Table of Contents

Introduction

This is the manual document of Basic Programming Language that we have developed. This document will help you to know the syntax and run our language. The code for this language can be written in an input session and run in its own UI based on Java. Errors are displayed in the output console with color coded strings, i.e., red for error and black for normal output.

Overview

JFlex is a lexical analyzer generator (also known as scanner generator) for Java, written in Java.

A lexical analyzer generator takes as input a specification with a set of regular expressions and corresponding actions. It generates a program (a lexer) that reads input, matches the input against the regular expressions in the spec file, and runs the corresponding action if a regular expression matched. Lexer usually are the first front-end step in compilers, matching keywords, comments, operators, etc., and generating an input token stream for parsers. Lexers can also be used for many other purposes.

JFlex supports JDK 1.8 or above for build and JDK 7 and above for run-time.

JFlex Lexers are based on deterministic finite automata (DFAs). They are fast, without expensive backtracking. A standard tool for development of lexical analyzers is **JFlex**.

JFlex is designed to work together with the LALR parser generator CUP, and the Java modification. It can also be used together with other parser generators like ANTLR (ANother Tool for Language Recognition) or as a standalone tool.

A JFlex program consists of three parts:

- ✓ **User code**: is copied verbatim into the beginning of the Java source file of the generated lexer. This is the place for *package* declaration and *import* statements.
- ✓ **Options**: customize the generated lexer, declare constants, variables, regular definitions
- ✓ **Translation rules**: have the form *p { action }*

  where *p* is a regular expression and action is Java code specifying what the lexer executes when a sequence of characters of the input string matches *p*.

The lexical analyzer created by Lexer works as follows:

When activated, the lexical analyzer reads the remaining input one character at a time, until it has found the longest prefix that is matched by one of the regular expressions on the left-hand side of the translation rules.

Objective

This project was done as a partial fulfilment of course Programming Languages and Compilers (AT70.07). The main objective of this project is to create a simple programming language that supports assignment statement, if then else, while do, Sequential that can be run sequentially and type checking and its execution. The type includes int, real, char and Boolean/String.

Setting up to use the language.

The Ubuntu desktop is easy to use, easy to install and includes everything we need to run our work. It is also open source, secure, accessible, and free to download.

Requirements

We need to consider the following before starting the installation:

- Ensure you have at least 25 GB of free storage space, or 5 GB for a minimal installation.
- Have access to either a DVD or a USB flash drive containing the version of Ubuntu you want to install.
- Make sure we have a recent backup of our data. While it's unlikely that anything will go wrong.

Installation process

1. Once we have minimum hardware and software requirements follow the link given below:

https://ubuntu.com/tutorials/install-ubuntu-desktop#5-prepare-to-install-ubuntu

2. If we want to install Ubuntu on Virtual Box, then follow the link given below:

https://brb.nci.nih.gov/seqtools/installUbuntu.html

3. Once we have installed Ubuntu in our system then we need to install NetBean as it is open-source Integrated Development Environment (IDE).

For this assignment we have install the Ubuntu on Virtual Box since I can manage two Operating System and installation was based on Ubuntu 20.04.

Two more steps to follow to install NetBean:

1. Install openjdk 1.8.0 (https://openjdk.java.net/install/ - This link will help us to understand the process)

   sudo apt-get install openjdk-8-jdk

2. Install Netbean 8.0.2 (https://download.netbeans.org/netbeans/8.0.2/final/ - This link helps us to use the NetBean and how it is configured)

   we used Java EE and run the installation script.

Since now there are new version of NetBeans and it's not compatible with our cup file when I tried. Therefore, I would suggest using the old version of NetBeans which we can get from following link: https://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html

Even for Window OS or Mac OS, this project can be executed by using NetBeans 8.2. NetBeans can be used for Windows, MacOS and Ubuntu with following process:

Installing NetBeans 8.2:

1. Installing NetBeans on Windows
   - To be able to use NetBeans for Java programming, we need to first install Java Development Kit (JDK)
   - Download "NetBeans IDE" installer from https://netbeans-ide.informer.com/8.2/
   - Run the downloaded installer.
2. Installing NetBeans on MacOS
   - Download NetBeans from https://macdownload.informer.com/netbeans1/12.3/
   - Open the download Disk Image file.
   - Choose "netbeans 12.3.mpkg" and follow the instructions to install NetBeans.
   - Eject the Disk Image.
3. Installing NetBeans on Ubuntu
   - Download NetBeans from https://netbeans.apache.org/download/index.html. Select platform "Linux (x86/x64)"
   - Set the download sh file to executable and run the sh file by using these commands from Terminal and follow the instructions

```
$ cd ~/Downloads
$ chmod a+x netbeans-7.x-ml-javase-linux.sh   // Set to executable for all (a+x)
$ ./netbeans-7.x-ml-javase-linux.sh           // Run
```

   - To start NetBeans, run the script "netbeans" in the NetBeans'bin directory:

```
$ cd netbeans-bin-directory
$ ./netbeans
```

About Our Basic Programming Language

The main intention of this project is to create a small programming language that supports primitive types, arithmetic expressions, conditionals, and loops, and to know the key compiler concepts like lexical analyzers, semantic analysis, and parse trees.

Tools and Methods

The following table briefly shows the software and tools used for developing this language.

