Caml tail

Caml Light, but smaller and less useful

Final Report

Jennifer Lam

Fiona Rowan

Sandra Shaefer

Serena Shah-Simpson

May 11, 2016

Contents

1. Introduction

Caml tail is a functional programming language that is based on OCaml -- without the O. *Caml tail* will be strongly typed (with a smaller set of primitive types). It will support basic arithmetic, pattern matching, function definitions both imperative and recursive, anonymous functions, and nested functions. Our language compiles to C.

2. Language Tutorial

This part of the manual goes over a basic tutorial of the *tail* language.

Step 1: Create your *tail* file
Create a file named `tutorial.tail` in the directory of your choosing.

Step 2: Create Basic Let-Bindings
In tail, all variable definitions begin with a 'let' statement and require explicit type declarations.

Example:
```
let int greeting = “Hello World”;     (* Simple assign statement *)
print_string(“Hello World”);          (* Print hello world *)
```

Step 3: Compile
To compile and output C code from tail code, run the tail executable, input the tail file using a left arrow symbol, and output a C file using a right arrow symbol.
Example:
```
./tail < demo.tail > demo.c
```

Then, compile the C file normally, using the gcc C compiler, and run the C code.
Example:
```
gcc -o demo demo.c
./demo
```

Note:
When writing functions, the last line is returned.
When defining functions within other expressions, statements that come after this definition are unable to access it by name, since it is not within the global scope.

3. Language Reference Manual

A. Lexical Elements

This part of the manual describes the lexical elements of *tail*.

A.1 Separators

Horizontal tabs, newlines, carriage returns, and whitespace separate tokens. They are otherwise ignored. Double semicolons delineate phrases. A single semicolon is used as a sequencer of 1) multiple statements, or 2) list elements.

A.2 Statements

statements:
    expr
    | *expr-definition*
    | *function-definition*

A statement is defined as an expression, an expression definition, and a function definition. It is always terminated by a semicolon.

A.3 Comments

Comments are ignored, and are enclosed by (* and *).

A.4 Identifiers

ident:
    letter {letter | 0...9 | *_}**

letter:
    A...Z | a...z

Identifiers are a sequence of characters beginning with a letter.

A.5 Integer literals

int_literal:
    *[-] *{*0...9*}+*

Integer literals are a sequence of digits in the decimal system, optionally beginning with a minus sign to signify negative numbers.

A.6 Floating point literals

float_literal:
    *[-] *{*0...9*}*+ . *{*0...9*}*+

Floating point literals are a sequence of digits with a decimal point at the beginning, end, or any other intermediary position within the sequence. An optional minus sign at the beginning designates a negative number.

A.7 String literals

String literals are sequences of letters, digits, or symbols enclosed by two double quotes.

A.8 Keywords and special symbols

Keywords are special identifiers that are predefined by the language, and cannot clash with user-defined function or variable names. Below is the list of keywords in our language:

`int`	`string`	`float`	`bool`	`unit`	`match`
`if`	`then`	`else`	`let`	`in`	`with`
`fun`	`func`	`true`	`false`		

The following sequence of characters are special symbols:

`|`	`_`	`(`	`)`	`->`	`,`	`=`	`[`
`]`	`==`	`!=`	`<`	`<=`	`>`	`>=`	`&&`	`||`
`not`	`+`	`-`	`*`	`/`	`mod`	`“`	`;`

B. Values

B.1 Lists

typedecl list :
    [item; item; item;…] | [item] | [ ]
item:

int
| *float*
| *string*
| *bool*

A list is designated by opening and closing square brackets. All items in a list must be literals. The literals must be of the same type (no type mixing or tuples).

B.2 Constants

The following table shows constant values.

`false`	the boolean false
`true`	the boolean true
`unit`	the void value
`[]`	the empty list

C. Patterns

C.1 Variable patterns

These patterns consist of identifiers and match any value, thus binding that variable to said value. The wildcard symbol _ also matches any value, but there is no binding involved. The wildcard is required for any pattern matching.

C.2 Parenthesized patterns

A pattern enclosed in parentheses matches the same value as just that pattern would. Type constraints can be placed on patterns in this way as well, in the following manner: *(pattern : type).*

C.3 “Or” patterns

Or patterns are represented by two patterns separated by the symbol |. A value matches an “or” pattern if it either of these patterns.

C.4 Constant patterns

These patterns consist of constants, and only match values that are equal to those constants.

D. Expressions

D.1 Simple expressions

D1.1 Variables

An expression consisting of a variable always evaluate to the value bounded to that variable.

D.1.2 Parenthesized expressions

An expressions inside parentheses evaluate to the value of that expression.

D.1.3 Local definitions

We bind variables locally using the following formats:

$$\begin{aligned} &\textit{let } \textit{pattern}_1 = \textit{expr}_1 \\ &\mid\ \textit{pattern}_n = \textit{expr}_n \textit{ in } \textit{expr} \end{aligned}$$

Each of the indexed expressions are evaluated, and if their values match their corresponding patterns, then *expr* is evaluated as the value of the entire let statement. Local variables defined in the preceding pattern matches can be used in the evaluation of *expr*.

D.1.4 Function definitions

Declarative functions can be bound to an identifier. The last expression evaluates to the return type, and the functions have the following format:

let $type_{return}$ *function_name (*$type_1$ *formal_arg*$_1$*, … ,* $type_n$ *formal_arg*$_n$*) =*
    $expr_1$*;*
    …
    $expr_m$ *(* the last expr returns a value of* $type_{return}$ **);*
;

For function definitions that are made in other expressions:

let $type_{return}$ *function_name (*$type_1$ *formal_arg*$_1$*, … ,* $type_n$ *formal_arg*$_n$*) =*
    $expr_1$*;*
    …
    $expr_m$ *(* the last expr returns a value of* $type_{return}$ **);*
in $expr_o$*;*

The function function_name is only accessible within $expr_o$. Any of the statements made after

this statement block can not access function_name, since it is not declared within the proper scope.

D.2 Control constructs

D.2.1 Conditional

The expression *if* $expr_1$ *then* $expr_2$ *else* $expr_3$ evaluates the first function, and if that function evaluates to true, it evaluates the second function and returns its value. Otherwise, the third function is evaluated and its value is returned. The else part may be omitted.

D.2.2 Case Expression

The following expression matches *expr* with the following sequence of patterns. If it matches one of the patterns, its corresponding expression is evaluated, and the entire match expression evaluates to its value. If expr matches multiple patterns, the first pattern matched is considered the successful matching.

match expr
with $pattern_1$ *->* $expr_1$
  | $pattern_2$ *->* $expr_2$
…
  | $pattern_n$ *->* $expr_n$
  | _ *->* $expr_{default}$

D.3 Operators

The following are operators within boolean expressions:

`==`	infix	Equality test.
`!=`	infix	Inequality test.
`<`	infix	“Less than” integer test.
`<=`	infix	“Less than or equal to” integer test.
`>`	infix	“Greater than” integer test.
`>=`	infix	“Greater than or equal to” integer test.
`not`	prefix	Boolean negation.
`&&`	infix	Test if both expressions are true.
`||`	infix	Test if at least one expression is true.

The following are numerical operators:

`+`	infix	Integer addition.
`-`	infix	Integer subtraction.
`-`	prefix	Integer negation.

`*`	infix	Integer multiplication.
`/`	infix	Integer division.
`mod`	infix	Integer or float modulus.

E. Built-in Functions

The following functions can be used as built-ins in our language.

E.1 General

`print_bool`	Prints a boolean to standard output.
`print_string`	Prints a string to standard output.
`print_int`	Prints an integer to standard output.
`print_float`	Prints a float to standard output.

4. Project Plan

A. Planning Process

We met as a group frequently to plan the rest of our project, check in, or work together. We met at least four hours a week on any given week, and scheduled fixed blocks of time that we were all free for to talk about or work on our project. We kept in touch outside of those meetings on message apps, when we would ask each other questions or delegate tasks to each other. We also had rough milestones that we should achieve by certain dates, in order to feel how we were doing relative to how we thought we should be doing at any given time. We also kept a shared Google drive with documents where we described issues we had with the code and how we were working through it, so we could easily ask each other questions isolatedly.

B. Specification Process

Up until the Hello World Demo, we wanted our language to be a functional programming language that had similarities to Caml Lite, and compiled down to LLVM. We have changed our language, both on the frontend and on the backend, a lot since then, and even after our Hello World demo. After realizing the magnitude of difficulties related to translating functional programming (our Caml-like syntax) into imperative (our LLVM backend), we decided to compile

down to a higher-level imperative language, C. Our frontend changed as well, mostly in response to needing to simplify our project on the backend in order to come out with a complete project. The significant and late changes to our made it difficult to implement all the features we wanted to see in our language originally.

C. Development Process

We developed our language mostly so that we could comply with the due dates specified for the class. We wrote the Scanner and the Parser in time to write a coherent LRM, and wrote enough of the ast.ml and codegen.ml for users to be able to print strings to the terminal screen in time for the Hello World demo. In the last part of the semester, we made a lot of the changes described in part B, and completed the language in time for our language demo by May 10, 2016.

D. Testing Process

We defined exceptions in our language for the purposes of being able to test features of our language and have a better idea of what was going wrong. Generally, after succeeding in the development process of a given feature, we would immediately write a test in order to test that that feature worked as expected, and if it didn't, we would use the exceptions to try and trace where the problem was coming from.

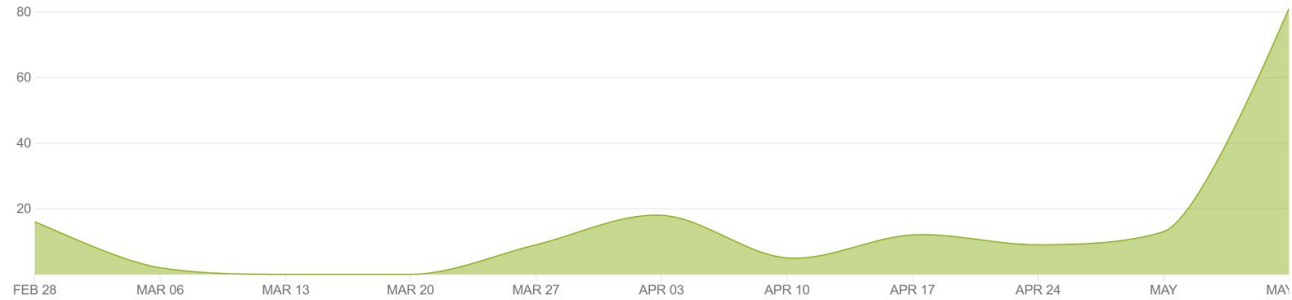
E. Programming Stye Guide

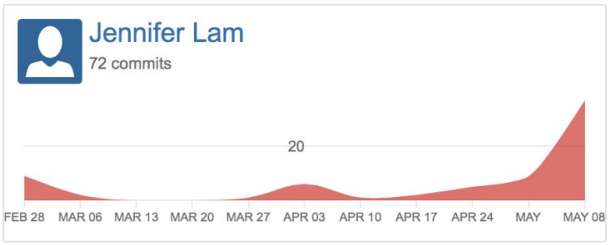
We recognize that organized and readable code was not only easier to understand for any onlooker, but also allowed for better debugging for us as team members. Generally, we tried to have our code conform to the following guidelines:

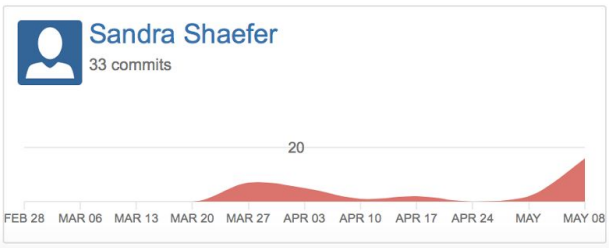
- Insert comments at the beginning of every block of code that explains rationale of code snippets undergoing the development process, or summarizes code snippets that have successfully been brought through the development process.
- Using the same number of tabs to indent lines that are in the same scope, and indent every time you begin a new scope.
- Try to keep the keyword "in" on its own line, so it is prominently displayed

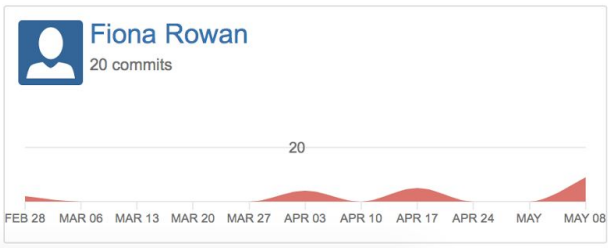
F. Project Timeline

Our commit history reflects the timeline of our project.









G. Roles and Responsibilities

Jennifer Lam	- Implemented Parser and AST. - Implemented “pretty-printing” debugging in AST. - Implemented program that reformats a functionally-expressed program into an imperative list-like structure (RobotsInDisguise). - Implemented program that generates actual C code (Codegen). - Group debugger.
Serena Shah-Simpson	- Implemented Scanner. - Implemented type-checking program (Analyzer) and SAST. - Implemented list construction feature in Parser. - Created test cases and demos. - Group debugger.
Sandra Shaefer	- Implemented Scanner. - Implemented basic pattern matching feature (RobotsInDisguise, Codegen). - Created test cases and demo.
Fiona Rowan	- Implemented testbench suite and test cases.

	- Project document drafter. - Group debuggger.

H. Tools and languages

Over the course of the implementation of our language, we used the following tools and languages.

- Ubuntu 15.10 instance on a virtual machine
- Virtual machine was created using VirtualBox (1 processor, 4096MB RAM, 75GB Hard Drive)
- Git and Bitbucket for version control
- Vim text editor
- Ocaml
- Menhir
- Opam, Ocamlfind
- LLVM
- C

I. Project log

```
commit a325729104b17b9391f3b4e14a497cc8af50941c
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 15:27:16 2016 -0400

    demo2.tail

commit 5cecbf1d0975475ee16db5659169b16f02a2515a
Merge: 5832652 0520dd5
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 14:57:16 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 5832652bf5c71a38a1ae090208276c094f102476
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 14:56:54 2016 -0400

    Added demo2

commit 0520dd593d43b5fdc508f0fad3ae3a5b965b37cd
```

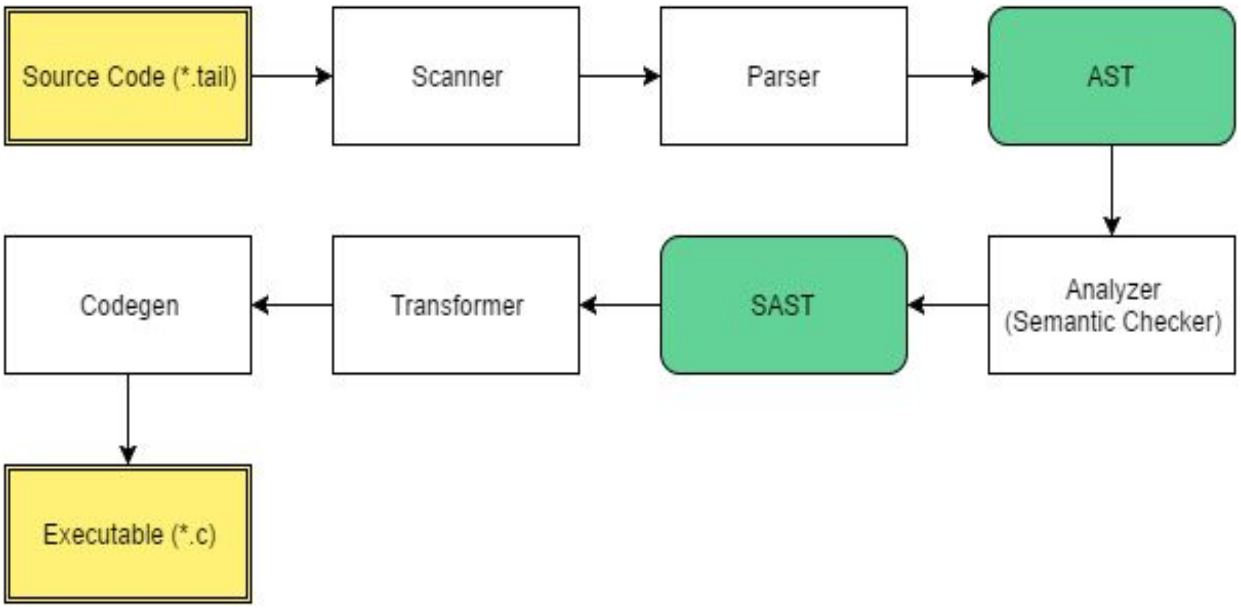
```
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 14:41:47 2016 -0400

    added print strings for expected output

commit dd9c66add4351672fb81df12388c409400afd887
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 14:31:20 2016 -0400

    Fixed some test.

commit 4aa52bad1ca06551a37ad82a2e744615e17deb31
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 14:21:33 2016 -0400

    JUST KIDDING THIS IS THE DEMO

commit 763cf38f30baaea50cd6ae027ae3b48f0180202d
Merge: e659704 93ce3d0
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 14:20:19 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit e65970491a30ec3e3a6b55240d419e925f39d7fe
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 14:20:09 2016 -0400

    Now incudes negative numbers (thanks Serena) and demo

commit 93ce3d0cf50d1d28213cdad0e1a983085ea26088
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 13:40:07 2016 -0400

    logged compiler failures on fail-* test files to be successes as
opposed to failures. also gives test suite status -- if all tests
compile (or don't) as expected, then 'everything works', or else
'there are  tests that don't work'

commit afe29890da96d5f9cd01377071dcc600e4de154c
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:40:24 2016 -0400
```

```
    comments work now

commit e82666e05fd9b7d2ac067246c88f95e43ef372f4
Merge: 3f165c4 9d21c1c
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:33:12 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 3f165c4006631fb189e08d37e5f55132f24ed0b7
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:33:06 2016 -0400

    changed division in Scanner to use /

commit 9d21c1cbc01d025f299df8ba1996c44088f6fe35
Merge: cd0d2d6 fd1c71a
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 12:32:09 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit cd0d2d6967e724801d6324329d60819dd1bf6f74
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 12:31:47 2016 -0400

    Fixed div op in scanner
```

Note: See the rest of the project log in Appendix A.

