Pumpkin

Joshua Boggs, Christopher Evans, Gabriela Melchior, Clement Robbins
jjb2175 - Testing, cme2126 - Language, gdm2118 - Manager, cjr2151 - Architecture

December 18, 2014

Contents

1 Introduction

Pumpkin is patchwork functional programming language. The Pumpkin programming language is a light-functional scripting language, which allows for coding flexibility and concise syntax. Pumpkin supports many syntactic optimizations for function nesting and chaining, such as pipes and partially applied functions. This language focuses on easily modeling a complex system of function calls.

1.1 Functional

Pumpkin follows functional programming principles: Functions are first class, they can be passed, returned and partially applied. We have both typed and untyped syntax, and allow for recursive functions. Anonymous functions are also allowed to compose functions on the fly, that capture a needed one time behavior but which doesn't need to be generalized.

1.2 Function Piping and Composition

Pumping allows one argument function calls to be piped together, this way resolving a long system of calls on one, easy to read, line.

Composition takes this idea one step further by nesting calls. Compose functions, create new functions out of nested ones, pipe arguments in: pumpkin made complicated calls easy to use and manipulate.

1.3 Concise

With type inference, piping and function composition we allow the programmer to write minimal code that accomplishes great functionality.

2 Language Tutorial

Here we present the fundamental building blocks of our language necessary to begin writing simple programs.

2.1 Variables

Declare variables with the keyword 'val':

```
val y : Bool = True
```

2.2 Functions

Declare functions with the keyword 'def':

```
def add(a: Int,  b: Int): Int => a + b
```

2.3 Comments

Create comments with \\or * *\:

```
\\ This is a comment

\* This is a
    multi-line comment *\
```

2.4 Piping

Pipe function arguments with '| >':

```
val x = [1,2,3] |> (a: List[Int] => len(a)%2)
if x is 0:
    print("Even")
else:
    print("Odd")
```

2.5 Function Composition

Create function compositions with '<<' or '>>':

```
val plusTwoTimesThree = (x:Int => x * 3) <<(x: Int => x + 2)
plusTwoTimesThree(4) # => 18
```

2.6 Type Inference

Our language includes type inference for variables and functions. The above declarations could be written concisely as such:

```
val y = True
def add(a, b) => a + b
```

2.7 Control Flow

Control flow is handled through if...else loops:

```
if(y is True):
    print("Y is True")
else:
    print("Y is not true")
```

2.8 Printing

Printing is handled with the 'print' keyword.

```
print("somestring")
print(variable)
```

2.9 Running Programs

Programs must be saved as '.pk' files. We include a makefile to compile our language, which must be done with 'make'. Then, our language can be compiled to Javascript by running the pumpking executable with the '-c' flag. Finally, you may use the platform of your choice to execute Javascript programs. We recommened Node, which is a popular and easy to install platform. A sample workflow is provided:

```
$ make
$ ./pkmn -c targetProgram.pk > targetProgram.js
$ node targetProgram.js
```

3 Language Manual

3.1 Types and Variables

3.1.1 Naming

Variable and function names must be a sequence of letters, digits and underscores. The first character must be a letter. By convention we will use CamelCase for names with multiple words.

3.1.2 Variables

A variable is a storage location paired with an associated name and type, which contains some value. All variables are statically typed and may be reassigned with a new primative or structure of a corresponding type. Variables can not be redeclared in any context.

Variable declarations and assignments, much like in C, are treated as expression. A variable, unlike C, cannot be declared as empty.

```
// Follows form val name = value, or name: Type = value.

// Legal declarations

val aNumber: Int = 5
aNumber = 10
aNumber = aNumber + 5

val anotherNumber = aNumber = 6

//Illegal

val aNumber = 10 // Error thrown on redeclaration of an already used variable

val emptyVar // Error thrown on empty variable
```

3.1.3 Native Types

- **Int**: a signed two's complement integer which has a minimum value of -2^{31} and a maximum value of 2^{31}-1.
- **Float**: a floating point value that can represent either very small fractions or numbers of great magnitude.

- **Char**: a single character.
- **String**: an immutable type that contains a sequence of characters.
- **Bool**: a boolean type whose only two possible values are True and False.
- **Unit**: represents an output of nothing, similar to unit in most functional languages.

3.1.4 Boolean Operations

Boolean variables are either True or False. They can be manipulated with logical expressions and operators.

Relational Expressions:
Boolean variable can be compared and combined with logic symbols to form expressions. Equality tests can be written with the keyword 'is' or with the symbol '==.' Other comparisons use the standard symbols used in Java:

```
1 < 2    // Less than           => True
3 > 4    // Greater than        => False
1 <= 3   // Less or equal to    => True
2 >= 2   // Greater or equal to => True
3 is 4   // Equality            => False
5 == 5   // Equality            => True
```

Logical Operators:
Pumpkin supports the three basic logic operators. Logical and can be written with the keyword 'and' or with the symbol '&&.' Logical or can be written with the keyword 'or' or with the symbol '||.' Negation can be written as the keyword 'not' or with the symbol '!.'

Examples:

```
False is False     // => True
True is False      // => False
True is not False  // => True

not True           // => False
!False             // => True
True and True      // => True
False && True      // => False
True or False      // => True
False || False     // => False

!True == False     // => True
not True == False  // => True
```

3.1.5 Derived Types (Tuples, Lists, & Maps)

N-tuples: Tuples are a basic immutable data structure that holds an ordered set of elements.

Unlike Lists or Maps, the elements with a tuple can be of any type and do not need to follow a consistent type declarati'on.

Tuples are most useful when a function may wish to return more than a single piece of information, or more commonly used as part of function nesting with 'pipe ins' and 'pipe outs'.

The following symbol scheme will be used to access consecutive elements of a tuple: `$0, $1, $2, ... $n-1`.

```
val t = (1, "hello", "world", 5)

t |> (x: Tuple => if ((x$0 + x$3) % 2 == 0) print(x$1 + " " + x$2)) // Prints
   "hello world";

//Alternatively
t |>
(x: Int, a: String, b: String, y: Int => if ((x + y) % 2 == 0) print(a + " " +
     b))

// Prints "hello world";
```

To declare a tuple with a single value it is necessary to follow it with a comma (like Python):

```
val t = (1,) //Type Tuple[Int,]
val i = (1) //Type Int
```

Lists: Lists replace arrays as the basic iterable data structure. Lists are zero-indexed and immutable, so that all operations create new copies of the list. Lists accept any type but must have a consistent type across all elements.

Pumpkin Lists also support basic head and List features, called as `hd` and `tl` respectively. Pumpkin also supports `is_empty` and `len`

Pumpkin supports the typical cons operator for new list creation.

```
//Normal construction
val myList: List[Int] = [1, 2, 3, 4]

// '::' is an operation that creates a new list by appending the element to
   the head
val newList = 10 :: myList // => [10, 1, 2, 3, 4)]

val head = hd(myList) //hd = 1
```

```
val tail = tl(myList) //tl = [2, 3, 4]
```

Maps: Maps act as a basic immutable Key:Value data structure that is staticly typed on both Key and Value. Maps are unordered and no guarantee can be made on the ordering of contents. Keys can be only primative types.

```
val myMap = ("x" -> "y", "a" -> "b", "t" -> "s")

val fetchedVal = myMap("x"); // => "y"
```

3.1.6 Arithmetic Operators

Pumpkin supports the following basic arithmetic operators:

- '+' Used for addition or String concatonation.
- '-' Used for subtraction or unary negative.
- '/' Used for division.
- '*' Used for multiplication.
- '%' Used for modulus.

There is no type elevation in Pumpkin. Therefore, these operators cannot be used to preform operations on mixed groups of Integers and Floats.

```
1 + 2    // => 3
4 / 2    // => 2
4.0 / 2  // => Type error
3 % 2    // => 1
6 - 2    // => 4
6 - -2   // => 8
```

3.2 Program Structure

Pumpkin is a compiled functional scripting language. Thus, the entry point of the code is the first line of a file, and the return type/value is whatever the last line of the file returns.

3.2.1 Comments

Single line comments are symbolized by two forward slash characters.

```
// Slashes mark the beginning of a single line comment.
```

Multi-line comments are written C-style.

```
/*
The slash and asterisk mark the beginning
and end of a multi-line comment.
 */
```

3.2.2 Indentation

Pumpkin does not use curly brackets to delimit scope, so correct white spacing is essential for programs written in the language. The indentation must be either a certain amount of spaces or a tab character, but it must be consistent throughout a program, as in the following example:

```
if (True):
    if (False):
        print("unreachable nested code")
    else:
      print("Even")
else:
  print("Odd")
```

3.3 Functions

In Pumpkin all flow-control is done through functions. Pumpkin does not support **for** or **while** loops.

Functions can be defined with the 'def' keyword. The types must not be specified due to type inference, but the the parameters must be written within parantheses. The exception to this rule is that recursive functions must always be declared with explicit return types.

Basic Syntax:

```
def funcName(parameter: type): returnType =>
  code
```

Single line functions are also allowed:

```
def x(y: Int) : Int =>  (y + 1)
x(1)
```

Here are a couple of ways to declare a value that answers "is 2 even?":

```
// value is determined when x is evaluated in an expression
def x: Bool =>
  even(2,)

val x = even(2,) // value is determined at compile time
```

Anonymous functions are also allowed for flow control. They must be declared on a single line.

```
(variableName: dataType => code): returnType
```

To pass a function as a parameter use the syntax name:(params $->$ return type).

```
def x(y: Int, z:(Int, Int -> String)): Bool => True
```

The function x takes two parameters; y, which is an int, and z, which is a function. z takes two int parameters and returns a string. x returns a boolean.

3.3.1 Function Chains

The symbols '| >' and '< |' can be used to chain nested function calls. The expression on the bar side is evaluated first. All arithmetic operators are applied before evaluating left to right, and parentheses are respected as in the traditional order of operations. All expressions on the call-side of the flow must be functions.

For example:

```
val a: Int = 3
a |> (x: Int => x + 1): Int // Returns 4;
```

NOTE: The function calls $funcName(x)$, $x| > funcName$ and $funcName < |x$ are semantically the same, but resolve differently with different precedence.

More examples of control flow:

```
val b: Int = 3
b |> (x: Int => x + 1): Int |> even
```

The above expression gets executed as follows:

1. Evaluate expression b: Int => 3.
2. Left-to-right: pipe into expr2, an anonymous function which takes one argument (an Int), adds 1 to it and returns a new Int.
3. Pipe result into even(), which is a function that takes an Int and returns a Bool.
4. No more pipes, left-to-right is done, return boolean value.

```
val b: Int = 3
even <| (b |> (x: Int => x + 1): Int)
```

The above expression gets executed as follows:

1. Evaluate expression b: Int => 3, since the parentheses give it precedence.
2. Pipe right into anonymous function that returns an Int.
3. Evaluate expression even: (x: Int => Bool) with the return of the anonymous call.
4. After evaluating there are no more pipes or operations, return boolean.

3.3.2 Multi-line Piping

Pipes will ignore whitespace. Thus if a line begins with a pipe it will push the return of the line above in. Once a line does not begin with a | > or < | sign it is outside the piped block

```
4
|> plusOne
|> even // return false

even(2) // outside pipe
```

3.3.3 Composing Functions

The << and >> operators can be used to call a fuction with the return type of another. For example:

```
(f << g) <| x or x |> (f << g) // is same as f(g(x))
(f >> g) <| x or x |> (f >> g) // is same as g(f(x))
```

NOTE: If at any time the order or type of the arguments don't match, compiler will throw errors. Furthermore only the inner-most function is allowed to take mutiple paramethers, in order to avoid unreachable paramethers.

Another example:

```
def timesTwo(x: Int):Int => x * 2
def plusOne(x: Int):Int => x + 1

def plusOneTimesTwo(x: Int): Int => x |> (plusOne << timesTwo)
def timesTwoPlusOne(x: Int): Int => x |> (timesTwo << plusOne)

plusOneTimesTwo(9) // => 20
timesTwoPlusOne(9) // => 19
```

3.3.4 Partially Applied Functions

Pumpkin supports partially applied functions. When a function is called without all of the necessary arguments, it returns another function that has the previously passed arguments already set. Arguments must be passed in order:

```
def plus(x, y):Int =>
x + y

val plusOne = plus(1)
val three = plusOne(2)
```

Another example:

```
[1, 2, 3, 4] |> filter(even) |> map(timesTwo) |> fold(add, 0)
// Returns '12'
```

3.3.5 Native Functions

Print: Printing to the standard output stream is akin to Java, and begins with the word *print* followed by the items to be printed in parentheses.

Escape characters include '\t' for tab, '\n' for newline, and '\\' for backslash as follows:

```
print("string")
print(variable)
print("This will print a tab \n")
print("This will print a newline \t")
print("This will print a backslah \\")
```

3.3.6 List Functions

A few built in functions are created specifically to operate on lists:

'hd' returns the first element of a list.

'tl' returns a new list with the first element removed.

'is_empty' returns boolean indicating if the list is empty or not.

'len' returns an integer indicating the length of the list.

4 Project Plan

As a group we decided on some intial deadlines that we wanted to meet. We tried to get language syntax and context free grammar done first. We then tried to write simple expressions all the way from scanner to sast. From there we kept adding functionality and refactoring as we went in oder to achieve the final result. Modulatrity was crucial in the process, as well as having a test oriented style of development where we would code to make certain tests passs or fail.

4.1 Programming Style Guidelines

- We used github for version control
- Commit often, with meaningful messages.
- Use pull –rebase to keep the commit tree in order
- Do NOT push broken code.
- Keep naming consistent and clear
- Avoid deep nesting
- Avoid code repetition
- Avoid very long lines of code

4.2 Project Time Line

End of September: Decide on language's goals and overall syntax.
October: Have a good grasp of Ocaml.
End of October: Have scanner, parser and ast done.
November: Analyzer and Sast.
End of November: Code generation and additional functionality.
December: Testing

4.3 Roles and Responsibilities

Joshua Boggs: Documentation and Testing.
Christopher Evans: Language decisions and code generation.
Gabriela Melchior: Managment and development.
Clement Robbins: Architecture and development.

4.4 Environment

Git was used for version control, the compiler was written in Ocaml. Our text editors varied, some of us used Sublime others Vim. For testing we used bash scripts. We compile down to javascript, using nodejs to run the executables.

4.5 Commit History

```
commit cd27f14a86e7b39768d03c3295af807a5f504afb
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 16 15:44:22 2014 -0500

    Example 2

commit 85f5588c65454bb51ee85c9f9d917dde34b75f7a
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 16 15:42:41 2014 -0500

    ex1 demo added

commit 109e25b6203fc26abb40d3fb791ec27819c40530
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 15:41:30 2014 -0500

    removed outdated tests

commit 97bed1bd5906bf0e58c83f7243c2d10c2a20804f
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 15:38:22 2014 -0500

    added more tests

commit 2a028719e129cc3bda6bde27e22813ca5cbe25ab
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 15:28:59 2014 -0500

    removed unop

commit 71acc83dc1230990f1a352ff71e64cf4c6276939
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 15:28:30 2014 -0500

    removed unit

commit b37e8afd7a49338740311764d6d2645087649cb5
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 16 15:23:41 2014 -0500

    fixed binops precedence in anon functions
```

```
42
43 commit 035b81fec53482901fa46a21164c50bc83bee6a7
44 Author: Joshua Boggs <joshua.j.boggs@gmail.com>
45 Date:   Tue Dec 16 15:15:43 2014 -0500
46
47     handle test output for javascript compile errors
48
49 commit e44dbacad4e9880ac499c256e4fd615a2b16197e
50 Author: Gabby2212 <gabymelchior22@gmail.com>
51 Date:   Tue Dec 16 14:55:51 2014 -0500
52
53     Fixing partial order
54
55 commit 4cc3cd016abf0e3cfd51f48fbb2d1788fabeec74
56 Author: Gabby2212 <gabymelchior22@gmail.com>
57 Date:   Tue Dec 16 14:44:55 2014 -0500
58
59     Letting u do things with library
60
61 commit 7721f9815b1ce5d192ffad8b3fb9f8d73df21b34
62 Author: Joshua Boggs <joshua.j.boggs@gmail.com>
63 Date:   Tue Dec 16 14:31:13 2014 -0500
64
65     renamed and completed more tests
66
67 commit 9bd1c9f7ae838748fdc0b1741e42e2d5d5d0430b
68 Author: Joshua Boggs <joshua.j.boggs@gmail.com>
69 Date:   Tue Dec 16 14:12:09 2014 -0500
70
71     removed intermidiary outputs
72
73 commit b352478b98ca7af61f41cc861a92a9a40a6c8926
74 Author: Chris Evans <chris.evans93@gmail.com>
75 Date:   Tue Dec 16 14:15:48 2014 -0500
76
77     reverse func calls (weird filtering)
78
79 commit f1ae626162b4d498b25c7f54e46c496bab3e6bfa
80 Merge: 600a42d d7ff89d
81 Author: Gabby2212 <gabymelchior22@gmail.com>
82 Date:   Tue Dec 16 14:12:23 2014 -0500
83
84     Merge branch 'master' of https://github.com/perks/pumpkin
85
86 commit d7ff89d046e6fd5fc6a332e1756cdd1e20264362
87 Author: Chris Evans <chris.evans93@gmail.com>
88 Date:   Tue Dec 16 14:12:02 2014 -0500
89
90     somefix
91
92 commit 600a42d6c9dab7663615b049b783949f5209a8f3
```

```
Merge: b7443e8 3d15943
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 16 14:12:08 2014 -0500

    Merge

commit b7443e8b81174d7a357f82ddfbe9859cc97c5d72
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 16 14:08:13 2014 -0500

    Reversing list

commit 3d15943d4e7bfea241410e76485477d346a62416
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 14:06:39 2014 -0500

    removed already tested files

commit 42b48e3d4e06b435ae7f78508e9fca60ecafa63c
Merge: 8193832 e05e3ed
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 16 14:05:19 2014 -0500

    Merge branches 'master' and 'master' of github.com:perks/pumpkin

commit e05e3ed2cf00999f3ffc4c76f8804847bc904a07
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 14:02:13 2014 -0500

    if else test

commit b8fc60a06158849a946359470f3b544a2c3b4a96
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 13:49:41 2014 -0500

    continuous test automation improvements

commit 8193832dc4ad339133ab5c9fefa4b11a1ac76cf1
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 16 13:43:55 2014 -0500

    undo reverse e_list tuples and lists

commit 7cdee25f5819952ed3c23d0f77a1bb8e3a680a28
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 02:14:57 2014 -0500

    fixed test, yet still fails

commit 069624863ffa4319867bb4397b01b63f45da7ac1
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
```

```
Date:   Tue Dec 16 01:45:18 2014 -0500

    continued testing

commit 1275bb1820ed0a64e5edc15ce59df16826f538bc
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 16 01:57:05 2014 -0500

    demo

commit 50888ea7839765b5e903c2b59835380b569fe320
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 16 01:44:41 2014 -0500

    Indexing bug in tuples

commit c34faea3bf01282525b517fe49b372793aff18a3
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 16 01:38:17 2014 -0500

    OMG REVERSE BOOLEAN IS DEATH

commit cfdebb0e1f94c5ccadfcbb88cd26a51118f2d868
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 16 01:37:03 2014 -0500

    Reversed params

commit a6d2e2cfd64fb9675f22d4c0e6ba77e230bc5ed8
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 16 01:31:26 2014 -0500

    demo

commit e42929caa5b3afd1be27714d37a7af753df50cc8
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 16 01:22:07 2014 -0500

    Head and tail

commit a6f0459c650d3f3f6438f162d9abe1a3e2fbd5be
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 01:19:46 2014 -0500

    removed deprecated outputs

commit b285248832261b06c315b09c2e38f48cb112993d
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 01:19:03 2014 -0500

    testing booleans, chars, and maps
```

```
195
196 commit 05568f08640617a6a21e431192f95ad29cfb9a50
197 Author: Clement Robbins <cjr2151@columbia.edu>
198 Date:   Tue Dec 16 01:12:33 2014 -0500
199
200     demo
201
202 commit b4723c857edddd4a4221e3a868f320c83cb7a086
203 Merge: 21b3d9a aa4ae57
204 Author: Gabby2212 <gabymelchior22@gmail.com>
205 Date:   Tue Dec 16 01:11:49 2014 -0500
206
207     Merge
208
209 commit 21b3d9a0a8856450211d9879e92b8cf38f01c456
210 Author: Gabby2212 <gabymelchior22@gmail.com>
211 Date:   Tue Dec 16 01:10:38 2014 -0500
212
213     Issue with head and tail
214
215 commit aa4ae57061b96beaacbd537d5fee5f90fc1a4384
216 Author: Chris Evans <chris.evans93@gmail.com>
217 Date:   Tue Dec 16 01:08:25 2014 -0500
218
219     name fix is_empty
220
221 commit eccadd65675d139fcd01399c448e302629640702
222 Author: Chris Evans <chris.evans93@gmail.com>
223 Date:   Tue Dec 16 00:58:34 2014 -0500
224
225     brace codegen
226
227 commit 8bdfd3d218e5a7d7deb832b6e244ae6ae3683764
228 Author: Clement Robbins <cjr2151@columbia.edu>
229 Date:   Tue Dec 16 00:52:27 2014 -0500
230
231     demo
232
233 commit 56a0b9be807264245fbbb79fbeb34082014f8c07
234 Author: Clement Robbins <cjr2151@columbia.edu>
235 Date:   Tue Dec 16 00:52:00 2014 -0500
236
237     demo
238
239 commit 5e41f5d0e286dadbb58a149ee48ac504475f5910
240 Merge: 0b57c07 00b8f64
241 Author: Gabby2212 <gabymelchior22@gmail.com>
242 Date:   Tue Dec 16 00:50:37 2014 -0500
243
244     Merge branch 'master' of https://github.com/perks/pumpkin
245
```

```
commit 0b57c07169cc325327490f98283a519c4658355a
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 16 00:50:20 2014 -0500

    Library function checks

commit 00b8f645b082ecffb139da2b1a9192348e9c19fd
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 16 00:43:02 2014 -0500

    changed return is_empty

commit 54570df9e6cd59b57aab40ba9da2b9cb98d15f13
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 00:37:06 2014 -0500

    added test instructions to readme

commit 34a67404ee912ca3435ba1b4130179c21935262b
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Dec 16 00:31:41 2014 -0500

    colorized tests and removed useless compile test

commit ab61e274bfb20b51cfe673f7e54919762c5052fc
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 16 00:21:52 2014 -0500

    i hate this planet

commit a5b3b7cbcb40116c241bd38bb9db34e1fbad8340
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 16 00:10:06 2014 -0500

    demo

commit 3d86abb296b4fd7ee691698dd5ff28f7ceb6e12b
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 16 00:07:11 2014 -0500

    Added excpetion for reserved funcs

commit 807fcdb966fe691d4c6ddf0c5058d1ae6324df92
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Mon Dec 15 23:41:37 2014 -0500

    binary operator tests and expected output

commit 24eb7036a42c8a3a6cf1fdbe223d55e6fe102868
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Mon Dec 15 23:41:06 2014 -0500
```

```
297
298     ignore whitespace
299
300 commit a334912b7a083bf7ebf5b0265ddb801159be8590
301 Author: Joshua Boggs <joshua.j.boggs@gmail.com>
302 Date:   Mon Dec 15 23:22:09 2014 -0500
303
304     renamed and added print statements to binary operator test
305
306 commit c6604312a7f99e004752739dd4f466d06fe86df7
307 Merge: d85ba3c 50b2c4e
308 Author: Gabby2212 <gabymelchior22@gmail.com>
309 Date:   Mon Dec 15 23:38:13 2014 -0500
310
311     Merge branch 'master' of https://github.com/perks/pumpkin
312
313 commit d85ba3ca5f7a8480fce519e977f128405166d172
314 Author: Gabby2212 <gabymelchior22@gmail.com>
315 Date:   Mon Dec 15 23:37:52 2014 -0500
316
317     Added reserved functions
318
319 commit 50b2c4ee3659178f09ae3572d85ee16611cf319b
320 Author: Clement Robbins <cjr2151@columbia.edu>
321 Date:   Mon Dec 15 23:22:09 2014 -0500
322
323     big fixes
324
325 commit 2c4b559c0616f686ece453a2380e759e092d2439
326 Author: Clement Robbins <cjr2151@columbia.edu>
327 Date:   Mon Dec 15 23:16:48 2014 -0500
328
329     fixing codegen
330
331 commit 7e92ae3c1f118ebb07a5188ae7286cbdc44d8b23
332 Author: Chris Evans <chris.evans93@gmail.com>
333 Date:   Mon Dec 15 23:15:49 2014 -0500
334
335     Final codegenishhhh
336
337 commit b4f1e01fb44baf0e58fac6754215fb6da894ef6c
338 Author: Clement Robbins <cjr2151@columbia.edu>
339 Date:   Mon Dec 15 23:15:00 2014 -0500
340
341     removing matching
342
343 commit 468293e477b79f6a70b0d0fd40c3efcb4f3f244d
344 Author: Joshua Boggs <joshua.j.boggs@gmail.com>
345 Date:   Mon Dec 15 23:11:19 2014 -0500
346
347     assignment tests
```

```
commit f78ec0be604348bba28f235a4797002819b36ac8
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Mon Dec 15 22:58:15 2014 -0500

    created automatic testing suite

commit 1f79de13bc97e88fe753ba75d2b0cf2bc4c62be3
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Mon Dec 15 22:48:12 2014 -0500

    removing old analyzer

commit e5d88d146059b3362e7a5321f884b2c4efb093de
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Mon Dec 15 22:44:56 2014 -0500

    removing algebraic data types

commit 03c07ec25e9fc81572c73923be91f353be574487
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Dec 15 22:43:13 2014 -0500

    Added __cons__ and __compose__ libs

commit 389066996261c5011c9db6094b99a0d8af75b979
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Dec 15 22:18:44 2014 -0500

    pipes codegen and composition

commit 9883f5911ed5f61b432ee2e81d6be77d83eb69c1
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Dec 15 21:38:14 2014 -0500

    Sanitize maps in maps codegen

commit 9c53b7db3d610cb4af97432a8fd30e8979e25568
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 15 22:05:29 2014 -0500

    Fixed function scoping issue

commit 27a6b9fcca247a446c677d864ffca5c0d5a24d8c
Merge: 296cbfa 6f488f2
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 15 21:37:41 2014 -0500

    Merge branch 'master' of https://github.com/perks/pumpkin

commit 296cbfa2ba3014493921e040285509788b7ff13f
```

```
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 15 21:37:23 2014 -0500

    Fixed precedence

commit 6f488f2d5c85e48aaa1ecc873c4c8619aed5fa69
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Dec 15 21:25:55 2014 -0500

    tuples codegen

commit 2f3188864ae79adfafb6d8bd56b4522ae1f4bf42
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Dec 15 21:15:26 2014 -0500

    Fix funccall, Maps

commit 1ed168e967a099ed3f9612243d7ef40af2ceefdf
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 15 21:12:15 2014 -0500

    Piping

commit 926b2f2fdb0c00fece3671f8b546876cdb320ec2
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 15 21:07:40 2014 -0500

    Piping looks ok

commit b50ffaa01804d6126bcf0271bad6d21022b58287
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Dec 15 21:06:25 2014 -0500

    MapAccess + AFuncCall + Print for codegen

commit 8ac3a31e30d5828ec34fdc7103d47f99b45aa8ab
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Dec 15 20:47:10 2014 -0500

    MapAccess

commit 6dd9e69ec93e5c1b60cbf007fabde7ca72585d63
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 15 20:46:47 2014 -0500

    Added print

commit 9acbdf213844d77ea3d205d25401417dbca9f3b7
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Mon Dec 15 20:35:33 2014 -0500