Name	**Description**
JFlex	A lexical analyzer generator tool based on Java
CUP	A LALR parser generator that works together with JFlex
NetBeans	IDE for Java development
Java Swing	Framework used to build GUI for the application

Language Specifications

Terminal Symbols

These are the terminal symbols used by the language as follows:

PLUS	MINUS	TIMES	DIVIDE	MOD	SEMI	COMMA	EQUALS
LPAREN	RPAREN	ET	NET	LT	LTE	GT	GTE
AND	OR	INT_LITERAL	FLOAT_LITERAL	BOL	STRING_LITERAL	IF	ENDIF
ELSE	WHILE	BEGIN	END	PRINT	PRINTLN	INT	FLOAT
BOOL	STRING	VAR					

Non-Terminal Symbols

These are the non-terminal symbols used by the language:

program	declarations	declaration	statements	statement	type	assignment
ifelse	while	print	println	variables	expr	term
factor	relop					

Language Grammar

```
program := declarations statements | statements;
declarations := declarations declaration | declaration;
declaration := type variables SEMI;
variables := variables COMMA VAR | VAR;
statements := statements statement | statement;
statement := assignment SEMI | ifelse | while | print SEMI
                | println SEMI | BEGIN statements END;
ifelse := IF LPAREN expr RPAREN statement ENDIF
          | IF LPAREN expr RPAREN statement ELSE statement ENDIF;
while := WHILE LPAREN expr RPAREN statement;
print := PRINT LPAREN expr RPAREN;
println := PRINTLN LPAREN RPAREN;
type := INT | FLOAT | STRING | BOOL;
assignment := type VAR EQUALS expr | VAR EQUALS expr;
expr := expr PLUS factor | expr MINUS factor | factor | replop;
factor := factor TIMES term | factor DIVIDE term
          | factor MOD term | term;
relop := term AND term | term OR term | term LT term | term LTE term   | term GT term
          | term GTE term | term ET term | term NET term;
term := LPAREN expr RPAREN | INT_LITERAL | FLOAT_LITERAL
         | STRING_LITERAL | VAR | BOL;
```

Basic Syntax

Tokens

Our Language consists of various tokens. Some tokens are keyword, variable, string literal, int literal, float literal, bool literal, symbol.

For example, the following statement consists of five tokens:

print ('Hello World!');

The individual tokens are:

1. *print*
2. *(*
3. *'Hello World!'*
4. *)*
5. *;*

All the tokens that our language supports are as follows:

+	-	*	/	%	;	,	=
(	)	==	!=	<	<=	>	>=
&&	||	int	float	boolean	string	if	endif
else	while	{	}	print	println		

Semicolons

Semicolons act as a statement terminator. Each individual statement must be ended with a semicolon. However, if statement and while statement do not require semicolon to mark the end.

Identifiers (Variables)

Identifier is a name used to identify variable. In this programming language, identifier must start with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9). Punctuation characters are not allowed as identifiers. The identifiers are case sensitive. So, Name and NAME are two different identifiers in our language. Here are some examples of acceptable identifiers:

name	name24	Capital	_identity_	a
int_value	_temp	retValue	abc123abc_	b1

An identifier consists of alphanumeric characters. However, it may not start with digits but may start with any alphabetical character or underscore (_).

Whitespaces

Whitespaces are required when the compiler requires to distinguish between keyword and variable name. But however, in assignment statements, whitespaces can be ignored. These whitespaces increase readability of the program.

For example, these two statements are the same thing.

int age = 24;

int age=24;

Data Types

Data types refer to a system used for declaring variables of different types. The types of data that our language supports are namely Integer (int), Floating Point (float), Boolean (boolean) and String (string).

Data Type	**Acceptable Value**	**Keyword**
Integer	Decimal digits (integers)	int
Floating	Floating point numbers	float
Boolean	true or false	boolean
String (char)	alphabets, digits and special characters enclosed in single quotes	string

Variable Definition

A variable definition specifies a data type and contains a list of one or more variables of that type as follows:

i. *type variable;*
ii. *type variable1, variable2, variable3;*

Here type means a valid data type specified in token list. In the second declaration, three variables are defined of type "type". Some valid declarations are as follows:

- *int a, number, c;*
- *float pi, area;*
- *boolean hasPassed, isTrue;*
- *string office_name, personName;*

By default, variables are initialized as 0, 0.0, false and empty string for integer, float, boolean and string types respectively. Variables can also be initialized and assigned an initial value in their declaration. The initializer consists of an equal's sign followed by a constant expression as follows:

- *type variable_name = value;*

Following are some examples of initialization and assigning value:

- *int a = 34;*
- *float pi = 3.1415;*
- *boolean isOn = true;*
- *string name = 'PLC';*

Arithmetic Operators

The following table shows all the arithmetic operators supported by our language.

Operator	Description
+	Adds two operands
-	Subtracts second operand from first
*	Multiplies two operands
/	Divides first operand by second operand
%	Modulus Operator shows the remainder of after division operation

A successful operation is carried out only if two operands are of same type (Type Checking done).

For example, operating on two different types of operands int and float result in an error as follows:

- *int number1 = 56;*
- *float number2 = 5.2;*
- *print(number1 + number2); // error message will be generated*

Relational Operators

The following table shows all the relational operators supported by our language.

Operator	Description
==	true if two operands are equal, else false
!=	true if two operands are not equal, else false
>	true if first operand is greater than the second operand
>=	true if first operand is greater than or equal to the second operand
<	true if first operand is less than the second operand
<=	true if first operand is less than or equal to the second operand

Logical Operators

The following table shows two logical operators supported by our language.

Operator	Description
&&	true if two operands are true, else false
||	true if one operand among two operands is true, else false

Assignment Operator

Our language supports assignment operator as follows:

Operator	Description
=	Assigns values from right side operand to left side operand

Statements

Declaration of the single statement need not to be enclosed in the curly braces but if multiple statements need to be executed, they need to be enclosed by "{"and "}".

