# C++ PROJECT

# INTERPRETER

# (e--)

## *(MANUAL)*

Prashant Mahesh
12 A
Roll no: 18

# CONTENTS

# INTRODUCTON

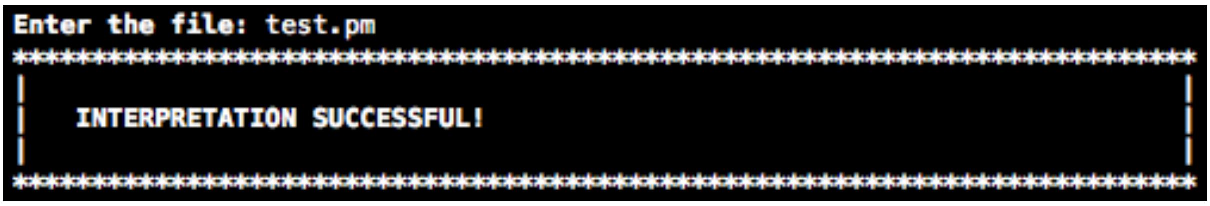
The e in “e--” stands for effort. The effort required to code using e-- or learn to code this language is significantly less. This Interpreted language is a hybrid of some of the features given by a number of languages. It is interpreted in c++. Some keywords and operators have been changed from those of c++ or other languages, so as to improve readability and understanding. This language is easy to learn, as it has straightforward syntax. It is a good language for a beginner to use, to appreciate programming using a very simple tool, to expand his ability to solve problems using programming.

# INTERPRETER INSTRUCTIONS

The first thing that is asked by the program is the file you wish to run. This file has to have a ".pm" extension in order to run. If the file reading was successful, the program begins to interpret. If it was unsuccessful, an error message "FILE READING FAILED" appears..

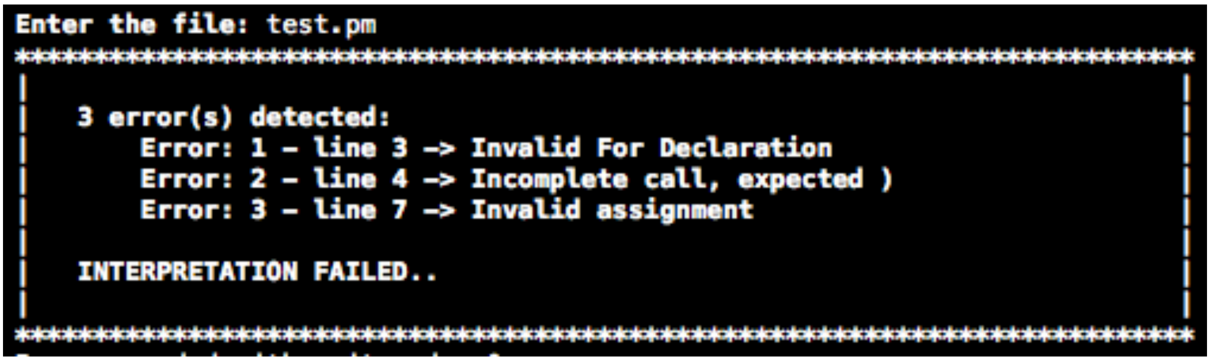
## INTERPRETATION

If the interpretation was successful, the following screen appears:



```
Enter the file: test.pm
****************************************************************************
|                                                                          |
|   INTERPRETATION SUCCESSFUL!                                             |
|                                                                          |
****************************************************************************
```

