



PLOTTER

Ibrahima Niang
Ranjith Kumar Shanmuga Visvanathan
Sania Arif

in2190
rs3579
sa3311

10th May, 2016.

Contents

I	Introduction	5
II	Tutorial	6
1	Environment Setup	6
2	Using the Compiler	6
3	Basics	6
3.1	Primitives	6
3.2	List	6
3.3	Operators	7
3.4	Loops	7
3.5	Functions	8
3.6	Line	8
3.7	Typography	9
3.7.1	print	9
3.7.2	printXY	9
III	Language Reference Manual	10
4	Lexical Conventions	10
4.1	Identifiers	10
4.2	Keywords	10
4.3	Comments	10
4.4	Constants	10
4.5	White space	11
5	Types	11
5.1	Operators	11
6	Expressions	12
6.1	<i>num</i>	12
6.2	<i>bool</i>	12
6.3	<i>string</i>	12
6.4	<i>list</i>	12
6.5	<i>point</i>	13
6.6	Arithmetic operators:	13
6.7	Relational and logical operators:	14
7	Statements	14
7.1	Declaration & Assignment	14
7.2	Return	15

8 Control Statements	15
8.1 Conditions	15
8.2 Loops	15
9 Scope	16
10 Functions	16
10.1 Function Declaration & Definition	16
10.2 Function Call	17
10.3 Built-in functions	17
11 Sample Program	18
IV Project Plan	21
1 Project Timeline	22
2 Development Environment	23
3 Version Control	23
3.1 Contributions	23
3.2 Contributors	24
3.3 Code frequency	25
3.4 Punch Card	25
V Architecture	26
1 High Level (plotter.ml)	26
2 Scanner (scanner.mll)	26
3 Parser (parser.mly)	27
4 Abstract Syntax Tree (ast.ml)	27
5 Semantic Checker (semcheck.ml)	27
6 Code Generation (codegen.ml)	28
7 C++ Compiler	28
8 Compiler Contributions	28
VI Additional Features	29
1 Libraries	29
2 Customizability	32

VII Test Plan	34
1 Test Automation	34
2 Test Cases	34
3 Testing Roles	34
VIII Future Work	35
IX Lessons Learned	36
1 Sania & Ranjith Kumar	36
2 Ibrahima	37
X Appendix	38

Part I

Introduction

plOtter is a data manipulation language built on the principles and appreciation of a minimalist aesthetic. The goal of plOtter is to provide a means for users to design and implement their own plots to visualize data, that would otherwise be done by rigid, automated software (such as Microsoft Excel). Our language will simplify charting maps through domain-specific types and operations. The users can build their custom graph templates using our language and can reuse the template to visualize different data points.

Very primitive examples could be building a bar-graph, histogram etc. Higher level examples could be drawing small album art, or other complex examples could be to visualize/plot a binary tree, Gantt chart, etc.

Following is a manual that provides reference information for using plOtter. It describes the lexical conventions, basic types scoping rules, built-in functions and also displays a sample program and output.

Part II

Tutorial

This covers you through the basic things in starting with Plotter.

1 Environment Setup

The environment required are Linux based, Ocaml and g++ 11 compiler. Install OCaml and g++ 11, as the target language for Plotter is c++ and its compiled with g++ 11.
Then the Plotter source code can be cloned from
↳ git clone https://github.com/saniaarif22/PlOtter.git

2 Using the Compiler

The code is inside src/ folder. Follow the README.md for information and crunched steps.
Run 'make' inside src, to build the Plotter.
Then run your .plt code with

And Voila! This will generate the svg at the same path of your .plt file, if it compiled successfully and throws the corresponding errors if it didnt.

3 Basics

This sections guides you through some of the basics of Plotter and implementing them in Plotter.

3.1 Primitives

The primitives supported are num, string, bool and point. An example program showing declaration and usage of primitive variables is shown.

```
1 num a
2 bool b
3 string c
4 point d
5 a = 10
6 b = true
7 c = "Hello Word!"
8 d = (a,20)
```

Listing 1: "Primitives in Plotter"

3.2 List

We apart from primitive also support non primitive data structure. The main this is List. List can contain any of the primitive as elements. A small sample is given below to

show the declaration and usage of list with its operations.

```

1 #List of num elements
2 list num a
3
4 #setting it variables
5 a = [10,3,8,4,10]
6
7 #Append element to it
8 a.append(20)
9
10 #pop an element
11 a.pop() /* a is now [10,3,8,4,10] */
12
13 #remove
14 a.remove(1) /* a is now [10,8,4,10] .. a[1] is removed*/
15
16 #similar operations with bool or string
17 list bool b
18 list string c
19
20 #setting it variables
21 b = [10,3,8,4,10]
22 c = [ "this", "is", "a", "sample"]

```

Listing 2: "Lists in Plotter"

3.3 Operators

Operators are listed in order of precedence. All operators associate left to right, except for assignment, which associates right to left.

<i>Arithmetic</i>	$+, -, *, /, \%, **$
<i>Comparison</i>	$==, !=, <, >, \leq, \geq, \text{and}, \text{or}$
<i>Assignment</i>	$=, +=, -=, *=, /=, \% =, * =$
<i>Membership</i>	in, not in

```

1 #List of num elements
2 num a
3 a = 10
4 a = a+1 *(5-4)
5 if a > 0 && a <10 :
6     print "Between 0 and 10"
7 else:
8     print "Not between 0 and 10"
9 end

```

Listing 3: "Operators in Plotter"

3.4 Loops

Plotter provides both for loop and while loop. A sample for and while loop in Plotter.

3.5 Functions

In Plotter, functions are supported which comes in handy when we want to manipulate data or creating your own plotting functions like draw rectangle, draw square etc. Also the internal library contains full of functions like these.

```
1 fn printAll(list num a):
2     num i
3     for i=0;i<a.length();i++:
4         print i
5     end
6 end
7
8 #List of num elements
9 list num a
10 a = [3,4,5,6]
11 printAll(a)
```

Listing 4: "Functions in Plotter"

3.6 Line

This is the essence of the language. The ability to draw beautiful images, and in anyway we want. The very core available in-built function to do this is Line.

```
1
2 fn rect(point a, num h, num w):
3     line(a,(a[0]+w,a[1]))
4     line(a,(a[0],a[1]+h))
5     line((a[0]+w,a[1]),(a[0]+w,a[1]+h))
6     line((a[0],a[1]+h),(a[0]+w,a[1]+h))
7 end
```

Listing 5: "Using built-in line"

Below shown is a simple implementation of drawing a rectangle with line.
We also support drawing the same line with color. Below shown is a implementation of filled rectangle with colored lines.

```
1 fn rectFill(point a, num h, num w, string color):
2     num i
3     point x
4     point y
5     string s
6
7     /* Make a rectangle by drawing multiple lines */
8     for i=0;i<w; i=i+1:
9         x = (a[0]+i, a[1])
10        y = (a[0]+i, a[1]+h)
11        line(x, y, color)
12    end
13 end
```

Listing 6: "Using built-in line with color"

3.7 Typography

The second important feature when it comes to plotting is printing the text. By default we support two inbuilt functions. They are print and printXY.

3.7.1 print

The print, can be used to print and debug simple things, it posts the print expr in the middle of the svg. It might not be very advantageous in printing lots of data, as they may overlap.

```
1 num a
2 a = 10
3 print 4
4 print a
5 print "Hello"
```

Listing 7: "Simple Print"

3.7.2 printXY

The printXY, can be used to print an expression at a specific (x,y) coordinate in the output SVG. This comes really helpful in printing things like axes, labels, titles etc.

```
1 num a
2 a = 10
3 #Prints 4 at location (10,10)
4 print (4, (10,10))
5 point p
6 p = (100.5,100.5)
7
8 #prints hello string at 100.5,100.5
9 print ("Hello",p)
```

Listing 8: "PrintXY"

Part III

Language Reference Manual

4 Lexical Conventions

The lexical structure consists of the set of basic rules that define how to write programs in plOtter. The normal token types are identifiers, keywords, operators, delimiters, and literals, as covered in the following sections.

4.1 Identifiers

An identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter (A to Z or a to z) or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9). plOtter is case sensitive, lowercase and uppercase letters are distinct.

4.2 Keywords

num	string	bool	point
true	false	while	for
if	else	return	end
fn	list		

4.3 Comments

Both single line and multi-line comments are supported.

Single Line	#
Multi Line	/* */

The pound or hash-tag symbol # is used for comment. Everything from the # to the end of the line is ignored. It can be used for either single line comment multiple line comments as follow:

```
#This is a single line comment
```

/* ... */ is used for multi line comments. Multi-line comments cannot be nested.

```
/* this is  
an example of  
multi-line comment */
```

4.4 Constants

false: The false value of the bool type.

true: The true value of the bool type.

4.5 White space

- Blank lines

A logical line that contains only spaces, tabs, formfeeds and possibly a comment, is ignored (i.e., no NEWLINE token is generated).

- Line statement

In plOtter, end of a statement is marked by a newline character

- Indentations

plOtter provides no braces to indicate blocks of code for function definitions or flow control. Blocks of code are denoted by line indentation, which must be ended by end. For example:

```
if true:  
    print "True"  
end  
else:  
    print "False"  
end
```

5 Types

Our language supports the following data-types:

1. Primitive types

<i>num</i>	32 bit floating point number
<i>bool</i>	Boolean value

2. Non-Primitive or object types

<i>point</i>	A bi-element tuple of type <i>num</i>
<i>list</i>	List of <i>num</i> or <i>string</i> elements
<i>hash</i>	Dictionary with <i>num</i> or <i>string</i> keys
<i>string</i>	Chain of characters enclosed by double quotes

Note: We do not have a primitive type ‘char’, and an index of a string points to a string of length one.

5.1 Operators

Operators are listed in order of precedence. All operators associate left to right, except for assignment, which associates right to left.

<i>Arithmetic</i>	$+, -, *, /, \%, **$
<i>Comparison</i>	$==, !=, <, >, \leq, \geq, \text{and}, \text{or}$
<i>Assignment</i>	$=, +=, -=, *=, /=, \% =, *= =$

6 Expressions

An expression is a combination of values, variables, and operators.

6.1 num

```
num a  
a = 10  
num b = 10
```

6.2 bool

```
bool a  
a = false  
bool b = false
```

6.3 string

```
string a  
a = "plOtter"  
string b = "plOtter"
```

6.4 list

Lists support both lists of num and lists of string. the list is written as a list of comma separated elements inside square brackets. Each element of the list has an index with which it can be accessed directly. The list supports access, appending, deleting. The size can change dynamically during the course of the program with appending or deletion of an element in the list. Some examples of list initializations are given below.

```
list a = [ 1, 2, 3, 4, 5]  
list b = [ 1.0, 2.0, 3.0, 4.0, 5.0]  
list c = [ "string 1","String 2","String 3"]  
list a = []
```

Operations

append(x) : Appends the given element, x, to the end of the list. x should be of the same type as the existing elements in list. If this is the first element then that is set as the type for that list. And once this type is fixed it cant be changed after that. i.e After filling a list with num type, we cant pop all elements and insert string. That would throw an error.

```
a.append(6)  
will make a to [ 1, 2, 3, 4, 5, 6 ]
```

pop() : this operations pops the last element of the list. If there are no elements present, it

doesnt do anything.

a.pop()

will make a to [1, 2, 3, 4, 5]

`remove(x)` : This operation is to remove the element at the x'th position. And the index starts with 0.

a.remove(3)

will remove index 3 element '4' and make a to [1, 2, 3, 5]

`at(x)` or "[x]" : The elements in the list can be accessed by their index using 'at' and providing the index, or by directly providing the index at square brackets. This will return the element at the index x.

a.at(2) or a[2]

will return 3

`length()` : This returns the length of the list at that point.

a.length() with return 4

Also the type of the elements in the list need not be specified during the initialization. The language automatically determines the type. However all the elements in a single list variable should be of same type. It should either be all string or all num.

6.5 point

point pMax = (10, 10)

6.6 Arithmetic operators:

2 + 3

5

num a = 5.0

num b = 3.0

a * b

15.0

14 % 3

2

22.0 12.0

1.8333333333333333

6.7 Relational and logical operators:

True or False

True

True and False

False

$1 < 2$

True

$23 == 45$

False

$34 != 323$

True

7 Statements

7.1 Declaration & Assignment

Declaration statements are to declare variables to use in the program. Assignment statements are used to declare variables and initialize a value to a variable, or assign a value to an initialized variable. They can be expressed in the following ways:

1. $Type\ Name\ Variable\ Name$
 $Variable\ Name = Value$

For example:

num x = 3
string s = "hello"

Note: Strings values are expressed by enclosing characters in double quotes.

2. $Type\ Name\ Variable\ Name$
 $Variable\ Name = Value1$
 $Variable\ Name = Value2$

For example:

num x = 3
x = 4.5

Note: An un-initialized variable will be initialized to the following:

<i>num</i>	0
<i>string</i>	null
<i>point</i>	(0,0)
<i>list</i>	[]
<i>hash</i>	{ }

7.2 Return

The return statement is used to exit a function, and it may optionally pass back an expression to the caller. If required, the expression is converted, by assignment, to the type of the function in which it appears. A return statement with no arguments is the same as return None. It can be expressed as:

1. *return expression*
2. *return*

8 Control Statements

8.1 Conditions

The 2 forms of conditional statements are:

- *if- else*

This is of the form:

```
if expression :
    statements
else :
    statements
end
```

Note: The end indicates the end of the conditional block.

The expression is evaluated, and if the condition is satisfied, the set of statements associated in the block is executed. If none of the expressions' result is satisfied, the alternate set of statements associated with the *else* block are executed.

8.2 Loops

- *while*

This has the form:

```
while expression :
    statements
end
```

The expression is evaluated, and if the condition is satisfied, the set of statements

associated in the block are executed. The test takes place before each iteration of execution of the statements, and the statements are executed repeatedly, as long as the value of the expression remains satisfied.

- *for*

This has the following form:

– *for Initialize ; Condition ; Increment/ Decrement :*

 Statements

end

Initialization specifies the initialization for the loop. Condition specifies an expression to test if the loop should continue. The loop is exited when the expression is satisfied. The final part is typically an increment or decrement operation to be made on the loop variable.

– *for Type name = start ; end :*

 Statements

end

Here, start specifies the value from where to begin loop iteration. End is the final part, i.e. till when the loop should continue. The value of the loop variable will increment or decrement by 1, till 'end' is reached.

9 Scope

We have broadly 3 categories for scope of variables:

- Local Scope

These are for variables defined outside a function or control statement block, that are accessible throughout the program. They exist throughout the lifetime of the program and can be accessed inside any function or control statement block.

- Function parameters

Formal parameters of a function are treated as local variables.

10 Functions

Functions allow structure programs as segments of code to perform tasks that also can be reused.

10.1 Function Declaration & Definition

Function declaration is done in the following format:

fn name of the function (arguments):

end

Here, arguments can be a set of arguments or no arguments. The function definition does not convey anything about the return type of the function. Some examples are given below.

Examples :

Function to convert Celsius to Fahrenheit. It takes one argument and returns Num type.

```
1 fn convertToCelcius(num f):  
2     return (f-32)/1.8  
3 end  
4
```

Function with no argument, returning type string

```
1 fn getWelcomeString():  
2     return "Hello World"  
3 end
```

10.2 Function Call

Functions are called in the following way. They are basically identified by their function name and the arguments it has, to be sent. Error will be thrown if the number of arguments during the call didn't match the function signature. However defaulting the argument values can be done, in such a case the passing arguments could be less.

10.3 Built-in functions

These are the functions our language provides. However, most of these can be written by using the primitive blocks in plOtter.

Note : In all these functions and also the program all size/thickness units are *pixel* and all *color is 6 byte hex string* Eg "FFFAAF". By default color is black. Also if not specified in options default values are substituted.

line (point p1, point p2, hash options)

This function lets users draw a line from one point to another point in the SVG. It takes in the starting point, the ending point and at last options like color size etc. The options are thickness and color.

Eg. line((0,0),(10,10),"thickness":10)

printXY (point p, string s, bool dir, hash options)

This function lets users print string in the SVG. The printXY function, takes as argument point in the screen the point specifying the top right corner of the print, the text to be printed, boolean dir, representing whether the string should be horizontal(True) or vertical(False), and at last options like color size etc. The options are size and color.

Eg. printXY((10,10),"Hello World", true, "size":10)

We will have more functions, specific to each type of plot (For example: line plot, pie charts, etc.) similar to the barGraph function, for the user to call.

The purpose of providing these built-in functions is to demonstrate the power of our language, enabling the user to build his own custom functions, as desired. Since most of these built in functions are written in plOtter

11 Sample Program

This is a sample program that gets data from a csv file, applies some options to the data and plots the data in a bar-graph. We also supply sample code for the bar-graph function, written in plOtter, which will be provided as an in-built function to the user, to show the ability to build custom graphs/plots using our language.

```
1 # Function written by user
2 list num a
3 a = [11,4,25.6,10,12,50,10,30,5]
4 barGraph(a)
```

Listing 9: Bar Graph Example

The output of the above code will be like something below.

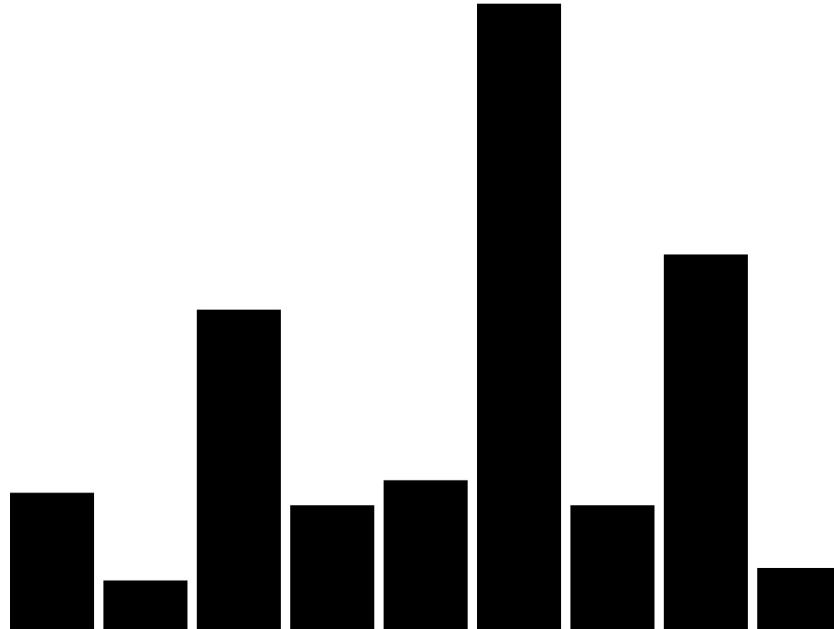


Figure 1: Simple Bar Graph

```
1 #Bar Graph From the Library
2 /*
3     Method :
4         barGraph(<num list var>)
```

```

5
6     Return :
7         void
8             Prints a fitting barGraph for the given data
9
10    Description :
11        The function bargraph supported as library. Users can write their own
12        too. It takes the list of num and automatically fits it in the graph based
13        on the data.
14    The user just needs to call the function with the data
15
16    */
17 fn barGraph(list num a):
18     num maxLength
19     num maxHeight
20     num maxDataHt
21     num maxDataLn
22     num gap
23     num scaleFactor
24     num padHz
25     num padVt
26     num barWidth
27     num i
28
29
30     #Setting the dimensions of the graph
31     maxLength = 640
32     maxHeight = 480
33
34     #Max ht
35     maxDataHt = a[0]
36     for i=0;i<a.length();i=i+1:
37         if a[i] > maxDataHt:
38             maxDataHt = a[i]
39         end
40     end
41
42     #max length
43     maxDataLn = a.length()
44
45     #padding
46     padHz = 10
47     padVt = 10
48
49     #bar graph settings 10% of the graph
50     gap      = 0.1 * (maxLength - padHz) / maxDataLn
51     barWidth = 0.9 * (maxLength - padHz) / maxDataLn
52     scaleFactor = (maxHeight - padVt) / maxDataHt
53
54     #Draw the bars, scaled and with the gap
55     num x
56     x = padHz
57     for i=0;i<a.length();i=i+1:
58         #Drawing the bar
59         rect( (x,maxHeight- a[i]*scaleFactor) , a[i]*scaleFactor , barWidth )
60         #line( (x,maxHeight- a[i]*scaleFactor) , (x,maxHeight) )
61         #print x

```

```
62         x = x + barWidth + gap
63     end
64
65
66 end
```

Listing 10: Snippet of code from Library barGraph function in plOtter.