```

```
450     getting rid of codegen call
451
452 commit 45238843fd841422acf5419fc7af14ca21e2bcdd
453 Author: Gabby2212 <gabymelchior22@gmail.com>
454 Date:   Mon Dec 15 20:33:43 2014 -0500
455
456     Changed calls and map access
457
458 commit 3232b6adf38c98087a4ff6958b9aea8da83ac23d
459 Author: Chris Evans <chris.evans93@gmail.com>
460 Date:   Mon Dec 15 20:10:16 2014 -0500
461
462     If/else return functionf fix
463
464 commit c8840a17759a406994191bbc3067499429e79732
465 Author: Joshua Boggs <joshua.j.boggs@gmail.com>
466 Date:   Mon Dec 15 19:58:05 2014 -0500
467
468     test all javascript files
469
470 commit 00ecee3a262e73c166bfd5ae5ebd3e0a8f8639af
471 Author: Clement Robbins <cjr2151@columbia.edu>
472 Date:   Mon Dec 15 19:46:36 2014 -0500
473
474     scanner fix
475
476 commit e1f6456dbeca438b23a78a785a97693f8b7acc15
477 Author: Gabby2212 <gabymelchior22@gmail.com>
478 Date:   Mon Dec 15 19:16:32 2014 -0500
479
480     Changed scoping for if and ifelse block (vars declared inside are only
    seen inside
481
482 commit c20113d6008488dd224f090a321a8fb87c6d8bc1
483 Author: Gabby2212 <gabymelchior22@gmail.com>
484 Date:   Mon Dec 15 18:29:01 2014 -0500
485
486     All if blocks return unit
487
488 commit 9fb072e9aa1bb540227aa2243b720f1a0a6583d4
489 Merge: d817550 ea75802
490 Author: Gabby2212 <gabymelchior22@gmail.com>
491 Date:   Mon Dec 15 18:26:17 2014 -0500
492
493     Merge branch 'master' of https://github.com/perks/pumpkin
494
495 commit d8175508ae0226176b1a0ae1971599b5f9cdbaf6
496 Author: Gabby2212 <gabymelchior22@gmail.com>
497 Date:   Mon Dec 15 18:26:01 2014 -0500
498
499     Added composition
```

```
commit ea75802743782e3f1d706c88f2b1a893eea5075b
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Mon Dec 15 18:09:04 2014 -0500

    wrote script to compile tests to js

commit ebd78b41bd1ec972d8085436e3595041be76480e
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Mon Dec 15 18:08:34 2014 -0500

    added shebang to script

commit 265d67ae4053197dbf018adf4bc8a8e195b940d5
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Dec 15 17:14:51 2014 -0500

    ACall Codegen

commit 915f904c368f38eee8da641b755da228a0587061
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Dec 14 11:58:49 2014 -0500

    Func decl + func Anon w/ gabi empty params =  Unit:()

commit b2848aac35af3413d2101303d5f95908fbf49f0d
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Mon Dec 15 16:47:55 2014 -0500

    added newline after compiled output

commit cffbaf0f94cc2be46e236e4022b0879bd6e80295
Merge: a9b7524 b69274f
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 15 16:17:27 2014 -0500

    Merge branch 'master' of https://github.com/perks/pumpkin

commit a9b75243a6e8bf9bf2866d6b7e9ecc2a7748425f
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 15 16:17:12 2014 -0500

    Fixed call mismatch issue

commit b69274faf9b668d50f0b93d7750cdfbd47e7a736
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Mon Dec 15 00:10:02 2014 -0500

    removed test output from git tracking

commit 3322d9e09190f501575547c3583da828cb2bc44e
```

```
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Sun Dec 14 18:27:57 2014 -0500

    created a script to run all *.pk tests and output errors and a log

commit 1a032e91da16c26e5acdfe321e4086b8f640a6c7
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 14 17:35:53 2014 -0500

    ACall printing

commit f89aba89be6c47ee654fb1da2f00f24af7a1d61f
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 14 17:27:28 2014 -0500

    Changed empty params back

commit 6564b522cf9c6558470f414a54f8e7346b00409c
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 14 02:06:40 2014 -0500

    Fixed empty parameters

commit c894741607368de189c48e7a4cbddbbca6e85635
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 14 01:22:24 2014 -0500

    Calls

commit 78978d6d686506d21358873a359b78b4315e6c4e
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 14 00:44:54 2014 -0500

    Added printing for anon

commit 14e872334baef70b057159be97b56ff32e5273a9
Merge: 0e136e3 0aee171
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 14 00:29:10 2014 -0500

    Merge branch 'master' of https://github.com/perks/pumpkin

commit 0e136e32ca59d86d48dc73018f0630b6457d0425
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 14 00:28:46 2014 -0500

    Function declarations

commit 0aee1712471b12045e5f3bba3b85ee1a810d12b0
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Dec 14 00:13:52 2014 -0500
```

```
602
603     Added new func decls
604
605 commit d729d3d4b156708fd4bc5c01f9be14f933ba6925
606 Author: Gabby2212 <gabymelchior22@gmail.com>
607 Date:   Sat Dec 13 23:56:19 2014 -0500
608
609     Check for unit functions
610
611 commit 0c34677ecc87261bd5dc5a17e2f9b18792f3efc2
612 Merge: 7771847 a47bafb
613 Author: Gabby2212 <gabymelchior22@gmail.com>
614 Date:   Sat Dec 13 23:53:24 2014 -0500
615
616     Merge branch 'master' of https://github.com/perks/pumpkin
617
618 commit 7771847f148754da97e50e424bfbf3a5466e5d74
619 Author: Gabby2212 <gabymelchior22@gmail.com>
620 Date:   Sat Dec 13 23:53:11 2014 -0500
621
622     Did TypedFuncDecl
623
624 commit a47bafb362f70b9d9e93de499217bfae0843c019
625 Author: Chris Evans <chris.evans93@gmail.com>
626 Date:   Sat Dec 13 23:22:54 2014 -0500
627
628     Added map creation codegen (need access)
629
630 commit 2f178f3287e6815aecbab4b98abdfe42629064a3
631 Author: Chris Evans <chris.evans93@gmail.com>
632 Date:   Sat Dec 13 22:57:54 2014 -0500
633
634     Codegen progress
635
636 commit c6e379127c7ea2cc3d7b0ed66f1c2b09a333ad67
637 Author: Chris Evans <chris.evans93@gmail.com>
638 Date:   Sat Dec 13 21:55:09 2014 -0500
639
640     Redoing codegen
641
642 commit be8482d0cbb8cabae419edff775c4f65eea7ecad
643 Author: Chris Evans <chris.evans93@gmail.com>
644 Date:   Sat Dec 13 22:50:53 2014 -0500
645
646     Update README.md
647
648 commit ce26180baf94b30eec1b936f51943fe2e83a5b17
649 Author: Gabby2212 <gabymelchior22@gmail.com>
650 Date:   Sat Dec 13 21:53:31 2014 -0500
651
652     Fixed printing
```

```
commit d2e275ed49e734726bcd4448895b2045f4c69a58
Merge: 66880ef c3bfb12
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Dec 13 21:43:36 2014 -0500

    Merge branch 'master' of https://github.com/perks/pumpkin

commit 66880ef347515667e65dfd0b57dee7369aab987a
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Dec 13 21:43:17 2014 -0500

    If else re-added

commit c3bfb123562dccc4f76cd0c1fde6b237a5a57eca
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Fri Dec 12 16:04:40 2014 -0500

    Anonymous functions working

commit c139cf6ad679b0e1c501af0ee0b65178cbdfd629
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Fri Dec 12 15:15:02 2014 -0500

    If/else block + semicolon cleanups

commit 75ebf28636f1ebd767694e92fff5d07c3dab299e
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Fri Dec 12 14:29:04 2014 -0500

    Added proper returns to last line of non unit functions

commit cbfb5e89d5cff98b31483986a1d45455899b0548
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Dec 13 20:21:56 2014 -0500

    Numbers.pk and Tuples.pk done

commit 752ed97535357bdc4ae916f54c57a1b71b951d22
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Dec 13 17:50:50 2014 -0500

    Putting things back and testing

commit 65ea88b79aa3eeb8166e25ee510fdf32ed813938
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Fri Dec 12 23:17:45 2014 -0500

    Fixed the tuple thing

commit c35dbcfe8d3f248a5799112b8803bf14accbd1ad
```

```
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Fri Dec 12 22:04:55 2014 -0500

    major change

commit 16085605b193bdc0e94fd9fc990a5799d2b61132
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Fri Dec 12 20:26:11 2014 -0500

    Update README.md

commit 5a46e50e1c48ae08ecd5c96be7cf068dbc873481
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Wed Dec 10 07:17:16 2014 -0500

    cleaning up top level

commit c758edbe5fab6358bd73cb3e4f6cccaddbf57010
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Wed Dec 10 07:15:22 2014 -0500

    finishing scanner and parser

commit 8ec57ac101d67c7be2e5bd7f4f187af4f7ae8a02
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Wed Dec 10 06:28:06 2014 -0500

    adding function typed arguments

commit 7838eebccd79cd6e7ae7a43abc5c072aee919211
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Wed Dec 10 03:43:58 2014 -0500

    many bug fixes, fixed #8

commit 23528f2f9b03fce471a83a9eac35279eddec7494
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 9 22:39:36 2014 -0500

    added error checking for parser, misc bug fixes

commit f4346ebe59d9e63fcd5735072ca8bad8635f7955
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 9 20:32:27 2014 -0500

    general code cleanup

commit 9bac4ba0f2101580acf18abbf8717c20c9767f34
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 9 15:36:55 2014 -0500

```

```
    changed readme tasks

commit 35cdeb5d8b2f0d7fb4656a4da49c965bec3544c4
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 9 15:18:50 2014 -0500

    removing .future files

commit 086625febf1b9481da040aff2b9d8eb466f6a225
Merge: b8df059 afb3f23
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 9 15:07:37 2014 -0500

    Merge branch 'master' of https://github.com/perks/pumpkin

commit b8df0593aee7cb1feb790b4e4b937ce33dcc38e6
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 9 15:07:20 2014 -0500

    Type checking for pipes almost done

commit afb3f23013ab5119c5085d64be5545e1fcfc5c4a
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 9 15:02:05 2014 -0500

    changing algebraic datatype ssyntax

commit c0b5ed96a1b2b9c9b7fd656f999362f25643a456
Merge: f90207a 1118dfa
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 9 14:08:11 2014 -0500

    Merging

commit f90207a1314c1094943d25ff7d99a6af044b7a36
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 9 14:03:45 2014 -0500

    Working on pipe type checking

commit 1118dfacb3c71e9f2cb9dbf904a9d68c007d8cc3
Merge: 2c7cb28 a08a724
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 9 12:47:27 2014 -0500

    Merge branch 'master' of github.com:perks/pumpkin

commit 2c7cb28afd075cec6507efe3e2f2e9649c621759
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Tue Dec 9 12:47:23 2014 -0500

```

```
    Starting javascript codegen (testing)

commit a08a7248af27c97a116ad2f0ae571e6aa2c429da
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 9 12:36:47 2014 -0500

    changes for expression list

commit 10b0329aa71efae6c3d6d3b9d9086956f1cd3e34
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Tue Dec 9 12:09:08 2014 -0500

    fixed issue #7

commit bcde592f039a7e15a3170348dcaf542ff4c464cf
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Dec 9 09:44:10 2014 -0500

    Fixed make

commit 6b53572b412218eff29bf8a3641d32789adfa9e4
Merge: 878a939 991ba35
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 8 20:03:50 2014 -0500

    Merged

commit 878a939c98c64ca0b6afafe20718adf9342d2489
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 8 20:01:29 2014 -0500

    Added composition type checking

commit 991ba352e15174800ce6cfce3bc094c7614dbd39
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Mon Dec 8 19:46:58 2014 -0500

    added readme issues to github issues

commit 0f2b7da5aa23f9661bae82abbee30046fc20b939
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Mon Dec 8 19:32:18 2014 -0500

    fixing comment issue

commit b945480a345e1707d3da4e612c6f5fa6f7444dc1
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Mon Dec 8 19:11:14 2014 -0500

    added uptyped functions to parser and scanner

