0. Table of Contents

1. Introduction

1.1. Overview

My project, Hopeful - A First Programming Language, will involve the development of a programming language. This language will be aimed at beginning programmers in a university setting. It will have a simple and clean syntax that will allow students to focus on the fundamentals of programming, instead of a complicated syntax. Teaching programming to beginners can be a very difficult task. This project aims to create a programming language that will both help lecturers to teach and students to learn.

The language will be used through a compiler, and then executed on the command line. The language and compiler will be constructed in JavaCC and LLVM.

Since this project is more research orientated, this Functional Specification will follow a format more suited to a project such as this. So, this means that instead details on System Architecture and High Level Design, this Functional Specification will include my research and evaluation methodology. The research and evaluation will be carried on Dublin City University students and staff. This will allow me to tailor my language to suit the needs of this university. The most crucial part of this project is the evaluation because it shows the success of this project in its goal of being a good first programming language.

1.2. Glossary

1.2.1. Integrated Development Environment (IDE):

An integrated development environment (IDE) is a software application that provides facilities to programmers for software development.

1.2.2. JavaCC:

JavaCC is a lexer and parser generator. It is a tool that reads a grammar specification and converts it to a Java programme that can recognize matches to the grammar.

1.2.3. LLVM:

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies.

2. General Description

2.1. Product / System Functions

This programming language will allow the user to write the source code in an editor of their choice and save the file with the appropriate extension. Then, the user will be able to compile the programme on the command line. Finally, the user will be run the executable code.

2.2. User Characteristics and Objectives

This programming language is aimed at students in a university setting who are beginning to learn how to programme. It is particularly aimed at people who have little to no previous experience in programming. This language is a tool to help people learn the very basic fundamentals of programming and allows them to create a good groundwork to continue to programme using other languages.

From a teaching perspective, the language should cover basics and topics such as strings, integers, Booleans, arrays/lists, and so forth. From a learning perspective, the language should have a clear syntax that is easy to understand at a glance to allow the student to understand key programming concepts sooner.

2.3. Operational Scenarios

2.3.1. Compile Programme

The user, on the command line, can compile the source code.

2.3.2. Execute Programme

The executable, created by the compiler, can be run by the user and will print onto the command line if dictated by the programme.

2.3.3. Print to Terminal

When the programme is run, if it contains an executed print statement, then the appropriate text will be displayed on the command line.

2.3.4. Variables

Within the programme, the user is able to create variables of different types which are capable of storing data.

2.3.5. Types

Within the scope of this language, a type is an attribute of data that tells the compiler how the user intends to use it. Below are the types that will be a part of this language.

2.3.5.1. Integer

Within the scope of this language, an Integer is a whole number, that can be positive, negative, or zero. Examples include *0*, *25*, and *-349*.

2.3.5.2. String

Within the scope of this language, a String is a sequence of characters. Examples include *“cat”* and *“This is a sentence.”*.

2.3.5.3. Boolean

Within the scope of this language, a Boolean is a data type with two possible values, *True* and *False*.

2.3.5.4. Double/Float

Within the scope of this language, a Double/Float is a data type that holds numerical values that can be whole or fractional. Examples include *3.5* and *10.0*.

2.3.6. Condition Statements

The user can use condition Statements to execute different statement blocks, depending on an expression that will evaluate to a Boolean.

2.3.7. Control Flow

The user can use control flows to continuously execute a statement block as long as an expression that will evaluate to a Boolean remains True.

2.3.8. Methods

A method is a named section of a programme that performs a specific task and returns some value. The user can use methods to perform that certain task multiple times on different data.

2.3.9. Error Handling

At compile/run time, if there exists a bug within the user’s programme, the user will be alerted to this issue. The user may also be given warnings, which may not cause issues when the programme is executed, but something that should be noted and avoided. The user can use these errors to fix bugs in their code.

2.3.10. Input from Terminal

When the programme dictates, the user will prompt to type text into the command line. The text the user inputs can be used within the programme itself.