The function barGraph is built using primitive blocks and is provided as a built-in function. However, its shown here to emphasize the power user can have for creating his own graph.

Part IV

Project Plan

We met once a week as a group, worked together every alternate day. The first two milestones, the proposal and language reference manual gave us a broad idea of the scope of our project and language semantics. Most of the language specifications were finalized by then, which was an over-estimate of what we would be able to accomplish in the semester.

Here on, we first worked to get an initial working version of the whole pipeline- from scanner, parser, abstract syntax tree, to code generation. By the time this reached completion, we reached the next milestone, which was a hello world program in our language.

From there, we incorporated a semantic checker, that maintained a symbol table and carried out scope checking and type validation. We then incrementally added features, layer by layer, going from scanner, to parser, to ast, to semantically checked ast, to target language ast, to code generation.

We also went on to set up a regression test suite that made sure the development of each feature would happen end-to-end. We added tests incrementally too, and in 2 divisions-pass tests (tests that should pass) and fail tests (tests that should not pass). Our test script would run all tests and auto-detect the kind of test, from the name of the test and report individual passed and failed tests, along with complete test statistics.

Any errors, bugs or discrepancies were resolved during the timely meetings of the system architects.

1 Project Timeline

Date	Accomplishment
Jan 25	Discuss possible languages, assign roles
Jan 27	Finalize language idea
Jan 28	Finish and submit project proposal
Feb 02	Locked down language syntax and semantics
Feb 03	Discussed the need of 2 different int and float data types
Feb 04	Finalize language data types and structures
Feb 05	Discussed the need for conditionals
Feb 07	Finalized syntax for functions
Feb 08	Compiled and submitted Language reference manual
Feb 17	Started tweaking MicroC
Feb 20	Initialized git repository
Feb 27	Added features to MicroC
Feb 29	Scrapped out MicroC and started building own compiler from scratch
Mar 4	Started the Scanner
Mar 20	Started the Parser
Mar 22	Completed an early version of Scanner, Parser and AST
Mar 24	Added Pretty print for Parser
Mar 25	Wrote sample target C++ program to write to .svg
Mar 27	Incrementally added to Scanner, Parser
Mar 31	Completed Scanner, Parser, AST
April 3	Started code generation, wrote a hello world program, wrote one test for hello world, updated the makefile
April 5	Successfully completed the pipeline, compiled a hello world program in PIOTter
April 6	Added regression test script
April 15	Added a semantic check in pipeline
April 22	Compiled an early version of semcheck
April 24	Completed pipeline with semcheck
April 26	Added line function in language
April 27	Added lists and loops
May 04	Added sample programs
May 05	Completed list operations
May 07	Added function calls
May 08	Added precise error checking
May 10	Release 1.0 of PIOTter is out and available
May 11	Final presentation, report and project files submitted Let's get some sleep.

2 Development Environment

Language	OCaml, python, C++
Build System	Make
Editor	Sublime, Brackets, Vim
Version Control	Git
Bug Tracker	GitHub Issues
Operating System	OS X/ Linux

3 Version Control

We used Git as a distributed version control system to allow all members of the group to work in collaboration and independently.

Here are some graphs produced by GitHub:

3.1 Contributions

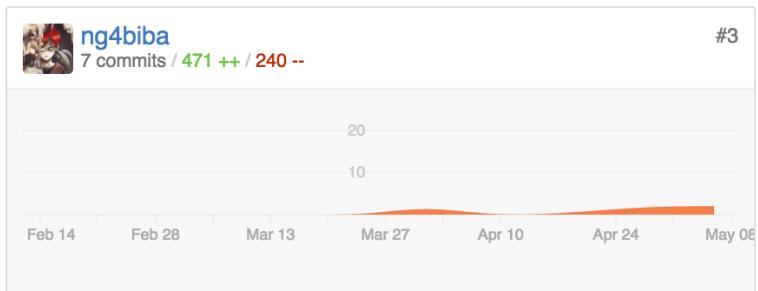
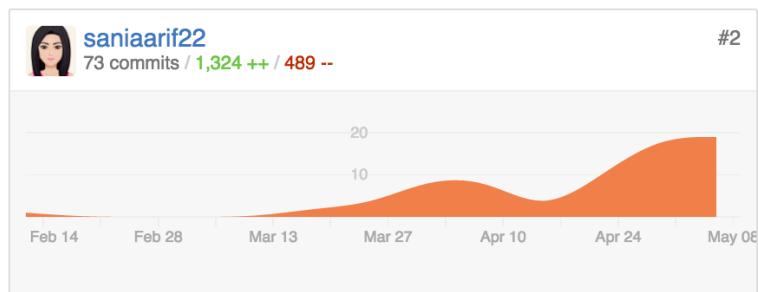
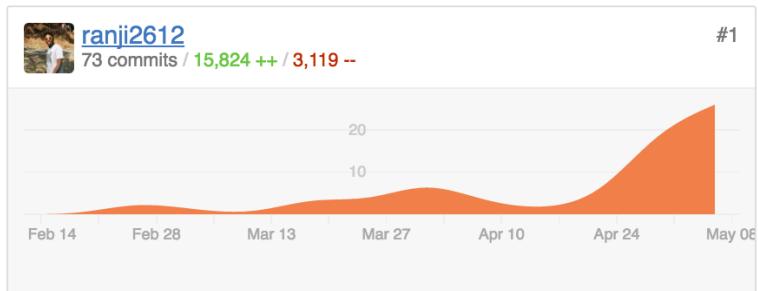
Feb 14, 2016 – May 11, 2016

Contributions to master, excluding merge commits

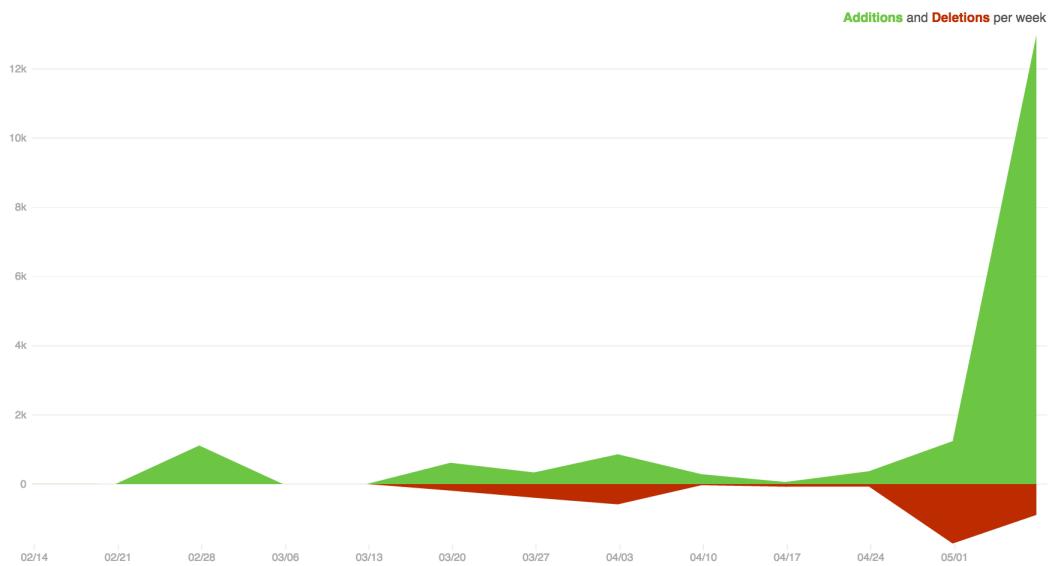
Contributions: **Commits** ▾



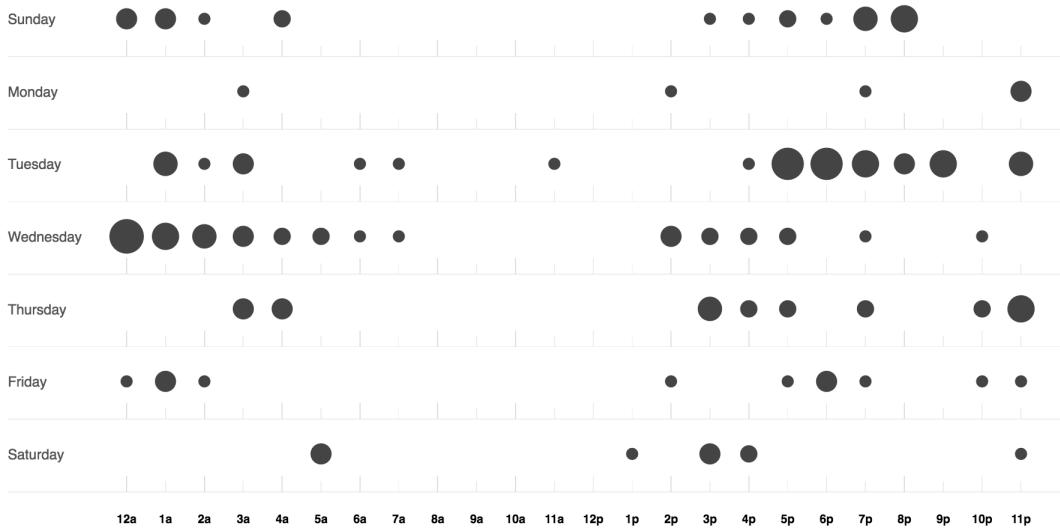
3.2 Contributors



3.3 Code frequency

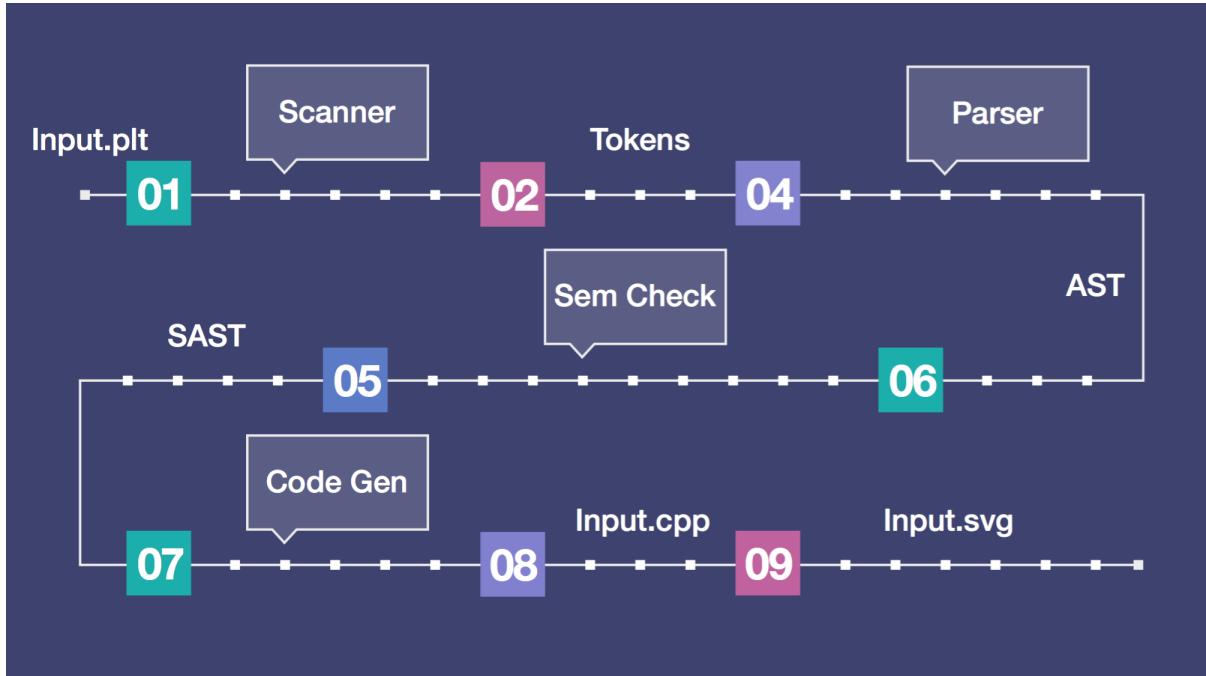


3.4 Punch Card



Part V

Architecture



1 High Level (plotter.ml)

This is the entry point to our compiler. It takes as input, the code in PlOtter from the lexbuffer as stdin and calls each stage of the compiler.

If an invalid input is encountered, an exception is thrown in the scanning stage and if incorrect syntax is encountered, it is reported to the user and the parser continues parsing.

This stage thus, performs error reporting and throws exceptions, exiting the program if required. It also prints out the abstract syntax tree and the semantically checked tree, depending on the compiler flags given as input.

2 Scanner (scanner.mll)

The scanner performs the following functions:

- Generates tokens, which are identifiers, operators, literals, and symbols.
- Removes single and multi-line comments
- Removes whitespaces, except for newlines, we parse as tokens.

- Checks for illegal input and throws and exception for an unrecognized character along with the character and it's exact location.
(This is our first stage of error- reporting)

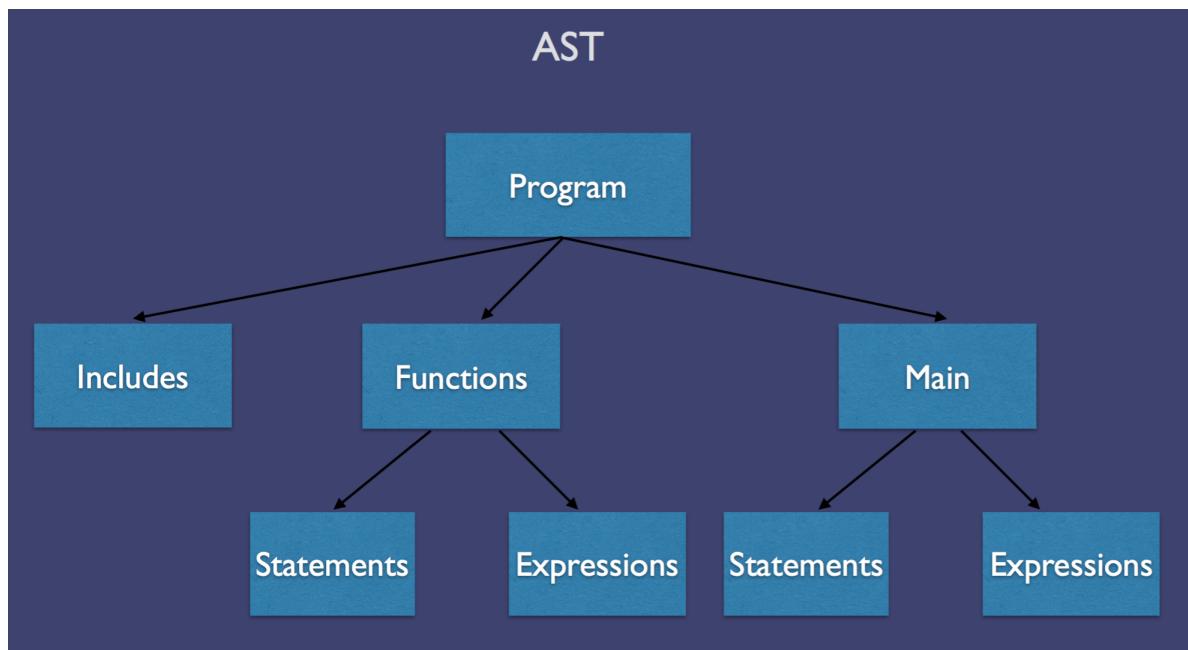
3 Parser (parser.mly)

The parser takes the stream of tokens as input and generates an abstract syntax tree as output. Our parser uses the EOL (end of line whitespace token) to denote the end of a line.

Additionally, we implemented detailed and precise error checking. Every mistype or incorrect syntax is caught and displayed to the user as an erroneous entry, a possible resolution, along with the exact location of this error. (This is our second stage of error- reporting)

4 Abstract Syntax Tree (ast.ml)

The AST has the following structure:



5 Semantic Checker (semcheck.ml)

The semantic checker enforces the rules that we layed out in the Reference Manual, that includes type and scope validation and type safety. If the AST complies successfully, the semantic checker emits a semantically checked AST (SAST). We do this by maintaining a global symbol table for all statements and expressions in the main and for function names,

and different symbol tables for each user-defined function.

This is the third and last stage where an error can be caught and reported. All compile time errors are caught and reported at semcheck. If the semcheck passes, all the stages of our compiler will pass successfully.

6 Code Generation (codegen.ml)

The code generator uses the semantically checked abstract syntax tree, passed to it by the semantic checker to construct the C++ file which contains the final code for the program, and which writes to .svg.

7 C++ Compiler

We chose to use the g++ 11 compiler. The main reason for this was the ability to use compound literals, which were essential for list and point operations.

8 Compiler Contributions

File	Authors
scanner.ml	Ibrahima, Ranjith, Sania
parser.mly	Ranjith, Sania
ast.ml	Ranjith, Sania
semcheck.ml	Ranjith, Sania
sast.ml	Ranjith, Sania
codegen.ml	Ranjith, Sania
err.ml	Sania
plotter.ml	Ranjith
cppsast.ml	Ibrahima, Ranjith, Sania
tast.ml	Ibrahima, Ranjith, Sania

Part VI

Additional Features

1 Libraries

We have built-in functions that we write in PlOtter, made available to the user as built-in functions. While these functions are not forced on the user to be used, the user can very well access them using:

include plots

```
1  /*
2   Method :
3     rect(point xy, num h, num w)
4
5   Return :
6     void
7
8   Description :
9     Prints a rectangle , from point x,y with height h and width w The
10    function rectangle supported as library .
11
12  Future Todo :
13    Send hash , to set the properties of the graph like axis , grid , title
14    etc
15  */
16 fn rect(point a, num h, num w):
17   line(a,(a[0]+w,a[1]))
18   line(a,(a[0],a[1]+h))
19   line((a[0]+w,a[1]),(a[0]+w,a[1]+h))
20   line((a[0],a[1]+h),(a[0]+w,a[1]+h))
21 end
22
23 fn rectFill(point a, num h, num w, string color):
24   num i
25   point x
26   point y
27   string s
28
29   /* Make a rectangle by drawing multiple lines */
30   for i=0;i<w; i=i+1:
31     x = (a[0]+i, a[1])
32     y = (a[0]+i, a[1]+h)
33     line( x , y , color )
34   end
35 */
36 Method :
37   drawAxes(num xorigin , num yorigin , num maxWidth , num maxHeight)
38
39 Return :
40   void
41
42 Description :
43   Draws cross axis from the passed origin point . Takes max width and
max Height to plot lines
```

```

44
45     Future Todo :
46         Send hash , to set the properties of the graph like axis , grid , title
47         etc
48 */
49 fn drawAxes(num ox , num oy , num maxLength , num maxHeight):
50
51     #Draw axes
52     rect ((ox,0) , maxHeight , 3)
53     rect ((0,maxHeight-oy) ,3 ,maxLength)
54 end
55 /*
56     Method :
57     barGraph(<num list var>)
58
59     Return :
60         void
61
62     Description :
63         Prints a fitting barGraph for the given data. The function bargraph
64         supported as library. Users can write their own too. It takes the list of
65         num and automatically fits it in the graph based on the data.
66         The user just needs to call the function with the data
67
68     Future Todo :
69         Send hash , to set the properties of the graph like axis , grid , title
70         etc
71 */
72 fn barGraph(list num a):
73     num maxLength
74     num maxHeight
75     num maxDataHt
76     num maxDataLn
77     num gap
78     num scaleFactor
79     num padHz
80     num padVt
81     num barWidth
82     num i
83
84     #Setting the dimensions of the graph
85     maxLength = 640
86     maxHeight = 480
87
88     #Max ht
89     maxDataHt = a[0]
90     for i=0;i<a.length();i=i+1:
91         if a[i] > maxDataHt:
92             maxDataHt = a[i]
93         end
94
95     #max length
96     maxDataLn = a.length()
97
98     #padding
99     padHz = 10
100    padVt = 10

```

```

100
101 #bar graph settings 10% of the graph
102 gap = 0.1 * (maxLength - padHz) / maxDataLn
103 barWidth = 0.9 * (maxLength - padHz) / maxDataLn
104 scaleFactor = (maxHeight - padVt) / maxDataHt
105
106 #Draw the bars, scaled and with the gap
107 num x
108 x = padHz
109 for i=0;i<a.length();i=i+1:
110     #Drawing the bar
111     rect( (x,maxHeight- a[i]*scaleFactor) , a[i]*scaleFactor , barWidth )
112     #line( (x,maxHeight- a[i]*scaleFactor) , (x,maxHeight) )
113     #print x
114     x = x + barWidth + gap
115 end
116
117
118 end
119 /*
120 Method :
121     lineGraph(num xorigin , num yorigin , num maxWidth , num maxHeight)
122
123 Return :
124     void
125
126 Description :
127     Draws a line graph based on the points give. To draw axes call it
128     seperately
129
130 Future Todo :
131     Send hash, to set the properties of the graph like axis , grid , title
132     etc
133 */
134 fn lineGraph(list num a):
135     num maxLength
136     num maxHeight
137     num maxDataHt
138     num maxDataLn
139     num gap
140     num scaleFactor
141     num padHz
142     num padVt
143     point p1
144     point p2
145     num i
146
147     #Setting the dimensions of the graph
148     maxLength = 640
149     maxHeight = 480
150
151     #Max ht
152     maxDataHt = a[0]
153     for i=0;i<a.length();i=i+1:
154         if a[i] > maxDataHt:
155             maxDataHt = a[i]
156         end
157     end

```

```

158
159     #max length
160     maxDataLn = a.length()
161
162     #padding
163     padHz = 10
164     padVt = 10
165
166     #Line graph settings 10% of the graph
167     gap      = (maxLength - padHz) / maxDataLn
168     scaleFactor = (maxHeight - padVt) / maxDataHt
169
170     #Draw the bars, scaled and with the gap
171     num x
172     num j
173     num y1
174     num y2
175     x = padHz
176
177     if a.length() > 0:
178
179         for i=1;i<a.length();i=i+1:
180             line( (x , maxHeight - a[ i - 1 ] * scaleFactor) , ( x + gap ,
181                         maxHeight- a[ i ]*scaleFactor ) )
182
183             #dots on the data points
184             rect ( (x - 2.5, maxHeight - a[ i - 1 ] * scaleFactor - 2.5) , 5
185             , 5)
186             rect ( (x - 2.5, maxHeight - a[ i - 1 ] * scaleFactor - 2.5) , 5
187             , 5)
188             x = x + gap
189         end
189     end

```

2 Customizability

Alternatively, the user can implement their own functions using our most basic, primitive functions print and line. Some of the features are **rect(point p, num width, num height)**

This draws a rectangle of width and height, starting from p, with options. The options we are considering now are simple like, thickness(of the border), color(of the border) and fill(boolean variable specifying fill should be done). If fill is true, additional option called fillColor is also required, if not specified default of black fill is used.