```

```
commit 620749db3d41b8f8b77d6a465a2b2eebcb827d0d
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Mon Dec 8 18:48:47 2014 -0500

    match statements

commit 59dde43475b58a34b2a511b45997adf2a7b0c7cb
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Mon Dec 8 18:02:41 2014 -0500

    rebasing from upstream

commit 632de694f1dec39357f1e4c5255d9406f9d050c1
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 8 18:07:27 2014 -0500

    Idk

commit 24559b3eb23643c3a154a27fcf89e3b286e58064
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Sun Dec 7 21:03:04 2014 -0500

    match statements

commit 3d8ee020eb0eddb3c3f1fc1ff3953cc2e8d64cec
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Sun Dec 7 21:01:28 2014 -0500

    beginning a more robust test file and documenting errors in the README

commit e77ea8162595dceb1727b73bf5f69e86a3bbf025
Merge: c01ca90 dff1610
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 7 20:44:35 2014 -0500

    Merged

commit c01ca90441b6d5b01ce91b0f2e2617e8a6a517ef
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 7 20:43:54 2014 -0500

    Added partial functions type checking

commit dff16108244160a4717d6e8e39822aa745e7d4c9
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Sun Dec 7 20:34:54 2014 -0500

    adding match statements, objserving function error

commit 68e265391518a1d758812d51001187e0adedc4e6
Author: Clement Robbins <cjr2151@columbia.edu>
```

```
Date:   Sun Dec 7 20:10:58 2014 -0500

    error in parser

commit eab0ea1b18f2f6da7d732eb013f69278da907aae
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Sun Dec 7 20:08:53 2014 -0500

    fixing toplevel make, fixing alg args in parser

commit e46fb89f72559fbc8566c4c58f52d924298aedfe
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Sun Dec 7 19:35:46 2014 -0500

    merged and changed type annotations and typos

commit 5a594437986445472ad869eadae7ea949371ab88
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Sun Dec 7 19:22:52 2014 -0500

    merged and changed type annotation style

commit 5bd4044f4c6b4add894ab74cdbc8cac14dcb2ee0
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Sun Dec 7 19:03:52 2014 -0500

    utils code cleanup, changing toplevel makefile

commit c32012e77f0e602c1333498270be555014d18949
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Sun Dec 7 18:15:11 2014 -0500

    added alg data types to parser and scanner

commit 548a6c239ed013290a96c6da4e925f4c4235f555
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 7 16:49:59 2014 -0500

    Added simple parameter checking for function calls

commit ef8eb145100bf547f6f38366303355a539bf2364
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Dec 7 13:50:21 2014 -0500

    Minor changes based on intial sast testing

commit eea83e1257d50d40ab3550c2c1989ffd305f6461
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Sun Dec 7 13:10:05 2014 -0500

    completed printing for SAST
```

```
commit 04b2a7b23c890fd153e7cf38ad127799319b5749
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Sat Dec 6 19:19:47 2014 -0500

    continued SAST printer and removed extra whitespace

commit 0045854723c7eb409dd26d291bd6b85f44d7fc36
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Dec 6 18:43:37 2014 -0500

    Started added Sast printing for tests

commit 7b46248cc2338a60da8e779166631f3088dcfb7c
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Dec 6 18:12:33 2014 -0500

    We are not using that symbols table file

commit b726234b990a5a7496d27b0579d970d725893c33
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Dec 6 18:11:10 2014 -0500

    Created symbols table and added everything we have to the sast and
    analyzer

commit 6133e364bb05486b483c7cc549c82e9d4100b53a
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Fri Dec 5 21:47:49 2014 -0500

    Added in line declarations

commit 7eed6fb2431c0fa7139b054cf6be592caad4aabb
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Fri Dec 5 21:06:51 2014 -0500

    Added mixed pipe and composition

commit a8440eae872e4eaf9f8ba2fc14399afacb560142
Merge: bc88953 9e7d78b
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Fri Dec 5 19:58:01 2014 -0500

    Merge conflicts solved

commit bc8895339285b6f6a67a53eedddf2cce6d881934
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Fri Dec 5 19:57:13 2014 -0500

    Added function composition
```

```
commit 9e7d78bfde7168f9a40e53560dce7a571ef18dfe
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Fri Dec 5 19:14:12 2014 -0500

    fixing tupal and list accessors

commit 84ecd91f5935e7db16c807de96610c9900eb7ddb
Merge: e108ab0 c9f8561
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Fri Dec 5 18:33:49 2014 -0500

    Merged with maps

commit e108ab0c73553f9e66e6db9911dcf19292d93240
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Fri Dec 5 18:32:49 2014 -0500

    Added function piping (oh yeeah)

commit c9f8561c6a8f23ec4912dc0d7c8fcc94472714e9
Author: Clement Robbins <cjr2151@columbia.edu>
Date:   Fri Dec 5 17:39:34 2014 -0500

    added maps to parser and scanner

commit 842cea8f0811356716bed11ece9eeac24eddc62e
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Fri Dec 5 16:17:50 2014 -0500

    Added simple function decl and calling

commit 9199840a0852655bfd17d02e8f8b43c290abf80c
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 1 12:12:40 2014 -0500

    Added simple function declaration and arguments

commit 355bbcc2fc9b1c2c486ef6d5201835fed5eb1b29
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 1 10:19:58 2014 -0500

    Added float to sast and analyzer

commit 992731b2257222ade672b98d56efb04d3b7c6f1a
Merge: 70ec0dd 78db75f
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 1 10:13:48 2014 -0500

    Merged changes in

commit 70ec0ddeca98eed6156f0d54ba03b13b3d59a738
```

```
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Mon Dec 1 09:53:34 2014 -0500

    Updated README

commit 78db75ff66ffca02d3828a3a14eccabcc177055b
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Mon Dec 1 00:18:46 2014 -0500

    added list accessor

commit 1a3c17d33c7d5884ed241a58754f8d9395793efb
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Mon Dec 1 00:06:05 2014 -0500

    adding tupal accessor, fix general parse errors

commit d73b58e2389858948592e240c9082c89a0d00a38
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Nov 30 22:18:35 2014 -0500

    parser cleanup, change uniop to unop

commit ef5241b9b6b1c535030d3e7cdc7be541026cd5ec
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Nov 30 20:56:05 2014 -0500

    fixing control flow parsing and scanning

commit c3e0e43f20706a33d0e42aec936b44b2fa00638d
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Nov 30 19:48:03 2014 -0500

    push code with parser broken so that everyone has scoping rules

commit e30f668fe58d8366c09fc4a95f7e80022a00c9bb
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Nov 30 19:09:12 2014 -0500

    fixing whitespace scoping

commit 23c5bbe184f2beb3e8bde009ec4620d02ded5b01
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 30 18:01:05 2014 -0500

    I think I fixed string interpolation

commit c636260e7669e86a0a68bbeb7dd06ea184b66d90
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 30 18:01:05 2014 -0500

```

```
    I think I fixed string interpolation

commit 17efccfc516a8d771fe728b968bf332d407b807e
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 30 17:59:09 2014 -0500

    String interpolation done maybe

commit f5b5a9366d0a02cf8f17786984cddbdf893db25b
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 30 17:59:09 2014 -0500

    String interpolation done maybe

commit 62be91f46bcd95f83d2fe86bdace74440e4d06cb
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 30 01:29:10 2014 -0500

    Working on string interpolation

commit a5a44125db3848a10d9b3ed2430fe1a25a48434f
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 30 01:29:10 2014 -0500

    Working on string interpolation

commit 0fe718f507fe9c79b4f057057976d411ce4ae01c
Merge: 920ca43 86b3e0e
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 21:35:43 2014 -0500

    Merge branch 'master' of https://github.com/perks/pumpkin

commit 6f9c704bbdca8fecc8ea5e145e2990fa2ca1c347
Merge: 8c1080e 5dd4935
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 21:35:43 2014 -0500

    Merge branch 'master' of https://github.com/perks/pumpkin

commit 920ca437ad0b958c0da60aa0ff0213e8990cc50c
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 21:35:37 2014 -0500

    Just modified readme

commit 8c1080e419c0c309be0fb4180a43b00b7442640f
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 21:35:37 2014 -0500

    Just modified readme
```

```
commit 86b3e0ea6ac83e56305a39b4d0805eef7619891b
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Sat Nov 29 21:34:37 2014 -0500

    fixing grammar rules for tupals

commit 5dd4935cd25b84cc84a0a91b792100fc19c5f43f
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sat Nov 29 21:34:37 2014 -0500

    fixing grammar rules for tupals

commit 615d52a7d759cd35a5d99bc894f62fb078a71244
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 21:10:23 2014 -0500

    Changed Num to Int

commit 55fcbc1f656870fdf961bedc29467a5a23091499
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 21:10:23 2014 -0500

    Changed Num to Int

commit 1cd583c57f726d73076f1ed21b55358db4969462
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 21:06:20 2014 -0500

    Added binop and uniop validation

commit 7054a1257420bd605b033172e92293f08ab9319b
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 21:06:20 2014 -0500

    Added binop and uniop validation

commit 5a06049d5278fc2d4573bd533a4e1bbfb151c589
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Sat Nov 29 20:57:22 2014 -0500

    merging in gaby's code for control flow, type inference, untypes
    assignment, and booleans

commit c2b54393aadb7bbd1a524e203acc2d51d2615385
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sat Nov 29 20:57:22 2014 -0500

    merging in gaby's code for control flow, type inference, untypes
    assignment, and booleans

```

```
commit 9c6a341eb6761f65c59f1f95ab875fc669357ac8
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Sat Nov 29 20:02:42 2014 -0500

    fixing makefile, adding utilities, raw printer, lexing and syntax errors

commit 8269b20a430c1930e0efdad4b0dd0e100d5c65dc
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sat Nov 29 20:02:42 2014 -0500

    fixing makefile, adding utilities, raw printer, lexing and syntax errors

commit a3fe5d12f2d6f09eb966d4ea7bc747d39329de04
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 19:41:11 2014 -0500

    Added AND and OR, and maybe type inference

commit b8f1563a36e46f0bd7dc63bd4f57e869c9de865f
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 19:41:11 2014 -0500

    Added AND and OR, and maybe type inference

commit 05f8e6abd093410386159ed3f0405cba868531c8
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 19:13:14 2014 -0500

    Added List(1, 2, 3) list declaration type

commit 3c24f60e589658af1b9b3dc0ed2ee94671131a27
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 19:13:14 2014 -0500

    Added List(1, 2, 3) list declaration type

commit 7817b48efb91106e092f43dd28ba9d6d060fe312
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 19:09:27 2014 -0500

    Added if else

commit 2b805d120f8ba2229ff61dc7860b3e4ddd6e22aa
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 19:09:27 2014 -0500

    Added if else

commit 8d386791d8916866744b9ec6a8cf420b46f6a430
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 16:57:00 2014 -0500
```

```
1262
1263     Tuples, lists and some little fixes
1264
1265 commit 2bd5ae1d15fb01329cf5383f4b39e0f26c16ead0
1266 Author: Gabby2212 <gabymelchior22@gmail.com>
1267 Date:   Sat Nov 29 16:57:00 2014 -0500
1268
1269     Tuples, lists and some little fixes
1270
1271 commit b096005e755b2935a4f9d277b5550def4ad9a072
1272 Author: Clement Robbins <cjr2151@columbia.com>
1273 Date:   Sat Nov 29 15:57:42 2014 -0500
1274
1275     fixing makes
1276
1277 commit 7a2158ad0e9ab0e12592200a5c923c8f4df2eceb
1278 Author: thebicycle2 <cjr2151@columbia.edu>
1279 Date:   Sat Nov 29 15:57:42 2014 -0500
1280
1281     fixing makes
1282
1283 commit 45b2709808245c06d234d3d4874770aa3c7fb6a2
1284 Author: Gabby2212 <gabymelchior22@gmail.com>
1285 Date:   Sat Nov 29 15:26:52 2014 -0500
1286
1287     Added char
1288
1289 commit fc60eb6960b9069c10b66e774c3e050d668b753c
1290 Author: Gabby2212 <gabymelchior22@gmail.com>
1291 Date:   Sat Nov 29 15:26:52 2014 -0500
1292
1293     Added char
1294
1295 commit a97bf22e35552ad565705078e5394615fdb3169d
1296 Author: Gabby2212 <gabymelchior22@gmail.com>
1297 Date:   Sat Nov 29 15:12:34 2014 -0500
1298
1299     Added strings
1300
1301 commit 8caeff14e5b29d8929c44762d33b89850be17f17
1302 Author: Gabby2212 <gabymelchior22@gmail.com>
1303 Date:   Sat Nov 29 15:12:34 2014 -0500
1304
1305     Added strings
1306
1307 commit 58c233cc01e059292d397357891ed71e847ce17d
1308 Author: Clement Robbins <cjr2151@columbia.com>
1309 Date:   Sat Nov 29 14:53:33 2014 -0500
1310
1311     added makefile
1312
```

```
commit 2b342dfca878aa7063bd308e570a499482eda47f
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sat Nov 29 14:53:33 2014 -0500

    added makefile

commit ee9854e5e806d61cdbb7b822a273e9e2255da6ac
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 14:53:08 2014 -0500

    Added binop and assignment

commit 77398c2d6050a285a88af89e5d9c9aa10a4e8dce
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sat Nov 29 14:53:08 2014 -0500

    Added binop and assignment

commit 10ca22070c1372f3868c3129c53f6564c60529fe
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 23 17:27:32 2014 -0500

    Added bool and assign

commit 6f52078316786d66197dcd6af0171cb74b4c4106
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 23 17:27:32 2014 -0500

    Added bool and assign

commit 154f8a2fac3cff8537c6a491aa7029e0a42d184a
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 23 16:44:11 2014 -0500

    Fixed parser rule error on expression

commit 3245d2c2b0b8d852e7950b13d2d54dea41a08827
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 23 16:44:11 2014 -0500

    Fixed parser rule error on expression

commit f0fdb999ed1f91e1e4fd1bbcc7f9141e980edfb4
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Sun Nov 23 16:12:51 2014 -0500

    finalized nums as integer type

commit 02409b56fdeca3d91f82261bdb2cacb9d39c4a3f
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Sun Nov 23 16:12:51 2014 -0500
```

```
1364
1365     finalized nums as integer type
1366
1367 commit 7cabc1cac68d1cf6a1fac4c9d9b82bb681b28872
1368 Author: Chris Evans <chris.evans93@gmail.com>
1369 Date:   Sun Nov 23 16:32:55 2014 -0500
1370
1371     Cleanup + indentation parser and scanner
1372
1373 commit ca2c2cbacd81bf7db7e9cbdcea0a5e55ac0eb1af
1374 Author: Chris Evans <chris.evans93@gmail.com>
1375 Date:   Sun Nov 23 16:32:55 2014 -0500
1376
1377     Cleanup + indentation parser and scanner
1378
1379 commit f5ca93d49ad6b8e51f943a0d11fe375b4974844f
1380 Author: Clement Robbins <cjr2151@columbia.com>
1381 Date:   Sun Nov 23 15:29:41 2014 -0500
1382
1383     deleted exec, update makefile
1384
1385 commit 686f217a7c48ab0790e1c66954e9df332c401fd2
1386 Author: thebicycle2 <cjr2151@columbia.edu>
1387 Date:   Sun Nov 23 15:29:41 2014 -0500
1388
1389     deleted exec, update makefile
1390
1391 commit 822eedf5ba23cabf96d179deffca09906f9dee3b
1392 Merge: 4b67c65 e5f5419
1393 Author: Gabby2212 <gabymelchior22@gmail.com>
1394 Date:   Sun Nov 23 15:25:59 2014 -0500
1395
1396     Solved conflicts
1397
1398 commit 12205c5d9d5a8a427f11cb021bd5ada546f0052d
1399 Merge: 6e6bbd4 06dd046
1400 Author: Gabby2212 <gabymelchior22@gmail.com>
1401 Date:   Sun Nov 23 15:25:59 2014 -0500
1402
1403     Solved conflicts
1404
1405 commit 4b67c657bf1b690f381bf613eff91981c24943f9
1406 Author: Gabby2212 <gabymelchior22@gmail.com>
1407 Date:   Sun Nov 23 15:20:15 2014 -0500
1408
1409     Idk
1410
1411 commit 6e6bbd408f7eeac4ea5b9c9267279d2096f1482c
1412 Author: Gabby2212 <gabymelchior22@gmail.com>
1413 Date:   Sun Nov 23 15:20:15 2014 -0500
1414
```

```
    Idk

commit e5f541931213a5e4f728c7fd04fcc075abaa09bd
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Sun Nov 23 15:15:24 2014 -0500

    restructuring directory/first time making anythin work

commit 06dd0466b6d1868543d8c203a70ca94242424947
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Nov 23 15:15:24 2014 -0500

    restructuring directory/first time making anythin work

commit e2d49b7f1e9ec7a7e6f260b2b93ee79a53a83b85
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 16 15:44:12 2014 -0500

    Crazy make file

commit 7649d58bd1cc8ded7f7b5a226dd2c76c3ae0cf07
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 16 15:44:12 2014 -0500

    Crazy make file

commit 0891c725d9b3f705e13a43ec72e0e47651b158f5
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Sun Nov 16 15:18:14 2014 -0500

    getting rid of extraneous files

commit e3f522647bf5047610de9f6a90a179cd2e87b38b
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Nov 16 15:18:14 2014 -0500

    getting rid of extraneous files

commit cc989bc905307f86c28bc90f3098c2f5e66310a7
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Tue Nov 11 19:53:17 2014 -0500

    working on simple numerical expressions and assignment

commit 27639ff6654be459261df80cb0ca7f50a7ac484a
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Tue Nov 11 19:53:17 2014 -0500

    working on simple numerical expressions and assignment

commit 889b42c91e435e4814323ae8482b41d942913671
```

```
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Tue Nov 11 18:49:44 2014 -0500

    added sast.ml, working on simple expressions

commit efcb9233093e98423555b7ad4a2d4f6bc0489d0b
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Tue Nov 11 18:49:44 2014 -0500

    added sast.ml, working on simple expressions

commit 0ebc60f7494d2c9e67645786a4708aea1c2b69cf
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Nov 11 18:27:47 2014 -0500

    Working on parser

commit 9c3f2fee8422cc22fd46657761f9816cd6bc84d9
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Tue Nov 11 18:27:47 2014 -0500

    Working on parser

commit 369a9cbcf50cfd9170c94a74454630b98c074dad
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Nov 11 18:37:20 2014 -0500

    whitespace

commit a1537ff5cfa7ba68141c3226fd6141e2d2505c1e
Author: Joshua Boggs <joshua.j.boggs@gmail.com>
Date:   Tue Nov 11 18:37:20 2014 -0500

    whitespace

commit 300fb3486cf53dc015c29261b59ac1e34061ddb2
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Tue Nov 11 15:32:46 2014 -0500

    added task list to readme

commit 023e04af80f70bfbcfe3d6a459591fe899947d85
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Tue Nov 11 15:32:46 2014 -0500

    added task list to readme

commit 40902b762dde62abe199d7a1b6b79d25bcc87aba
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Tue Nov 11 15:19:51 2014 -0500

```

```
    reorganizing files

commit 3b88f35a4ef82fd576919e0953e5a27d379833f5
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Tue Nov 11 15:19:51 2014 -0500

    reorganizing files

commit 7fece72bcbd1ce69916c80fe415b0a56321be8ae
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 9 15:42:04 2014 -0500

    Added more to expr statemnts etc in parser

commit 9bee78e69ffc942cb6d56288953169930d5cbc56
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 9 15:42:04 2014 -0500

    Added more to expr statemnts etc in parser

commit d730a0a5e4f36cafad13be8ae7109761fca4895f
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 9 14:00:34 2014 -0500

    Working on parser

commit 0ae8ab51d3e8818e18d9e2f1234eecd46c236bae
Author: Gabby2212 <gabymelchior22@gmail.com>
Date:   Sun Nov 9 14:00:34 2014 -0500

    Working on parser

commit c804215855665246ca287866f81463c9a5248cdc
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 14:00:06 2014 -0500

    Fixed ast

commit a4125817342db6df84d9927050e980d8a7f1e4b0
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 14:00:06 2014 -0500

    Fixed ast

commit 34b9d3de94343ddfad6e5b7c2dccdf8cdd6fd105
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 13:01:11 2014 -0500

    Type fix + scanner fixes

commit 2275801c45e7101b6af3748ce021076a90007720
```

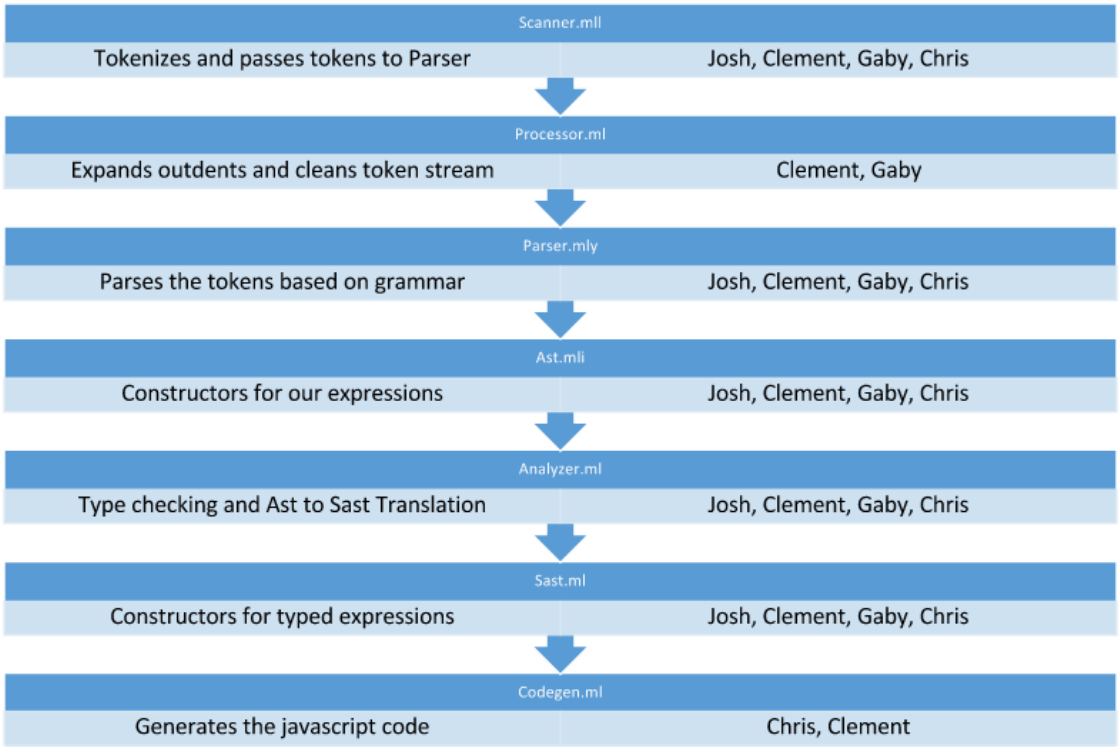
```
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 13:01:11 2014 -0500

    Type fix + scanner fixes

commit 26f3cef89cde0b9cea18bb1bc04c69c68f61c6b5
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 12:38:45 2014 -0500

    Fixed string escape

commit 803962e5ed17ac6b12ec8c709b403526320a1fcf
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 12:38:45 2014 -0500

    Fixed string escape

commit 54d095001931621b9f53cf2f0830e90b39e3dece
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 12:36:15 2014 -0500

    error check to indentation scoping

commit 11602b5c5c891101af91e026ea9402d352eaddb1
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 12:36:15 2014 -0500

    error check to indentation scoping

commit c1f9a7a68ca52b35d2413cd55feec29ff20a0757
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 12:34:05 2014 -0500

    Simplified lexing tree for now

commit 22cae8d6aee74a8f840c7eef963b93cd716c8e58
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 12:34:05 2014 -0500

    Simplified lexing tree for now

commit 18d5daa8e4de275380990c0dae72daab4299b180
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 12:13:34 2014 -0500

    refactor scanner regexes

commit 848565efb7a6365d37b742502159be00d234b2ef
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Nov 9 12:13:34 2014 -0500