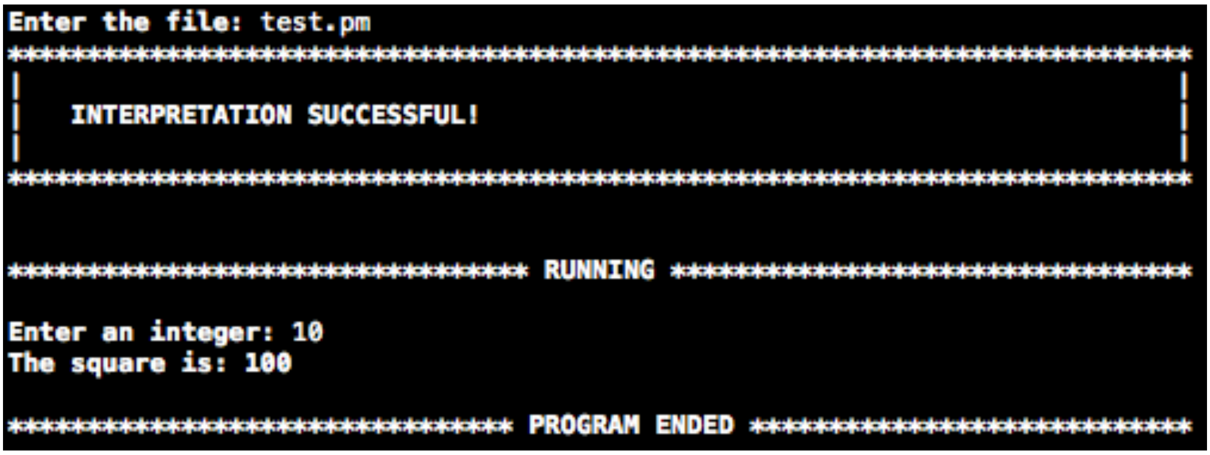
If the interpretation was unsuccessful, the following screen appears with the errors displayed in the following format:



```
Enter the file: test.pm
****************************************************************************
|                                                                          |
|   3 error(s) detected:                                                   |
|       Error: 1 - line 3 -> Invalid For Declaration                       |
|       Error: 2 - line 4 -> Incomplete call, expected )                   |
|       Error: 3 - line 7 -> Invalid assignment                            |
|                                                                          |
|   INTERPRETATION FAILED..                                                |
|                                                                          |
****************************************************************************
```

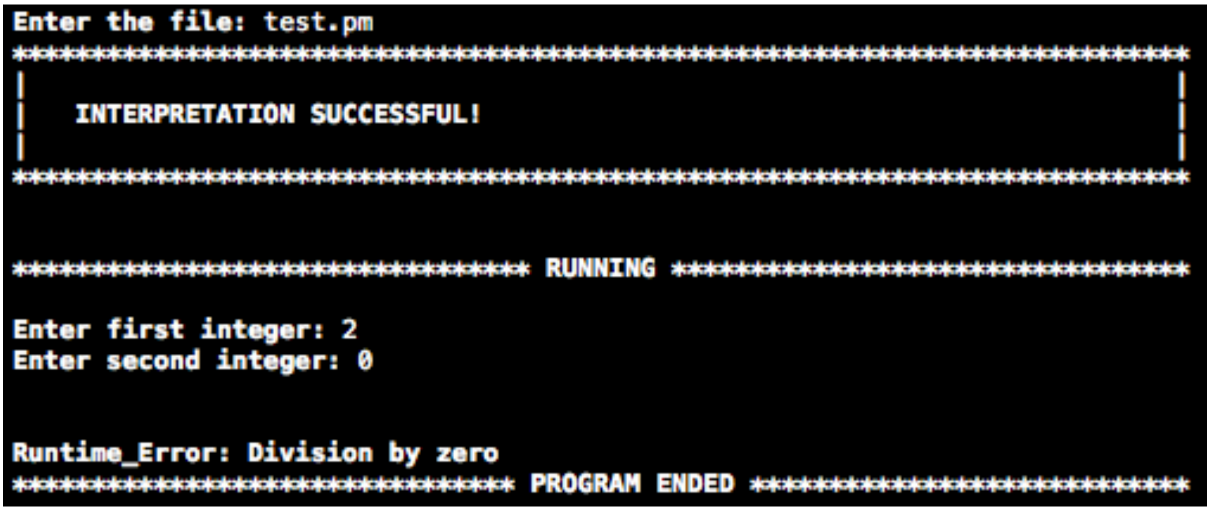
## RUNNING THE PROGRAM

When interpretation is successful, the program written by the user starts to run. The “console” for the user is between the lines “**RUNNING**” and “**PROGRAM ENDED**” as shown:



```
Enter the file: test.pm
*****************************************************************************
|                                                                           |
|   INTERPRETATION SUCCESSFUL!                                              |
|                                                                           |
*****************************************************************************


********************************** RUNNING **********************************

Enter an integer: 10
The square is: 100

********************************* PROGRAM ENDED *****************************
```