Eg. rectangle((10,10),5, 10,"thickness":1, "color":"#FFFFFF","fill":false)

rectFill(point p, num width, num height)

This draws a rectangle of width and height, starting from p, with options. The options we are considering now are simple like, thickness(of the border), color(of the border) and fill(boolean variable specifying fill should be done). If fill is true, additional option called fillColor is also required, if not specified default of black fill is used.

Eg. rectangle((10,10),5, 10,"thickness":1, "color":"#FFFFFF","fill":false)

barGraph(list data)

This draws a barGraph given the data and specified options. The options include, num max-width, num max-height, num color (of the bar), xaxis(boolean representing whether

the axis should be there), bool yaxis, string xlabel, string ylabel, bool grid(to have/not have grid lines) and string title (For example: padding, max width, max height, etc.)

Eg.

```
hash options = { "xAxis": true, "yAxis": true, "xLabel": "Time", "yLabel": "Celsius", "title": "Temperature with Time", "grid": true}  
barGraph([5,1,4,2,7])
```

lineGraph(list data, hash options)

This draws a lineGraph given the data and specified options. The options include, num max-width, num max-height, num color (of the bar), xaxis(boolean representing whether the axis should be there), bool yaxis, string xlabel, string ylabel, bool grid(to have/not have grid lines) and string title (For example: padding, max width, max height, etc.). Although the options feature is not there now, it'll be updated further.

Eg.

```
hash options = { "xAxis": true, "yAxis": true, "xLabel": "Time", "yLabel": "Celsius", "title": "Temperature with Time", "grid": true}  
barGraph([5,1,4,2,7])
```

Part VII

Test Plan

1 Test Automation

Our testing automation program is invoked by calling make test. Our script either takes as input, specific files to test on, or iterates through each file in the test directory ending that is ending in .plt.

We define 2 kinds of tests:

- Pass tests- Tests that are supposed to pass
- Fail tests- Tests that are supposed to fail

After running the program, the script first detects the type of test (pass test vs fail test), and if the test was supposed to pass, it compares the generated result (a .svg file) with the corresponding .svg file. It then maintains a list of the files that pass tests and the files that fail and generates statistics at the end.

2 Test Cases

Our test directory contains all of our tests. The pass tests are preceded by the word ‘pass’ and the fail tests are preceded by the word ‘fail’. As of commit bdb5887, we have 85 tests. We have tests for the following categories:

1. Assignment
2. Operators
3. Comments
4. Printing
5. Loops
6. Conditionals
7. Lists
8. List operations
9. Functions
10. Libraries
11. Scope

3 Testing Roles

The test script was written by Ranjith. Tests were written by either Sania or Ranjith, the person who implemented each feature. We also raised issues for each bug as we encountered it and were responsible collectively for closing an issue on resolution.

Part VIII

Future Work

- More customizability
Adding more features such as line type, line width and weight, background color, etc.
- Support for math functions
Such as square, cube, sine, cosine, etc.
- Import Data
Ability to import data from additional sources (eg: csv)
- More responsive REPL window
- More libraries

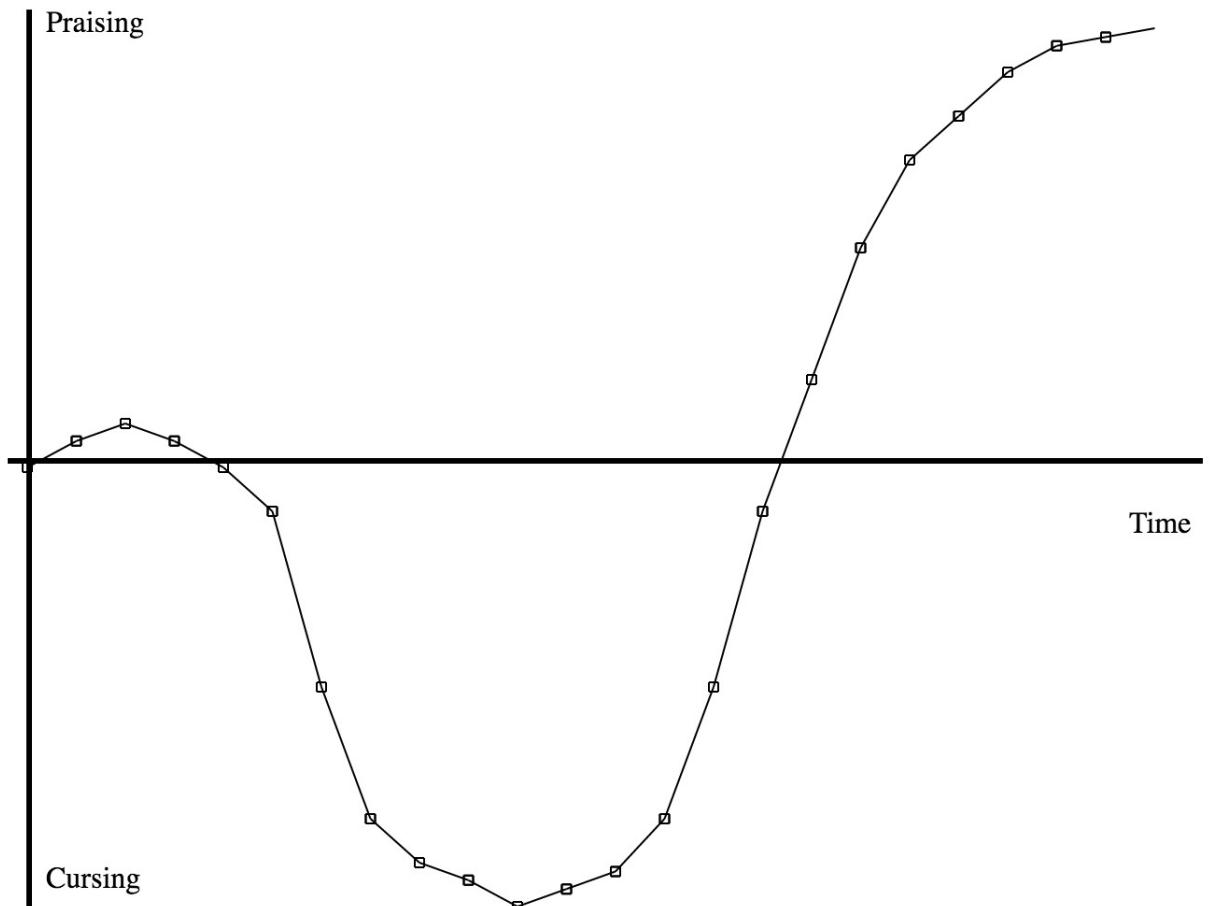
Part IX

Lessons Learned

1 Sania & Ranjith Kumar

- Pair Programming saves lives.
- Use Prof. Edwards' slides.
- Choose teammates wisely, you'll be stuck with them for the term. Courtesy: Prof. Edwards
- OCaml is awesome, give it time.
Our graph of appreciating OCaml:

Output SVG



2 Ibrahima

Learn Ocaml and get to familiarize with the syntax as early as possible. Understanding the architecture of the compiler and how its pieces fit together is a must in order to keep the progress going. Lastly group project is amazing,specially when you feel included in the work, when members work with one another and shared their knowledge with the rest of the team.

Part X

Appendix

./src/.gitignore

```
1 tests/*.cpp
2 tests/*.svg
3 programs/*.cpp
4
5
```

./src/ast.ml

```
1 (* operations *)
2 type ops =
3     | Add | Sub | Mul | Div | Mod
4     | Equal | Neq | Less | Leq | Greater | Geq
5     | And | Or
6     | Square
7
8 type bool =
9     | True | False
10
11
12 (* expressions *)
13 type expr =
14     Literal_Num of float
15     Literal_Str of string
16     Point of expr * expr
17     Literal_List of expr list
18     Binop of expr * ops * expr
19     Id of string
20     Bool of bool
21     Length of expr
22     Access of expr * expr
23
24
25
26 type stmt = (* Statements *)
27     Expr of expr
28     | Var_Decl of string * string
29     | List_Decl of string * string
30     | Passign of expr * expr * stmt
31     | Assign of expr * expr
32     | Append of expr * expr
33     | Pop of expr
34     | Remove of expr * expr
35     | Fcall of string * expr list
36     | PrintXY of expr * expr
37     | Print of expr
38     | LineVar of expr * expr
39     | LineVarColor of expr * expr * expr
40     | LineRaw of expr * expr * expr * expr
41     | LinePX of expr * expr * expr
42     | For of stmt * expr * stmt * stmt list
43     | While of expr * stmt list
44     | Ifelse of expr * stmt list * stmt list
45     | Return of expr
46     | Noexpr
47     | Fdecl of fdecl and
48         fdecl = {
49             fname : string;
50             args : stmt list;
51             body : stmt list;
52         }
53
54
55
56 type program = {
57     funcs : stmt list;
58     main : stmt list;
```

```

59
60
61
62
63 (* Pretty Print *)
64
65 let rec string_of_expr = function
66   | Literal_Num(l) -> string_of_float l ^ "0"
67   | Literal_Str(l) -> l
68   | Point(e1, e2) -> "(" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ ")"
69   | Literal_List(l) -> "[" ^ (String.concat "," (List.map string_of_expr l)) ^ "]"
70
71   | Id(s) -> s
72   | Binop(e1, o, e2) ->
73     string_of_expr e1 ^ " " ^ (match o with
74       | Add -> "+"
75       | Sub -> "-"
76       | Mul -> "*"
77       | Div -> "/"
78       | Equal -> "==" | Neq -> "!="
79       | Mod -> "%"
80       | And -> "&&" | Or -> "||"
81       | Square -> "**"
82       | Less -> "<" | Leq -> "<="
83       | Greater -> ">" | Geq -> ">="
84       ) ^ " " ^ string_of_expr e2
85
86
87
88
89 let rec string_of_stmt = function
90   | Expr(expr) -> string_of_expr expr ^ ""
91   | Var_Decl(tp, id) -> tp ^ " " ^ id ^ "\n"
92   | List_Decl(tp, id) -> "list " ^ tp ^ " " ^ id ^ "\n"
93   | Passign(v, e1, e) -> " " ^ string_of_expr v ^ " = " ^ (string_of_expr e1) ^ "\n"
94   | Assign(v, e) -> " " ^ string_of_expr v ^ " = " ^ (string_of_expr e)
95   | Append(v, e) -> string_of_expr v ^ ".append(" ^ (string_of_expr e) ^ ")"
96   | Pop(v) -> string_of_expr v ^ ".pop()\n"
97   | Remove(v, e) -> string_of_expr v ^ ".remove(" ^ (string_of_expr e) ^ ")"
98   | Fcall(v, el) -> v ^ "(" ^ (String.concat "," (List.map string_of_expr el)) ^ ")"
99   | PrintXY(e1, e2) -> "printXY( " ^ string_of_expr e1 ^ " , " ^ string_of_expr e2 ^ ")"
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
636
637
638
639
639
640
641
642
643
644
645
645
646
647
648
648
649
649
650
651
652
653
654
655
655
656
657
658
658
659
659
660
661
662
663
663
664
665
665
666
666
667
667
668
668
669
669
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
```

```

108     string_of_stmt body) )
109   ^ "\nend\n"
110 | While(e, body) -> "while " ^ string_of_expr e ^ " :\n" ^ (String.concat
111   "\n\t" (List.map string_of_stmt body)) ^ "\nend\n"
112 | Ifelse(e, succ_stmt, else_stmt) -> "if " ^ string_of_expr e ^ " :\n" ^ (
113   String.concat "\n\t" (List.map string_of_stmt succ_stmt)) ^ "\nelse :\n" ^
114   (String.concat "\n\t" (List.map string_of_stmt else_stmt)) ^ "end\n"
115 | Return(expr) -> "return " ^ string_of_expr expr ^ "\n"
116 | Noexpr -> ""
117 | Fdecl(f) -> string_of_fdecl f and
118   string_of_fdecl fdecl =
119     "fn " ^ fdecl.fname ^ "(" ^
120       (String.concat ", " (List.map (fun s -> string_of_stmt s) fdecl.
121         args)) ^
122       ") :\n" ^
123       (String.concat "" (List.map string_of_stmt fdecl.body)) ^
124       "\nend\n"
125
126
127 let string_of_program prog =
128   String.concat "\n" (List.map string_of_stmt prog.funcs)
129   ^ "\n" ^ "\n"
130   String.concat "\n" (List.map string_of_stmt prog.main)

```

./src/codegen.ml

```
1 open Ast
2 open Semcheck
3
4 let convert prog =
5   check prog.funcs;
6   check prog.main;
7   let rec create_expr = function
8     | Ast.Literal.Num(l) -> (string_of_float l) ^ "0"
9     | Ast.Literal.Str(l) -> l
10    | Ast.Point(e1,e2) -> "(float [2]) {(float)(" ^ create_expr e1 ^ "),(float)" ^ create_expr e2 ^ ")}"
11    | Ast.Literal.List(l) -> "{" ^ (String.concat "," (List.map
12      create_expr l)) ^ "}"
13    | Ast.Id(s) -> "(" ^ s ^ ")"
14    | Ast.Binop(e1, o, e2) ->
15      "(" ^ create_expr e1 ^ " " ^
16      (match o with
17        Add -> "+"
18        | Sub -> "-"
19        | Mul -> "*"
20        | Div -> "/"
21        | Equal -> "==" | Neq -> "!="
22        | Mod -> "%"
23        | And -> "&&"
24        | Or -> "||"
25        | Square -> "**"
26        | Less -> "<" | Led -> "<="
27        | Greater -> ">" | Geq -> ">="
28        ) ^ " " ^ create_expr e2 ^ ")"
29    | Ast.Bool(x) -> if x = True then "true" else "false"
30    | Ast.Length(v) -> create_expr v ^ ".size()"
31    | Ast.Access(v,e) -> create_expr v ^ "[int(" ^ (create_expr e) ^ ")]"
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
```

```

56   let rec create_stmt = function
57     | Ast.Expr(expr) -> create_expr expr
58     | Ast.Var_Decl(tp, id) ->
59       (match tp with
60        "num" -> "float" ^ " " ^ id ^ ";"^"\n"
61        | "string" -> "string" ^ " " ^ id ^ ";"^"\n"
62        | "point" -> "float" ^ " " ^ id ^ "[2];\n"
63        | _ -> "bool" ^ " " ^ id ^ ";"^"\n"
64      )
65     | Ast.List_Decl(tp, id) ->
66       (match tp with
67        "num" -> "vector <float>" ^ " " ^ id ^ ";"^"\n"
68        | "string" -> "vector <string>" ^ " " ^ id ^ ";"^"\n"
69        | "point" -> "vector <array<float , 2>>" ^ " " ^ id ^ ";"^"\n"
70        | _ -> "vector <bool>" ^ " " ^ id ^ ";"^"\n"
71      )
72    | Ast.Passign(v, e, e2) ->
73      (* Setting the point elements seperately *)
74      "_inbuildt_cp(" ^ create_expr v ^ ", (float [2])" ^ create_expr
75      e ^ ")"
76      | Ast.Assign(v, e) -> printAssign v e
77      | Ast.Append(v, e) -> create_expr v ^ ".push_back(" ^ ( create_expr
78      e ) ^ ");\n"
79      | Ast.Pop(v) -> create_expr v ^ ".pop_back();\n"
80      | Ast.Remove(v,e) -> create_expr v ^ ".erase(" ^ (create_expr v) ^ "
81      .begin() + " ^ ( create_expr e ) ^ ");\n"
82      | Ast.PrintXY(e1,e2) -> "put_in_svg(" ^ create_expr e1 ^ " , " ^
83      create_expr e2 ^ ");\n"
84      | Ast.Print(e) -> "put_in_svg( " ^ create_expr e ^ ");\n"
85      | Ast.LineVar(e1, e2) -> "put_in_svg (" ^ create_expr e1 ^ " , " ^
86      create_expr e2 ^ ");\n"
87      | Ast.LineVarColor(e1, e2, c) -> "put_in_svg (" ^ create_expr e1 ^ " ,
88      " ^ create_expr e2 ^ " , " ^ create_expr c ^ ");\n"
89      | Ast.LineRaw(e1, e2, e3, e4) -> "put_in_svg (" ^ create_expr e1 ^ " ,
90      " ^ create_expr e2
91      ^ " , " ^ create_expr e3 ^ " , " ^
92      create_expr e4 ^ ");\n"
93      | Ast.LinePX(e1, e2, e3) -> "put_in_svg (" ^ create_expr e1 ^ " , " ^
94      create_expr e2
95      ^ " , " ^ create_expr e3 ^ ");\n"
96      | Ast.For(s1, e1, s2, body) -> "for (" ^ create_stmt s1 ^ " ; "
97      create_expr e1 ^ " ; "
98      ^ ( remSemColon (create_stmt s2) ) ^ "
99      ) { \n"

