SHIL
Simulated Human Input Language

Moses Vaughan - mjv2123@columbia.edu
Binh Vo - bdv2112@columbia.edu
Ian Vo - idv2101@columbia.edu
Chun Yai Wang - cw2244@columbia.edu

Why SHIL?

- SHIL is a language used primarily for developing HTML based automated bots.
  - Provides the developer with tools to represent web interactions.
  - With these tools users can write applications that automate a variety of site interactions.
    - Manipulating look & feel of a given site.
    - Ability to write bots/spammers.
    - DataMining.

Perspectives

- User
  - SHIL can be used to implement command line user interfaces.
    - Useful for writing shell scripts that make use of website functionality .
- Server
  - SHIL can be used to simulate user interaction by with automating server interaction . Useful for many applications ranging from creating spiders to website test scripts.

Tutorial – The Basics I

- Comments: The usual /* and */ are used for all comments.
- Assignment: The arrow points the way.   ‘<-’

  -
    ```
    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:

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

Keywords

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

Operators

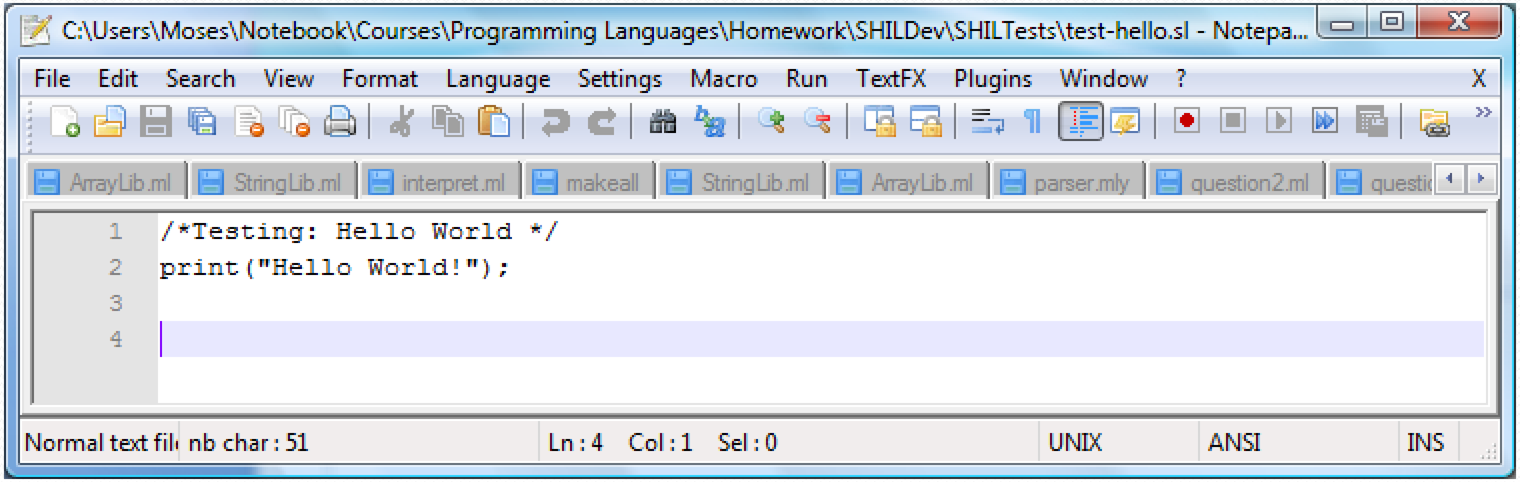
Lexeme	Usage
<-	assignment
+ - / *	math
"	string**
;	statement termination
.	struct reference
[ ]	array reference
( )	Logical grouping
& | ! = < > >=	Boolean Operators

Data Types

- SHIL uses the usual suspects.
  - Integer
  - Boolean
  - Map
  - Array
  - String
  - Real – Acts as a floating point number

“Hello World!” in SHIL

- Control Flow resembles that of a scripting language.
  - No main().
  - Hello World needs only one line of actual code!


C:\Users\Moses\Notebook\Courses\Programming Languages\Homework\SHILDev\SHILTests\test-hello.sl - Notepa...
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window ?
ArrayLib.ml
StringLib.ml
interpret.ml
makeall
StringLib.ml
ArrayLib.ml
parser.mly
question2.ml
questi
Normal text fil
nb char : 51
Ln : 4 Col : 1 Sel : 0
UNIX
ANSI
INS

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


```

Selection

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

Data Structures

- 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};
```

How SHIL was made

- Based structure on microC.
  - Scanner
  - Parser
  - AST
  - Interpreter
  - Test Library

AST

- Needed support for multiple types.
  - Represented by a specific Type “type”.
  - Along with a Literal type which unions our various types
- Also program is a list of statements, due to lack of main

Interpreter

- Variables and functions stored in separate lists (thus separate namespaces)
- Array and Map referencing
- Declares
- Expressions now take and return Literal instead of int
- Library functions added as overloads to call pattern
- Exceptions used both to return from functions and break from loops
- Type checking implemented here by comparing Type to Literal

Test library

- Ocaml test library that allows us to specify functions
  - Named tests
  - Expected results
  - Identify and localize problems to scanner or parser
- Script testing
  - Specification of scripts and expected prints
  - Automated comparison
  - Segmented according to specific functionalities in the interpreter

Lessons Learned

- Ocaml is a language that was well worth learning.
  - Superior typing system which makes polymorphic and type checking abilities the forefront of its implementation. The strictness of the compiler caught many bugs before they came to life in the code.
  - 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.
  - Offers new perspective on view of algorithms and analysis.
- Importance of Unit Testing for every minute component of the language.
  - Small functionality specific tests .

Lessons Learned

- Set modest goals, just do them well.
  - Good base is critical to success.
- Document every step of the way.
  - Ocamlnet.
- Don’t reinvent the wheel.
  - Research what’s available thoroughly before you begin coding.
  - Regex is not the solution to all problems

Lessons Learned

- Become familiar with you environment before you step into it.

- Meet early and often.
  - Regular meetings.
- Importance of version control systems.
  - Copy and Paste from IM doesn’t work well. ☺