Final Report: SHIL

Simulated Human Input Language

COMS W4115 Programming Languages and Translators

Fall 2008

Moses Vaughan (mjv2123@columbia.edu)

Binh Vo (bdv2112@columbia.edu)

Ian Vo (idv2101@columbia.edu)

Chun Yai Wang (cw2244@columbia.edu)

Contents

1. Introduction

1.1 Overview

SHIL is a language used primarily for developing HTML based automated bots. It provides the developer with an abstraction for automating interaction with web sites and users. From the server's perspective SHIL can be used to simulate user interactions, which is useful for many applications ranging from creating spiders to website test scripts. From the user's perspective SHIL can be used to implement custom user interfaces. In conjunction with automated server interaction this potentially can be used to alter existing interfaces for websites or provide interfaces to additional functionality built on top of existing website functionality.

The implementation language was Objective Caml (OCaml). Ocaml is a reliable language from the standpoint that many of the bugs that are difficult to catch with many other programming languages, even during run-time, are in fact caught during compile time. This quality is mainly due to its superior typing system which makes polymorphic and type checking abilities the forefront of its implementation. Ocaml has been wonderful for writing this SHIL in general because of its concise nature, which makes the source readable, as well as its strong sense of pattern matching that we could exploit for data types all over our interpreter’s structure. Looking at comparable interpreter’s source written in other languages we now truly begin to appreciate the value that OCaml’s various strengths bring.

1.2 Motivations

One of the main motivations for the production of our language is that many automated browsing tasks are written now in various languages, primarily PERL and Python. For example many services such as web search engines need to crawl across existing pages on the internet, or independent users often wish to automate data collection over various sites. SHIL intends to provide a language designed specifically for this task which will reduce the complexity of writing applications of this nature.

2. Language Tutorial

The Basics

Comments: The usual /* and */ are used for all comments.

Assignment: The arrow points the way. ‘<-’

One important fact to note is that SHIL is a dynamically typed language because of the fact that the environment is passed from the calling scope.

- `a <- 4;`

Comparison: Single Equals ‘=‘

- `“asdf” = “asdf”`

Functions: Use the “function” keyword using the template.

- function functName(<type> <var>, <type> <var>) ->

  <return type> {<body>}

- Ex/

```
function add(integer a, integer b) -> integer{
                          return a + b;}
```

Function Call: Note that the evaluation order of all functions in SHIL is that of applicative evaluation. A given call’s arguments are evaluated before passing control to the function for its evaluation.

- `result <- func_name(arg1, arg2);`

The data types for SHIL include those in the chart below:

integer	**real**	**boolean**
function	**map**	**array**
If	**while**	**foreach**
break	**end**	**fun**
use	**return**	**TRUE**
FALSE		**string**

Special Symbols also play a key role in the use of SHIL as they are used within the formation of expressions and various statements which exploit the power in SHIL.

For a conclusive listing of the symbols and their respective meanings in SHIL see the chart below:

Lexeme	Usage
<-	assignment
+ - / *	math
"	string**
;	statement termination
.	struct reference
[ ]	array reference
( )	Logical grouping
& | ! = < > >=	Boolean Operators

Data Types are used to hold specific forms of data. SHIL is of course no different from other typed languages since it is in fact strongly typed.

SHIL uses the usual suspects.

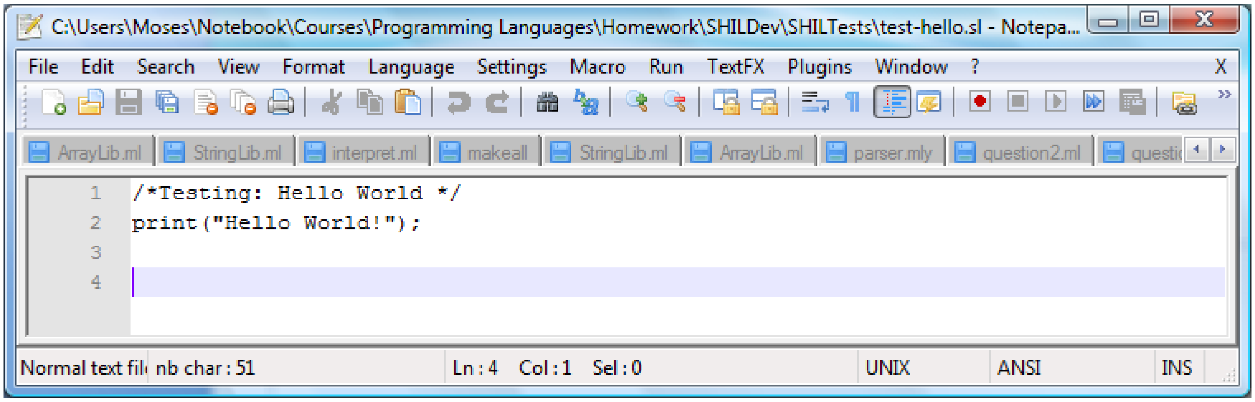
- Integer
- Boolean
- Map
- Array
- String
- Real – Acts as a floating point number

No introduction to a language would be complete without the infamous “Hello World!” example. It displays the control flow of SHIL is similar to that of a scripting language, and not of one like C++,Java as there is no necessary main method. The statements will get executed in the order of which they are received.

“Hello World!” in SHIL

Control Flow resembles that of a scripting language.

- No main().
- Hello World needs only one line of actual code!



```
/*Testing: Hello World */
print("Hello World!");


```

Like many programming languages SHIL provides a way similar to that of C++, Java of choosing between two of more execution paths within a program.

Selection in SHIL is accomplished by the use of:

- If statements || If then else
  - If <conditional> then<expression>
  - If <conditional> then<expression>else <expression>
    - /* Testing: If-Else Statement and execution afterwards*/

Example:

```
function main () -> integer {
     if (FALSE) then
          print("True");
      else print("False");
     print("After"); /* Outside of if scope */
}
main();
```

Data Structures: The data structures that are supported by the SHIL language are the Array and the Map. One great thing to note about arrays is that they are dynamically sized initially to whatever you evaluate their contents to be, but once they are sized they will keep this static size for their lifetime.

- Arrays
  - Declaration: <type>[] <varName>;

```
integer[] arr; /* No need for numerical sizing */
```

  - Assignment: <varName><- array{element 1,…., elementN};

```
arr <- array{5,10,15};
arr[1] <- 5;
```

- Maps
  - Declaration: <type>[[<type>]] <varName>;

```
integer[[string]] mymap;
```

- Assignment: <varName><- map{key1 -> val1 ,…., keyN -> valN};

```
mymap <- map {"a" -> 11, "b" -> 26, "c" -> 52};
```

3. Language Reference Manual

Lexical Conventions

- Overview

This section covers the lexical conventions within the SHIL language that constitute various tokens including elements such as data types, data structures, reserved words and symbols. A token is a series of contiguous characters that the compiler will treat as one individual element. The scanner will parse tokens to be the longest string of characters that can create a token type.

- White Space

White space is classified only by blank spaces, newlines, tabs, and within the scanner comments are considered whitespace. The only purpose of whitespace is to separate tokens, and can essentially be rendered useless except for human readability issues.

- Comments

/* is the opening of a comment and */ is the closing the respective block. There are no single line only comment lexemes. Once a comment opening is seen, everything up until the end of the */ lexeme is considered invisible to the compiler.

- Identifier

An identifier is a sequence of alphanumeric and non-alphanumeric characters. Note that the first character can be anything other than a digit. Casing is distinctive in all positions of an identifier string, meaning that one identifier is not equivalent to another unless they both follow identical character order and their characters must have identical casing in their respective positions.

- Keywords

The following list is the keywords within the SHIL language. They cannot be used for any programmatic purpose other than their distinct function.

integer	real	boolean
function	map	Array
If	while	foreach
break	end	fun
use	return	true
false	maybe	string

- Operators

An operator is used to specify an operation to be performed. The chart below gives operators as well as their necessary functions.

Operator	*Functions*
<-	assignment
+ - / *	math
"	string**
;	statement termination
[ ]	array reference
( )	Logical grouping
& | ! =	Boolean
< > >=	Operators

Note: ** signifies that this operator must occur in a pairing

Data Types and Structures

- Data Types

String - *string* is any finite sequence of characters which include letters, numerals, symbols and punctuation marks. A " is used to signify the beginning of a string and an additional one used to signify the end of the string.

Integer - An integer is a whole number that can be positive, negative, or zero.

Real - Real numbers include rational and irrational numbers, but must be signified in decimal format within SHIL. So therefore pi is not an acceptable value.

Boolean - Boolean represents logical variables and can be of the values true or false.

- Data Structures

Map – An associated array which holds key value pairings. The operation of finding a value with a given key is called a lookup.

Array - A linear data structure where each element holds the same data type. The structure itself occupies a contiguous block of storage.

Functions

SHIL functions can take multiple arguments, return either a single basic data type value or nothing, and modify multiple existing values

```
function (type1 arg1, type2 arg2 …) -> <return_type> {

        /* arbitrary code */
};
```

Functions can use the keyword ‘function’ or just ‘fun’ as shorthand.

Args are passed by value (with the exception of data structures). The return value is specified with the 'return' operator.

For example, the following function can return a sum:

```
sum <− function (integer x, integer y) −> integer {
        return x + y;
};
```

User-defined libraries of functions can be stored in a separate file and included

with the 'use' directive:

```
use “filename”;
```

Expressions

Expressions are token groups which result in a value, they fall into several categories.

Constants: result <− “string”; result <− 1; result <− 1.0; result = true;

String constants are always surrounded by double quotes. Digits containing a '.' are real valued, otherwise integer-valued, and booleans are one of either 'true' or 'false'.

