to create the names for variable and functions. For example, the identifiers, "ifTiffanyExists" and "IfTiffanyExists" are two different identifiers are the same. Identifiers always begin with a letter, uppercase or lowercase. Identifiers cannot equal any keyword, as defined in the Keyword section.

1.2 Keywords

These keywords are reserved and are to be used by the language. These keywords can not be used as regular identifiers and are case sensitive:

- int, string, char, bool, void
- if, else, while, return
- true, false

1.3 Literals

Literals are immutable and have a primitive data type corresponding to their value. The data type that a literal can be of is a string, and int or a boolean. Trying to assign a value to an incorrect data type will raise an error and you can not cast a value as a different type.

Examples: `"hello" , 69, true`

1.3.1 String Literals

A String literal is a sequence of characters that are enclosed by single quotes and/or escape characters enclosed in double quotes.

Examples: `"\n", "\t" "string"`

1.3.2 Integer Literals

An integer literals are a sequence of one or more integers, 0-9. Floating point numbers won't be recognized by the language and will raise an error.

Examples: `48, 17, 2748`

1.3.3 Boolean Literals

A boolean literal is a value of true or false. Assigning any other value of any other data type to a boolean variable will raise an error.

Examples: `true, false`

1.4 Operators

Operators are tokens that are used to denote an operation to be done on an element or combination of elements. Simple mathematical operations are common operators, such as addition, subtraction, multiplication and division. More on operators are explained in Section 3.

1.5 Delimiters

Delimiters help the compiler interpret tokens and separate tokens from each other.

1.5.1 Parentheses

Parentheses will be used to enclose arguments for a function, which is exemplified in section 5, Functions. Parentheses will also be used to make the compiler understand to evaluate a certain order of a program first.

1.5.2 Commas

Commas will be used to separate arguments of a function.

1.5.3 Brackets

Brackets will be used to declare arrays, used to access indices of an array and assignment of indices of an array.

1.5.4 Semicolon

The semicolon will designate the end of a statement.

1.5.5 Curly Braces

Curly braces will be used designate the statements that will be evaluated in a function. Examples are provided in the function section.

1.5.6 Whitespace

Whitespace will be used to separate tokens for the compiler and has no special meaning. Examples of whitespace is a space, tab, newline and carriage return with the newline.

Section 2: Data Types

2.1 Primitive Data Types

2.1.1 int

This token int is case sensitive and will be used to store 32-bit signed integer, from -2,147,483,648 to 2,147,483,647.

2.1.2 boolean

This token is case sensitive and will be used to store values of `true` and `false`.

2.1.3 char

This token will be used to store a single character.

2.1.4 string

This token will be used to store sequences of characters and will be where text values will be saved.

2.1.5 void

This token will only be used for return types in function definitions. An error will be raised if a developer tries to assign void to an identifier.

I.e.: `void iD = "tiffany";`

Section 3: Operators & Expressions

3.1 Operators

Operator	Description
* / %	multiplication, integer division, modulo
+ -	Addition, subtraction
< <= > =>	less than, less than or equal to, greater than, greater than or equal to
== !=	equal, not equal
&&	Logical AND
||	Logical OR
!	Logical NOT
=	assignment

3.2 Operator Precedence

Operator precedence is according to the order in which they appear in the above table; the attribute access has highest precedence and the assignment has lowest.

3.3 Expressions

Expressions consist of one or more operands and zero or more operators.

3.4 Expression Precedence

Expressions are evaluated in a way such that the innermost expressions are evaluated first, as indicated by parentheses. Each expression is evaluated left to right, accounting for operator precedence.

Section 4: Statements

4.1 if-else Statements

The if-else statement is used to execute a block of code if a specified condition is met, and if the "if" condition is not met, another block of code after the "else" is executed. The general form of an if-else statement is as follows:

```
if (condition) {
      expression;
```

```
} else {
      expression1;
}
```

In the case where there is no alternative block of code you'd like to execute with the "else", that block of code can be left empty, but if-else statements require both an "if" and an "else":

```
if (condition) {
      expression;
} else {
}
```

4.2 while Statements

The while statement is used to execute a block of code repeatedly (in a loop) until the specified condition is no longer met. The condition is checked immediately before entering each iteration of the loop. The general structure of a while loop is as follows:

```
while (condition) {
      expression;
}
```

Section 5: Functions

5.1 Function Definitions

Function definitions consist of a return type, a function identifier, a set of parameters and their types, and then a block of code to execute when that function is called with the specified parameters. An example of an addition function definition is as follows:

```
int sum( int a , int b ){
return a + b; }
```

5.2 Calling Functions

A function can be called by its identifier followed by its parameters in parentheses, such as in the following example:

```
sum(1, 2);
```

Section 6: Program Structure and Scope

6.1 Program Structure

Venture programs will be one source file. Venture programs consist of:

1. Global Variable Declarations
2. Function Declarations
3. Main

6.2 Libraries

The standard library `stdlib` is automatically imported through linking the files during compilation at the beginning of the Venture program.

6.3 Scope

Global declarations are made outside of the block of a function definition or if statement or while statement. Globals can be referenced in the program following their declaration. Declarations made within blocks of an if statement, a while statement, or a function definition are only available for reference within that block. Brackets can be used for nested scoping. Scoping in Venture is static.

```
string hello;
int main(){
      int value1;
      value1 = 1
      {
            int value2;
            value2 = 2;
      }
            ## value2 is out of scope here ##
}
## both value1 and value2 are out of scope here##
## hello can be referenced throughout the program ##
```

Section 7: Built In Functions

The `print` function can be used to print integers, booleans, or strings. After the argument is printed, a newline character is printed.

```
print(arg);
```

The `scompare` function takes two strings as arguments, casts the strings to lowercase, compares them, and returns 0 if the two strings are the same, and an integer other than 0 if they are not. This function is meant for use with strings that result from user input.

```
scompare(string1, string2);
```

The `input` function accepts text input from i/o with each '\n'. This function returns a string.

```
input();
```

Section 8: Context Free Grammar

expr → literal
id
id actuals
expr Add expr
expr Sub expr
expr Mult expr
expr Div expr
expr And expr
expr Or expr
expr Equal exor
expr Neq expr
expr Less expr
expr Greater expr
expr Geq expr

type → int
string
boolean
void

vdcel → type id expr
type id assign
type id

Project Plan

Planning, Specification, Development, and Testing

The Venture team met on a roughly weekly basis. Each week individual members of the group met with our TA, Julie, to discuss code and gauge our progress. We used Facebook Messenger as our primary means of communication, Google Drive to share documents such as our shared "To Do List", and Github as the VCS for out code and lower level notes.

We initially created a plan with project benchmarks and when we wanted them to be completed, but in the end scrapped that to follow roughly along with the lectures, creating code in line with the lectures provided any given week. At all times, each team member had a concrete coding task. We all work at different paces, so when one team member completed a task he or she would be given (or pick) a new one immediately rather than waiting for other team members to complete theirs. For a task which took longer than expected or had persistent issues, we would meet as a group to discuss it, then bring our issue to Julie if necessary.

We found that once the "hello world" was created, tasks became more modular and we were able to work more efficiently and effectively on separate code functionality.

Each team member was responsible for testing their code and ensuring its functionality before pushing to github. As a team we agreed to make small commits more often to allow for easy rollback if necessary. Thankfully, due to our policy of testing before pushing, this was never necessary.

Style Guide

The following outlines our style guide for OCaml, Github, and C

Ocaml

- snake_case
- 2 space indentation
- 100 character line length
- Commenting code blocks
- No pipe for first case in pattern matching
- "in" follows "let" at the end of same line, or one tab to the right on the following line. Whichever is more fitting.
- Newlines to divide code into conceptual blocks and function definitions

Git

- Commit frequently to allow for rollback if necessary
- Use illustrative commit messages

C

- 8 character tab indentation
- Variable and function names lowercase with no underscores

Project Timeline

Date	**Benchmark**
February 6	Git repo created. Decide on project ideas.
February 8	Submit Project Proposal
February 22	Submit Language Reference Manual
March 18	Download VirtualBox and ensure Ubuntu environments working
March 22-28	Build Venture basic compiler for Hello World
April 2	Initial Semant Checker
April 6	Binary Operators Working for integers
April 10-20	Control Statements, Primitive Types, Unary Operators, User Defined Functions, String Type Variables created
April 21	Beginning of C standard library
April 27	Working Test Shell Committed, Success Case test suite started
May 4	C standard library completed, Cleaned up compiler code to be legible and nicely commented
May 5	Initial demo game
May 6	Full demo game
May 6	Working full semantic checker
May 7	Fail test suite added
May 7-9	Final report
May 10	Final Presentation

Roles and Responsibilities

Our group had the assigned roles of James as System Architect, Naina as Project Manager, Zach as Language Guru, and Ben as Tester. We stuck to our roles fairly strictly for the first few weeks. Our group's role responsibilities dissolved slightly after submitting the Hello World; the emphasis of our roles was still there, but the work divisions became more fluid. We tended to work together on specific problems more often as the semester pushed on. The final deliverables of each member of the team are reflected in the Git Commit logs in the Project Plan section.

Software Development Environment

- **Languages Used:** Ocaml, C, Bash
- **Programming editor:** vim
- **VCS:** Github, Git
- **Documentation:** Google Docs

Project Commit Log

Group members by Github email accounts:

- zpa2001@columbia.edu - Zachary Adler
- nsahra14@gmail.com - Naina Sahrawat
- js4597@columbia.edu - James Sands
- bc2620@columbia.edu - Ben Carlin

```
commit a9cafd4ff965d4b3643de599569acd66bdee4d17
Author: nsahra14 <nsahra14@gmail.com>
Date:   Tue May 9 20:28:41 2017 -0400

    venture.ml clean up

commit 0839dc19527254e4c8619a562100a74eac3d1916
Author: nsahra14 <nsahra14@gmail.com>
Date:   Tue May 9 20:23:24 2017 -0400

    moved demo.vtr to main directory

commit 847371d249705bedaf7ce71924d2ea85d59f6bdf
Author: nsahra14 <nsahra14@gmail.com>
Date:   Tue May 9 20:22:55 2017 -0400

    delted temporary test files and notes

commit 4a623fbe93d0d128228dd91dbc80a01c8b619578
Author: nsahra14 <nsahra14@gmail.com>
Date:   Tue May 9 20:12:01 2017 -0400

    cleanup of testall script
```

```
commit 6c7c3c80ed65a054e01b4938026249ea76b18149
Author: nsahra14 <nsahra14@gmail.com>
Date:   Tue May 9 20:07:04 2017 -0400

    cleanup of Makefile

commit e59750380a35d89065e100f3066b1f91b05a6c10
Author: nsahra14 <nsahra14@gmail.com>
Date:   Tue May 9 19:59:18 2017 -0400

    clean up to commented out scanner code

commit 8bc920efd580db66a250486cb4420dc1e300f6ef
Author: nsahra14 <nsahra14@gmail.com>
Date:   Tue May 9 18:21:24 2017 -0400

    removed placeholder intcompare fn

commit d5ae088b32f1b03bd337f7f976fbcf9d7d8f572c
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 11:17:29 2017 -0400

    globals

commit dd8f135447318bc22de6fe9641b908d453df5f52
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 11:07:53 2017 -0400

    returns

commit 20f56a081be889b979e34dadcf33acfab9187c80
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 11:03:46 2017 -0400

    fail: while

commit c3677a07b6114e731440c745f38a8b372a814e95
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 10:52:16 2017 -0400

    func fails

commit 4648def918aa8e56bd2166b5079c8f0b2a251a1a
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 10:46:22 2017 -0400

    no main
```

```
commit 706b6305eaf418b337a3f1a199f7071ccc8ae079
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 10:42:38 2017 -0400

    expr fails and err msgs

commit b83061bad89ed46a55655465dcf9fa81d50c2408
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 10:36:58 2017 -0400

    dead code tests

commit 4f1c60f262c065f358b0df2af64bcba1267bbcde
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 10:28:17 2017 -0400

    error messages of assigns

commit 915c532d50075cf8fe8ee0053723a702b7a823a7
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sun May 7 10:21:12 2017 -0400

    fail suite initiation: assign failures

commit e9b570d03ae12c5f60673b01158ad09b551a3124
Author: Zachary Adler <zpa2001@columbia.edu>
Date:   Sat May 6 23:12:38 2017 -0400

    adding new ast.ml, ast.mli is deleted, semant and venture all updated

commit 9032a56939f91d71f1a8082d7e60f3097d9dc404
Author: jssands <js4597@columbia.edu>
Date:   Sat May 6 20:05:34 2017 -0400

    updated todo

commit 59a6c7184d4e576c4e578b1444a89d5606676e6d
Author: jssands <js4597@columbia.edu>
Date:   Sat May 6 20:02:35 2017 -0400

    general venture.sh run file which takes command line arg for file

commit a222f406c9c0ebcb9b3eafe73b02cee05a6d1c97
Author: jssands <js4597@columbia.edu>
Date:   Sat May 6 19:26:34 2017 -0400

    changed help options in demo
```

```
commit 7f16f6e45da778dbf03c4f0d67d6f30f0a7f8ea4
Author: jssands <js4597@columbia.edu>
Date:   Fri May 5 13:28:31 2017 -0400

    Small change in demo game.

commit fc4ee5022cff5e5479f6f657f136107b9103b354
Author: jssands <js4597@columbia.edu>
Date:   Fri May 5 13:16:12 2017 -0400

    working demo, small change

commit 8adcca52f9f352b41ec04ed5743e2352a872ced2
Author: jssands <js4597@columbia.edu>
Date:   Fri May 5 13:09:06 2017 -0400

    Full demo completed! Debugged and ready to go.

commit aebbb4474aec8337fe0011f5dc04756e3e0978f2
Author: jssands <js4597@columbia.edu>
Date:   Thu May 4 21:58:23 2017 -0400

    codegen unused var warnings fixed. testall updated

commit 1880669cfb73e651359cf7dbbefbb9580c56c5fc
Author: jssands <js4597@columbia.edu>
Date:   Thu May 4 21:39:50 2017 -0400

    small update to stdlib so scompare works with fgets input

commit bf7a4f98410955d9c04f3a141618eacb7cbd1743
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 19:00:16 2017 -0400

    test5 tests for strcmp and lowercase casting

commit 2e28d3098abd55b4c51a7467fc9b8827f4206cf0
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 18:59:39 2017 -0400

    strcmp casts to lowercase before comparing

commit 6efdf6fc25c8f2a8b378d41b972bdb9f1ec6ee88
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 18:26:55 2017 -0400

    cleaned up codegen of comments
```

```
commit 460c3232c749542c90bf5cac54a5d8f85c672ff4
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 16:48:13 2017 -0400

    linkrun.sh links to stdlib and runs all test_files, test_files updated

commit 1d9acbf56067b2ec844e5fc99c34b19fe4cc9f35
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 16:42:02 2017 -0400

    stdlib.c now contains all fns, input.c removed

commit 67f9ceb5def4ec3a4c50ce9e2cbfe5a9c687e720
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 16:14:38 2017 -0400

    updated test_files

commit 3509041f1d56c8125bf6b4743d2f3ebd28ae9696
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 16:13:37 2017 -0400

    removed scompare.c, combined with input.c fns

commit a3dbf4d50e5a723f25f7bfd25cb2fb8df3159cb3
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 15:59:26 2017 -0400

    scompare implemented and working with fixes to codegen

commit 92f5721e2e0e9c25ad73c214296f7e7f3ed4b81f
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 15:31:25 2017 -0400

    test file 2 corrected for intcompare

commit 10828c7ed105da70e6647305b80c00b18c8c9771
Merge: 62ad5c5 8e7834c
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 15:04:09 2017 -0400

    Merge branch 'master' of https://github.com/jssands/PLTgame

commit 62ad5c5ebbad7e0e25975b5eea00613947b37615
Author: nsahra14 <nsahra14@gmail.com>
Date:   Thu May 4 15:03:30 2017 -0400

    working intcompare fn as fix for scompare
```

```
commit 5e3e525d57fe74eea5678d7aab767d2b399399ac
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Apr 28 18:01:55 2017 -0400

    additional test files for linkrun script

commit c9a93ef474840c57d841682fd5dfd04b937ee488
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Apr 28 18:01:28 2017 -0400

    removed dummy lib/printbig c libraries

commit 4610dfc811bb8faa43fa21b5908c4296ae7b6391
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Apr 28 17:54:32 2017 -0400

    updated with correct linking to external c libs, input working

commit 986f9abbb1c3c862999a1f226476a31af068d7e0
Merge: 593f310 7124497
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Apr 28 17:45:14 2017 -0400

    Merge branch 'master' of https://github.com/jssands/PLTgame

    Merge of N's local changes post getting external libs working

commit 593f310f6a0a760f5887ef8a891094847caecf23
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Apr 28 17:31:47 2017 -0400

    added script to run vtr files with c libs linked

commit 28ad420de13c0befb2ba8dda5e37bb4d371ab68d
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Apr 28 17:09:59 2017 -0400

    input function file created

commit 8e7834c4d7de7b4c26e406e0dc0693e788d64e7a
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 27 17:28:19 2017 -0400

    change printbig to input in testall.sh

commit d33f30f196b37a7a7b310ae3ceeab89d3e5d558e
Author: jssands <js4597@columbia.edu>
```

```
Date:   Thu Apr 27 17:27:30 2017 -0400

    another while test added

commit 71244975c14dc47278c9e7603575963cc18bd909
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 27 13:24:36 2017 -0400

    Delete test.sh

commit 91107507d64a41216d31404fac31b94b012b9f95
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 27 13:23:58 2017 -0400

    work test shell and test suite

commit 2b172624113492548d399f9091a2709b28b68916
Author: nsahra14 <nsahra14@gmail.com>
Date:   Mon Apr 24 16:07:21 2017 -0400

    rename fns/file for string compare

commit 99933848a45cb2f35599c8e1c705925a56276448
Merge: ba8f486 baf7d53
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Apr 21 14:42:45 2017 -0400

    Merge branch 'master' of https://github.com/jssands/PLTgame

    adding strings code

commit ba8f48653ebf9f9a5e4439b961cc63ab0069d527
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Apr 21 13:47:23 2017 -0400

    start of wrapper class for strings in C

commit baf7d53c53244e61420b8e508cc140c814cc7dd4
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 16:09:47 2017 -0400

    string working

commit d7b2adc2a580364c6264c4db09026db3fde0882e
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 15:50:58 2017 -0400

    strings might be working?
```

```
commit 3209c63c625a5c038b05ae61bce09fedbc57d2d3
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 14:13:41 2017 -0400

    updated todo and bugs file

commit 236f62283dc439fe8a0e8b34b57dbd5059c2bb31
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 14:10:44 2017 -0400

    custom functions now working. test more extensively later

commit 2dab3d3f5c904e84ce0a85c1e81f98a479dfd110
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 13:26:48 2017 -0400

    Unary operators added. One bug known, but is minor. Will fix later.

commit ab01069b91a5cc08389d7e6070404f1c2839eaef
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 12:52:42 2017 -0400

    todo updated

commit 2b8962bb94e4ac32459f0712abdccaa2830819df
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 12:52:09 2017 -0400

    primitives finished, AND, OR finished, TRUE, FALSE finished

commit beeeb52fc2d08dc5737ac5c54d17ade386eb9fe3
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 11:50:50 2017 -0400

    todo list updated

commit b016833b833bc118879079b678e311d2fb74e81a
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 11:43:08 2017 -0400

    while loops working

commit 2edfc4b9e4867e0c376b5c4014e359cecedfac70
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 20 09:55:57 2017 -0400

    if else statements working
```

```
commit 60486e2402f367e182846a848586fbcba668df54
Author: Bencd <bc2620@columbia.edu>
Date:   Mon Apr 10 11:49:57 2017 -0400

    first version of testsuite w/o check/checkfailed

commit 1b6b37b220d7c208baa0707a0b9cbbf16c11f3ae
Author: jssands <js4597@columbia.edu>
Date:   Thu Apr 6 14:26:19 2017 -0400

    binops working

commit 2d8aa7ee7335fa0f9439194b3f40426290ab3d78
Author: zpa2001 <zpa2001@columbia.edu>
Date:   Sun Apr 2 19:56:24 2017 -0400

    semant now compiles, just report_duplicate, check_not_void, check_assign

commit 935e6da3b3950743d034e9f3787189cf1ff2ec8d
Author: zpa2001 <zpa2001@columbia.edu>
Date:   Sun Apr 2 18:07:42 2017 -0400

    init commit of semant.ml

commit a317ae2693250faf3cfcc88a6450f743c281d0db
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 18:08:29 2017 -0400

    compile and print llvm code now doable

commit d9db8cd9503ed332c963e343cf40796637ccdcd3
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 16:58:30 2017 -0400

    global variables working

commit 3ccd7abbc0e5d9bad9470e10861815a8c429bd3b
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 14:28:45 2017 -0400

    local vars added to function bodies. Will work on variable assignment next

commit ce2a3797f067b30bdfde538171774d59ef24e09e
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 12:49:32 2017 -0400

    instructions to run code
```

```
commit 8478930bed5b54ce5f370be942750de93dd0ef9f
Merge: b3269f3 35d458e
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 12:47:41 2017 -0400

    Merge branch 'master' of https://github.com/jssands/PLTgame

commit b3269f39ec195fe12209e1ea4578fb56b1673b8a
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 12:45:57 2017 -0400

    sh file to run code

commit 35d458e026b4ca3fc23efb8546a03333cfc97e58
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 12:44:26 2017 -0400

    Delete test.vtr

commit 745eb5faf5cca905c3148547626bc032d73d46a3
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 12:43:52 2017 -0400

    working hello world with printstr function

commit b3fa1c15c73cbb9f5d20efc25306054924986ec1
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 10:39:41 2017 -0400

    Delete make.sh

commit 35483be1a7317cd36103a9cd585dbdcfcdb97671
Author: jssands <js4597@columbia.edu>
Date:   Tue Mar 28 10:39:34 2017 -0400

    Delete clean.sh

commit 1fea84d3957f0b1520d7a596afe01783414d7f29
Author: jssands <js4597@columbia.edu>
Date:   Mon Mar 27 19:46:32 2017 -0400

    Delete example_venture_program.vtr

commit 8145b5921b7e764d6f6d9e999a0355c7350effd9
Author: jssands <js4597@columbia.edu>
Date:   Mon Mar 27 19:46:04 2017 -0400
```

```
    test file

commit 9e6136ad10e11cbb37ab5bd83f5a8ab4872807b4
Author: jssands <js4597@columbia.edu>
Date:   Mon Mar 27 19:42:06 2017 -0400

    update with print

commit bbf9fa1b2b00e1cdc1fad933cdb7276a91259f25
Author: jssands <js4597@columbia.edu>
Date:   Mon Mar 27 19:40:56 2017 -0400

    hello world example with return

commit 3e947a6beec54b2823bcf731be451775c5299691
Author: jssands <js4597@columbia.edu>
Date:   Mon Mar 27 19:40:00 2017 -0400

    working print function?

commit 2208bd6ac054ef749979d45d2baf6e55f5d684be
Author: jssands <js4597@columbia.edu>
Date:   Sun Mar 26 17:13:50 2017 -0400

    accepts main function

commit 7e32016e0a5c455b5142b727a9da47200a3edcd1
Author: jssands <js4597@columbia.edu>
Date:   Sun Mar 26 15:51:55 2017 -0400

    now accepts list of functions

commit 5ffb6fa3dee2364694e47707dbd3e1dd9cad3e44
Merge: e6d5aa6 3631f10
Author: jssands <js4597@columbia.edu>
Date:   Sun Mar 26 13:27:35 2017 -0400

    Merge branch 'master' of https://github.com/jssands/PLTgame

commit e6d5aa68abc2b0e1247a12d8ceac9b84e5ea9a4c
Author: jssands <js4597@columbia.edu>
Date:   Sun Mar 26 13:27:25 2017 -0400

    better formatted codegen - more intuitive indentations

commit 3631f102955daf31ab287c3e84b6f82d07e806c8
Author: jssands <js4597@columbia.edu>
Date:   Sun Mar 26 13:23:43 2017 -0400
```