Conditionals (If Else Endif)

Conditionals are decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally other statements to be executed if the condition is determined to be false.

This language supports "if statement" and "if-else statement". **"if" is to be ended by "endif".**

An example code is shown below:

```
int input = 5;
 if ((input % 2) == 0)
        print('even');
else
        print('odd');
endif
```

- Multiple "if" statements can be nested to make deeper level of decision making.

For example:

```
int n1 = 10;
int n2 = 100;
string greater = 'Greater';
string bigger = 'Bigger';
string smaller = 'Smaller';
int number = 1;
if (number > n1)
 {
          if (number > n2)
          {
                    print(bigger);
          }
          else
           {
                    print(greater);
          }
endif
}
else
          print(smaller);
endif
print('nested if statements');
```

Loop (While)

When block of code needs to be executed several numbers of times, loops are used. Our language supports while loop. While loop allows to run block of statements until some condition is met. The programmer needs to be careful about the condition, **if condition is never met, the loop can run indefinitely**. The syntax for while loop is shown below:

```
while ( boolean_expression ) {
          statements;
          statements;
}
```

This example code below outputs integers from 0 to 10 using while loop.

```
int count = 0;
 while (count <= 10) {
          print(count);
          count = count + 1;
}
```

User Interface

The user interface of this language is simple. There is a ‘User Input’ where programmer writes codes. It is the only editable area. Output Console is not editable and is just meant to show the output.

‘Run Input’ button runs the code. Output Console shows the output or errors if any. ‘Reset Input Output’ button resets the whole application and puts it into the initial state. ‘Clear Console’ button clears the Output Console box. For more details figure 1 will give better picture of design.

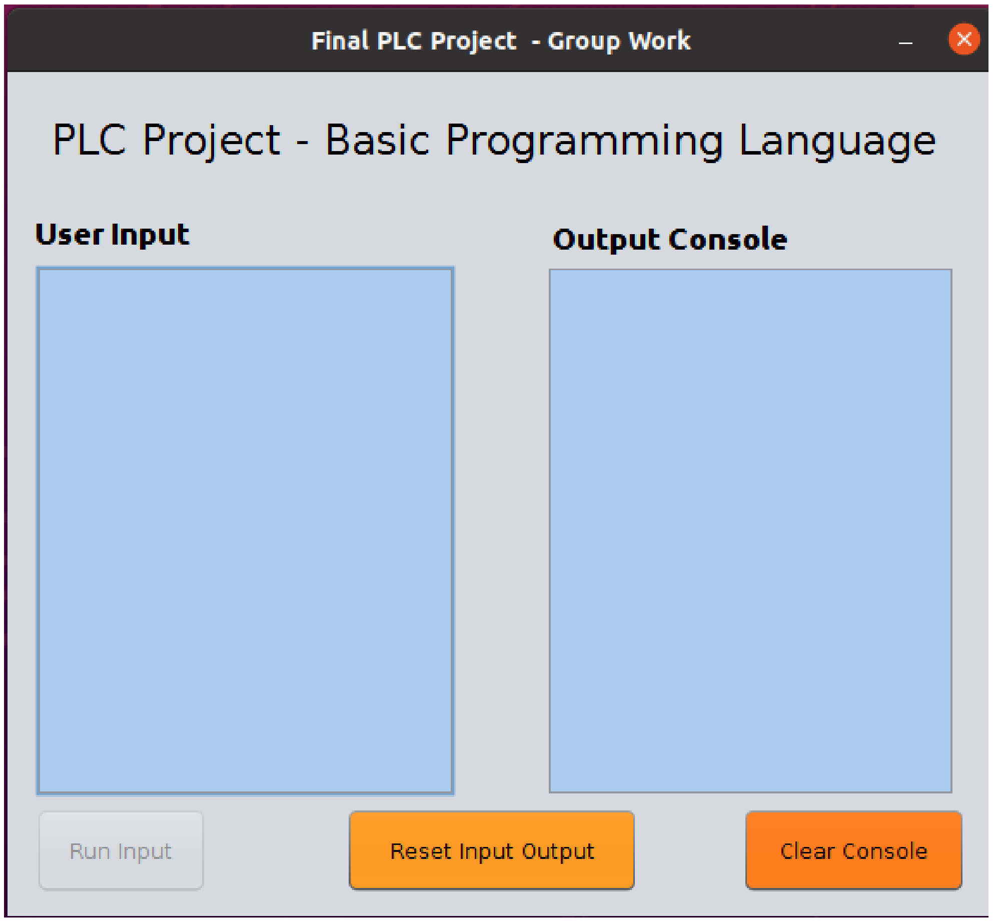
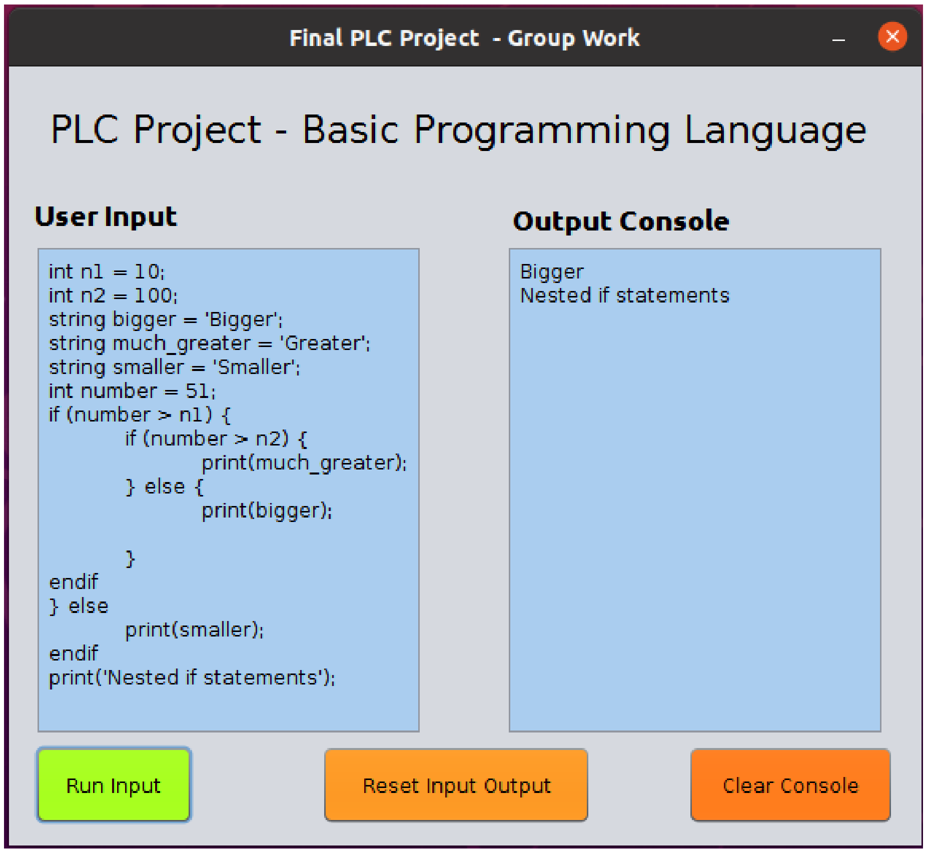

Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

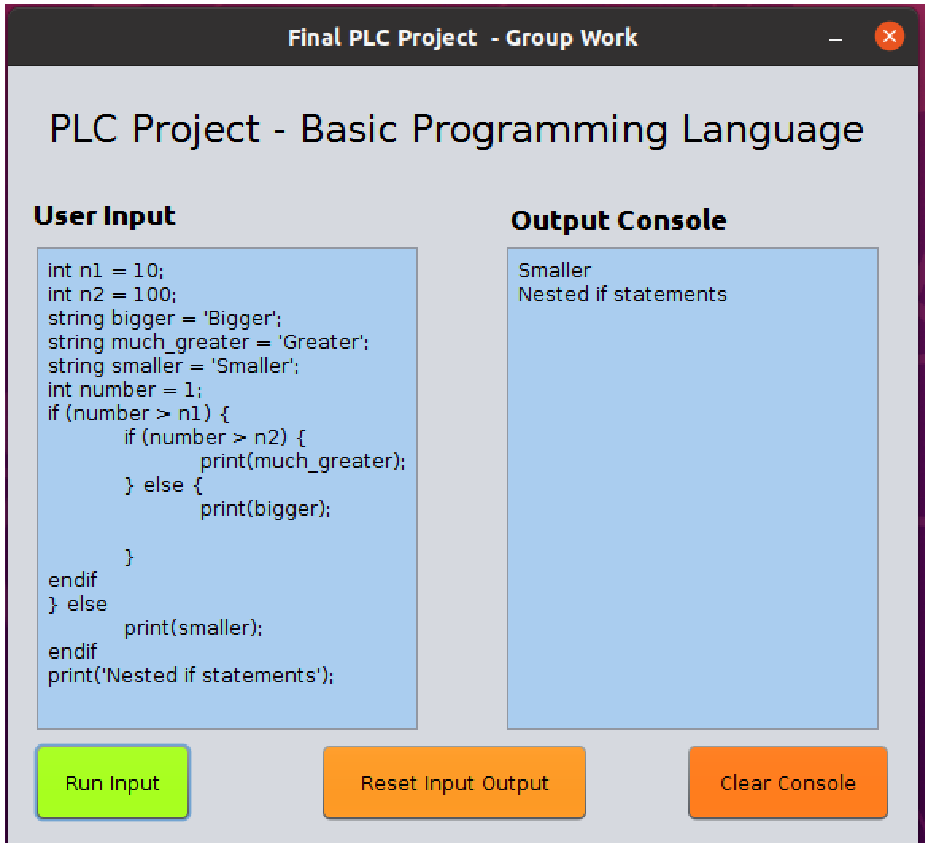
Figure 1. User Interface of Our Language


Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

```
int n1 = 10;
int n2 = 100;
string bigger = 'Bigger';
string much_greater = 'Greater';
string smaller = 'Smaller';
int number = 51;
if (number > n1) {
        if (number > n2) {
                print(much_greater);
        } else {
                print(bigger);

        }
endif
} else
        print(smaller);
endif
print('Nested if statements');
```