Function Calls: result <− func_name(arg1, arg2);

Function calls return at most one value.

Math/boolean operators: result <− val1 <operator> val2;

Boolean operators ('&', '|', '!') always return a boolean value. Mathematical operators ('+', '-', '*', '/', '%') return a real value if either of the input values is real, and an integer value otherwise.

Comparison operators: result <− “asdf” = “asdf”;

Comparison operators ('=', '!=', '<' , '>', '<=', '>=') return a boolean value from two values of same type.

Assignment: result <− foo <− bar;

Assignment operators also return the value assigned.

Statements

Statements are complete SHIL instructions, are terminated by the ';' character, and fall into four categories:

Expression

<expression>;

A lone expression may be evaluated as a statement.

Block

{<statement>; <statement>; <statement>;};

Curly braces can be used to group several statements into one statement.

Conditional

if <expression> then <statement>; else <statement>;

'if' can be used with a boolean-valued expression to execute one of two statements.

Iteration

while <expression> <statement>;

'while' can be used with a boolean-valued expression to repeatedly execute a statement.

foreach <key_name> <value_name> in <array_or_map_name> statement;

'foreach' can be used to execute a statement once for each element in a map or array. key_name and value_name will become variables within the context of this statement. For an array, key_name is an integer index from 0 to the length of the array, and for a map, key_name is the key of the map.

Return

return <expression>;

Within a function body, this can be used to terminate execution of the function and return a value.

Namespace

Variable names and function names will occupy the same case sensitive namespace, and can be assigned with the <- operator. Function bodies will use a private namespace, and statement blocks will inherit the parent namespace, however any variables declared within the block will expire with the termination of the block.

SHIL specific functions

The SHIL language has a number of built-in functions that are always available. They are categorized here according to purpose:

Internet Interaction

string <- ***sendRequest(****map)*

Returns the HTML result as a string, given an HTTP request of type map.

map <− ***parseHTML****(string)*

Returns a nested map representation of an HTML page in string format.

string <− ***generateHTML****(map)*

Returns a string representation of an HTML page in map format.

*map <− s****howHTML****(string)*

Displays a given HTML code of string in the user's default web browser and returns the next HTTP request as a map.

String Manipulation

string <− ***substring****(string, int, int)*

substring(inputString startInd endInd)
Returns a string that is the substring of the given input string, bounded by the starting and ending integer indexes.

int <− ***stringLength****(string)*

stringLength(inputString)
Returns the length of the given input string.

*int <– **stringFind**(string, string)*

stringFind(inputString searchString)
Returns the index within the given input string of the first occurrence of the specified search string.

*Int[ ] <– **stringFindAll**(string, string)*

stringFindAll(inputString searchString)
Returns an array of indexes within the given input string of all occurrences of the specified search string.

*String[ ] <– **splitString**(string, string)*

***splitString**(delimString inputString)*
Returns an array of strings resulting from splitting the given delimiter string according to a regular expression provided in string format upon the input string provided. For instance, (`splitString` "[\t]+" s) splits `s` into blank-separated words.

*string <– **stringReplace**(string, string, string)*

stringReplace(searchString replaceString inputString)
Replaces all instances of the search string with the replacement string in the given input string.

*string <– **stringToUpper**(string)*

stringToUpper(inputString)
Returns the given input string with all its characters converted to upper case using the rules of the default locale.

*string <− **stringToLower**(string)*

stringToUpper(inputString)
Returns the given input string with all its characters converted to lower case using the rules of the default locale.

Data Manipulation

*array <− **sort**(array, boolean)*

sort(inputArr ascending)
Returns a sorted version of the provided array in ascending order if ascending boolean is true and returns a sorted version of the provided array in descending order if false.

*array <− **randomize**(array)*

randomize(inputArr)
Returns a randomized permutation of the provided input array.

*array <− **getKeys**(map)*

getKeys(inputMap)
Returns an array of the keys for the provided input map.

*array <− **getValues**(map)*

getKeys(inputMap)
Returns an array of the values for the provided input map.

int <- ***length****(array)*

length(inputArr)
Returns the number of elements for the provided input array.

4. Project Plan

4.1. Planning, Specification, Development and Testing processes

Planning

- Multiple meetings were initially created to discuss planning topics such as:

    1.) Language's purpose

    2.) What type of functionality would be necessary

    3.) How best to represent this functionality

    4.) Syntax details that would simplify semantic goals

    5.) Developing a specification to follow through on our initial plans

    6.) Develop specific hard and soft deadlines for the work to be completed in a timely manner.

    7.) How to divide implementation specifics amongst team

As a direct result of the planning phase the specification was produced.

Specification

Create target objectives for the following:

Lexical Conventions - Syntactic nature of the language

- Overview
- White Space
- Comments
- Identifier
- Keywords
- Operators

Data Types and Structures - Representation of available user data

- Data Types
- Data Structures

Functions – Executable blocks of instructions.

Expressions - Single or Nested evaluated instruction

- Constants
- Function Calls
- Math/boolean operators
- Comparison operators

Statements

- Assignment
- Block
- Conditional
- Iteration
- Return

Namespace – Module Scoping

SHIL specific functions

- Internet Interaction
- String Manipulation
- Data Manipulation

Adhering to this spec a systematic and uniform development process would occur.

Development- Once the requirements and the spec were created design which the project would follow emulated that which would closely follow a typical interpreter's design. The implementation thus produced the following products in their respective orders listed.

- Scanner/Lexer
- Parser
- AST
- Interpreter

The process was not that of a clear cut waterfall approach but instead it had more of a cyclic nature. As we developed the interpreter we saw design flaws in our language and would thus have to back up and correct those misguided errors in both the scanner and parser. This became a very iterative process as it happened quite frequently.

Testing- The process of testing we chose was that of an iterative one that followed closely to the development process in a whitebox as well as blackbox fashion.

As we developed any new key addition to the system we would rerun our regression test suite to ensure that the project base was as stable as before we made those changes. For a complete testing plan that was followed in the creation of the SHIL language see Test Plan.

4.2. Programming Style

We followed a strict style guideline between the members of our team to ensure a readable yet productive product which could be handed off to each of the members if need be. See below:

Commenting

- All modules submitted to repository must begin a brief summary comment on what its use.
- Give descriptions of complicated code segments explaining its purpose
- Avoid unobvious abbreviations.
- Each Routine must provide a description prior to implementation.
- Avoid adding extraneous commentary.
- Avoid commenting obvious functionality.

Statements

- Avoid exceeding 80 characters for any LOC (line of code).
- Indent based on the enclosing block's indentation.
- Ensure at most one data declaration per LOC.

Functions & Modules

- Functions must adhere to Camel Casing convention
- Modules names should reflect their real world counterparts in regards the interpreter.

Variables

- Variable should be between 5- 12 characters long
- Provide descriptive names describing what is being stored, unlike that of simply 'i' or 'j'.
- Literals should be coded of the form Literal_<data type>
- Arguments to modules should follow Camel Case conventions.

General

- All code submitted to repository must compile and pass all tests which it passed before the revision.
- All submissions to the repository must have time stamped comments within the repository log reflecting what was changed.

4.3. Project Timeline

The table below displays the dates of key deadlines towards the development of the SHIL language.

Date (DD-MM-YYYY)	Accomplishment
09-10-2008	Group Formation Complete
09-22-2008	Language Proposal Defined
09-25-2008	Lexical Conventions/ Data Types and Structures Defined
10-03-2008	Functional/Expressional syntax Defined
10-13-2008	SHIL library content Defined
10-20-2008	Language Reference Manual Complete
10-27-2008	Scanner Complete
11-03-2008	Parser Complete
11-15-2008	Testing Suite Complete
11-28-2008	Code Generation Complete
12-3-2008	SHIL Libraries Defined and Tested
12-15-2008	Project Features Complete

4.4. Team Member Roles and Responsibilities

The team member distribution of work regarding this project is displayed in the table below.

Member Name	SHIL Contributions
Moses Vaughan	SHIL Libraries, Testing Modules, Interpreter
Chun Yai Wang	Scanner, Parser, Interpreter
Binh Vo	Interpreter , AST Design, Testing Modules
Ian Vo	SHIL Libraries, Testing Module, Interpreter

Listed below are the team tasks that we accomplished together in order to get a general consensus of the direction of the SHIL language.

- Language Proposal
- Language Reference Manual
- Final Report
- Grammar/ Expression/Statement Design
- Library Functionality Design

4.5. Software Development Environment Used

The group used a variety of developing environments including:

- Cygwin 1.5, a linux emulator for Vista.
- Ubuntu 8.4

The version control system utilized was that of Google Code. In addition to this we utilized the interaction system that Ocaml comes with quite a lot for building units of the code and ensuring their compatibility with existing counterparts.