5. Architectural Design

5.A Block Diagram of Translator Components



5.B Scanner/Parser/AST

5.B.1 Motivating challenges

- Dealing with nesting and scoping was the hardest problem. There are almost an infinite number of ways a user can choose to nest variable bindings, function bindings, let...in's, etc. We want to give the users the sense that they can conveniently nest whatever bindings they choose to, and the program would "just work" for them. As programmers however, we need a way to 1) systematically categorize the different types of bindings, and 2) where in the nest tree each binding is called. For example, a variable might be bound at the beginning of an expression, but it may not be evaluated until the program has traveled deep into the nesting tree. We need to keep track of which scopes can access that variable
- Enforcing correctness of bindings. We could not just dump all sorts of bindings under one user-defined type exprs, because we must prevent, for example, integer variables from being bound to a function definition, and vice versa. We also had to differentiate between expressions that were bound as part of let…in statements or a bindings meant as assignments (for example, let int x = 3; vs. let int x = 3 in s * 5;)

5.B.2 Implementation

Scanner—works as one expects.
Parser/AST
A correct program consists of a list of "statements." A statement can take on three different types: 1) Function binding, 2) Variable binding, 3) Expression statements. The first two act as

simple bindings that return no value but store the variable/function to which the values were bound. The last covers all statements that actually return values and is further divided into separate binding and assignment cases. This division solves both our problems of tracking bindings and enforcing correctness, because 1) we could define Expression statements recursively, so that its nodes fan out to allow for deeper let-bindings without worrying about how that affects function and variable bindings that return no values, and 2) it allows us to define function and variable bindings separately to correspond only to the correct declaration.

Our theme of tracking scopes and understanding the bindings reflects itself in our syntax—although our language is flexible in that it allows the user to nest pretty much infinitely, we require that add semicolons at the end of each overarching statement. This syntax forces the user to keep track of where their nesting trees start and think about exactly what is being nested. Visually, it also helps the programmer see their scopes and eliminate hanging let's or extra in's.

5.C Analyzer/SAST

5.C.1 Challenges

- Enforcing type correctness. OCaml is a strongly-typed language, and we wanted to make sure that expressions matched the type to which they were being assigned. We also wanted to make sure that the last statement of a function evaluates correctly to its return type. Match statements needed to be checked to ensure that values were being matched only on same datatypes. If statemented needed to be checked to ensure that the first expression evaluated to a boolean. All of the function and variable calls of IDs needed to be checked to ensure that the variable or function existed within the scope that was calling it. Exceptions are thrown for each situation in which an invalid operation is attempted.

5.C.2 Implementation

The analyzer takes an AST as input, checks to ensure that all type-matching is correct, and producing an SAST that is guaranteed to have correct typing and that includes additional attached datatypes to certain structures and return values. These datatypes can then be checked in the Transformer and Codegen.

5.D Transformer / RobotsInDisguise and Codegen

5.D.1 Challenges

- Translating a functionally-expressed program into an output that must be imperatively expressed was my biggest problem. Functionally-expressed programs are very elegant, and OCaml in particular has rich scoping rules that encourage nesting definitions with each other. On the other hand, imperative programs express instructions in a list-like format. Translating the scoping rules of OCaml over to correct C was an extremely big challenge.

5.D.2 Implementation

RobotsInDisguise “flattens” out OCaml’s syntax tree from something that branches very deeply into a more list-like imperative format and passes that format (called “env” in our code) to Codegen, which can then more easily output imperativec ode. RobotsInDisguise solves the scoping issue by implementing the concept of access links. It rolls through the statements, gathers all Function Statements and Variable Statements into two StringMaps (key is the name of the function or variable, value is a struct that stores type information and the expressions to which they evaluate), and allows those names to be “seen” by Expression Statements. These correspond to global variables and function definitions outside of main in the C language. In order to prevent OCaml-like features, such as function definitions within function definitions, from showing up in C code (which has no syntax to express this), the actual expressions assigned in Function Statements/Variable Statements are optimized down to simple expressions (literals, binary operations, etc) in RobotsInDisguise before being passed onto Codegen. In addition, in order to take care of the scoping (variables that are defined within a local scope and cannot be “seen” globally), the StringMaps of global variables are passed through the local scopes, which then add their own variables and function definitions. Because OCaml’s structures are immutable, the local variables “disappear” from the maps as soon as they roll out of scope (actually, a new map with the appropriate variables is being created with every new scope, and that map is being destroyed as it rolls out of scope as well), this works very well, because then, expressions in a local scope can see the appropriate variables without having these variables persist past their scopes. This technique is used in both RobotsInDisguise to flatten the scopes. Codegen slightly overlaps responsibilities with RobotsInDisguise in that it recursively travels a list of expressions (that may still be nested), but it produces C output as it travels instead of evaluating the statements.

6. Test Plan

Our testing plan consisted of writing tests for features that we implemented, as we implemented them. These tests would either be expected to fail to compile or fail to run, or succeed to compile and succeed to run. We wrote a script to make sure that each test file resulted in the expected output called run-all.sh. This script was executed after a feature of our compiler was added, and a test for that feature was written. Generally, we held to the practice of defining

descriptive exceptions in our Exceptions module and raise those exceptions. Raising exceptions was both for the purpose of understanding why certain files were not resulting in expected output, as well as helping users debug their own code.

A. Unit Testing

Especially in the early stages, we endeavored to test each module we wrote isolatedly before integrating the module with the rest of the compiler. As we were writing the Scanner, Parser, and AST, we used small test programs that should pass through those modules successfully, and used various unit testing methods and tools to make sure that those programs passed. We used the menhir tool to make sure that the Parser accepted all these programs, using the following shell command:

```
menhir --interpret --interpret parser.mly
```

Additionally, we implemented pretty printing functions in the AST to determine whether the program passed through to that module, and if it did, what the AST layout looked like. We call this unit test by using the following shell command:

```
./tail -a < test.tail
```

B. Integration Testing

Testing the integration of all of our modules involved writing test programs in our language (*.tail files) and compiling them in our language. In order for a test program to compile, it must pass through all of our modules. We wrote tests that involved everything ranging from making sure that data types matched what was expected in the statement (testing the SAST and the Analyzer), making sure referenced identifiers were in scope (Analyzer, RobotsInDisguise), and generally testing to see if a given feature (such as match statements) resulted in the expected output. Tests were written by everyone on the team, either by the implementer of the feature or delegated by the implementer of the feature. When testing, we aimed to try and to find holes in our compiler -- in other words, find test cases that should compile and run but do not.

C. Automation

To automatically run all tests in our test suite, the user can run the following shell command:

```
./run-all.sh
```

The run-all.sh file is a bash script that we implemented to make sure all tests work or don't work as expected. The script evaluates the compilation (translating an arbitrary test file test.tail from tail syntax to C syntax, and then executing the shell command `gcc -o test test.c`) and execution of each *.tail file in the test suite directory "tests". It then outputs these evaluations, and then outputs how many files in the test suite do not result in expected output (i.e how many tests fail).

D. Test Suite

The test suite consists of all test-*.tail and fail-*.tail files in the test directory. The test-*.tail files are programs written in correct syntax and semantics as defined by our language manual. The fail-*.tail files are programs written in either incorrect syntax or incorrect semantics. If these programs do not result in expected output, we know that there is something wrong in the compiler. We have 32 files in our test suite.

E. Caml tail to C

E.1 Testing binary operations

E.1.1 test-binops.tail

```
let int a = 1;
let int b = 2;
let int c = a/b;
```

E.1.2 test-binops.c

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

int a=1;
int b = 2;
int main(){
     return
}
```

E.2 Testing simple match statements

E.2.1 test-match.tail

```
let int a = 1;
match a with
     1 -> print_int(1)
     | 2 -> print_int(2)
     | _-> print_string("wildcard")
```

E.2.2 test-match.c

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

```
int a = 1;
int main(){
      switch(a){
            case(1):
                  printf(“%d\n”, 1);
                  break;
            case(2):
                  printf(“%d\n”, 2);
                  break;
            default:
                  printf(“%s\n”, “wildcard”);
      }
      ;
      return 0;
}
```

E.3 Testing match statements inside functions

E.3.1 test-match-in-func.tail

```
let int func (int b) = fun
      let int a = 1 in
      match a with
            1 -> print_int(1)
            | 2 -> print_int(2)
            | _ -> print_string(“wildcard”);
      0;
;
func(5);
```

E.3.2 test-match-in-func.c

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

int func(int b) {
      int a = 1;
      switch(a) {
            case(1) :
                  printf(“%d\n”, 1);
                  break;
            case(2) :
```

```
                printf("%d\n", 2);
                break;
           default :
                printf("%s\n", "wildcard");
     }
;
return 0;
}
Int main(){
     func(5);
     return 0;
}
```

7. Lessons Learned

A. Jennifer

- OCaml and functional programming as a way of thought. After learning OCaml, I think I regret choosing not major in Math a little less, because I have discovered that a similar elegance exists outside of imperative programming.
- Communication is more important than starting early (I know my group members disagree here, ha). It does not matter how early we start if the coder can't express her thought process to her team members, and it's understandably very hard for contributors to climb that hill on their own. As an example, the Scanner, Parser, and AST were actually completed very early on. If I were to do it again, I would dedicate meetings solely to laying out my thought process and my code and explaining OCaml. As the old saying goes, communication is key.
- Patience. I feel like the project would have gone much more smoothly had I taken the time to just explain the direction I had in mind and provided a more thorough understanding of the project as a whole. Goes hand in hand with communication and starting early.
- Attitude is just as important as ability. I've taken this class once before with a group that prioritized ability and ended up dropping out due to hostility. I'm glad I found a group this semester of just very great people.

B. Serena

My favorite take-away from this project is knowledge of a new language - Ocaml. It can seem like an intimidating foreign language at first, but make sure you dive into it as soon as possible so that you can fully utilize its graceful constructions. Another important takeaway from this project is that it is absolutely essential to have a team leader to set schedules and deadlines for a project. If no one is steering the ship, it will drift around idly for months until it beaches itself on an island from which there is no return. Starting early is not enough, your team needs to make sure that you are keeping momentum in making progress throughout the semester.

C. Sandra

Here are some of the things I learned while working on this project:
1) All plans go out the window when things get desperate and you have less than twenty-four hours to finish.
2) Life gets sort of fuzzy and surreal when you have been working for countless hours straight and it's five in the morning. Your group members start acting crazy and everything seems to relate to OCaml in some way:
Serena: *gets on chair armrest* "You see what I just did? It's like…you know what, it's hard to explain, but it's just like an ocaml construction."
3) START EARLY. Even if you start early, IT'S NOT EARLY ENOUGH. START EARLIER.
4) OCaml is hard to learn, but worth it.

D. Fiona

My biggest takeaway from this project is that it is of the utmost importance to find a way to prioritize it the same as you would other assignments that are smaller and have due dates throughout the semester. Apart from the Proposal, LRM and Hello World demo, there isn't anything that is due until the entire project is due, and that can tend to mean that it is hard to keep a group of four people dedicated to working on features and through problems over the midterms and projects and assignments that are happening in other classes. I think if I were to do this class over again, I would define milestones for the project really well, and have more a more goal-oriented style of working on features weekly.