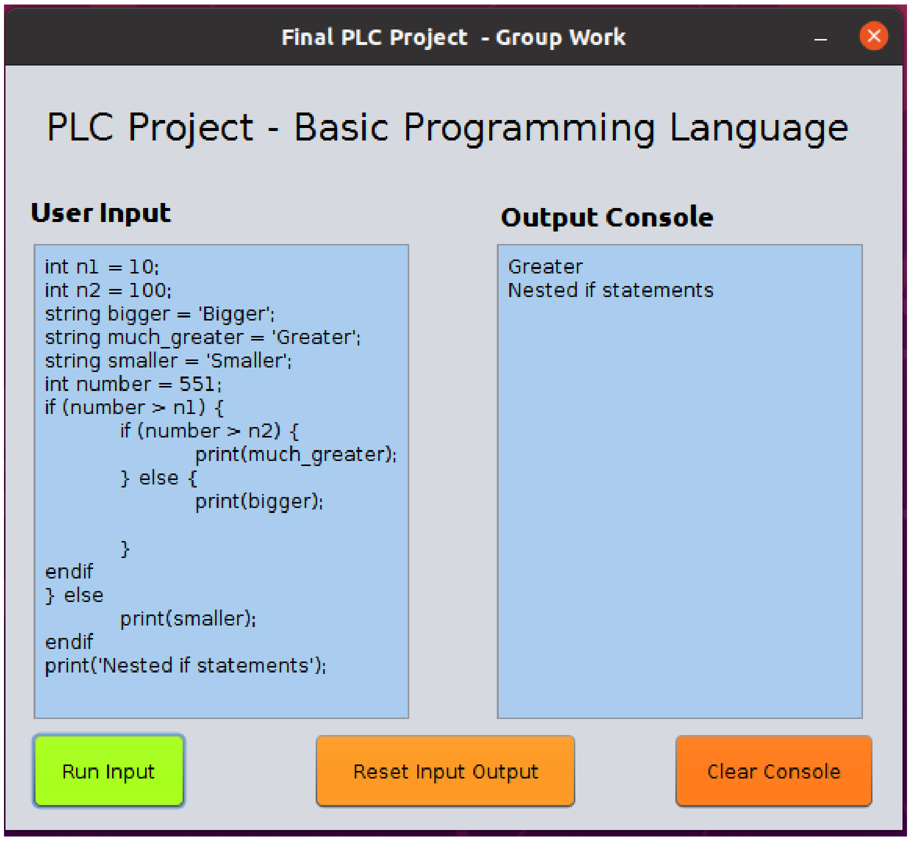
```
Bigger
Nested if statements
```


Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

```
int n1 = 10;
int n2 = 100;
string bigger = 'Bigger';
string much_greater = 'Greater';
string smaller = 'Smaller';
int number = 1;
if (number > n1) {
        if (number > n2) {
                print(much_greater);
        } else {
                print(bigger);

        }
endif
} else
        print(smaller);
endif
print('Nested if statements');
```

```
Smaller
Nested if statements
```

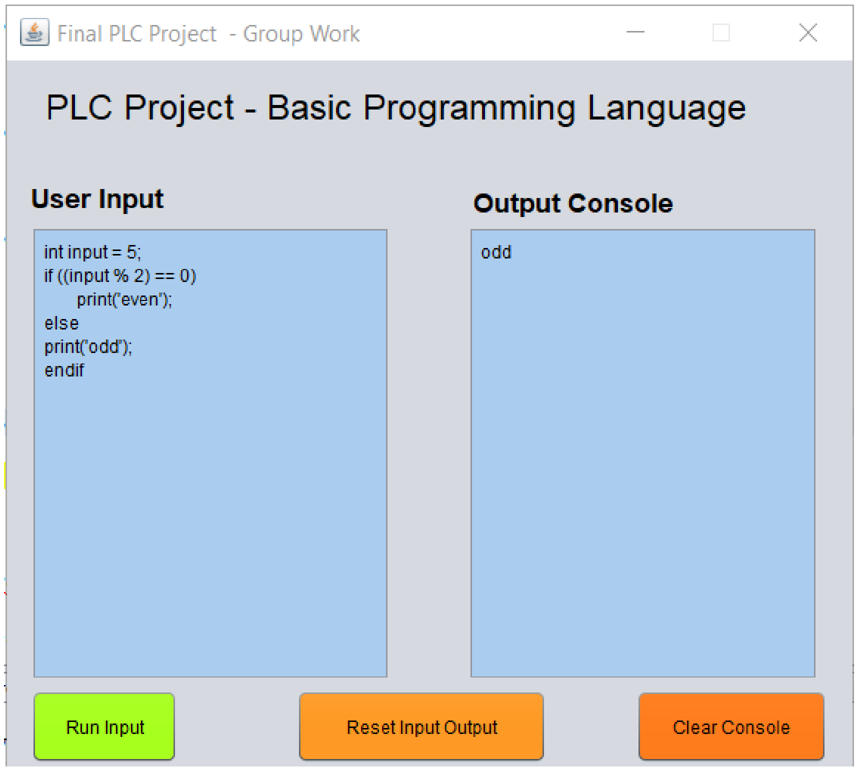

Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

```
int n1 = 10;
int n2 = 100;
string bigger = 'Bigger';
string much_greater = 'Greater';
string smaller = 'Smaller';
int number = 551;
if (number > n1) {
        if (number > n2) {
                print(much_greater);
        } else {
                print(bigger);

        }
endif
} else
        print(smaller);
endif
print('Nested if statements');
```

```
Greater
Nested if statements
```