4.6. Project Log

The project log regarding the implementation of SHIL is listed below:

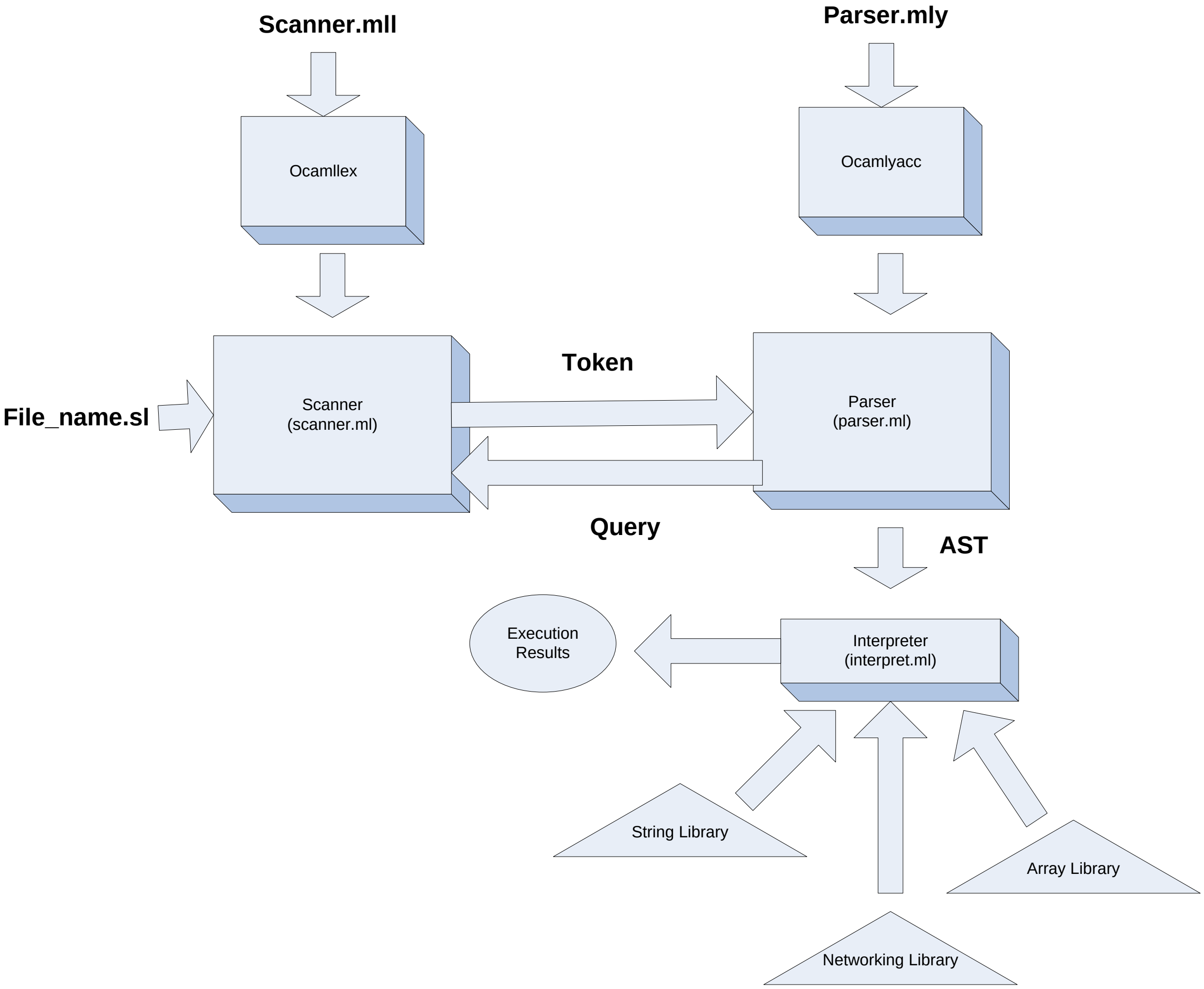
Date (DD-MM-YYYY)	Accomplishment
09-14-2008	Project Initialed
09-18-2008	Project Concept Complete
09-20-2008	Code Design First Draft
09-21-2008	Code Design Complete/ Language Ref. Manual Complete
09-23-2008	Development Environment/ Code Repository Configured
09-25-2008	Lexical Conventions/ Data Types and Structures Defined
10-01-2008	Functional/Expressional syntax Drafted
10-03-2008	Functional/Expressional syntax Defined
10-10-2008	SHIL library concept initiated
10-13-2008	SHIL library content Defined
10-17-2008	Language Reference Manual First Draft
10-20-2008	Language Reference Manual Complete
10-25-2008	Scanner Initiated
10-27-2008	Scanner Complete
10-29-2008	Parser Initiated
10-31-2008	Parser Bugs Detected
11-03-2008	Rework Design /Parser Bugs Resolved/ Parser Complete
11-05-2008	Testing Phase Begins
11-09-2008	Initial Tests Implemented
11-15-2008	Tests Reworked/ Testing Suite Complete

11-20-2008	Interpreter Initially Developed
11-28-2008	Basic Code Generation Complete
11-29-2008	String/Networking Libraries Implemented
11-30-2008	Data Structure Manipulation Libraries Implemented
12-01-2008	String/Networking Lib’s Tested/Reworked
12-02-2008	Data Structure Lib’s Tested/Reworked
12-3-2008	SHIL Libraries Defined and Tested
12-5-2008	Code Generation incorporates Maps and Arrays
12-8-2008	Code Generation incorporated variable sized Arrays
12-15-2008	Project Features Complete
12-16-2008	Final Report Documentation Initiated
12-18-2008	Final Report Documentation Complete

5. Architectural Design

5.1. Major Components Translator

The major components of the translator are seen below:



Note the modules for SHIL specific ocaml files (.ml) are shown within each component to display their representation within our implementation.

5.2. Interfaces between Components

Scanner.mll: This is the scanner representation that will be taken by ocamllex to create the finite state automaton model that is the scanner represented in scanner.ml.

Parser.mly: This is the parser representation that will be taken as input by ocamlyacc and thus will result in the parser component of our system as in parser.ml.

The scanner and parser will interact together by the process of the scanner breaking the source file up by lexemes, then the parser needing the next token thus querying the scanner for it will get the token returned. Once the tokens are fully returned then an AST will become the result of this process. The AST will then be fed as input to the Interpreter and will thus become the execution results.

5.3. Who Implemented Each Component

The implementation of the scanner and parser was worked on by Chun-Yai with some edits being made by Binh.

Every group member had a hand in the implementation of the Interpreter since it is the largest piece of functionality in our system. Binh built a stable skeleton that was functional and tested where the other members could make necessary additions to it. Moses added functionality to incorporate each of the library functions within the String and Array Libraries. Ian added functionality to incorporate all the library functions within the Networking Library. Chun-Yai and Binh modified the infrastructure, including the parser and interpreter, to incorporate many aspects of the system that were initially looked over in the design and thus needed to be added to adhere to the spec of our language.

The String and Array Libraries were implemented by Moses, while the Networking Library was implemented by Ian.

All group members had a hand in developing the various tests to ensure all the functionality acted as intended.

The Final Project Report was written primarily by Moses and Ian and Chun-Yai.

6. Test Plan

6.1 Representative Source Language Programs

Author: Ian Vo

GetGoogleLinks.sl

```
HtmlDoc[] x;

x <- url("http://www.google.com/search?q=" + getarg(1));


string[[string]][] tags <- getArrayByTag("a", x);


tags <- filterArrayByAttribute("class", "l", tags);


integer index;

string[[string]] value;


print("Here are the search links:");

foreach (index value in tags) {

      print(value[["href"]]);

}
```

output:

```
      moses@digita1:~/plt-final/plt-final$ ./shil "plt" <
SHILTests/getGoogleLinks.sl

      Here are the search links:

      http://www.plt.org/

      http://www.plt-scheme.org/
```

```
            http://filext.com/file-extension/PLT

            http://www.plt.com/

            http://en.wikipedia.org/wiki/PLT

            http://www.cs.brown.edu/research/plt/

            http://www.freedownloadscenter.com/Best/plt-viewer-free.html

            http://www.chem.wisc.edu/areas/reich/plt/winplt.htm

            http://www.cs.rice.edu/CS/PLT/

            http://www.wipo.int/treaties/en/ip/plt/
```

getFlickrImages.sl

```
      HtmlDoc[] x;

      x <- url("http://flickr.com/search/?q=" + getarg(1));


      string[[string]][] tags <- getArrayByTag("img", x);


      tags <- filterArrayByAttribute("class", "pc_img", tags);


      integer index;

      string[[string]] value;


      print("Here are the image:");

      foreach (index value in tags) {

            print(value[["alt"]] + ": " + value[["src"]]);

                  print("----------------------------------------------------
-------------");

      }
```

output：

```
moses@digital:~/plt-final/plt-final$ ./shil "car" <
            SHILTests/getFlickrImages.sl

Here are the images:

London Car in Black &amp; White B/W Dream Car by davidgutierrez2007:
http://farm4.static.flickr.com/3016/2317733535_7eab96bc76_m.jpg

-----------------------------------------------------------------

Police Staff Car - Ottawa 05 07 by Mikey G Ottawa:
http://farm1.static.flickr.com/224/494531076_a9f6164a24_m.jpg

-----------------------------------------------------------------

A Gaudi Car. by Osvaldo:
http://farm1.static.flickr.com/11/16717432_a0ec8852bf_m.jpg

-----------------------------------------------------------------

Baby you can drive my car by in touch:
http://farm3.static.flickr.com/2130/2405330222_a9142ef874_m.jpg

-----------------------------------------------------------------

Miranda Car Park HDR 2 by alexkess:
http://farm1.static.flickr.com/84/419953013_3e3d649cb0_m.jpg

-----------------------------------------------------------------

My Car is Looking Hot! by blueoneiam:
http://farm1.static.flickr.com/60/229227200_b6899ed500_m.jpg

-----------------------------------------------------------------

my new car... by caucasiandora:
http://farm2.static.flickr.com/1059/1050720611_7fcd0002cd_m.jpg

-----------------------------------------------------------------

... etc ...
```

6.2 Test Suites & Who Did What

Functionality Tests

Tests were designed to rigorously ensure that all of our underlying types, expressions and statements were functioning properly.