```

```
    refactor scanner regexes

commit 23c6b68b01c8a8cd85c4781ec3f39a252fe554ef
Merge: 672ec80 d8a8d6a
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Oct 27 14:00:59 2014 -0400

    Fixed scanner type

commit 499ad9ea6329b3a633ec6c0662610e2a4b0f5784
Merge: ae85b0f 0fa4ee6
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Mon Oct 27 14:00:59 2014 -0400

    Fixed scanner type

commit 672ec801a23c007b544ef083f7c064ab42802ec3
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Sun Oct 26 18:58:11 2014 -0400

    starting ast.ml

commit ae85b0f1ad60452150b6652f0ab3077baa568668
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Oct 26 18:58:11 2014 -0400

    starting ast.ml

commit d8a8d6a26931d9b4a79142c24c3b59c7b60716e2
Author: Clement Robbins <cjr2151@columbia.com>
Date:   Sun Oct 26 18:58:11 2014 -0400

    starting ast.ml

commit 0fa4ee684978279bc56c7502edf9684f90c54057
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Oct 26 18:58:11 2014 -0400

    starting ast.ml

commit b28ea5573dca8b758d10c3904ff0823739fc4bc7
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Oct 26 18:25:38 2014 -0400

    Added modulo to scanner.ml

commit eced64dfdbef38e4e164c715ffa554483bbe3a27
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Sun Oct 26 18:25:38 2014 -0400

    Added modulo to scanner.ml
```

```
1670
1671 commit 319bbddb872791f4dbc9aa8cb91094edab20f2a3
1672 Author: Chris Evans <chris.evans93@gmail.com>
1673 Date:   Sun Oct 26 18:23:49 2014 -0400
1674
1675     Get mad Robbins ;)
1676
1677 commit d03b9aac905f47aa021a4ee3e0a6f72f272d9453
1678 Author: Chris Evans <chris.evans93@gmail.com>
1679 Date:   Sun Oct 26 18:23:49 2014 -0400
1680
1681     Get mad Robbins ;)
1682
1683 commit e05d048d48dda13cecbb68f8280b92bb985f59a8
1684 Author: Clement Robbins <cjr2151@columbia.com>
1685 Date:   Sun Oct 26 18:07:51 2014 -0400
1686
1687     scanner.mll clean up
1688
1689 commit a472947212efecb3207db15e544631e0a80546bd
1690 Author: thebicycle2 <cjr2151@columbia.edu>
1691 Date:   Sun Oct 26 18:07:51 2014 -0400
1692
1693     scanner.mll clean up
1694
1695 commit 7adf225c870cd23d3fd0cd362df146dff7d97e31
1696 Author: Clement Robbins <cjr2151@columbia.com>
1697 Date:   Sun Oct 26 18:02:51 2014 -0400
1698
1699     added eof to scanner
1700
1701 commit 1d6bddd7af81b065d13a262440f56af3407fa9e1
1702 Author: thebicycle2 <cjr2151@columbia.edu>
1703 Date:   Sun Oct 26 18:02:51 2014 -0400
1704
1705     added eof to scanner
1706
1707 commit 60ba4740b6501a81d1ddac8d92041992d80bb828
1708 Author: Clement Robbins <cjr2151@columbia.com>
1709 Date:   Sun Oct 26 17:58:22 2014 -0400
1710
1711     comment on indent parser
1712
1713 commit 7093e5acdd822b0a3caa27fb7ac7393b4a3cd378
1714 Author: thebicycle2 <cjr2151@columbia.edu>
1715 Date:   Sun Oct 26 17:58:22 2014 -0400
1716
1717     comment on indent parser
1718
1719 commit 58f0f51389c0348ccbe8e850d46f795fdb6ff0a8
1720 Author: Clement Robbins <cjr2151@columbia.com>
```

```
Date:   Sun Oct 26 17:57:30 2014 -0400

    initial commit, first try at scanner

commit 111de2968a47f3d81959cef0319c7e42bab966e9
Author: thebicycle2 <cjr2151@columbia.edu>
Date:   Sun Oct 26 17:57:30 2014 -0400

    initial commit, first try at scanner

commit 996fb754947b3b28928cef1a384ec0d8fc927950
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Wed Oct 22 16:51:11 2014 -0400

    README: Initial git setup

commit 6f4c63de1666e4b1c7b2dc5d4b77641d75b3d8fb
Author: Chris Evans <chris.evans93@gmail.com>
Date:   Wed Oct 22 16:46:38 2014 -0400

    Initial commit
```

5 Architectural Design



6 Test Plan

A few different types of tests encompass each part of our language from successful compilation to Javascript and final outputs. We create both unit-type tests and entire code blocks to check full functions, like GCD. Our tests are run through bash scripts. All test and scripts are contained within the 'tests' folder of the project, and this is additionally where javascript and intermediary outputs are placed.

The automated testing suite allowed us to easily locate errors within our code. The log files show which parts of our comparisons to expected outputs came out different, as well as telling the location of improperly formatted

6.1 Scripts

The script 'js.sh' compiles all pumpkin code to the target language so that it can be check for successful compilation and visually analyzed for completeness.

```
#!/bin/bash

#js.sh

yellow='\033[0;33m'
green='\033[0;32m'
red='\033[0;31m'
default='\033[0m' # No Color

echo -e "${yellow}----------------------------------------"
echo -e " Compiling all files to javascript..."
echo -e "----------------------------------------${default}"

for f in *.pk;
do out="$(./../pmkn -c ./$f 2>&1 > ${f%.pk}.js)"
if [[ $out == *"Fatal error"* ]]
then
  echo -e "${red}$out ${default} in $f"
fi
echo -e "Compiled $f to javascript."
done

echo -e "${green}Compilation complete.${default}"
```

The script 'node.js' runs all Javascript files with Node to ensure that the target code runs without errors.

```
#!/bin/bash

#node.js

blue='\033[0;33m'
green='\033[0;32m'
red='\033[0;31m'
default='\033[0m' # No Color

echo -e "${blue}---------------------------------"
echo -e "  Testing all javscript files"
echo -e "---------------------------------${default}"

echo -e "Test Results\n-------------\n" > node.log

for f in *.js;
do echo -en "Testing $f..."
out="$(node ./$f 2>&1)"
if [[ $out == *"Error"* ]]
then
  echo -e $out >> node.log
  echo >> node.log
  echo -e "${red}$out ${default} in $f"
else
  echo -en "  ${green}complete.${default}\n"
fi

done

echo -e "${green}Testing complete.${default}"
```

Ultimately, our test suite is completed with 'runtests.sh' which compiles all pumpkin code with names formatted as 'test-*.pk' to the target language, Javascript. It then runs each of these files in Node, which produces an intermediary output from print statements. This file is checked against a text file with out expected outputs. Any discrepancies aside from whitespace will fire errors, which are output to the console. After all test cases are run, a log file is created that details the results of each test case, including compilation exceptions or discrepencies in the final output. Finally, the intermediary javascript and output files are cleaned from the folder.

```
#!/bin/bash

#runtests.sh

#Set colors for output
yellow='\033[0;33m'
green='\033[0;32m'
red='\033[0;31m'
default='\033[0m'

```

```
# Set time limit for all operations
ulimit -t 30

testlog=runtests.log
rm -f $testlog

error=0
globalerror=0

#Prints errors to console
PrintError() {
    if [ $error -eq 0 ] ; then
      echo -e "${red}FAILED${default}"
      error=1
    fi
    #echo "  $1"
}

# Compare <output> <expected>
Compare() {
    echo diff -Bw $1 $2 1>&2
    diff -Bw "$1" "$2" 1>&2 || {
    PrintError "$1 differs"
    echo -e "FAILED $1 differs from $2" 1>&2
    }
}

# Check <testfile>
Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.pk//'`

    echo -n "$basename..."

    echo 1>&2
    echo "----- $basename results -----" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.out" &&
    generatedfiles="$generatedfiles ${basename}.js" &&
    stdout="$(./../pmkn -c ./${basename}.pk 2>&1 > ${basename}.js)" &&
    node ./${basename}.js > ${basename}.out &&
    Compare ${basename}.out ${basename}.txt

    # Report the status and clean up the generated files
    if [[ $stdout == *"Fatal error"* ]]
    then
      echo -e "${red}$stdout ${default}"
      echo -e "$stdout" 1>&2
```

```
62        echo "---------- FAILURE ----------" 1>&2
63        globalerror=$error
64      elif [ $error -eq 0 ] ; then
65        echo -e "${green}OK${default}"
66        echo "---------- SUCCESS ----------" 1>&2
67      else
68        echo "---------- FAILURE ----------" 1>&2
69        globalerror=$error
70      fi
71      rm -f $generatedfiles
72  }
73
74  echo -e "${yellow}------------------------"
75  echo -e " Testing all files..."
76  echo -e "------------------------${default}"
77
78  echo -e "----------------\n Test Results\n----------------" > $testlog
79
80  shift `expr $OPTIND - 1`
81
82  #Check all files
83  files="test-*.pk"
84  for file in $files
85  do
86    Check $file 2>> $testlog
87  done
88
89  exit $globalerror
```

6.2 Test cases

demo.pk

example1.pk

pipescompo.pk

test-assign.pk

test-binop.pk

test-blockc.pk

test-boolean.pk

test-char.pk

test-ifelse.pk

test-inlinec.pk

test-lists.pk

test-map.pk

test-numbers.pk

test-pipe.pk

test-string.pk

test-tuple.pk

test-unop.pk

6.3 Output Logs

One of the essential tools of our automatic testing suite is output logs, which provide a saved and scannable anlysis of test cases. A sample is provided below:

```
----------------
 Test Results
----------------

------ test-assign results ------
diff -Bw test-assign.out test-assign.txt
---------- SUCCESS ----------

------ test-binop results ------
diff -Bw test-binop.out test-binop.txt
9c9
< -3
---
> 3
FAILED test-binop.out differs from test-binop.txt
---------- FAILURE ----------

------ test-blockc results ------
diff -Bw test-blockc.out test-blockc.txt
---------- SUCCESS ----------

------ test-boolean results ------
diff -Bw test-boolean.out test-boolean.txt
---------- SUCCESS ----------

------ test-char results ------
diff -Bw test-char.out test-char.txt
---------- SUCCESS ----------

------ test-ifelse results ------
diff -Bw test-ifelse.out test-ifelse.txt
---------- SUCCESS ----------
```

```
33
34 ------ test-inlinec results ------
35 diff -Bw test-inlinec.out test-inlinec.txt
36 ---------- SUCCESS ----------
37
38 ------ test-lists results ------
39 diff -Bw test-lists.out test-lists.txt
40 ---------- SUCCESS ----------
41
42 ------ test-map results ------
43 diff -Bw test-map.out test-map.txt
44 ---------- SUCCESS ----------
45
46 ------ test-numbers results ------
47 diff -Bw test-numbers.out test-numbers.txt
48 ---------- SUCCESS ----------
49
50 ------ test-pipe results ------
51 diff -Bw test-pipe.out test-pipe.txt
52 ---------- SUCCESS ----------
53
54 ------ test-string results ------
55 diff -Bw test-string.out test-string.txt
56 ---------- SUCCESS ----------
57
58 ------ test-tuple results ------
59 Fatal error: exception Exceptions.TypeMismatch
60 ---------- FAILURE ----------
61
62 ------ test-unop results ------
63 diff -Bw test-unop.out test-unop.txt
64 ---------- SUCCESS ----------
```

6.4 Colorization

One of the most simple but often overlooked ways to improve automated testing suites is through colorization and formatting. Upon running the test suite, you will find that successful tests output as green in the terminal and failed tests output as red.

7 Lessons Learned

Gaby: It is not easy to know what will be hard to implement until you get simple things out of the way.

Josh: Look towards successful precedents for inspiration and guidance. I wouldn't have been able to complete testing without looking at some good examples.

8 Appendix

8.1 Makefile

```
OBJS = utils.cmo exceptions.cmo scanner.cmo parser.cmo analyzer.cmo processor.
    cmo codegen.cmo pumpkin.cmo

pmkn : $(OBJS)
  ocamlc -o pmkn $(OBJS)

scanner.ml : scanner.mll
  ocamllex scanner.mll

parser.ml parser.mli : parser.mly
  ocamlyacc -v parser.mly

analyzer.cmo : sast.cmo ast.cmi utils.cmo
analyzer.cmx : sast.cmx ast.cmi utils.cmi
exceptions.cmo :
exceptions.cmx :
interpret.cmo : sast.cmo
interpret.cmx : sast.cmx
parser.cmo : ast.cmi parser.cmi
parser.cmx : ast.cmi parser.cmi
processor.cmo : scanner.cmo parser.cmi
processor.cmx : scanner.cmx parser.cmx
pumpkin.cmo : utils.cmo processor.cmo parser.cmi exceptions.cmo \
    analyzer.cmo codegen.cmo
pumpkin.cmx : utils.cmx processor.cmx parser.cmx exceptions.cmx \
    analyzer.cmx codegen.cmx
sast.cmo : ast.cmi
sast.cmx : ast.cmi
scanner.cmo : parser.cmi exceptions.cmo
scanner.cmx : parser.cmx exceptions.cmx
utils.cmo : sast.cmo parser.cmi ast.cmi
utils.cmx : sast.cmx parser.cmx ast.cmi
codegen.cmo : sast.cmo parser.cmi ast.cmi exceptions.cmo
codegen.cmx : sast.cmx parser.cmx ast.cmi exceptions.cmx
ast.cmi :
parser.cmi : ast.cmi

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

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

.PHONY : clean
clean :
  rm -f pmkn parser.ml parser.mli scanner.ml \
```

```
  *.cmo *.cmi *.out *.diff *.orig *.output
```

8.2 demo.pk

```
def reduce(func: (Int, Int => Int), acc: Int, l: List[Int]): Int =>
  if(is_empty(l)):
    acc
  else:
    reduce(func, func(hd(l), acc), tl(l))

def map(f: (Int => Int), l: List[Int]): List[Int] =>
  if(is_empty(l)):
    l
  else:
    f(hd(l))::(map(f, tl(l)))

def even(n: Int): Bool =>
  if(n % 2 is 0):
    True
  else:
    False

val x = [1,2,3,4] |> map((x:Int => x + 5 :Int)) |> reduce((x: Int, y: Int => x
    + y : Int), 0) |> even

print("Example 2:")
print(x)

print("\nExample 1:")
def gcd(a : Int, b : Int) : Int =>
  if(b is 0):
    a
  else:
    gcd(b, a % b)

def relativePrimes(a: Int) =>
  if (a is 1):
    True
  else:
    False

val p = relativePrimes <<  gcd

if(p(25, 15)):
  print("You have relative primes")
else:
  print("Not relative primes")
```

8.3 printalloutput.py

```
from subprocess import call
import os

tests = [os.path.join("tests/", test) for test in os.listdir("tests/")]

for test in tests:
    print "Testings : " + test
    print "Tokens :"
    print
    call (['./pmkn', '-t', test])
    print
    print "Ast :"
    print
    call (['./pmkn', '-a', test])
    print
```

8.4 example1.pk

```
def reduce(func: (Int, Int => Int), acc: Int, l: List[Int]): Int =>
  if(is_empty(l)):
    acc
  else:
    reduce(func, func(hd(l), acc), tl(l))

def map(f: (Int => Int), l: List[Int]): List[Int] =>
  if(is_empty(l)):
    l
  else:
    f(hd(l))::(map(f, tl(l)))

def even(n: Int): Bool =>
  if(n % 2 is 0):
    True
  else:
    False

val x = [1,2,3,4] |> map((x:Int => x + 5 :Int)) |> reduce((x: Int, y: Int => x
    + y : Int), 0) |> even
print(x)
```

8.5 pipescompo.pk

```
def gcd(a : Int, b : Int) : Int =>
  if(b is 0):
    a
  else:
    gcd(b, a % b)

def relativePrimes(a: Int) =>
  if (a is 1):
```

```
    True
  else:
    False

val p = relativePrimes <<  gcd

if(p(25, 15)):
  print("You have relative primes")
else:
  print("Not relative primes")
```

8.6 test-assign.pk

```
val x = 1
print(x)
val y : Bool = True
print(y)
print(x + 2)
print(y || False)
```

8.7 test-assign.txt

```
1
true
3
true
```

8.8 test-binop.pk

```
print(3 + 4)
print(3 - 4)
print(3 * 4)
print(3 / 4)
print(3 % 4)
print(15 + --5)
print(15 - -5)
print(15 * --5)
print(15 / -5)
print(3 == 4)
print(3 != 4)
print(3 > 4)
print(3 < 4)
print(3 <= 4)
print(3 >= 4)

print(3.5 + 4.2)
print(3.5 - 4.5)
print(3.5 * 4.5)
print(3.5 / 4.0)
```

```
print(3.5 % 4.5)
print(3.5 == 4.5)
print(3.5 != 4.5)
print(3.5 > 4.5)
print(3.5 < 4.5)
print(3.5 <= 4.5)
print(3.5 >= 4.5)

print("test1" + "test2")
print("test1" * "test2")
print("test1" == "test2")
print("test1" != "test2")
print("test1" > "test2")
print("test1" < "test2")
print("test1" <= "test2")
print("test1" >= "test2")

print('t' + 'b')
print('t' * 'b')
print('t' == 'b')
print('t' != 'b')
print('t' > 'b')
print('t' < 'b')
print('t' <= 'b')
print('t' >= 'b')

print(True && False)
print(True || False)
print(True == False)
print(False != False)

print(2::[3, 4])
```

8.9 test-binop.txt

```
7
-1
12
0.75
3
20
20
75
3
false
true
false
true
true
false

```

```
7.7
-1
15.75
0.875
3.5
false
true
false
true
true
false

test1test2
NaN
false
true
false
true
true
false

tb
NaN
false
true
true
false
false
true

false
true
false
false

[2,3,4]
```

8.10 test-blockc.pk

```
/*
    This is a multiline comment.
    print("no")
*/
print("yes")
```

8.11 test-blockc.txt

```
yes
```

8.12 test-boolean.pk

```
print(1 is 2 and 3 != 4)
print(1 == 1 && !True)
print(not False or !True)
print(1 < 2 || 1 > 2)
print(1 <= 2 || 1 >= 2)
val x : Bool = True
print(x)
val y = False
print(y)
```

8.13 test-boolean.txt

```
false
false
true
true
true
true
false
```

8.14 test-char.pk

```
print('c')
val d = 'd'
print(d)
```

8.15 test-char.txt

```
c
d
```

8.16 test-ifelse.pk

```
if 3 > 2 :
    if True :
        if False :
            print(1)
        else :
            if True :
                print(2)
            else :
                print(3)
```

8.17 test-ifelse.txt

```
2
```

8.18 test-inlinec.pk

```
//this is an inline comments
// print("no")
print("yes")
```

8.19 test-inlinec.txt

```
yes
```

8.20 test-lists.pk

```
val x : List[Int] = [1, 2, 3, 4]
print(x)
val i = 2
[1]

print(x[1+i])
val y = 1::x
print(y)

val z = True::[]
print(z)

def map (f : (Int => Int), l : List[Int]) : List[Int] =>
    val head = hd(l)
    val tail = tl(l)

    f(head)::map(f, tail)
```

8.21 test-lists.txt

```
[ 1, 2, 3, 4 ]
4
[ 1, 1, 2, 3, 4 ]
[ true ]
```

8.22 test-map.pk

```
val m : Map[String, String] = ("123" -> "abc", "456" -> "def")
print(m("123"))
m = ("789" -> "ghi")
print(m("789"))
```

8.23 test-map.txt

```
abc
ghi
```

8.24 test-numbers.pk

print(1) print(2.1) print(-1) print(-2.1) print(–5)

8.25 test-numbers.txt

```
1
2.1
−1
−2.1
5
```

8.26 test-pipe.pk

```
val x = [1,2,3] |> (a: List[Int] => len(a)%2)
if x is 0:
  print("Even")
else:
  print("Odd")
```

8.27 test-pipe.txt

```
Odd
```

8.28 test-string.pk

```
print("absdla\\")
val s : String = "ab\ncd"
print(s)
val z : String = 'c' + 'c'
print(z)
val w = "ab" + "cd"
print(w)
val x = 'c' + "abcde"
print(x)
```

8.29 test-string.txt

```
absdla\
ab
cd
cc
abcd
cabcde
```

8.30 test-tuple.pk

```
val x : Tuple[Int] = (1,)
val y = (1, 2, 3)
val z = (1, 2, 3,)
val p = ("hello", 2, 3,)
val a = x$0
val b = z$1 + 1
val c = p$1 + y$2
val d = p$0 + p$2
print(a)
print(b)
print(c)
print(d)
```

8.31 test-tuple.txt

```
1
666
5
```

8.32 test-unop.pk

```
print(+3)
print(-3.0)
print(!True)
```

8.33 test-unop.txt

```
3
-3
false
```

8.34 analyzer.ml

```
open Sast
open Ast
open Utils

module Env = Map.Make(String)

let env_to_string id t =
  print_string(id ^ " -> " ^ a_type_to_string t ^ "\n")

let get_func_return_type = function
  Function(_, t) -> t
  | _ -> raise(Exceptions.PipingIntoNonFunc)