Runtime errors may occur, and they stop the program immaediately and abruptly, telling the user what the error was. An example is shown where the two integers are to be divided:



```
Enter the file: test.pm
*****************************************************************************
|                                                                           |
|   INTERPRETATION SUCCESSFUL!                                              |
|                                                                           |
*****************************************************************************


********************************** RUNNING **********************************

Enter first integer: 2
Enter second integer: 0


Runtime_Error: Division by zero
********************************* PROGRAM ENDED *****************************
```

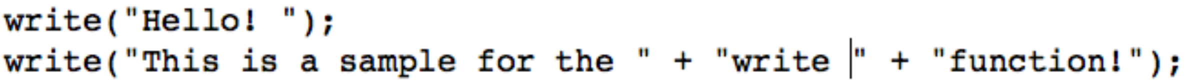
# INPUT / OUTPUT FUNCTIONS

## OUTPUT FUNCTIONS

*write()*

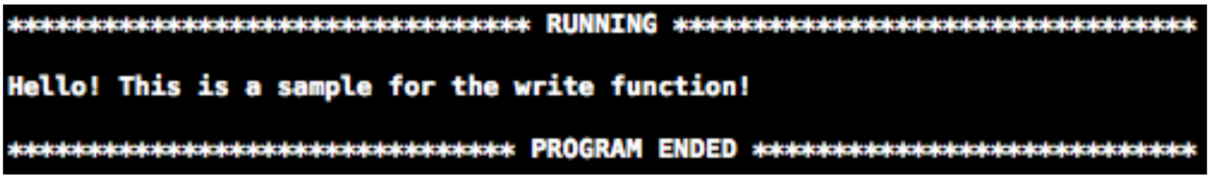
This function is used to write a string passed as parameter to the function. Integers, numerics and characters can also be given. The cursor stays in the same line after writing the text.

Example 1:



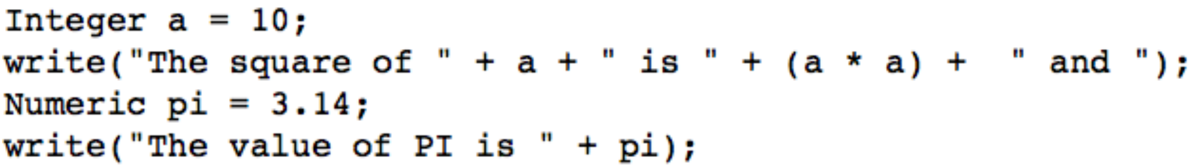
```
write("Hello! ");
write("This is a sample for the " + "write " + "function!");
```

Output:



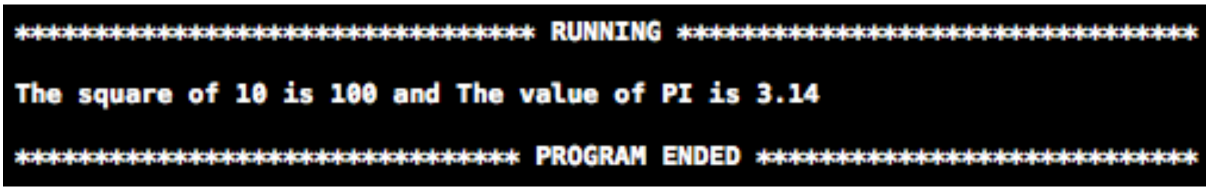
```
********************************* RUNNING *********************************

Hello! This is a sample for the write function!

******************************** PROGRAM ENDED ****************************
```

Example 2:



```
Integer a = 10;
write("The square of " + a + " is " + (a * a) +  " and ");
Numeric pi = 3.14;
write("The value of PI is " + pi);
```

Ouput:



```
********************************* RUNNING *********************************

The square of 10 is 100 and The value of PI is 3.14

******************************** PROGRAM ENDED ****************************
```