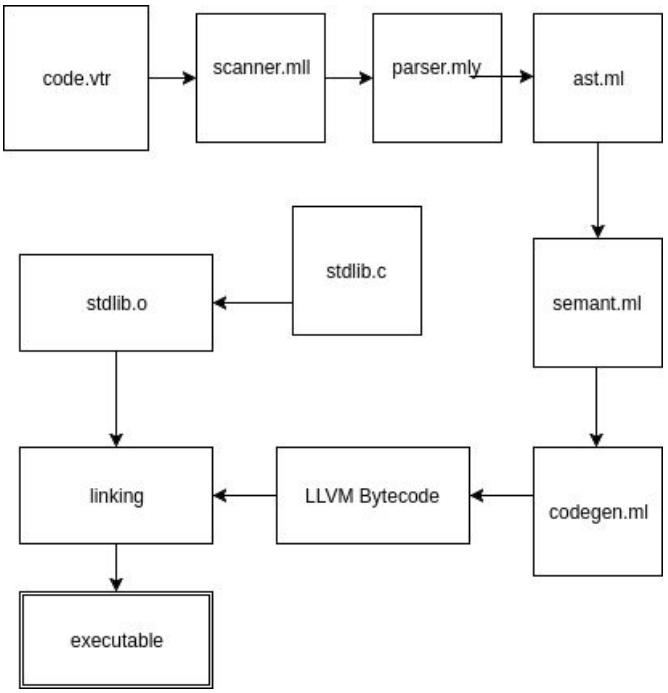
```
    updated to include return statement. Working on codegen now

commit 06a4f97b68c381fc19ac4f7e1e466b77188e3663
Author: jssands <js4597@columbia.edu>
Date:   Sun Mar 26 12:16:56 2017 -0400

    removed option for multiple declarations so parser is working for what we
need

commit bdd058b6a1d5829688ab6afa1080d1f75257406f
Author: jssands <js4597@columbia.edu>
Date:   Sat Mar 25 16:45:02 2017 -0400

    We compile!

commit 6686dc41f96e68d44f7cb9ce93d841d88a9beca0
Merge: e0cf708 7ff9bfd
Author: zpa2001 <zpa2001@columbia.edu>
Date:   Sat Mar 25 16:10:36 2017 -0400

    Merge branch 'master' of https://github.com/jssands/PLTgame

commit e0cf708277584b17f5436832f8ff1a15afd56fe2
Author: zpa2001 <zpa2001@columbia.edu>
Date:   Sat Mar 25 16:07:01 2017 -0400

    modify codegen fdecl and parser has list of fdecl

commit 7ff9bfd8c27e1c2f80af656e2582be874907fd97
Author: nsahra14 <nsahra14@gmail.com>
Date:   Sat Mar 25 15:48:27 2017 -0400

    VENTURE tester file

commit 68c762186bb1abef2336ef7c13de227ca272a215
Author: zpa2001 <zpa2001@columbia.edu>
Date:   Fri Mar 24 19:33:30 2017 -0400

    removed scannerprint.mll that was outside of notes

commit 4826c32afcf69ba23db9d18257025195001b15b9
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Mar 24 17:50:37 2017 -0400

    moved scannerprint for makefile

commit 6e2a08bfc6c2b71e2c649a5c0be59d09e6a0aeda
```

```
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Mar 24 16:42:47 2017 -0400

    initial codegen file; just function decls

commit 4695d7777f1d366af3d529b2ebafb39be69b160c
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Mar 24 16:41:24 2017 -0400

    added bool and void types

commit 069354eebd33ec5bec990e4b80c7af66a164681c
Author: jssands <js4597@columbia.edu>
Date:   Fri Mar 24 16:27:06 2017 -0400

    driver and Makefile added - will work with Julie on them now

commit 6d53e5ded2b0e118c352a698adda9cc01f569198
Author: jssands <js4597@columbia.edu>
Date:   Fri Mar 24 16:26:34 2017 -0400

    Makefile in progress

commit e0384428621fe4bc07f6afce0d2da7241c7a1c61
Author: jssands <js4597@columbia.edu>
Date:   Fri Mar 24 16:15:39 2017 -0400

    driver file added - not yet tested

commit 98468ef39a319a91d75f0f8034b44d9269bb1983
Author: nsahra14 <nsahra14@gmail.com>
Date:   Fri Mar 24 14:39:58 2017 -0400

    scannerprint script to print tokens in a program

commit c548e1a8b8779da6843c6286fb736ee73d390aba
Author: nsahra14 <nsahra14@users.noreply.github.com>
Date:   Thu Mar 23 21:30:03 2017 -0400

    Update what-we-need-codegen.ml

commit 42c0511e5fe74005733788e16f06fea5746ed3ef
Author: jssands <js4597@columbia.edu>
Date:   Thu Mar 23 21:06:20 2017 -0400

    cut this file up as needed

commit 89a51bd86b1e4b180b82887ffc75c099daef1e57
```

```
Author: jssands <js4597@columbia.edu>
Date:   Thu Mar 23 20:35:45 2017 -0400

    todo list

commit 9d0cd347fddbc0990c71aeb06766042af74b565c
Author: jssands <js4597@columbia.edu>
Date:   Thu Mar 23 19:55:47 2017 -0400

    commented codegen file from OH with Julie. Good to look at and learn from

commit 37dbd14fa6b37db92dc374ddd810d325254359bf
Author: jssands <js4597@columbia.edu>
Date:   Thu Mar 23 19:49:30 2017 -0400

    instructions on how to use git

commit 4648b667ad82748e9d8e1151698d9ebecb226f03
Author: jssands <js4597@columbia.edu>
Date:   Thu Mar 23 19:46:00 2017 -0400

    notes from office hours - no focus in particular

commit 2bed60a3eee9bc3267eaeb1a73112c634ecb05fd
Merge: 439a021 fb0b077
Author: jssands <js4597@columbia.edu>
Date:   Thu Mar 23 19:44:49 2017 -0400

    Merge branch 'master' of https://github.com/jssands/PLTgame

commit 439a0212c37a1ff376b041593ee74bc4ea6dba89
Author: jssands <js4597@columbia.edu>
Date:   Thu Mar 23 19:43:50 2017 -0400

    code compiles with simple function declaration

commit fb0b077e05a519be227cf319fb99cd73ed4d401f
Author: jssands <js4597@columbia.edu>
Date:   Wed Mar 22 16:55:09 2017 -0400

    README.md with instructions for shell scripts

commit dab41ac58d09ee6241d67f9cca8a628508eba6f1
Author: jssands <js4597@columbia.edu>
Date:   Wed Mar 22 14:02:10 2017 -0400

    dummy interpreter added for now
```

```
commit d4779cf2d0dd049d0361c2d6c84ca2baa8326a68
Author: jssands <js4597@columbia.edu>
Date:   Wed Mar 22 14:01:28 2017 -0400

    shell scripts added

commit 91331de8bd0237b73c83e45f074cfb5a8a507c65
Author: jssands <js4597@columbia.edu>
Date:   Wed Mar 22 12:25:58 2017 -0400

    added helloworld notes

commit 0b0ec5b1a1fa2ee132b335e79c281c81ad686b64
Author: jssands <js4597@columbia.edu>
Date:   Wed Mar 22 12:22:58 2017 -0400

    scanner.mll

commit b573fd18e16f8a0aeec384d75835bff3a4403f79
Author: jssands <js4597@columbia.edu>
Date:   Mon Feb 6 11:56:47 2017 -0500

    Create README.md
```

Architectural Design



semant.ml: This module checks an ast representation of a Venture program to ensure its adherence to the rule established in the LRM.

codegen.ml: This module takes as input a semantically checked ast and constructs the equivalent LLVM IR code.

venture.ml: This is the driver module which calls the other modules to compile Venture source code.

Compilation:

Makefile: The makefile is used to build the "venture.native" executable file which compiles input code into LLVM IR.

Testing:

testall.sh: Compiles and runs a series of tests specified in the "tests" directory to ensure the Venture compiler is running as expected

External Library:

stdlib.c: Contains C code which provides extra functionality not implemented in the Venture compiler. Linked to LLVM executable before runtime.

scanner.mll:

The scanner is responsible for converting the Venture code from the source file into a series of tokens. The scanner tokenizes input until it reaches an 'eof'.

Contributions: James, Ben

parser.mly:

The parser receives tokens as input from the scanner. From these tokens, the parser constructs an abstract syntax tree, a representation of the program as a whole which takes into account precedence and associativity. Tokens are grouped into subtrees based on the program structure (eg. a function block of code is a subtree). The structure and grouping of the subtrees and the ast as a whole is defined by the CFG in "parser.mly". The parser then passes this ast to the semantic checker.

Contributions: James, Zach

ast.ml:

The abstract syntax tree creates a representation of the data types, expressions and statements of a program which is recognized Venture code. The specifications in "ast.ml" are used later in both code generation and semantic checking.

Contributions: James

semant.ml:

The function of the semantic checker is to comb over the constructed ast for logical correctness. A very important step in the compilation process, semantic checker is validates or disallows a given program passed its simple syntax check. Walking through the ast, the semantic checker checks to make sure that the use of variables are plausible, given its scope and the manner in which it is used makes logical sense. Along with preventing faulty code from attempting to compile, it returns error messages that tells the programmer where the program fails and why. For example, if a user tries to define a boolean and set that variable to the value 4, it would throw an error explaining that the boolean variable was expecting the values: true or false. The program would then not compile and this process of semantically checking the ast would then repeat to make sure the newly rewritten program makes sense.

