PLT Final project - tabLe

Ciaran Beckford: cab2299@columbia.edu,
Selena Huang: sh3696@columbia.edu,
Yalon David Gordon: ydg2102@columbia.edu,
Joshua Dallas Hazlett: jdh2197@columbia.edu

May 15, 2020

Contents

1 Introduction

tabLe is an imperative programming language with the simplicity of Python-like syntax and the strict typing of Java. With an imperative paradigm, strongly typed system, statically scoped, strict evaluation order and features like Algebraic Data Types. tabLe also includes a built-in math library that makes calling operations intuitive to the user.
This programming language is oriented towards statisticians who have zero or limited programming experience. This combination of features will help minimize unwanted errors that might arise for someone new to programming. tabLe is a very simple language with high functionality for data dissection. The elegance of the language lies in the simple syntax and ready-made statistical functions.
tabLe is an imperative language, with strongly and statically typed, static scoping, and strict evaluation order semantics. tabLe features algebraic data types, and has a simple, built-in library of mathematical and statistics functions in order to streamline usage for new programmers.

2 Language Tutorial

This is a very basic language reference manual for tabLe.

2.1 Comments

Comments are handled using the tilde:
The comments are enclosed between two tildes, as follows:

```
1 ~ This is a comment. ~
```

2.2 Literals

Boolean literals are either `true` or `false`.
Integer literals are a sequence of digits, representing a number in base 10.
Floating-point literals consists of two sequences of digits separated by ., a decimal point. The sequence to the left of the decimal is the integral component, while the sequence to the right is the fractional component.
String Literals are any sequence of characters surrounded by double quotes (excluding the " double quotation character).
List literals are a series of floats within a list, in the format `[float, float, ...]`.

2.3 Key Words

These are special use words that cannot be used in variables or otherwise:
`true, false, if else, while return, int, float, bool, string, null, arrays, def`.

2.4 Data Types

In this language, we have booleans, integers, floats, strings, null and lists.

2.4.1 boolean

Booleans are of values either `true` or `false`.
They can be declared in either of the following ways:

```
bool foo;
bool foo = true;
```

2.4.2 int

Integers are of the 32-bit data type.
They can be declared in either of the following ways:

```
int foo;
int foo = 5;
```

2.4.3 float

Floats are actually a double type representation, to carry more information.
They can be declared in either of the following ways:

```
float foo;
float foo = 5.0;
```

2.4.4 string

String types is a type used to store and manipulate strings of ASCII characters.
They can be declared in either of the following ways:

```
string foo;
string foo = "hello";
```

2.4.5 List

Lists are a series of floats.
They can be declared in either of the following ways:

```
float arrays foo;
float arrays foo = [1.0, 2.0, 3.0];
```

2.4.6 null

The null type is an empty type, equivalent to the void type in C.

2.5 Statements

This language handles statements by separating them with a semicolon: `int foo; foo = 5`.
Assignment is used with an equal sign: `foo = 5`.
Conditionals use `if (statement) else (statement)` format.

Return statements include the keyword return and an expression that follows: `return expr;`

2.6 Operations

This language handles all basic mathematical operations, such as addition, subtraction, multiplication, division, and modulo.

2.6.1 Stat functions

tabLe also handles basic statistics functions on a float array, such as min, max, mean, variance, and standard deviation.

3 Architectural Design

The architecture of the tabLe compiler consists of multiple phases which coverts the .tabLe file into a binary executable file.

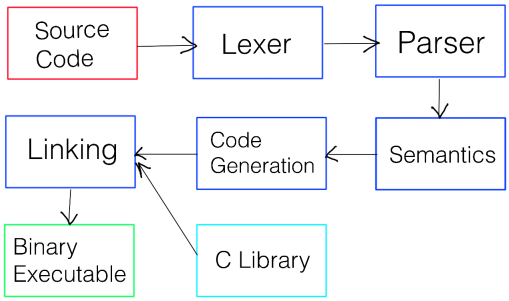

Figure 1: Architectural design diagram

The scanner scans a stream of ASCII text and generates tokens based on the input and the valid characters in our language.
The parser constructs an abstract syntax tree (AST) based on the stream of tokens created by the scanner, and will check whether the program is valid syntactically, sending an error if it is not correct.
The semantics analysis enforces the sematic rules of tabLe. It checks for variable declaration before use and are statically typed, sending an error if not defined correctly.
At the end, code generation creates the program and links the C libraries into an executable binary code.
All parts were worked on by Ciaran and Selena. Each file corresponds to a section, and work was distributed by topic, not by architectural design.

4 Test Plan

Shortened versions of our test cases are as follows:

test_stats.tabLe

```
def int main<>
{
  float mean_res;
  float stdev_res;

  float arrays x;
  x = [2.0, 4.0, 6.22, 4.5, 6.0];

  mean_res = mean<x>;
  prints<"Mean: ">;
  printf<mean_res>;

  stdev_res = stdev<x>;
  prints<"stdev: ">;
  printf<stdev_res>;

  return 0;
}
```

test_arithmetic.tabLe

```
def int main<>
{
  int x;
  int y;

  x=10;
  y=3;

  prints<"Variable x:">;
  print<x>;
  prints<"Variable y:">;
  print<y>;
  prints<"Addition (x + y):">;
  print<x + y>;

  return 0;
 }
```

4.1 Software Development Environment

The following describes the development environment used
Libraries and Language: Across the different computers, we used OCaml version 4.08.1, Ocamllex version 4.08.1, Clang version 11.0.3 and Clang version 6.0.
Version control: All project related files were managed with a git repository on GitHub.
Text Editing: Sublime Text and Atom.
Operating System: Across the different computers, we used: OSX 10.15.4 and

Windows OS 10 with a Debian Linux subsystem.

5 Summary

5.1 Role assignments

For this assignment, we were all originally assigned different sections to complete. However, in the end, it ended up being mixed up. The language proposal was worked on by all of us. Ciaran implemented the floats, strings and arrays, as well as the string concatenation. Selena wrote up and linked the the statistics functions in C, also worked on the arrays, and wrote the test suites and the LRM.

5.2 Ciaran

I found that starting small and slowly building up that pyramid one feature at a time was really helpful. Also, doing vertical slices of implementation was a lot easier than attempting everything at once.

5.3 Selena

Not unexpectedly, I realized that starting early makes a huge difference, especially in a language I am not particularly fluent in. In addition, I learned to be patient, especially because not all the effort that I put into the assignment ended up working, and I had to scrap it and start over.

5.4 Advice

Start early and start small! Every step counts!