Test File	Author	Code	Output
test-arith1.sl	Moses Vaughan	/*Testing: Basic arithmetic within a function arg */ print(39 + 3);	42

test-arith2.sl	Ian Vo	/* Testing: Arithmetic & order of operations */
		function main () -> integer {
		        print(1 + 2 * 3 + 4); /*Should be 11 */
		}
		main();
			11

test-arith3.sl	Moses Vaughan	/*Testing: Basic arithmetic within a function arg */
		integer a;
		integer b;
		a <- 39;
		b <- 3;
		print(a + b);
			42

test-array1.sl	Moses Vaughan	/* Testing: ForEach loop with and without index, defined as declared*/
		integer[] arr <- array{5,10,15};
		arr[1] <- 100;
		print(arr[1]);
		print(arr[2]);
			100
			15

test-array2.sl	Ian Vo	/* Testing: ForEach loop with and without index defined after declaration*/
		integer[] arr;
		arr <- array{5,10,15};
		
		function main () -> integer {
		  arr[1] <- 100;
		  print(arr[1]);
		  print(arr[2]);
		}
		
		main();
			100
			15

test-arraylib.sl	Moses Vaughan	integer[] arr <- array {5,2,7,1,2};
		integer[[string]] mapIntStr <- map {5->"five", 1->"one", 3->"three"};
		
		integer index;
		integer value;
		
		inprint("Array has: ");
		index <- 0;
		value <- 0;
		foreach (index value in arr) {
		        inprint(value); inprint(",");
		}
		print("");
		inprint("Length is: ");
		print(length(arr));
		print("Sorting array in ascending order...");
		arr <- sort(arr, TRUE); /* Sort ascending order */
		inprint("**Sorted array is now: ");
		index <- 0;
		value <- 0;
		foreach (index value in arr) {
		        inprint(value); inprint(",");
		}
		print("");
		
		/* We know this function works
		print("Randomizing Array");
		arr <- randomize(arr);
		index <- 0;
		value <- 0;
		foreach (index value in arr) {
			Array has: 5,2,7,1,2,
			Length is: 5
			Sorting array in ascending order...
			**Sorted array is now: 1,2,2,5,7,
			Displaying keys in map:
			5,1,3,
			Displaying values in map:
			five,one,three,

		inprint(value); inprint(",");
		}
		print("");
		*/
		
		/**** MAPS ****/
		integer[] keys <- getKeys(mapIntStr);
		string[] values <- getValues(mapIntStr);
		print("Displaying keys in map:");
		index <- 0;
		value <- 0;
		foreach (index value in keys) {
		        inprint(value); inprint(",");
		}
		print("");
		print("Displaying values in map:");
		index <- 0;
		value <- 0;
		foreach (index value in values) {
		        inprint(value); inprint(",");
		}
		print("");	

test-fib.sl	Ian Vo	-------------------------------
		/*Testing: Recursive Function Calls */
		function fib (integer x) -> integer {
		  if (x < 2) then return 1;
		  else return fib(x-1) + fib(x-2);
		}
		
		print(fib(0));
		print(fib(1));
		print(fib(2));
		print(fib(3));
		print(fib(4));
		print(fib(5));
			1
			1
			2
			3
			5
			8

test-for1.sl	Ian Vo	/* Testing: ForEach loop with and without index defined on array*/
		integer[] arr <- array{10,20,30};
		integer index <- 0;
		integer value <- 0;
		foreach (index value in arr) {
		        print(index);
		        print(value);
		         if (index = 1) then break;
		}
		print(index);
			0
			10
			1
			20
			1

test-map1.sl	Moses Vaughan	/* Testing: ForEach loop with and without index defined on map*/
		integer[[string]] mymap;
		mymap <- map {"a" -> 11, "b" -> 26, "c" -> 52};
		print(mymap[["b"]]);
		string index <- "";
		integer value <- 0;
		foreach (index value in mymap) {
		        print(index);
		        print(value);
		         if (value = 26) then break;
		}
		print(index);
			26
			a
			11
			b
			26
			b

test-func1.sl	Ian Vo	/* Testing: Function call and return value */
		function add (integer a, integer b) -> integer {
		  return a + b;
		}
		
		integer a;
		a <- add(39,3);
		print(a);
			42

test-	Moses Vaughan	/* Testing: Global Variable Assignment */	42

global1.sl		integer a;
		integer b;
		
		function printA () -> integer {
		  print(a);
		}
		
		function printB () -> integer {
		  print(b);
		}
		
		function incAB () -> integer {
		  a <- a + 1;
		  b <- b + 1;
		  return 0;
		}
		
		function main () -> integer {
		        a <- 42;
		        b <- 21;
		        printA();
		        printB();
		        incAB();
		        printA();
		        printB();
		}
		
		main();
			21
			43
			22

test-hello.sl	Ian Vo	/*Testing: Hello World */ print("Hello World!");	Hello World!

test-if1.sl	Ian Vo	/* Testing: If-Else statement (no afterwards) */
		if (TRUE) then print(42);
		else print(17);
		if (FALSE) then print("foo");
		else print("bar");
			42
			bar

test-if2.sl	Moses Vaughan	/* Testing: If-Else Statement and execution after */
		function main () -> integer {
		  if (TRUE) then print("Yes"); else print("No");
		  print("After");
		}
		
		main();
			Yes
			After

test-if3.sl	Moses Vaughan	/* Testing: If Statement (No else) and execution after */
		function main () -> integer {
		  if (FALSE) then print("True");
		  print("After");
		}
		
		main();
			After

test-if4.sl	Ian Vo	/* Testing: If-Else Statement and execution afterwards */
		function main () -> integer {
		  if (FALSE) then print("True"); else print("False");
		  print("After");
		}
		main();
			False
			After

test-stringlib.sl	Moses Vaughan	/* String Library Functions available: * substring, stringLength, stringFind, stringReplace, * stringToUpper, stringToLower, stringFindAll, splitString */ string a <- "String LibRaRy"; print("**String is:");	**String is: String LibRaRy **Length: 14 **Substring from 0 to 5 is: Strin **Capitalizing... STRING LIBRARY

		print(a);
		print("**Length:");
		print(stringLength(a));
		print("**Substring from 0 to 5 is:");
		print(substring(a,0,5));
		print("**Capitalizing...");
		print(stringToUpper(a));
		print("**Lower Casing...");
		print(stringToLower(a));
		print("**Restoring string and finding string \"RaR\"");
		print(stringFind(a,"RaR"));
		print("**Replacing string \"RaR\" with \"rar\"");
		a <- stringReplace("RaR","rar",a);
		print(a);
		print("**Searching for lower case r\'s");
		integer[] arr <- stringFindAll(a,"r");
		integer index <- 0;
		integer value <- 0;
		foreach (index value in arr) {
		        print(value);
		}
		print("**Splitting string by space");
		string[] arr1 <- splitString(" ",a);
		index <- 0;
		value <- 0;
		foreach (index value in arr1) {
		        print(value);
		}
			**Lower Casing...
			string library
			**Restoring string and finding string "RaR"
			10
			**Replacing string "RaR" with "rar"
			String Library
			**Searching for lower case r's
			2
			10
			12
			**Splitting string by space
			String
			Library

test-var1.sl	Ian Vo	/* Testing: Variable Assignment */
		function main () -> integer {
		
		  integer a;
		  a <- 4;
		  print(a);
		}
		
		main();
			4

test-var2.sl	Ian Vo	/* Testing: Variable Assignment */
		function main () -> integer {
		
		  integer a <- 4;
		  print(a);
		}
		
		main();
			4

test-while1.sl	Moses Vaughan	/* Testing: While loop */
		function main () -> integer {
		  integer i;
		  i <- 5;
		  while (i > 0) {
		    print(i);
		    i <- i - 1;
		  }
		}
		main();
			5
			4
			3
			2
			1

test-while2.sl	Moses Vaughan	/* Testing: While loop */
		function main () -> integer {
		  integer i;
		  i <- 5;
		  while (i > 0) {
		    print(i);
		    if (i = 3) then break;
		    i <- i - 1;
		  }
		  print(i+10);
		}
		main();
			5
			4
			3
			13

Component Tests

Scanner:

```
#load "scanner.cmo";;
#use "testlib.ml";;

addtest "SingleValueTokenization"
        (function () -> Scanner.token (Lexing.from_string "integer"))
        (Parser.INT);;

addtest "DifferentValueTokenization"
        (function () -> Scanner.token (Lexing.from_string "real"))
        (Parser.REAL);;

addtest "HeadTokenization"
        (function () -> Scanner.token (Lexing.from_string "integer integer"))
        (Parser.INT);;

addtest "TokenizationHandlesWhitespace"
        (function () -> Scanner.token (Lexing.from_string "   integer  "))
        (Parser.INT);;

addtest "TokenizationRequiresWhitespace"
        (function () -> Scanner.token (Lexing.from_string "integerinteger"))
        (Parser.ID "integerinteger");;

addtest "TokenizationIgnoresComments"
        (function () -> Scanner.token (Lexing.from_string "/*foo*/ integer
/*bar*/"))
        (Parser.INT);;

addtest "TokenizationRecognizesInt"
        (function () -> Scanner.token (Lexing.from_string "12"))
        (Parser.LITERAL_INT(12));;

addtest "TokenizationRecognizesReal"
```

```
        (function () -> Scanner.token (Lexing.from_string "12.0"))
        (Parser.LITERAL_REAL(12.0));;

addtest "TokenizationRecognizesBool"
        (function () -> Scanner.token (Lexing.from_string "FALSE"))
        (Parser.LITERAL_BOOL(false));;

addtest "TokenizationRecognizesString"
        (function () -> Scanner.token (Lexing.from_string "\"Hello\""))
        (Parser.LITERAL_STR("Hello"));;

addtest "TokenizationRecognizesStringWithSpace"
        (function () -> Scanner.token (Lexing.from_string "\"Hello World\""))
        (Parser.LITERAL_STR("Hello World"));;

runtests();;
```