8. Appendix

A. Full project log

```
commit a325729104b17b9391f3b4e14a497cc8af50941c
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 15:27:16 2016 -0400

    demo2.tail

commit 5cecbf1d0975475ee16db5659169b16f02a2515a
Merge: 5832652 0520dd5
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 14:57:16 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler
```

```
commit 5832652bf5c71a38a1ae090208276c094f102476
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 14:56:54 2016 -0400

    Added demo2

commit 0520dd593d43b5fdc508f0fad3ae3a5b965b37cd
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 14:41:47 2016 -0400

    added print strings for expected output

commit dd9c66add4351672fb81df12388c409400afd887
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 14:31:20 2016 -0400

    Fixed some test.

commit 4aa52bad1ca06551a37ad82a2e744615e17deb31
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 14:21:33 2016 -0400

    JUST KIDDING THIS IS THE DEMO

commit 763cf38f30baaea50cd6ae027ae3b48f0180202d
Merge: e659704 93ce3d0
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 14:20:19 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit e65970491a30ec3e3a6b55240d419e925f39d7fe
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 14:20:09 2016 -0400

    Now incudes negative numbers (thanks Serena) and demo

commit 93ce3d0cf50d1d28213cdad0e1a983085ea26088
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 13:40:07 2016 -0400
```

```
    logged compiler failures on fail-* test files to be successes as
opposed to failures. also gives test suite status -- if all tests
compile (or don't) as expected, then 'everything works', or else
'there are  tests that don't work'

commit afe29890da96d5f9cd01377071dcc600e4de154c
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:40:24 2016 -0400

    comments work now

commit e82666e05fd9b7d2ac067246c88f95e43ef372f4
Merge: 3f165c4 9d21c1c
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:33:12 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 3f165c4006631fb189e08d37e5f55132f24ed0b7
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:33:06 2016 -0400

    changed division in scanner to use /

commit 9d21c1cbc01d025f299df8ba1996c44088f6fe35
Merge: cd0d2d6 fd1c71a
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 12:32:09 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit cd0d2d6967e724801d6324329d60819dd1bf6f74
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 12:31:47 2016 -0400

    Fixed div op in scanner

commit fd1c71a19187a8902bfce4ee2884d760178b6fe8
Merge: 1e87840 e0301b2
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 12:29:32 2016 -0400
```

```
    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 1e87840e7ec87b2fc84bfcf2227fb2e8da239c72
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 12:29:15 2016 -0400

    Added tests for match

commit e0301b2eac90c2443e285b58a66c52497d7a6b67
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:25:38 2016 -0400

    taking out semicolons from return statements with ifs

commit bf23afa9e8f615c01717930008148428e4561e5a
Merge: 797ecd1 32418a3
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:15:41 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 797ecd151d8de2df1d53994e05b784d072fb71c4
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 12:15:28 2016 -0400

    variable orders and functions work now

commit 32418a35f94e7c666421b46030742ec0095264fe
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 11:53:26 2016 -0400

    added some messages

commit 6e31a94526f0ec2be2ce70deb636fbd3309c015e
Merge: f4e86d7 b8d75ac
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 11:45:04 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler
```

```
commit f4e86d7b8a03a4eb44428890726dd6f0ba90c0c4
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 11:44:44 2016 -0400

    this file now compiles / runs the test files, prints fatal
errors, says whether compile / run was successful

commit b8d75ac438c61202e9ebe56da294c774f756f031
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 11:42:46 2016 -0400

    all my print comments are taken out.

commit fdb6903b227a54151befa7dcc964ff8e9a3de620
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 11:37:34 2016 -0400

    booleans now print out correctly

commit 800c4d2ae0c3f32cda916701c04579ed10e4ff49
Merge: 26de314 f3fc897
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 11:28:49 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 26de314750b832fc267a30aed5ac5d8c34592177
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 11:28:30 2016 -0400

    fixed binops not printing

commit f3fc897c9f7111688548db1e5323047967051b47
Merge: 0882492 e951808
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 11:05:36 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 088249275ee5a33ad3a4a60f0767dadc35a4ac1e
```

```
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 11:05:16 2016 -0400

    runs tests, evaluates, and determines whether tests fail

commit e95180837d8cbf63abab9bfc430de3407430f70b
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 10:28:52 2016 -0400

    Fixed some indentation stuff

commit 1ce871ae0d6f26e151a82b486a1bd894b88e4655
Merge: 97d2a04 766bc13
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 10:24:34 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 97d2a04a797962f17bcddbbcf0b45422a81af50e
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 10:24:16 2016 -0400

    Scanner cleared for float ops and analyzer passing on return

commit 766bc13c5d62a2164dea5776c541be433ea4264c
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 10:24:08 2016 -0400

    Fixed some indentation

commit 8bee6d1355c7e0ddaf5c42b29bfeeb68712b8cf3
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 10:19:25 2016 -0400

    Fixed string literal quote problem hopefully

commit 43008897be14f49e7d6739937624b39b5e46b463
Merge: 95408d9 dbcf9df
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 10:13:35 2016 -0400
```

```
    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 95408d99c949775957288693a8d1874b4c90b90b
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 10:13:20 2016 -0400

    match statements works for flow control

commit dbcf9df64cbffdd8a5cca831441a8c455800b022
Merge: a0dce39 e0afc47
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 09:59:13 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit a0dce39fdd03d28f71f64ea931ff60974bfe1c20
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed May 11 09:58:53 2016 -0400

    file that compiles and runs all test- and fail- files in the
tests/ directory

commit e0afc47821a12dcc08c0733099f6ec27974bf242
Merge: 45273f7 dd713f7
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 04:46:34 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit dd713f7ed85f9f54b08c2b1babe8d313e179b428
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 04:46:16 2016 -0400

    If statements recognized for bool lits

commit 45273f71d318c8ba731f030f6cd32e0e31f5d2be
Merge: 813a09d 60ced48
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 04:40:56 2016 -0400
```

```
    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 813a09df4bf4603d8a5c5ff09c0a22deb8485c57
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 04:40:50 2016 -0400

    printing strings fixed

commit 60ced4818eaec270148403c7fc9c6e89f1d6f2fe
Merge: 6f1f565 4d9d385
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 04:40:41 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 6f1f5656fb5658fd4825412fe027d5809024db8c
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 04:40:14 2016 -0400

    Fixed analyzer for if statments on binop booleans

commit 4d9d385c8ae0c9fd1fcdd2b6c25c0c0eebd84bb4
Merge: 4bbe294 8b87fd7
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 04:34:11 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 4bbe29495b91c9499fe97ddeb9bdc259b78eaa35
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 04:34:01 2016 -0400

    tryna fix matching, so commented stuff in robots

commit 8b87fd7886d82cd1143717118e54988c0b351625
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 04:31:13 2016 -0400

    Took out all the wird print statements
```

```
commit 728256ae9c454dd4e92964edac1c24da44b6ea5c
Merge: ef75d98 08b4503
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 04:18:14 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit ef75d9832284d823cc442c8f1db1f15d99245e16
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 04:18:00 2016 -0400

    okay now shit stopped printing out backwards.

commit 08b4503f823a9fc38ddb05cc0b0a00d21ec6c599
Merge: cd34e0a f3ba985
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 04:17:08 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit cd34e0a2e5bde263292ecf627a14ca68d78409fd
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 04:16:41 2016 -0400

    Pushed changes made to attempt to fix match

commit f3ba9853c6971c928289839b9f4c48d0199dbf23
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 04:01:59 2016 -0400

    look at this fibo test case

commit 045c42ed7cefd9a2a6373c81a4c80f97a5cb6e53
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 03:43:38 2016 -0400

    everything fucking works on a basic level

commit 076337d6d222e32d9bc03e988094ce9342842ca3
Merge: 874d6fe 8992fe3
Author: Jennifer Lam <jl3953@columbia.edu>
```

```
Date:   Wed May 11 03:05:51 2016 -0400

    sorry for messing up your indents

commit 874d6fe9e98f393d9a6d8683960737a28527fb93
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 03:01:20 2016 -0400

    BUGS BUGS bzzz

commit 8992fe38652cdf4fceda4a909b1738871ec0ab2a
Merge: 45bacde e6a23d2
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 02:26:30 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 45bacde1116311a9697b549126422f1a7afc42ec
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 11 02:25:56 2016 -0400

    Lists checked for prepending in analyzer

commit e6a23d20d06a7f9929ebf581c390e1d3fb0db105
Merge: 6213160 666207b
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 02:18:24 2016 -0400

    Codegen match compiles and codegen cleaned up and indented

commit 621316032b1dc209d1773f1fba9ff68886e5a336
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed May 11 01:07:28 2016 -0400

    working on adding SMatch to possible expressions, only int
implemented, but mostly commented out bc trying to fix syntax errors.
now going to pull jenns working code to hopefully solve error

commit 666207b2324b2a91d019c362cd4e233dea4578eb
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 01:04:43 2016 -0400
```

```
    Sandra now does it werk

commit 6a65774beb336344c6029f98b1f02346c50b161f
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed May 11 00:53:10 2016 -0400

    Everything works for VERY simple cases

commit d89632c8a9a5e3c89095f3665a15319413fd609c
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 23:53:48 2016 -0400

    Fixed evaluate checking fun_def name

commit cd40516103f21c5cb0917d8a249bff7aff44856b
Merge: 90623ac 3a1521c
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 23:46:07 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 90623ac84a2a870f8b7224887bf7b6472196b9e2
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 23:45:31 2016 -0400

    Analyzer checks prepend operation

commit 3a1521cc7093c5f4e3ec020cda1b65e42f9e3213
Merge: 803099b 380622f
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 23:45:27 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 803099b84117a821b1d0d88fb21ef6ce041bb25b
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 23:45:08 2016 -0400

    compiling version--debugging printing statements

commit 380622fc20e10396324329ce2d84de199ab0c647
```

```
Merge: 8e8dd0b 473d9d2
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 22:56:42 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 8e8dd0b5236f0fc3a0a4d9e6c2eb24a6c6b15e49
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 22:55:42 2016 -0400

    tests added to test suite

commit 473d9d22e966c231ebbc8a731ad8d2194f7101ee
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 22:27:19 2016 -0400

    going to JJ's.

commit f9f08a31e3d2acd38a2794a4f006149c196b7fad
Merge: 759099c b165990
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 22:15:35 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 759099cb45356fbabef291bedc3da00dc0f3d2be
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 22:15:29 2016 -0400

    getting expressions to print out

commit b16599020ae33f4bda6fcd962fcde3765e3857f1
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 22:14:09 2016 -0400

    Function return type checking removed for rec

commit 6ca898b6b21050015ffb7205b677c77773c5bd97
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 22:09:10 2016 -0400
```

```
    Analyzer checks return type, iterative ONLY

commit ec69efc552da6491d977d5bb70241fba59cdc59e
Merge: 941f1ee 1579ee3
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 21:53:27 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 1579ee365b29272c0a027853fdc251df6e67e9e0
Merge: 7004bdf 3929a3b
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 21:52:43 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 7004bdfaa71baa2a64f52f068736b6fb94b279a8
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 21:52:34 2016 -0400

    codegen compiles

commit 941f1ee7f22b76f4383ce2b10f7da95a5fe89dc5
Merge: 71814b8 3929a3b
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 21:51:22 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 3929a3b786fe4036267261a1ea9ee28de64081bc
Merge: 03dd20a aaed066
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 21:50:50 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 03dd20a44e8b863d27b5cb1a12deeffd3139e0e0
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 21:50:43 2016 -0400
```

```
    Analyzes recognizes variables defined in expr

commit aaed0664b891c8eea7f2dbd46049ec117adbdc52
Merge: 3f845ab 051c3dc
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 21:40:58 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 3f845aba4b110ba9946ed348a68e2d7796b2104f
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 21:40:49 2016 -0400

    codegen has print stuff, but doesnt compile

commit 71814b80323f54bbc15a9f50c9917ac06e249fd0
Merge: ef83732 051c3dc
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 21:35:53 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit ef83732bcb3a6fd08acc4381bce3570ebd44bc73
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 21:35:44 2016 -0400

    fixed scanner to stop recognizing double semicolons and now envs
retursn a list of literals.

commit 051c3dc5afaa65d740a8cf531387e61e23e3c032
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 21:34:48 2016 -0400

    Analyzer recognizes function parameters as valid IDs

commit 01996e1a8dd2608a4097f0cd7b8ba01c399b4dc2
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 18:43:09 2016 -0400

    Okay it should work end to end
```

```
commit 2bd5c273ffbc2838afdc95ed0db43489ff30e8db
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 18:41:00 2016 -0400

    Fixed all bugs with codegen.ml

commit d1a50a7f0e21250eb82f242483cd225916f373ab
Merge: ddf2c15 6d05b4e
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 18:22:31 2016 -0400

    Fixed merge

commit ddf2c15595a7f7076f93a385f4e143cfe80962df
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 18:19:21 2016 -0400

    Changed codegen, tail, robots

commit 6d05b4eea429825d62e7e1e896c172fe385f9e88
Merge: 1364126 ff78e7b
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 18:16:57 2016 -0400

    mergey shit

commit 1364126b9f97601b236064d6261e04ed21a2ffbe
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 18:14:58 2016 -0400

    trying to print functions

commit ff78e7bb971a960b272398e7becd2f8b230346d0
Merge: fa5ab81 818196c
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 17:24:37 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit fa5ab81484afaa45d679a7eab453a90f52f303c2
Author: Serena Shah-Simpson <ss4354@columbia.edu>
```

```
Date:   Tue May 10 17:23:50 2016 -0400

    re-committing exceptions and analyzer fixes

commit 818196c152d03952ddb7259dbb2081bbfbe698e2
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 17:19:29 2016 -0400

    Added var generation

commit 6246d4241d0fb22c52dc7dd0712b64846b1d02fc
Merge: 23e05d6 fe92125
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 16:46:59 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 23e05d6f320bd9d357254fc3b1c8203d0caa0c7b
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue May 10 16:46:43 2016 -0400

    Restarting codegen in c

commit fe921256d1e27ea2780ed2dc88793f6616af1cdd
Merge: 02ce7a0 2c296eb
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue May 10 15:23:35 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 2c296ebb14f2316e81fb66f7a748820e97b1405b
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 15:03:34 2016 -0400

    Sandra, Fiona I added a bunch of comments--does this help
clarity...?

commit 1d3786d79811b0c28d7903cfa83ba67bd8820079
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 14:49:35 2016 -0400
```

```
    robotsInDisguise complete without exceptions. tail includes
robotsInDisguise.

commit 406c39d114369e2d52da21d91b41c175701429b6
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 13:56:32 2016 -0400

    changed all the types in robots to sast nodes. minor changes to
sast so literals are referred to on their own.'

commit 5fb09efea744c8ba402ab499cc33360781fb4d32
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 13:19:06 2016 -0400

    robotsInDisguise compiles. Need to add exceptions and change the
types

commit 170b4d0bd7d7d70fd620638c11e34d52ace1054a
Merge: 2c7c897 e4dcaaa
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 00:01:40 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 2c7c897606225f99243f45aa1a20f691d978866e
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue May 10 00:01:31 2016 -0400

    robotsInDisguise...with a bug

commit e4dcaaa24f6dbd07967bb3be554063e2b94caba0
Merge: fca8270 b8b5391
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Mon May 9 22:57:10 2016 -0400

    Pushing merge conflict

commit fca827094f056cf79d11afd43a4319a449b28ac3
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Mon May 9 22:55:22 2016 -0400
```

```
    Fixed (hopefully) issues in codegen, now just waiting on robots
in disguise

commit b8b53915f386cf00034be774a7c5426fd5a65177
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Mon May 9 22:51:15 2016 -0400

    fixed some compiler errors, robotsindisguise has to be able to
compile for me to continue debugging

commit 50207a19ca025d2ccf55192486a30c2b93ea0c78
Merge: d1667a2 c7b0cfc
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Mon May 9 22:08:06 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit d1667a2c28e80afcac11eb616f43cb2143a05c19
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Mon May 9 22:07:29 2016 -0400

    draft of building function defs and decls and var decls

commit 02ce7a0deb48a1f62d14b94de6fa391d0c60cd96
Merge: d3ff938 c7b0cfc
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 21:37:40 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit d3ff938f0f967b1c67e3b5e8c465816427f99414
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 21:37:32 2016 -0400

    Test suite modified

commit c7b0cfc74f930d0eb9a69f2f134677abb6588f9e
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Mon May 9 20:13:20 2016 -0400

    flattens most things in the ast now
```

```
commit fa1f581f0aa4bb010da719e6a80b0165cb33bc16
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Mon May 9 17:20:10 2016 -0400

    Okay, trying to unnest the expressions

commit 353991967304a3a9f41c89ffeadff304b4e8418e
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Mon May 9 16:16:10 2016 -0400

    robots in disguise now includes flattening of if statements,
literals, and operators

commit 7cbc22fbb311b44e569952b9a2aafa2be35c1549
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 15:40:59 2016 -0400

    Fully reverted

commit aad6f0b164049fb4e154d622092b3e0e07155683
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 19:37:45 2016 +0000

    analyzer.ml edited online with Bitbucket because git is a total
pain

commit 3637ac9ea545ebd3a3ef9ef9cbf63e9ef3cc3a8b
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Mon May 9 15:30:17 2016 -0400

    Trying to evaluate the expressions

commit 19177fb534a840cc351cc4a202b902bfb9ea6a3f
Merge: b705553 96a57ec
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 15:23:07 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit b705553548813da2273c03d8100fd9835e8f03b3
Author: Serena Shah-Simpson <ss4354@columbia.edu>
```

```
Date:   Mon May 9 15:22:46 2016 -0400

    Reverted to lists of exprs. Merging handled

commit 96a57ecec4424edb18289e776805a8f37a1b025f
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Mon May 9 15:01:09 2016 -0400

    In process of switching lists to contain only litereals

commit bc9469f5ba06fd6ed5eb8eaab44b4a29717fc63d
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 14:21:12 2016 -0400

    Adding exceptions

commit c2ed63094fff0812af1cf67653dee2b82c29670b
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Mon May 9 14:00:49 2016 -0400

    Okay. I added this file. This better work. I hope it works.
Please?