Contributions: Zach

codegen.ml:

The "codegen.ml" file takes as input an abstract syntax tree, and walks the tree generating the equivalent code in LLVM IR.

Contributions: James, Naina

Compilation:

When compiling the LLVM code into an executable, the "stdlib.c" standard library object file is linked.

Contributions: Naina

Testing Plan

Testing Suite and Justification

The test cases used in our test suite are designed to ensure that Venture programs are compiled and executed as expected as per the LRM. They are divided into two sections: success tests and fail tests. Success tests are meant to execute without error. Fail tests are designed to fail and produce a specific error message. This error message is generated by the semantic checker, and can be predicted accurately as long as the semantic checker is functioning properly. Each test case is named "test-<case>.vtr" and the expected output is in a separate file named "test-<case>.out". Each fail case is named "fail-<case>.vtr" and the expected error output is in a separate file named "fail-<case>.err".

Success Cases

The success tests were designed to ensure the functionality of Venture code including user defined function, global and local variables, control statements, operators. Many of the test cases draw their design from the microc test suite.

Contributions:James

test-add1.out	test-func3.out	test-gcd2.out	test-if2.out	test-var2.out
test.add1.vtr	test-func3.vtr	test-gcd2.vtr	test-if2.vtr	test-var2.vtr
test-arith1.out	test-func4.out	test-global1.out	test-if3.out	test-while1.out
test-arith1.vtr	test-func4.vtr	test-global1.vtr	test-if3.vtr	test-while1.vtr
test-arith2.out	test-func5.out	test-global2.out	test-local1.out	test-while2.out
test-arith2.vtr	test-func5.vtr	test-global2.vtr	test-local1.vtr	test-while2.vtr
test-fib.out	test-func6.out	test-global3.out	test-local2.out	test-while3.out
test-fib.vtr	test-func6.vtr	test-global3.vtr	test-local2.vtr	test-while3.vtr
test-func1.out	test-func7.out	test-hello.out	test-ops.out	
test-func1.vtr	test-func7.vtr	test-hello.vtr	test-ops.vtr	
test-func2.out	test-gcd1.out	test-if1.out	test-var1.out	
test-func2.vtr	test-gcd1.vtr	test-if1.vtr	test-var1.vtr	

Fail Cases

The fail cases were designed to test where the semantic checker would throw an error. Using the test cases as a model, we changed some of the variables, changed assignments, function calls and definitions so that the program wouldn't work. Successful compilations of programs that were expected to throw errors and unsuccessful compilations of semantically correct programs indicated issues with our semantic checker, which we proceeded to fix.

Contributions: Zach

fail-assign1.err	fail-dead2.err	fail-func1.err	fail-func7.err	fail-nomain.err
fail-assign1.vtr	fail-dead2.vtr	fail-func1.vtr	fail-func7.vtr	fail-nomain.vtr
fail-assign2.err	fail-expr1.err	fail-func3.err	fail-func8.err	fail-return1.err
fail-assign2.vtr	fail-expr1.vtr	fail-func3.vtr	fail-func8.vtr	fail-return1.vtr
fail-assign3.err	fail-expr2.err	fail-func4.err	fail-func9.err	fail-return2.err
fail-assign3.vtr	fail-expr2.vtr	fail-func4.vtr	fail-func9.vtr	fail-return2.vtr
fail-assign4.err	fail-expr3.err	fail-func5.err	fail-global1.err	fail-while1.err
fail-assign4.vtr	fail-expr3.vtr	fail-func5.vtr	fail-global1.vtr	fail-while1.vtr
fail-dead1.err	fail-func1.err	fail-func6.err	fail-global2.err	fail-while2.err
fail-dead1.vtr	fail-func1.vtr	fail-func6.vtr	fail-global2.vtr	fail-while2.vtr

Testing Script

To automate testing via the testing suite described above, we made slight modifications to the shell script "testall.sh" provided with the microc code. In order to test the code, simply run "./testall.sh". Ensure that you first type "make" to compile the code for the venture compiler.

The script matches the output of each test case to the matching ".out" or ".err" file. If there is any discrepancy, the testing suite outputs an error message.

Contributions: James, Ben

Representative Source Language Programs

Source Program 1:

```
int main(){

  int i;
  i = 0;
```

```
  while(i < 10){
    print("hello world");
    i = i+1;
  }
  return 0;
}
```

Target Language Program 1:

```
; ModuleID = 'Venture'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt2 = private unnamed_addr constant [3 x i8] c"%c\00"
@str = private unnamed_addr constant [12 x i8] c"hello world\00"

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

declare i32 @scompare(i8*, i8*)

declare i32 @intcompare(i8*)

declare i8* @input()

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

while:                                            ; preds = %while_body, %entry
  %i2 = load i32* %i
  %tmp3 = icmp slt i32 %i2, 10
  br i1 %tmp3, label %while_body, label %merge

while_body:                                       ; preds = %while
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt1, i32 0, i32 0), i8* getelementptr inbounds ([12 x i8]* @str, i32 0, i32
0))
  %i1 = load i32* %i
  %tmp = add i32 %i1, 1
  store i32 %tmp, i32* %i
  br label %while

merge:                                            ; preds = %while
  ret i32 0
}
```

Source Program 2:

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

int main()
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}
```

Target Program 2:

```
; ModuleID = 'VENTURE'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt2 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt3 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt4 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt5 = private unnamed_addr constant [3 x i8] c"%c\00"

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

declare i32 @scompare(i8*, i8*)

declare i32 @intcompare(i8*)

declare i8* @input()

define i32 @main() {
entry:
  %gcd_result = call i32 @gcd(i32 2, i32 14)
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt, i32 0, i32 0), i32 %gcd_result)
  %gcd_result1 = call i32 @gcd(i32 3, i32 15)
  %printf2 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt, i32 0, i32 0), i32 %gcd_result1)
  %gcd_result3 = call i32 @gcd(i32 99, i32 121)
  %printf4 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt, i32 0, i32 0), i32 %gcd_result3)
```

```
  ret i32 0
}

define i32 @gcd(i32 %a, i32 %b) {
entry:
  %a1 = alloca i32
  store i32 %a, i32* %a1
  %b2 = alloca i32
  store i32 %b, i32* %b2
  br label %while

while:                                            ; preds = %merge, %entry
  %a11 = load i32* %a1
  %b12 = load i32* %b2
  %tmp13 = icmp ne i32 %a11, %b12
  br i1 %tmp13, label %while_body, label %merge14

while_body:                                       ; preds = %while
  %a3 = load i32* %a1
  %b4 = load i32* %b2
  %tmp = icmp sgt i32 %a3, %b4
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  br label %while

then:                                             ; preds = %while_body
  %a5 = load i32* %a1
  %b6 = load i32* %b2
  %tmp7 = sub i32 %a5, %b6
  store i32 %tmp7, i32* %a1
  br label %merge

else:                                             ; preds = %while_body
  %b8 = load i32* %b2
  %a9 = load i32* %a1
  %tmp10 = sub i32 %b8, %a9
  store i32 %tmp10, i32* %b2
  br label %merge

merge14:                                          ; preds = %while
  %a15 = load i32* %a1
  ret i32 %a15
```

Lessons Learned & Advice

James:

Lesson 1: Office Hours are your friend.

The most important lesson I learned during the course of this semester I learned very quickly: if you are stuck ask for help. Too often in the past I have been cavalier about a bug or lack of understanding about a problem. I found that simply the process of formulating the question concisely was often enough to help solve it. Julie was helpful every step of the way and going to her office hours was always beneficial.

Lesson 2: Communicate an annoying amount.

No amount of intra-group communication will hurt. Pinging the chat to say you pushed code, or simply just letting each other know what you are working on or struggling with creates a very healthy mentality within the group. With more communication, everybody becomes more supportive of each other, and everybody learns more.

Lesson 3: Spend more time looking at code and less time coding.

The only real progress we made during this project came from going line by line through existing code (ours, previous projects, microc, etc.) and understanding every bit of it. The more time you spend studying code, the less time you will spend wrestling with your own.

Zach:

This was one of the bigger group projects I've worked on in my time at Columbia and this whole experience to me was very new to me. I learned many things throughout the course of the project:

Lesson 1: Don't be afraid to ask any question that pops in your head. Yes, professors always say "Ask the dumb questions, there is a good chance someone else has the exact same question." However, if you ask a seemingly dumb question it could potentially lead to another question that could prove to be more imperative for your group to answer.

Lesson 2: Don't be afraid to defend your design idea. While you need to take into account what your other group members have in mind for the project, you should also understand that what you see for the project is equally as important. And do this all respectfully of course.

Lesson 3: Same as everyone will tell you. START EARLY and HIT OFFICE HOURS. This project is not supposed to be easy, so go and ask questions because you aren't going to find a whole lot of Ocaml help StackOverflow web pages on your own.

Naina:

Lesson 1: Do your job, but don't let that be all you do. As helpful as it was to have the project roles defined at the start, the success of this project absolutely depended on all of us stepping into each other's roles from time to time. At various stages in the project, specific team members had more on their plate, and the more each of us stepped up to help distribute the load, the easier it was for all of us to stay on schedule.

Lesson 2: Be nice to your TA. Going to office hours was our saving grace during this project, but I think the reason those hours were so helpful was that we had a good working relationship with our TA. We did our best to not waste her time, so for us that meant showing up with attempts at code, specific questions, and the ability

to explain what parts we did and didn't understand. In return, she was able to give us very clear and directed advice. And of course, one more open line of communication only helps.

Ben:

Lesson 1: Move beyond HELLO WORLD. Most of my computer science classes prior to this were trying to shift my mindset to think like a programer; however, the level of difficulty of these classes was fairly novice. Most of my time doing coding assignments was spent learning the syntax or debugging simple errors rather than fully understanding what was going on. This was my first class where I learned the necessity and the importance of having deep and full conceptual knowledge of the underlying code before you try to implement it .

Lesson 2: Independence is an asset. This class was also one of my first classes where I felt that I was truly on my own. Not having such a structured timeline and teacher dependent environment at first felt very overwhelming because I couldn't rely on lecture notes, textbooks and due dates. Instead what I found was that you had to rely on learning how to learn and finding what works best for you. For example, while having large coding assignment due every week lead to me always cramming in previous classes, but the looser structure for the project forced me to learn how to pace myself.

Lesson 3: Students/ your group is your greatest resource. Everyone has access to the internet, a textbook and notes; however, those will only get you so far. A large portion of the course was spent solving problems as team and thinking through each problem one part at a time. Together we found that the most successful way to solve a problem was not through a google search or a piazza post but rather by sitting down to together and using four heads instead of one.

Appendix

Source Code:

ast.ml (Written by James)

```
type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Lessthan | Greaterthan | Leq |
Geq | And | Or

type uop = Neg | Not

type typ = Int | String | Char | Bool | Void

type bind = typ * string

type expr =
         Int_Literal of int
       | String_Lit of string
       | Char_Literal of char
       | BoolLit of bool
       | Id of string
       | Binop of expr * op * expr
       | Unop of uop * expr
       | Assign of string * expr
       | Call of string * expr list
       | Noexpr