<u>Parser:</u>

```
#load "parser.cmo";;
#load "scanner.cmo";;
#use "ast.mli";;
#use "testlib.ml";;

addtest "DeclarationParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "integer a;"))
        ([Ast.Declare (Ast.Type_int,"a")]);;

addtest "AssignmentParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "a <- 5;"))
        ([ Ast.Expr(Ast.Assign ((Ast.IdVar "a"),(Ast.Literal (Ast.Literal_int
5)))) ] );;
```

```
addtest "ArithmeticParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "1 + 5;"))
        ([ Ast.Expr(Ast.Binop ((Ast.Literal (Ast.Literal_int 1)),
                      Ast.Add,
                      (Ast.Literal (Ast.Literal_int 5))))] );;


addtest "NestedExpressionParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "a <- 1 + 5;"))
        ([Ast.Expr(Ast.Assign ((Ast.IdVar "a"),
                       (Ast.Binop ((Ast.Literal (Ast.Literal_int 1)),
                                   Ast.Add,
                                   (Ast.Literal (Ast.Literal_int 5)))))))] );;


addtest "SequentialExpressionParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "integer a; a <- 1;"))
        ([Ast.Declare (Ast.Type_int,"a");
          Ast.Expr (Ast.Assign ((Ast.IdVar "a"),(Ast.Literal (Ast.Literal_int
1))))]);;


addtest "FunctionDeclarationParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "function foo () -> integer
{return 5;}"))
        ([Ast.FuncDecl ("foo",[],Ast.Type_int,Ast.Block [(Ast.Return
(Ast.Literal (Ast.Literal_int 5)))])]);;


addtest "FunctionDeclarationParsingWithArgs"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "function foo (integer x, integer
y) -> integer {return 5;}"))
        ([Ast.FuncDecl ("foo",[(Ast.Type_int, "x"); (Ast.Type_int,
"y")],Ast.Type_int,Ast.Block [(Ast.Return (Ast.Literal (Ast.Literal_int
5)))])]);;


addtest "FunctionCallParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "print(5);"))
        ([Ast.Expr (Ast.Call ("print", [(Ast.Literal (Ast.Literal_int
5))]))]);;


(*
addtest "StructDeclareParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "struct foo {};"))
        ( [ Ast.DeclareStruct("foo",Ast.Block([])) ] );;


addtest "StructDeclareParsingWithInt"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "struct foo {integer a;};"))
        ( [ Ast.DeclareStruct("foo",Ast.Block([Ast.Expr(Ast.Declare
("integer","a"))])) ] );;


addtest "StructDeclareVariable"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "struct foo myvar;"))
        ( [ Ast.Expr( Ast.Struct("foo","myvar")) ] );;
*)


addtest "ArrayDeclareParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "integer[5] intArr;"))
        ([Ast.Declare( Ast.Type_Array(Ast.Type_int, 5), "intArr")]);;


addtest "MapDeclareParsing"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "integer[[string]] strMap;"))
        ([Ast.Declare( Ast.Type_Map(Ast.Type_int, Ast.Type_str),
"strMap")]);;


addtest "TestArrayLiteral"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "a <- array {5,2};"))
        ([
Ast.Expr(Ast.Assign(Ast.IdVar("a"),Ast.Literal(Ast.Literal_array([Ast.Literal
_int 5; Ast.Literal_int 2])))) ]);;
```

```
addtest "TestArrayLiteralEmpty"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "a <- array {};"))
        ([
Ast.Expr(Ast.Assign(Ast.IdVar("a"),Ast.Literal(Ast.Literal_array([]))))]);;


addtest "TestMapLiteralInt"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "a <- map { 5->1, 2->0};" ))
        ([Ast.Expr(Ast.Assign(Ast.IdVar("a"),
                   Ast.Literal(Ast.Literal_map([(Ast.Literal_int 5,
                                                 Ast.Literal_int 1);
                                                (Ast.Literal_int 2,
                                                 Ast.Literal_int 0)
                                               ]))))]);;


addtest "TestMapLiteralEmpty"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "a <- map {};" ))
        ([Ast.Expr(Ast.Assign(Ast.IdVar("a"),
                   Ast.Literal(Ast.Literal_map([]))))]);;


(*
addtest "StructAssignReference"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "a.b <- c.d;" ))
        ([ Ast.Expr( Ast.Assign( Ast.Structref("a","b"),
Ast.Var(Ast.Structref("c","d")) ) ) ]);;
*)


addtest "ArrayAssignReference"
        (function () -> Parser.program Scanner.token
                        (Lexing.from_string "a[0] <- a[1];" ))
        ([ Ast.Expr(Ast.Arrayassign(Ast.IdVar("a"),
                                    Ast.Literal(Ast.Literal_int(0)),

Ast.Arrayref(Ast.IdVar("a"),Ast.Literal(Ast.Literal_int(1)))))]);;
```

```
runtests();;
```

Reasons Behind the Tests - Why and How these test cases were chosen

Test Name	Reason Needed
test-arith1.sl	Testing: Basic arithmetic within a function arg.
test-arith2.sl	Testing: Arithmetic & order of operations
test-arith3.sl	Basic arithmetic within a function arg
test-array1.sl	Testing: ForEach loop with and without index defined
test-array2**.sl**	Testing: ForEach loop with and without index defined after declaration
test-arraylib.sl	Testing: Ensure functionality of array Lib’s work as intended
test-fib.sl	Testing: Recursive Function Calls
test-for1.sl	ForEach loop with and without index defined on array
test-map1.sl	ForEach loop with and without index defined on map
test-func1.sl	Testing: Function call and return value
test-global1.sl	Testing: Global Variable Assignment

test-hello.sl	Testing: Hello World
test-if1.sl	Testing: If-Else statement (followed by other if statements afterwards)
test-if2.sl	Testing: If-Else Statement and execution followed by regular statement
test-if3.sl	Testing: If-Else Statement and execution followed by return statement
test-if4.sl	Testing: If-Else Statement look for dangling else.
test-stringLib.sl	Testing: String Library Functions.
test-var1.sl	Testing: Variable Assignment
test-var2.sl	Testing: Variable Assignment and Declaration together
test-while1.sl	Testing: While loop
test-while2.sl	Testing: While loop with sequential statement following

Automation used in testing

A bash script (runtests) was written to automatically run each unit test, and to notify the user if any disparities with what was expected arose. For the code listing of the this script see below:

Main File:

```
#!/bin/bash

for i in $( ls *.out | awk '{print substr($1,0,(length($1)-3))}' ); do

  echo "Running" $i

  SCRIPTFILE=$i".sl"

  OUTFILE=$i".out"

  RESULTFILE=$i".result"

  ../shil < $SCRIPTFILE > $RESULTFILE
  if !(diff $RESULTFILE $OUTFILE >/dev/null); then
    echo "...failed"
    echo "RESULT:"
    echo "------"
    cat $RESULTFILE
    echo "------"
    echo
    echo "EXPECTED:"
    echo "--------"
    cat $OUTFILE
    echo "--------"
    echo
  fi
done
```

Helper File:

```
c_#load "extLib.cma";;
let tests = ref [];;
let addtest desc test expected =
  tests := (desc, test, expected) :: !tests;;
let cleartests() =
  tests := [];;

let rec runtestshelper = function
    [] -> [];
  | (desc, test, expected)::rest ->
    let _ = print_endline ("Running " ^ desc) in
    let result = test() in let _ =
      if result = expected
        then []
        else let _ =
          print_endline (desc ^ " failed.") in let _ =
          print_string "\tExpected: " in let _ =
          Std.print expected in let _ =
          print_string "\tActual: " in let _ =
          Std.print result in []
      in
    runtestshelper rest;;

let runtests() =
  let _ = runtestshelper (List.rev !tests) in
  print_endline "Done.";;
```

7. Lessons Learned

7.1. Most Important Learned

-Moses

Compiler creation to me initially seemed like a mystical and magical process that was not for the faint of heart but thanks to Dr. Edwards' breakdown the main thing that I learned was that the process seemed almost obvious. It was obvious in the way how one component simply passes a representation to the next, ultimately providing the desired target program.

Another lesson learned was that the communication of your team is essential in your success. If a member is having trouble completing a task they should speak up as soon as possible so that other can begin to reorganize the work. If this is done in a timely manner then the efficiency of your team will lead to a great experience.

-Binh

The most important lessons I learned were regarding group coordination and management. Establishing a code repository with version control, and especially establishing a unit testing architecture was a key benefit to us, as from that point on we avoided miscommunications between the ways we expected our coded pieces to interact. If I were to go back and do this again, I would furthermore require that members make changes to the repository in separate branches and submit to code review from at least one other member before committing changes to the main code tree. This would not only avoid some bugs early on, but would also increase general understanding of the entire code.

-ChunYai

I learned that communication is key in large-scaled team projects such as our SHIL programming language project. Especially when the structure of the language and the syntax is so variable to change at the very start of the project, people need to know these changes as soon as possible so they can make changes to their parts. For example, if one person was working on the parser and decided to change the syntax of a data structure declaration, he needs to let the person who is writing test code know about this as well.

Another lesson I learned is that never underestimate your work. We were lucky in that we realized a couple weeks before the deadline that we may have underestimated the work we needed to do and therefore accelerated our progress. Overall, this project has bettered my technical skills and team-working skills. It has been a great experience.