commit fc83356fa6fa735188101e0680a1a65739eb47e7
Merge: 4e35c11 81978f5
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 13:59:01 2016 -0400

    Still attempting...

commit 4e35c112a3f173dc9b4812ee5b35ba02908057a8
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 13:56:31 2016 -0400

    Attempting to reolve git issues

commit 7b6573969e3f3916ac523a12204ed5d72a238b0e
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 13:31:21 2016 -0400

    Sast checks for prev assigned funcs and vars

commit 81978f52f1136b5c4cbdf160d967f0e8cc635d0b
```

```
Merge: 3cabb03 68193b6
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Mon May 9 00:08:07 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 68193b68eb8b11c3e21f840fdc8e02d0976a754b
Merge: 7b65739 406d827
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon May 9 13:37:47 2016 -0400

    Merging conflicts... overwrote Jen's analyzer to be renamed
transformer

commit 3cabb032d154213cd935b086fbd8cef7b48ad320
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Mon May 9 00:03:51 2016 -0400

    i'm not sure what i changed, but i can't pull w/o commiting.

commit 5fc3d96ca82778e5b7c13df5a70157f77b562e8f
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Sun May 8 20:52:25 2016 -0400

    Majority type checking implemented

commit 406d827f9c67dc782f6ac4221f23bc47b0ffef4b
Merge: 32a34f4 a1c8254
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Fri May 6 23:56:04 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit a1c825495bf4c04705fc0028225e721e20316d9f
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Fri May 6 23:55:43 2016 -0400

    analyzer now adds global functions to a list

commit 32a34f46479c088db9a7b3e02ce3916eed60ccc3
Author: Sandra Shaefer <sns2153@columbia.edu>
```

```
Date:   Fri May 6 23:55:12 2016 -0400

    Added some code into codegen for expr def and types, but probably
doesnt work so its commented out

commit e4c7b9e0c5926e83810e1703e1e612f956059a0e
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Fri May 6 23:37:32 2016 -0400

    analyzer--implements going thru and collecting global variables

commit 46b262c3b5790a231cd0d1ccac8a4602e98cd05c
Merge: c2979ad f3830b7
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Fri May 6 19:55:56 2016 -0400

    merging conflicts

commit c2979ad1de545c17006fbbbf00d1dae143a47e15
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Fri May 6 19:52:21 2016 -0400

    added list to parser, scanner, ast

commit f3830b7ae2e92fbb7a497642f4a39c54bd96fb34
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Fri May 6 17:42:55 2016 -0400

    fixed shift reduce conflicts--now all function defs end in
semicolons

commit 24c4ee98fc1eeee38d1f82dbb308793558005b87
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Fri May 6 01:17:09 2016 -0400

    added a pretty printer to ast. function definitions now take
expressions, not statement lists

commit 9b66fee2e1e5eda159ed449bc7a41ce9781d0864
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Thu May 5 23:21:06 2016 -0400

    fixed the parser to accept expr, not exprs in statements
```

```
commit b91118a77a9cd58a46d3a066ac8d617d73f6af74
Merge: 6f73c89 d0dbfb3
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Thu May 5 22:49:24 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit d0dbfb3893fbf13a88fe2f55207968670d99defc
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Thu May 5 22:48:38 2016 -0400

    got rid of things

commit 6f73c894bf66f86e1e1767fad41cd3ad02889a0e
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Thu May 5 22:47:26 2016 -0400

    Added Id to expr in ast and parser

commit 3a318d916632d9f20b899bf55e14ddc94f2c94d8
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Thu May 5 20:15:43 2016 -0400

    program is now just a list of statements

commit 7d7d913acae87125086047bf18261260d123f6ec
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Thu May 5 17:18:05 2016 -0400

    Parser and ast complete.

commit a177fa5fa2ee8fdf2a7cc73221d6d3db27b593ee
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Thu May 5 16:41:46 2016 -0400

    made some changes to ast and parser. let's see if they work.

commit 14b641020d09aa2fe61ea1c1c38010380e6b6f99
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed May 4 13:53:30 2016 -0400
```

```
    Commented out match code to focus on through connection... type
mismatch

commit b125f32574b2ee5037782abe3fd3806662fef821
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed Apr 27 18:18:54 2016 -0400

    Sorted more in codegen. There is something wrong with the way
I've set up expr and exprs in Ast and parser... see error

commit 26408e6457e239afafc8840c65c17ad7de9dcd6d
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue Apr 26 21:08:53 2016 -0400

    match sorted in parser. more shift reduce conflicts tho -_-'

commit 26f381f91e8596836982e890450c2cda64b5d975
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue Apr 26 20:04:07 2016 -0400

    separated parts for expr, defs, cond, match does NOT work yet

commit 7dd5420e150e8a8f00e11d668d52184ab7f0f1db
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue Apr 26 19:30:10 2016 -0400

    function declarations and expression declarations are now allowed
standalone

commit 0d7a6fe792c0383ea66d01bbf9f0f6c3289eceba
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue Apr 26 19:18:24 2016 -0400

    trying to allow function definitions and expression assignments
as their own standalone statements

commit f57f203a0011b2e1868c2a747d6ebe263996bc29
Merge: b4b37ae fb1adcf
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue Apr 26 18:36:58 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler
```

```
commit b4b37ae8a76c292710d1d504f3e2ddb73883bb18
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue Apr 26 18:36:26 2016 -0400

    printf separated into print_int, print_float, etc. print_char and
print_bool not functional

commit fb1adcfbc05d6cefa728552de204da9954f2c859
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue Apr 26 18:32:31 2016 -0400

    deleted fiona_parser.mly

commit 7207c74532de1df09b36f014a36937d6f18133aa
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue Apr 26 18:21:34 2016 -0400

    parser works now--structure of a program is now a list of blocks

commit 6d0324acf537a231aa7e906c7b936b31e20c5dbf
Merge: 8e263de c0bfe49
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue Apr 26 17:08:31 2016 -0400

    merge?

commit 8e263de14cd0a4d391c744726757cef3a001331b
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Tue Apr 26 16:54:46 2016 -0400

    Resolved function and expression definitions.

commit c0bfe49ebaec63cab4912fe473f62108c9088b87
Merge: 83b4ac2 7dbb98a
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 19:07:11 2016 -0400

    pulling the comment in tail.ml
    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 83b4ac2787745db6b08f1eca2c4a694763751940
```

```
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 19:05:25 2016 -0400

    the ast, parser, and scanner compile with new let-binding
packaging

commit 7dbb98a01384439f94ca3b06f0d86d14df5e60c5
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 19:04:52 2016 -0400

    added a comment for how to run our language at the moment

commit 31b7f3c9bdcf922d7920b847b89c76a1d5847f92
Merge: 95e4d85 20aa941
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 18:28:47 2016 -0400

    accidentally removed the scanner from repository so retrieved the
most recent commit
    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 20aa941919df914b5e8944f7f2e0b6a49f11a0de
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 18:28:24 2016 -0400

    retrieved most recently pushed scanner

commit a4d48936488735d49e38520db9930c05416b2dd5
Merge: c7c0adb bba31cf
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 17:31:28 2016 -0400

    trying to get the scanner back

commit 95e4d85a4a531b3847f5df59784a0df4e24ad4f3
Merge: b653259 bba31cf
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 16:39:18 2016 -0400

    merge conflict with parser

commit bba31cff7a27ac82f58fda10bb6f10557fd914ab
```

```
Merge: f6b313a 32c9204
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Apr 23 16:29:01 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit f6b313af63a0d0c4a1da0f03418bf4a5eb2692ea
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Apr 23 16:28:54 2016 -0400

    shift reduce errors went away...and floats are commented out

commit b6532598d5178d4e5d8000d7f832ca3370a0e431
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 16:16:03 2016 -0400

    integrated new let_binding packaging into the real parser, with
new tags

commit 32c92049f24c4ce4edc883d7696c793f7a83601d
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 16:11:30 2016 -0400

    program now takes in list of let_binding_simple, which makes sure
the scope of complex let_binding is not let_binding_Simple

commit 53b58ce3280633bfe323fb6f7f3d1dc1640ac281
Merge: 3b3c9ca a360805
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Apr 23 16:08:34 2016 -0400

    resolved merge conflict

commit a360805106b9b0fa38fb09c579acb480461a02af
Merge: 574903d 8078102
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Apr 23 15:29:39 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 574903dc316d3eb9383e294e7e92fc18feeab062
```

```
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Apr 23 15:29:27 2016 -0400

    implemented function declarations

commit 8078102df8d6a82e252e68d02d448cb340413521
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Sat Apr 23 14:08:54 2016 -0400

    added functionality for float operations

commit d06e94f7058aeb6ade0625d6229ca98a1144c735
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue Apr 19 02:08:25 2016 -0400

    Passing the string to the print function is now better designed
in codegen, i.e. calls a method

commit a7e2ab62cf24baf588032e554c38f75abffbf6ae
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue Apr 19 00:43:09 2016 -0400

    Yoooo hello world worksssssssss, basically the
Llvm_analysis.assert_valid_module line in the tail.ml was breaking
the module (asserts are harmful, according to Phil), and then the
string literal being passed to the printf_func in the expr call in
codegen.ml needed to be returning a i8*, i.e. string pointer, not a
constant literal

commit 911fa46c215377a32913ae256dc540b7268bf601
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon Apr 18 16:48:19 2016 -0400

    added wildcard to scanner, matching on let_binding tag in
codegen. there is an llvm error on generated code when run with
test... broken module found

commit 0a3ea6282d1f72136b0e1db82ee817e2555bd350
Merge: f387fa6 fd2d3c8
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon Apr 18 13:26:51 2016 -0400
```

```
    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit f387fa6707f6c6bc002531dfdd18948e5bf0c328
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon Apr 18 13:26:15 2016 -0400

    let-bindings packaged in lists, each containing internal lists of
let_bindings where applicable

commit fd2d3c8d4a0aa9f0ec504db94e5ef98de7be8b53
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Thu Apr 14 17:14:41 2016 -0400

    Modified pattern in parser to only take literals for now

commit 37ab79b9317c8dfbfe6136f2fdb4c2e54b27ae6b
Merge: a96f5d9 30f52f0
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Thu Apr 14 16:31:37 2016 -0400

    Merge branch 'master' of
    https://bitbucket.org/plt_sandwich/compiler

commit a96f5d9ee159f4667241556728e64c50aaad1d8d
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Thu Apr 14 16:31:10 2016 -0400

    changed ast to recognize different program makeup

commit bcdfd2c604f4815f12658cc24b74ceb71d18c13f
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Thu Apr 14 16:29:51 2016 -0400

    started implementing pattern matching in parser/scanner. syntax
changes made to let_binding packaging, packagging still not cohesive

commit 30f52f08517de261db04bb140734c7bc1a4f7c72
Merge: ae0d634 edaf4c0
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Thu Apr 14 15:50:00 2016 -0400

    Fixing merge conflicts
```

```
commit edaf4c0c87b7b26c014c8866ba9b531918d9afa2
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Thu Apr 14 14:56:06 2016 -0400

    Added type decls for let_binding statements, program now consists
of list of let_bindings

commit 18cafe54bd06129a441bd1101f61f3b5e8fa220e
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed Apr 13 01:54:09 2016 -0400

    corrected function definition in parser, took out cheater_func

commit ae0d6349e6d9588bba1c8c062f1a7ddaaf03befa
Merge: 2faeba1 edf7566
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Fri Apr 8 23:10:31 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit 2faeba1688a2d808d9e1830909c64837424bfe76
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Fri Apr 8 23:09:21 2016 -0400

    Added most of expr to codegen and changed type of char to an int

commit edf756658893700db208d4b12f1ba10ff5005871
Merge: 1f26539 1033e33
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Fri Apr 8 15:20:40 2016 -0400

    Merged jennbranch into master

commit 1033e3366a3b0ca424cf66f55e30c43177d7f83d
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed Apr 6 23:06:27 2016 -0400

    reformatted parser file

commit 437f9f1940379e8ab2cf246bdf28f60ac654c216
Author: Fiona Rowan <fmr2112@columbia.edu>
```

```
Date:   Wed Apr 6 22:52:22 2016 -0400

    organized tokens

commit 3b3c9ca0564fc1517543663de670826aca78d6f0
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed Apr 6 22:19:48 2016 -0400

    organized tokens using variables

commit a6227654d6883cf550c1286e6cd68bd78fcec404
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Wed Apr 6 22:01:41 2016 -0400

    organize comments of tail.ml

commit a2845569abd65bb5864dbc5997ace587665761ed
Merge: ba3e660 edf7566
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed Apr 6 16:14:44 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit ba3e660042ef5cccee0f01864cf45b743dffebcd
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed Apr 6 13:54:51 2016 -0400

    okay everything works submit yes

commit 1f26539ceb236a533107e8fbf457145ee64ca20b
Merge: 4624674 63e9baa
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed Apr 6 13:35:46 2016 -0400

    Commiting merge

commit 63e9baa3e4a6d68728fa2f143226f98b5eaa2747
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed Apr 6 13:32:40 2016 -0400

    parser.mly now has the actual newlines in it
```

```
commit aea47253b272b2b3f05c0c27417f1c85ec4f377e
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed Apr 6 13:29:47 2016 -0400

    segfaults everywhere.

commit 4624674fbaa9449f9312ab0211b9cecabb6874f0
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed Apr 6 13:13:25 2016 -0400

    Fixed lexbuf empty token error by adding newlines to scanner
whitespace

commit 547db0ab4ea045152fa20e525ce05f659d4b06be
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed Apr 6 11:12:49 2016 -0400

    omg it works now. script added too.

commit aa2f01866b0f2d0f2eeca66f78f916901420e0ef
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed Apr 6 10:56:52 2016 -0400

    okay it generates real llvm code now

commit 95776bb1cee474ada7e727918732f5ff30729c81
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Wed Apr 6 10:43:28 2016 -0400

    got rid of segfault

commit 6186ddaafb173fe6c2fd1d88e3cd51d3e2a7c045
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue Apr 5 16:12:37 2016 -0400

    now compiles again, but expr is not being used in code gen and
running tail on a file results in lexing: empty token error

commit 288cd52672035d3deb9ab3a996fa088638da8c0f
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon Apr 4 16:44:09 2016 -0400
```

```
    Made changes to codegen. Now no longer compiles, but I think
we're on the right track...

commit 65b6cd6d1a51473485c3119b5b0fcd7466d37fa0
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Mon Apr 4 14:57:17 2016 -0400

    Code compiles! Changes made with what is sent to codegen. Errors
on parsing

commit 98f8f510430a4f2f4dfad514f71f1a792f3f1753
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Sun Apr 3 18:34:07 2016 -0400

    Worked around core

commit 3424965ad1e0aeb59e0c475307e111eab321f016
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Sun Apr 3 18:05:58 2016 -0400

    Trying to fix Core unbound module error

commit d77786a690ea506674e7595f41758df58a5d491f
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Sun Apr 3 17:02:29 2016 -0400

    Working through compile errors, now having issues with
scanner.mll

commit d94b1724b9c0fb7d382ca2c88569007eb9d206fe
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Fri Apr 1 22:04:31 2016 -0400

    Modified makefile, trying to get codegen to work

commit b203d451d276d132917e3e8f95b9005329612f40
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed Mar 30 14:22:37 2016 -0400

    Bc serena asked

commit 159c69365b43d23b66e59614dd750efc118398dd
Author: Sandra Shaefer <sns2153@columbia.edu>
```

```
Date:   Wed Mar 30 14:16:06 2016 -0400

    Change in tail.ml

commit 375f385f38eddc3585ec6d30adabe4003492b989
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Wed Mar 30 14:09:23 2016 -0400

    Modifed tail.ml and codegen.ml

commit cf1da0333b16bb634fd73b883f79dc9899de3b87
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Tue Mar 29 12:40:41 2016 -0400

    Adding tail.ml

commit d1ae10baa67d7013198b8a4115a36e2f0b6453ad
Merge: c0cc345 4407ee0
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Sun Mar 27 22:06:59 2016 -0400

    Merge branch 'master' of
https://bitbucket.org/plt_sandwich/compiler

commit c0cc3456a5730c35cb30de300fd2edafa00cc904
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Sun Mar 27 22:06:42 2016 -0400

    Makefile

commit 4407ee0fa487b3f6ac70f30fc2a04925dc9fdec5
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Sun Mar 27 22:05:10 2016 -0400

    CHEATER for hell world separated to take strings

commit fab0b7c5c47732c9a8eaeed33c8405f5cd6a9939
Author: Sandra Shaefer <sns2153@columbia.edu>
Date:   Sun Mar 27 20:23:38 2016 -0400

    Changed some tokens

commit f3f306aa42935ebb056948f273cdc9e6c522f38b
```

```
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sun Mar 27 19:49:11 2016 -0400

    Getting started on Hello World

commit 4646ebe060c609ffa074c1736ec97f4b626dd587
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Mar 5 19:08:37 2016 -0500

    scanner updated to reflect logical operators. and now
commentedgit add scanner.mll!

commit 294ec4ed0dd28663171faa7bd365cc6e76ee9583
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Mar 5 19:01:56 2016 -0500

    Added in logical operators

commit 180756e41cbdb97afc6cfde1606e2c6360d6f1dd
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Mar 5 18:56:33 2016 -0500

    expression list corrected--now takes in single expressions

commit 7c0dc23d25c95c191990e7c1bd7fbc6b8c7ae03f
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Mar 5 02:39:42 2016 -0500

    Changed the keyword function to fun. Ooops.

commit 23f29913abb29a64ac9d23e0a2fce4142172b089
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Mar 5 02:23:38 2016 -0500

    Updated the scanner to reflect the new parser

commit 513576a6e965c87ba53f218f2d76cc7838656726
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sat Mar 5 02:17:01 2016 -0500

    Function definitions now work.

commit 76198683065ceaf9bc552bee79ca1a83a78365cb
```

```
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Fri Mar 4 22:45:56 2016 -0500

    Expressions works with ast file

commit 03a5a4202593cc8667c3d38973135c54697a625a
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Fri Mar 4 18:14:09 2016 -0500

    Expressions work now

commit 2e60cab440fdce4f3e73f9d0144fad5c14eeb89b
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Wed Mar 2 14:17:57 2016 -0500

    Added functionality to parser

commit e2b859c60268a889fff4789309d1f528540a0204
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Tue Mar 1 21:39:52 2016 -0500

    parser working with menhir for simple let statements

commit 0e801fb0f45e6158040daa45fb0b551499dea140
Author: Serena Shah-Simpson <ss4354@columbia.edu>
Date:   Sun Feb 28 15:57:29 2016 -0500

    Added tokens to parser.mll

commit 8ad3f8374d42ce5e94f2887c3545e1dcac780291
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sun Feb 28 15:00:29 2016 -0500

    all fixed. oops

commit f1f1af23e548abfa5e7ce8d577a76ef2e2a0956f
Author: Jennifer Lam <jl3953@columbia.edu>
Date:   Sun Feb 28 14:54:14 2016 -0500

    environment setup added

commit 77015f97ab968c423f03e461e83ea4c18066e01f
Author: Jennifer Lam <jl3953@columbia.edu>
```

```
Date:   Sun Feb 28 14:49:31 2016 -0500

    Added correct microc-llvm tarball to repo.

commit c7c0adbdf9488642c44c88b9c65cdea4a31919f6
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Feb 27 23:25:49 2016 -0500

    edited scanner

commit 1b77687120fcd0a71285215027dae94d60deaf9b
Author: Fiona Rowan <fmr2112@columbia.edu>
Date:   Sat Feb 27 22:53:40 2016 -0500

    wrote scanner.mll with lexical elements detailed in latest
proposal

commit cdf4bc278088e335d007c35c128230f913104963
Author: Sandra N Shaefer <sandra@sherlock.is.private>
Date:   Sat Feb 27 21:59:58 2016 -0500

    More files!

commit 371728896fbe1e0a241cf8372a13b78eacc041ba
Author: Sandra N Shaefer <sandra@sherlock.is.private>
Date:   Sat Feb 27 21:54:40 2016 -0500

    Hi guys\!
```