```

```

100      ( String.concat ", " (List.map (fun s -> printFunArgs s)
101        fdecl.args) ) ^
102          "\n" ^
103          ( String.concat "" (List.map create_stmt fdecl.body) ) ^
104          "\n}\n"
105
106 in
107 "#include <iostream>\n#include <fstream>\n#include <vector>\n" ^
108 "using namespace std;\n"
109
110 "ofstream f;\n" ^
111 " // SVG content\n" ^
112 "void __inbuildt_cp (float a[2], float b[2]) {" ^
113 "a[0] = *b;" ^
114 "a[1] = *(b+1); " ^
115 "}" ^
116 "void put_in_svg (float p1[]) \n" ^
117 "{\n" ^
118 "f << '\n' << <text x='250' y='150'>\n';\n" ^
119 "f << p1[0] << '\n' << p1[1];\n" ^
120 "f << '\n' </text>\n';\n" ^
121 "}\n" ^
122 "void put_in_svg (float p1[], float p2[], string color=\"black\")\n" ^
123 "{\n" ^
124 "f << '\n' <line x1='\" + to_string(p1[0]) + '\" y1='\" + to_string(p1
125 [1]) + '\" x2='\" + to_string(p2[0]) + '\" y2='\" + to_string(p2[1]) + '\" style
126 ='stroke:\\" + color + \";stroke-width:1' />\n';\n" ^
127 "}\n" ^
128 "void put_in_svg (float p1[], float p2, float p3)\n" ^
129 "{\n" ^
130 "f << '\n' <line x1='\" + to_string(p1[0]) + '\" y1='\" + to_string(p1
131 [1]) + '\" x2='\" + to_string(p2) + '\" y2='\" + to_string(p3) + '\" style='
132 stroke:rgb(0,0,0);stroke-width:1' />\n';\n" ^
133 "}\n" ^
134 "void put_in_svg (float p1, float p2, float p3[])\n" ^
135 "{\n" ^
136 "f << '\n' <line x1='\" + to_string(p1) + '\" y1='\" + to_string(p2)+\"'
137 x2='\" + to_string(p3[0]) + '\" y2='\" + to_string(p3[1]) + '\" style='stroke:
138 rgb(0,0,0);stroke-width:1' />\n';\n" ^
139 "}\n" ^
140 "void put_in_svg (float x1, float y1, float x2, float y2)\n" ^
141 "{\n" ^
142 "f << '\n' <line x1='\" + to_string(x1) + '\" y1='\" + to_string(y1)+\"'
143 x2='\" + to_string(x2) + '\" y2='\" + to_string(y2) + '\" style='stroke:rgb
144 (0,0,0);stroke-width:1' />\n';\n" ^
145 "}\n" ^
146 "void put_in_svg (std::string content)\n" ^
147 "{\n" ^
148 "f << '\n' <text x='250' y='150'>\n';\n" ^
149 "f << content;\n" ^
150 "f << '\n' </text>\n';\n" ^
151 "}\n" ^

```

```

151     "    f << \"<text x='\" + to_string(p[0]) + '\" y='\"+ to_string(p[1])\n"
152     "    f << content;\n" ^
153     "    f << \"\\n</text>\\n\";\n" ^
154     "}\n" ^
155
156     "void put_in_svg(float content)\n" ^
157     "{\n" ^
158     "    f << \"<text x='250' y='150'>\\n\";\n" ^
159     "    f << content;\n" ^
160     "    f << \"\\n</text>\\n\";\n" ^
161     "}\n" ^
162
163     "//All user & library functions goes here\n" ^
164
165     String.concat "" (List.map create_stmt prog.funcs) ^
166
167     "//Main prog starts\n" ^
168
169     "int main() {\n" ^
170     (* change the name to be the filename.svg based on the file which is ran
*)
171     "    f.open (\"hello.svg\");\n" ^
172
173     "    // Prolog for the SVG image\n" ^
174     "    f << \"<svg xmlns='\"http://www.w3.org/2000/svg'\\\" xmlns:xlink='\"
http://www.w3.org/1999/xlink\\\" width='\"1024\\\" height='\"768\\\">\\";
175     "    f << \"\\n\";\n" ^
176
177     String.concat "" (List.map create_stmt prog.main) ^
178
179     "    f << \"</svg>\\\";\n" ^
180     "return 0;\n}\n"
181
182

```

./src/cppast.ml

```
1 (* converts plotter AST to C++ AST *)
2 open Tast
3 open Ast
4
5 (* Convert ast expr to Tast expr *)
6 let match_of e = match e with
7 | Ast.Add -> Tast.Add
8 | Ast.Sub -> Tast.Sub
9 | Ast.Mul -> Tast.Mul
10 | Ast.Div -> Tast.Div
11 | Ast.Mod -> Tast.Mod
12 | Ast.Equal -> Tast.Equal
13 | Ast.Neq -> Tast.Neq
14 | Ast.Less -> Tast.Less
15 | Ast.Leq -> Tast.Leq
16 | Ast.Greater -> Tast.Greater
17 | Ast.Geq -> Tast.Geq
18 | Ast.And -> Tast.And
19 | Ast.Or -> Tast.Or
20 | Ast.Square -> Tast.Square
21
22 let convert_to_cppast stmts_list =
23   let rec convert_to_texpr = function
24     | Ast.Literal_Num(v) -> Tast.Literal_Num(v)
25     | Ast.Literal_Str(v) -> Tast.Literal_Str(v)
26     | Ast.Literal_List(e) ->
27       let te = List.map (fun s -> convert_to_texpr s) e in
28       Tast.Literal_List(te)
29     | Ast.Bool(v)           -> if v = True then Tast.Bool(True) else Tast.Bool(
30       False)
31     | Ast.Length(v)         -> Tast.Length(convert_to_texpr v)
32     | Ast.Binop(e1, op, e2) ->
33       let tel = convert_to_texpr e1 in
34       let te2 = convert_to_texpr e2 in
35       let top = match_of op in
36       Tast.Binop(tel, top, te2)
37     | Ast.Id(v)             -> Tast.Id(v)
38     | Ast.Access(v, e) ->
39       let texpr_acc = convert_to_texpr e in
40       let tv = convert_to_texpr v in
41       Tast.Access(tv, texpr_acc)
42     | Ast.Point(e1, e2) ->
43       let tel = convert_to_texpr e1 in
44       let te2 = convert_to_texpr e2 in
45       Tast.Point(tel, te2)
46
47
48 (* Convert ast stmt to Tast stmt *)
49 let rec convert_to_tstmt stmt = function
50   | Ast.Expr(e) ->
51     let texpr = convert_to_texpr e in
52     Tast.Texpr(texpr)
53   | Ast.Var_Decl(t, e) -> Tast.Var_Decl(t, e)
54   | Ast.List_Decl(t, id) -> Tast.List_Decl(t, id)
55   | Ast.Passign(v, e1, e2) -> Tast.Noexpr
56   | Ast.Assign(v, e) ->
57     let tel = convert_to_texpr e in
58     let tv = convert_to_texpr v in
```

```

59      Tast.Assign(tv, tel)
60  | Ast.Print(e) -> Tast.Print(convert_to_texpr e)
61  | Ast.Append(v,e) ->
62    let texpr_app = convert_to_texpr e in
63    let tv = convert_to_texpr v in
64    Tast.Append(tv, texpr_app)
65  | Ast.Remove(v,e) ->
66    let texpr_rm = convert_to_texpr e in
67    let tv = convert_to_texpr v in
68    Tast.Remove(tv, texpr_rm)
69  | Ast.Pop(e) -> Tast.Pop(convert_to_texpr e)
70  | Ast.LineVar(e1,e2) ->
71    let texpr1 = convert_to_texpr e1 in
72    let texpr2 = convert_to_texpr e2 in
73    Tast.LineVar(texpr1, texpr2)
74  | Ast.LineRaw(e1,e2,e3,e4) ->
75    let texpr1 = convert_to_texpr e1 in
76    let texpr2 = convert_to_texpr e2 in
77    let texpr3 = convert_to_texpr e3 in
78    let texpr4 = convert_to_texpr e4 in
79    Tast.LineRaw(texpr1, texpr2, texpr3, texpr4)
80  | Ast.For(s1, e1, s2, body) ->
81    let tstmt1 = convert_to_tstmt stmt s1 in
82    let texpr1 = convert_to_texpr e1 in
83    let tstmt2 = convert_to_tstmt stmt s2 in
84    let tbody = List.map (fun s -> convert_to_tstmt stmt s) body in
85    Tast.For(tstmt1, texpr1, tstmt2, tbody)
86  | Ast.While(e, body) ->
87    let texpr = convert_to_texpr e in
88    let tbody = List.map (fun s -> convert_to_tstmt stmt s) body in
89    Tast.While(texpr, tbody)
90
91  | Ast.Ifelse(e, succ_stmt, else_stmt) ->
92    let texpr = convert_to_texpr e in
93    let tsucc_stmt = List.map (fun s -> convert_to_tstmt stmt s) succ_stmt
94    in
95    let telse_stmt = List.map (fun s -> convert_to_tstmt stmt s) else_stmt
96    in
97    Tast.Ifelse(texpr, tsucc_stmt, telse_stmt)
98  | Ast.Return(e) -> Tast.Return(convert_to_texpr e)
99  | Ast.Fdecl(f) ->
100    let f_name = f.fname in
101    let f_args = List.map (fun s -> convert_to_tstmt stmt s) f.args in
102    let f_body = List.map (fun s -> convert_to_tstmt stmt s) f.body in
103    Tast.Fdecl({
104      fname = f_name;
105      args = f_args;
106      body = f_body;
107    })
108  | Ast.Fcall(f, el) ->
109    let tel = List.map (fun s -> convert_to_texpr s) el in
110    Tast.Fcall(f, tel)
111  | Ast.PrintXY (el, e2) ->
112    let te1 = convert_to_texpr el in
113    let te2 = convert_to_texpr e2 in
114    Tast.PrintXY(te1, te2)
115  | Ast.LinePX(el, e2, e3) ->
116    let te1 = convert_to_texpr el in
117    let te2 = convert_to_texpr e2 in
118    let te3 = convert_to_texpr e3 in

```

```
117      Tast.LinePX(te1, te2, te3)
118      | Ast.Noexpr -> Tast.Noexpr
119
120  in
121  List.map (fun s -> convert_to_tstmt s) stmts_list;
122
```

./src/err.ml

```
1 open Printf
2 open Lexing
3 open Parsing
4
5 exception ScanError of string
6
7 let print_position lexbuf msg =
8   let start = lexeme_start_p lexbuf in
9   let finish = lexeme_end_p lexbuf in
10  (fprintf stderr "Line %d: char %d..%d: %s: \"%s\"\n"
11   start.pos_lnum
12   (start.pos_cnum - start.pos_bol)
13   (finish.pos_cnum - finish.pos_bol)
14   msg
15   (Lexing.lexeme lexbuf))
16
17 let lexer_from_channel fname ch =
18   let lex = Lexing.from_channel ch in
19   let pos = lex.lex_curr_p in
20   lex.lex_curr_p <- { pos with pos_fname = fname; pos_lnum = 1; } ;
21   lex
22
23 let lexer_from_string str =
24   let lex = Lexing.from_string str in
25   let pos = lex.lex_curr_p in
26   lex.lex_curr_p <- { pos with pos_fname = ""; pos_lnum = 1; } ;
27   lex
28
29
30
```

./src/incLib.py

```
1 import sys
2 f = open(sys.argv[1])
3 s = f.readlines()
4 f.close()
5
6 #Get the include list
7 incLibs = []
8 codeLines = []
9 for i in s:
10     j=i.strip()
11     if len(j.split()) == 2 and j.split()[0] == "include":
12         incLibs.append(j.split()[1])
13         continue
14     codeLines.append(i)
15
16 #Get all the library code to be attached
17 libCode = []
18 for lib in incLibs:
19     f = open("library/" + lib + ".plt")
20     lc = f.readlines()
21     f.close()
22     libCode += lc
23
24 #Save the original code in .plt_tmp file
25 fn= open(sys.argv[1] + '_tmp', 'w')
26 for item in s:
27     fn.write("%s" % item)
28 fn.close()
29
30 #overwrite the new with this code
31 fullCode = libCode + codeLines
32 fn= open(sys.argv[1], 'w')
33 for item in fullCode:
34     fn.write("%s" % item)
35 fn.close()
36
37
```

./src/Makefile

```
1 OBJS = ast.cmo err.cmo sast.cmo parser.cmo scanner.cmo semcheck.cmo codegen.cmo plotter.cmo
2
3 TARFILES = Makefile scanner.mll parser.mly \
4             ast.ml codegen.ml plotter.ml
5
6 plotter : $(OBJS)
7     ocamlc -o plotter $(OBJS)
8
9 scanner.ml : scanner.mll
10    ocamllex scanner.mll
11
12 parser.ml parser.mli : parser.mly
13    ocamlyacc parser.mly
14
15 %.cmo : %.ml
16    ocamlc -c $<
17
18 %.cmi : %.mli
19    ocamlc -c $<
20
21 .PHONY : clean
22
23 clean :
24     rm -f plotter parser.ml parser.mli scanner.ml *.svg testall.log \
25     *.cmo *.cmi *.out *.diff
26 test:
27     python test.py
28 #
29 ast.cmo:
30 ast.cmx:
31 sast.cmo:
32 sast.cmx:
33 semcheck.cmo:
34 semcheck.cmx:
35 codegen.cmo: ast.cmo
36 codegen.cmx: ast.cmx
37 plotter.cmo: scanner.cmo parser.cmi codegen.cmo \
38         ast.cmo
39 plotter.cmx: scanner.cmx parser.cmx codegen.cmx \
40         sast.cmx ast.cmx
41 parser.cmo: err.cmo ast.cmo parser.cmi
42 parser.cmx: ast.cmx parser.cmi
43 scanner.cmo: err.cmo parser.cmi
44 scanner.cmx: err.cmx parser.cmx
45 parser.cmi: ast.cmo
46 err.cmo:
```

./src/parser.mly

```
1 %{
2   open Ast
3   open Lexing
4   open Parsing
5
6   let num_errors = ref 0
7
8   let parse_error msg = (* called by parser function on error *)
9   let start = symbol_start_pos() in
10  let final = symbol_end_pos() in
11  Printffprintf stdout "Characters: %d..%d: %s\n"
12    (start.pos_cnum - start.pos_bol) (final.pos_cnum - final.pos_bol) msg;
13  incr num_errors;
14  flush stdout;
15  exit 0
16 %}
17 %token EOL LPAREN RPAREN LBRACK RBRACK
18 %token PLUS MINUS TIMES DIVIDE MOD ASSIGN
19 %token EQUAL NEQ LESS GREATER LEQ GEQ
20 %token AND OR NOT
21 %token SEMI COMMA COMMENT OF COLON
22 %token STRING NUM BOOL POINT NONE LIST HASH
23 %token APPEND POP REMOVE AT LENGTH OF
24 %token TRUE FALSE
25 %token RETURN IF ELSE FOR WHILE END BREAK CONTINUE THEN FN
26 %token PRINTXY PRINT
27 %token LINE
28 %token <float> LIT_NUM
29 %token <string> LIT_STR
30 %token <string> ID
31 %token EOF
32
33
34 %nonassoc ELSE END BREAK CONTINUE
35
36 %right ASSIGN
37 %left AND OR
38 %left NOT
39 %left EQUAL NEQ
40 %left LESS GREATER LEQ GEQ
41 %left PLUS MINUS
42 %left TIMES DIVIDE MOD
43
44 %start program
45 %type < Ast.program> program
46
47 %%
48
49 program:
50   stmt EOF { $1 }
51
52 stmt:
53   /* nothing */      { { funcs=[]; main=[] } }
54   | func_stmt stmt  { { funcs = $1::$2.funcs; main= $2.main } }
55   | other_stmt stmt { { funcs = $2.funcs; main= $1::$2.main } }
56
57
58
59
```

```

60  /* =====
61          Variable
62  ===== */
63
64  literal:
65      | LIT_NUM      { Literal_Num($1) }
66      | LIT_STR      { Literal_Str($1) }
67      | point        { $1 }
68      | literal_list { $1 }
69
70  literal_list:
71      | LBRACK list_elements RBRACK { Literal_List($2) }
72      | LBRACK list_elements { (parse_error "Syntax error: Left [ is
unmatched with right ].");}
73
74
75  list_elements:
76      /* nothing */ { [] }
77      | list_content { List.rev $1 }
78
79  list_content:
80      expr           { [$1] }
81      | list_content COMMA expr { $3 :: $1 }
82
83  point:
84      | LPAREN expr COMMA expr RPAREN   { Point($2, $4) }
85      | LPAREN expr COMMA expr { (parse_error "Syntax error: Left ( is
unmatched with right ).");}
86      | LPAREN expr COMMA expr { (parse_error "Right ) is unmatched with
left ( .");}
87      | LPAREN expr expr RPAREN { (parse_error "Missing , .");}
88      | LPAREN expr COMMA RPAREN { (parse_error "Missing y co-od of point
."); }
89      | LPAREN COMMA RPAREN { (parse_error "Missing x and y co-od of point
."); }
90      | LPAREN COMMA expr RPAREN { (parse_error "Missing x co-od of point
."); }
91
92
93  primitive:
94      | BOOL          {"bool"}
95      | NUM           {"num"}
96      | STRING         {"string"}
97      | POINT          {"point"}
98
99  /*non_primitive:
100   | LIST primitive */
101
102  data_type:
103      | primitive { $1 }
104      /*| non_primitive { $1 }*/
105      /* Point, List and hash are to be added here */
106
107  vdecl:
108      | vdecl_single    { $1 }
109
110  /* only to be used to declare one variable */
111  /* Reusability in functions */
112  vdecl_single:
113      | primitive_var_decl { $1 }

```

```

114     | list_decl { $1 }
115
116 primitive_var_decl:
117     | primitive ID { Var_Decl($1, $2) }
118     | primitive { (parse_error "Missing variable name"); }
119
120 list_decl:
121     | LIST primitive ID { List_Decl($2, $3) }
122     | LIST ID { (parse_error "Missing list type"); }
123     | LIST primitive { (parse_error "Missing list name"); }
124
125 /*
126     For futute to do, multiple variable declaration
127     eg : num a,b,c
128
129 id_list:
130     ID {$1}
131     | ID COMMA id_list { $1, $3 }
132
133 */
134
135 /*
136 =====
137             Statements
138 ===== */
139
140 func_stmt:
141     | fdecl { $1 }
142
143 other_stmt:
144     | expr EOL { Expr($1) }
145     | cond_stmt EOL { $1 }
146     | list_stmt EOL { $1 }
147     | assign_stmt EOL { $1 }
148     | PRINTXY LPAREN expr COMMA expr RPAREN EOL { PrintXY($3, $5) }
149     | PRINT expr EOL { Print($2) }
150     | PRINT EOL { (parse_error "Nothing to print!"); }
151     | line EOL { $1 }
152     | fcall EOL { $1 }
153     | RETURN expr EOL { Return($2) }
154     | RETURN EOL { (parse_error "Nothing to return!"); }
155     | vdecl EOL { $1 }
156     | loop_stmt EOL { $1 }
157     | EOL { Noexpr }
158
159 cond_stmt:
160     | IF expr COLON EOL other_stmt_list END { Ifelse($2, $5, $9) }
161     | IF expr EOL other_stmt_list END { (parse_error "Missing colon :"); }
162     | IF expr COLON EOL other_stmt_list END { (parse_error "Missing end"); }
163     | IF COLON EOL other_stmt_list END { (parse_error "Missing if condition"); }
164     | IF expr COLON EOL other_stmt_list ELSE COLON EOL other_stmt_list END { Ifelse($2, $5, $9) }
165     | IF expr EOL other_stmt_list ELSE COLON EOL other_stmt_list END { (parse_error "Missing colon : after if"); }
166     | IF expr COLON EOL other_stmt_list ELSE EOL other_stmt_list END { (parse_error "Missing else"); }
167

```

```

168     (parse_error "Missing colon : after else "); }
169     | IF expr EOL other_stmt_list ELSE COLON EOL other_stmt_list      { 
170     (parse_error "Missing end "); }
171     | IF EOL other_stmt_list ELSE COLON EOL other_stmt_list END      { 
172     (parse_error "Missing if condition"); }

173 list_stmt:
174     | ID OF APPEND LPAREN expr RPAREN      { Append( Id($1), $5) }
175     | ID OF POP LPAREN RPAREN             { Pop( Id($1) ) }
176     | ID OF REMOVE LPAREN expr RPAREN    { Remove( Id($1), $5 ) }

177 list_assign:
178     | ID ASSIGN literal_list {Assign(Id($1), $3) }
179     | ID literal_list      {(parse_error "Missing assignment operator
")}; }
180     | ID ASSIGN          {((parse_error "Missing element(s) ")); }

181 assign_stmt:
182     | ID ASSIGN expr      { Assign(Id($1), $3) }
183     | list_assign         { $1 }

184

185 line:
186     | LINE LPAREN ID COMMA ID RPAREN { LineVar(Id($3), Id($5)) }
187     | LINE LPAREN ID COMMA ID COMMA expr RPAREN { LineVarColor(Id($3),
188     Id($5), $7) }
189     | LINE ID COMMA ID RPAREN { (parse_error "Missing left paren ");}
190     | LINE LPAREN ID COMMA ID { (parse_error "Missing right paren ");}
191     | LINE LPAREN ID ID RPAREN { (parse_error "Missing ,");}
192     | LINE LPAREN LPAREN expr COMMA expr RPAREN COMMA LPAREN expr COMMA
193     expr RPAREN RPAREN { LineRaw($4, $6, $10, $12) }
194     | LINE LPAREN expr COMMA expr RPAREN COMMA LPAREN expr COMMA expr
195     RPAREN RPAREN { (parse_error "Missing left paren ");}
196     | LINE LPAREN LPAREN expr COMMA expr RPAREN COMMA expr COMMA expr
197     RPAREN RPAREN { (parse_error "Missing left paren ");}
198     | LINE LPAREN LPAREN expr COMMA expr COMMA LPAREN expr COMMA expr
199     RPAREN RPAREN { (parse_error "Missing right paren ");}
200     | LINE LPAREN LPAREN expr COMMA expr RPAREN COMMA LPAREN expr COMMA
201     expr RPAREN { (parse_error "Missing right paren ");}
202     | LINE LPAREN LPAREN expr expr RPAREN COMMA LPAREN expr COMMA expr
203     RPAREN RPAREN { (parse_error "Missing ,");}
204     | LINE LPAREN LPAREN expr COMMA expr RPAREN LPAREN expr COMMA expr
205     RPAREN RPAREN { (parse_error "Missing ,");}
206     | LINE LPAREN LPAREN expr COMMA expr RPAREN COMMA ID RPAREN { LinePX
207     ($4, $6, Id($9)) }
208     | LINE LPAREN expr COMMA expr RPAREN COMMA ID RPAREN { (parse_error
209     "Missing left paren ");}
210     | LINE LPAREN LPAREN expr COMMA expr RPAREN COMMA ID { (parse_error
211     "Missing right paren ");}
212     | LINE LPAREN LPAREN expr COMMA expr COMMA ID RPAREN { (parse_error
213     "Missing right paren ");}
214     | LINE LPAREN LPAREN expr expr RPAREN COMMA ID RPAREN { (parse_error
215     "Missing ,");}
216     | LINE LPAREN LPAREN expr COMMA expr RPAREN ID RPAREN { (parse_error
217     "Missing ,");}
218     | LINE LPAREN ID COMMA LPAREN expr COMMA expr RPAREN RPAREN { LinePX
219     ($6, $8, Id($3)) }
220     | LINE ID COMMA LPAREN expr COMMA expr RPAREN RPAREN { (parse_error