```

```
let get_func_params = function
  Function(t, _) -> (List.rev t)
  | _ -> raise(Exceptions.PipingIntoNonFunc)

let filter_params (op, num_gp) =
  let rec sublist i l =
    match l with
      [] -> []
    | h :: t -> if (i = 0) then h::t else sublist (i - 1) t
  in let new_params = sublist num_gp op in
  List.rev new_params

let rec aType_to_sType = function
    TInt -> Int
  | TFloat -> Float
  | TBool -> Bool
  | TString -> String
  | TChar -> Char
  | TUnit -> Unit
  | TTuple(t) -> Tuple((List.map aType_to_sType t))
  | TList(t) -> List(aType_to_sType t)
  | TMap(t1, t2) -> Map(aType_to_sType t1, aType_to_sType t2)
  | TFunction(t1, t2) -> Function(List.map aType_to_sType t1, aType_to_sType
    t2)

let rec type_of = function
    AIntLiteral(_) -> Int
  | AFloatLiteral(_) -> Float
  | ABoolLiteral(_) -> Bool
  | AStringLiteral(_) -> String
  | ACharLiteral(_) -> Char
  | AUnitLiteral -> Unit
  | ATupleLiteral(_, t) -> t
  | AListLiteral(_, t) -> t
  | AMapLiteral(_, t) -> t
  | AIdLiteral(_, t) -> t
  | ABinop(_, _, _, t) -> t
  | AUnop(_, _, t) -> t
  | AAssign(_, _, t) -> t
  | AReassign(_, _, t) -> t
  | ATupleAccess(_, _, t) -> t
  | AListAccess(_, _, t) -> t
  | AIfBlock(_, _, t) -> t
  | AIfElseBlock(_, _, _, t) -> t
  | AFuncCall(_, _, t) -> t
  | AFuncDecl(_, _, _, t) -> t
  | AFuncAnon(_, _, t) -> t
  | AFuncComposition (_, _, t) -> t
  | AFuncPiping(_, _, t) -> t
  | AMapAccess(_, _, t) -> t

```

```
(* Auxiliary functions for type checks *)
let rec evaluate_index = function
    ABinop(e1, op, e2, Int) ->
      (match op with
        Plus -> (evaluate_index e1) + (evaluate_index e2)
      | Minus -> (evaluate_index e1) - (evaluate_index e2)
      | Times -> (evaluate_index e1) * (evaluate_index e2)
      | Divide -> (evaluate_index e1) / (evaluate_index e2)
      | Modulo -> (evaluate_index e1) mod (evaluate_index e2)
      | _ -> raise(Exceptions.InvalidIndexing("Invalid operation in index")))
  | AUnop(op, e1, Int) ->
      (match op with
        Plus -> evaluate_index e1
      | _ -> raise(Exceptions.InvalidIndexing("Invalid operation in index")))
  | AIntLiteral(n) -> n
  | AIdLiteral(_, _) -> raise(Exceptions.InvalidIndexing("Cannot use variables
    for tuple index"))
  | _ -> raise(Exceptions.InvalidIndexing("Invalid expression in index"))

let check_reserved_functions (id, params, env) =
  if (List.length params) = 1 then
  let t_p = type_of (List.hd params) in
  match t_p with
    List(t) ->
      (match id with
      AIdLiteral(i, _)->
      if i = "hd" then AFuncCall(id, params, t)
      else if i = "tl" then AFuncCall(id, params, t_p)
      else if i = "len" then AFuncCall(id, params, Int)
      else if i = "is_empty" then AFuncCall(id, params, Bool)
      else raise(Exceptions.UnimplementedCallType(1))
      | _ -> raise(Exceptions.UnimplementedCallType(2)))
  | Unit ->
    (match id with
    AIdLiteral(i, _)->
    if i = "len" then AFuncCall(id, params, Function([List(Int)], Int))
    else if i = "is_empty" then AFuncCall(id, params, Function([List(Int)],
   Bool))
    else raise(Exceptions.UnimplementedCallType(1))
    | _ -> raise(Exceptions.UnimplementedCallType(2)))
  |_ -> raise(Exceptions.UnimplementedCallType(3))
  else raise(Exceptions.UnimplementedCallType(4))

let valid_binop (t1, t2, op) =
  if op = Cons then
    match t2 with
    List(t) ->
    if t = Unit then List(t1)
    else if t<>t1 then raise(Exceptions.InvalidOperation(operation_to_string
   op))
    else t2
```

```
112     | _ -> raise(Exceptions.InvalidOperation(operation_to_string op))
113   else
114   match t1 with
115   Int | Float ->
116   if t1 <> t2 then raise(Exceptions.TypeMismatch)
117   else if op = And || op = Or || op = Cons then
118     raise(Exceptions.InvalidOperation(operation_to_string op))
119   else if op = Eq || op = Neq || op = Gt || op = Lt || op = Gte || op = Lte
      then Bool
120   else t1
121   | String | Char ->
122   if op = Plus then if t2 <> String && t2 <> Char then raise(Exceptions.
      TypeMismatch) else String
123   else if t1 <> t2 then raise(Exceptions.TypeMismatch)
124   else if op = Minus || op = Divide || op = Modulo || op = And || op = Or ||
      op = Cons then
125     raise(Exceptions.InvalidOperation(operation_to_string op))
126   else if op = Eq || op = Neq || op = Gt || op = Lt || op = Gte || op = Lte
      then Bool
127   else t1
128   | Bool ->
129   if t1 <> t2 then raise(Exceptions.TypeMismatch)
130   else if op <> Eq && op <> Neq && op <> And && op <> Or then
131     raise(Exceptions.InvalidOperation(operation_to_string op))
132   else t1
133   | _ -> raise(Exceptions.UnimplementedOperation(operation_to_string op,
      a_type_to_string t1))
134
135 let valid_unop (op, t) =
136   match t with
137   Int | Float ->
138   if op = Not then
139     raise(Exceptions.InvalidOperation(operation_to_string op))
140   | Bool ->
141   if op <> Not then
142     raise(Exceptions.InvalidOperation(operation_to_string op))
143   | _ -> raise(Exceptions.UnimplementedOperation(operation_to_string op,
      a_type_to_string t))
144
145 let annotate_parameter (id, t) = (id, aType_to_sType t)
146
147 let rec match_expression_list_type = function
148     fst::snd::tail ->
149       if type_of fst <> type_of snd then
150         false
151       else
152         match_expression_list_type (snd::tail)
153   | _ -> true
154
155 let rec annotate_expression env = function
156     IntLiteral(n) -> AIntLiteral(n), env
```

```
157   | FloatLiteral(f) -> AFloatLiteral(f), env
158   | BoolLiteral(b) -> ABoolLiteral(b), env
159   | StringLiteral(s) -> AStringLiteral(s), env
160   | CharLiteral(c) -> ACharLiteral(c), env
161   | UnitLiteral -> AUnitLiteral, env
162   | IdLiteral(id) ->
163       if Env.mem id env then
164         let t = Env.find id env in
165         AIdLiteral(id, t), env
166       else
167         raise (Exceptions.IDNotFound id)
168   | TupleLiteral(e_list) ->
169     let a_e_list, env = annotate_expression_list env e_list in
170     let t = List.map type_of a_e_list in
171     ATupleLiteral(a_e_list, Tuple(t)), env
172   | ListLiteral(e_list) ->
173       let a_e_list, env = annotate_expression_list env (List.rev e_list) in
174       if match_expression_list_type a_e_list then
175         if List.length a_e_list = 0 then
176           AListLiteral(a_e_list, List(Unit)), env
177         else
178           let t = type_of (List.hd a_e_list) in
179           AListLiteral(a_e_list, List(t)), env
180       else
181         raise Exceptions.TypeMismatch
182   | MapLiteral(elist) ->
183     let helper(expr1, expr2) =
184       let key, env = annotate_expression env expr1 in
185       let value, env = annotate_expression env expr2 in
186         (key, value)
187     in
188     let s_map = List.map helper (List.rev elist) in
189     let key_list = List.map (fun (expr1, expr2) -> expr1) s_map and
190     value_list = List.map (fun (expr1, expr2) -> expr2) s_map in
191     if not(match_expression_list_type key_list && match_expression_list_type
    value_list) then
192       raise(Exceptions.TypeMismatch)
193     else
194     let k_type = type_of (List.hd key_list) in
195     if(k_type = Int || k_type = String || k_type = Float || k_type = Char)
    then
196       AMapLiteral(s_map, Map((type_of (List.hd key_list)), (type_of (List.hd
    value_list)))), env
197     else raise(Exceptions.InvalidMapKeyType)
198   | TypedAssign(id, e, t) ->
199       if Env.mem id env then
200         raise (Exceptions.NameCollision(id))
201       else
202         let a_e, env = annotate_expression env e in
203         let t_a_e = type_of a_e in
204         let a_t = aType_to_sType t in
```

```
205         if t_a_e <> a_t then
206           raise (Exceptions.TypeMismatch)
207         else
208           let env = Env.add id t_a_e env in
209           AAssign(id, a_e, t_a_e), env
210   | Assign(id, e) ->
211       if Env.mem id env then
212         raise (Exceptions.NameCollision(id))
213       else
214         let a_e, env = annotate_expression env e in
215         let t_a_e = type_of a_e in
216         let env = Env.add id t_a_e env in
217         AAssign(id, a_e, t_a_e), env
218   | Reassign(id, e) ->
219     if Env.mem id env then
220       let t = Env.find id env in
221       let a_e, env = annotate_expression env e in
222       let t_a_e = type_of a_e in
223       if t = t_a_e then AReassign(id, a_e, t), env
224       else raise(Exceptions.TypeMismatch)
225     else raise(Exceptions.IDNotFound(id))
226   | Binop(e1, op, e2) ->
227     let ae1, env = annotate_expression env e1 in
228     let ae2, env = annotate_expression env e2 in
229     let t = valid_binop((type_of ae1), (type_of ae2), op) in
230     ABinop(ae1, op, ae2, t), env
231   | Unop(op, e) ->
232     let ae, env = annotate_expression env e in
233     let et = type_of ae in
234     valid_unop(op, et);
235     AUnop(op, ae, et), env
236   | ListAccess(e, index) ->
237     let ae, env = annotate_expression env e in
238     let ind, env = annotate_expression env index in
239     let ae_t = type_of ae and
240     ind_t = type_of ind in
241     (
242       match ae_t with
243           List(t) ->
244           (
245             match ind_t with
246                 Int -> AListAccess(ae, ind, t), env
247               | _ -> raise(Exceptions.InvalidIndexing(a_type_to_string ind_t))
248           )
249         | _ -> raise(Exceptions.InvalidIndexing(a_type_to_string ae_t))
250     )
251   | TupleAccess(e, index) ->
252     let ae, env = annotate_expression env e in
253     let ind, env = annotate_expression env index in
254     let ae_t = type_of ae in
255     (
```

```
256       match ae_t with
257         Sast.Tuple(t) ->
258           let ind_n = evaluate_index(ind) in
259           if ind_n >= 0 then
260             if ind_n > (List.length t) then raise(Exceptions.ArrayOutOfBounds)
261             else
262             let param_type = List.nth t ind_n in
263             ATupleAccess(ae, ind, param_type), env
264           else raise(Exceptions.InvalidIndexing("Negative index"))
265         | _ -> raise(Exceptions.InvalidIndexing(a_type_to_string ae_t))
266     )
267   | IfBlock(e, e_list) ->
268     let a_list, _ = annotate_expression_list env e_list in
269     let ae, _ = annotate_expression env e in
270     let ae_s_type = type_of ae in
271     if ae_s_type = Bool then
272       AIfBlock(ae, a_list, Unit), env
273     else raise(Exceptions.IfRequiresBool(a_type_to_string ae_s_type))
274   | IfElseBlock(e1, l1, l2) ->
275     let ae, tempEnv = annotate_expression env e1 in
276     let a_list1, _ = annotate_expression_list env l1 in
277     let a_list2, _ = annotate_expression_list env l2 in
278     let le1 = (List.hd (List.rev a_list1)) in
279     let le2 = (List.hd (List.rev a_list2)) in
280     let ae_s_type = type_of ae in
281     let le1_s_type = type_of le1 and
282     le2_s_type = type_of le2 in
283     if ae_s_type <> Sast.Bool then
284       raise (Exceptions.IfRequiresBool(a_type_to_string ae_s_type))
285     else if le1_s_type <> le2_s_type then
286       raise (Exceptions.TypeMismatch)
287     else AIfElseBlock(ae, a_list1, a_list2, le1_s_type), env
288   | TypedFuncDecl(id, params, code, t) ->
289     if Env.mem id env then
290         raise (Exceptions.NameCollision(id))
291     else
292     let s_params = List.map annotate_parameter (List.rev params) in
293     let param_types = List.map (fun (i, t) -> t) s_params in
294     let env = Env.add id (Function(param_types, (aType_to_sType t))) env in
295     let tempEnv =  List.fold_left (fun cenv p -> (Env.add (fst p) (snd p) cenv
    )) env s_params in
296     let s_code, tempEnv = annotate_expression_list tempEnv code in
297     let le = (List.hd (List.rev s_code)) in
298     let le_s_type = type_of le in
299     let s_type = aType_to_sType t in
300     if le_s_type <> s_type && s_type <> Unit then
301       raise (Exceptions.TypeMismatch)
302     else
303       AFuncDecl(id, s_params, s_code, Function(param_types, s_type)), env
304   | FuncDecl(id, params, code) ->
305     if Env.mem id env then
```

```
306         raise (Exceptions.NameCollision(id))
307       else
308       let s_params = List.map annotate_parameter (List.rev params) in
309       let param_types = List.map (fun (i, t) -> t) s_params in
310       let tempEnv =  List.fold_left (fun cenv p -> (Env.add (fst p) (snd p) cenv
      )) env s_params in
311       let s_code, tempEnv = annotate_expression_list tempEnv code in
312       let le = (List.hd (List.rev s_code)) in
313       let le_s_type = type_of le in
314       let env = Env.add id (Function(param_types, le_s_type)) env in
315       AFuncDecl(id, s_params, s_code, Function(param_types, le_s_type)), env
316     | TypedFuncAnon(params, exp, t) ->
317       let s_params = List.map annotate_parameter (List.rev params) in
318       let param_types = List.map (fun (i, t) -> t) s_params in
319       let tempEnv =  List.fold_left (fun cenv p -> (Env.add (fst p) (snd p) cenv
      )) env s_params in
320       let s_e, tempEnv = annotate_expression tempEnv exp in
321       let s_e_type = type_of s_e in
322       let s_type = aType_to_sType t in
323       if s_e_type <> s_type && s_type <> Unit then
324         raise (Exceptions.TypeMismatch)
325       else
326         AFuncAnon(s_params, s_e, Function(param_types, s_type)), env
327     | FuncAnon(params, exp) ->
328       let s_params = List.map annotate_parameter params in
329       let param_types = List.map (fun (i, t) -> t) s_params in
330       let tempEnv =  List.fold_left (fun cenv p -> (Env.add (fst p) (snd p) cenv
      )) env s_params in
331       let s_e, tempEnv = annotate_expression tempEnv exp in
332       let s_e_type = type_of s_e in
333       AFuncAnon(s_params, s_e, Function(param_types, s_e_type)), env
334     | Call(e1, params) ->
335       let s_params, tempEnv = annotate_expression_list env (List.rev params) in
336       let id, env = annotate_expression env e1 in
337       let t = type_of id in
338       (match t with
339         Function(p, rt) ->
340         if rt = Reserved then check_reserved_functions(id, s_params, env), env
341         else if rt = Print then AFuncCall(id, s_params, Print), env else
342         let n_params = List.length p in
343         let sn_params = List.length s_params in
344         let rec match_types l1 l2 =
345         match l1 with
346           [] -> true
347         | hd::tl -> if (List.length l2 > 0 ) && (type_of hd) = (List.hd l2) then
348           match_types tl (List.tl l2) else if (type_of hd = Unit) then true else
      false
349         in
350         let s_type =
351           if not(match_types (List.rev s_params) p) then
352             raise(Exceptions.WrongParameterType(aexpression_to_string id))
```

```
353         else if sn_params <> n_params then Function((filter_params (p,
    sn_params)), rt)
354         else rt in
355       AFuncCall(id, s_params, s_type), env
356     | Map(kt, vt) ->
357       if (List.length s_params) = 1 then
358         let key = List.hd s_params in
359         if (kt = (type_of key)) then AMapAccess(id, key, vt), env
360         else raise(Exceptions.TypeMismatch)
361       else raise(Exceptions.InvalidIndexing(aexpression_to_string id))
362     | _ -> raise(Exceptions.UnimplementedCallType(111)))
363   | FuncComposition(exp1, exp2) ->
364     let ae1, env = annotate_expression env exp1 in
365     let ae2, env = annotate_expression env exp2 in
366     let t1 = type_of ae1 in
367     let t2 = type_of ae2 in
368     let params = get_func_params t2 in
369     if (List.length params) > 1 then raise(Exceptions.
    ComposedIntermediateTakesMultipleArguments)
370     else
371     let p_type = List.hd(params) in
372     let r_type = get_func_return_type t1 in
373     if(p_type <> r_type) then raise(Exceptions.TypeMismatch)
374     else
375     let nr_type = get_func_return_type t2 in
376     let n_params = get_func_params t1 in
377     AFuncComposition(ae1, ae2, Function(n_params, nr_type)), env
378   | FuncPipe(exp1, exp2) ->
379     let ae1, env = annotate_expression env exp1 in
380     let ae2, env = annotate_expression env exp2 in
381     let t1 = type_of ae1 in
382     let t2 = type_of ae2 in
383     let params = get_func_params t2 in
384     if (List.length params) > 1 then raise(Exceptions.
    ComposedIntermediateTakesMultipleArguments)
385     else
386     let p_type = List.hd(params) in
387     if(p_type <> t1) then raise(Exceptions.TypeMismatch)
388     else
389     let nr_type = get_func_return_type t2 in
390     AFuncPiping(ae1, ae2, nr_type), env
391
392 and annotate_expression_list env e_list =
393   let env_ref = ref(env) in
394   let rec helper = function
395       head::tail ->
396         let a_head, env = annotate_expression !env_ref head in
397         env_ref := env;
398         a_head::(helper tail)
399     | [] -> []
400   in (helper e_list), !env_ref
```

```
401
402 let annotate_program expression_list : Sast.aRoot =
403   let env = Env.empty in
404   let env = Env.add "print" (Function([Unit], Print)) env in
405   let env = Env.add "hd" (Function([Unit], Reserved)) env in
406   let env = Env.add "tl" (Function([Unit], Reserved)) env in
407   let env = Env.add "len" (Function([Unit], Reserved)) env in
408   let env = Env.add "is_empty" (Function([Unit], Reserved)) env in
409   let a_expression_list, env = annotate_expression_list env expression_list in
410   a_expression_list
```