B. Source Files

B.1 scanner.mll

```
(* Ocamllex scanner for tail *)

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

let digit = ['-']*['0'-'9']
```

```
let lowercase = ['a'-'z']
let uppercase = ['A'-'Z']
let letter = (lowercase | uppercase)

rule token = parse
  [' ' '\t' '\n' '\r' ]      {token lexbuf}
(* Data declarations *)
| "(*"         { comment lexbuf }
| "int"        { INT }
| "string"      { STRING }
| "char"        { CHAR }
| "float"       { FLOAT }
| "bool"        { BOOL }
(* Keywords *)
| "let"        { LET }
| "if"         { IF }
| "then"       { THEN }
| "else"       { ELSE }
| "fun"         { FUN }
| "in"          { IN    }
| "match"      { MATCH }
| "with"       { WITH }
| "rec"        { REC }
(* Int Operators *)
| '+'          { PLUS }
| '-'          { MINUS }
| '*'          { MULT }
| '/'          { DIV  }
| '%'          { MOD }
| "=="         { EQ }
| "!="         { NEQ }
| "&&"         { LOG_AND}
| "||"         { OR }
| "not"        { NOT }
| '>'          { GREATER }
| '<'          { LESS }
| "<="         { LEQ }
| ">="         { GEQ }
(* Other symbols *)
| '_'          { WILDCARD }
| '='          { ASSIGN }
| '('          { LPAREN }
| ')'          { RPAREN }
```

```
| '['           { LBRACKET }
| ']'           { RBRACKET }
| ','            { COMMA }
| ":"           { COLON }
| "::"          { COLONS }
| ';'            { SEMICOLON }
| "->"          { RARROW }
| "|"           { BAR }
(* Literals and identifiers *)
| digit+ as int_lit { INT_LITERAL(int_of_string int_lit)}
| digit+ '.' digit* as float_lit { FLOAT_LITERAL(float_of_string float_lit) }
| '\'' (digit | letter | '_') '\'' as char_lit { CHAR_LITERAL(int_of_string char_lit) }
| '"' (('\\' '"'| [^'"'])* as str) '"' { STRING_LITERAL(unescape str) }
| ("true" | "false") as bool_lit { BOOL_LITERAL(bool_of_string bool_lit)}
| letter (letter | digit | '_')* as lxm { ID(lxm) }
| eof          { EOF }

and comment = parse
       | "*)" { token lexbuf }
       | _    { comment lexbuf }
```

B.2 parser.mly

```
/* Ocamlyacc parser for tail */
%{ open Ast %}

/*keywords*/
%token LET FUNC REC FUN ITER
%token IN
%token IF THEN ELSE
%token MATCH WITH

/*key symbols*/
%token PLUS MINUS MULT DIV MOD EQ NEQ LOG_AND OR NOT LESS GREATER GEQ LEQ
%token FPLUS FMINUS FMULT FDIV FLESS FLEQ FGREATER FGEQ
%token EOF
%token ASSIGN COMMA SEMIS SEMICOLON COLON COLONS
%token LPAREN RPAREN LBRACKET RBRACKET
%token RARROW BAR
%token WILDCARD

/*data types, literals, id*/
```

```
%token <int> INT_LITERAL
%token <int> CHAR_LITERAL
%token <float> FLOAT_LITERAL
%token <string> STRING_LITERAL
%token <bool> BOOL_LITERAL
%token INT STRING CHAR FLOAT BOOL
%token <string> ID

%nonassoc ELSE
%right BAR
%nonassoc RARROW
%right IN
%nonassoc SEMICOLON
%right ASSIGN
%left LOG_AND OR
%left EQ NEQ
%left GEQ LEQ LESS GREATER
%left PLUS MINUS
%left MULT DIV MOD
%right NOT

%start program
%type <Ast.program> program

%%

program:
     stmt_list EOF { Program(List.rev $1) }

stmt_list:
     stmt { [$1] }
     | stmt_list stmt { $2::$1 }

stmt:
     fun_def SEMICOLON              { Fun_Def_Stmt($1) }
     | expr_def SEMICOLON    { Ass_Stmt($1) }
     | expr SEMICOLON        { Expr_Stmt($1)       }

type_decl:
     primitive { $1 }

primitive:
     INT               { Int_Decl }
```

```
      | FLOAT           { Float_Decl }
      | CHAR            { Char_Decl }
      | STRING          { String_Decl }
      | BOOL            { Bool_Decl }

expr:
     exprs            { $1 }
     | defs           { $1 }
     | conds          { $1 }
     | match_expr     { $1 }

exprs:
     literal                        { Literal($1) }
     | ID                           { Id($1) }
     | LPAREN expr RPAREN           { Expr($2) }
     | expr PLUS expr               { Binop($1, Add, $3) }
     | expr MINUS expr              { Binop($1, Minus, $3) }
     | expr MULT expr               { Binop($1, Mult, $3) }
     | expr DIV expr                { Binop($1, Div, $3) }
     | expr MOD expr                { Binop($1, Mod, $3) }
     | expr EQ expr                 { Binop($1, Eq, $3) }
     | expr NEQ expr                { Binop($1, Neq, $3) }
     | expr GEQ expr                { Binop($1, Geq, $3) }
     | expr LEQ expr                { Binop($1, Leq, $3) }
     | expr LESS expr               { Binop($1, Less, $3) }
     | expr GREATER expr            { Binop($1, Greater, $3) }
     | expr LOG_AND expr            { Binop($1, Log_And, $3) }
     | expr OR expr                 { Binop($1, Or, $3) }
     | NOT expr                     { Uniop($2, Not) }
     | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
     /*| items_opt                  { $1 }*/

conds:
     | IF expr THEN expr ELSE expr  { If($2, $4, $6) }

defs:
     | fun_def IN expr                    { Evaluate($1, $3)}
     | expr_def IN expr                   { Ass($1, $3) }

fun_decl:
       | type_decl ID LPAREN formals_opt RPAREN
           { {return_typ = $1;
              func_typ = Iter;
```

```
                func_name = $2;
                formals = $4 } }
        | type_decl REC ID LPAREN formals_opt RPAREN
            { {return_typ = $1;
                func_typ = Rec;
                func_name = $3;
                formals = $5 } }
fun_def:
        | LET fun_decl ASSIGN FUN expr_list SEMICOLON      {Fun_Def($2, $5)}

expr_def:
     | LET type_decl ID ASSIGN expr  { Expr_Def($2, $3, $5) }
     | LET type_decl LBRACKET RBRACKET ID ASSIGN list_substance {List_Def($2, $5, $7) }

list_substance:
     | LBRACKET items_opt RBRACKET { NewList($2) }
     | ID COLONS ID             { Prepend($1, $3) }

items_opt:
     /*empty*/  {[]}
     | item_list { List.rev $1 }

item_list:
     expr                           { [$1] }
     | item_list SEMICOLON expr     { $3::$1 }

actuals_opt:
        {[]}
        | actuals_list { List.rev $1 }

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

expr_list:
     expr                           { [$1] }
     | expr_list SEMICOLON expr     { $3::$1 }

literal:
     INT_LITERAL              { Int_Lit($1)   }
     | FLOAT_LITERAL     { Float_Lit($1) }
     | CHAR_LITERAL      { Char_Lit($1)  }
     | STRING_LITERAL    { String_Lit($1)}
```

```
    | BOOL_LITERAL      { Bool_Lit($1)  }
    | LPAREN RPAREN     { Unit        }
    | LBRACKET RBRACKET { Unit        }

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

formal_list:
    formal_arg              { [$1] }
    | formal_list COMMA formal_arg { $3::$1 }

formal_arg:
    type_decl ID     { Formal($1, $2) }

match_expr:
    MATCH expr WITH match_list { Match ($2, $4) }

match_list:
    wildcard                    { [$1] }
    | match_case BAR match_list     { $1::$3 }

wildcard:
    WILDCARD RARROW expr    { MatchCase(Literal(String_Lit("_")), $3) }

match_case:
    exprs RARROW expr { MatchCase($1, $3) }
```

B.3 ast.ml

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

type primitive = Int_Decl | Float_Decl | Char_Decl | String_Decl | Bool_Decl | Int_List |
Float_List | Char_List | String_List | Bool_List | Unit
type op = Add | Minus | Mult | Div | Mod | Eq | Neq | Geq | Leq | Less | Greater | Log_And |
Or | FAdd | FMinus | FMult | FDiv | FGeq | FLeq | FLess | FGreater
type uniop = Not
type func_type = Iter | Rec
(* type boolean = true | false *)

type formal = Formal of primitive * string
```

```
type fun_decl = {
        return_typ : primitive;
        func_typ   : func_type;
        func_name  : string;
        formals    : formal list;
}

and expr =
    | Literal of literal
    | Expr of expr
    | Id of string
    | Binop of expr * op * expr
    | Uniop of expr * uniop
    | Call of string * expr list
    | If of expr * expr * expr
    | Evaluate of fun_def * expr
    | Ass of expr_def * expr
    | Match of expr * match_case list

and literal =
    | Int_Lit of int
    | Float_Lit of float
    | Char_Lit of int
    | String_Lit of string
    | Bool_Lit of bool
    | Unit

and fun_def = Fun_Def of fun_decl * expr list

and expr_def = Expr_Def of primitive * string * expr
               | List_Def of primitive * string * list_substance

and list_substance =
    NewList of expr list
    | Prepend of string * string

and match_case = MatchCase of expr * expr

and stmt =
        | Fun_Def_Stmt of fun_def
        | Ass_Stmt of expr_def
        | Expr_Stmt of expr
```

```
type program = Program of stmt list

(* Pretty-printing functions *)
let string_of_primitive = function
        | Int_Decl -> "int"
        | Float_Decl -> "float"
        | Char_Decl -> "char"
        | String_Decl -> "string"
        | Bool_Decl -> "bool"
        | _ -> "list"

let string_of_op = function
        | Add -> "+"
        | Minus -> "-"
        | Mult -> "*"
        | Div -> "/"
        | Mod -> "%"
        | Eq -> "=="
        | Neq -> "!="
        | Geq -> ">="
        | Leq -> "<="
        | Less -> "<"
        | Greater -> ">"
        | Log_And -> "&&"
        | Or -> "||"
        | _ -> "error string_of_op"

let string_of_uniop = function
        | Not -> "!"

let string_of_func_type = function
        | Iter -> ""
        | Rec -> "rec"

let string_of_formal = function
        | Formal(p, s) -> string_of_primitive(p)^" "^s

let string_of_formals_list fl = List.fold_left (fun s f -> s^(string_of_formal f)^", ") "" fl

let string_of_fun_decl fd =
        (string_of_primitive fd.return_typ)^" "^(string_of_func_type fd.func_typ)^"
"^fd.func_name^"("^(string_of_formals_list fd.formals)^")"
```

```
let rec string_of_expr = function
    Literal(l) -> string_of_lit l
    | Id(i) -> i
    | Binop(e1, o, e2) -> (string_of_expr e1)^(string_of_op o)^(string_of_expr e2)
    | Uniop(e, u) -> (string_of_uniop u)^(string_of_expr e)
    | Match(e, ml) -> "Match("^(string_of_expr e)^","^(string_of_match_list ml)^")"
    | Call(s, el) -> "Call("^s^"("^(string_of_expr_list el)^")"^")"
    | If(e1, e2, e3) -> "if("^(string_of_expr e1)^") then "^(string_of_expr e2)^" else
"^string_of_expr e3
    | Evaluate(fundef, exp) -> "Evaluate("^(string_of_fun_def fundef)^" \n\tin
"^(string_of_expr exp)^")"
    | Ass(ed, e) -> "Ass("^(string_of_expr_def ed)^","^(string_of_expr e)^")"
    | _ -> "hey"

and string_of_lit = function
    Int_Lit(i) -> string_of_int i
    | Float_Lit(f) -> string_of_float f
    | Char_Lit(c) -> string_of_int c
    | String_Lit(s) -> s
    | Bool_Lit(b) -> string_of_bool b
    | Unit -> "()"

and string_of_fun_def = function
       | Fun_Def(fd, el) -> (string_of_fun_decl fd)^" = fun\n\t"^(string_of_expr_list el)

and string_of_match_list ml = List.fold_left (fun str form -> str^(string_of_match_case
form)^";") "" ml

and string_of_match_case = function
       | MatchCase(e1, e2) -> "MatchCase("^(string_of_expr e1)^","^(string_of_expr e2)^")"

and string_of_expr_def = function
       | Expr_Def(p, s, e) -> "Expr_Def("^(string_of_primitive p)^" "^s^" = "^(string_of_expr
e)^")"
       | _ -> "yo"

and string_of_stmt = function
       | Fun_Def_Stmt(fd) -> "Fun_Def_Stmt(\n\t"^(string_of_fun_def fd)^"\n)"
       | Ass_Stmt(ed) -> "Ass_Stmt(\n\t"^(string_of_expr_def ed)^"\n)"
       | Expr_Stmt(e) -> "Expr_Stmt(\n\t"^(string_of_expr e)^"\n)"

and string_of_expr_list el = List.fold_left (fun str form -> str^(string_of_expr
```

```
form)^";\n\t") "" el

let string_of_program sl = match sl with
        Program(stmt_list) ->
        let result = List.fold_left (fun str form -> str^(string_of_stmt form)^";\n") ""
stmt_list
        in result^"\n"
```

B.4 analyzer.ml

```
open Ast
open Sast

module StringMap = Map.Make(String)

let rec expr_to_sexpr expr vars = match expr with
        (* exprs *)
        Ast.Literal(i) -> check_literal i vars
        | Ast.Id(s) -> vars, Sast.SId(s, get_id_type s vars)
        | Ast.Expr(e) -> expr_to_sexpr e vars
        | Ast.Binop(e1, op, e2) -> check_binop e1 op e2 vars
        | Ast.Uniop(e, op) -> check_uniop e op vars
        | Ast.Call(s, e) -> check_call s e vars
        (* defs *)
        | Ast.Evaluate(fd, e) -> check_eval fd e vars
        | Ast.Ass(ed, e) -> check_assign ed e vars
        (* conds *)
        | Ast.If(e1, e2, e3) -> check_if e1 e2 e3 vars
        (* match_expr *)
        | Ast.Match(e, ml) -> check_match e ml vars

and check_prepend s1 s2 vars =
        let d1 = get_id_type s1 vars in
        let d2 = get_id_type s2 vars in
        if (get_type_equivalence d1 d2) then
                SPrepend(s1,s2)
        else raise (Exceptions.InvalidPrepend "Must preprend lists of same type")

and check_match e ml vars =
        let vars', s = expr_to_sexpr e vars in
        let d = get_type_from_sexpr s in
        matchl_to_smatchl s d ml vars'
```

```
and matchl_to_smatchl s d ml vars =
	let rec helper vars smatchl = function
		| [] -> vars, Sast.SMatch(s,d, smatchl)
		| mat :: tl -> let vars', smatch = match_to_smatch d mat vars in
			helper vars' (smatch :: smatchl) tl
	in helper vars [] ml
(*
	let rec create_vars_list vl c =
		if c >= (List.length ml) then vl
		else create_vars_list (List.append vl [vars]) (c+1)
	in
	let vars_list = create_vars_list [] 0 in
	let sml = List.map2 (match_to_smatch d) ml vars_list in
	Sast.SMatch(s,d,sml)
*)

and match_to_smatch d ml vars = match ml with
	MatchCase(e1,e2) ->
		let vars1, s1 = expr_to_sexpr e1 vars in
		let vars2, s2 = expr_to_sexpr e2 vars1 in
		let d1 = get_type_from_sexpr s1 in
		let d2 = get_type_from_sexpr s2 in
		if (get_type_equivalence d d1) then vars2, SMatchCase(s1,d1,s2,d2)
		else raise (Exceptions.InvalidMatch "Must match between matching types")

and check_assign ed e1 vars = match ed with
	Ast.Expr_Def(t, s, e2) ->
		let d = Datatype(t) in
		let vars' = StringMap.add s d vars in
		let vars1, s1 = expr_to_sexpr e1 vars' in
		let t1 = get_type_from_sexpr s1 in
		vars', Sast.SAss((check_expr_assign t s e2 vars'),s1,t1)
	| Ast.List_Def(t, s, l) -> match l with
		NewList(nl) ->
			let vars1, s1 = expr_to_sexpr e1 vars in
			let t1 = get_type_from_sexpr s1 in
			vars1, Sast.SAss((check_list_assign t s nl vars), s1, t1)
		| Prepend(s1,s2) -> raise (Exceptions.InvalidBinopExpr "i")

and check_list_assign t s l vars =
	let d1 = get_list_dt t in
	let vars1 = StringMap.add s d1 vars in
```

