University of Glasgow | School of Computing Science

Design and implement your own programming language

Kyle Simpson
Kristiyan Dimitrov
Darren Findlay
David Creigh
Gerard Docherty

Level 3 Project — 27 March 2015

Abstract

The abstract goes here

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

Chapter 2

Language Tutorial

This chapter comprises of a brief introduction of how to use our language.

2.1 1.1 Getting Started

In every language, the first program to write is always 'hello world' - where you would print the words "hello world". However, as this language is mainly a graphical language, the equivalent would be 'hello square', and the aim is to draw a square. In MLang, the program required to draw a square is:

```
#EXAMPLE 1.1
Update(){
        Polygon square = ({0,0} + {5,0}) * 4}
        draw(square)}
}
```

to run this program on machines running a UNIX operating system, you must create a file with the the suffix ".m" **Need compiling commands**

Any MLang program you write will have to consist of at least one function, and variables. These functions and variables can be named anything you like. You can call other functions to help carry out the task, only ones that you have written, as there are no libraries provided as such.

In Mlang, implicit semi-colons exist at the end of every line. This means that, you dont have to end each line wih a semi-colon, and whether you do or not will not have an effect on the compilation of the program.

One way in which you can pass data between functions is by including variables in the calling statement as arguments. However, you can only do this if the function that is being called is expecting the same number and type of variables. An example is:

```
#EXAMPLE 1.2
main(){
        num n = 2;
        takesParams(num n){
                print(n);
        }
        takesParams(n);
}
```

Here, you see that the calling statement - *takesParams(n);* - provides the correct number and type of arguments. If there was another argument included, for example *takesParams(n, 7)*, or the wrong types, then it would not compile. This will be talked about in more detail in 1.7. As you can see in example 1.1, the code inside of a function is enclosed by curly braces . The statement draw takes in one argument - either a Line, Point or Polygon, and draws it to the canvas.

newlines?

2.2 1.2 Variables and arithmetics operators

This next section will use a more complicated program than before, introducing more features, such as comments, loops, and exapnd on previously touched on features, such as variables.

```
#EXAMPLE 1.3
#This Program will draw 3 different shapes - triangle, square and pentagon.
main(){
        num sides = 5;
        num counter = 3;
        Point pt1 = {3,1}
        pt2 = {1,3}
        Line l1 = (pt1 + pt2);
        while(counter <= sides){
                Polygon shape = 1 * counter;
                draw(shape);
        }
}
```

The initial lines indicate how to show comments in your code in MLang. Using a hashsign will be ignored by the compiler, and show that the current line is a comment. These can be used to explain how your programs works, and make it easier to read and understand, for you or other users.

Chapter 3

Language Reference Manual

Chapter 4

Project Plan

Chapter 5

Language Evolution

Chapter 6

Compiler Architecture

Chapter 7

Development Environment

Chapter 8

Test Plan and test suites

Chapter 9

Conclusion

9.1 Contributions

Here we explain that Lewis Carroll wrote chapter John Wayne was out riding his horse every day and didn't do anything. Marilyn Monroe was great at getting the requirements specification and coordinating the writing of the report. Betty Davis did the coding of the kernel of the project, described in Chapter **??**. James Dean handled the multimedia content of the project.

Chapter 10

Appendix A

Includes full source listing of compiler