Project Name	Hopeful - A First Programming Language
Author	Gillian Mullen
Supervisor	David Sinclair
Document Title	Technical Guide
Summary	This project, Hopeful - A First Programming Language, will involve the development of a programming language. This language will be aimed at beginning programmers in an undergraduate 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.

Motivation

The idea for this project came to me earlier in my degree. In the second year of Computer Applications, students begin to learn their second programming language. For my class, we changed from Python to Java. Python, which looks similar to pseudocode, was well received by my class initially for its simple syntax and dynamic variable typing. However, with our first language being Python, this made learning Java for the first time difficult. This meant we spent time learning more than the basic syntax to program in Java. For example, we had to learn about having class in files and static variable typing. From this came the idea of creating a programming language that created a good foundation for beginner programmers that allows them to progress to more complex languages more easily.

Research

Since this is a research project, much research was done before I began to implement my programming language. From the outset, my intent was to create a language that had a clear and simple syntax that allows the students to focus on the fundamentals of programming. To help myself achieve this end, I surveyed Engineering and Computing students, as well as School of Computing staff. I believe that this survey provided a very good base to create my language upon, as well as backing up my belief that this language needs a clear and simple syntax to succeed in helping to teach students programming.

In the process of this research, four sample groups were taken into account. First is beginning programmers, accounting for first-year Computer Applications students. Secondly is advanced programmers, accounting for fourth-year Computer Applications students. Thirdly is the School of Computing staff. Lastly is Engineering students, second-year and up for all Engineering disciplines. In analysing these results, I weighed the results from beginning programmers and staff more than the other groups. I did this because I believe that these groups will give me a better insight into what beginning programmers need to learn better, while the other groups would be weighed more by personal bias.

Design

Since I was creating a programming language, the design element of my project consisted of the design of my language. The features and aesthetic of my language were decided on following the outcome of my survey, as mentioned previously. The rest of the design was decided upon by myself. The full design of the language can be seen in the Language Definition document. The language itself is contained in a *jar* along with its compiler, then the *jar* also allows the program to be executed.

Implementation

Hopeful and its compiler were implemented using JavaCC and LLVM. Like any compiler, Hopeful's compiler followed the phases: Lexical Analysis, Syntax Analysis, Semantic Analysis, and Code Generation.

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

Sample Code

Language definition of declaration:

```
void declaration() : { String type; String id; }
{
    type = type() id = lhs_identifier() ( <EQUALS> expression() | {} ) <SEMIC> { if(type.equals("int")) { type = "integer"; } st.insert(scope, id, type, "VAR"); }
}
```

Semantic check to see if a variable has been declared before use:

```
public Object visit (ASTlhs_identifier node, Object data) {
    SimpleNode parent = (SimpleNode) node.jjtGetParent();
    String type = parent.toString();
    String value = (String) node.value;
    if(type.equals("assignment")) {
      if(!st.lookup(value, scope)) {
        declaredBeforeUse = false;
        System.out.println("Fail: " + value + " Not Declared Before Use!");
      }
    }
    return DataType.Id;
  }
```

Entering a new variable into the Symbol Table:

```
public void insert(String scope, String id, String type, String declType) {
    if (checkScope(scope) == false) {
      createNewScope(scope);
    }
    LinkedList<String> currentScope = st.get(scope);
    currentScope.add(id);
    String key = id + scope;
    types.put(key, type);
    declTypes.put(id, declType);
  }
```

Writing IR code for a variable:

```
  public Object visit(ASTrhs_identifier node, Object data) {
```

```
    Context context = (Context) data;
    BufferedWriter buffer = context.buffer;

    String id = (String) node.value;
    String type;

    if(symbolTable.getDeclType(id).equals("FUNC")) {
      return id;
    }
    else {
      type = symbolTable.getSymbol(scope, id);
    }

    String tmp = getTmp();
    String command = tmp + " = " + "load " + type + ", " + type + "* %.p." + id;
    registerTypes.put(tmp,type);

    try {
      buffer.write(command);
      buffer.newLine();
    }
    catch(IOException e) {
      System.out.println("Failed to write IR code for RHS identifier to file");
      e.printStackTrace(System.out);
    }

    return tmp;
  }
```

Problems Solved

Hopeful offers a simple first programming language that allows beginners to focus on learning how to program, instead of complex syntax. While Hopeful is not the only solution to this problem, it is offered as an alternative which may aligned more closely with programming circulums than other traditional programming languages.

Results

Since this project is research based and a programming language for beginners, the language needed to be tested by programmers to determine the outcome of the project. To help myself achieve this end, I create an evaluation scenario involving Engineering and Computing students. I structured the evaluation so that I would gain a good idea of my progress of creating this language, and how successful I have been thus far.

Overall, the results were positive. I believe that the overall painted here by the evaluation is a positive picture of Hopeful as a programming language for beginners. Thus, I

believe this project achieved its goal of being a simple language that allows beginners to focus on learning how to program, instead of complicated syntax.

Future Work

As for future work, since this project has a limited scope and time frame, these directions would be for someone who intended to continue with the project. The main goal would be to implement more features in the language, such as arrays. However, implementing features that would help the student to learn would also be part of the goal. For example, making indentation required in the language would teach the student to create readable code. Most of this work would be taking suggestions from the evaluation that could not be implemented by myself, such as for loops and converting between types.