```
        let vars', sl = expr_list_to_sexpr_list l vars in
        let d2 = get_list_type sl in
        get_list_assign_type s sl vars1 (d1,d2)

and get_list_dt t = match t with
        Int_Decl|Int_List -> Datatype(Int_List)
        | Float_Decl|Float_List -> Datatype(Float_List)
        | Char_Decl|Char_List -> Datatype(Char_List)
        | String_Decl|String_List -> Datatype(String_List)
        | Bool_Decl|Bool_List -> Datatype(Bool_List)
        | Unit -> raise (Exceptions.IllegalListType "Cannot have a void list")

and get_type_equivalence t1 t2 = match (t1,t2) with
        (Datatype(Int_Decl),Datatype(Int_Decl)) -> true
        | (Datatype(Float_Decl),Datatype(Float_Decl)) -> true
        | (Datatype(Char_Decl),Datatype(Char_Decl)) -> true
        | (Datatype(String_Decl),Datatype(String_Decl)) -> true
        | (Datatype(Bool_Decl),Datatype(Bool_Decl)) -> true
        | (Special(Wildcard("_")), _) -> true
        | (_, Special(Wildcard("_"))) -> true
        | _ -> false


and get_list_type sl =
        let hed = (List.hd sl) in
        let typ = get_type_from_sexpr hed in
        let type_check s =
                let t = get_type_from_sexpr s in
                if (get_type_equivalence typ t) then typ
                else raise (Exceptions.InvalidAssignment "Assignments must be made between
matching types")
        in
        List.hd (List.map type_check sl)


and get_list_assign_type s sl vars = function
        (Datatype(Int_Decl),Datatype(Int_Decl)) ->
Sast.SList_Def(Datatype(Int_Decl),s,SNewList(sl))
        | (Datatype(Float_Decl),Datatype(Float_Decl)) ->
Sast.SList_Def(Datatype(Float_Decl),s,SNewList(sl))
        | (Datatype(Char_Decl),Datatype(Char_Decl)) ->
Sast.SList_Def(Datatype(Char_Decl),s,SNewList(sl))
        | (Datatype(String_Decl),Datatype(String_Decl)) ->
```

```
Sast.SList_Def(Datatype(String_Decl),s,SNewList(sl))
	| (Datatype(Bool_Decl),Datatype(Bool_Decl)) ->
Sast.SList_Def(Datatype(Bool_Decl),s,SNewList(sl))
	| _ -> raise(Exceptions.InvalidAssignment "Assigments must be made between matching
types")

and check_expr_assign t s e vars =
	let d1 = Sast.Datatype(t) in
	let vars', se = expr_to_sexpr e vars in
	let d2 = get_type_from_sexpr se in
	get_assign_type s se vars' (d1, d2)

and get_assign_type s se vars = function
	(Datatype(Int_Decl),Datatype(Int_Decl)) ->  Sast.SExpr_Def(Datatype(Int_Decl),s,se)
	| (Datatype(Float_Decl),Datatype(Float_Decl)) ->
Sast.SExpr_Def(Datatype(Float_Decl),s,se)
	| (Datatype(Char_Decl),Datatype(Char_Decl)) ->
Sast.SExpr_Def(Datatype(Char_Decl),s,se)
	| (Datatype(String_Decl),Datatype(String_Decl)) ->
Sast.SExpr_Def(Datatype(String_Decl),s,se)
	| (Datatype(Bool_Decl),Datatype(Bool_Decl)) ->
Sast.SExpr_Def(Datatype(Bool_Decl),s,se)
	| _ -> raise(Exceptions.InvalidAssignment "Assigments must be made between matching
types")

and check_eval fd e vars =
	let vars1, sfd = fd_to_sfd fd vars in
	let vars', s = expr_to_sexpr e vars1 in
	vars', Sast.SEvaluate(sfd, s)

and check_fun_rt f_dec el vars =
	let ftyp = Datatype(f_dec.Ast.return_typ) in
	let rev_el = List.rev el in
	let final_e = (List.hd rev_el) in
	let vars', s = expr_to_sexpr final_e vars in
	let typ = get_type_from_sexpr s in
	if (get_type_equivalence ftyp typ) then ()
	else raise (Exceptions.IncorrectReturn "Function return type does not match type
returned")


and fd_to_sfd fd vars = match fd with
	Ast.Fun_Def(f_dec, el) ->
```

```
        let vars', vars_fun, sf_dec = f_dec_to_sf_dec f_dec vars in
        match f_dec.Ast.func_typ with
                Rec -> let vars2, sel = expr_list_to_sexpr_list el vars_fun in
                        vars', Sast.SFun_Def(sf_dec, sel)
                | Iter -> check_fun_rt f_dec el vars_fun;
                let vars3, sel = expr_list_to_sexpr_list el vars_fun in
                                vars', Sast.SFun_Def(sf_dec, sel)

and f_dec_to_sf_dec f_dec vars =
        let fname = f_dec.Ast.func_name in
        let rtype = Sast.Datatype(f_dec.Ast.return_typ) in
        let vars' = StringMap.add  fname rtype vars in
        let sf_dec =
        {
                sreturn_typ = Sast.Datatype(f_dec.Ast.return_typ);
                sfunc_typ = f_dec.Ast.func_typ;
                sfunc_name = f_dec.Ast.func_name;
                sformals = (formals_to_sformals f_dec.Ast.formals);
        } in
        let vars_fun = add_parameters vars' sf_dec.sfunc_typ  fname sf_dec.sformals in
        vars', vars_fun, sf_dec

and add_parameters vars fun_typ fname sformals =
        let rec helper_rec var_fun = function
                | [] -> var_fun
                | SFormal(d,s) :: tl ->
                let vars' = StringMap.add s d var_fun
                in helper_rec vars' tl
        in helper_rec vars sformals

and formals_to_sformals formals =
        List.map formal_to_sformal formals

and formal_to_sformal = function
        Ast.Formal(p,s) -> Sast.SFormal(Sast.Datatype(p), s)

and check_if e1 e2 e3 vars =
        let vars1', s1 = expr_to_sexpr e1 vars in
        let vars2', s2 = expr_to_sexpr e2 vars1' in
        let vars3', s3 = expr_to_sexpr e3 vars1' in
        let d = check_if_bool s1 (*get_type_from_sexpr s1*) in
        get_if_type s1 s2 s3 d vars1'
```

```
and get_if_type s1 s2 s3 d vars = match d with
	Datatype(Bool_Decl) -> vars, Sast.SIf(s1, s2, s3)
	| _ -> raise (Exceptions.InvalidIfConditional "If conditional can only operate on a
boolean datatype 3!.")

and expr_list_to_sexpr_list el vars =
	let rec helper vars sexprs n_el = match n_el with
		| [] -> vars, sexprs
		| expr :: tl -> let vars', sexpr = expr_to_sexpr expr vars in
			helper vars' (sexpr :: sexprs) tl
	in helper vars [] el

and check_literal literal vars = match literal with
	Ast.Int_Lit i -> vars, Sast.SLiteral(SInt_Lit(i))
	| Ast.Float_Lit f -> vars, Sast.SLiteral(SFloat_Lit(f))
	| Ast.String_Lit s -> vars, Sast.SLiteral(SString_Lit(s))
	| Ast.Char_Lit c -> vars, Sast.SLiteral(SChar_Lit(c))
	| Ast.Bool_Lit b -> vars, Sast.SLiteral(SBool_Lit(b))
	| Ast.Unit -> vars, Sast.SLiteral(SUnit)

and check_call s e vars =
	let vars', sexprs = expr_list_to_sexpr_list e vars in
	let d = get_id_type s vars in
	let d1 = match s with
	| "return" -> get_type_from_sexpr (List.hd sexprs)
	| _ -> d
	in
	vars', Sast.SCall(s, sexprs, d1, (List.length sexprs))

and get_id_type s vars = match s with
	"print_int"|"print_char"|"print_float"|"print_string"|"print_bool"
 -> Datatype(Unit)
	| "return" -> Datatype(Unit)
	| _ -> try StringMap.find s vars
	with | Not_found -> raise (Exceptions.VariableNotFound (s^" variable could not be
found"))

and get_type_from_sexpr sexpr = match sexpr with
	Sast.SLiteral(SInt_Lit(_)) -> Sast.Datatype(Int_Decl)
	| Sast.SLiteral(SFloat_Lit(_) )-> Sast.Datatype(Float_Decl)
	| Sast.SLiteral(SChar_Lit(_)) -> Sast.Datatype(Char_Decl)
	| Sast.SLiteral(SString_Lit("_")) -> Sast.Special(Wildcard("_"))
	| Sast.SLiteral(SString_Lit(_)) -> Sast.Datatype(String_Decl)
```

```
        | Sast.SLiteral(SBool_Lit(_)) ->    Sast.Datatype(Bool_Decl)
        | Sast.SLiteral(SUnit) -> Sast.Datatype(Unit)
        | Sast.SId(_,d) -> d
        | Sast.SBinop(_,_,_,d) -> d
        | Sast.SUniop(_,_,d) -> d
        | Sast.SCall(_,_,d,_) ->d
        | Sast.SEvaluate(SFun_Def(sfd,_),_) -> sfd.sreturn_typ
        | Sast.SAss(_,_,d) -> d
        | Sast.SMatch(_,d,_) -> d
        | Sast.SIf(e,_,_) -> check_if_bool e

and check_if_bool e = match e with
        |Sast.SLiteral(SBool_Lit(_)) -> Sast.Datatype(Bool_Decl)
        |Sast.SBinop(_,op,_,_) -> check_binop_bool op
        | Sast.SUniop(_,op,_) -> Sast.Datatype(Bool_Decl)
        | _ -> raise (Exceptions.InvalidIfConditional "If conditional can only operate on a
boolean datatype 2!.")

and check_binop_bool = function
        Eq | Neq | Geq | Leq | Less | Greater | Log_And | Or -> Sast.Datatype(Bool_Decl)
        | _ -> raise (Exceptions.InvalidIfConditional "If conditional can only operate on a
boolean datatype 1!.")


and check_uniop e op vars =
        let vars', sexpr = expr_to_sexpr e vars in
        let t = get_type_from_sexpr sexpr in
         match op with
        Ast.Not -> get_uniop_type sexpr op vars' t

and get_uniop_type s o vars = function
        Datatype(Bool_Decl) -> vars, Sast.SUniop(s, o, Datatype(Bool_Decl))
        | _ -> raise (Exceptions.InvalidBinopExpr "Wrong uniop")

and check_binop e1 op e2 vars =
        let vars1, sexpr1 = expr_to_sexpr e1 vars in
        let vars2, sexpr2 = expr_to_sexpr e2 vars in
        let t1 = get_type_from_sexpr sexpr1 in
        let t2 = get_type_from_sexpr sexpr2 in
        match op with
        Add | Minus | Mult | Div | Mod -> get_math_type sexpr1 op sexpr2 vars (t1, t2)
        | Eq | Neq | Geq | Leq | Less | Greater -> get_comparison_type sexpr1 op sexpr2 vars
(t1, t2)
```

```
        | Log_And | Or -> get_logical_type sexpr1 op sexpr2 vars (t1, t2)
        | _ -> raise (Exceptions.InvalidBinopExpr "binop not supported")

and get_math_type sexpr1 op sexpr2 vars = function
        (Datatype(Int_Decl), Datatype(Int_Decl)) -> vars, Sast.SBinop(sexpr1, op, sexpr2,
Datatype(Int_Decl))
        | (Datatype(Float_Decl), Datatype(Float_Decl)) -> vars,
Sast.SBinop(sexpr1,op,sexpr2,Datatype(Float_Decl))
        | _ -> raise (Exceptions.InvalidBinopExpr "Cannot perform math operations on different
datatypes")

and get_comparison_type sexpr1 op sexpr2 vars = function
        (Datatype(Int_Decl), Datatype(Int_Decl)) -> vars, Sast.SBinop(sexpr1, op, sexpr2,
Datatype(Int_Decl))
        | (Datatype(Float_Decl), Datatype(Float_Decl)) -> vars,
Sast.SBinop(sexpr1,op,sexpr2,Datatype(Float_Decl))
        | _ -> raise(Exceptions.InvalidBinopExpr "Cannot peform comparison on different
datatypes")

and get_logical_type sexpr1 op sexpr2 vars = function
        (Datatype(Bool_Decl), Datatype(Bool_Decl)) -> vars,
Sast.SBinop(sexpr1,op,sexpr2,Datatype(Bool_Decl))
        | (_,_) -> raise(Exceptions.InvalidBinopExpr "type mismatch")

and check_expr_def t s e vars =
        let d1 = Datatype(t) in
        let vars1 = StringMap.add s d1 vars in
        let vars2, sexpr = expr_to_sexpr e vars1 in
        let d2 = get_type_from_sexpr sexpr in
        if (get_type_equivalence d1 d2) then vars2, Sast.SExpr_Def(d1,s,sexpr)
        else raise  (Exceptions.InvalidAssignment "Variable assignment attempted between
mismatched types")

and check_list_def t s el vars =
        let d1 = Datatype(t) in
        let vars1 = StringMap.add s d1  vars in match el with
        NewList(el) ->
                let vars', sl = expr_list_to_sexpr_list el vars in
                let d2 = get_list_type sl in
                if (get_type_equivalence d1 d2) then vars1, Sast.SList_Def(d1,s,SNewList(sl))
                else raise  (Exceptions.InvalidAssignment "Variable list assignment attempted
between mismatched types")
        | Prepend(s1,s2) -> vars1, Sast.SList_Def(d1, s,  check_prepend s1 s2 vars)
```

```
and expr_def_to_sexpr_def e vars = match e with
       Ast.Expr_Def(t,s,ex) -> check_expr_def t s ex vars
       | Ast.List_Def(t,s,el) -> check_list_def t s el vars

let convert_stmt_to_sstmt (stmt:Ast.stmt) vars = match stmt with
       Ast.Fun_Def_Stmt(e) ->
              let vars', sfun_def = fd_to_sfd e vars
              in vars', Sast.SFun_Def_Stmt(sfun_def)
       | Ast.Ass_Stmt (e)  ->
              let vars', sexpr_def =  expr_def_to_sexpr_def e vars
              in vars', Sast.SAss_Stmt(sexpr_def)
       | Ast.Expr_Stmt(e)  ->
              let vars', sexpr = expr_to_sexpr e vars
              in vars', SExpr_Stmt(sexpr)

let convert_stmts_to_sstmts (stmt_list:Ast.stmt list)  =
       let vars_begin = StringMap.empty in
       let rec helper vars sstmts = function
              | [] -> List.rev sstmts
              | stmt :: tl ->
                     let vars', sstmt = convert_stmt_to_sstmt stmt vars in
                     helper vars' (sstmt :: sstmts) tl
       in helper vars_begin [] stmt_list
(*     let sstmt_list = List.map convert_stmt_to_sstmt stmt_list
       in
       sstmt_list
*)

let analyze program = match program with
    Ast.Program(stmt_list) ->
(*     let vars = StringMap.empty in*)
        let sast = convert_stmts_to_sstmts stmt_list in
       Sast.SProgram(sast)
```

B.5 sast.ml

```
open Ast

type wildcard = Wildcard of string

type datatype = Datatype of primitive
```

```
                | Special of wildcard

type sformal = SFormal of datatype * string

type sfun_decl = {
        sreturn_typ : datatype;
        sfunc_typ   : func_type;
        sfunc_name  : string;
        sformals    : sformal list;
}

and sexpr =
        | SLiteral of sliteral
        | SId of string * datatype
        | SBinop of sexpr * op * sexpr* datatype
        | SUniop of sexpr * uniop * datatype
        | SCall of string * sexpr list * datatype * int
        | SIf of sexpr * sexpr * sexpr
        | SEvaluate of sfun_def * sexpr
        | SAss of sexpr_def * sexpr * datatype
        | SMatch of sexpr * datatype * smatch_case list

and sliteral =
        | SInt_Lit of int
        | SFloat_Lit of float
        | SChar_Lit of int
        | SString_Lit of string
        | SBool_Lit of bool
        | SUnit

and sfun_def = SFun_Def of sfun_decl * sexpr list

and sexpr_def = SExpr_Def of datatype * string * sexpr
        | SList_Def of datatype * string * slist_substance

and slist_substance = SNewList of sexpr list
        | SPrepend of string * string

and smatch_case = SMatchCase of sexpr * datatype * sexpr * datatype

and sstmt =
        | SFun_Def_Stmt of sfun_def
        | SAss_Stmt of sexpr_def
```