```

```

    "Missing left paren ");}
208     | LINE LPAREN ID COMMA expr COMMA expr RPAREN RPAREN { (parse_error
    "Missing left paren");}
209     | LINE LPAREN ID LPAREN expr COMMA expr RPAREN RPAREN { (
210         parse_error "Missing ,");}
211     | LINE LPAREN ID COMMA LPAREN expr expr RPAREN RPAREN { (
212         parse_error "Missing ,");}
213     | LINE LPAREN ID COMMA LPAREN expr COMMA expr RPAREN { (parse_error
    "Missing right paren");}
214
215     loop_stmt:
216         | FOR assign_stmt SEMI expr SEMI assign_stmt COLON EOL
217             other_stmt_list END { For($2, $4, $6, List.rev $9) }
218         | FOR assign_stmt expr SEMI assign_stmt COLON EOL other_stmt_list
219             END { (parse_error "Missing ;");}
220         | FOR assign_stmt SEMI expr assign_stmt COLON EOL other_stmt_list
221             END { (parse_error "Missing ;");}
222         | FOR assign_stmt SEMI expr SEMI assign_stmt EOL other_stmt_list
223             END { (parse_error "Missing :");}
224         | FOR assign_stmt SEMI expr SEMI assign_stmt COLON EOL other_stmt_list
225             { (parse_error "Missing end");}
226         | FOR SEMI expr SEMI assign_stmt COLON EOL other_stmt_list END { (
227             parse_error "Missing initialization statement");}
228         | FOR assign_stmt SEMI SEMI assign_stmt COLON EOL other_stmt_list
229             END { (parse_error "Missing condition statement");}
230         | FOR assign_stmt SEMI expr SEMI COLON EOL other_stmt_list END { (
231             parse_error "Missing increment/decrement statement");}
232         | WHILE expr COLON EOL other_stmt_list END {While($2, List.rev $5)}
233         | WHILE expr COLON EOL other_stmt_list END { (parse_error "Missing :
   ");}
234         | WHILE expr COLON EOL other_stmt_list END { (parse_error "Missing
end");}
235         | WHILE expr COLON EOL other_stmt_list END { (parse_error "Missing
expression in while");}
236
237     other_stmt_list:
238         { [] }
239         | other_stmt_list other_stmt { $2 :: $1 }
240
241     stmt_list:
242         { [] }
243         | stmt_list other_stmt { $2 :: $1 }
244
245 /* =====
246          Functions
247 ===== */
248
249 /* No locals. as variables can be declared at any point */
250
251     fdecl:
252         | FN ID LPAREN args_opt RPAREN COLON EOL stmt_list END EOL
253             { Fdecl({ fname = $2;
254                 args = List.rev $4;
255                 body = List.rev $8 }) }
256         | FN ID LPAREN args_opt RPAREN COLON EOL stmt_list END EOL { (
257             parse_error "Missing colon :");
258             | FN ID args_opt RPAREN COLON EOL stmt_list END EOL { (
259                 parse_error "Missing left paren");}
260             | FN ID LPAREN args_opt COLON EOL stmt_list END EOL { (
261                 parse_error "Missing right paren");}
262

```

```

249      | FN LPAREN args_opt RPAREN EOL stmt_list END EOL           { (
250      parse_error "Missing function name : "); }
251
252      args_opt:
253      { [] }
254      | args_list { List.rev $1 }
255
256      args_list:
257      arg                      { [$1] }
258      | args_list COMMA arg    { $3 :: $1 }
259      | args_list arg          { (parse_error "Missing , "); }
260      | args_list COMMA        { (parse_error "Missing arg "); }
261
262      arg:
263      vdecl_single             { $1 }
264
265      /* Function Call */
266      fcall:
267      | ID LPAREN fparam RPAREN { Fcall($1, $3) }
268
269      fparam:
270      { [] }
271      | expr                   { [$1] }
272      | fparam COMMA expr     { $3 :: $1 }
273      | fparam expr            { (parse_error "Missing , "); }
274      | fparam COMMA           { (parse_error "Missing expr "); }
275
276  /* =====
277      Expressions
278  ===== */
279
280      expr:
281      | arith_expr              { $1 }
282      | log_expr                { $1 }
283      | LPAREN expr RPAREN     { $2 }
284      | LPAREN RPAREN           { (parse_error "Missing expression "); }
285      | LPAREN expr              { (parse_error "Missing right paren "); }
286
287      log_expr:
288      | expr EQUAL expr         { Binop($1, Equal, $3) }
289      | expr NEQ expr           { Binop($1, Neq, $3) }
290      | expr LESS expr          { Binop($1, Less, $3) }
291      | expr LEQ expr           { Binop($1, Leq, $3) }
292      | expr GREATER expr       { Binop($1, Greater, $3) }
293      | expr GEQ expr           { Binop($1, Geq, $3) }
294      | expr EQUAL               { (parse_error "Missing second expression"); }
295      | expr NEQ                 { (parse_error "Missing second expression"); }
296      | expr LESS                { (parse_error "Missing second expression"); }
297      | expr LEQ                 { (parse_error "Missing second expression"); }
298      | expr GREATER              { (parse_error "Missing second expression"); }
299      | expr GEQ                 { (parse_error "Missing second expression"); }
300      | log_expr AND log_expr   { Binop($1, And, $3) }
301      | log_expr OR log_expr    { Binop($1, Or, $3) }
302      | log_expr OR               { (parse_error "Missing second expression"); }
303      | log_expr AND              { (parse_error "Missing second expression"); }
304
305
306      arith_expr :
307      | LPAREN arith_expr RPAREN { $2 }

```

```

308 | LPAREN arith_expr      { (parse_error "Missing right paren "); }
309 | LPAREN RPAREN         { (parse_error "Missing expr "); }
310 | arith_expr PLUS      { Binop($1, Add, $3) }
311 | arith_expr MINUS     { Binop($1, Sub, $3) }
312 | arith_expr TIMES     { Binop($1, Mul, $3) }
313 | arith_expr DIVIDE    { Binop($1, Div, $3) }
314 | arith_expr MOD       { Binop($1, Mod, $3) }
315 | arith_expr PLUS      { (parse_error "Missing second arithmetic
316 | expression "); }
316 | arith_expr MINUS     { (parse_error "Missing second arithmetic
317 | expression "); }
317 | arith_expr TIMES     { (parse_error "Missing second arithmetic
318 | expression "); }
318 | arith_expr DIVIDE    { (parse_error "Missing second arithmetic
319 | expression "); }
319 | arith_expr MOD       { (parse_error "Missing second arithmetic
320 | expression "); }
320 | list_exprs            { $1 }
321 | atom                  { $1 }
322
323 list_exprs:
324 | ID OF LENGTH LPAREN RPAREN { Length( Id($1) ) }
325 | ID LENGTH LPAREN RPAREN { (parse_error "Missing . ");}
326 | ID OF LENGTH RPAREN   { (parse_error "Missing left paren
327 | "); }
327 | ID OF LENGTH LPAREN   { (parse_error "Missing right
328 | paren "); }
328 | ID OF AT LPAREN expr RPAREN { Access( Id($1), $5 ) }
329 | ID AT LPAREN expr RPAREN { (parse_error "Missing of ");}
330 | ID OF AT expr RPAREN  { (parse_error "Missing left paren
331 | "); }
331 | ID OF AT LPAREN expr  { (parse_error "Missing right
332 | paren "); }
332 | ID LBRACK expr RBRACK { Access( Id($1), $3 ) }
333 | ID LBRACK RBRACK     { (parse_error "Missing expr ");}
334 | ID LBRACK expr        { (parse_error "Missing right
335 | brack "); }
336
337
338 atom:
339 | literal               { $1 }
340 | TRUE                  { Bool(True) }
341 | FALSE                 { Bool(False) }
342 | ID                     { Id($1) }
343
344

```

./src/plotter.ml

```
1  type action = Ast | Codegen (* | Tast | Cppast *)
2
3
4  let _ =
5    let action = if Array.length Sys.argv > 1 then
6      List.assoc Sys.argv.(1) [ ("‐a", Ast);
7                            ("‐c", Codegen); (* ("‐t", Tast) *) ]
8    else Codegen in
9  let lexbuf = Lexing.from_channel stdin in
10 let program = Parser.program Scanner.token lexbuf in
11 match action with
12   Ast → print_string (Ast.string_of_program (program))
13 | Codegen → print_string (Codegen.convert (program))
14 (*| Tast → Cppast.convert_to_cppast (List.rev program)*)
15
16
```

./src/plt

```
1 #!/bin/bash
2
3 # Simple runnable shell script to avoid the manual work
4 # This script runs the plotter code, gets the c++ output
5 # compiles it and executes its output(which generates
6 # the svg). This is then renamed, & temp files are
7 # removed
8
9 # 1st arg - input .plt file
10
11 f=$1
12 l=${#f}
13
14 file=${f::l-4}
15
16 #attach the libraries
17 python incLib.py $1
18
19 #EOL EOF issue fix
20 printf "\n" >> $1
21
22 ./plotter < $1 > $file.cpp
23
24 g++ -std=c++11 $file.cpp
25
26 ./a.out
27
28 mv hello.svg $file.svg
29
30 #reset the file to normal
31 rm $file.plt
32 cp $file.plt.tmp $file.plt
33 rm $file.plt.tmp
34
35 #remove the cpp and out files
36 rm $file.cpp a.out
37
38
```

./src/sast.ml

```
1  open Ast
2
3
4  type t = Num | Bool | String | Point | List
5      | ListNum | ListString | ListPoint | ListBool
6
7  type texpr =
8      Literal_Num of float * t
9      Literal_Str of string * t
10     Literal_List of texpr list * t
11     Point of texpr * texpr * t
12     Binop of texpr * Ast.ops * texpr * t
13     Id of string * t
14     Bool of bool * t
15     Length of texpr * t
16     Access of texpr * texpr * t * t
17
18
19  type tstmt =
20      Expr of texpr * t
21      | Var_Decl of string * string * t
22      | List_Decl of string * string * t
23      | Passign of texpr * texpr
24      | Assign of texpr * texpr
25      | Append of texpr * texpr
26      | Remove of texpr * texpr
27      | Pop of texpr
28      | Fcall of string * texpr list
29      | PrintXY of texpr * texpr
30      | Print of texpr
31      | LineVar of texpr * texpr
32      | LineVarColor of texpr * texpr * texpr
33      | LineRaw of texpr * texpr * texpr * texpr
34      | LinePX of texpr * texpr * texpr
35      | For of tstmt * texpr * tstmt * tstmt list
36      | While of texpr * tstmt list
37      | Ifelse of texpr * tstmt list * tstmt list
38      | Return of texpr
39      | Noexpr
40      | Fdecl of fdecl and
41          fdecl = {
42              fname : string;
43              args : tstmt list;
44              body : tstmt list;
45          }
46
47  type program = tstmt list
48
49  (* Pretty Print Stuff *)
50
51  let typeof t =
52      match t with
53          | Num -> "num"
54          | Bool -> "bool"
55          | String -> "string"
56          | Point -> "point"
57          | ListNum -> "listNum"
58          | ListString -> "listString"
59          | ListBool -> "listBool"
```

```

60      | ListPoint -> "listPoint"
61      | List -> "list"
62
63
64  let rec string_of_texpr = function
65    Literal_Num(l, t) -> string_of_float l ^ typeof t
66    | Literal_Str(l, t) -> l ^ typeof t
67    | Point(e1, e2, t) -> "(" ^ string_of_texpr e2 ^ "," ^ string_of_texpr e2
68    ^ ")" ^ typeof t
69    | Literal_List(l, t) -> typeof t
70    | Id(s, t) -> s ^ typeof t
71    | Length(v, _) -> string_of_texpr v ^ ".length()"
72    | Binop(e1, o, e2, t) ->
73      string_of_texpr e1 ^ " " ^
74      (match o with
75       Add -> "+" | Sub -> "-" | Mul -> "*" | Div -> "/"
76       | Equal -> "==" | Neq -> "!="
77       | Mod -> "%"
78       | And -> "&&" | Or -> "||"
79       | Square -> "*"
80       | Less -> "<" | Leq -> "<="
81       | Greater -> ">" | Geq -> ">="
82       ) ^ " " ^ string_of_texpr e2 ^ typeof t
83    | Bool(x, t) -> if x = True then "true" else "false" ^ typeof t
84    | Access(v, e, t1, t2) -> string_of_texpr v ^ ".at(" ^ (string_of_texpr e
85    ) ^ ")" ^ " //of type " ^ typeof t1 ^ typeof t2
86
87
88  let rec string_of_tstmt = function
89    Expr(expr, t) -> string_of_texpr expr ^ "\n" ^ typeof t
90    | Noexpr -> ""
91    | Var_Decl(tp, id, t) -> tp ^ " " ^ id ^ "\n" ^ typeof t
92    | List_Decl(tp, id, t) -> "list" ^ tp ^ " " ^ id ^ "\n" ^ typeof t
93    | Passign(v, e1) -> string_of_texpr v ^ " = " ^ (string_of_texpr e1) ^ "\n"
94    | Assign(v, e) -> string_of_texpr v ^ " = " ^ (string_of_texpr e)
95    | Append(v, e) -> string_of_texpr v ^ ".append(" ^ (string_of_texpr e) ^
96    ")"
97    | Remove(v, e) -> string_of_texpr v ^ ".remove(" ^ (string_of_texpr e) ^
98    ")"
99    | Pop(v) -> string_of_texpr v ^ ".pop()"
100   | Fcall(v, e1) -> v ^ "(" ^ (String.concat ", " (List.map string_of_texpr
101     e1)) ^ ")\n"
102   | PrintXY(e1, e2) -> "printXY(" ^ string_of_texpr e1 ^ "," ^ string_of_texpr
103     e2 ^ ")\n"
104   | Print(e) -> "print " ^ string_of_texpr e ^ "\n"
105   | LineVar(e1, e2) -> "line (" ^ string_of_texpr e1 ^ "," ^ string_of_texpr
106     e2 ^ ") " ^ "\n"
107   | LineVarColor(e1, e2, c) -> "line (" ^ string_of_texpr e1 ^ "," ^ string_of_texpr
108     e2 ^ "," ^ string_of_texpr c ^ ")" ^ "\n"
109   | LineRaw(e1, e2, e3, e4) -> "line (" ^ string_of_texpr e1 ^ "," ^ string_of_texpr
110     e2 ^ ")" ^ "," ^ "(" ^ string_of_texpr e3 ^ "," ^ string_of_texpr e4
111     ^ ") " ^ "\n"
112   | LinePX(e1, e2, e3) -> "line (" ^ string_of_texpr e1 ^ "," ^ string_of_texpr
113     e2 ^ ")" ^ "," ^ string_of_texpr e3 ^ ") " ^ "\n"
114   | For(s1, e1, s2, body) -> "for " ^ string_of_tstmt s1 ^ ";" ^ string_of_texpr
115     e1 ^ ";" ^ string_of_tstmt s2 ^ ":" ^ "\n" ^
116     (String.concat "\n\t" (List.map

```

```

107     string_of_tstmt body) ) ^ "\nend\n"
108   | While(e, body) -> "while " ^ string_of_texpr e ^ " :\n" ^ (String.concat
109     "\n\t" (List.map string_of_tstmt body)) ^ "\nend\n"
110   | Ifelse(e, s1, s2) -> "if " ^ string_of_texpr e ^ " :\n" ^ (String.concat
111     "\n\t" (List.map string_of_tstmt s1)) ^ "\nelse :\n" ^ (String.concat "\n\
112       " (List.map string_of_tstmt s2)) ^ "\nend\n"
113   | Return(expr) -> "return " ^ string_of_texpr expr ^ "\n"
114   | Fdecl(f) -> string_of_fdecl f and
115     string_of_fdecl fdecl =
116       "fn " ^ fdecl.fname ^ "(" ^
117         ( String.concat ", " (List.map (fun s -> string_of_tstmt s) fdecl.
118           args) ) ^
119         "):\n" ^
120         ( String.concat "" (List.map string_of_tstmt fdecl.body) ) ^
121         "\nend\n"
122 let string_of_tprogram stmts =
123   String.concat "\n" (List.map string_of_tstmt stmts)
124
125

```

./src/scanner.mll

```
1  {
2    open Err
3    open Parser
4    open Lexing
5  }
6
7  rule token = parse
8    [ ' ' '\t' '\r' ] { token lexbuf } (*whitespace*)
9  (*punctuations*)
10 | '\n' { EOL }
11 | "/*" { token lexbuf }
12 | '+' { PLUS } (*operators*)
13 | '-' { MINUS }
14 | '*' { TIMES }
15 | '/' { DIVIDE }
16 | '%' { MOD }
17 | ">=" { GEQ }
18 | "<=" { LEQ }
19 | "==" { EQUAL }
20 | "!=" { NEQ }
21 (*| "/*" { SQUARE }*)
22 | '>' { GREATER }
23 | '<' { LESS }
24 | '(' { LPAREN }
25 | ')' { RPAREN }
26 | '[' { LBRACK }
27 | ']' { RBRACK }
28 | ',' { COMMA }
29 | '.', { OF }
30 | ':=' { ASSIGN }
31 | "and" { AND }
32 | "or" { OR }
33 (*| "in" { IN }*)
34 | "not" { NOT }
35 | ';' { SEMI }
36 | ':' { COLON }
37 | "string" { STRING }
38 | "num" { NUM }
39 | "bool" { BOOL }
40 | "point" { POINT }
41 | "if" { IF } (*controlling sequence*)
42 | "else" { ELSE }
43 | "then" { THEN }
44 | "end" { END }
45 | "for" { FOR }
46 | "while" { WHILE }
47 | "break" { BREAK }
48 | "continue" { CONTINUE }
49 | "print" { PRINT }
50 | "printXY" { PRINTXY }
51 | "line" { LINE }
52 | "none" { NONE }
53 | "list" { LIST }
54 | "hash" { HASH }
55
56 (* List related stuff *)
57 (* the '.' in front of append ensures its used as a function *)
58 | "append" { APPEND }
59 | "pop" { POP }
```

```

60  | "remove" { REMOVE }
61  | "at" { AT }
62  | "length" { LENGTH }
63
64  | "fn" { FN }
65  | "return" { RETURN }
66  | "true" { TRUE }
67  | "false" { FALSE }
68  | '-' ?([ '0'-'9']+(['.' [ '0'-'9']]*)?) as lxm { LIT_NUM( float_of_string lxm ) }
   (*Change to add negative*)
69  | [ ""] [^ ""] * [ ""] as str { LIT_STR(str) }
70  | [ 'A'-'Z' 'a'-'z' ] + [ 'A'-'Z' 'a'-'z' '0'-'9' ] * as i { ID(i) }
71  | eof { EOF }
72  | '#', { singleLineComment lexbuf }
73  | /* { multiLineComment lexbuf }
74  | _ as c { raise (Failure("Illegal character : " ^ Char.escaped c)) }
75
76 and singleLineComment = parse
77 | '\n' { token lexbuf }
78 | _ {singleLineComment lexbuf}
79
80 and multiLineComment = parse
81 | /* { token lexbuf }
82 | _ { multiLineComment lexbuf }
83