8.35 analyzer.ml

```
1  open Sast
2  open Ast
3  open Utils
4
5  module Env = Map.Make(String)
6
7  let env_to_string id t =
8    print_string(id ^ " -> " ^ a_type_to_string t ^ "\n")
9
10 let get_func_return_type = function
11   Function(_, t) -> t
12   | _ -> raise(Exceptions.PipingIntoNonFunc)
13
14 let get_func_params = function
15   Function(t, _) -> (List.rev t)
16   | _ -> raise(Exceptions.PipingIntoNonFunc)
17
18 let filter_params (op, num_gp) =
19   let rec sublist i l =
20     match l with
21       [] -> []
22     | h :: t -> if (i = 0) then h::t else sublist (i - 1) t
23   in let new_params = sublist num_gp op in
24   List.rev new_params
25
26 let rec aType_to_sType = function
27     TInt -> Int
28   | TFloat -> Float
29   | TBool -> Bool
30   | TString -> String
31   | TChar -> Char
32   | TUnit -> Unit
33   | TTuple(t) -> Tuple((List.map aType_to_sType t))
34   | TList(t) -> List(aType_to_sType t)
35   | TMap(t1, t2) -> Map(aType_to_sType t1, aType_to_sType t2)
36   | TFunction(t1, t2) -> Function(List.map aType_to_sType t1, aType_to_sType
       t2)
37
```

```
let rec type_of = function
    AIntLiteral(_) -> Int
  | AFloatLiteral(_) -> Float
  | ABoolLiteral(_) -> Bool
  | AStringLiteral(_) -> String
  | ACharLiteral(_) -> Char
  | AUnitLiteral -> Unit
  | ATupleLiteral(_, t) -> t
  | AListLiteral(_, t) -> t
  | AMapLiteral(_, t) -> t
  | AIdLiteral(_, t) -> t
  | ABinop(_, _, _, t) -> t
  | AUnop(_, _, t) -> t
  | AAssign(_, _, t) -> t
  | AReassign(_, _, t) -> t
  | ATupleAccess(_, _, t) -> t
  | AListAccess(_, _, t) -> t
  | AIfBlock(_, _, t) -> t
  | AIfElseBlock(_, _, _, t) -> t
  | AFuncCall(_, _, t) -> t
  | AFuncDecl(_, _, _, t) -> t
  | AFuncAnon(_, _, t) -> t
  | AFuncComposition (_, _, t) -> t
  | AFuncPiping(_, _, t) -> t
  | AMapAccess(_, _, t) -> t

(* Auxiliary functions for type checks *)
let rec evaluate_index = function
    ABinop(e1, op, e2, Int) ->
      (match op with
        Plus -> (evaluate_index e1) + (evaluate_index e2)
      | Minus -> (evaluate_index e1) - (evaluate_index e2)
      | Times -> (evaluate_index e1) * (evaluate_index e2)
      | Divide -> (evaluate_index e1) / (evaluate_index e2)
      | Modulo -> (evaluate_index e1) mod (evaluate_index e2)
      | _ -> raise(Exceptions.InvalidIndexing("Invalid operation in index")))
  | AUnop(op, e1, Int) ->
      (match op with
        Plus -> evaluate_index e1
      | _ -> raise(Exceptions.InvalidIndexing("Invalid operation in index")))
  | AIntLiteral(n) -> n
  | AIdLiteral(_, _) -> raise(Exceptions.InvalidIndexing("Cannot use variables
    for tuple index"))
  | _ -> raise(Exceptions.InvalidIndexing("Invalid expression in index"))

let check_reserved_functions (id, params, env) =
  if (List.length params) = 1 then
  let t_p = type_of (List.hd params) in
  match t_p with
    List(t) ->
      (match id with
```

```
88        AIdLiteral(i, _)->
89        if i = "hd" then AFuncCall(id, params, t)
90        else if i = "tl" then AFuncCall(id, params, t_p)
91        else if i = "len" then AFuncCall(id, params, Int)
92        else if i = "is_empty" then AFuncCall(id, params, Bool)
93        else raise(Exceptions.UnimplementedCallType(1))
94        | _ -> raise(Exceptions.UnimplementedCallType(2)))
95    | Unit ->
96      (match id with
97      AIdLiteral(i, _)->
98      if i = "len" then AFuncCall(id, params, Function([List(Int)], Int))
99      else if i = "is_empty" then AFuncCall(id, params, Function([List(Int)],
      Bool))
100     else raise(Exceptions.UnimplementedCallType(1))
101     | _ -> raise(Exceptions.UnimplementedCallType(2)))
102   |_ -> raise(Exceptions.UnimplementedCallType(3))
103   else raise(Exceptions.UnimplementedCallType(4))
104
105 let valid_binop (t1, t2, op) =
106   if op = Cons then
107     match t2 with
108     List(t) ->
109     if t = Unit then List(t1)
110     else if t<>t1 then raise(Exceptions.InvalidOperation(operation_to_string
      op))
111     else t2
112     | _ -> raise(Exceptions.InvalidOperation(operation_to_string op))
113   else
114   match t1 with
115   Int | Float ->
116   if t1 <> t2 then raise(Exceptions.TypeMismatch)
117   else if op = And || op = Or || op = Cons then
118     raise(Exceptions.InvalidOperation(operation_to_string op))
119   else if op = Eq || op = Neq || op = Gt || op = Lt || op = Gte || op = Lte
      then Bool
120   else t1
121   | String | Char ->
122   if op = Plus then if t2 <> String && t2 <> Char then raise(Exceptions.
      TypeMismatch) else String
123   else if t1 <> t2 then raise(Exceptions.TypeMismatch)
124   else if op = Minus || op = Divide || op = Modulo || op = And || op = Or ||
      op = Cons then
125     raise(Exceptions.InvalidOperation(operation_to_string op))
126   else if op = Eq || op = Neq || op = Gt || op = Lt || op = Gte || op = Lte
      then Bool
127   else t1
128   | Bool ->
129   if t1 <> t2 then raise(Exceptions.TypeMismatch)
130   else if op <> Eq && op <> Neq && op <> And && op <> Or then
131     raise(Exceptions.InvalidOperation(operation_to_string op))
132   else t1
```

```
133   | _ -> raise(Exceptions.UnimplementedOperation(operation_to_string op,
      a_type_to_string t1))
134
135 let valid_unop (op, t) =
136   match t with
137   Int | Float ->
138   if op = Not then
139     raise(Exceptions.InvalidOperation(operation_to_string op))
140   | Bool ->
141   if op <> Not then
142     raise(Exceptions.InvalidOperation(operation_to_string op))
143   | _ -> raise(Exceptions.UnimplementedOperation(operation_to_string op,
      a_type_to_string t))
144
145 let annotate_parameter (id, t) = (id, aType_to_sType t)
146
147 let rec match_expression_list_type = function
148     fst::snd::tail ->
149       if type_of fst <> type_of snd then
150         false
151       else
152         match_expression_list_type (snd::tail)
153   | _ -> true
154
155 let rec annotate_expression env = function
156     IntLiteral(n) -> AIntLiteral(n), env
157   | FloatLiteral(f) -> AFloatLiteral(f), env
158   | BoolLiteral(b) -> ABoolLiteral(b), env
159   | StringLiteral(s) -> AStringLiteral(s), env
160   | CharLiteral(c) -> ACharLiteral(c), env
161   | UnitLiteral -> AUnitLiteral, env
162   | IdLiteral(id) ->
163       if Env.mem id env then
164         let t = Env.find id env in
165         AIdLiteral(id, t), env
166       else
167         raise (Exceptions.IDNotFound id)
168   | TupleLiteral(e_list) ->
169     let a_e_list, env = annotate_expression_list env e_list in
170     let t = List.map type_of a_e_list in
171     ATupleLiteral(a_e_list, Tuple(t)), env
172   | ListLiteral(e_list) ->
173       let a_e_list, env = annotate_expression_list env (List.rev e_list) in
174       if match_expression_list_type a_e_list then
175         if List.length a_e_list = 0 then
176           AListLiteral(a_e_list, List(Unit)), env
177         else
178           let t = type_of (List.hd a_e_list) in
179           AListLiteral(a_e_list, List(t)), env
180       else
181         raise Exceptions.TypeMismatch
```

```
182   | MapLiteral(elist) ->
183     let helper(expr1, expr2) =
184       let key, env = annotate_expression env expr1 in
185       let value, env = annotate_expression env expr2 in
186         (key, value)
187     in
188     let s_map = List.map helper (List.rev elist) in
189     let key_list = List.map (fun (expr1, expr2) -> expr1) s_map and
190     value_list = List.map (fun (expr1, expr2) -> expr2) s_map in
191     if not(match_expression_list_type key_list && match_expression_list_type
      value_list) then
192       raise(Exceptions.TypeMismatch)
193     else
194     let k_type = type_of (List.hd key_list) in
195     if(k_type = Int || k_type = String || k_type = Float || k_type = Char)
      then
196       AMapLiteral(s_map, Map((type_of (List.hd key_list)), (type_of (List.hd
      value_list)))), env
197     else raise(Exceptions.InvalidMapKeyType)
198   | TypedAssign(id, e, t) ->
199       if Env.mem id env then
200         raise (Exceptions.NameCollision(id))
201       else
202         let a_e, env = annotate_expression env e in
203         let t_a_e = type_of a_e in
204         let a_t = aType_to_sType t in
205         if t_a_e <> a_t then
206           raise (Exceptions.TypeMismatch)
207         else
208           let env = Env.add id t_a_e env in
209           AAssign(id, a_e, t_a_e), env
210   | Assign(id, e) ->
211       if Env.mem id env then
212         raise (Exceptions.NameCollision(id))
213       else
214         let a_e, env = annotate_expression env e in
215         let t_a_e = type_of a_e in
216         let env = Env.add id t_a_e env in
217         AAssign(id, a_e, t_a_e), env
218   | Reassign(id, e) ->
219     if Env.mem id env then
220       let t = Env.find id env in
221       let a_e, env = annotate_expression env e in
222       let t_a_e = type_of a_e in
223       if t = t_a_e then AReassign(id, a_e, t), env
224       else raise(Exceptions.TypeMismatch)
225     else raise(Exceptions.IDNotFound(id))
226   | Binop(e1, op, e2) ->
227     let ae1, env = annotate_expression env e1 in
228     let ae2, env = annotate_expression env e2 in
229     let t = valid_binop((type_of ae1), (type_of ae2), op) in
```

```
230     ABinop(ae1, op, ae2, t), env
231   | Unop(op, e) ->
232     let ae, env = annotate_expression env e in
233     let et = type_of ae in
234     valid_unop(op, et);
235     AUnop(op, ae, et), env
236   | ListAccess(e, index) ->
237     let ae, env = annotate_expression env e in
238     let ind, env = annotate_expression env index in
239     let ae_t = type_of ae and
240     ind_t = type_of ind in
241     (
242       match ae_t with
243           List(t) ->
244           (
245             match ind_t with
246                 Int -> AListAccess(ae, ind, t), env
247               | _ -> raise(Exceptions.InvalidIndexing(a_type_to_string ind_t))
248           )
249         | _ -> raise(Exceptions.InvalidIndexing(a_type_to_string ae_t))
250     )
251   | TupleAccess(e, index) ->
252     let ae, env = annotate_expression env e in
253     let ind, env = annotate_expression env index in
254     let ae_t = type_of ae in
255     (
256       match ae_t with
257         Sast.Tuple(t) ->
258           let ind_n = evaluate_index(ind) in
259           if ind_n >= 0 then
260             if ind_n > (List.length t) then raise(Exceptions.ArrayOutOfBounds)
261             else
262             let param_type = List.nth t ind_n in
263             ATupleAccess(ae, ind, param_type), env
264           else raise(Exceptions.InvalidIndexing("Negative index"))
265         | _ -> raise(Exceptions.InvalidIndexing(a_type_to_string ae_t))
266     )
267   | IfBlock(e, e_list) ->
268     let a_list, _ = annotate_expression_list env e_list in
269     let ae, _ = annotate_expression env e in
270     let ae_s_type = type_of ae in
271     if ae_s_type = Bool then
272       AIfBlock(ae, a_list, Unit), env
273     else raise(Exceptions.IfRequiresBool(a_type_to_string ae_s_type))
274   | IfElseBlock(e1, l1, l2) ->
275     let ae, tempEnv = annotate_expression env e1 in
276     let a_list1, _ = annotate_expression_list env l1 in
277     let a_list2, _ = annotate_expression_list env l2 in
278     let le1 = (List.hd (List.rev a_list1)) in
279     let le2 = (List.hd (List.rev a_list2)) in
280     let ae_s_type = type_of ae in
```

```
281     let le1_s_type = type_of le1 and
282     le2_s_type = type_of le2 in
283     if ae_s_type <> Sast.Bool then
284       raise (Exceptions.IfRequiresBool(a_type_to_string ae_s_type))
285     else if le1_s_type <> le2_s_type then
286       raise (Exceptions.TypeMismatch)
287     else AIfElseBlock(ae, a_list1, a_list2, le1_s_type), env
288   | TypedFuncDecl(id, params, code, t) ->
289     if Env.mem id env then
290         raise (Exceptions.NameCollision(id))
291     else
292     let s_params = List.map annotate_parameter (List.rev params) in
293     let param_types = List.map (fun (i, t) -> t) s_params in
294     let env = Env.add id (Function(param_types, (aType_to_sType t))) env in
295     let tempEnv =  List.fold_left (fun cenv p -> (Env.add (fst p) (snd p) cenv
    )) env s_params in
296     let s_code, tempEnv = annotate_expression_list tempEnv code in
297     let le = (List.hd (List.rev s_code)) in
298     let le_s_type = type_of le in
299     let s_type = aType_to_sType t in
300     if le_s_type <> s_type && s_type <> Unit then
301       raise (Exceptions.TypeMismatch)
302     else
303       AFuncDecl(id, s_params, s_code, Function(param_types, s_type)), env
304   | FuncDecl(id, params, code) ->
305     if Env.mem id env then
306         raise (Exceptions.NameCollision(id))
307     else
308     let s_params = List.map annotate_parameter (List.rev params) in
309     let param_types = List.map (fun (i, t) -> t) s_params in
310     let tempEnv =  List.fold_left (fun cenv p -> (Env.add (fst p) (snd p) cenv
    )) env s_params in
311     let s_code, tempEnv = annotate_expression_list tempEnv code in
312     let le = (List.hd (List.rev s_code)) in
313     let le_s_type = type_of le in
314     let env = Env.add id (Function(param_types, le_s_type)) env in
315     AFuncDecl(id, s_params, s_code, Function(param_types, le_s_type)), env
316   | TypedFuncAnon(params, exp, t) ->
317     let s_params = List.map annotate_parameter (List.rev params) in
318     let param_types = List.map (fun (i, t) -> t) s_params in
319     let tempEnv =  List.fold_left (fun cenv p -> (Env.add (fst p) (snd p) cenv
    )) env s_params in
320     let s_e, tempEnv = annotate_expression tempEnv exp in
321     let s_e_type = type_of s_e in
322     let s_type = aType_to_sType t in
323     if s_e_type <> s_type && s_type <> Unit then
324       raise (Exceptions.TypeMismatch)
325     else
326       AFuncAnon(s_params, s_e, Function(param_types, s_type)), env
327   | FuncAnon(params, exp) ->
328     let s_params = List.map annotate_parameter params in
```

```
329     let param_types = List.map (fun (i, t) -> t) s_params in
330     let tempEnv =  List.fold_left (fun cenv p -> (Env.add (fst p) (snd p) cenv
    )) env s_params in
331     let s_e, tempEnv = annotate_expression tempEnv exp in
332     let s_e_type = type_of s_e in
333     AFuncAnon(s_params, s_e, Function(param_types, s_e_type)), env
334   | Call(e1, params) ->
335     let s_params, tempEnv = annotate_expression_list env (List.rev params) in
336     let id, env = annotate_expression env e1 in
337     let t = type_of id in
338     (match t with
339       Function(p, rt) ->
340       if rt = Reserved then check_reserved_functions(id, s_params, env), env
341       else if rt = Print then AFuncCall(id, s_params, Print), env else
342       let n_params = List.length p in
343       let sn_params = List.length s_params in
344       let rec match_types l1 l2 =
345       match l1 with
346         [] -> true
347       | hd::tl -> if (List.length l2 > 0 ) && (type_of hd) = (List.hd l2) then
348         match_types tl (List.tl l2) else if (type_of hd = Unit) then true else
    false
349       in
350       let s_type =
351         if not(match_types (List.rev s_params) p) then
352           raise(Exceptions.WrongParameterType(aexpression_to_string id))
353         else if sn_params <> n_params then Function((filter_params (p,
  sn_params)), rt)
354         else rt in
355       AFuncCall(id, s_params, s_type), env
356     | Map(kt, vt) ->
357       if (List.length s_params) = 1 then
358         let key = List.hd s_params in
359         if (kt = (type_of key)) then AMapAccess(id, key, vt), env
360         else raise(Exceptions.TypeMismatch)
361       else raise(Exceptions.InvalidIndexing(aexpression_to_string id))
362     |  _ -> raise(Exceptions.UnimplementedCallType(111)))
363   | FuncComposition(exp1, exp2) ->
364     let ae1, env = annotate_expression env exp1 in
365     let ae2, env = annotate_expression env exp2 in
366     let t1 = type_of ae1 in
367     let t2 = type_of ae2 in
368     let params = get_func_params t2 in
369     if (List.length params) > 1 then raise(Exceptions.
  ComposedIntermediateTakesMultipleArguments)
370     else
371     let p_type = List.hd(params) in
372     let r_type = get_func_return_type t1 in
373     if(p_type <> r_type) then raise(Exceptions.TypeMismatch)
374     else
375     let nr_type = get_func_return_type t2 in
```

```
    let n_params = get_func_params t1 in
    AFuncComposition(ae1, ae2, Function(n_params, nr_type)), env
  | FuncPipe(exp1, exp2) ->
    let ae1, env = annotate_expression env exp1 in
    let ae2, env = annotate_expression env exp2 in
    let t1 = type_of ae1 in
    let t2 = type_of ae2 in
    let params = get_func_params t2 in
    if (List.length params) > 1 then raise(Exceptions.
   ComposedIntermediateTakesMultipleArguments)
    else
    let p_type = List.hd(params) in
    if(p_type <> t1) then raise(Exceptions.TypeMismatch)
    else
    let nr_type = get_func_return_type t2 in
    AFuncPiping(ae1, ae2, nr_type), env

and annotate_expression_list env e_list =
  let env_ref = ref(env) in
  let rec helper = function
      head::tail ->
        let a_head, env = annotate_expression !env_ref head in
        env_ref := env;
        a_head::(helper tail)
    | [] -> []
  in (helper e_list), !env_ref

let annotate_program expression_list : Sast.aRoot =
  let env = Env.empty in
  let env = Env.add "print" (Function([Unit], Print)) env in
  let env = Env.add "hd" (Function([Unit], Reserved)) env in
  let env = Env.add "tl" (Function([Unit], Reserved)) env in
  let env = Env.add "len" (Function([Unit], Reserved)) env in
  let env = Env.add "is_empty" (Function([Unit], Reserved)) env in
  let a_expression_list, env = annotate_expression_list env expression_list in
  a_expression_list
```

8.36 ast.mli

```
type operator =
  Plus | Minus | Times | Divide | Modulo | Eq | Neq | Gt | Lt | Gte | Lte |
   And | Or | Not | Cons

type tTypes =
    TInt
  | TUnit
  | TBool
  | TString
  | TChar
  | TTuple of tTypes list
  | TList of tTypes
```

```
  | TFloat
  | TMap of tTypes * tTypes
  | TFunction of tTypes list * tTypes

type parameter = string * tTypes

type expression =
    IntLiteral of int
  | FloatLiteral of float
  | BoolLiteral of bool
  | StringLiteral of string
  | CharLiteral of char
  | UnitLiteral
  | IdLiteral of string
  | TupleLiteral of expression list
  | ListLiteral of expression list
  | MapLiteral of (expression * expression) list
  | Binop of expression * operator * expression
  | Unop of operator * expression
  | TypedAssign of string * expression * tTypes
  | Assign of string * expression
  | Reassign of string * expression
  | TupleAccess of expression * expression
  | ListAccess of expression * expression
  | IfBlock of expression * expression list
  | IfElseBlock of expression * expression list * expression list
  | Call of expression * (expression list)
  | TypedFuncDecl of string * parameter list * expression list * tTypes
  | FuncDecl of string * parameter list * expression list
  | TypedFuncAnon of parameter list * expression * tTypes
  | FuncAnon of parameter list * expression
  | FuncPipe of expression * expression
  | FuncComposition of expression * expression

type root = expression list
```