```
        | SExpr_Stmt of sexpr

type sprogram = SProgram of sstmt list
```

B.6 robotsInDisguise.ml

```
open Ast
open Sast

module StringMap = Map.Make(String)

(* Record that stores a variable's datatype (Int_Decl, Char_Decl, etc),
name (given by the programmer), and value assigned *)
type var_struct = {
        data_type: datatype;
        name: string;
        value: sexpr
}

(* Record that stores a functions *)
type fun_struct = {
        return_typ : datatype; (* Return type of the function *)
        func_typ   : func_type; (* Iterative or recursive *)
        func_name  : string; (* Function's name *)
        formals    : sformal list; (* Function's formal parameters, given by programmer *)
        expr_list  : sexpr list; (* list of expressions defining a function *)
}

(* a "type template" that stores 1) a map of the program's global variables.
The hashmap maps the variable's name to its var_struct (see line 8 for var_struct)
2) stores a map of the program's functions. Maps the function's name to its
fun_struct (see line 15 for fun_struct). 3) stores a list of the expressions a
user chose to evaluate, but not assign*)
type environment = {
        vars: var_struct StringMap.t;
        funs: fun_struct StringMap.t;
        expressions: sexpr list;
}

(* "Instantiating" the type template environment *)
type real_environment = {
        vars: var_struct StringMap.t;
```

```
        vars_order: string list;
        funs: fun_struct StringMap.t;
        funs_order: string list;
        expressions: sexpr list;
}

let env = {
        vars = StringMap.empty;
        vars_order = [];
        funs = StringMap.empty;
        funs_order = [];
        expressions = [];
}


(* Takes two floats and determines what operation was applied to them.
See evalExpr (match on SBinop) for use *)
let evalBinopFloat f1 o f2 =
        match o with
        | Add -> SFloat_Lit(f1 +. f2)
        | Minus -> SFloat_Lit(f1 -. f2)
        | Mult -> SFloat_Lit(f1 *. f2)
        | Div -> SFloat_Lit(f1 /. f2)
        | Mod -> SFloat_Lit(mod_float f1 f2)
        | Eq -> SBool_Lit(f1 = f2)
        | Neq -> SBool_Lit(f1 <> f2)
        | Geq -> SBool_Lit(f1 >= f2)
        | Leq -> SBool_Lit(f1 <= f2)
        | Less -> SBool_Lit(f1 < f2)
        | Greater -> SBool_Lit(f1 > f2)
        | _ -> raise (Exceptions.OperationNotPermitted (" "^(string_of_float f1)^"
,"^(string_of_float f2)^" are floats. "^(string_of_op o)^" cannot be applied to floats."))

(* Takes two ints and determines what operation was applied to them. See evalExpr
(match on SBinop) for use *)
let evalBinopInt i1 o i2 =
        match o with
        | Add -> SInt_Lit(i1 + i2)
        | Minus -> SInt_Lit(i1 - i2)
        | Mult -> SInt_Lit(i1 * i2)
        | Div -> SInt_Lit(i1 / i2)
        | Mod -> SInt_Lit((mod) i1 i2)
        | Eq -> SBool_Lit(i1 = i2)
```

```
	| Neq -> SBool_Lit(i1 <> i2)
	| Geq -> SBool_Lit(i1 >= i2)
	| Leq -> SBool_Lit(i1 <= i2)
	| Less -> SBool_Lit(i1 < i2)
	| Greater -> SBool_Lit(i1 > i2)
	| _ -> raise (Exceptions.OperationNotPermitted (" "^(string_of_int i1)^"
,"^(string_of_int i2)^" are ints. "^(string_of_op o)^" cannot be applied to ints."))

(* Takes two bools and determins which operation was applied to them. See evalExpr
(match on SBinop) for use*)
let evalBinopBool b1 o b2 =
	match o with
	| Log_And -> SBool_Lit(b1 && b2)
	| Or -> SBool_Lit(b1 || b2)
	| _ -> raise (Exceptions.OperationNotPermitted ((string_of_op o)^" not a legal
uniop."))

(* Takes an SBinop node (for binary operation o applied to two arguments e1 and e2)
and determines what type the arguments are to figure out which operators to apply
to them. See evalExpr (match on SBinop) for use *)
let evalBinop e1 o e2 =
	match [e1; e2] with
	| [SInt_Lit(i1); SInt_Lit(i2)] -> evalBinopInt i1 o i2
	| [SFloat_Lit(f1); SFloat_Lit(f2)] -> evalBinopFloat f1 o f2
	| [SBool_Lit(b1); SBool_Lit(b2)] -> evalBinopBool b1 o b2
	| _ -> raise (Exceptions.TypeMismatch (" Type mismatch between arguments 1 and 2 for
operator "^(string_of_op o)))

let evalUniop b u =
	match b with
	| SBool_Lit(c) ->
		(match u with
		| Not -> SBool_Lit(not c))

	| _ -> raise (Exceptions.OperationNotPermitted " Uniop only permitted on bools.")


let rec evalExpr expression vars funs =
	match expression with
	| SLiteral(l) -> expression
	| SId(var_name, dtype) -> (resolveScope var_name dtype vars funs)
	| SBinop(e1, o, e2, dtype) -> SBinop((evalExpr e1 vars funs), o, (evalExpr e2 vars
funs), dtype)
```

```
        | SUniop(e, u, dtype) -> SUniop((evalExpr e vars funs),u, dtype)
        | SCall(funcname, sexprList, dtype, _) -> functionCall funcname sexprList vars funs
        | SIf(cond, texpr, fexpr) -> if ((evalExpr cond vars funs) = SLiteral(SBool_Lit(true)))
                                     then evalExpr texpr vars funs
                                     else evalExpr fexpr vars funs
        | SEvaluate(sfunDef, se) -> evalE sfunDef se vars funs
        | SAss(sexprDef, se, dtype) -> evalAss sexprDef se vars funs
         | SMatch(pattern, dtype, caseList) -> expression
         (*| SMatch(pattern, dtype, caseList) -> ignore(matchEval pattern dtype caseList vars
funs); expression*)
        | _ -> raise (Exceptions.DatatypeNotPermitted)

and matchEval pattern dtype caseList vars funs =
        let matchCase l case = match case with
                | SMatchCase(p, dtype1, e, dtype2) ->
                             (if ( (evalExpr pattern vars funs) = (evalExpr p vars funs))
                             then e::l
                             else l)
        in
        let cl = (List.rev caseList) in
        let evalMatches = List.fold_left matchCase [] cl in
         if ((List.length evalMatches) = 0) then raise (Exceptions.OperationNotPermitted "Don't
fucking do this.") else
        evalExpr (List.hd evalMatches) vars funs

and evalAss sexprDef se vars funs =
        match sexprDef with
        | SExpr_Def(declTyp, n, v) ->
                (let mymap = StringMap.add n
                        {       data_type = declTyp;
                                name = n;
                                value = evalExpr v vars funs; } vars
                        in (evalExpr se mymap funs))
        | _ -> raise (Exceptions.DatatypeNotPermitted)

and resolveScope id dtype vars funs=
        if (StringMap.mem id vars) then
                let v_struct = StringMap.find id vars in
                if (dtype = v_struct.data_type) then (evalExpr v_struct.value vars funs)
                else raise (Exceptions.TypeMismatch " declaration of datatype and stored don't
match")
        else raise (Exceptions.OperationNotPermitted (id^" not in scope or undefined."))
```

```
and evalExprList eList local_vars local_funs =
	let derp = List.fold_left (fun struc e -> let ex = evalExpr e struc.vars struc.funs in
				{ vars = struc.vars; vars_order = []; funs = struc.funs;
funs_order = []; expressions = ex::struc.expressions}  )
				{ vars = local_vars; vars_order = []; funs = local_funs;
funs_order = []; expressions = [] } eList
	in (List.hd (derp.expressions))


and evalE funDef e vars funs =
	match funDef with
	| SFun_Def(sfuncDecl, sexprList) -> let myFunMap = StringMap.add sfuncDecl.sfunc_name {
					return_typ = sfuncDecl.sreturn_typ;
					func_typ = sfuncDecl.sfunc_typ;
					func_name = sfuncDecl.sfunc_name;
					formals = sfuncDecl.sformals;
					expr_list = sexprList; } funs in (evalExpr e vars
myFunMap)

and functionCall funcname actualsList vars funs =
	if (StringMap.mem funcname funs) then
		let funny = StringMap.find funcname funs
		in
		let newfuns =
			if (funny.func_typ == Rec) then StringMap.add funny.func_name funny funs
else funs
		in
		let tempfun v a b = match b with
			| SFormal(p, id) -> StringMap.add id { data_type = p; name = id; value =
a; } v
		in
		let newVars = List.fold_left2 tempfun vars actualsList funny.formals in
		let s = evalExprList funny.expr_list newVars newfuns in s
	else raise (Exceptions.OperationNotPermitted ("\""^funcname^"\" function out of scope
or undefined"))

(* Main method for analyzer *)
and analyzer sprogram globalVars globalFuns globalEx v_order f_order=
	let evalProgram l s = match s with
		| SAss_Stmt(SExpr_Def(declTyp, n, v)) -> let n_globalVars = StringMap.add n {
				data_type = declTyp;
				name = n;
```

```
                        value = evalExpr v l.vars l.funs; } l.vars in
                let n_vorder = n::l.vars_order in
                {       vars = n_globalVars;
                        vars_order = n_vorder;
                        funs = l.funs;
                        funs_order = l.funs_order;
                        expressions = l.expressions; }
        | SFun_Def_Stmt(SFun_Def(sfuncDecl, sexprList)) ->
                let n_globalFuns = StringMap.add sfuncDecl.sfunc_name {
                        return_typ = sfuncDecl.sreturn_typ;
                        func_typ = sfuncDecl.sfunc_typ;
                        func_name = sfuncDecl.sfunc_name;
                        formals = sfuncDecl.sformals;
                        expr_list = sexprList; } l.funs
                in let n_forder = sfuncDecl.sfunc_name::l.funs_order
                in {    vars = l.vars;
                        vars_order = l.vars_order;
                        funs = n_globalFuns;
                        funs_order = n_forder;
                        expressions = l.expressions; }
        | SExpr_Stmt(sexpr) -> let n_globalEx = sexpr::l.expressions in
                {       vars = l.vars;
                        vars_order = l.vars_order;
                        funs = l.funs;
                        funs_order = l.funs_order;
                        expressions = n_globalEx; }
    in List.fold_left evalProgram { vars = globalVars;
                                    vars_order = v_order;
                                    funs = globalFuns;
                                    funs_order = f_order;
                                    expressions = globalEx; } sprogram

let analyze sprogram = match sprogram with
        Sast.SProgram(sp) -> analyzer sp env.vars env.funs env.expressions env.vars_order
env.funs_order
```

B.7 codegen.ml

```
open Llvm
open Ast
open Sast
```

```
open RobotsInDisguise
open Exceptions
module StringMap = Map.Make(String)


let translate env =

    (* Stuff for main & includes *)
    let include_stdio = "#include <stdio.h>\n\n" in
    let include_string = "#include <string.h>\n\n" in
    let includes = include_stdio ^ include_string in

    let main_str = "int main(){\n" in
    let main_end = "\n}\n" in

    (* Unwrap the vars and funcs from robot *)
    let vars = env.vars in
    let vars_order = List.rev env.vars_order in
    let funs = env.funs in
    let funs_order = List.rev env.funs_order in
    let ees = env.expressions in

    (* returns string of type *)
    let match_type dtype =
        match dtype with
        | Datatype(Int_Decl) -> "int"
        | Datatype(Float_Decl) -> "float"
        | Datatype(Bool_Decl) -> "int"
        | Datatype(Char_Decl) -> "char"
        | Datatype(String_Decl) -> "char*"
        | _ -> "void"
    in

    (* Converts literals to string representation *)
    let string_of_sliteral l = match l with
        | SInt_Lit(i) -> string_of_int i
        | SFloat_Lit(f) -> string_of_float f
        | SChar_Lit(i) -> string_of_int i
        | SString_Lit(s) -> "\"" ^ s ^ "\""
        | SBool_Lit(b) -> (if (b = true) then "1" else "0")
        | SUnit -> "void"
    in
```

```
    let string_of_uniop u = match u with
        | Not -> "!"
    in

    (* Function that recursively converts expressions into strings *)
    let rec string_of_expr expression v f =
       match expression with
       | SLiteral(l) -> string_of_sliteral l
       | SId(var_name, dtype) -> ignore (resolveScope var_name dtype v f); var_name
       | SBinop(e1, o, e2, dtype) -> "(" ^ (string_of_expr e1 v f) ^ ") "
                                ^ (string_of_op o) ^ " ("
                                ^ (string_of_expr e2 v f) ^") "
       | SUniop(e, u, dtype) -> (string_of_uniop u)^" ("^(string_of_expr e v f)^") "
       | SCall(funcname, sexprList, dtype, _) ->
             (match funcname with
                    | "print_int" -> "printf(\"%d\\n\", "^(string_of_actuals sexprList v
f)^")"
                    | "print_float" -> "printf(\"%f\\n\", "^(string_of_actuals sexprList v
f)^")"
                    | "print_bool" -> "printf(\"%d\\n\", "^(string_of_actuals sexprList v
f)^")"
                    | "print_char" -> "printf(\"%c\\n\", "^(string_of_actuals sexprList v
f)^")"
                    | "print_string" -> "printf(\"%s\\n\", "^(string_of_actuals sexprList v
f)^")"
                    | _ ->
                           ignore (functionCall funcname sexprList v f);
                           funcname ^ "(" ^(string_of_actuals sexprList v f) ^") ")
       | SIf(cond, texpr, fexpr) -> "if ("^(string_of_expr cond v f) ^ ") {\n"
                                    ^ (string_of_expr texpr v f) ^ ";\n}\nelse {\n"
                                    ^ (string_of_expr fexpr v f) ^";\n}"
       | SEvaluate(sfunDef, se) -> string_of_expr (evalE sfunDef se v f) v f
       | SAss(sexprDef, se, dtype) -> ignore(evalAss sexprDef se v f);
              (*match se with
              | SMatch(pattern, dtype, caseList) -> build_match_assign pattern se dtype
caseList v f
              | _ -> (string_of_sexpr_def sexprDef v f)^";\n"^(string_of_expr se (strmap
sexprDef v f) f)*)
              (string_of_sexpr_def sexprDef v f) ^ ";\n" ^ (string_of_expr se (strmap
sexprDef v f) f)
       | SMatch(pattern, dtype, caseList) -> build_match pattern dtype caseList v f
       | _ -> raise (Exceptions.OperationNotPermitted "Match not permitted")
```

```
    and string_of_sexpr_def sd v f=
        match sd with
        | SExpr_Def(dtype, name, se) -> (match_type dtype)^" "^name^" = "^(string_of_expr se v
f)
        | _ -> "You turd."

    and string_of_actuals sl v f =
        match sl with
        | [] -> ""
        | [s] -> string_of_expr s v f
        | s::l -> (string_of_actuals l v f)^", "^(string_of_expr s v f)

    and strmap s v f =
        match s with
        | SExpr_Def(dtype, n, se) -> StringMap.add n {
                data_type = dtype;
                name = n;
                value = (evalExpr se v f); } v
        | _ -> v

    and  build_match_num caseList v f =
        let rec helper str = function
            [SMatchCase(_,_,e,_)] -> e, str
            | SMatchCase(e1,d1,e2,d2) :: tl ->
                let str' = str ^ "case("
                               ^ (string_of_expr e1 v f) ^ ") :\n\t "
                               ^ (string_of_expr e2 v f) ^ ";\n\t break;\n"
            in helper str' tl
        in
       let cl = (List.rev caseList) in
        let final_e, case_str = helper "" cl in
        case_str ^ "\t default :\n\t" ^ (string_of_expr final_e v f) ^ ";\n"

    and build_match pattern dtype caseList v f =
        match dtype with
        | Datatype(Int_Decl) ->
            let switchstmt = "switch(" ^ (string_of_expr pattern v f) ^ "){\n\t" in
            let end_switch = "}\n" in
            switchstmt ^ (build_match_num caseList v f) ^ end_switch
        (*| Datatype(Float_Decl) | Datatype(Bool_Decl) -> *)
        (*| Datatype(Char_Decl) | Datatype(String_Decl) -> build_match_str pattern caseList v
f*)
        | _ -> raise (Exceptions.InvalidMatch "Cannot match on a void value.")
```