2.4. Constraints

2.4.1. Time

As with any project with a deadline, there is a time constraint. The project has the potential to scale into a fully developed programming language but for the project, I want to implement the basic functionality of a beginner's programming language before the deadline in May 2019 and leave enough time for plentiful user evaluation. This will require myself to plan and follow a feasible schedule and be able to adapt to any problems I encounter.

2.4.2. Research and Evaluation

As a major component of my project is based on research and evaluation, I may encounter the problems which are common in these subject areas. Issues like untruthful answers in surveys will be solved by asking the same question in two different ways as to ensure truthfulness. Leading will be abolished by carefully checking each question and judging what is being asked.

3. Functional Requirements

The functional requirements of this project are the same as the functional requirements of any compiler, and so can be grouped into the following stages.

3.1. Lexical Analysis

3.1.1. **Description:** Lexical Analysis is the process of converting a stream of characters from a programme into a stream of tokens that recognisable keywords, identifiers, punctuation within my language.

3.1.2. **Criticality:** Very high, as this is the first and very vital stage.

3.1.3. **Technical Issues:** There are none apparent at this time.

3.1.4. **Dependencies on Other Requirements:** None, as this is the first stage.

3.2. Syntax Analysis

3.2.1. **Description:** Syntax Analysis is the process of combining the tokens generated by Lexical Analysis into a valid combination. A grammar is a set of rules on how these tokens can be combined.

3.2.2. **Criticality:** Very high as errors here may lead to issues when the programme is executed.

3.2.3. **Technical Issues:** There are none apparent at this time.

3.2.4. **Dependencies on Other Requirements:** Lexical Analysis as that phase generates the tokens needed.

3.3. Semantic Analysis

3.3.1. **Description:** Semantic Analysis is the process of checking the source programme for semantic errors, such type inconsistencies, and gathers type information for Intermediate Code Generation.

3.3.2. **Criticality:** Very high as it is needed for Intermediate Code Generation.

3.3.3. **Technical Issues:** There are none apparent at this time.

3.3.4. **Dependencies on Other Requirements:** Syntax Analysis as both phases are checking for errors within the source programme. The programme needs to be syntactically correct before it can be checked for semantic errors.

3.4. Intermediate Code Generation

3.4.1. **Description:** Intermediate Code Generation is the process of creating abstract machine code that does not rely on a specific target machine for registers and memory locations.

3.4.2. **Criticality:** Very high as the Intermediate Code is what’s created by the compiler.

3.4.3. **Technical Issues:** There are none apparent at this time.

3.4.4. **Dependencies on Other Requirements:** Semantic Analysis as this gathers the type information for this phase.

3.5. Code Optimisation

3.5.1. **Description:** Code Optimisation is the process of improving intermediate code to become more efficient, although it is an optional phase.

3.5.2. **Criticality:** Low as this phase is optional.

3.5.3. Technical Issues: There are none apparent at this time.

3.5.4. Dependencies on Other Requirements: None.

3.6. Code Generation

3.6.1. Description: Code Generation is the process of translating intermediate code into object code, allocating memory locations, and selecting registers.

3.6.2. Criticality: Very high as this phase is necessary to run the source program.

3.6.3. Technical Issues: There are none apparent at this time.

3.6.4. Dependencies on Other Requirements: Intermediate Code Generation as the Intermediate Code is used to generate the final code.

4. Research

As this project is more of a research-based project, plenty of research will be carried out before the implementation of the programming language. As this language is aimed at a university setting, research will be carried out on both staff and students within Dublin City University.

Research will be carried out in the form of surveys. As for students, I have broken down the wider group into three demographics. These demographics are *beginner* programmers, meaning first and second year Computer Applications students, *advanced* programmers, meaning third and fourth year Computer Applications students, and finally *engineering* students which will comprise of second year and above Electronic and Computer Engineering and Mechatronic Engineering students, and second year Mechanical Engineering and Biomedical Engineering students. Engineering students have been included in my research to given some diversity to my research groups. Since Engineering students learn programming differently and use different languages to Computer Applications students, I believe that their input will be valuable to my research as they will have a different perspective on problems facing beginning programmers.