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


type func_decl = {
    typ : typ;
    fname: string;
    formals : bind list;
    locals : bind list;
    body : stmt list;
}

type program = bind list * func_decl list
```

```
(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Lessthan -> "<"
  | Leq -> "<="
  | Greaterthan -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Mod -> "%"
  | Or -> "||"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    Int_Literal(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | String_Lit l -> l
  | Char_Literal(l) -> String.make 1 l
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | String -> "string"
```

```
  | Char -> "char"
  | Void -> "void"

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

Codegen.ml (Written by James and Naina)

```
module L = Llvm
(*open Ast*)
module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "MicroC"
  and i32_t  = L.i32_type  context
  and i8_t   = L.i8_type   context
  and i1_t   = L.i1_type   context
  (* and ptr_t  = L.pointer_type *)
  and void_t = L.void_type context
  and str_t  = L.pointer_type (L.i8_type context) in

  let ltype_of_typ = function
      Ast.Int -> i32_t
    | Ast.Bool -> i1_t
    | Ast.Void -> void_t
    | Ast.Char -> i8_t
    | Ast.String -> str_t 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

  (*Declaration of 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

  (*Declare other builtin fns included in stdlib c file*)
  let scompare_formals = [(Ast.String, "s1"); (Ast.String, "s2")] in
  let sformal_types = Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
scompare_formals) in
  let scompare_t = L.function_type i32_t sformal_types in
  let scompare_func = L.declare_function "scompare" scompare_t the_module in

  let intcompare_t = L.function_type i32_t [| L.pointer_type i8_t|] in
  let intcompare_func = L.declare_function "intcompare" intcompare_t the_module in

  let input_t = L.function_type str_t [| |] in
  let input_func = L.declare_function "input" input_t the_module in


  (* Define each function (arguments and return type ) so we can call it  *)
  let function_decls =
    let function_decl m fdecl =
      let name = fdecl.Ast.fname
      and formal_types = Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
fdecl.Ast.formals)
      in let ftype = L.function_type (ltype_of_typ fdecl.Ast.typ) 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

(* function bodies built by building expressions and statements *)

(* Fill in the body of the given function *)
  let build_function_body fdecl =   (* function to build body of function *)
    let (the_function, _) = StringMap.find fdecl.Ast.fname function_decls
      in
      let builder = L.builder_at_end context (L.entry_block the_function)
       in
        let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
       let str_format_str = L.build_global_stringptr "%s\n" "fmt" builder in
       let char_format_str = L.build_global_stringptr "%c" "fmt" builder in


    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 (t, n) =
       let local_var = L.build_alloca (ltype_of_typ t) n builder
         in
         StringMap.add n local_var m
           in

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

     (* Return the value for a variable or formal argument *)
       let lookup n = try StringMap.find n local_vars
                      with Not_found -> StringMap.find n global_vars
       in
     let type_of_val = function
             "i32*" -> int_format_str (*int*)
           | "i8**" -> str_format_str (*string*)
           | "i8*" -> char_format_str (*char*)
           | "i1*" -> int_format_str (*bool*)
           | _ -> str_format_str
         in

         let check_print_input = function
             Ast.Int_Literal _e -> int_format_str
           | Ast.String_Lit _e -> str_format_str
           | Ast.Char_Literal _c -> char_format_str
           | Ast.Binop (_e1, _op, _e2) -> int_format_str
           | Ast.BoolLit _b -> int_format_str
           | Ast.Id s -> type_of_val(L.string_of_lltype(L.type_of (lookup s)))
           | Ast.Assign (_s, _e) -> int_format_str
           | Ast.Noexpr -> int_format_str
           | Ast.Unop (_op, _e) -> int_format_str
           | Ast.Call (_s, _actuals) -> int_format_str
         in

    (* Construct code for an expression; return its value *)
    let rec expr builder = function
       Ast.Int_Literal i -> L.const_int i32_t i
     | Ast.String_Lit s -> L.build_global_stringptr s "str" builder
     | Ast.Char_Literal c -> L.const_int i8_t (int_of_char c)
     | Ast.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
     | Ast.Noexpr -> L.const_int i32_t 0
     | Ast.Id s -> L.build_load (lookup s) s builder
     | Ast.Assign (s, e) -> let e' = expr builder e in
            ignore (L.build_store e' (lookup s) builder); e'
     | Ast.Binop(e1, op, e2) ->
```

```
            let e1' = expr builder e1 and e2' = expr builder e2 in
            (match op with
                Ast.Add  -> L.build_add
              | Ast.Sub  -> L.build_sub
              | Ast.Mult -> L.build_mul
              | Ast.Div  -> L.build_sdiv
              | Ast.Mod  -> L.build_srem
              | Ast.And     -> L.build_and
              | Ast.Or      -> L.build_or
              | Ast.Equal-> L.build_icmp L.Icmp.Eq
              | Ast.Neq  -> L.build_icmp L.Icmp.Ne
              | Ast.Lessthan  -> L.build_icmp L.Icmp.Slt
              | Ast.Greaterthan -> L.build_icmp L.Icmp.Sgt
              | Ast.Leq -> L.build_icmp L.Icmp.Sle
              | Ast.Geq -> L.build_icmp L.Icmp.Sge

            ) e1' e2' "tmp" builder
      | Ast.Unop(op, e) ->
         let e' = expr builder e in
         (match op with
           Ast.Neg     -> L.build_neg
         | Ast.Not     -> L.build_not) e' "tmp" builder
      | Ast.Call ("print", [e]) ->
        L.build_call printf_func [| check_print_input e ; (expr builder e) |]
              "printf" builder
      | Ast.Call("scompare", [e1; e2]) ->
          L.build_call scompare_func [| (expr builder e1) ; (expr builder e2) |]
"scompare" builder
      | Ast.Call("intcompare", [e]) ->
          L.build_call intcompare_func [| (expr builder e) |] "intcompare" builder
      | Ast.Call("input", []) ->
          L.build_call input_func [| |] "input" builder

       | Ast.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.Ast.typ with Ast.Void -> ""
                                             | _ -> f ^ "_result") in
         L.build_call fdef (Array.of_list actuals) result builder

       in

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

    let rec stmt builder = function
       Ast.Block sl -> List.fold_left stmt builder sl
      | Ast.Expr e -> ignore (expr builder e); builder
```

```
    | Ast.Return e -> ignore (match fdecl.Ast.typ with
        Ast.Void -> L.build_ret_void builder
      | _ -> L.build_ret (expr builder e) builder); builder
    | Ast.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

    | Ast.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

  in

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

  (* Add a return if the last block falls off the end *)
  add_terminal builder (match fdecl.Ast.typ with
   Ast.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
```

Parser.mly (Written by James)

```
%{ open Ast %}

%token ASSIGN
%token LPAREN RPAREN LBRACE RBRACE SEMI COMMA
%token RETURN
%token INT STRING CHAR BOOL VOID
%token IF ELSE WHILE
%token PLUS MINUS TIMES DIVIDE MOD
%token LT LEQ GT GEQ EQ NEQ TRUE FALSE NOT
%token AND OR
%token <char> CHAR_LIT
%token <int> INT_LIT
%token <string> ID
%token <string> STRING_LITERAL

%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN NOT NEG
%left PLUS MINUS
%left TIMES DIVIDE MOD
%left EQ NEQ LT GT LEQ GEQ
%left AND OR

%start program
%type <Ast.program> program

%%

program:
       decls EOF { $1 }

decls:
       { [], [] }
       | decls vdecl { ($2 :: fst $1), snd $1 }
       | decls fdecl { fst $1, ($2 :: snd $1) }

fdecl:
       typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
       { { typ = $1;
           fname = $2;
           formals = $4;
           locals = List.rev $7;
           body = List.rev $8
       } }

formals_opt:
```

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

formal_list:
      typ ID { [($1, $2)] }
    | formal_list COMMA typ ID { ($3, $4) :: $1 }

typ: /* Datatype - see MAZE */
      INT    { Int }
    | STRING { String }
    | CHAR   { Char }
    | BOOL { Bool }
    | VOID { Void }

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

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


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

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

expr:
         INT_LIT                     { Int_Literal($1) }
       | STRING_LITERAL              { String_Lit($1) }
       | CHAR_LIT                { Char_Literal($1) }
       | TRUE                    { BoolLit(true) }
       | FALSE                         { BoolLit(false) }
       | ID                        { Id($1) }
       | ID ASSIGN expr          { Assign($1, $3) }
       | ID LPAREN actuals_opt RPAREN { Call($1, $3) } /* function call */
       | 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 MOD expr                  { Binop($1, Mod, $3) }
```

```
      | expr LT expr                    { Binop($1, Lessthan, $3) }
      | expr LEQ expr                   { Binop($1, Leq, $3) }
      | expr GT expr                    { Binop($1, Greaterthan, $3) }
      | expr GEQ expr                   { Binop($1, Geq, $3) }
      | expr EQ expr                    { Binop($1, Equal, $3) }
      | expr NEQ expr                   { Binop($1, Neq, $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) }
      | LPAREN expr RPAREN { $2 }

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

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

scanner.mll (Written by James)

```
{ open Parser
(*let unescape s = Scanf.sscanf ("\"" ^ s ^ "\"") "%S!" (fun x -> x)
*)}

let alpha = ['a'-'z' 'A'-'Z']
let digit = ['0' - '9']
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let int = digit+
let char = '''(ascii | digit ) '''
let string = '"' ( (ascii | escape)* as s) '"'
let id = alpha(alpha | digit | '_')*

rule token = parse
      [ ' ' '\t' '\r' '\n']      { token lexbuf }(* whitespace *)
      | "##"                     { comment lexbuf }(* comment start *)
       | '('                     { LPAREN }    (* delimiters *)
      | ')'                      { RPAREN }
      | '{'                      { LBRACE }
      | '}'                      { RBRACE }
      | ','                      { COMMA }
      | ';'                      { SEMI }
      | '='                      { ASSIGN }
      | '+'                      { PLUS }
      | '-'                      { MINUS }
      | '*'                      { TIMES }
      | '/'                      { DIVIDE }
```

```
      | '%'                       { MOD }
      | '<'                       { LT }
      | "<="                            { LEQ }
      | '>'                             { GT }
      | ">="                            { GEQ }
      | "=="                            { EQ }
      | "!="                            { NEQ }
      | "&&"                      { AND }
      | "||"                      { OR }
      | '!'                       { NOT }
       | "true"                   { TRUE }
      | "false"                   { FALSE }
      | "if"                      { IF }
      | "else"                    { ELSE }
      | "while"                   { WHILE }
      | "return"                  { RETURN }
      | "int"                           { INT }
      | "string"                  { STRING }
      | "char"                    { CHAR }
      | "bool"                    { BOOL }
      | "void"                    { VOID }
       | int       as lxm { INT_LIT(int_of_string lxm ) }
      | string     { STRING_LITERAL(s) }
      | char          as lxm     { CHAR_LIT( String.get lxm 1) }
      | id as lxm { ID(lxm) }
      | eof                             { EOF }

      and comment = parse
      | "##"                            { token lexbuf }    (* comment end *)
      | _                               { comment lexbuf }  (* eat everything else *)
```