-Ian

I learned the importance of doing extensive research on what you're trying to accomplish. I originally had only a vague idea of my end goal, and so my approach was equally vague. I spent a lot of time working with libraries, only to find that these were not the libraries I needed to be using. Often I was reinventing the wheel, or inventing wheels that were entirely unnecessary. However, on the bright side of all of this, jumping into the project somewhat blind like this aided me in familiarizing myself with what I could and could not achieve within our timeframe.

I also learned the importance of communicating with the group. They were an invaluable resource for helping me debug and learn how to approach a project such as this one. A project on this scale would be very difficult without constant feedback, and cooperation between a group.

7.2. Advice For Future Teams

The best advice that team SHIL could provide is to meet early and often, and also the earlier you learn to embrace OCaml instead of resisting it the better off you will be in regards to creating the language. The language may seem initially touch. Especially if you are unaware of the nuisances of the functional programming, but once you get the hang of it you will be hooked and the flow of the coding will go much smother.

Be sure to document every step of the way, especially in regards to using a versioning control system, because if changes ever have to be rolled back then it will be clear as what must be done in order to achieve this.

-Team SHIL

8. Appendix

8.1. Complete Code Listing

Ast.mli

```
type binop = Add | Sub | Mult | Div | Equal | Neq | Less | Greater | Geq |
Leq | And | Or


type uniop = Not


type variable =

    IdVar of string


type lit =

    Literal_int of int

  | Literal_bool of bool

  | Literal_float of float

  | Literal_str of string

  | Literal_array of lit list

  | Literal_map of (lit*lit) list

  | Literal_html of (Nethtml.document)


type datatype =

    Type_int

  | Type_bool

  | Type_float

  | Type_str

  | Type_Array of datatype
  | Type_Map of datatype*datatype
  | Type_html

type expr =
    Literal of lit
  | Arrayref of variable * expr
  | Mapref of variable * expr
  | Arrayassign of variable * expr * expr
  | Mapassign of variable * expr * expr
  | Var of variable
  | Binop of expr * binop * expr
  | Uniop of uniop * expr
  | Assign of variable * expr
  | Call of string * expr list
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | Use of string
  | Break
  | If of expr * stmt * stmt
  | Foreach of string * string * expr * stmt
  | While of expr * stmt
  | Declare of datatype * string
  | DeclareAssign of datatype * string * expr
  | FuncDecl of string*((datatype*string) list)*datatype*stmt
```

```
type program = stmt list
```