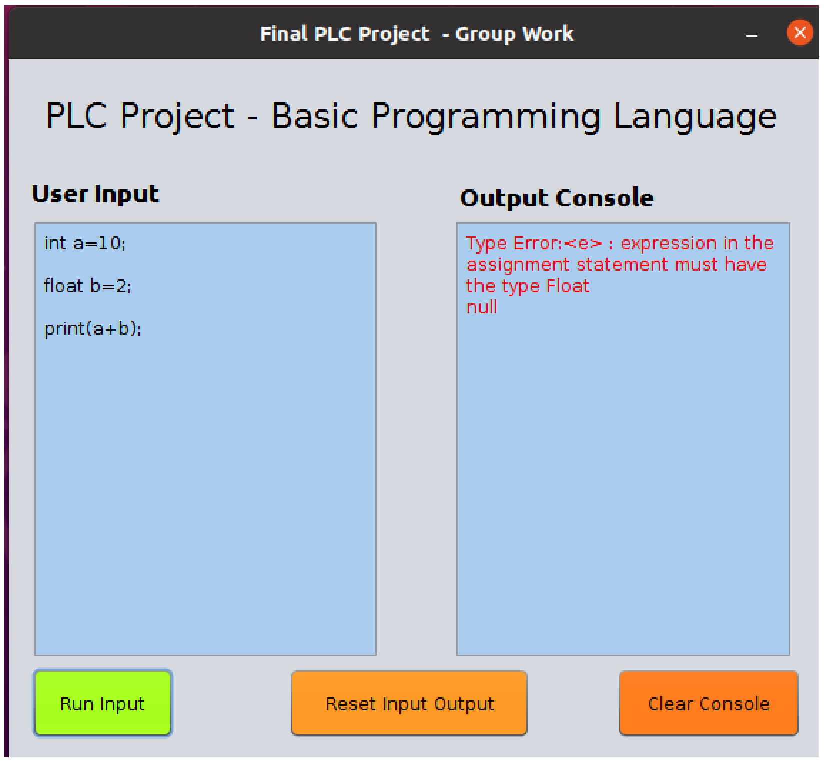
Figure 2. Executing Nested if statements program


Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

```
int input = 5;
if ((input % 2) == 0)
    print('even');
else
print('odd');
endif
```

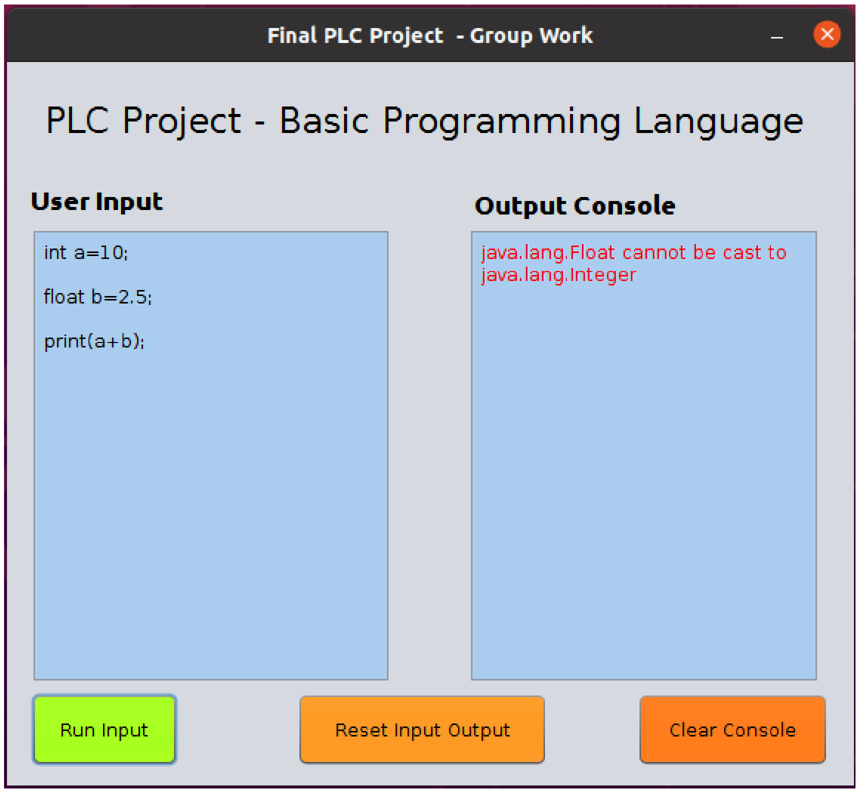
```
odd
```

Figure 3. Simple if statement


Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

```
int a=10;
float b=2;
print(a+b);
```

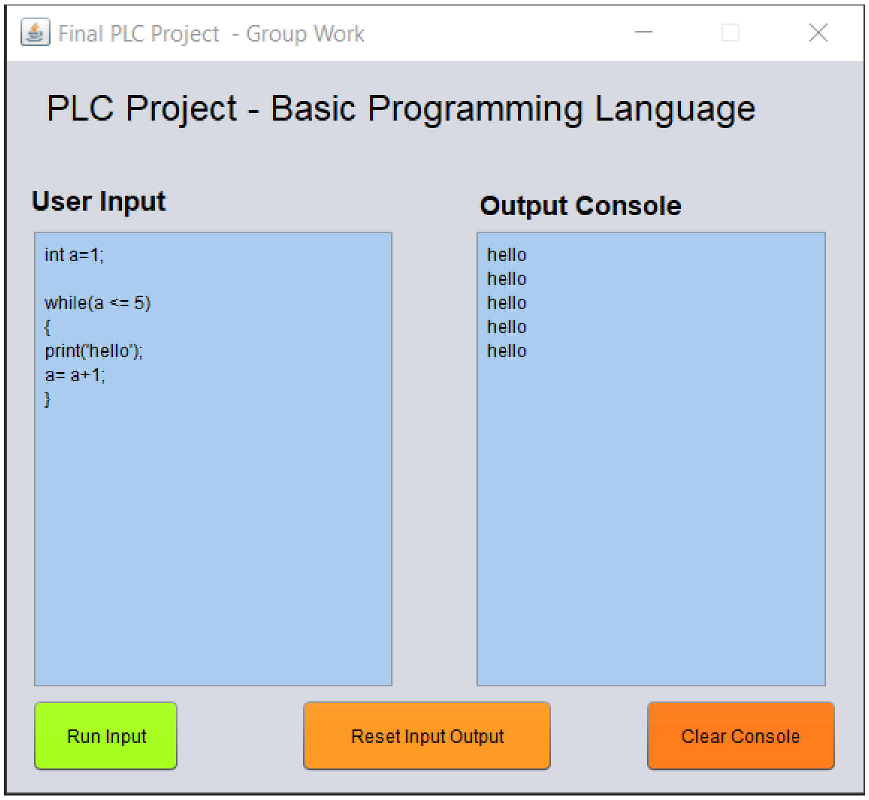
```
Type Error:<e> : expression in the
assignment statement must have
the type Float
null
```


Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

```
int a=10;
float b=2.5;
print(a+b);
```

```
java.lang.Float cannot be cast to
java.lang.Integer
```

Figure 2. Showing type error in console. Errors are displayed in Red.


Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

```
int a=1;

while(a <= 5)
{
print('hello');
a= a+1;
}
```

```
hello
hello
hello
hello
hello
```

Figure 3. while statements program

In figure 2. We have executing Nested if statements program, where we have defined the input number of n1, n2 and number with ‘10’, ‘100’and ‘551’ respectively in the ‘User input’. Then we also define string with bigger, much_greater and smaller as ‘Bigger’, ‘Greater’ and ‘Smaller’ respectively. After that, we have our nested if statements and its output base on the condition being fulfilled.

In figure 3, it is simple if statement for checking the input number is odd or even using the relational operator and printing the result depending on the statement.

In figure 4, we have tested the type checking and execution of program. If error, it should show us the errors message of type or execution error. There are two pictures where in one, we have initialized the wrong value to the datatype and in next picture we have correct datatype but executed the wrong statement of adding integer and float data type.

In figure 5, it shows our while loop running until the condition is failed and result according to it the output will be generated.

Sample code or testing code

Sample code 1:

```
string a;
a='hello';
print(a);
```

Sample code 2:

```
float b;
b=1.1111;
print(b);
```

Sample code 3:

```
int input = 5;
if ((input % 2) == 0)
        print('even');
else
print('odd');
endif
```

Sample code 4:

```
int n1 = 10;
int n2 = 100;
string greater = 'Greater';
string bigger = 'Bigger';
string smaller = 'Smaller';
int number = 1;
if (number > n1)
 {
        if (number > n2)
        {
                print(bigger);
        }
        else
         {
                print(greater);
        }
endif
}
else
        print(smaller);
```

```
endif
print('nested if statements');
```

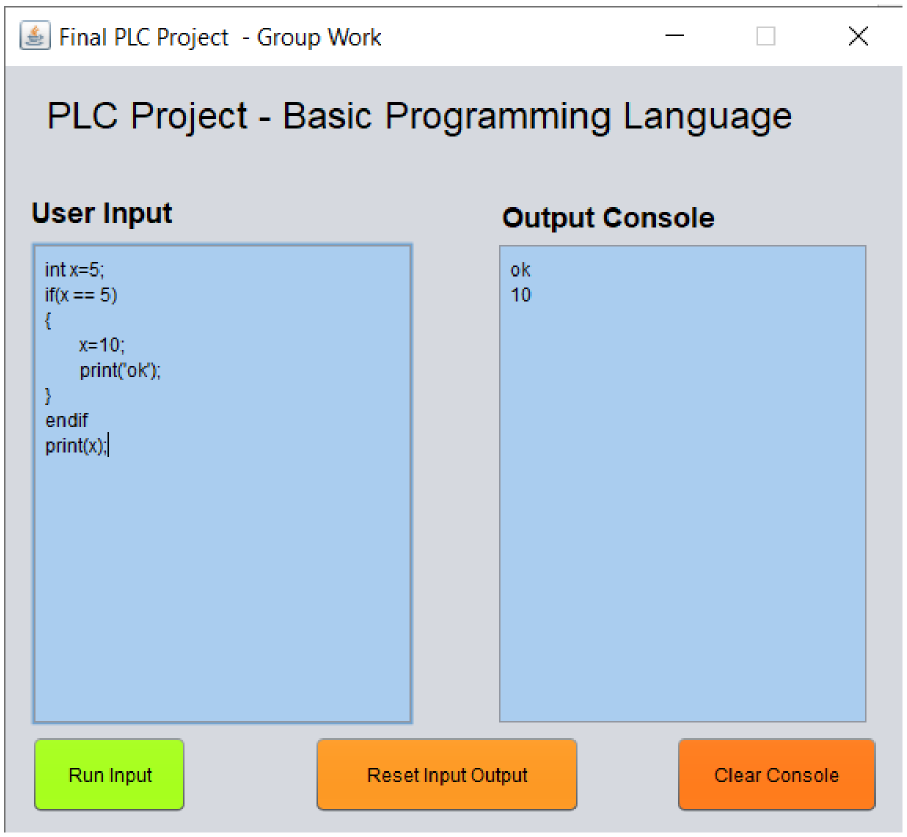
Sample code 5:

```
int count = 0;
  while (count <= 10) {
            print(count);
            count = count + 1;
}
```

Sample code 6:

```
int x=5;
if(x == 5)
{
            x=10;
            print('ok');
}
endif
print(x);
```

Output:



Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

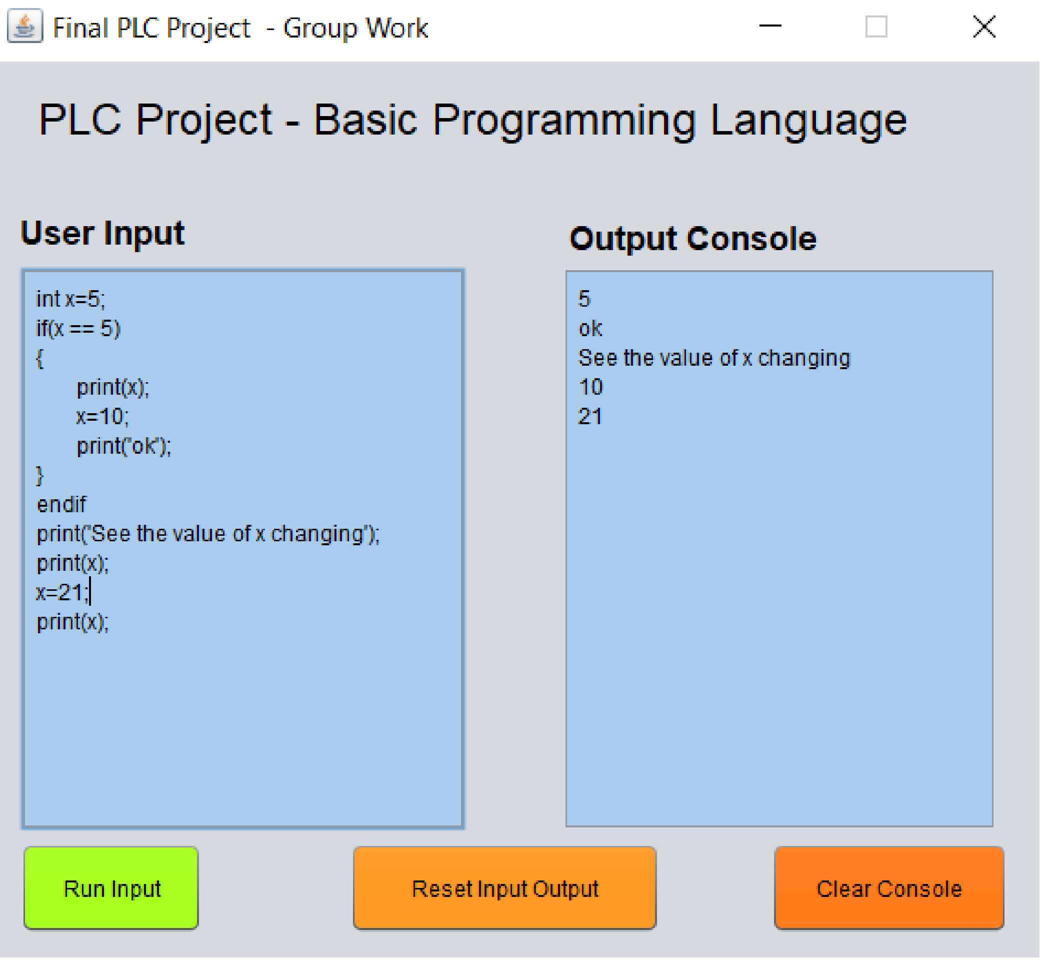
```
int x=5;
if(x == 5)
{
     x=10;
     print('ok');
}
endif
print(x);
```

```
ok
10
```

Sample code 7:

```
int x=5;
if(x == 5)
{
	print(x);
	x=10;
	print('ok');
}
endif
print('See the value of x changing');
print(x);
x=21;
print(x);
```

Output:



Final PLC Project - Group Work
PLC Project - Basic Programming Language
User Input
Output Console
Run Input
Reset Input Output
Clear Console

```
int x=5;
if(x == 5)
{
	print(x);
	x=10;
	print('ok');
}
endif
print('See the value of x changing');
print(x);
x=21;
print(x);
```

```
5
ok
See the value of x changing
10
21
```

Conclusion

The execution of the statements in the language is based on ***Abstract Syntax Tree (AST)***. The parse tree records a sequence of rules the parser applies to recognize the input. In our grammar, the start symbol is "**program**". It is the root of the parse tree. Each interior node represents a non-terminal as represented in the grammar rule like statements, declarations, variables, relop, expr etc. Each leaf node represents a token or a terminal symbol.

The variables declared are stored in the ***Environment Table*.** Environment Table is a *HashTable* that keeps track of variable names and type that variable holds. The variable names are set as keys and whenever new variable is declared, this key is checked upon and errors are shown accordingly.

All the ***evaluation*** of the expressions and statements are recursively done by the concept of ASTs. ***Type checking*** is also done where two operands are checked for their compatibility and type errors are shown if incompatible types are found.

All the evaluation of the expressions and statements are recursively done. Error checking is also done where two operands are checked for their compatibility with operator and are shown if incompatible expressions are found.

Reference

Apache NetBeans. (2020). The Apache Software Foundation. Retrieved from https://netbeans.apache.org/download/index.html

A. Aho, R. Sethi, and J. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley Publishing, Reading, MA, 1986.

Jflex (2020, May 3). *What is it?* Retrieved from https://jflex.de/

Lexical Analysis using Jflex. (n.d). Retrieved from https://www.cs.auckland.ac.nz/courses/compsci330s1c/lectures/330ChaptersPDF/Chapt1.pdf

Wielenga, G. (2014, February 26). Top 5 features of NetBeans 8. *Geertjan's Blog*. Retrieved from https://blogs.oracle.com/geertjan/top-5-features-of-netbeans-8

https://github.com/moy/JFlex/tree/master/jflex/examples/cup

http://laure.gonnord.org/pro/teaching/IF1112/codes_java.pdf

https://medium.com/@mikhail.barash.mikbar/grammars-for-programming-languages-fae3a72a22c6

https://ruslanspivak.com/lsbasi-part7/ https://www.javatpoint.com/java-swing