semant.ml (Written by Zach)

```
open Ast

module StringMap = Map.Make(String)

let check (globals, functions) =

  (* sorts a given list to see if there are multiple entires of same keyword *)
  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

 (* used for checking to see if a variable has type void, returns unit if not*)
```

```
let check_not_void exceptf = function
     (Void, n) -> raise (Failure (exceptf n))
  | _ -> ()
in

let check_assign lvaluet rvaluet err =
  (* types are equal*)
  if lvaluet == 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);

(*reserved function names *)
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 f -> f.fname) functions);

(* built in decls *)
let built_in_decls = StringMap.empty in
let built_in_decls = StringMap.add "input"
     { typ = String; fname = "input"; formals = [];
       locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "intcompare"
     { typ = Int; fname = "intcompare"; formals = [(Int,"x")];
       locals =[]; body = [] } built_in_decls in
let built_in_decls = StringMap.add "scompare"
     { typ = Int; fname = "scompare"; formals = [(String,"x"); (String,"y")];
       locals = []; body = []} built_in_decls

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
(* layer of indirection, need to return binding of print with something???
let check_for_print s = if *)
(* main function must be defined *)
let _ = function_decl "main" in

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);

    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

    let rec expr = function
        Int_Literal _-> Int
      | BoolLit _ -> Bool
      | String_Lit _ -> String
      | Char_Literal _ -> Char
      | Id s -> type_of_identifier s
      | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
       (match op with
          Add | Sub | Mult | Div when t1 = Int && t2 = Int -> Int
       | Equal | Neq when t1 = t2 -> Bool
       | Lessthan | Leq | Greaterthan | Geq when t1 = Int && t2 = Int -> Bool
       | And | Or when t1 = Bool && t2 = Bool -> Bool
        | _ -> raise (Failure ("illegal binary operator " ^
              string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
              string_of_typ t2 ^ " in " ^ string_of_expr e))
        )
      | Unop(op, e) as ex -> let t = expr e in
        (match op with
          Neg when t = Int -> Int
        | Not when t = Bool -> Bool
         | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                       string_of_typ t ^ " in " ^ string_of_expr ex))
        )
      | Noexpr -> Void
      | Assign(var, e) as ex -> let lt = type_of_identifier var
                                and rt = expr e in
```

```
      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 ->
    if fname = "print"
        then
            let _ = List.iter (fun e -> ignore(expr e)) actuals in Void
    else
       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
        List.iter2 (fun (ft, _) e -> let et = expr e in
           ignore (check_assign ft et
             (Failure ("illegal actual argument found" ^ string_of_typ et ^
             " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e))))
          fd.formals actuals;
        fd.typ
  in

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

  (* Verify a statement or throw an exception *)
  let rec stmt = function
      Block sl -> 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.typ then () else
       raise (Failure ("return gives "^ string_of_typ t ^ " expected " ^
                       string_of_typ func.typ ^ " in " ^ string_of_expr e))

    | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
    | While(p, s) -> check_bool_expr p; stmt s
  in

  stmt (Block func.body)

in List.iter check_function functions
```

stdlib.c (Written by Naina)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

char* input( ) {

       char *str = malloc(100);
       fgets(str, 100, stdin);
       return str;

}

char* lower(char *s) {

       char *retstr = strdup(s);
       int i = 0;

       while(retstr[i]) {
         retstr[i] = tolower(retstr[i]);
         i++;
       }
       return (retstr);
}


int scompare(char *s1, char *s2) {

       s1[strcspn(s1, "\n")] = '\0';
        int retval = strcmp(lower(s1), lower(s2));
        return retval;

}
```

venture.ml (Written by James)

```
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)
```

Makefile (Written by James, Naina)

```
all : venture.native stdlib.o

venture.native :
	ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
		venture.native

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf *.diff venture scanner.ml parser.ml parser.mli
	rm -rf printbig
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe
	rm -rf *.lli
	rm -rf stdlib

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx venture.cmx

venture: $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o
venture

scanner.ml : scanner.mll
	ocamllex scanner.mll

parser.ml parser.mli : parser.mly
	ocamlyacc parser.mly

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

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

%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<
```

```
# Inclusion of c files

stdlib: stlib.c
	cc -o stdlib -DBUILD_TEST stdlib.c


### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
venture.cmo : scanner.cmo parser.cmi codegen.cmo ast.cmo
venture.cmx : scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
parser.cmi : ast.cmo
```

Venture.sh (Written by James)

```
./venture.native < $1 > code.ll
llc code.ll > code.s
cc -o code.exe code.s stdlib.o
./code.exe
```

Testall.sh (Written by James, Ben)

```
#!/bin/sh

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

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

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

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

```
#MICROC="_build/microc.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] [.vtr files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

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

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

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

# RunFail <args>
# Report the command, run it, and expect an error
```

```
RunFail() {
    echo $* 1>&2
    eval $* && {
       SignalError "failed: $* did not report an error"
       return 1
    }
    return 0
}

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
    Run "$VENTURE" "<" $1 ">" "${basename}.ll" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "stdlib.o" &&
    Run "./${basename}.exe" > "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

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

CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.vtr//'`
    reffile=`echo $1 | sed 's/.vtr$//'`
```

```
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

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

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

shift `expr $OPTIND - 1`

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

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

if [ ! -f stdlib.o ]
then
```

```
    echo "Could not find stdlib.o"
    echo "Try \"make stdlib.o\""
    exit 1
fi

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

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

exit $globalerror
```