```
    and  build_match_num_assign variable caseList v f =
        let rec helper str = function
            [SMatchCase(_,_,e,_)] -> e, str
            | SMatchCase(e1,d1,e2,d2) :: tl ->
                let str' = str ^ "case("
                               ^ (string_of_expr e1 v f) ^ ") :\n\t "
                               ^ (string_of_expr variable v f) ^ " = "
                               ^ (string_of_expr e2 v f) ^ ";\n\t break;\n"
            in helper str' tl
        in
       let cl = (List.rev caseList) in
        let final_e, case_str = helper "" cl in
        case_str ^ "\t default :\n\t" ^ (string_of_expr final_e v f)

    and build_match_assign pattern variable dtype caseList v f =
        match dtype with
        | Datatype(Int_Decl) ->
            let switchstmt = "0;\n switch(" ^ (string_of_expr pattern v f) ^ "){\n\t" in
            let end_switch = "}\n" in
            switchstmt ^ (build_match_num_assign variable caseList v f) ^ end_switch
        (*| Datatype(Float_Decl) | Datatype(Bool_Decl) -> *)
        (*| Datatype(Char_Decl) | Datatype(String_Decl) -> build_match_str pattern caseList v
f*)
        | _ -> raise (Exceptions.InvalidMatch "Cannot match on a void value.")

    in



    (* Functions to build variable assign *)
    let match_value v =
       match v with
        | SLiteral(SInt_Lit(i)) -> string_of_int i
        | SLiteral(SFloat_Lit(f)) -> string_of_float f
        | SLiteral(SBool_Lit(b)) -> (if (b = true) then (string_of_int 1) else (string_of_int
0))
        | SLiteral(SChar_Lit(c)) -> string_of_int c
        | SLiteral(SString_Lit(s)) -> s
       | SBinop(e1,o,e2,dtype) -> ((string_of_expr e1 vars funs)^" "^(string_of_op o)^"
"^(string_of_expr e2 vars funs))
       | SUniop(se, u, dtype) -> (string_of_uniop u)^(string_of_expr se vars funs)
        | _ -> print_endline("shit");" "
    in
```

```
    (* Generate global vars *)
    let add_vars acc n =
       let v_s = StringMap.find n vars in
       let name = v_s.name in
       let typ = match_type v_s.data_type in
       let value = match_value v_s.value in
       acc^typ^" "^name^" = "^value ^";\n"
    in

    let global_vars = List.fold_left add_vars "" vars_order in

    (* Make formal parameters string *)
    let rec string_of_formals_list fl =
        match fl with
        | [] -> ""
        | [SFormal(dtype, name)] -> (match_type dtype)^" "^name
        | SFormal(dtype, name)::l -> (match_type dtype)^" "^name^", "^(string_of_formals_list
l)
    in

    (* Make expression strings *)
    let rec string_of_expr_list el formal_list =
       let rec addVars fl v = match fl with
            | [] -> v
            | [SFormal(p, n)] ->
                   StringMap.add n { data_type = p; name = n; value = SLiteral(SInt_Lit(1));
} v
            | SFormal(p, n)::l -> let blah = StringMap.add n {
                                    data_type = p;
                                    name = n;
                                    value = SLiteral(SInt_Lit(1));
                             } v in (addVars l blah)
       in
       let newvar_map = addVars formal_list vars in
            match el with
            | [] -> ""
            | [e] -> (match e with
                        | SAss(sexprDef, se, dtype) ->
                               ignore(evalAss sexprDef se newvar_map funs);
                               (string_of_sexpr_def sexprDef newvar_map funs) ^ ";\nreturn "
                               ^ (string_of_expr se (strmap sexprDef newvar_map funs) funs)
                        | SIf(cond, texpr, fexpr) -> "if (" ^ (string_of_expr cond newvar_map
```

```
funs) ^ ") {\n return "
                                          ^ (string_of_expr texpr newvar_map funs) ^
";\n}\nelse {\n return "
                                          ^ (string_of_expr fexpr newvar_map funs) ^ ";\n}"
                        | _ -> "return " ^ (string_of_expr e newvar_map funs)) ^ ";"
            | e::l -> (match e with
                      | SIf(cond, texpr, fexpr) -> (string_of_expr e newvar_map funs) ^ "\n"
                      | _ -> (string_of_expr e newvar_map funs) ^ ";\n" ^ (string_of_expr_list
l formal_list))
    in

    (* Make function declarations from list of fun structs *)
    let fun_list = List.fold_left (fun acc n -> let f_s = StringMap.find n funs in
                                          let ret_typ = match_type f_s.return_typ in
                                          let func_name = f_s.func_name in
                                          let formals = (string_of_formals_list f_s.formals)
in
                                          let expr_list = (string_of_expr_list f_s.expr_list
f_s.formals) in
                                          acc^"\n"^ret_typ^" "^func_name^" ("^formals^")
{\n"^expr_list^"\n}")
                                          "" funs_order
   in

    let s_ees = (string_of_expr_list (List.rev (SLiteral(SInt_Lit(0))::(ees))) []) in

    let the_string = includes ^ global_vars ^"\n"^ fun_list ^"\n"^ main_str ^ s_ees ^ main_end

in the_string
```

B.8 tail.ml

```
(*tail.ml

to run:
$ ./tail < test > test.ll
$ lli tail.ll

*)

open Ast
```

```
open Analyzer
open RobotsInDisguise

type action = Ast | Compile

let _ =

  let action = if Array.length Sys.argv > 1 then
       List.assoc Sys.argv.(1) [ ("-a", Ast);      (* Print the AST only *)
                        ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in

(*pipe program file into compiler's lexbuf*)
  let lexbuf = Lexing.from_channel stdin  in
  let ast = Parser.program Scanner.token lexbuf in
  match action with
       Ast -> print_string (Ast.string_of_program ast)
       | Compile -> let sast = Analyzer.analyze ast in
                    let envi = RobotsInDisguise.analyze sast in
                    let result = Codegen.translate envi
                      in print_string result
                    (*in print_string (Llvm.string_of_llmodule result)*)
(*let result =  Codegen.translate ast in print_string("hi")*)
  (*print_string (Llvm.string_of_llmodule result)*)
```

B.9 exceptions.ml

```
exception InvalidBinopExpr of string
exception InvalidIfConditional of string
exception InvalidAssignment of string
exception InvalidMatch of string
exception InvalidFunctionDec of string
exception InvalidPrepend of string
exception VariableNotFound of string
exception IncorrectReturn of string
exception IllegalListType of string

(*robotsInDisguise errors *)
exception OperationNotPermitted of string
exception TypeMismatch of string
```

```
exception DatatypeNotPermitted
exception FunctionUndefined of string
```

B.10 run-all.sh

```
#!/bin/sh

GCC="gcc"
TAIL="./tail"
ulimit -t 30
error=0
globalerror=0
keep=0
globallog=run-all.log
rm -f $globallog

Compile(){
	eval $1 -o $2 $3 || {
		SignalError $2 "compile"
		error=1
		return 1
	}
	if [ $error -eq 0 ] ; then
		$1 -o $2 $3
		echo "compile SUCCESS!"
	fi


}

CompileFail(){
	eval $1 -o $2 $3 && {
		SignalError $2 "compile"
		error=1
		return 1
	}
	if [ $error -eq 0 ] ; then
		echo "compile failed (which is good)"
	fi

}
```

```
SignalError(){
        echo "$1 $2 FAILED"
        error=1
        globalerror=1
        echo "  $1"
}

Run(){
        if [ $error -eq 0 ] ; then
                eval ./$1 || {
                        SignalError $1 "run"
                        error=1
                        return 1
                }
        fi
        if [ $error -eq 0]; then
                echo "run SUCCESS!"
        fi
        error=0
}

RunFail(){
        if [ $error -eq 0 ] ; then
                eval ./$1 && {
                        SignalError $1 "run"
                        error=1
                }
        fi
        error=0
}

FileIterator(){
basename=`echo $1 | sed 's/.*\\///
                                s/.tail//'`
        echo ""
        echo "########testing $basename"
        if [ $2 -eq 1 ] ; then
                $TAIL < $1 > tests/$basename.c
                Compile $GCC tests/$basename tests/$basename.c 2>> globallog
                Run tests/$basename $basename 2>> globallog
        fi

        if [ $2 -eq 0 ] ; then
```

```
                $TAIL < $1 > tests/$basename.c
                CompileFail $GCC tests/$basename tests/$basename.c 2>> globallog
                RunFail tests/$basename $basename 2>> globallog
        fi

}
failFiles="tests/fail-*.tail"
testFiles="tests/test-*.tail"
for tFile in $testFiles
do
        FileIterator $tFile 1
done

for fFile in $failFiles
do
        FileIterator $fFile 0
done

echo ""
echo "########TEST SUITE STATUS"

if [ $globalerror -eq 0 ] ; then
        echo "everything works as planned!"

fi

if [ $globalerror -ne 0 ] ; then
        echo "there are $globalerror tests that don't have expected"
fi
```

C. Test Files

C.1 Demo 1

C.1.1 `demo.tail`

```
print_string("comment -> expected output, nothing");
(* Here is a demo with a match *)

print_string("");
```

```
let int add(int q, int s) = fun
       let int y = q + s in y;
;

let int c = 2;


let int matchDemo (int b) = fun
       let int a = add(b, c) in
        match a with
                1 -> print_int(1)
                | 2 -> print_int(2)
                | _ -> print_string("wildcard");
       0;
;

print_string("matchDemo(0) -> expected output, 2");
matchDemo(0);

print_string("");
print_string("matchDemo(-1) -> expected output, 1");
matchDemo(-1);

print_string("");
print_string("matchDemo(4) -> expected output, wildcard");
matchDemo(4);

let int a = 4;
let int b = 2;

print_string("");
print_string("d = a/b where a is 4 and b is 2 -> expected output, 2");
let int d = a / b;
print_string("d = ");
print_int(d);
```

C.1.2 demo.c

```
#include <stdio.h>

#include <string.h>

int c = 2;
int a = 4;
```

```
int b = 2;
int d = 4 / 2;


int add (int q, int s) {
int y = (q) + (s) ;
return y;
}
int matchDemo (int b) {
int a = add(b, c) ;
switch(a){
       case(1) :
        printf("%d\n", 1);
        break;
case(2) :
        printf("%d\n", 2);
        break;
        default :
       printf("%s\n", "wildcard");
}
;
return 0;
}
int main(){
printf("%s\n", "comment -> expected output, nothing");
printf("%s\n", "");
printf("%s\n", "matchDemo(0) -> expected output, 2");
matchDemo(0) ;
printf("%s\n", "");
printf("%s\n", "matchDemo(-1) -> expected output, 1");
matchDemo(-1) ;
printf("%s\n", "");
printf("%s\n", "matchDemo(4) -> expected output, wildcard");
matchDemo(4) ;
printf("%s\n", "");
printf("%s\n", "d = a/b where a is 4 and b is 2 -> expected output, 2");
printf("%s\n", "d = ");
printf("%d\n", d);
return 0;
}
```

C.1.3 Output

```
comment -> expected output, nothing

matchDemo(0) -> expected output, 2
2

matchDemo(-1) -> expected output, 1
1
```

```
matchDemo(4) -> expected output, wildcard
wildcard

d = a/b where a is 4 and b is 2 -> expected output, 2
d =
2
```

C.2 Demo 2

C.2.1 demo2.tail

```
(* Hi, this is a funny demo *)

let int add(int q, int s) = fun
       let int y = q + s in y;
;

let int people = 4;
print_string("Number of people = 4");
let int sleep = 120;
print_string("Number of hours of sleep lost (conservative estimate) = 120");

let int total = add(people,sleep);
let int bugs = 20000;

print_string("Number of bugs = 20000");
print_string("When you add those all together and match on mix, you get: ");

let int matchDemo (int b) = fun
       let int mix = add(b, bugs) in
        match mix with
                1 -> print_int(1)
                | 20124 -> print_string("Output = tail")
                | _ -> print_string("wildcard");
       0;
;

matchDemo(total);

print_string("Number of things learned = uncountable");
```

```
(* matchDemo(-1); matchDemo(4); *)
```

C.2.2 demo2.c

```
#include <stdio.h>

#include <string.h>

int people = 4;
int sleep = 120;
int total = 120 + 4;
int bugs = 20000;


int add (int q, int s) {
int y = (q) + (s) ;
return y;
}
int matchDemo (int b) {
int mix = add(b, bugs) ;
switch(mix){
       case(1) :
        printf("%d\n", 1);
        break;
case(20124) :
        printf("%s\n", "Output = tail");
        break;
        default :
       printf("%s\n", "wildcard");
}
;
return 0;
}
int main(){
printf("%s\n", "Number of people = 4");
printf("%s\n", "Number of hours of sleep lost (conservative estimate) = 120");
printf("%s\n", "Number of bugs = 20000");
printf("%s\n", "When you add those all together and match on mix, you get: ");
matchDemo(total) ;
printf("%s\n", "Number of things learned = uncountable");
return 0;
}
```

C.2.3 Output

```
Number of people = 4
Number of hours of sleep lost (conservative estimate) = 120
```

```
Number of bugs = 20000
When you add those all together and match on mix, you get:
Output = tail
Number of things learned = uncountable
```

C.3 test-binops

C.3.1 test-binops.tail

```
let int a = 1;
let int b = 2;

let int c = a / b;
```

C.3.2 test-binops.c

```
#include <stdio.h>

#include <string.h>

int a = 1;
int b = 2;
int c = 1 / 2;



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

C.4 test-fun-in-expr

C.4.1 test-fun-in-expr.tail

```
let int add (int a, int b) = fun
       let int x = a + b in
       x;
in add(2,3);
```

C.4.2 test-fun-in-expr.c

```
#include <stdio.h>

#include <string.h>
```

```
int main(){
(3) + (2) ;
return 0;
}
```

C.5 test-if

C.5.1 test-if.tail

```
if (true) then print_int(1) else print_int(0);
```

C.5.2 test-if.c

```
#include <stdio.h>

#include <string.h>



int main(){
if (1) {
printf("%d\n", 1);
}
else {
printf("%d\n", 0);
}

}
```

C.6 test-match-in-func

C.6.1 test-match-in-func.tail

```
let int func (int b) = fun
        let int a = 1 in
        match a with
                1 -> print_int(1)
                | 2 -> print_int(2)
                | _ -> print_string("wildcard");
       0;
;
```

```
func(5);
```

C.6.2 test-match-in-func.c

```
#include <stdio.h>

#include <string.h>



int func (int b) {
int a = 1;
switch(a){
       case(1) :
        printf("%d\n", 1);
        break;
case(2) :
        printf("%d\n", 2);
        break;
        default :
       printf("%s\n", "wildcard");
}
;
return 0;
}
int main(){
func(5) ;
return 0;
}
```

C.7 test-match

C.7.1 test-match.tail

```
let int a = 1;
match a with
        1 -> print_int(1)
        | 2 -> print_int(2)
        | _ -> print_string("wildcard")
;
```

C.7.2 test-match.c

```
#include <stdio.h>

#include <string.h>

int a = 1;


int main(){
switch(a){
       case(1) :
        printf("%d\n", 1);
        break;
case(2) :
        printf("%d\n", 2);
        break;
        default :
       printf("%s\n", "wildcard");
}
;
return 0;
}
```

C.8 test-math

C.8.1 test-math.tail

```
let int a = 3;
let int b = 4;
let int c = a + b;
```

C.8.2 test-math.c

```
#include <stdio.h>

#include <string.h>

int a = 3;
int b = 4;
int c = 3 + 4;


int main(){
return 0;
```

```
}
```

C.9 test-printing

C.9.1 test-printing.tail

```
print_string("hello");
```

C.9.2 test-printing.c

```
#include <stdio.h>

#include <string.h>



int main(){
printf("%s\n", "hello");
return 0;
}
```

C.10 test-printing2

C.10.1 test-printing2.tail

```
print_int(4+4);
```

C.10.2 test-printing2.c

```
#include <stdio.h>

#include <string.h>



int main(){
printf("%d\n", (4) + (4) );
return 0;
}
```

C.11 fail-assign

C.11.1 fail-assign.tail

```
let int a = "hello";
let string b = 3;
```

C.12 fail-if

C.12.1 fail-if.tail

```
if "hello" then print_string("hello") else print_string("this should not work")
```

C.13 fail-list

C.13.1 fail-list.tail

```
let int [] a = [1;2;3];;
let int [] b = a;
```

C.14 fail-listassign

C.14.1 fail-listassign.tail

```
let int [] a = ["string"; "should"; "not"; "work];
```

C.15 fail-match-on-assign

C.15.1 fail-match-on-assign.tail

```
let int x = match 3 with
        5 -> 0
        | _ -> 4
in x;
```