```

./src/semcheck.ml

```
1 open Sast
2 open Ast
3
4 module StringMap = Map.Make(String)
5
6 type s_env =
7   {
8     var_types: string StringMap.t ref list;
9     var_inds : int StringMap.t ref list;
10    f_list : string StringMap.t ref list;
11  }
12
13 let check stmts =
14   let fail msg = (* raise (Failure msg) *)
15     print_string "Error : ";
16     print_string msg;
17     print_string "\n";
18     exit 0
19   in
20
21   let find_max_index map =
22     let bindings = StringMap.bindings map in
23     let rec max cur = function
24       | [] -> cur
25       | hd :: tl -> if snd hd > cur then max (snd hd) tl else max cur
26     tl
27     in
28     max 0 bindings
29   in
30
31   let type_to_str t = match t with
32     | Sast.Num      -> "num"
33     | Sast.String    -> "string"
34     | Sast.Bool      -> "bool"
35     | Sast.Point     -> "point"
36     | Sast.ListBool  -> "listbool"
37     | Sast.ListNum   -> "listnum"
38     | Sast.ListString -> "liststring"
39     | Sast.ListPoint  -> "listpoint"
40     | Sast.List      -> "list"
41   in
42
43 (* Setting Environment for Sast *)
44 let find_var var map_list =
45   let rec finder var = function
46     | m :: tl ->
47       (try (StringMap.find var !m)
48        with
49         | Not_found -> finder var tl)
50     | [] -> raise (Not_found)
51   in
52   finder var map_list
53   in
54
55 (* build default symbol tables: *)
56 let sast_env =
57   {
58     var_types    = [ref StringMap.empty];
59     var_inds     = [ref StringMap.empty];
60   }
```

```

59         f_list      = [ ref StringMap.empty ];
60     }
61
62     in
63
64     (*Check parts*)
65
66
67     let typeof elem = match elem with
68     | Sast.Literal_Num(_,t) -> t
69     | Sast.Literal_Str(_,t) -> t
70     | Sast.Point(_,_,t) -> t
71     | Sast.Literal_List(_,t) -> t
72     | Sast.Binop(_,_,_,t) -> t
73     | Sast.Id(_,t) -> t
74     | Sast.Bool(_,t) -> t
75     | Sast.Length(_,t) -> t
76     | Sast.Access(_,_,_,t) -> t
77
78   in
79
80   (* Converting Ast to Sast *)
81   (* uses the sast_env and statement list which is recieived as input *)
82   let convert_to_sast stmts_list env=
83
84     let rec expr env = function
85     | Ast.Literal_Num(v) -> Sast.Literal_Num(v, Sast.Num)
86     | Ast.Literal_Str(v) -> Sast.Literal_Str(v, Sast.String)
87     | Ast.Point(el, e2) ->
88       let sel = expr env el in
89       let se2 = expr env e2 in
90       let tel = typeof sel in
91       let te2 = typeof se2 in
92       if( tel=te2 && tel=Sast.Num)
93       then Sast.Point(sel, se2, Sast.Point)
94       else fail("Point's value should only be of type num.")
95     | Ast.Literal_List(v) ->
96       let tv = List.map (fun s -> expr env s) v in
97       (match tv with
98       | [] -> Sast.Literal_List(tv, Sast.List)
99       | hd::tl ->
100        (match (type_to_str (typeof hd)) with
101        | "num"      -> Sast.Literal_List(tv, Sast.ListNum)
102        | "string"   -> Sast.Literal_List(tv, Sast.ListString)
103        )
104        | "bool"     -> Sast.Literal_List(tv, Sast.ListBool)
105        | _          -> fail("Lists can contain only bool/
106 string/num")
107      )
108
109     | Ast.Bool(v)      -> Sast.Bool(v, Sast.Bool)
110     | Ast.Id(v)        ->
111       (* uses find_var to determine the type of id *)
112       (try
113         let tp = find_var v env.var_types in
114         (match tp with
115         | "num"           -> Sast.Id(v, Sast.Num)
116         | "string"        -> Sast.Id(v, Sast.String)
117         | "point"         -> Sast.Id(v, Sast.Point)

```

```

117           | "bool"      -> Sast.Id(v, Sast.Bool)
118           | "listnum"   -> Sast.Id(v, Sast.ListNum)
119           | "liststring" -> Sast.Id(v, Sast.ListString)
120           | "listpoint"  -> Sast.Id(v, Sast.ListPoint)
121           | "listbool"   -> Sast.Id(v, Sast.ListBool)
122           | _            -> fail(" Invalid syntax..")
123       )
124     with
125     | Not_found -> fail ("undeclared variable: " ^ v)
126   )
127 | Ast.Access(v, e) ->
128   let sv = expr env v in
129   let se = expr env e in
130   let tv = typeof sv in
131   let te = typeof se in
132   if (te=Sast.Num)
133   then (
134     if (tv = Sast.ListNum)
135     then Sast.Access(sv, se, Sast.ListNum, Sast.Num)
136   else (
137     if (tv= Sast.ListString)
138     then Sast.Access(sv, se, Sast.ListString, Sast.
String)
139     else (
140       if (tv= Sast.ListBool)
141       then Sast.Access(sv, se, Sast.ListBool, Sast.
Bool)
142       else (
143         if (tv= Sast.ListPoint)
144         then Sast.Access(sv, se, Sast.ListPoint,
Sast.Point)
145         else (if (tv= Sast.Point)
146         then Sast.Access(sv, se, Sast.Point,
Sast.Num)
147           else ( fail("'access' operations can be
performed only on List variables. Here its applied on " ^ type_to_str tv)
148           )
149         )
150       )
151     )
152   )
153   else fail ("The 'index' in list_elem.at(index) should be of
num type only." ^ (type_to_str tv))
154 | Ast.Length(v) ->
155   let sv = expr env v in
156   let tv = typeof sv in
157   if ( (tv = Sast.ListNum) || (tv = Sast.ListPoint) || (tv =
Sast.ListString) || (tv = Sast.ListBool) )
158   then Sast.Length(sv, Sast.Num)
159   else fail ("'length()' can be performed only on List
variables.")
160 | Ast.Binop(e1, op, e2) ->
161   let sel = expr env e1 in
162   let se2 = expr env e2 in
163   let e1.data = typeof sel in
164   let e2.data = typeof se2 in
165   (match op with
166   | Add | Sub | Mul | Div ->
167     (match e1.data with

```

```

169      | Num ->
170          (match e2_data with
171              | Num -> Sast.Binop(se1, op, se2, Sast.Num)
172              | String -> fail ("Cannot Add Num and string")
173              | Bool -> fail ("Cannot Add Bool")
174              | _ -> fail ("Incorrect type " ^ (type_to_str
175                  e2_data) ^ " with Num"))
176          | _ -> fail ("Operation on incompatible types")
177      )
178      | Equal | Neq ->
179          (match e1_data with
180              | Num ->
181                  (match e2_data with
182                      | Num -> Sast.Binop(se1, op, se2, Sast.Bool)
183                      | _ -> fail("Incorrect type with Num == or != "))
184                  )
185              | String ->
186                  (match e2_data with
187                      | String -> Sast.Binop(se1, op, se2, Sast.Bool)
188                      | _ -> fail("Incorrect type with String == or != "))
189                  )
190              | Bool ->
191                  (match e2_data with
192                      | Bool -> Sast.Binop(se1, op, se2, Sast.Bool)
193                      | _ -> fail("Incorrect type with Bool == or != "))
194                  )
195          | _ -> fail("Type which is not num, string or bool
196      cannot be used in equal or neq")
197          (*| Void -> fail ("Cannot perform binop on void") *)
198      )
199      | And | Or ->
200          if ( (e1_data=Num || e1_data=Bool) && (e2_data=Num ||
201              e2_data=Bool) )
202              then Sast.Binop(se1, op, se2, Sast.Bool)
203              else fail("Incorrect type with 'and' and 'or'")
204
205      | Less | Leq | Greater | Geq ->
206          (match e1_data with
207              | Num ->
208                  (match e2_data with
209                      | Num -> Sast.Binop(se1, op, se2, Sast.Bool)
210                      | _ -> fail("Incorrect type with Num < or <= or
211                          > or >= "))
212                  )
213              | String ->
214                  (match e2_data with
215                      | String -> Sast.Binop(se1, op, se2, Sast.Bool)
216                      | _ -> fail ("cannot mix string and < or <= or
217                          > or >= "))
218                  )
219          | _ -> fail ("Cannot perform less and grt ops on these
220              types")
221
222      | Mod | Square ->
223          (match e1_data with
224              | Num ->
225                  (match e2_data with

```

```

221 | Num -> Sast.Binop(sel, op, se2, Sast.Bool)
222 | _ -> fail("Incorrect type for mod. both should
223 be num")
224 | _ -> fail("Mod & square can only be used with num")
225 )
226 )
227 in
228
229 (* Convert ast stmt to sast stmt *)
230 let rec stmt env = function
231 | Ast.Noexpr -> Sast.Noexpr
232 | Ast.Expr(e) ->
233     let se = expr env e in
234     let tp = typeof se in
235     Sast.Expr(se, tp)
236 | Ast.Passign(v,e1,e) ->
237     let sv = expr env v in
238     let sel = expr env e1 in
239     let tv = typeof sv in
240     let tel = typeof sel in
241     if ( tv = Sast.Point && tel = tv )
242         then Sast.Passign(sv, sel)
243     else fail ("Invalid type assign. cannot assign " ^ (
244         type_to_str tel) ^ " to type " ^ (type_to_str tv))
245 | Ast.Assign(v,e) ->
246     let sv = expr env v in
247     let se = expr env e in
248     let tv = typeof sv in
249     let te = typeof se in
250     if ( tv = te || (te=Sast.List && (tv=Sast.ListNum || tv=Sast
251 .ListString || tv=Sast.ListBool || tv=Sast.ListPoint) ) )
252         then Sast.Assign(sv, se)
253     else fail ("Invalid type assign. cannot assign " ^ (
254         type_to_str te) ^ " to " ^ (type_to_str tv))
255 | Ast.Append(v, e) ->
256     let sv = expr env v in
257     let se = expr env e in
258     let tv = typeof sv in
259     let te = typeof se in
260     if ( (tv = Sast.ListNum && te=Sast.Num) || (tv = Sast.
261 ListPoint && te=Sast.Point) || (tv = Sast.ListString && te=Sast.String)
262 || (tv = Sast.ListBool && te=Sast.Bool) )
263         then Sast.Append(sv, se)
264     else fail ("Invalid type append. cannot append " ^ (
265         type_to_str te) ^ " to " ^ (type_to_str tv))
266 | Ast.Remove(v, e) ->
267     let sv = expr env v in
268     let se = expr env e in
269     let tv = typeof sv in
270     let te = typeof se in
271     if ( (tv = Sast.ListNum) || (tv = Sast.ListPoint) || (tv =
272 Sast.ListString) || (tv = Sast.ListBool) )
273         then
274             if ( te=Sast.Num )
275                 then Sast.Remove(sv, se)
276             else fail("The 'index' in *.pop(index) should be of num
277 type only. It cannot be of type " ^ (type_to_str te))
278             else fail ("'access' operations can be performed only on
279 List variables.")

```

```

271   | Ast.Pop(v) ->
272     let sv = expr env v in
273     let tv = typeof sv in
274       if ( (tv = Sast.ListNum) || (tv = Sast.ListPoint) || (tv =
275           Sast.ListString) || (tv = Sast.ListBool))
276         then Sast.Pop(sv)
277         else fail ("'pop()' can be performed only on List variables.
278           ")
279
280   | Ast.Fcall(v, el) ->
281     let sel = List.map (fun s -> expr env s) el in
282       (* Check if function is present *)
283       Sast.Fcall(v, sel)
284
285   | Ast.Var_Decl(dt, id) ->
286     (try
287       ignore (StringMap.find id !(List.hd env.var_types));
288       fail ("variable already declared in local scope: " ^ id)
289       with | Not_found -> (List.hd env.var_types) := StringMap.
290             add id dt !(List.hd env.var_types);
291             (List.hd env.var_inds) := StringMap.add id (
292               find_max_index !(List.hd env.var_inds)+1 !(List.hd env.var_inds);
293               | Failure(f) -> raise (Failure (f)))
294             );
295     let tp = find_var id env.var_types in
296       if (tp="num") then
297         Sast.Var_Decl(dt, id, Sast.Num)
298       else
299         if (tp="string") then
300           Sast.Var_Decl(dt, id, Sast.String)
301         else
302           if (tp="point") then
303             Sast.Var_Decl(dt, id, Sast.Point)
304           else
305             Sast.Var_Decl(dt, id, Sast.Bool)
306
307   | Ast.List_Decl(dt, id) ->
308     (try
309       ignore (StringMap.find id !(List.hd env.var_types));
310       fail ("variable already declared in local scope: " ^ id)
311       with | Not_found -> (List.hd env.var_types) := StringMap.
312             add id ("list" ^ dt) !(List.hd env.var_types);
313             (List.hd env.var_inds) := StringMap.add id (
314               find_max_index !(List.hd env.var_inds)+1 !(List.hd env.var_inds);
315               | Failure(f) -> raise (Failure (f)))
316             );
317     let tp = find_var id env.var_types in
318       if (tp="listnum") then
319         Sast.List_Decl(dt, id, Sast.ListNum)
320       else
321         if (tp="liststring") then
322           Sast.List_Decl(dt, id, Sast.ListString)
323         else
324           if (tp="listpoint") then
325             Sast.List_Decl(dt, id, Sast.ListPoint)
326           else
327             Sast.List_Decl(dt, id, Sast.ListBool)
328
329   | Ast.Print(e) ->
330     let sel = expr env e in
331     let tel = typeof sel in
332       Sast.Print(sel)
333
334   | Ast.PrintXY(e1,e2) ->
335     let sel = expr env e1 in

```

```

325     let se2 = expr env e2 in
326     let tel = typeof se1 in
327     let te2 = typeof se2 in
328     if (te2 = Sast.Point)
329     then Sast.PrintXY(se1, se2)
330     else fail("The second argument of printXY should be of type
331           point")
332   | Ast.LineVar(e1, e2) ->
333     let sel = expr env e1 in
334     let se2 = expr env e2 in
335     let tel = typeof sel in
336     let te2 = typeof se2 in
337     if( tel = Sast.Point && te2 = Sast.Point )
338     then Sast.LineVar(sel, se2)
339     else fail ("LineVar has to be called with 2 points")
340   | Ast.LineVarColor(e1, e2, c) ->
341     let sel = expr env e1 in
342     let se2 = expr env e2 in
343     let sc = expr env c in
344     let tel = typeof sel in
345     let te2 = typeof se2 in
346     let tc = typeof sc in
347     if( tel = Sast.Point && te2 = Sast.Point && tc=Sast.String )
348     then Sast.LineVarColor(sel, se2, sc)
349     else fail ("Line has to be called with 2 points and string
350           for color (optional)")
351   | Ast.LineRaw(e1, e2, e3, e4) ->
352     let sel = expr env e1 in
353     let se2 = expr env e2 in
354     let se3 = expr env e3 in
355     let se4 = expr env e4 in
356     let tel = typeof sel in
357     let te2 = typeof se2 in
358     let te3 = typeof se3 in
359     let te4 = typeof se4 in
360     if( tel = Sast.Num && te2 = Sast.Num && te3 = Sast.Num &&
361         te4 = Sast.Num )
362       then Sast.LineRaw(sel, se2, se3, se4)
363       else fail ("LineRaw has to be called with 4 nums")
364   | Ast.LinePX(e1, e2, e3) ->
365     let sel = expr env e1 in
366     let se2 = expr env e2 in
367     let se3 = expr env e3 in
368     let tel = typeof sel in
369     let te2 = typeof se2 in
370     let te3 = typeof se3 in
371     if( tel = Sast.Num && te2 = Sast.Num && te3 = Sast.Point )
372     then Sast.LinePX(sel, se2, se3)
373     else fail ("Line has to be called with 2 points")
374   | Ast.For(s1, e1, s2, body) ->
375     let ss1 = stmt env s1 in
376     let sel = expr env e1 in
377     let ss2 = stmt env s2 in
378     Sast.For(ss1, sel, ss2, List.map (fun s -> stmt env s) body)
379   | Ast.While(e, body) ->
380     let se = expr env e in
381     let te = typeof se in
382     if ( te = Sast.Num || te = Sast.Bool)
383     then Sast.While(se, List.map (fun s -> stmt env s) body)
384     else fail("The condition in while should give eiether num or

```

```

  bool. Not_of_type " ^type_to_str te)
382    | Ast.Ifelse(e, s1, s2) ->
383      let se = expr env e in
384        Sast.Ifelse(se, List.map (fun s -> stmt env s) s1, List.map
385          (fun s -> stmt env s) s2)
386    | Ast.Return(e) -> Sast.Return(expr env e)
387    | Ast.Fdecl(f) ->
388      let fnEnv = {
389        var_types = [ref StringMap.empty];
390        var_inds = [ref StringMap.empty];
391        f_list = [ref StringMap.empty];
392      } in
393      let f_name =
394        (try
395          ignore (StringMap.find f.fname !(List.hd env.f_list))
396        ) ;
397        fail ("Function already declared in local scope:
398          " ^ f.fname)
399      with | Not_found -> (List.hd env.f_list) :=
400        StringMap.add f.fname "function" !(List.hd env.f_list);
401        | Failure(f) -> raise (Failure (f))
402      );
403
404      in
405      let fargs = List.map (fun s -> stmt fnEnv s) f.args in
406      let fstms = List.map (fun s -> stmt fnEnv s) f.body in
407      Sast.Fdecl({
408        fname = f.fname;
409        args = fargs;
410        body = fstms;
411      })
412      in
413      List.map (fun s -> stmt env s) stmts_list
414    in
415    Sast.string_of_tprogram (convert_to_sast stmts sast_env)
416
417