interpret.ml

```
open Ast

open Http_client.Convenience

module StringMap = Map.Make (String);;


type symbol_table = {

  parent : symbol_table option;

  vars : lit StringMap.t;

  funcs : (((datatype*string) list)*stmt) StringMap.t;

}


type translation_environment = {

    return_type : datatype;     (* Function's return value *)

    in_loop : bool;             (* whether break and continue are valid *)

    scope : symbol_table;       (* symbol table for vars *)

  }


exception TypeError

exception ReturnCall of lit*symbol_table

exception BreakStatement of symbol_table

exception Fatal of string


(* Main entry point: run a program *)

let run (stmts) =
  (* initial symbol table *)
  let empty_symbol = {
    parent = None;
    vars = StringMap.empty;
    funcs = StringMap.empty;}
  in

  (* Perform variable lookup *)
  let rec findvar env = function
      IdVar(name) ->
        try (StringMap.find name env.vars)
          with Not_found -> match env.parent with
                                None -> raise Not_found
                              | Some p -> findvar p (IdVar name)
  in

  (* Perform a function lookup *)
  let rec findfunc env = function
      name ->
        try (StringMap.find name env.funcs)
          with Not_found -> match env.parent with
                                None -> raise Not_found
                              | Some p -> findfunc p name
  in

  (* find the scope of a var and set it *)
  let rec setvar env = function
      IdVar(name), v ->
```

```
      try (let _ = StringMap.find name env.vars in
           v, {parent = env.parent;
               vars = StringMap.add name v env.vars;
               funcs = env.funcs;})
        with Not_found -> match env.parent with
                              None -> raise Not_found
                            | Some p ->
                                let v, p = setvar p ((IdVar name), v) in
                                v, {parent = Some p;
                                    vars = env.vars;
                                    funcs = env.funcs;}
in

(* find the scope of an array and set an element *)
let rec sete i v l = (i, v) :: (List.remove_assoc i l) in
let rec setmap env = function
    IdVar(name), i, v ->
      (try (match StringMap.find name env.vars with
                Literal_map(old) ->
                  v, {parent = env.parent;
                      vars = StringMap.add name
                               (Literal_map (sete i v old)) env.vars;
                      funcs = env.funcs;}
              | _ -> raise TypeError)
         with Not_found -> match env.parent with
                               None -> raise Not_found
                             | Some p ->
                                 let v, p = setmap p
```

```
                                               ((IdVar name), i, v) in
                                 v, {parent = Some p;
                                     vars = env.vars;
                                     funcs = env.funcs;})
in

(* find the scope of a map and set an element *)
let rec setn i v = function
    hd :: rest ->
      if (i < 0) then (hd :: rest) else
        if (i = 0) then (v :: rest) else
          (hd :: (setn (i - 1) v rest))
  | [] -> []
in
let rec setarr env = function
    IdVar(name), Literal_int(i), v ->
      (try (match StringMap.find name env.vars with
                Literal_array(old) ->
                  v, {parent = env.parent;
                      vars = StringMap.add name
                               (Literal_array (setn i v old)) env.vars;
                      funcs = env.funcs;}
              | _ -> raise TypeError)
         with Not_found -> match env.parent with
                               None -> raise Not_found
                             | Some p ->
                                 let v, p = setarr p
                                               ((IdVar name),
                                               (Literal_int i),
                                               v) in
                                 v, {parent = Some p;
                                     vars = env.vars;
                                     funcs = env.funcs;})
  | _ -> raise TypeError
in

(* Check if a type and literal match *)
let rec check_type dtype lit =
  (match dtype,lit with
       Type_int,Literal_int(a) -> true
     | Type_bool,Literal_bool(a) -> true
     | Type_float,Literal_float(a) -> true
     | Type_str,Literal_str(a) -> true
     | Type_Array(dtype),
       Literal_array(hd::rest) -> check_type dtype hd
     | Type_Array(dtype), Literal_array([]) -> true
     | Type_Map(ktype, vtype),
       Literal_map((klit, vlit)::rest) ->
         ((check_type ktype klit) && (check_type vtype vlit))
     | Type_html,Literal_html(_) -> true
     | _ -> false)
in

(* Evaluate an expression and return (value, updated environment) *)
let rec eval env = function
    Literal(i) ->  i, env
```

```
| Arrayref(name, e) ->
    let v, env = eval env e in
    let arr = findvar env name in
    (match v, arr with
         Literal_int(i), Literal_array(l) -> List.nth l i
       | _ -> raise TypeError), env
| Mapref(name, e) ->
    let v, env = eval env e in
    let map = findvar env name in
    (match map with
         Literal_map(mapcontent) ->
          (List.assoc v mapcontent)
       | _ -> raise TypeError), env
| Arrayassign(name, e1, e2) ->
    let v1, env = eval env e1 in
    let v2, env = eval env e2 in
    setarr env (name, v1, v2)
| Mapassign(name, e1, e2) ->
    let v1, env = eval env e1 in
    let v2, env = eval env e2 in
    setmap env (name, v1, v2)
| Var(var) -> (findvar env var), env
| Binop(e1, op, e2) ->
    let l1, env = eval env e1 in
    let l2, env = eval env e2 in
    (match (l1, l2) with
      (Literal_int(v1), Literal_int(v2)) ->
        (match op with
          Add -> Literal_int(v1 + v2)
        | Sub -> Literal_int(v1 - v2)
        | Mult -> Literal_int(v1 * v2)
        | Div -> Literal_int(v1 / v2)
        | Equal -> Literal_bool(v1 = v2)
        | Neq -> Literal_bool(v1 != v2)
        | Less -> Literal_bool(v1 < v2)
        | Leq -> Literal_bool(v1 <= v2)
        | Greater -> Literal_bool(v1 > v2)
        | Geq -> Literal_bool(v1 >= v2)
        | _ -> raise TypeError)
    | (Literal_str(v1), Literal_str(v2)) ->
        (match op with
             Add -> Literal_str(v1 ^ v2)
           | Equal -> Literal_bool(v1 = v2)
           | Neq -> Literal_bool(v1 != v2)
           | _ -> raise TypeError)
    | _ -> raise TypeError), env
| Uniop(op, e) ->
    let l, env = eval env e in
      (match l with
           Literal_bool(v) ->
             (match op with
                Not -> Literal_bool(not v))
         | _ -> raise TypeError), env
| Assign(IdVar(varname), e) ->
    let v, env = eval env e in
    setvar env ((IdVar varname), v)
```

```
(* Start of Libraries *)

  (* String Library *)

    | Call("substring", [e1; e2; e3]) ->

        let v1, env = eval env e1 in

        let v2, env = eval env e2 in

        let v3, env = eval env e3 in

        (match [v1;v2;v3] with [Literal_str(v1); Literal_int(v2);
Literal_int(v3)]  -> (

               try

                 Literal_str( String.sub v1 v2 ((v3 - v2)) )

               with (Invalid_argument(z)) -> raise (Invalid_argument(z)) )

        | _ -> raise TypeError), env


    | Call("stringLength", [e1]) ->

        let v1, env = eval env e1 in

        (match v1 with Literal_str(v1)  ->

          Literal_int( String.length v1 );

        | _ -> raise TypeError), env


    | Call("stringFind", [e1;e2]) ->

        let v1, env = eval env e1 in

      let v2, env = eval env e2 in

        (match [v1;v2] with [Literal_str(v1); Literal_str(v2)]  ->

          Literal_int( try Str.search_forward (Str.regexp v2) v1 0 with
Not_found->0 );

        | _ -> raise TypeError), env


    | Call("stringReplace", [e1;e2;e3]) ->
    let v1, env = eval env e1 in
   let v2, env = eval env e2 in
   let v3, env = eval env e3 in
    (match [v1;v2;v3] with [Literal_str(v1);
Literal_str(v2);Literal_str(v3)]  ->
     Literal_str(Str.global_replace (Str.regexp v1) v2 v3);
    | _ -> raise TypeError), env

  | Call("stringToUpper", [e1]) ->
    let v1, env = eval env e1 in
    (match v1 with Literal_str(v1)  ->
     Literal_str( String.uppercase v1);
    | _ -> raise TypeError), env

  | Call("stringToLower", [e1]) ->
    let v1, env = eval env e1 in
    (match v1 with Literal_str(v1)  ->
     Literal_str( String.lowercase v1);
    | _ -> raise TypeError), env

  | Call("stringFindAll", [e1; e2]) ->
    let v1, env = eval env e1 in
    let v2, env = eval env e2 in
    (match [v1;v2] with
     [Literal_str(v1); Literal_str(v2)]  ->
        let rec stringFindAllHelper inputString searchString index
indexList =
          try
```

```
                      let currIndex = Str.search_forward (Str.regexp
searchString) inputString index in
                      let nextIndex = (currIndex + (String.length
searchString)) in
                      let currIndex2 = Literal_int(currIndex) in
                      (stringFindAllHelper inputString searchString nextIndex
(currIndex2::indexList))
                   with Not_found -> indexList;
               in
               Literal_array (List.rev(stringFindAllHelper v1 v2 0 []));
        | _ -> raise TypeError), env


    | Call("splitString", [e1;e2]) ->
        let v1, env = eval env e1 in
      let v2, env = eval env e2 in
        (match [v1;v2] with [Literal_str(v1); Literal_str(v2)]  ->
          Literal_array( List.map (fun f-> Literal_str f) (Str.split
(Str.regexp v1) v2) );
        | _ -> raise TypeError), env


  (* Array Library *)
      (* Works for array and maps *)
    | Call("length", [e1]) ->
        let v1, env = eval env e1 in
        (match v1 with
          Literal_array(v1) -> Literal_int(List.length v1)
        | Literal_map(v1)   -> Literal_int(List.length v1)
        | _ -> raise TypeError), env


      (* Maps are represented as (lit*lit) lists *)
    | Call("getKeys", [e1]) ->
        let v1, env = eval env e1 in
        (match v1 with Literal_map(v1)  ->
          Literal_array( List.rev( List.fold_left (fun l (a,b)->a::l) [] v1)
);
        | _ -> raise TypeError), env

    | Call("getValues", [e1]) ->
        let v1, env = eval env e1 in
        (match v1 with Literal_map(v1)  ->
          Literal_array( List.rev( List.fold_left (fun l (a,b)->b::l) [] v1)
);
        | _ -> raise TypeError), env

      (* Works for array and maps according to key, e2 true for ascending *)
    | Call("sort", [e1;e2]) ->
        let v1, env = eval env e1 in
      let v2, env = eval env e2 in
        (match [v1;v2] with
          [Literal_array(v1); Literal_bool(v2)] -> Literal_array( if v2 then
(List.sort compare v1)
                                                                        else
List.rev (List.sort compare v1) )
        | [Literal_map(v1);   Literal_bool(v2)] -> Literal_map ( let x =
List.sort (fun (a,b) (c,d) -> if a<c then -1 else

(if a=c then 0 else 1)) v1
                                                                 in
                                                                 if v2 then x
else List.rev x )
        | _ -> raise TypeError), env
```

```
  | Call("randomize", [e1]) ->
    let v1, env = eval env e1 in
    (match v1 with
     Literal_array(v1) -> let _ = Random.self_init() in (* DO NOT REMOVE, REQUIRED FOR RANDOM #s *)
                Literal_array( (List.sort (fun x y -> Random.int 2) v1) );
    | _ -> raise TypeError), env

  | Call("numargs", []) ->
    Literal_int(Array.length Sys.argv), env
  | Call("getarg", [e1]) ->
    let v1, env = eval env e1 in
    (match v1 with
      Literal_int(v1) -> Literal_str(Sys.argv.(v1))
     | _ -> raise TypeError), env

   (*  HTML Library   *)
  | Call("url", [e1]) ->
    let v1, env = eval env e1 in
    ( match v1 with Literal_str(v1) -> Literal_array ( List.map (fun f->Literal_html(f)) (Nethtml.parse_document (Lexing.from_string (http_get v1))))
    | _ -> raise TypeError), env

  | Call("filterArrayByAttribute", [e1;e2;e3]) ->
    let v1, env = eval env e1 in
    let v2, env = eval env e2 in
    let v3, env = eval env e3 in
```

```
        ( match [v1;v2;v3] with
[Literal_str(v1);Literal_str(v2);Literal_array(v3)] ->
          let filterByAttribute attr value attributeMap =
            if ( List.mem_assoc attr attributeMap)
             then if ((List.assoc attr attributeMap) = value)
               then [Literal_map(attributeMap)]
               else []
             else [] in
          let strippedMaps = (List.map (function Literal_map(x) -> x
                                               | _ -> raise TypeError) v3) in
          let filteredMaps = (List.fold_left
                               (fun retList am -> List.append retList
(filterByAttribute (Literal_str v1) (Literal_str v2) am)) [] strippedMaps) in
          Literal_array(filteredMaps)
        | _ -> raise TypeError), env


   | Call("getArrayByTag", [e1;e2]) ->
        let v1, env = eval env e1 in
        let v2, env = eval env e2 in
        (match [v1;v2] with [Literal_str(l1);Literal_array(l2)] ->
               let rec find tag = (function Nethtml.Element(content,a,c) ->
if (tag=content) then [(List.map (function (x,y) -> Literal_str(x),
Literal_str(y))a)] else List.concat(List.map (find tag) c)
                                          | Nethtml.Data(e) -> []) in
               let getListByTag tag docList = (List.concat (List.map (find
tag) docList)) in
               Literal_array(List.map (fun a -> Literal_map(a)) (getListByTag
l1 (List.map (function Literal_html(x) -> x | _ -> raise TypeError) l2)))
               | _ -> raise TypeError), env
```

```
(* End of Libraries *)

  | Call("print", [e]) ->
    let v, env = eval env e in
    (match v with
     Literal_int(intval) ->
      print_endline (string_of_int intval);
      Literal_int(0)
    | Literal_bool(boolval) ->
      print_endline (string_of_bool boolval);
      Literal_int(0)
    | Literal_float(floatval) ->
      print_endline (string_of_float floatval);
      Literal_int(0)
    | Literal_str(strval) ->
      let rec f s =
       try let x = String.index s '\\' in
         print_string((String.sub s 0 x)^(String.sub s (x+1) 1));
         f((String.sub s (x+2) ((String.length s)-(x+2))));
       with Not_found -> print_endline(s); in
      f(strval);
      Literal_int(0)
    | _ -> raise TypeError), env

 (* Print without ending line *)
| Call("inprint", [e]) ->
    let v, env = eval env e in
    (match v with
      Literal_int(intval) ->
        print_string (string_of_int intval);
        Literal_int(0)
    | Literal_bool(boolval) ->
        print_string (string_of_bool boolval);
        Literal_int(0)
    | Literal_float(floatval) ->
        print_string (string_of_float floatval);
        Literal_int(0)
    | Literal_str(strval) ->
        let rec f s =
          try let x = String.index s '\\' in
              print_string((String.sub s 0 x)^(String.sub s (x+1) 1));
              f((String.sub s (x+2) ((String.length s)-(x+2))));
          with Not_found -> print_string(s); in
        f(strval);
        Literal_int(0)
    | _ -> raise TypeError), env

| Call(func, args) ->
    let exp_args, body = findfunc env func in
    let actuals, env = List.fold_left
        (fun (actuals, env) arg ->
          let v, env = eval env arg in
            (v :: actuals, env)) ([], env) args in
    let topass = List.combine exp_args actuals in
    let newenv = {
      parent = Some env;
```

```
          vars = List.fold_left
                   (fun map ((_, argname), argval) ->
                     StringMap.add argname argval map) StringMap.empty
topass;
          funcs = StringMap.empty; } in
        let v, newenv = try Literal_int(0), (exec newenv body)
                          with ReturnCall(v,env) -> v, env in
        (match newenv.parent with
             Some p -> v, p
          | _ -> raise (Fatal "Function call destroyed environment"))
    | Noexpr -> Literal_int(0), env

  (* Execute a statement and return an updated environment *)
  and exec env = function
      Block(stmts) -> List.fold_left exec env stmts
    | Expr(e) -> let _, env = eval env e in env
    | Return(e) -> let v, env = eval env e in raise (ReturnCall(v,env))
    | Use(name) -> env
    | Break -> raise (BreakStatement env)
    | If(e, s1, s2) ->
        let v, env = eval env e in
        exec env (if v = Literal_bool(true) then s1 else s2)
    | Foreach(id1,id2,e,s) ->
        let v, env = eval env e in
        (match v with
             Literal_array(l) ->
               let size = List.length l in
               let rec loop env i = function
                   hd :: rest ->
```

```
                 let _, env = setvar env (IdVar(id1), Literal_int(i)) in
                 let _, env = setvar env (IdVar(id2), hd) in
                 if i >= size then
                   env
                 else
                   loop (exec env s) (i + 1) rest
             | [] -> env in
           (try (loop env 0 l) with BreakStatement(env) -> env)
       | Literal_map(l) ->
           let rec loop env = function
               (kval, vval) :: rest ->
                  let _, env = setvar env (IdVar(id1), kval) in
                  let _, env = setvar env (IdVar(id2), vval) in
                  loop (exec env s) rest
             | [] -> env in
           (try (loop env l) with BreakStatement(env) -> env)
       | _ -> raise TypeError)
| While(e, s) ->
    let rec loop env =
      let v, env = eval env e in
      if v = Literal_bool(true) then loop (exec env s) else env
    in (try (loop env) with BreakStatement(env) -> env)
| Declare(dtype, name) ->
    let rec makedefault = function
           Type_int -> Literal_int(0)
         | Type_bool -> Literal_bool(true)
         | Type_float -> Literal_float(0.0)
         | Type_str -> Literal_str("")
```

```
           | Type_Array(dtype) -> Literal_array([])
           | Type_Map(ktype, ltype) ->
              (Literal_map [])
           | Type_html -> Literal_html(Nethtml.Data("")) in
      let nv = (makedefault dtype) in
        {parent = env.parent;
         vars = StringMap.add name nv env.vars;
         funcs = env.funcs;}
  | DeclareAssign(dtype, name, e) ->
      let v, env = eval env e in
      if (check_type dtype v) then
        {parent = env.parent;
         vars = StringMap.add name v env.vars;
         funcs = env.funcs;}
      else
        raise TypeError
  | FuncDecl(name, args, rettype, body) ->
      {parent = env.parent;
       vars = env.vars;
       funcs = StringMap.add name (args, body) env.funcs;}
in
try (List.fold_left exec empty_symbol stmts)
  with
      TypeError ->
        print_endline "Error: type mismatch";
        empty_symbol
    | ReturnCall(_) ->
        print_endline "Error: return from outside function body";
```

```
          empty_symbol

      | Invalid_argument(_) ->

          print_endline "Invalid arguments";

          empty_symbol

      | Not_found ->

          print_endline "Variable does not exist";

          empty_symbol
```