8.37 codegen.ml

```
open Ast
open Sast
open Exceptions

let rec strip_semicolon l =
  match l with
      [] -> ' '::[]
    | (hd::tl) -> if (hd = ';') then strip_semicolon tl else hd::(
   strip_semicolon tl)

let explode s =
  let rec exp i l =
    if i < 0 then l
```

```
13      else exp(i - 1)(s.[i] :: l) in
14        exp(String.length s - 1)[]
15
16  let implode l =
17    let res = String.create(List.length l) in
18    let rec imp i = function
19        [] -> res
20      | c :: l -> res.[i] <-c; imp(i + 1) l in
21        imp 0 l
22
23
24  let flip_last = fun l ->
25    let r = List.rev l
26    in (List.hd r) :: (List.rev (List.tl r))
27
28  let sanitize str =
29    implode ( strip_semicolon ( explode ( str) ) )
30
31  let type_list = function
32        List(_) -> true
33      | _ -> false
34
35  let reserve_mismatch = function
36        AListLiteral(_, _) -> false
37      | AIdLiteral(id, t) when (type_list t) -> false
38      | _ -> true
39
40  let operation_to_string = function
41      Plus -> "+"
42    | Minus -> "-"
43    | Times -> "*"
44    | Divide -> "/"
45    | Modulo -> "%"
46    | Eq -> "==="
47    | Neq -> "!=="
48    | Gt -> ">"
49    | Lt -> "<"
50    | Gte -> ">="
51    | Lte -> "<="
52    | And -> "&&"
53    | Or -> "||"
54    | Not -> "!"
55
56  let param_to_js (id, t) = id
57
58  let param_to_type (id, t) = t
59
60  let rec returns_unit = function
61      Unit -> true
62    | Function(_, ret) -> returns_unit ret
63    | _ -> false
```

```
64
65
66  let is_partial = function
67      Function(_, _) -> true
68    | _ -> false
69
70  let is_IfElseBlock = function
71      AIfElseBlock(_,_,_,_,_) -> true
72    | _ -> false
73
74  let get_IfElseExprList(n, expr) =
75    match expr with
76        AIfElseBlock(_,if_expr, else_expr, _) ->
77          if n = 0 then if_expr else else_expr
78      | _ -> []
79
80  let get_IfElseE = function
81        AIfElseBlock(e, _, _, _) -> e
82      | _ -> AUnitLiteral
83
84  let rec aexpression_to_js lines =
85    let rec build_return = function
86       [] -> "\n\t"
87    |  [single] -> "\n\treturn " ^ (aexpression_to_js single)
88    |  hd::tl -> "\n\t" ^ (aexpression_to_js hd) ^ (build_return tl)
89    in
90    match lines with
91      AIntLiteral(i) -> string_of_int(i)
92    | AFloatLiteral(f) -> string_of_float(f)
93    | ABinop(e1, op, e2, t) ->
94        if op = Cons then
95          "__cons__(" ^
96          sanitize(aexpression_to_js e1) ^ "," ^
97          sanitize(aexpression_to_js e2) ^ ");"
98        else
99          "(" ^ sanitize(aexpression_to_js e1) ^ " " ^
100         operation_to_string op ^ " " ^
101         sanitize(aexpression_to_js e2) ^ ");"
102   | AUnop(op, e1, t) ->
103     "(" ^  operation_to_string(op) ^ " " ^
104       aexpression_to_js(e1) ^ ")"
105       (* Unops won't be bythemselves as a single expression
106        * so no semicolon *)
107   | ABoolLiteral(b) ->
108       if b then "true"
109       else "false"
110   | AStringLiteral(s) -> s
111   | ACharLiteral(c) -> "'" ^ Char.escaped c ^ "'"
112   | AUnitLiteral -> "void"
113   | AIdLiteral(id, t) -> id
114   | AAssign(id, e, t) ->
```

```
115       "var" ^ " " ^ id ^ " = " ^
116       aexpression_to_js e ^ ";"
117   | AReassign(id, e, t) ->
118       id ^ " = " ^ aexpression_to_js e ^ ";"
119   | AMapLiteral(map_list, t) ->
120       let map_expression_tupal_to_string (e1, e2) =
121         (aexpression_to_js e1) ^ ": " ^ (aexpression_to_js e2)
122       in "{" ^
123       sanitize(String.concat ", " (List.map map_expression_tupal_to_string
124       map_list)) ^ "};"
125   | AMapAccess(id, param, s_type) ->
126       aexpression_to_js id ^ "[" ^ aexpression_to_js param ^ "];"
127   | ATupleLiteral(e_list, t) ->
128       "[" ^ String.concat ", " (List.map aexpression_to_js e_list) ^ "];"
129   | AListLiteral(e_list, t) ->
130       "[" ^ String.concat ", " (List.map aexpression_to_js e_list) ^ "];"
131   | ATupleAccess(id, indx, t) ->
132       sanitize(aexpression_to_js id) ^ "[" ^ sanitize(aexpression_to_js indx)
    ^ "];"
133   | AListAccess(id, indx, t) ->
134       sanitize(aexpression_to_js id) ^ "[" ^ sanitize(aexpression_to_js indx)
    ^ "];"
135
136   | AIfBlock(e, e_list, _) ->
137       "\nif(" ^ sanitize(aexpression_to_js e) ^ ") {" ^
138       "\n" ^ String.concat "\n\t" (List.map aexpression_to_js e_list) ^
139       "\n}\n"
140   | AIfElseBlock(e, e_list1, e_list2, _) ->
141       "\nif(" ^ sanitize(aexpression_to_js e) ^ ") {" ^
142       "\n\t" ^ String.concat "\n\t" (List.map aexpression_to_js e_list1) ^
143       "\n}\n" ^
144       "else {" ^
145       "\n\t" ^ String.concat "\n\t" (List.map aexpression_to_js e_list2) ^
146       "\n}\n"
147   | AFuncDecl(id, p_list, e_list, t) ->
148       if returns_unit t then
149         if param_to_type(List.hd p_list) <> Unit then
150           "function " ^ id ^ "(" ^
151           String.concat ", " (List.map param_to_js p_list) ^ ")" ^
152           " \n{\n" ^ String.concat "\n\t" (List.map aexpression_to_js e_list)
    ^
153           "\n};\n"
154         else
155          "function " ^ id ^ "() {" ^
156          "\n" ^ String.concat "\n\t" (List.map aexpression_to_js e_list) ^
157          "\n};\n"
158       else
159         if param_to_type(List.hd p_list) <> Unit then
160           let flipped_list = flip_last e_list in
161             "function " ^ id ^ "(" ^
162             (String.concat ", " (List.map param_to_js p_list)) ^
```

```
163             ") {" ^
164             (String.concat "\n\t" (List.map aexpression_to_js (List.tl
    flipped_list))) ^
165             (if is_IfElseBlock(List.hd flipped_list) then
166               "\nif(" ^
167               sanitize(aexpression_to_js(get_IfElseE (List.hd flipped_list)))
    ^
168               ") {" ^
169               (build_return (get_IfElseExprList(0, (List.hd flipped_list)))) ^
170               "\n} else {" ^
171               (build_return (get_IfElseExprList(1, (List.hd flipped_list)))) ^
172               "}\n"
173              else
174               "\n\treturn " ^
175               (aexpression_to_js (List.hd flipped_list))) ^
176             "\n};\n"
177         else
178           let flipped_list = flip_last e_list in
179             "function () {" ^
180             (String.concat "\n\t" (List.map aexpression_to_js (List.tl
    flipped_list))) ^
181             (if is_IfElseBlock(List.hd flipped_list) then
182               "\nif(" ^
183               sanitize(aexpression_to_js(get_IfElseE (List.hd flipped_list)))
    ^
184               ") {" ^
185               (build_return (get_IfElseExprList(0, (List.hd flipped_list)))) ^
186               "\n} else {" ^
187               (build_return (get_IfElseExprList(1, (List.hd flipped_list)))) ^
188               "}\n"
189              else
190               "\n\treturn " ^
191               (aexpression_to_js (List.hd flipped_list))) ^
192             "\n};\n"
193
194   | AFuncAnon(p_list, exp, t) ->
195       if returns_unit t then
196         if param_to_type(List.hd p_list) <> Unit then
197           "function(" ^
198           String.concat ", " (List.map param_to_js p_list) ^ ")" ^
199           " \n{\n\t" ^ aexpression_to_js exp ^
200           "\n};\n"
201         else
202          "function() {" ^
203          "\n\t" ^ aexpression_to_js exp ^
204          "\n};\n"
205       else
206         if param_to_type(List.hd p_list) <> Unit then
207           "function(" ^
208           String.concat ", " (List.map param_to_js p_list) ^ ")" ^
209           "\n{\n\treturn " ^ aexpression_to_js exp ^
```

```
210           "\n};\n"
211         else
212           "function() {" ^
213           "\n\treturn " ^ aexpression_to_js exp ^
214           "\n};\n"
215   | AFuncCall(id, params, s_type) ->
216       if (s_type = Reserved && ((List.length params <> 1) || (
    reserve_mismatch(List.hd params)))) then
217         raise(ReservedFuncTypeMisMatch)
218       else if s_type = Print then
219         "console.log(" ^ sanitize(aexpression_to_js (List.hd params)) ^ ");"
220       else
221         if List.hd params <> AUnitLiteral then
222           if is_partial s_type then
223             aexpression_to_js id ^ ".bind(this, " ^
224             sanitize(String.concat ", " (List.map aexpression_to_js (List.rev
    params))) ^
225             ");"
226           else
227             aexpression_to_js id ^ ".call(this, " ^
228             sanitize(String.concat ", " (List.map aexpression_to_js (List.rev
    params))) ^
229             ");"
230         else
231           aexpression_to_js id ^ "();"
232   | AFuncPiping(exp1, exp2, t) ->
233       sanitize(aexpression_to_js exp2) ^ "(" ^ sanitize(aexpression_to_js exp1)
    ^ ")"
234   | AFuncComposition(exp1, exp2, t) ->
235       "__compose__(" ^ sanitize(aexpression_to_js exp2) ^ ", " ^
236       sanitize(aexpression_to_js exp1) ^ ")"
237
238 let pumpkin_to_js a_expressions =
239   " var __compose__ = function() {
240       var funcs = arguments;
241       return function() {
242           var args = arguments;
243           for (var i = funcs.length; i --> 0;) {
244               args = [funcs[i].apply(this, args)];
245           }
246           return args[0];
247       };
248     };
249
250     var __cons__ = function(elem, lst) {
251       var temp = lst.slice(0);
252       temp.unshift(elem);
253       return temp
254     };
255     var hd = function(lst) {
256       var temp = lst.slice(0)
```

```
      if (temp.length > 0) {
        return temp[0];
      } else {return [];}
    };

    var tl = function(lst) {
      var temp = lst.slice(0);
      if (temp.length > 0) {
        temp.shift();
        return temp;
      } else {
        return []
      }
      };

    var len = function(lst) {
        return lst.length
    };

    var is_empty = function(lst) {
      return lst.length > 0 ? false : true

    };

      \n" ^
  String.concat "\n" (List.map aexpression_to_js a_expressions)
```

8.38 exceptions.ml

```
(* Processor Exception*)
exception MissingEOF

(* Scanning Exception *)
exception IllegalCharacter of char * int
exception IndentationError of int
exception UnmatchedQuotation of int
exception IllegalToken of string

(* Analyzer Exception *)
exception NameCollision of string
exception TypeNotFound of string
exception IDNotFound of string
exception TypeMismatch
exception InvalidOperation of string
exception UnimplementedOperation of string * string
exception InvalidIndexing of string
exception ArrayOutOfBounds
exception IfRequiresBool of string
exception WrongParameterType of string
exception ComposedIntermediateTakesMultipleArguments
exception InvalidMapKeyType
```

```
exception UnimplementedCallType of int
exception PipingIntoNonFunc
exception InvalidWildcard

(* Compiler exception!! *)
exception ReservedFuncTypeMisMatch
```

8.39 interpret.ml

```


let run (prog : Sast.aRoot) : unit = ignore(prog)
```

8.40 parser.mly

```
%{ open Ast %}

%token TERMINATOR INDENT DEDENT
%token LPAREN RPAREN COLON COMMA LBRACK RBRACK TYPEARROW DEFARROW
%token FPIPE BPIPE RCOMPOSE LCOMPOSE
%token PLUS MINUS TIMES DIVIDE MODULO EQ NEQ GT LT GTE LTE AND OR NOT
%token UMINUS UPLUS
%token CONS
%token VAL ASSIGN DEF
%token IF ELSE
%token TINT TUNIT TBOOL TSTRING TCHAR TTUPLE TLIST TFLOAT TMAP
%token <string> ID
%token <int> INT
%token <int> DEDENT_COUNT
%token <bool> BOOL
%token <string> STRING
%token <char> CHAR
%token <float> FLOAT
%token TUPLEACC
%token UNIT
%token EOF
%token <int> DEDENT_EOF

%right ASSIGN
%right DEFARROW
%left LBRACK RBRACK
%left FPIPE
%right BPIPE
%right CONS
%nonassoc LPAREN UNIT
%left RCOMPOSE
%right LCOMPOSE
%left OR
%left AND
%left EQ NEQ
```

```
%left LT GT LTE GTE
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right UMINUS UPLUS
%right NOT
%left TUPLEACC ACCESSOR

%start root
%type < Ast.root > root

%%
root:
    /* nothing */                  { [] }
  | root expression TERMINATOR     { $2::$1 }

expression:
    LPAREN expression RPAREN             { $2 }
  | controlflow                          { $1 }
  | assignment                           { $1 }
  | binop                                { $1 }
  | unop                                 { $1 }
  | literal                              { $1 }
  | call                                 { $1 }
  | funct                                { $1 }

indent_block:
    INDENT expression_block DEDENT { List.rev $2 }

expression_block:
    expression TERMINATOR                  { [$1] }
  | expression_block expression TERMINATOR { $2::$1 }

controlflow:
    if_statement indent_block   { IfBlock($1, $2) }
  | if_statement indent_block
    else_statement indent_block { IfElseBlock($1, $2, $4) }

if_statement:
    IF expression COLON TERMINATOR { $2 }

else_statement:
    ELSE COLON TERMINATOR { }

assignment:
    VAL ID COLON types ASSIGN expression { TypedAssign($2, $6, $4) }
  | VAL ID ASSIGN expression             { Assign($2, $4) }
  | ID ASSIGN expression                 { Reassign($1, $3) }

types:
    TINT                                 { TInt }
  | TFLOAT                               { TFloat }
```

```
87    | TBOOL                                    { TBool }
88    | TSTRING                                  { TString }
89    | TCHAR                                    { TChar }
90    | TUNIT                                    { TUnit }
91    | TTUPLE LBRACK type_list RBRACK           { TTuple($3) }
92    | TLIST LBRACK types RBRACK                { TList($3) }
93    | TMAP LBRACK types COMMA types RBRACK     { TMap($3, $5) }
94    | LPAREN funct_type RPAREN                 { $2 }
95
96  funct_type:
97      type_list DEFARROW types       { TFunction($1, $3) }
98    | type_list DEFARROW funct_type  { TFunction($1, $3) }
99
100 binop:
101     expression PLUS   expression          { Binop($1, Plus, $3) }
102   | expression MINUS  expression          { Binop($1, Minus, $3) }
103   | expression TIMES  expression          { Binop($1, Times, $3) }
104   | expression DIVIDE expression          { Binop($1, Divide, $3) }
105   | expression MODULO expression          { Binop($1, Modulo, $3) }
106   | expression EQ     expression          { Binop($1, Eq, $3) }
107   | expression NEQ    expression          { Binop($1, Neq, $3) }
108   | expression GT     expression          { Binop($1, Gt, $3) }
109   | expression LT     expression          { Binop($1, Lt, $3) }
110   | expression LTE    expression          { Binop($1, Lte, $3) }
111   | expression GTE    expression          { Binop($1, Gte, $3) }
112   | expression AND    expression          { Binop($1, And, $3) }
113   | expression OR     expression          { Binop($1, Or, $3) }
114   | expression CONS   expression          { Binop($1, Cons, $3) }
115
116 unop:
117     MINUS expression %prec UMINUS         { Unop(Minus, $2) }
118   | PLUS expression %prec UPLUS           { Unop(Plus, $2) }
119   | NOT expression                        { Unop(Not, $2) }
120
121 literal:
122     INT                                   { IntLiteral($1) }
123   | FLOAT                                 { FloatLiteral($1) }
124   | BOOL                                  { BoolLiteral($1) }
125   | STRING                                { StringLiteral($1) }
126   | CHAR                                  { CharLiteral($1) }
127   | UNIT                                  { UnitLiteral }
128   | LPAREN tupal_elements RPAREN          { TupleLiteral(List.rev $2) }
129   | expression TUPLEACC expression        { TupleAccess($1, $3) }
130   | LBRACK expression_list RBRACK         { ListLiteral($2) }
131   | LBRACK RBRACK                         { ListLiteral([]) }
132   | expression LBRACK expression RBRACK   { ListAccess($1, $3) }
133   | LPAREN map_list RPAREN                { MapLiteral($2) }
134   | ID                                    { IdLiteral($1) }
135
136 map_list:
137     map_item                { [$1] }
```

```
138   | map_list COMMA map_item { $3::$1 }
139
140 map_item:
141     expression TYPEARROW expression  { $1, $3 }
142
143 expression_list:
144     expression                       { [$1] }
145   | expression_list COMMA expression { $3::$1 }
146
147 tupal_elements:
148     expression COMMA                     { [$1] }
149   | tupal_elements_head expression       { $2::$1 }
150   | tupal_elements_head expression COMMA { $2::$1 }
151
152 tupal_elements_head:
153     expression COMMA                     { [$1] }
154   | tupal_elements_head expression COMMA { $2::$1 }
155
156 parameter_list:
157     parameter                      { [$1] }
158   | parameter_list COMMA parameter { $3::$1 }
159
160 parameter:
161     ID COLON types { $1, $3 }
162
163 type_list:
164     types                    { [$1] }
165   | type_list COMMA types    { $3::$1 }
166
167 call:
168     expression UNIT                          { Call($1, [UnitLiteral]) }
169   | expression LPAREN expression_list RPAREN { Call($1, List.rev $3) }
170
171 funct:
172     function_declaration { $1 }
173   | function_anon        { $1 }
174   | function_pipe        { $1 }
175   | function_composition { $1 }
176
177 function_declaration:
178     DEF ID function_parameters COLON types DEFARROW TERMINATOR indent_block {
      TypedFuncDecl($2, $3, $8, $5) }
179   | DEF ID function_parameters COLON types DEFARROW expression              {
      TypedFuncDecl($2, $3, [$7], $5)}
180   | DEF ID function_parameters DEFARROW TERMINATOR indent_block             {
      FuncDecl($2, $3, $6) }
181   | DEF ID function_parameters DEFARROW expression                          {
      FuncDecl($2, $3, [$5]) }
182
183 function_parameters:
184     /* nothing */               { ["()", TUnit] }
```

```
185   | LPAREN parameter_list RPAREN { $2 }
186
187 function_anon:
188     LPAREN parameter_list DEFARROW expression COLON types RPAREN {
      TypedFuncAnon($2, $4, $6) }
189   | LPAREN parameter_list DEFARROW expression RPAREN              { FuncAnon($2
      , $4) }
190
191 function_pipe:
192     expression FPIPE expression { FuncPipe($1, $3) }
193   | expression BPIPE expression { FuncPipe($3, $1) }
194
195 function_composition:
196     expression RCOMPOSE expression { FuncComposition($1, $3) }
197   | expression LCOMPOSE expression { FuncComposition($3, $1) }
```

8.41 processor.ml

```
1  open Parser
2
3  let line_number = ref 1
4  let last_token = ref EOF
5
6  (* Custom parser to account for dedent, carries line numbers from scanner *)
7  let dedent_list_from_count count ln =
8    let rec helper count dedent_list =
9      if count > 0 then (DEDENT, ln)::(TERMINATOR, ln)::(helper (count-1)
       dedent_list)
10     else dedent_list
11   in helper count []
12
13 let build_token_list lexbuf =
14   let rec helper lexbuf token_list =
15     let ln = !Scanner.lineno in
16     match Scanner.token lexbuf with
17         DEDENT_EOF(_) as eof -> (eof, ln)::token_list
18       | t -> (t, ln)::(helper lexbuf token_list)
19   in helper lexbuf []
20
21 let expand_token_list token_list =
22   let rec expand = function
23       (INDENT, ln)::tail -> (TERMINATOR, ln)::(INDENT, ln)::(expand tail)
24     | (DEDENT_COUNT(c), ln)::tail ->
25         (TERMINATOR, ln)::(List.append (dedent_list_from_count c ln) (expand
       tail))
26     | (DEDENT_EOF(c), ln)::tail ->
27         (TERMINATOR, ln)::(List.append (dedent_list_from_count c ln) (expand
       ((EOF, ln)::tail)))
28     | head::tail -> head::(expand tail)
29     | [] -> []
30   in expand token_list
```

```
31
32 let clean_token_list token_list =
33   let rec clean = function
34     | (TERMINATOR, _)::(ELSE, ln)::tail -> clean ((ELSE, ln)::tail)
35     | head::tail -> head::(clean tail)
36     | [] -> []
37   in clean token_list
38
39 let get_token_list lexbuf =
40   let token_list = build_token_list lexbuf in
41   let expanded_token_list = expand_token_list token_list in
42   let cleaned_token_list = clean_token_list expanded_token_list in
43   match cleaned_token_list with
44       (TERMINATOR, _)::tail -> tail
45     | _ as tl -> tl
46
47 let parser token_list =
48   let token_list = ref(token_list) in
49   let tokenizer _ =
50     match !token_list with
51       | (head, ln) :: tail ->
52           line_number := ln;
53           last_token := head;
54           token_list := tail;
55           head
56       | [] -> raise (Exceptions.MissingEOF)
57   in
58   let program = Parser.root tokenizer (Lexing.from_string "") in
59   List.rev program
```

8.42 pumpkin.ml

```
1 type action = Tokens | Ast | Sast | Interpret | Compile
2
3 let _ =
4   if Array.length Sys.argv < 2 then
5     print_string (
6       "Usage: pmkn [required-option] <source file>\n" ^
7         "required-option:\n" ^
8         "\t-t: Prints token stream\n" ^
9         "\t-a: Pretty prints Ast as a program\n" ^
10        "\t-s: Prints Sast\n" ^
11        "\t-i: Runs interpreter\n" ^
12        "\t-c: Compiles to Java\n"
13     )
14   else
15     let action = List.assoc Sys.argv.(1) [ ("-t", Tokens);
16                                            ("-a", Ast);
17                                            ("-s", Sast);
18                                            ("-i", Interpret);
19                                            ("-c", Compile) ] and
```

```
    filename = Sys.argv.(2) in
    let file_in = open_in filename in
    try
      let lexbuf = Lexing.from_channel file_in in
      let token_list = Processor.get_token_list lexbuf in
      let program = Processor.parser token_list in
      let sast_output = Analyzer.annotate_program program in
      match action with
          Tokens ->
            print_string (Utils.token_list_to_string token_list)
        | Ast ->
            print_string (Utils.program_to_string program)
        | Sast ->
            print_string (Utils.a_program_to_string sast_output)
        | Interpret -> print_string("\nInterpret\n")
        | Compile ->
            print_string (Codegen.pumpkin_to_js sast_output ^ "\n")

    with
        Exceptions.IllegalCharacter(c, ln) ->
          print_string
          (
            "In \"" ^ filename ^ "\", Illegal Character, '" ^
            Char.escaped c ^ "', line " ^ string_of_int ln ^ "\n"
          )
      | Exceptions.UnmatchedQuotation(ln) ->
          print_string("Unmatched Quotation, line " ^ string_of_int ln ^ "\n")
      | Exceptions.IndentationError(ln) ->
          print_string("Indentation Error, line " ^ string_of_int ln ^ "\n")
      | Parsing.Parse_error ->
          print_string
          (
            (
              if !Processor.last_token = Parser.INDENT then
                "Indentation Error"
              else
                "Syntax Error"
            ) ^ ", line " ^ string_of_int !Processor.line_number ^
            ", token " ^ Utils.token_to_string !Processor.last_token ^ "\n"
          )
```

8.43 sast.ml

```
open Ast

type sTypes =
    Int
  | Unit
  | Bool
  | String
  | Char
```

```
  | Tuple of sTypes list
  | List of sTypes
  | Float
  | Function of sTypes list * sTypes
  | Map of sTypes * sTypes
  | Reserved
  | Print

and aParameter = string * sTypes

and aExpression =
    AIntLiteral of int
  | AFloatLiteral of float
  | ABoolLiteral of bool
  | AStringLiteral of string
  | ACharLiteral of char
  | AUnitLiteral
  | ATupleLiteral of aExpression list * sTypes
  | AListLiteral of aExpression list * sTypes
  | AMapLiteral of (aExpression * aExpression) list * sTypes
  | AIdLiteral of string * sTypes
  | ABinop of aExpression * operator * aExpression * sTypes
  | AUnop of operator * aExpression * sTypes
  | AAssign of string * aExpression * sTypes
  | AReassign of string * aExpression * sTypes
  | ATupleAccess of aExpression * aExpression * sTypes
  | AListAccess of aExpression * aExpression * sTypes
  | AMapAccess of aExpression * aExpression * sTypes
  | AIfBlock of aExpression * aExpression list * sTypes
  | AIfElseBlock of aExpression * aExpression list * aExpression list * sTypes
  | AFuncCall of aExpression * (aExpression list) * sTypes
  | AFuncDecl of string * aParameter list * aExpression list * sTypes
  | AFuncAnon of aParameter list * aExpression * sTypes
  | AFuncComposition of aExpression * aExpression * sTypes
  | AFuncPiping of aExpression * aExpression * sTypes

and aRoot = aExpression list
```