```

./src/tast.ml

```
1 (* operations *)
2 type ops =
3     | Add | Sub | Mul | Div | Mod
4     | Equal | Neq | Less | Leq | Greater | Geq
5     | And | Or
6     | Square
7
8 type bool =
9     | True | False
10
11
12 (* expressions *)
13 type texpr =
14     Literal_Num of float
15     | Literal_Str of string
16     | Point of texpr * texpr
17     | Literal_List of texpr list          (* Eg [ texpr , texpr , ... ] *)
18     | Binop of texpr * ops * texpr      (* Binary Ops *)
19     | Id of string                      (* identifiers *)
20     | Bool of bool                      (* True *)
21     | Length of texpr                  (* a.length() *)
22     | Access of texpr * texpr          (* a.at(3) , a[3] *)
23
24
25
26 type tstmt = (* Statements *)
27     Texpr of texpr
28     | Var_Decl of string * string      (* (type , id) *)
29     | List_Decl of string * string
30     | Passign of texpr * texpr * tstmt (* (type , p1 , p2) *)
31     | Assign of texpr * texpr
32     | Append of texpr * texpr
33     | Pop of texpr
34     | Remove of texpr * texpr
35     | Fcall of string * texpr list    (* a = 2 *)
36     | PrintXY of texpr * texpr        (* a.append(7) *)
37     | Print of texpr
38     | LineVar of texpr * texpr
39     | LineRaw of texpr * texpr * texpr (* a.pop() *)
40     | LinePX of texpr * texpr * texpr (* a.remove(3) *)
41     | For of tstmt * texpr * tstmt * tstmt list (* a.(5) *)
42     | While of texpr * tstmt list
43     | Ifelse of texpr * tstmt list * tstmt list (* print 5 *)
44     | Return of texpr
45     | Noexpr
46     | Fdecl of fdecl and
47         fdecl = {
48             fname : string ;
49             args  : tstmt list ;
50             body   : tstmt list ;
51         }
52
53
54
55 type tprogram = {
56     funcs : tstmt list ;
57     main   : tstmt list ;
58 }
```

```

59
60
61
62 (* Pretty Print
63
64 let rec string_of_texpr = function
65   | Literal_Num(l) -> string_of_float l ^ "0"
66   | Literal_Str(l) -> l
67   | Point(e1, e2) -> "(" ^ string_of_texpr e1 ^ "," ^ string_of_texpr e2 ^ ")"
68   | Literal_List(l) -> "[" ^ (String.concat "," (List.map string_of_texpr l)) ^ "]"
69   | Id(s) -> s
70   | Binop(e1, o, e2) ->
71     string_of_texpr e1 ^ " " ^
72     (match o with
73      | Add -> "+"
74      | Sub -> "-"
75      | Mul -> "*"
76      | Div -> "/"
77      | Equal -> "==" | Neq -> "!="
78      | Mod -> "%"
79      | And -> "&&" | Or -> "||"
80      | Square -> "**"
81      | Less -> "<" | Leq -> "<="
82      | Greater -> ">" | Geq -> ">="
83      ) ^ " " ^ string_of_texpr e2
84
85
86
87
88 let rec string_of_tstmt = function
89   | texpr(texpr) -> string_of_texpr texpr ^ ""
90   | Var_Decl(tp, id) -> tp ^ " " ^ id ^ "\n"
91   | List_Decl(tp, id) -> "list " ^ tp ^ " " ^ id ^ "\n"
92   | Passign(v, e1, e) -> " " ^ string_of_texpr v ^ " = " ^ (string_of_texpr e1) ^ "\n"
93   | Assign(v, e) -> " " ^ string_of_texpr v ^ " = " ^ (string_of_texpr e)
94   | Append(v, e) -> string_of_texpr v ^ ".append(" ^ (string_of_texpr e) ^ ")\n"
95   | Pop(v) -> string_of_texpr v ^ ".pop()\n"
96   | Remove(v, e) -> string_of_texpr v ^ ".remove(" ^ (string_of_texpr e) ^ ")\n"
97   | Fcall(v, el) -> v ^ "(" ^ (String.concat "," (List.map string_of_texpr el)) ^ ")\n"
98   | PrintXY(e1, e2) -> "printXY( " ^ string_of_texpr e1 ^ " , " ^ string_of_texpr e2 ^ ")\n"
99   | Print(e) -> "print " ^ string_of_texpr e ^ "\n"
100  | LineVar(e1, e2) -> "line (" ^ string_of_texpr e1 ^ " , " ^ string_of_texpr e2 ^ ")\n"
101  | LineRaw(e1, e2, e3, e4) -> "line ( (" ^ string_of_texpr e1 ^ " , " ^ string_of_texpr e2 ^ " ) " ^ " , " ^ "(" ^ string_of_texpr e3 ^ " , " ^ string_of_texpr e4 ^ " ) )\n"
102  | LinePX(e1, e2, e3) -> "line ( ( " ^ string_of_texpr e1 ^ " , " ^ string_of_texpr e2 ^ " ) , " ^ string_of_texpr e3 ^ " )\n"
103  | For(s1, e1, s2, body) -> "for " ^ string_of_tstmt s1 ^ " ; " ^ string_of_texpr e1 ^ " ; " ^ string_of_tstmt s2 ^ " : " ^ string_of_texpr e1 ^ " " ^ (String.concat "\n\t" (List.map string_of_tstmt body))
104
105

```

```

106           ^ "\nend\n"
107   | While(e, body) -> "while " ^ string_of_texpr e ^ " :\n" ^ (String.concat
108     "\n\t" (List.map string_of_tstmt body)) ^ "\nend\n"
109   | Ifelse(e, succ_tstmt, else_tstmt) -> "if " ^ string_of_texpr e ^ " :\n"
110     ^ (String.concat "\n\t" (List.map string_of_tstmt succ_tstmt)) ^ "\nelse:\n"
111     ^ (String.concat "\n\t" (List.map string_of_tstmt else_tstmt)) ^ "end\n"
112
113   | Return(expr) -> "return " ^ string_of_texpr expr ^ "\n"
114   | Notexpr -> ""
115   | Fdecl(f) -> string_of_fdecl f and
116     string_of_fdecl fdecl =
117       "fn " ^ fdecl.fname ^ "(" ^ "
118         ( String.concat ", " (List.map (fun s -> string_of_tstmt s) fdecl.
119           args) ) ^
120           "):\n" ^
121             ( String.concat "" (List.map string_of_tstmt fdecl.body) ) ^
122               "\nend\n"
123
124
125 let string_of_program prog =
126   String.concat "\n" (List.map string_of_tstmt prog.funcs)
127   ^ "\n" ^ "\n"
128   String.concat "\n" (List.map string_of_tstmt prog.main) *)

```

./src/test.py

```
1 import sys
2 import os
3
4 #If user gives a specific set of files from command line
5 if len(sys.argv)>1:
6     testFiles = sys.argv[1:]
7 else:
8     #Get all the files in the tests dir
9     testFiles = os.listdir('./tests/')
10    testFiles = [ x for x in testFiles if ( x[-3]== 'plt' and ( x[:4] in [ 'pass',
11        'fail'] ) )]
12
13 nof = len(testFiles)
14
15 #passing and failing
16 passed = []
17 failed = []
18 i=0
19 print 'Starting the tests..'
20 #Proceed if make succeeds
21 for file in testFiles:
22     #if i%(nof/20) == 0:
23     #    sys.stdout.write('.')
24     #For each test file perform the test. And print pass or failure
25     runStr = './plt tests/' + file + '> temp.out'
26     #print 'Running for file : '+file+'\n'
27     os.system(runStr)
28     f = open('temp.out')
29     s = f.readlines()
30     f.close()
31     if (len(s)>0 and file[:4]== 'pass') or (len(s)==0 and file[:4]== 'fail'):
32         failed.append( '*** FAILED for file '+file+'\n' + ''.join(s) )
33     else:
34         passed.append( 'PASSED for -- '+file)
35     i+=1
36 #Printing the results
37 print '\n----- PASSED TESTS -----',
38 for i in passed:
39     print i
40 print '----- FAILED TESTS -----',
41 for i in failed:
42     print i
43 print '----- TESTS STATS -----',
44 print 'Passed : ' + str(len(passed))
45 print 'Failed : ' + str(len(failed))
```

./src/library/plots.plt

```
1  /*
2   * Method :
3   *     rect( point xy , num h , num w)
4   *
5   * Return :
6   *     void
7   *
8   * Description :
9   *     Prints a rectangle , from point x,y with height h and width w The
10  *     function rectangle supported as libray .
11  *
12  * Future Todo :
13  *     Send hash , to set the properties of the graph like axis , grid , title
14  *     etc
15  */
16  fn rect( point a , num h , num w):
17      line(a,(a[0]+w,a[1]))
18      line(a,(a[0] ,a[1]+h))
19      line((a[0]+w,a[1]) ,(a[0]+w,a[1]+h))
20      line((a[0] ,a[1]+h),(a[0]+w,a[1]+h))
21  end
22
23  fn rectFill( point a , num h , num w, string color):
24      num i
25      point x
26      point y
27      string s
28
29      /* Make a rectanle by drawing multiple lines */
30      for i=0;i<w; i=i+1:
31          x = (a[0]+i , a[1])
32          y = (a[0]+i , a[1]+h)
33          line( x , y , color )
34  end
35 */
36  * Method :
37  *     drawAxes(num xorigin , num yorigin , num maxWidth , num maxHeight)
38  *
39  * Return :
40  *     void
41  *
42  * Description :
43  *     Draws cross axis from the passed origin point . Takes max width and
44  *     max Height to plot lines
45  *
46  * Future Todo :
47  *     Send hash , to set the properties of the graph like axis , grid , title
48  *     etc
49  */
50  fn drawAxes(num ox , num oy , num maxLength , num maxHeight):
51
52      #Draw axes
53      rect((ox,0) , maxHeight , 3)
54      rect((0,maxHeight-oy) ,3,maxLength)
55  end
```

```

56     Method :
57         barGraph(<num list var>)
58
59     Return :
60         void
61
62     Description :
63         Prints a fitting barGraph for the given data. The function bargraph
64         supported as library. Users can write their own too. It takes the list of
65         num and automatically fits it in the graph based on the data.
66         The user just needs to call the function with the data
67
68     */
69 fn barGraph( list num a):
70     num maxLength
71     num maxHeight
72     num maxDataHt
73     num maxDataLn
74     num gap
75     num scaleFactor
76     num padHz
77     num padVt
78     num barWidth
79     num i
80
81     #Setting the dimensions of the graph
82     maxLength = 640
83     maxHeight = 480
84
85     #Max ht
86     maxDataHt = a[0]
87     for i=0;i<a.length();i=i+1:
88         if a[i] > maxDataHt:
89             maxDataHt = a[i]
90         end
91     end
92
93     #max length
94     maxDataLn = a.length()
95
96     #padding
97     padHz = 10
98     padVt = 10
99
100    #bar graph settings 10% of the graph
101    gap        = 0.1 * (maxLength - padHz) / maxDataLn
102    barWidth   = 0.9 * (maxLength - padHz) / maxDataLn
103    scaleFactor = (maxHeight - padVt) / maxDataHt
104
105    #Draw the bars, scaled and with the gap
106    num x
107    x = padHz
108    for i=0;i<a.length();i=i+1:
109        #Drawing the bar
110        rect( (x,maxHeight- a[i]*scaleFactor) , a[i]*scaleFactor , barWidth
111    )

```

```

112     #line( (x,maxHeight- a[ i ]* scaleFactor) , (x,maxHeight) )
113     #print x
114     x = x + barWidth + gap
115 end
116
117 end
118 /*
119 Method :
120     lineGraph(num xorigin , num yorigin , num maxWidth, num maxHeight)
121
122 Return :
123     void
124
125 Description :
126     Draws a line graph based on the points give. To draw axes call it
127 seperately
128
129 Future Todo :
130     Send hash, to set the properties of the graph like axis, grid, title
131     etc
132 */
133 fn lineGraph( list num a):
134     num maxLength
135     num maxHeight
136     num maxDataHt
137     num maxDataLn
138     num gap
139     num scaleFactor
140     num padHz
141     num padVt
142     point p1
143     point p2
144     num i
145
146
147 #Setting the dimensions of the graph
148 maxLength = 640
149 maxHeight = 480
150
151 #Max ht
152 maxDataHt = a[ 0 ]
153 for i=0;i<a.length() ; i=i+1:
154     if a[ i ] > maxDataHt:
155         maxDataHt = a[ i ]
156     end
157 end
158
159 #max length
160 maxDataLn = a.length()
161
162 #padding
163 padHz = 10
164 padVt = 10
165
166 #Line graph settings 10% of the graph
167 gap = (maxLength - padHz) / maxDataLn
168 scaleFactor = (maxHeight - padVt) / maxDataHt
169

```

```

170 #Draw the bars, scaled and with the gap
171 num x
172 num j
173 num y1
174 num y2
175 x = padHz
176
177 if a.length() > 0:
178
179     for i=1;i<a.length();i=i+1:
180         line( (x , maxHeight - a[ i - 1 ] * scaleFactor) , ( x + gap ,
181             maxHeight- a[ i ]*scaleFactor) )
182
183         #dots on the data points
184         rect ( (x - 2.5, maxHeight - a[ i - 1 ] * scaleFactor - 2.5) , 5
185             , 5)
186         rect ( (x - 2.5, maxHeight - a[ i - 1 ] * scaleFactor - 2.5) , 5
187             , 5)
188         x = x + gap
189     end
190 end
191

```

./src/programs/bargraph.plt

```
1 fn rect(point a, num h, num w):
2     num i
3     num x
4     num y
5     point b
6     /* Make a rectangle by drawing multiple lines */
7     for i=0;i<w; i=i+1:
8         x = a[0]
9         y = a[1]
10        line( (x+i , y) , (x+i , y+h))
11    end
12 end
13 /*
14 Method :
15     barGraph(<num list var>)
16
17 Return :
18     void
19     Prints a fitting barGraph for the given data
20
21 Description :
22     The function bargraph supported as library. Users can write their own
23     too. It takes the list of num and automatically fits it in the graph
24     based on the data.
25     The user just needs to call the function with the data
26
27 Future Todo :
28     Send hash, to set the properties of the graph like axis, grid, title
29     etc
30 */
31 fn barGraph(list num a):
32     num maxLength
33     num maxHeight
34     num maxDataHt
35     num maxDataLn
36     num gap
37     num scaleFactor
38     num padHz
39     num padVt
40     num barWidth
41     num i
42
43 #Setting the dimensions of the graph
44 maxLength = 640
45 maxHeight = 480
46
47 #Max ht
48 maxDataHt = a[0]
49 for i=0;i<a.length(); i=i+1:
50     if a[i] > maxDataHt:
51         maxDataHt = a[i]
52     end
53 end
54
55 #max length
56 maxDataLn = a.length()
```

```

57  #padding
58  padHz = 10
59  padVt = 10
60
61  #bar graph settings 10% of the graph
62  gap      = 0.1 * (maxLength - padHz) / maxDataLn
63  barWidth = 0.9 * (maxLength - padHz) / maxDataLn
64  scaleFactor = (maxHeight - padVt) / maxDataHt
65
66  #Draw the bars, scaled and with the gap
67  num x
68  x = padHz
69  for i=0;i<a.length();i=i+1:
70    #Drawing the bar
71    rect( (x,maxHeight- a[i]*scaleFactor) , a[i]*scaleFactor , barWidth
72    )
73    #line( (x,maxHeight- a[i]*scaleFactor) , (x,maxHeight) )
74    #print x
75    x = x + barWidth + gap
76  end
77
78 end
79
80 list num a
81 a = [11,4,25.6,10,12,50,10,30,5]
82 barGraph(a)
83
84

```

./src/programs/box.plt

```
1 fn rect(point a, num h, num w):
2     line(a,(a[0]+w,a[1]))
3     line(a,(a[0],a[1]+h))
4     line((a[0]+w,a[1]),(a[0]+w,a[1]+h))
5     line((a[0],a[1]+h),(a[0]+w,a[1]+h))
6 end
7
8 rect ((100,100), 50, 20)
9
10
```

./src/programs/colorline.plt

```
1 point p1
2 point p2
3 p1 = (100,100)
4 p2 = (100,200)
5 line( p1 , p2 , "blue")
6
7
8
```

./src/programs/comicBookTemplate.plt

```
1  /*
2   Method :
3     rect( point xy , num h , num w)
4
5   Return :
6     void
7
8   Description :
9     Prints a rectangle , from point x,y with height h and width w The
10    function rectangle supported as libray .
11
12  Future Todo :
13    Send hash , to set the properties of the graph like axis , grid , title
14    etc
15  */
16 fn rect( point a , num h , num w):
17   line(a,(a[0]+w,a[1]))
18   line(a,(a[0] ,a[1]+h))
19   line((a[0]+w,a[1]) ,(a[0]+w,a[1]+h))
20   line((a[0] ,a[1]+h),(a[0]+w,a[1]+h))
21 end
22
23 fn rectFill( point a , num h , num w, string color):
24   num i
25   point x
26   point y
27   string s
28
29   /* Make a rectanle by drawing multiple lines */
30   for i=0;i<w; i=i+1:
31     x = (a[0]+i , a[1])
32     y = (a[0]+i , a[1]+h)
33     line( x , y , color )
34   end
35 */
36 Method :
37   drawAxes(num xorigin , num yorigin , num maxWidth , num maxHeight)
38
39 Return :
40   void
41
42 Description :
43   Draws cross axis from the passed origin point . Takes max width and
44   max Height to plot lines
45
46 Future Todo :
47   Send hash , to set the properties of the graph like axis , grid , title
48   etc
49 */
50 fn drawAxes(num ox , num oy , num maxLength , num maxHeight):
51
52   #Draw axes
53   rect((ox,0) , maxHeight , 3)
54   rect((0,maxHeight-oy) ,3,maxLength)
55 end
```

```

56     Method :
57         barGraph(<num list var>)
58
59     Return :
60         void
61
62     Description :
63         Prints a fitting barGraph for the given data. The function bargraph
64         supported as library. Users can write their own too. It takes the list of
65         num and automatically fits it in the graph based on the data.
66         The user just needs to call the function with the data
67
68     */
69 fn barGraph( list num a):
70     num maxLength
71     num maxHeight
72     num maxDataHt
73     num maxDataLn
74     num gap
75     num scaleFactor
76     num padHz
77     num padVt
78     num barWidth
79     num i
80
81     #Setting the dimensions of the graph
82     maxLength = 640
83     maxHeight = 480
84
85     #Max ht
86     maxDataHt = a[0]
87     for i=0;i<a.length();i=i+1:
88         if a[i] > maxDataHt:
89             maxDataHt = a[i]
90         end
91     end
92
93     #max length
94     maxDataLn = a.length()
95
96     #padding
97     padHz = 10
98     padVt = 10
99
100    #bar graph settings 10% of the graph
101    gap        = 0.1 * (maxLength - padHz) / maxDataLn
102    barWidth   = 0.9 * (maxLength - padHz) / maxDataLn
103    scaleFactor = (maxHeight - padVt) / maxDataHt
104
105    #Draw the bars, scaled and with the gap
106    num x
107    x = padHz
108    for i=0;i<a.length();i=i+1:
109        #Drawing the bar
110        rect( (x,maxHeight- a[i]*scaleFactor) , a[i]*scaleFactor , barWidth
111    )

```

```

112     #line( (x,maxHeight- a[ i ]* scaleFactor) , (x,maxHeight) )
113     #print x
114     x = x + barWidth + gap
115 end
116
117 end
118 /*
119 Method :
120     lineGraph(num xorigin , num yorigin , num maxWidth, num maxHeight)
121
122 Return :
123     void
124
125 Description :
126     Draws a line graph based on the points give. To draw axes call it
127 seperately
128
129 Future Todo :
130     Send hash, to set the properties of the graph like axis, grid, title
131     etc
132 */
133 fn lineGraph( list num a):
134     num maxLength
135     num maxHeight
136     num maxDataHt
137     num maxDataLn
138     num gap
139     num scaleFactor
140     num padHz
141     num padVt
142     point p1
143     point p2
144     num i
145
146
147 #Setting the dimensions of the graph
148 maxLength = 640
149 maxHeight = 480
150
151 #Max ht
152 maxDataHt = a[ 0 ]
153 for i=0;i<a.length() ; i=i+1:
154     if a[ i ] > maxDataHt:
155         maxDataHt = a[ i ]
156     end
157 end
158
159 #max length
160 maxDataLn = a.length()
161
162 #padding
163 padHz = 10
164 padVt = 10
165
166 #Line graph settings 10% of the graph
167 gap = (maxLength - padHz) / maxDataLn
168 scaleFactor = (maxHeight - padVt) / maxDataHt
169

```

```

170     #Draw the bars, scaled and with the gap
171     num x
172     num j
173     num y1
174     num y2
175     x = padHz
176
177     if a.length() > 0:
178
179         for i=1;i<a.length();i=i+1:
180             line( (x , maxHeight - a[ i - 1 ] * scaleFactor) , ( x + gap ,
181             maxHeight- a[ i ]*scaleFactor ) )
182
183             #dots on the data points
184             rect ( (x - 2.5, maxHeight - a[ i - 1 ] * scaleFactor - 2.5) , 5
185             , 5)
186             rect ( (x - 2.5, maxHeight - a[ i - 1 ] * scaleFactor - 2.5) , 5
187             , 5)
188             x = x + gap
189
190         end
191     end
192
193 fn comicBookTemplate( list num rowDetails , list num colDetails):
194     num maxHeight
195     num maxLength
196     num x
197     num y
198     num i
199     num j
200     num k
201     num h
202     maxHeight= 600
203     maxLength= 480
204     x=0
205     y=0
206     j = 0
207     for i=0;i<rowDetails.length();i=i+2:
208         x = 0
209
210         #Border for the row
211         h = maxHeight * rowDetails[ i ]
212         rectFill((0,y) , h , maxLength,"brown")
213         rectFill( ( 5 , y+5), h - 10, maxLength - 10, "white" )
214
215         #Putting the columns
216         for k=0; k +1 < rowDetails[i + 1];k=k+1:
217             x = x + colDetails[j+k]*maxLength
218             rectFill(( x , y) , h , 5 , "black" )
219             #printXY("*",(" x - 20 , y+h/2))
220             end
221             j = j + k + 1
222             y = y+h - 5
223         end
224
225     list num rd
226     list num cd

```

```
227 #Give row height % and no of columns in that row
228 rd = [0.3, 2, 0.4, 1, 0.3, 2]
229
230 #Give column % wrt the whole width
231 cd = [0.3, 0.7, 1, 0.7, 0.3]
232
233 comicBookTemplate(rd, cd)
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
```