demo.vtr (Written by James)

```
bool coffee;
bool librarian;
bool parser;
bool scanner;
bool ast;
bool codegen;
bool key;
bool dollar;
bool vending;

bool help(string user_input){
       if ((scompare(user_input, "<help>") == 0)
           || (scompare(user_input, "help") == 0)
           || (scompare(user_input, "") == 0)
           || (scompare(user_input, "h") == 0))       {
              return true;
       } else {
```

```
		return false;
	}
}

bool inventory(){
	print("");
	print("Your inventory:");
	if (coffee){
		print("Coffee");
	}
	if (key) {
		print("Key");
	}
	if (dollar) {
		print("Dollar");
	}
	if (ast) {
		print("ast file");
	}
	if (scanner) {
		print("scanner file");
	}
	if (codegen) {
		print("codegen file");
	}
	if (parser) {
		print("parser file");
	}
	print("");
	return true;
}


int character(){
	string character_prompt;
	string user_input;
	int character_choice;

	character_prompt = "Please pick your character type: [1] Manager [2] Language
Guru [3] System Architect [4] Test Master.";

	print(character_prompt);
	print("");

	user_input = input();

	if ((scompare(user_input, "manager") == 0) || (scompare(user_input, "1") ==
0)){
		print("");
		print("You are the team manager... good luck...");
```

```
		print("");
		return 0;
	}

	if ((scompare(user_input, "language guru")==0) || (scompare(user_input, "2") ==
0)){
		print("");
		print("You picked language guru? Well then...");
		print("");
		return 0;
	}

	if ((scompare(user_input, "system architect")==0) || (scompare(user_input,
"3")==0)){
		print("");
		print("You are the System Architect. Hopefully you enjoy headaches.");
		print("");
		return 0;
	}

	if ((scompare(user_input, "test master")==0) || (scompare(user_input,
"4")==0)){
		print("");
		print("You picked test master. Probably a good call. Other people's
messes can be fun to clean up too...");
		print("");
		return 0;
	}

	if (help(user_input)){
		print("");
		print("Enter a number 1-4");
		print("");
		character();
		return 0;
	}


	print("");
	print("Sorry. That is not a recognized choice.");
	print("");
	character();

	return 0;
}

int outdoors(bool description, bool header){
	string text;
	string user_input;
```

```
	if (description){
		text = "You stand outside Mathematics with your team. Rays of sun peak
through the leaves of the oak, showering you four brave souls. You are optimistic for
the semester ahead.";
		print(text);
		print("");
		print("Type <Enter Mathematics building> to get started on your
journey.");
		print("");
	}

	user_input = input();

	if ((scompare(user_input, "enter mathematics building")==0) ||
(scompare(user_input, "enter math") == 0)){
		mathtwo(true, true);
		return 0;
	}

	if ((scompare(user_input, "drop out of school") == 0)){
		print("");
		print("The only winning move is not to play.");
		print("");
		return 0;
	}

	if ((scompare(user_input, "inventory") == 0)){
		inventory();
		outdoors(false, false);
		return 0;
	}

	if (help(user_input)){
		print("");
		print("Your options:");
		print("Enter Mathematics building");
		print("Drop out of school");
		print("Inventory");
		print("");
		outdoors(false, false);
		return 0;
	}

	print("");
	print("Sorry. That is not an option.");
	print("Your options:");
	print("Enter Mathematics building");
	print("Drop out of school");
	print("Inventory");
	print("");
```

```
	outdoors(false, false);

	return 0;
}


int mathtwo(bool description, bool header){
	string text;
	string user_input;

	if (header){
		print("~~~~~~~~~~~~~~~~~~~~");
		print("[Math 3rd Floor]");
		print("");
	}

	if (description){
		print("");
		print("You walk through the double doors into the math building. Your
teammates immediately vanish. You won't be seeing them again.");
		print("You look around. To your left is the library, to your right is a
door labeled [307]. There are stairs leading up, and stairs leading down. There is an
elevator. A student walks out of the library looking... slightly off.");
		print("");
	}


	user_input = input();

	##help##
	if (help(user_input)){
		print("");
		print("Your options:");
		if (!scanner) {
			print("Talk to student");
		}
		print("Enter library");
		print("Enter [307]");
		print("Go upstairs");
		print("Go downstairs");
		print("Enter elevator");
		if (librarian && coffee) {
			print("Throw coffee in trash");
		}
		print("Inventory");
		print("");
		mathtwo(false, false);
		return 0;
	}
```

```
	if ((scompare(user_input, "inventory") == 0)){
		inventory();
		mathtwo(false, false);
		return 0;
	}

	if ((scompare(user_input, "talk to student") != 0) && (!scanner)){
		print("");
		print("The student blocks your path.");
		print("");
		mathtwo(false, false);
		return 0;
	}

	if ((scompare(user_input, "talk to student") == 0) && (!scanner)){
		print("");
		print("You ask: What is your problem?");
		print("");
		print("The student hands you a piece of paper.");
		print("");
		print("You ask: What is this?");
		print("");
		print("The student responds: Your scanner. The rest won't be so easy.");
		print("The student runs off.");
		print("");
		scanner = true;
		if (finished()){
			return 0;
		}
		mathtwo(false, false);
		return 0;
	}

	if (scompare(user_input, "throw coffee in trash") == 0){
		coffee = false;
		print("");
		print("You threw away your coffee. Now you can go into the library.");
		print("");
		mathtwo(false, false);
		return 0;
	}

	if ((scompare(user_input, "enter [307]")==0) || (scompare(user_input, "enter
307")==0)){
		if (key){
			print("You unlock the door with the key from Julie.");
			room307(true, true);
			return 0;
		} else {
			print("");
```

```
                print("This door is locked, you'll need a key.");
                print("");
                mathtwo(false, false);
                return 0;
        }
}

if (scompare(user_input, "go upstairs") == 0){
        print("");
        maththree(true, true);
        return 0;
}

if (scompare(user_input, "go downstairs") == 0){
        print("");
        mathone(true, true);
        return 0;
}

if (scompare(user_input, "enter elevator") == 0){
        elevator(true, true);
        return 0;
}

if (scompare(user_input, "enter library") == 0){
        library(true, true);
        return 0;
}

##help##
if (help(user_input)){
        print("");
        print("Your options:");
        print("Enter library");
        print("Enter [307]");
        print("Go upstairs");
        print("Go downstairs");
        print("Enter elevator");
        if (librarian && coffee) {
                print("Throw coffee in trash");
        }
        print("Inventory");
        print("");
        mathtwo(false, false);
        return 0;
}

print("");
print("Sorry. That is not an option.");
print("Type <help> if you're stuck.");
```

```
	print("");
	mathtwo(false, false);
	return 0;
}

int elevator(bool description, bool header){
	string user_input;
	if (header){
		print("~~~~~~~~~~~~~~~~~~~~");
		print("[Elevator]");
		print("");
	}
	if (description){
		print("Over head an LED flickers. A student stands in the corner, staring
at their phone. None of the buttons are pressed. The buttons read:");
		print("[4]");
		print("[3]");
		print("[2]");
		print("[Drop out of school]");
		print("");
	}

	user_input = input();

	if ((scompare(user_input, "inventory") == 0)){
		inventory();
		elevator(false, false);
		return 0;
	}

	if ((scompare(user_input, "2")==0)
		|| (scompare(user_input, "[2]")==0)
		|| (scompare(user_input, "press [2]")==0)
		|| (scompare(user_input, "press 2")==0)){
		print("");
		mathone(true, true);
		return 0;
	}

	if ((scompare(user_input, "3")==0)
		|| (scompare(user_input, "[3]")==0)
		|| (scompare(user_input, "press [3]")==0)
		|| (scompare(user_input, "press 3")==0)){
		print("");
		mathtwo(false, true);
		return 0;
	}

	if ((scompare(user_input, "4")==0)
		|| (scompare(user_input, "[4]")==0)
```

```
		|| (scompare(user_input, "press [4]")==0)
		|| (scompare(user_input, "press 4")==0)){
		print("");
		maththree(true, true);
		return 0;
	}

	if ((scompare(user_input, "drop out")==0)
		|| (scompare(user_input, "drop out of school")==0)
		|| (scompare(user_input, "press drop out of school")==0)){
		print("");
		print("The only winning move is not to play.");
		print("");
		return 0;
	}

	##help##
	if (help(user_input)){
		print("");
		print("Your options:");
		print("Press 2");
		print("Press 3");
		print("Press 4");
		print("Drop out of school");
		print("Inventory");
		print("");
		elevator(false, false);
		return 0;
	}


	print("");
	print("Sorry. That is not an option.");
	print("Type <help> if you're stuck.");
	print("");
	elevator(false, false);
	return 0;
}

int mathone(bool description, bool header){
	string user_input;

	if (header){
		print("~~~~~~~~~~~~~~~~~~~~");
		print("[Math 2nd floor]");
		print("");
	}

	if (description){
```

```
		print("This floor has a bathroom, the elevator, a vending machine, and a
door leading to Room [207]");
		print("");
	}

	user_input = input();

	if ((scompare(user_input, "inventory") == 0)){
		inventory();
		mathone(false, false);
		return 0;
	}

	if ((scompare(user_input, "inspect vending machine")==0)
		|| (scompare(user_input, "vending machine")==0)){
		print("");
		if (!dollar){
			print("You have no money.");
			print("");
		} else {
			print("Everything in this vending machine costs more than a
dollar. Except... What is that?! A rolled up piece of paper in the bottom row costs
exactly $1.");
			print("");
			vending = true;
		}
		mathone(false, false);
		return 0;
	}

	if ((scompare(user_input, "buy paper")==0) ||
		(scompare(user_input, "buy rolled up paper") == 0)){
		if (!dollar){
			print("You have no money.");
			print("");
		} else {
			print("You buy the paper. Upon inspection you find out it is a
codegen file.");
			print("");
			dollar = false;
			codegen = true;
			if (finished()){
				return 0;
			}

		}
		mathone(false, false);
		return 0;
	}
```

```
if (scompare(user_input, "go upstairs") == 0){
	print("");
	mathtwo(false, true);
	return 0;
}

if ((scompare(user_input, "elevator")==0)
	|| (scompare(user_input, "enter elevator")==0)){
	elevator(true, true);
	return 0;
}

if (scompare(user_input, "Enter bathroom") == 0){
	print("~~~~~~~~~~~~~~~~~~~~");
	print("[Bathroom]");
	print("");
	print("You spend a few minutes in the bathroom, then leave, relieved.");
	mathone(false, true);
	return 0;
}

if ((scompare(user_input, "enter room 207")==0)
	|| (scompare(user_input, "enter 207")==0)
	|| (scompare(user_input, "enter [207]")==0)){
	room207(true, true);
	return 0;
}

##help##
if (help(user_input)){
	print("");
	print("Your options:");
	print("Enter [207]");
	print("Go upstairs");
	print("Enter elevator");
	if (!vending){
		print("Inspect vending machine");
	} else {
		print("Buy paper");
	}
	print("Enter bathroom");
	print("Inventory");
	print("");
	mathone(false, false);
	return 0;
}

print("");
print("Sorry. That is not an option.");
print("Type <help> if you're stuck.");
```

```
	print("");
	mathone(false,false);
	return 0;
}

int room207(bool description, bool header){

	print("~~~~~~~~~~~~~~~~~~~~");
	print("[Room 207]");
	print("");

	print("This lecture hall is full. The lecturer is a handsome man with glasses
wearing a t-shirt with a joke on it. You don't get the joke. He stops lecturing and
turns towards you.");
	print("Tiffany? he asks.");
	print("N-no? you respond.");
	print("You decide maybe you should leave.");
	print("");

	mathone(false, true);
	return 0;
}

int maththree(bool description, bool header){
	string user_input;

	if (header){
		print("~~~~~~~~~~~~~~~~~~~~");
		print("[Math 4th floor]");
		print("");
	}

	if (description){
		print("To you left is the TA help room. There is a staircase behind you
and an elevator in front of you.");
		print("");
	}

	user_input = input();

	if ((scompare(user_input, "inventory") == 0)){
		inventory();
		maththree(false, false);
		return 0;
	}

	if ((scompare(user_input, "elevator")==0)
		|| (scompare(user_input, "enter elevator")==0)){
		elevator(true, true);
		return 0;
```

```
	}

	if (scompare(user_input, "go downstairs") == 0){
		print("");
		mathtwo(false, true);
		return 0;
	}

	if ((scompare(user_input, "enter TA room")==0)
		|| (scompare(user_input, "enter help room")==0)
		|| (scompare(user_input, "enter TA help room")==0)){
		taroom(true, true);
		return 0;
	}


	##help##
	if (help(user_input)){
		print("");
		print("Your options:");
		print("Enter TA help room");
		print("Go downstairs");
		print("Enter elevator");
		print("Inventory");
		print("");
		maththree(false, false);
		return 0;
	}

	print("");
	print("Sorry. That is not an option.");
	print("Type <help> if you're stuck.");
	print("");
	maththree(false,false);
	return 0;
}

int taroom(bool description, bool header){
	string user_input;

	if (header){
		print("~~~~~~~~~~~~~~~~~~~~");
		print("[TA help room]");
		print("");
	}

	if (description){
		print("Oh, thank god. Julie's in the TA room, ready to help you out.");
		print("");
	}
```

```
	user_input = input();

	if ((scompare(user_input, "inventory") == 0)){
		inventory();
		taroom(false, false);
		return 0;
	}

	if ((scompare(user_input, "talk to julie")==0)
		|| (scompare(user_input, "talk julie")==0)){
		if (ast) {
			print("");
			print("Julie can only do so much for you. Go figure something out
on your own.");
			print("");
			taroom(false, false);
			return 0;
		}
		print("");
		print("Julie answers all of your questions, staying well past her posted
office hours ended. Because of her, you now have a working ast!! She is incredible and
deserves a raise.");
		print("");
		ast = true;
		if (finished()){
			return 0;
		}
		taroom(false, false);
		return 0;
	}

	if ((scompare(user_input, "leave ta help room")==0)
		||(scompare(user_input, "leave ta room")==0)
		|| (scompare(user_input, "leave help room")==0)){
		if (ast){
			print("");
			print("Julie stops you: Hey! I think you dropped this.");
			print("She hands you a key. You thank her and leave.");
			key = true;
			print("");
		}
		maththree(false, true);
		return 0;
	}

	##help##
	if (help(user_input)){
		print("");
		print("Your options:");
```

```
                print("Talk to Julie");
                print("Leave TA room");
                print("Inventory");
                print("");
                taroom(false, false);
                return 0;
        }

        print("");
        print("Sorry. That is not an option.");
        print("Type <help> if you're stuck.");
        print("");
        taroom(false,false);

        return 0;
}

int room307(bool description, bool header){
        string user_input;

        if (header){
                print("~~~~~~~~~~~~~~~~~~~~");
                print("[Room 307]");
                print("");
        }

        if (description){
                print("Room 307 is almost empty. There is a large blackboard with strange
symbols on it. On the floor is a single dollar bill.");
                print("");
        }

        user_input = input();

        if ((scompare(user_input, "inventory") == 0)){
                inventory();
                room307(false, false);
                return 0;
        }

        if ((scompare(user_input, "pick up dollar bill")==0)
                || (scompare(user_input, "pick up dollar")==0)){
                print("");
                print("You are one dollar richer. Maybe use it for some snacks.");
                print("");
                dollar = true;
                room307(false, false);
                return 0;
        }
```

```
	if ((scompare(user_input, "leave 307")==0)
		||(scompare(user_input, "leave [307]")==0)
		|| (scompare(user_input, "leave room 307")==0)){
		mathtwo(false, true);
		return 0;
	}

	##help##
	if (help(user_input)){
		print("");
		print("Your options:");
		if (!dollar){
			print("Pick up dollar");
		}
		print("Leave 307");
		print("Inventory");
		print("");
		room307(false, false);
		return 0;
	}

	print("");
	print("Sorry. That is not an option.");
	print("Type <help> if you're stuck.");
	print("");
	room307(false,false);

	return 0;
}

int library(bool description, bool header){
	string user_input;

	if (header){
		print("~~~~~~~~~~~~~~~~~~~~");
		print("Library");
	}

	if (description){
		print("You walk into the library. You can see students with their heads
down studying. You should do that one of these days...");
		print("");
	}

	if (coffee){
		print("The librarian yells at you: YOU CAN'T COME IN HERE WITH FOOD! She
points to the coffee cup in your hand. You leave the library.");
		print("");
		librarian = true;
		mathtwo(false, true);
```

```
		return 0;
	}

	user_input = input();

	if ((scompare(user_input, "inventory") == 0)){
		inventory();
		library(false, false);
		return 0;
	}


	if (scompare(user_input, "leave library") == 0){
		mathtwo(false, true);
		return 0;
	}

	if (scompare(user_input, "study") == 0){
		if (parser){
			print("");
			print("You already did that. Go away.");
			print("");
			library(false, false);
			return 0;
		}

		parser = true;
		print("Congratulations! Your studies have paid off, and you now have a
working parser.");
		print("");
		if (finished()){
			return 0;
		}
		library(false, false);
		return 0;
	}

	##help##
	if (help(user_input)){
		print("");
		print("Your options:");
		print("Leave library");
		print("Study");
		print("Inventory");
		print("");
		library(false, false);
		return 0;
	}
```

```
	print("");
	print("Sorry. That is not an option.");
	print("Type <help> if you're stuck.");
	print("");
	library(false, false);

	return 0;
}

bool finished(){
	if (parser){
	  if (ast) {
	    if (scanner) {
	      if (codegen) {
	       print("");
	       print("Congratulations! You now have a working language. Go enjoy summer
break.");
	       print("");
	       return true;
	}}}}
	return false;
}

int main(){
	string intro_text;

	##set up globals##
	coffee = true;
	librarian = false;
	parser = false;
	scanner = false;
	ast = false;
	codegen = false;
	key = false;
	dollar = false;
	vending = false;

	print("");
	intro_text = "Welcome to 4115 the game. The objective of this game is to build
your own programming language. You will work on a team with three of your peers. In
this game, you must find the pieces to build your language. Good luck!";
	print(intro_text);
	print("");

	print("Type <help> if you're stuck.");
	print("");

	character();
	outdoors(true, true);
```

```
      return 0;
}
```