As for the approach to each survey, I considered each demographic in mind. For the *beginner* survey, for example, I gave code snippets and asked which code snippet they prefered. Each code snippet contained different versions of notation. For example, one code snippet used curly braces to indicate control structures while the other used indentation. Questions in the *beginner* survey followed such a format because beginner programmers would likely be unfamiliar with such concepts as static and dynamic data structures, and the discussion of arrays versus lists. A similar format is followed within the *engineering* survey, as there are many varying skill levels within each year and branch of engineering.

For the *advanced* survey, while the survey still contained code snippets to compare, the main focus was more in depth questions about the working of a programming language. Questions about static versus dynamic typing, the benefits of a text editor versus an IDE and a question about how that person's first programming language helped them to learn others. The final question is important because part of the goal of a first programming language is to develop a basic programming skill set, consisting of the fundamentals of programming, that can easily be transferred to using other languages.

Finally, for the *staff* survey, the focus instead moved onto the students. The questions were centered on how do the students learn best. Examples include do students learn better with dynamic or static typing, the use of an IDE versus a text editor. Other questions include what do students struggle with when learning to programme. A key question was, in that staff member's opinion, what first programming language has worked best in the past and why. This question will give a clear insight into what students respond best to, and should, hopefully, the best starting point as I create my own programming language.

Once all of the surveys have been filled out, I shall analyse the results and create a programming language that will best help the students to learn and the staff to teach.

5. Evaluation

As this project is research orientated and is aimed for use in a university setting, proper evaluation will need to be carried out to gauge whether the implementation of my programming language was successful or not.

For the evaluation process, I shall test my programming languages on a sample group of students. These students will mostly comprise of Computer Applications students, focusing on first and second year students. I shall also be testing with Engineering students and students from other faculties. I will be evaluating with students from other faculties to diversify my sample group. Also, many students from other faculties may learn how to programme during their time at university, so this ensures that my programming language is suitable for all beginning programmers, not just those from an Engineering or Computer Science background.

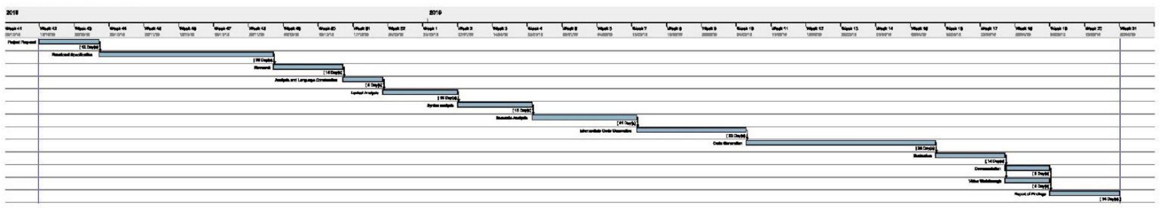
As part of the evaluation, the sample group shall be asked to write a small programme in both my language and another language of my choice, for example, Java or Python. The sample group shall also be given notes on both languages as a guide, this accommodates for people unfamiliar with the second language of my choice. This allows a more even judgement of both languages. The sample exercises will cover the most basic features of both languages. This will allow the sample group to compare the languages more easily and precisely.

After writing both programmes, the sample group shall be asked to fill out an evaluation sheet which will compare both languages. I will be focusing on how easy was my language to understand, which language was easier to read at a glance.

The evaluation will serve as a test to show how effective my language was compared to a common beginner's language. In many ways, the evaluation is the key part of this project. Both the research and implementation of the language build up to the evaluation. Along with a complete programming language, the main deliverable of this project will be a report detailing my findings, both from the research and evaluation.

6. Preliminary Schedule

Below is the Gantt chart of the provisional timeline of my project. This may be altered as I begin the project, as I cannot foresee what problems may occur in the research, development, implementation, and evaluation processes.



7. Appendices

Resources

- https://javacc.org/ - JavaCC Website

- https://llvm.org/ - LLVM Website