[./src/programs/empireStates.plt](#)

```
1 include plots
2 #Rectable with black border
3 fn rectWB(point p, num w, num h, string c):
4     rectFill(p,w,h,c)
5     rect(p,w,h)
6 end
7
8
9
10 rectWB((0,550),50, 300,"grey")
11
12 rect((65,135),465, 170)
13 rectWB((85,120),480, 130,"grey")
14 rectWB((90,120),480, 120,"white")
15
16 rectWB((100,100),500, 100,"grey")
17
18 #Centre row
19 rectWB((125,50),550, 50,"white")
20 rectWB((145,0),600, 10, "white")
21
22 #Pillars
23 rectWB((50,150),450, 70, "white")
24 rectWB((50,200),400, 70, "white")
25 rectWB((180,150),450, 70, "white")
26 rectWB((180,200),400, 70, "white")
27
28
29
30
31
32
33
34
35
36
37
38
39
```

./src/programs/forLoop.plt

```
1 num a
2 for a = 0; a < 5; a = a + 1 :
3 print a
4 end
5
6
```

./src/programs/hello.plt

```
1 print "Hello brothers and sisters"
2
3
```

./src/programs/ifelse.plt

```
1 num a
2 a = 5
3 if a > 0 :
4     print "A is positive"
5 end
6 else:
7     print "A is negative"
8 end
9
```

./src/programs/line.plt

```
1 line((50, 500), (500, 50))  
2  
3
```

./src/programs/lineGraph.plt

```
1 #include plots
2 fn rect(point a, num h, num w):
3     num i
4     num x
5     num y
6     point b
7     /* Make a rectangle by drawing multiple lines */
8     for i=0;i<w; i=i+1:
9         x = a[0]
10        y = a[1]
11        line( (x+i , y) , (x+i , y+h))
12    end
13 end
14
15 fn drawAxes(num ox, num oy, num maxLength, num maxHeight):
16
17     #Draw axes
18     rect((ox,0) , maxHeight , 3)
19     rect((0,maxHeight-oy) ,3 ,maxLength)
20 end
21
22 fn lineGraph( list num a):
23     num maxLength
24     num maxHeight
25     num maxDataHt
26     num maxDataLn
27     num gap
28     num scaleFactor
29     num padHz
30     num padVt
31     point p1
32     point p2
33     num i
34
35
36     #Setting the dimensions of the graph
37     maxLength = 640
38     maxHeight = 480
39
40     #Max ht
41     maxDataHt = a[0]
42     for i=0;i<a.length(); i=i+1:
43         if a[i] > maxDataHt:
44             maxDataHt = a[i]
45     end
46 end
47
48     #max length
49     maxDataLn = a.length()
50
51     #padding
52     padHz = 10
53     padVt = 10
54
55     #Line graph settings 10% of the graph
56     gap = (maxLength - padHz) / maxDataLn
57     scaleFactor = (maxHeight - padVt) / maxDataHt
58
59     #Draw the bars, scaled and with the gap
```

```

60      num x
61      num j
62      num y1
63      num y2
64      x = padHz
65
66      if a.length() > 0:
67
68          for i=1;i<a.length();i=i+1:
69              line( (x , maxHeight - a[ i - 1 ] * scaleFactor) , ( x + gap ,
70                  maxHeight- a[ i ]*scaleFactor ) )
71
72              #dots on the data points
73              rect ( (x - 2.5, maxHeight - a[ i - 1 ] * scaleFactor - 2.5) , 5
74                  , 5)
75              rect ( (x - 2.5, maxHeight - a[ i - 1 ] * scaleFactor - 2.5) , 5
76                  , 5)
77              x = x + gap
78
79      end
80  end
81
82  list num a
83  a = [50,53, 55, 53, 50, 45, 25, 10, 5, 3, 0, 2, 4, 10, 25, 45,60, 75, 85,
84  90,95, 98, 99, 100]
85  lineGraph(a)
86  drawAxes( 10,240,640,480)
87
88

```

./src/programs/list_access.plt

```
1 list num a
2 a = [1, 2, 3]
3 print a.at(0)
4 print a[0]
5
```

./src/programs/listdecl.plt

```
1 list num a
2 list string b
3 list bool d
4 a = []
5 a = [1]
6 a = [1, 2, 3]
7 b = []
8 b = ["c"]
9 b = ["a", "b", "c"]
10 d = []
11 d = [true]
12
13 d = [true, false, true]
14 a.append(6)
15
16
```

./src/programs/list_operations.plt

```
1 list num a
2 list bool b
3 a=[1,2,3,4,5,6]
4 b=[]
5 a.append(8)
6 a.pop()
7 a.remove(4)
8 a.at(3)
9 a[4]
10 a.length()
11 num x
12 x = 1 + 4
13
14
```

./src/programs/plot1.plt

```
1 num i
2   i = 0
3   while i <= 500:
4     line((50,i+50), (550, 500-i+50))
5     line((50+i,50),(550-i,550))
6     i = i + 10
7 end
8
9
```

./src/programs/print.plt

```
1 num a
2 a = 5 + 10
3 print a
4 print "a"
5
6
```

./src/programs/printXYsample.plt

```
1 num i
2   for i=0;i<10;i=i+1:
3     printXY("Ranjith", (100,100+20*i))
4   end
5
6
```

./src/programs/rectangle.plt

```
1 #In future it'll be from library
2 fn rect(point a, num h, num w):
3     num i
4     num x
5     num y
6     point b
7     /* Make a rectangle by drawing multiple lines */
8     for i=0;i<w; i=i+1:
9         x = a[0]
10        y = a[1]
11        line( (x+i , y) , (x+i , y+h))
12    end
13 end
14
15 point a
16 a = (100,100)
17
18 #Rectangles with increasing width, nice pattern too
19 rect(a, 200, 10)
20 rect((100+15,100), 200, 50)
21 rect((100+70,100), 200, 100)
22 rect((100+175,100), 200, 200)
23 rect((100+380,100), 200, 500)
24 rect((100+885,100), 200, 500)
25
26
```

./src/programs/rectFill.plt

```
1 fn rect( point a, num h, num w, string color):
2     line(a,(a[0]+w,a[1]))
3     line(a,(a[0],a[1]+h))
4     line((a[0]+w,a[1]),(a[0]+w,a[1]+h))
5     line((a[0],a[1]+h),(a[0]+w,a[1]+h))
6 end
7 fn rectFill( point a, num h, num w, string color):
8     num i
9     point x
10    point y
11    string s
12
13    /* Make a rectangle by drawing multiple lines */
14    for i=0;i<w; i=i+1:
15        x = (a[0]+i, a[1])
16        y = (a[0]+i, a[1]+h)
17        line( x , y , color )
18    end
19 end
20
21 rectFill ( (100,100) , 50 , 20,"blue")
22
23
24
25
26
```

./src/programs/simpleplot.plt

```
1 line((50, 500), (50, 50))
2 line((50, 500), (600, 500))
3 line((50, 500), (500, 50))
4
5
```

./src/programs/sine.plt

```
1 #In future it'll be from library
2 fn rect(point a, num h, num w):
3     num i
4     num x
5     num y
6     point b
7     /* Make a rectangle by drawing multiple lines */
8     for i=0;i<w; i=i+1:
9         x = a[0]
10        y = a[1]
11        line( (x+i , y) , (x+i , y+h))
12    end
13 end
14
15 num i
16 num ht
17 num flag
18
19 #Initial
20 flag = 1
21
22 # Trying to print a sine wave with rectangle function
23 for i=0;i<1000; i=i+1:
24     rect((100+5*i,100+ht) ,200 ,4)
25
26 ht = ht + flag*2
27 #Sine switch
28 if ht==100 or ht==0:
29     flag = flag* -1
30 end
31 end
32
33
```

./src/programs/sum.plt

```
1 num a
2 a = -5 + 10
3 print a
4
5
```

./src/programs/vdecl.plt

```
1 string a
2 num b
3 bool c
4
5
```

./src/programs/while.plt

```
1 num a
2 a = 0
3 while a < 10:
4 print a
5 a = a + 1
6 end
7
8
```

./src/tests/failboolassignnumfloat plt

```
1 bool a
2 a = 5.5
3
4
```

./src/tests/failboolassignnumint.plt

```
1 bool a
2 a = 5
3
4
```

./src/tests/failboolassignstring.plt

```
1 bool a
2 a = "hi"
3
4
```

./src/tests/fail_{bool}_{sum}_n_{um}.plt

```
1 bool a
2 a = true + 5
3
4
```

./src/tests/fail_{bool}_{sum}_{point}.plt

```
1 bool a
2 point b
3 a = true + b
4
5
```

./src/tests/failboolsumstring.plt

```
1 bool a
2 a = true + "hi"
3
4
```

./src/tests/failforcolon.plt

```
1 num a
2   for a = 0; a < 5; a = a + 1
3     print a
4   end
5
6
```

./src/tests/failforempty.plt

```
1 num a
2 for :
3 end
4
5
```

./src/tests/failforend.plt

```
1 num a
2   for a = 0; a < 5; a = a + 1 :
3     print a
4
5
```

./src/tests/fail for missing1.plt

```
1 num a
2   for a < 5; a = a + 1 :
3     print a
4   end
5
6
```

./src/tests/fail for missing23.plt

```
1 num a
2   for a = 0; :
3     print a
4   end
5
6
```

./src/tests/failformissing3.plt

```
1 num a
2   for a = 0; a < 5; :
3     print a
4   end
5
6
```

./src/tests/failformissingtwo.plt

```
1 num a
2   for a = 0;  a = a + 1 :
3     print a
4   end
5
6
```

./src/tests/fail_includelib.plt

```
1 list num a
2 a = [100,80,95,66,70]
3 barGraph(a)
4
5
```

./src/tests/fail_invalidfunctioncall.plt

```
1 fn plt():
2     print "plt"
3 end
4 fn pltr():
5     print "plt"
6 end
7 fn plt():
8     print "plt"
9 end
10 plt()
11
12
```

./src/tests/faillinebool.plt

```
1 bool a
2 a = true
3 line((a,a), (a,a))
4
5
```

./src/tests/fail_lineboolnum.plt

```
1 bool a
2 a = true
3 line((5,a), (10, a))
4
```

./src/tests/fail_line_numbert.plt

```
1 point x
2 x = (500 , 500 )
3 line(10, x)
4
5
```

./src/tests/fail_line_num_string.plt

```
1 string a
2 a = "5"
3 line((5,a), (10, a))
4
```

./src/tests/fail_lineplotnum.plt

```
1 point x
2 x = (500 , 500 )
3 line(x, 10)
4
5
```

./src/tests/fail_line_string.plt

```
1 string a
2 a = "5"
3 line ((a,a),(a,a))
4
5
```

./src/tests/fail_numassignbool.plt

```
1 num a
2 a = true
3
4
```

./src/tests/fail_numassign_string.plt

```
1 num a
2 a = "hi"
3
4
```

./src/tests/fail_num_sum_bool.plt

```
1 num a
2 a = 5 + true
3
4
```

./src/tests/failnumsumpoint.plt

```
1 point a
2 num b
3 b = 5 + a
4
5
```

./src/tests/failnumsumstring.plt

```
1 num a
2 a = 5 + "hi"
3
4
```

./src/tests/fail_point_decl_bool.plt

```
1 point a
2 a = (true, 5.5)
3
4
```

./src/tests/fail_{pointdeclboolvar}.plt

```
1 bool a
2 a = true
3 point b = (a,a)
4
5
```

./src/tests/fail_pointdecl_string.plt

```
1 point a
2 a = ("5.5", 5.5)
3
4
```

./src/tests/fail_point_decl_string_var.plt

```
1 string a
2 a = "5"
3 point b
4 b = (a,a)
5
6
```

./src/tests/fail_stringassignbool.plt

```
1 string a
2 a = true
3
4
```

./src/tests/fail_stringnumfloatasssign.plt

```
1 string a
2 a = 5.5
3
4
```

./src/tests/fail_stringnumintassign.plt

```
1 string a
2 a = 5
3
4
```

./src/tests/fail_stringsumbool

```
1 string a
2 a = "hi" + bool
3
4
```

./src/tests/fail_stringsumnum.plt

```
1 string a
2 a = "hi" + 5
3
4
```

./src/tests/fail_stringsumpoint.plt

```
1 point a
2 string b
3 b = "5" + a
4
5
```

./src/tests/failwhilecolon.plt

```
1 num a
2 a = 1
3 while (a < 10)
4 print a
5 a = a + 1
6 end
7
8
```

./src/tests/failwhilecondstring.plt

```
1 string a
2 a = "hi"
3 while (a):
4 print a
5 end
6
7
```

./src/tests/failwhileend.plt

```
1 num a
2   a = 1
3   while (a < 10):
4       print a
5       a = a + 1
6
7
```

./src/tests/passboolassign.plt

```
1 bool a
2 a = true
3
4
```

./src/tests/passbooldecl.plt

```
1 bool a
2 bool aaa
3 bool ab12pong
4
5
```

./src/tests/passbool_uassigned.plt

```
1 bool a
2 print a
3
4
```

./src/tests/pass_eol.plt

1 num a
2

./src/tests/pass_{for}regular.plt

```
1 num a
2   for a = 0; a < 5; a = a + 1 :
3     print a
4   end
5
6
```

./src/tests/passfunctionlistparam.plt

```
1 fn printList(list num a):
2     num j
3         for j=0; j<5; j=j+1 :
4             print a[j]
5         end
6     end
7
8 list num a
9 a = [1,2,3,4,5,6]
10 printList(a)
11
12
```

./src/tests/pass_function_np_nr.plt

```
1 #no parameter no return
2 fn addOne():
3     num a
4     a = 3
5 end
6
7 addOne()
8 num a
9 a = 5
10 a = a*5
11
12
13
```

./src/tests/passfunction_pnr.plt

```
1 #1+ parameter no return
2 fn addOne(num a, num b, num c):
3     a = a + 1
4     b = b + 1
5     c = c + 1
6     print (a+b+c)
7 end
8
9 addOne(0,0,0)
10 num a
11 a = 5
12 a = a*5
13
14
15
```

./src/tests/pass_{function}two_{fs}.plt

```
1 fn addOne() :
2     num a
3     a = 3
4     a = 3
5 end
6 fn addTwo(num aa, num b) :
7     num a
8     a = 5
9     print aa + b
10 end
11 fn addThree(num aa) :
12     num a
13     a = 3
14     print aa + a
15 end
16 fn addFour(num aa, num b) :
17     num a
18     a = 4
19     b = a + aa
20     print b
21 end
22
23 addOne()
24 addTwo(3,3)
25 addThree(3)
26 addFour(3,0)
27
28 num a
29 a = 5
30 a = a*5
31
32
33
```

./src/tests/pass_if plt

```
1 bool a
2 a = true
3 if a:
4     print "true"
5 end
6
7
```

./src/tests/pass_ifelse.plt

```
1 num a
2 a = 7
3 if a:
4     print "true"
5     if a:
6         print "its still true"
7     else:
8         print "there is something wrong"
9 end
10 end
11
12
```

./src/tests/pass/include_ib.plt

```
1 include plots
2 list num a
3 a = [100,80,95,66,70]
4 barGraph(a)
5
6
```

./src/tests/passlinefloat.plt

```
1 line ((5.5,5),(100,100.5))  
2  
3
```

./src/tests/passline_nums.plt

```
1 num a
2 a = 5
3 num b
4 b = 100.5
5 line ((a,a),(b,b))
6
7
```

./src/tests/passline_numspt.plt

```
1 point x
2 x = (500 , 500 )
3 line ((10 , 10) , x)
4
5
```

./src/tests/passlinepointnums.plt

```
1 point b
2 b = (100, 100)
3 line((5,5), b)
4
5
6
```

./src/tests/pass_{line}points

```
1 point a
2 a = (5,5)
3 point b
4 b = (100,100)
5 line(a, b)
6
7
```

./src/tests/pastlineptnumps.plt

```
1 point x
2 x = (500 , 500 )
3 line(x,( 10, 10))
4
5
```

./src/tests/passeLinegraph.plt

```
1 include plots
2
3 list num a
4 a = [50,53, 55, 53, 50, 45, 25, 10, 5, 3, 0, 2, 4, 10, 25, 45,60]
5 lineGraph(a)
6
7
```

./src/tests/pass/listappend.plt

```
1 list num a
2 a = [3, 5, 7, 9]
3 a.append(20)
4
5
```

./src/tests/passlistiterator plt

```
1 list num aa
2 aa = [1 ,2 ,3 ,4 ,5 ,6]
3 num j
4 for j=0; j<5; j=j+1 :
5 print aa[ j ]
6 end
7
8
```

./src/tests/pass/listlength.plt

```
1 num a
2 list num b
3 b = [1,2,3,4]
4 a = b.length()
5 print a
6
7
```

./src/tests/pass/listpop plt

```
1 list num a
2 a = [10, 20, 30, 40]
3 a.pop()
4
5
```

./src/tests/pass/list_remove.plt

```
1 list num a
2 a = [10, 20, 30, 40]
3 a.remove(1)
4
5
```

[./src/tests/pass_multilinecomment.plt](#)

```
1  /* yo
2   this
3   is
4   * MULTI LINR COMMENT
5   num a
6   print b
7   num num num string ;
8   these are not parsed
9  */
10
11 print "Hello World"
12 /* Simple Multi
13   Line comment
14   hi */ print "hello again"
15
16 /* backk */
17
18
```

./src/tests/passnumassignfloat.plt

```
1 num a
2 a = 5.5
3
4
```

./src/tests/passnumassignment.plt

```
1 num a
2 a = 5
3
4
```

./src/tests/passnumdecl.plt

```
1 num a
2 num abc
3 num A123
4 num a0
5
6
```

./src/tests/passnumdiff.plt

```
1 num a  
2 a = 10 - 5  
3  
4
```

./src/tests/passnumsum.plt

```
1 num a  
2 a = 10 + 5  
3  
4
```

./src/tests/passnumunassigned.plt

```
1 num a
2 print a
3
4
```

./src/tests/pass_{pointdeclfloat}.plt

```
1 point a
2 a = (5.5 , 5.5)
3
4
```

./src/tests/pass_{pointdeclint}.plt

```
1 point a
2 a = (5, 5)
3
4
```

./src/tests/passpointdeclnum.plt

```
1 num a
2 a = 5
3 point b
4 b = (a,a)
5 num aA01a
6
7 a = 5
8 point b007Bond
9 b007Bond = (aA01a, aA01a)
10
11
```

./src/tests/passpointdeclnumsum.plt

```
1 num a
2 a = 5
3 point p
4 p = (a+5, a+15)
5
6
```

./src/tests/pass_print_bool.plt

```
1 bool a
2 a = true
3 print a
4
5
```

./src/tests/passprintnum.plt

```
1 num a
2   a = 5
3   print a
4
5
```

./src/tests/passprintpoint.plt

```
1 print (1,2)  
2  
3
```

./src/tests/passprintstring plt

```
1 string a
2 a = "hi"
3 print a
4
5
```

./src/tests/pass_printXY.plt

```
1 num i
2   for i=0;i<10;i=i+1:
3     printXY("Ranjith", (100,100+20*i))
4   end
5
6
```

./src/tests/pass_return_stmt plt

```
1 num a
2 a = 8
3 if a > 5:
4     return 0
5 else:
6     print "something wrong"
7 end
8
9
```

./src/tests/pass_singlelinecomment.plt

```
1 num a
2
3 # this is a single line comment
4
5
6
```

./src/tests/pass_stringassign.plt

```
1 string a
2 a = "hi"
3
4
```

./src/tests/pass_stringdecl.plt

```
1 string a
2 string abc
3 string A123
4 string a0
5
6
```

./src/tests/pass_string_u_nassigned.plt

```
1 string a
2 print a
3
4
```

./src/tests/passwhile.plt

```
1 num a
2 a = 1
3 while (a < 10):
4     print a
5     a = a + 1
6 end
7
8
9
```

./src/tests/passwhilecondbool.plt

```
1 num a
2   a = 1
3   while (((a<5)) :
4     print a
5     a = a + 1
6   end
```

./src/tests/passwhilecondnum.plt

```
1 num a
2 a = 1
3 while (a):
4     print a
5     a = a + 1
6     if a==5:
7         a=0
8     end
9 end
10
11
```

./src/tests/testhello.plt

```
1 print "Hello world"  
2  
3
```

./src/tests/test_sum.plt

```
1 num a
2 a = -5 + 10
3 print a
4
5
```

./src/tests/test_vdecl.plt

```
1 string a
2 num b
3 bool c
4
5
```