8.44 scanner.mll

```
{
    open Parser

    let lineno = ref 1
    let indent_stack = Stack.create ()

    let get_eof () =
      let indent_length = Stack.length indent_stack - 1 in
      DEDENT_EOF(indent_length)
}

```

```
let alpha = ['a'-'z' 'A'-'Z']
let digit = ['0'-'9']
let id = alpha (alpha | digit | '_')*
let string = '"' [^ '"' '\\' '\n' '\r' '\t']* ('\\' [^ '\n' '\r' '\t'] [^ '"'
    '\\']*)* '"'
let char = ''' ( alpha | digit ) '''
let float = digit+ ['.'] digit+
let int = digit+
let whitespace = [' ' '\t']
let return = '\n' | "\r\n"

rule token = parse
    whitespace* "//"        { single_comment lexbuf }
  | whitespace* "/*"        { block_comment lexbuf }

  | return        { incr lineno; indent lexbuf }
  | whitespace    { token lexbuf }

  | '('  { LPAREN }
  | ')'  { RPAREN }
  | '['  { LBRACK }
  | ']'  { RBRACK }
  | '='  { ASSIGN }
  | '+'  { PLUS }
  | '-'  { MINUS }
  | '*'  { TIMES }
  | '/'  { DIVIDE }
  | '%'  { MODULO }
  | "::" { CONS }
  | ':'  { COLON }
  | ','  { COMMA }
  | '$'  { TUPLEACC }

  | "->"          { TYPEARROW }
  | "=>"          { DEFARROW }
  | "if"          { IF }
  | "else"        { ELSE }

  | "is"  | "==" { EQ }
  | "and" | "&&" { AND }
  | "or"  | "||" { OR }
  | "not" | '!'  { NOT }
  | "!="         { NEQ }
  | '>'          { GT }
  | '<'          { LT }
  | ">="         { GTE }
  | "<="         { LTE }

  | "val"          { VAL }
  | "def"          { DEF }

```

```
62    | "Int"          { TINT }
63    | "Float"        { TFLOAT }
64    | "String"       { TSTRING }
65    | "Unit"         { TUNIT }
66    | "Char"         { TCHAR }
67    | "Tuple"        { TTUPLE }
68    | "List"         { TLIST }
69    | "Map"          { TMAP }
70    | "Bool"         { TBOOL }
71
72    | "|>" { FPIPE }
73    | "<|" { BPIPE }
74    | ">>" { RCOMPOSE }
75    | "<<" { LCOMPOSE }
76
77    | "|>" whitespace* return { incr lineno; FPIPE }
78    | "<|" whitespace* return { incr lineno; BPIPE }
79    | ">>" whitespace* return { incr lineno; RCOMPOSE }
80    | "<<" whitespace* return { incr lineno; LCOMPOSE }
81
82    | "False"        { BOOL(false) }
83    | "True"         { BOOL(true) }
84    | "()"           { UNIT }
85    | int as lxm     { INT(int_of_string lxm) }
86    | float as lxm   { FLOAT(float_of_string lxm) }
87    | string as lxm  { STRING(lxm) }
88    | id as lxm      { ID(lxm) }
89    | char as lxm    { CHAR(String.get lxm 1)}
90
91    | eof { get_eof() }
92    | '"' { raise (Exceptions.UnmatchedQuotation(!lineno)) }
93    | _ as illegal
94      {
95        raise (Exceptions.IllegalCharacter(illegal, !lineno))
96      }
97
98  and single_comment = parse
99      return  { incr lineno; indent lexbuf }
100   | eof { get_eof() }
101   | _   { single_comment lexbuf }
102
103 and block_comment = parse
104     return { incr lineno; block_comment lexbuf }
105   | "*/" { token lexbuf }
106   | _    { block_comment lexbuf }
107
108 and indent = parse
109     whitespace* return { incr lineno; indent lexbuf }
110   | whitespace* "//"   { single_comment lexbuf }
111   | whitespace* "/*"   { block_comment lexbuf }
112   | whitespace* "|>"   { incr lineno; FPIPE }
```

```
  | whitespace* "<|"    { incr lineno; BPIPE }
  | whitespace* ">>"    { incr lineno; RCOMPOSE }
  | whitespace* "<<"    { incr lineno; LCOMPOSE }
  | whitespace* eof     { get_eof() }
  | whitespace* as indt
      {
        let indt_len = (String.length indt) in
        let top_len = (Stack.top indent_stack) in
        if indt_len > top_len then
          begin
          Stack.push indt_len indent_stack;
          INDENT
          end
        else if indt_len = top_len then
          TERMINATOR
        else
          let count =
            let rec helper inc =
                if (Stack.top indent_stack) > indt_len then
                    begin
                    ignore(Stack.pop indent_stack);
                    helper (inc + 1)
                    end
                else if (Stack.top indent_stack) < indt_len then -1
                else inc
            in helper 0
          in
          if count = -1 then raise (Exceptions.IndentationError !lineno)
          else DEDENT_COUNT(count)
      }

{
  Stack.push 0 indent_stack
}
```

8.45 utils.ml

```
open Ast
open Sast
open Parser

(* Ast Printer *)
let operation_to_string = function
    Plus -> "+"
  | Minus -> "-"
  | Times -> "*"
  | Divide -> "/"
  | Modulo -> "%"
  | Eq -> "is"
  | Neq -> "!="
  | Gt -> ">"
```

```
15    | Lt -> "<"
16    | Gte -> ">="
17    | Lte -> "<="
18    | And -> "and"
19    | Or -> "or"
20    | Not -> "not"
21    | Cons -> "::"
22
23  let rec type_to_string = function
24      TInt -> "Int"
25    | TUnit -> "Unit"
26    | TBool -> "Bool"
27    | TString -> "String"
28    | TChar -> "Char"
29    | TFloat -> "Float"
30    | TTuple(t)  -> "Tuple[" ^ String.concat ", " (List.map type_to_string t) ^
      "]"
31    | TList(t)-> "List[" ^ type_to_string t ^ "]"
32    | TMap(t1, t2) -> "Map[" ^ type_to_string t1 ^ ", " ^ type_to_string t2 ^ "]
      "
33    | TFunction(t1, t2) -> "(" ^ String.concat ", " (List.map type_to_string t1)
      ^ " => " ^ type_to_string t2 ^ ")"
34
35  let parameters_to_string (id, t) = id ^ ": " ^ type_to_string t
36
37  let rec expression_to_string indent_length = function
38      IntLiteral(i) -> string_of_int(i)
39    | FloatLiteral(f) -> string_of_float(f)
40    | BoolLiteral(b) ->
41        if b then "True"
42        else "False"
43    | StringLiteral(s) -> s
44    | CharLiteral(c) -> "'" ^ Char.escaped c ^ "'"
45    | UnitLiteral -> "()"
46    | IdLiteral(id) -> id
47    | Binop(e1, op, e2) ->
48        "(" ^
49        expression_to_string indent_length e1 ^ " " ^
50        operation_to_string op ^ " " ^
51        expression_to_string indent_length e2 ^
52        ")"
53    | Unop(op, e) ->
54      "(" ^ operation_to_string op ^
55      (
56        match op with
57            Not ->  " "
58          | _   ->  ""
59      ) ^ expression_to_string indent_length e ^ ")"
60    | TypedAssign(id, e, t) ->
61      "val " ^ id ^ ": " ^ type_to_string t ^ " = " ^ expression_to_string
     indent_length e
```

```
62  | Assign(id, e) ->
63    "val " ^ id ^ " = " ^ expression_to_string indent_length e
64  | Reassign(id, e) ->
65    id ^ " = " ^ expression_to_string indent_length e
66  | TupleLiteral(e_list) ->
67    "(" ^ String.concat ", " (List.map (expression_to_string indent_length)
     e_list) ^ ",)"
68  | TupleAccess(e, e_acc) ->
69    expression_to_string indent_length e ^ "$(" ^ expression_to_string
     indent_length e_acc ^ ")"
70  | ListLiteral(e_list) ->
71    "[" ^ String.concat ", " (List.map (expression_to_string indent_length)
     e_list) ^ "]"
72  | ListAccess(e, e_acc) ->
73    expression_to_string indent_length e ^ "[" ^ expression_to_string
     indent_length e_acc ^ "]"
74  | MapLiteral(map_list) ->
75      let map_expression_tupal_to_string (e1, e2) =
76        "(" ^ expression_to_string indent_length e1 ^ " -> " ^
     expression_to_string indent_length e2 ^ ")"
77      in
78      "Map(" ^ String.concat ", " (List.map map_expression_tupal_to_string
     map_list) ^ ")"
79  | IfBlock(e, e_list) ->
80      let indent_length = indent_length + 1 in
81      let tabs = String.make indent_length '\t' in
82      "if " ^ expression_to_string indent_length e ^ " :\n" ^
83      tabs ^ String.concat ("\n" ^ tabs) (List.map (expression_to_string
     indent_length) e_list)
84  | IfElseBlock(e, e_list1, e_list2) ->
85      let else_indent_length = indent_length and
86      indent_length = indent_length + 1 in
87      let else_tabs = String.make else_indent_length '\t'
88      and tabs = String.make indent_length '\t' in
89      "if " ^ expression_to_string indent_length e ^ " :\n" ^
90      tabs ^ String.concat ("\n" ^ tabs) (List.map (expression_to_string
     indent_length) e_list1) ^ "\n" ^
91      else_tabs ^"else :\n" ^
92      tabs ^ String.concat ("\n" ^ tabs) (List.map (expression_to_string
     indent_length) e_list2)
93  | Call(exp, e_list) ->
94      (expression_to_string indent_length exp) ^
95      (
96        match e_list with
97            UnitLiteral::[] -> "()"
98          | _ -> "(" ^ String.concat ", " (List.map (expression_to_string
     indent_length) e_list) ^ ")"
99      )
100 | TypedFuncDecl(id, p_list, e_list, t) ->
101     let indent_length = indent_length + 1 in
102     let tabs = String.make indent_length '\t' in
```

```
103       "def " ^ id ^ " (" ^ String.concat ", " (List.map parameters_to_string
    p_list) ^ ") : " ^ type_to_string t ^ " =>\n" ^
104       tabs ^ String.concat ("\n" ^ tabs) (List.map (expression_to_string
    indent_length) e_list)
105   | FuncDecl(id, p_list, e_list) ->
106       let indent_length = indent_length + 1 in
107       let tabs = String.make indent_length '\t' in
108       "def " ^ id ^ " (" ^ String.concat ", " (List.map parameters_to_string
    p_list) ^ ") =>\n" ^
109       tabs ^ String.concat ("\n" ^ tabs) (List.map (expression_to_string
    indent_length) e_list)
110   | TypedFuncAnon(p_list, e, t) ->
111       "(" ^ String.concat ", " (List.map parameters_to_string p_list) ^ " =>
    " ^
112       expression_to_string indent_length e ^ " : " ^ type_to_string t ^ ")"
113   | FuncAnon(p_list, e) ->
114       "(" ^ String.concat ", " (List.map parameters_to_string p_list) ^ " =>
    " ^
115       expression_to_string indent_length e ^ ")"
116   | FuncPipe(e1, e2) ->
117       "(" ^ expression_to_string indent_length e1 ^ " |> " ^
    expression_to_string indent_length e2 ^ ")"
118   | FuncComposition    (e1, e2) ->
119       "(" ^ expression_to_string indent_length e1 ^ " >> " ^
    expression_to_string indent_length e2 ^ ")"
120
121 let program_to_string expressions =
122   String.concat "\n" (List.map (expression_to_string 0) expressions) ^ "\n"
123
124 (* Tokens to String *)
125
126 let token_to_string = function
127     TERMINATOR -> "TERMINATOR" | INDENT -> "INDENT"
128   | DEDENT -> "DEDENT" | LPAREN -> "LPAREN"
129   | RPAREN -> "RPAREN" | COLON -> "COLON"
130   | COMMA -> "COMMA" | LBRACK -> "LBRACK"
131   | RBRACK -> "RBRACK" | TYPEARROW -> "TYPEARROW" | DEFARROW ->  "DEFARROW"
132   | FPIPE -> "FPIPE" | BPIPE -> "BPIPE" | LCOMPOSE -> "LCOMPOSE" | RCOMPOSE ->
    "RCOMPOSE"
133   | PLUS -> "PLUS" | MINUS -> "MINUS"
134   | TIMES -> "TIMES" | DIVIDE -> "DIVIDE"
135   | MODULO -> "MODULO" | EQ -> "EQ"
136   | NEQ -> "NEQ" | GT -> "GT"
137   | LT -> "LT" | GTE -> "GTE"
138   | LTE -> "LTE" | AND -> "AND"
139   | OR -> "OR" | NOT -> "NOT"
140   | UMINUS -> "UMINUS" | UPLUS -> "UPLUS"
141   | VAL -> "VAL" | ASSIGN -> "ASSIGN" | DEF -> "DEF"
142   | IF -> "IF" | ELSE -> "ELSE"
143   | TINT -> "TINT" | TUNIT -> "TUNIT"
144   | TBOOL -> "TBOOL" | TSTRING -> "TSTRING"
```

```
145   | TCHAR -> "TCHAR" | TTUPLE -> "TTUPLE"
146   | TLIST -> "TLIST" | TFLOAT -> "TFLOAT"
147   | TMAP -> "TMAP"
148   | UNIT -> "UNIT"
149   | CONS -> "CONS"
150   | EOF -> "EOF"
151   | ID(s) -> "ID(" ^ s ^ ")"
152   | INT(i) -> "INT(" ^ string_of_int i ^ ")"
153   | FLOAT(f) -> "FLOAT(" ^ string_of_float f ^ ")"
154   | DEDENT_COUNT(i) -> "DEDENT_COUNT(" ^ string_of_int i ^ ")"
155   | BOOL(b) -> "BOOL(" ^ (if b then "true" else "false") ^ ")"
156   | STRING(s) -> "STRING(" ^ s ^ ")"
157   | TUPLEACC -> "TUPLEACC"
158   | CHAR(c) -> "CHAR(" ^ Char.escaped c ^ ")"
159   | DEDENT_EOF(i) -> "DEDENT_EOF(" ^ string_of_int i ^ ")"
160
161 let token_list_to_string token_list =
162   let rec helper last_line_number = function
163       (token, line)::tail ->
164         (if line != last_line_number then "\n" ^ string_of_int line ^ ". "
      else " ") ^
165         token_to_string token ^ helper line tail
166     | [] -> "\n"
167   in helper 0 token_list
168
169 (* Analyzer Utils *)
170
171 (* Sast Printer*)
172
173 let rec a_type_to_string = function
174     Int -> "Int"
175   | Unit -> "Unit"
176   | Bool -> "Bool"
177   | String -> "String"
178   | Char -> "Char"
179   | Tuple(t) -> "Tuple[" ^ String.concat ", " (List.map a_type_to_string t) ^
      "]"
180   | List(t) -> "List[" ^ a_type_to_string t ^ "]"
181   | Float -> "Float"
182   | Function(t1, t2) -> "Function(" ^ String.concat ", " (List.map
      a_type_to_string t1)^ " => " ^ a_type_to_string t2 ^ ")"
183   | Map(t1, t2) -> "Map[" ^ a_type_to_string t1 ^ ", "^ a_type_to_string t1 ^
      "]"
184   | Print -> "PRINT"
185   | Reserved -> "RESERVED"
186
187 let a_param_list_to_string (id, t) = id ^ ": " ^ a_type_to_string t
188
189 let rec aexpression_to_string = function
190     AIntLiteral(i) -> string_of_int(i)
191   | AFloatLiteral(f) -> string_of_float(f)
```

```
192   | ABinop(e1, op, e2, t) ->
193     "( " ^ aexpression_to_string(e1) ^ " " ^
194     operation_to_string(op) ^ " " ^
195     aexpression_to_string(e2) ^ " " ^
196     ")" ^ "_" ^ a_type_to_string(t)
197   | AUnop(op, e1, t) ->
198     operation_to_string(op) ^ " " ^
199     aexpression_to_string(e1) ^ "_" ^
200     a_type_to_string(t)
201   | ABoolLiteral(b) ->
202     if b then "true"
203     else "false"
204   | AStringLiteral(s) -> s
205   | ACharLiteral(c) -> Char.escaped c
206   | AUnitLiteral -> "Unit"
207   | AIdLiteral(id, t) -> id ^ "_" ^ a_type_to_string(t)
208   | AAssign(id, e, t) ->
209     "Assign(" ^ "_" ^ a_type_to_string t ^ ") " ^
210     id ^ " = " ^
211     aexpression_to_string e
212   | AReassign(id, e, t) ->
213     "Reassign(" ^ "_" ^ a_type_to_string t ^ ") " ^
214     id ^ " = " ^
215     aexpression_to_string e
216   | ATupleLiteral(e_list, t) ->
217     "Tuple(" ^ String.concat ", " (List.map aexpression_to_string e_list) ^ ")
      " ^ "_" ^ a_type_to_string(t)
218   | ATupleAccess(id, indx, t) ->
219     "TupleAccess " ^ aexpression_to_string id ^ "(" ^ aexpression_to_string
      indx ^ ")" ^ "_" ^ a_type_to_string(t)
220   | AListLiteral(e_list, t) ->
221     "List(" ^ String.concat ", " (List.map aexpression_to_string e_list) ^ ")"
      ^ "_" ^ a_type_to_string(t)
222   | AListAccess(id, indx, t) ->
223     "ListAccess " ^ " " ^ aexpression_to_string id ^ "( " ^
      aexpression_to_string indx ^ ")_" ^ a_type_to_string(t)
224   | AMapLiteral(map_list, t) ->
225     let map_expression_tupal_to_string (e1, e2) =
226       "(" ^ aexpression_to_string e1 ^ " -> " ^ aexpression_to_string e2 ^ ")"
227     in
228     "Map(" ^ String.concat ", " (List.map map_expression_tupal_to_string
      map_list) ^ ")" ^ "_" ^ a_type_to_string(t)
229   | AIfBlock(e, e_list, t) ->
230     "\nIf(" ^ aexpression_to_string e ^ ")\n" ^
231     "\t" ^ String.concat "\n\t" (List.map aexpression_to_string e_list) ^ "_"
      ^ a_type_to_string(t) ^ "\n" ^
232     "EndIf\n"
233   | AIfElseBlock(e, e_list1, e_list2, t) ->
234     "\nIf(" ^ aexpression_to_string e ^ ")\n" ^
235     "\t" ^ String.concat "\n\t" (List.map aexpression_to_string e_list1) ^ "_"
      ^ a_type_to_string(t) ^ "\n" ^
```

```
236     "Else\n" ^
237     "\t" ^ String.concat "\n\t" (List.map aexpression_to_string e_list2) ^ "_"
        ^ a_type_to_string(t) ^ "\n" ^
238     "EndIf\n"
239   | AFuncDecl(id, p_list, e_list, t) ->
240     if (List.length p_list) <> 0 then
241       "\n def " ^ id ^ " (" ^ String.concat ", " (List.map
      a_param_list_to_string p_list) ^ ") : " ^ a_type_to_string t ^ " =>\n" ^
242       "\t" ^ String.concat "\n\t" (List.map aexpression_to_string e_list) ^ "\
      n"
243     else
244       "\n def " ^ id ^ " : " ^ a_type_to_string t ^ " =>\n" ^
245       "\t" ^ String.concat "\n\t" (List.map aexpression_to_string e_list) ^ "\
      n"
246   | AFuncAnon(p_list, exp, t) ->
247     if (List.length p_list) <> 0 then
248       "\n(" ^ String.concat ", " (List.map a_param_list_to_string (List.rev
      p_list)) ^ a_type_to_string t ^ " =>\n" ^
249       "\t" ^ aexpression_to_string exp ^ ")\n"
250     else
251       "\n ( =>\n" ^
252       "\t" ^ aexpression_to_string exp ^ ")\n"
253   | AFuncCall(id, params, s_type) ->
254     if (List.length params) <> 0 then
255       "\n" ^ aexpression_to_string id ^ " (" ^ String.concat ", " (List.map
      aexpression_to_string params) ^ ")" ^ "_" ^ a_type_to_string(s_type) ^ "\n
      "
256     else
257       "\n " ^ aexpression_to_string id ^ "()" ^ "_" ^ a_type_to_string(s_type)
258   | AFuncComposition(exp1, exp2, t) ->
259       "\n" ^ aexpression_to_string exp1 ^ ">>" ^ aexpression_to_string exp2 ^
      "_" ^ a_type_to_string(t) ^ "\n"
260   | AMapAccess(id, param, s_type) ->
261       "\nMapAccess" ^ aexpression_to_string id ^ " (" ^ aexpression_to_string
      param ^ ")" ^ "_" ^ a_type_to_string(s_type) ^ "\n"
262   | AFuncPiping(exp1, exp2, t) ->
263       "\n" ^ aexpression_to_string exp1 ^ "|>" ^ aexpression_to_string exp2 ^
      "_" ^ a_type_to_string(t) ^ "\n"
264
265 let a_program_to_string a_expressions =
266   String.concat "\n" (List.map aexpression_to_string a_expressions) ^ "\n"
```