parser.mly

```
%{ open Ast %}


%token SEMI LPAREN RPAREN LBRACE LARRAY RARRAY LMAP RMAP RBRACE COMMA

%token PLUS MINUS TIMES DIVIDE ASSIGN RASSIGN

%token EQ NEQ LT LEQ GT GEQ

%token NOT AND OR TRUE FALSE MAYBE

%token FUNCTION RETURN IF THEN ELSE FOREACH IN WHILE BREAK END USE BOOL INT
REAL STRING MAP ARRAY HTML

%token <int> LITERAL_INT

%token <float> LITERAL_REAL

%token <bool> LITERAL_BOOL

%token <string> ID

%token <string> LITERAL_STR

%token EOF


%nonassoc NOELSE

%nonassoc ELSE
```

```
%left NOT AND OR

%left ASSIGN

%left EQ NEQ

%left LT GT LEQ GEQ

%left PLUS MINUS

%left TIMES DIVIDE


%start program

%type <Ast.program> program


%%


program:

   stmt_list        { List.rev $1 }


stmt_list:

   /* nothing */  { [] }

 | stmt_list stmt { $2 :: $1 }


stmt:

    LBRACE stmt_list RBRACE                        { Block(List.rev $2) }

  | expr SEMI                                      { Expr($1) }

  | RETURN expr SEMI                               { Return($2) }

  | USE LITERAL_STR SEMI                           { Use($2) }

  | BREAK SEMI                                     { Break }

  | IF LPAREN expr RPAREN THEN stmt %prec NOELSE { If($3, $6, Block([])) }

  | IF LPAREN expr RPAREN THEN stmt ELSE stmt    { If($3, $6, $8) }

  | FOREACH LPAREN ID ID IN expr RPAREN stmt     { Foreach($3, $4, $6, $8) }
  | WHILE LPAREN expr RPAREN stmt                { While($3, $5) }
  | vartype ID SEMI                              { Declare($1,$2) }
  | vartype ID ASSIGN expr SEMI                  { DeclareAssign($1,$2,$4) }
  | FUNCTION ID LPAREN formals_opt RPAREN RASSIGN vartype stmt
       { FuncDecl($2, $4, $7, $8)  }


expr:
    LITERAL                       { Literal($1) }
  | varref LARRAY expr RARRAY ASSIGN expr     { Arrayassign($1,$3, $6) }
  | varref LMAP expr RMAP ASSIGN expr         { Mapassign($1, $3, $6) }
  | varref LARRAY expr RARRAY     { Arrayref($1,$3) }
  | varref LMAP expr RMAP         { Mapref($1, $3) }
  | varref                        { Var($1) }
  | expr PLUS   expr              { Binop($1, Add,   $3) }
  | expr MINUS  expr              { Binop($1, Sub,   $3) }
  | expr TIMES  expr              { Binop($1, Mult,  $3) }
  | expr DIVIDE expr              { Binop($1, Div,   $3) }
  | expr EQ     expr              { Binop($1, Equal, $3) }
  | expr NEQ    expr              { Binop($1, Neq,   $3) }
  | expr LT     expr              { Binop($1, Less,  $3) }
  | expr LEQ    expr              { Binop($1, Leq,   $3) }
  | expr GT     expr              { Binop($1, Greater,  $3) }
  | expr GEQ    expr              { Binop($1, Geq,   $3) }
  | expr AND    expr              { Binop($1, And,   $3) }
  | expr OR     expr              { Binop($1, Or,    $3) }
  | NOT expr                      { Uniop(Not, $2)       }
  | varref ASSIGN expr            { Assign($1, $3) }
  | ID LPAREN actuals_opt  RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN            { $2 }

LITERAL:
    LITERAL_INT    { Literal_int($1)   }
  | LITERAL_REAL   { Literal_float($1) }
  | LITERAL_BOOL   { Literal_bool($1)  }
  | LITERAL_STR    { Literal_str($1)   }
  | LITERAL_array                 { Literal_array($1) }
  | LITERAL_map                   { Literal_map($1)   }

LITERAL_array:
    ARRAY LBRACE RBRACE { [] }
  | ARRAY LBRACE array_list RBRACE {List.rev $3 }

array_list:
    LITERAL  { [$1] }
  | array_list COMMA LITERAL { $3 :: $1}

LITERAL_map:
    MAP LBRACE RBRACE { [] }
  | MAP LBRACE map_list RBRACE {List.rev $3 }

map_list:
    LITERAL RASSIGN LITERAL { [($1, $3)] }
  | map_list COMMA LITERAL RASSIGN LITERAL { ($3, $5) :: $1 }

primtype:
    INT    { Type_int }
  | BOOL   { Type_bool }
  | STRING { Type_str }
  | REAL   { Type_float }
  | HTML   { Type_html }

vartype:
    vartype LARRAY RARRAY    { Type_Array($1) }
  | vartype LMAP vartype RMAP            { Type_Map($1,$3) }
  | primtype { $1 }

varref:
    ID                           { IdVar($1) }

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

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

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

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

scanner.mll

```
{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }  (* Whitespace *)
| "/*"          { comment lexbuf }         (* Comments *)
| '('           { LPAREN }
| ')'           { RPAREN }
| '{'           { LBRACE }
| '}'           { RBRACE }
| "[["          { LMAP }
| "]]"          { RMAP }
| '['           { LARRAY }
| ']'           { RARRAY }
| ';'           { SEMI }
| ','           { COMMA }
| '+'           { PLUS }
| '-'           { MINUS }
| '*'           { TIMES }
| '/'           { DIVIDE }
| "<-"          { ASSIGN }
| "->"          { RASSIGN }
| '='           { EQ }
| "!="          { NEQ }
| '<'           { LT }
| '>'           { GT }
| "<="        { LEQ }
| ">="        { GEQ }
| '&'         { AND }
| '|'         { OR }
| '!'         { NOT }
| "if"        { IF }
| "then"      { THEN }
| "else"      { ELSE }
| "foreach"   { FOREACH }
| "while"     { WHILE }
| "return"    { RETURN }
| "break"     { BREAK }
| "end"       { END }
| "use"       { USE }
| "function"  { FUNCTION }
| "fun"       { FUNCTION }
| "boolean"   { BOOL }
| "integer"   { INT }
| "real"      { REAL }
| "string"    { STRING }
| "array"     { ARRAY }
| "map"       { MAP }
| "HtmlDoc"   { HTML }
| "in"        { IN }

| "TRUE"      { LITERAL_BOOL(true) }
| "FALSE"     { LITERAL_BOOL(false) }
| ['0'-'9']+'.'['0'-'9']* as lxm { LITERAL_REAL( float_of_string lxm )  }

| ['0'-'9']+ as lxm { LITERAL_INT(int_of_string lxm) }

| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }

| '"' (['a'-'z' 'A'-'Z' '0'-'9' '_' ' ' '+' '!' '@' '#' '$' '%' '^' '&' '*'
'(' ')' '-' '+' '=' '{' '[' '}' ']' '|' ';' ':' '<' ',' '.' '>' '/' '?' '`'
'~']|"\\\""|"\\\\"|"\\\'")*'"'

       as lxm { LITERAL_STR(String.sub lxm 1 ((String.length lxm)-2)) }

| eof { EOF }

| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }


and comment = parse

  "*/" { token lexbuf }

| _    { comment lexbuf }
```

shil.ml

```
let _ =

  let lexbuf = Lexing.from_channel stdin in

  let program = Parser.program Scanner.token lexbuf in

  ignore (Interpret.run program)
```