Test Files: (Written by Zach, James)

fail-assign1.err

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

fail-func3.err

```
Fatal error: exception Failure("illegal void formal a in foo")
```

fail-return2.err

```
Fatal error: exception Failure("return gives string expected int in x")
```

test-func5.out

```
42
```

test-if3.out

```
1
0
```

fail-assign1.vtr

```
int main()
{
  int i;
  bool b;

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; ## Fail: assigning a bool to an integer ##
}
```

fail-func3.vtr

```
int foo(void a, int b){} ## illegal void formal a ##

int main(){ return 0; }
```

fail-return2.vtr

```
int foo(){

  string x;
```

```
  x = "YO YO YO";
  if(true) return x; ## should return int ##
  else return 5;

}

int main(){ return 0; }
```

test-func5.vtr

```
void foo() {}

int bar(int a, bool b, int c) { return a + c; }

int main()
{
  print(bar(17, false, 25));
  return 0;
}
```

test-if3.vtr

```
int cond(bool b)
{
        int x;
        if (b)
          x = 1;
        else
          x = 0;
        return x;
}

int main()
{
  print(cond(true));
  print(cond(false));
  return 0;
}
```

fail-assign2.err

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

fail-func4.err

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

fail-while1.err

```
Fatal error: exception Failure("expected Boolean expression in 505")
```

test-func6.out

```
115
```

test-local1.out

```
84
```

fail-assign2.vtr

```
int main()
{
  int i;
  bool b;

  b = 48; ## Fail: assigning an integer to a bool ##
}
```

fail-func4.vtr

```
int doNothing(){
}

void print(){} ## can't redefine print ##

int main(){ return 0; }
```

fail-while1.vtr

```
int main(){

  int i;

  while(true) {
    i = i + 1;
  } ## forever loop ##

  while (505) { ## needs to be bool ##
    i = i + 1;
  }

}
```

test-func6.vtr

```
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
```

```
}
```

test-local1.vtr

```
void foo(bool i)
{
  int i; ## Should hide the formal i ##

  i = 42;
  print(i + i);
  return;

}

int main()
{
  foo(true);
  return 0;
}
```

fail-assign3.err

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

fail-func5.err

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

fail-while2.err

```
Fatal error: exception Failure("unrecognized function bar")
```

test-func7.out

```
43
```

test-local2.out

```
47
```

fail-assign3.vtr

```
void myvoid()
{
  return;
}

int main()
{
  int i;

  i = myvoid(); ## Fail: assigning a void to an integer ##
}
```

fail-func5.vtr

```
void baz(){
  int c;
  void b; ## illegal void local b ##
  bool f;
}

int main(){ return 0; }
```

fail-while2.vtr

```
int main(){

  int i;

  while(true) {
    i = i + 1;
  } ## forever loop ##

  while(true) { ## bar undefined ##
    bar();
  }

}
```

test-func7.vtr

```
void foo(int a)
{
  print(a + 3);
}

int main()
{
  foo(40);
  return 0;
}
```

Test-local2.vtr

```
int foo(int a, bool b)
{
  int c;
  bool d;

  c = a;

  return c + 10;
}

int main() {
 print(foo(37, false));
```

```
 return 0;
}
```

fail-assign4.err

```
Fatal error: exception Failure("illegal assignment string = int in str = i")
```

fail-func6.err

```
Fatal error: exception Failure("lexing: empty token")
```

test-add1.out

```
17
```

test-gcd1.out

```
2
3
12
```

test-ops.out

```
1
0
1
0
0
0
1
1
1
0
1
0
-10
42
```

fail-assign4.vtr

```
int main()
{
  int i;
  string str;
  i = 5;
  str = i;  ## Fail: assigning an int to a string ##
  return 0;
}
```

fail-func6.vtr

```
int something(int b){
        return b;
}
```

```
void nothing(bool c, int i){

}

int main(){

  int x;
  x = something(4);
  print(x);
  nothing(21); ## wrong number of args #
  return 0;
}
```

test-add1.vtr

```
int add(int x, int y){
        return x + y;
}

int main()
{
        print(add(12,5));
        return 0;
}
```

test-gcd1.vtr

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

int main()
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}
```

test-ops.vtr

```
int main()
{
  print(true);
  print(false);
  print(true && true);
  print(true && false);
  print(false && true);
```

```
  print(false && false);
  print(true || true);
  print(true || false);
  print(false || true);
  print(false || false);
  print(!false);
  print(!true);
  print(-10);
  print(--42);
}
```

fail-dead1.err

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

fail-func7.err

```
Fatal error: exception Failure(" expecting 2 arguments in nothing(true, 21, HEYO)")
```

test-arith1.out

```
114
6
```

test-gcd2.out

```
7
4
11
```

test-var1.out

```
42
```

fail-dead1.vtr

```
int main()
{
  int i;

  i = 15;
  return i;
  i = 32; ## Error: code after a return ##
}
```

fail-func7.vtr

```
int something(int b){
       return b;
}

void nothing(bool c, int i){

}
```

```
int main(){

  int x;
  x = something(4);
  print(x);
  nothing(true, 21, "HEYO"); ## wrong number of args ##
  return 0;
}
```

test-arith1.vtr

```
int main()
{
       print(4*18 + 42);
       print(1 + 2*3 - 1);
       return 0;
}
```

test-gcd2.vtr

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

int main()
{
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  return 0;
}
```

test-var1.vtr

```
int main()
{
  int a;
  a = 42;
  print(a);
  return 0;
}
```

fail-dead2.err

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

fail-func8.err

```
Fatal error: exception Failure("illegal actual argument foundbool expected int in
true")
```

test-arith2.out

```
47
```

test-global1.out

```
42
21
43
22
```

test-var2.out

```
115
```

fail-dead2.vtr

```
int main()
{
  int i;

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

fail-func8.vtr

```
int add(int a, int b){return a+b;}

void voidFunc(){}

int main(){

  add(1,17);
  add(true, voidFunc()); ## bool and void instead of int and int ##

}
```

test-arith2.vtr

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

int main()
{
```

```
        int a;
        a = 42;
        a = a+5;
        print(a);
        return 0;
}
```

test-global1.vtr

```
int a;
int b;

void printa()
{
  print(a);
}

void printb()
{
  print(b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
  return 0;
}
```

test-var2.vtr

```
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
```

```
  foo(73);
  print(a);
  return 0;
}
```

fail-expr1.err

```
Fatal error: exception Failure("illegal binary operator bool + int in d + a")
```

fail-func9.err

```
Fatal error: exception Parsing.Parse_error
```

test-fib.out

```
1
8
```

test-global2.out

```
84
```

test-var.out

```
42
```

fail-expr1.vtr

```
int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; ## Error: bool + int ##
}

int main()
{
  return 0;
}
```

fail-func9.vtr

```
int add(int a, int b){return a+b;}

void voidFunc(){}

int main(){
```

```
  add(1,17);
  string x;
  x = "YO YO YO";
  add(true, x); ## bool, instead of int##

}
```

test-fib.vtr

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

int main()
{
  print(fib(1));
  print(fib(5));
  return 0;
}
```

test-global2.vtr

```
bool i;

int main()
{
  int i; ## should hide the global i ##

  i = 42;
  print(i + i);
  return 0;
}
```

test-while1.out

```
5
4
3
2
1
42
```

fail-expr2.err

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

fail-global1.err

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

test-func1.out

```
42
```

test-global3.out

```
52
```

test-while1.vtr

```
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
}
```

fail-expr2.vtr

```
int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; ## Error: bool + int ##
}

int main()
{
  return 0;
}
```

fail-global1.vtr

```
int c;
bool b;
int c; ## duplicate global variable ##

int main(){
       return 0;
}
```

test-func1.vtr

```
int add(int a, int b)
```

```
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  print(a);
  return 0;
}
```

test-global3.vtr

```
int i;
bool b;
int j;

int main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
}
```

test-while2.out

```
14
```

fail-expr3.err

```
Fatal error: exception Failure("illegal binary operator bool + string in b + e")
```

fail-global2.err

```
Fatal error: exception Failure("illegal void global c")
```

test-func2.out

```
4
8
```

test-hello.out

```
42
71
1
```

test-while2.vtr

```
int foo(int a)
{
```

```
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}
```

```
int main()
{
 print(foo(7));
 return 0;
}
```

fail-expr3.vtr

```
int a;
bool b;

int main()
{
  string e;
  int z;
  a + z;
  b + e; ## Error: bool + string ##
  return 0;
}
```

fail-global2.vtr

```
int c;
bool b;
void c; ## illegal void global variable ##

int main(){
       return 0;
}
```

test-func2.vtr

```
int printem(int x, int y)
{
  print(x);
  print(y);
 }

int main()
{
  printem(4, 8);
```

```
  return 0;
}
```

test-hello.vtr

```
int main()
{
  print(42);
  print(71);
  print(1);
  return 0;
}
```

test-while3.out

```
works
```

fail-func1.err

```
Fatal error: exception Failure(" duplicate function bar")
```

fail-nomain.err

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

test-func3.out

```
62
```

test-if1.out

```
hello
world
```

test-while3.vtr

```
int main(){
       bool tmp;
       tmp = true;
       while (tmp){
              print("works");
              tmp = false;
       }

       return 0;
}
```

fail-func1.vtr

```
int baz (){}

string bar(){}

void foo(){}

bool bar(){} ## duplicate function name ##
```

```
int main()
{
  return 0;
}
```

fail-nomain.vtr

```
##empty file##
```

test-func3.vtr

```
int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

int main()
{
  int d;
  d = add(52, 10);
  print(d);
  return 0;
}
```

test-if1.vtr

```
int main()
{
  if (true) print("hello");
  print("world");
  return 0;
}
```

fail-func2.err

```
Fatal error: exception Failure("duplicate formal b in bar")
```

fail-return1.err

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

test-func4.out

```
##empty file##
```

test-if2.out

```
fail-func2.vtr
int foo(int a){}

void bar(bool b, int a, bool b){} ##duplicate formal bool b ##
```

```
int main(){
  return 0;
}
```

fail-return1.vtr

```
int main(){
       return false; ## should return int ##
}
```

test-func4.vtr

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

int main()
{
  return 0;
}
```

test-if2.vtr

```
int main(){
       if (true) print("true");
       else print("false");

       if (false) print("true");
       else print("false");
       return 0;
}
```