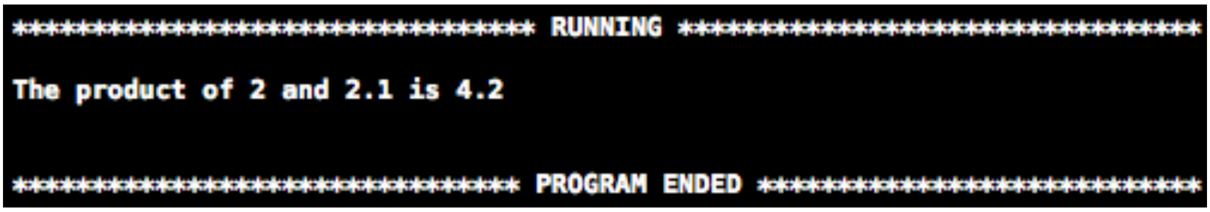
*writeLine()*

This function is similar to the write function except that the cursor moves to the next line after writing.

Example:

```
Integer a = 2;
Numeric b = 2.1;

writeLine("The product of " + a + " and " + b + " is " + (a * b));
```

Output:



```
********************************* RUNNING *********************************

The product of 2 and 2.1 is 4.2


******************************** PROGRAM ENDED ****************************
```

## INPUT FUNCTIONS

Input functions are used to obtain values from the user. All input functions have a common feature, they accept a string parameter which is the prompt to be displayed to the user before accepting.

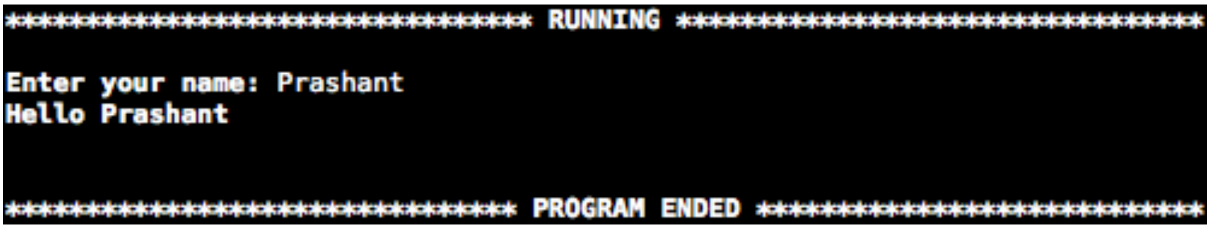
*read()*

This function is used to read a string till a white space character is identified. (To c++ users: this behaves in a manner very similar to cin’s >> operator) It returns a string, which can be assigned to variables.

Example:

```
write("Enter your name: ");
String name = read();
writeLine("Hello " + name);
```

Output:



```
********************************* RUNNING *********************************

Enter your name: Prashant
Hello Prashant


******************************* PROGRAM ENDED *****************************
```

The above code can be simlified as follows:

```
String name = read("Enter your name: ");
writeLine("Hello " + name);
```

The ouput is the same as above.

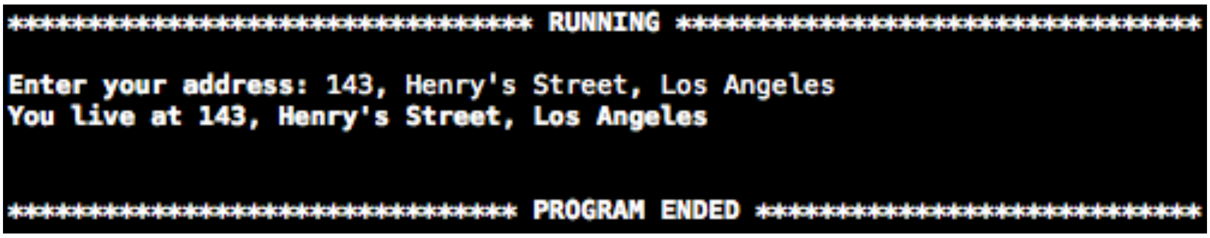
*readLine()*

This function is similar to read function. But it reads an entire line of text.(To c++ users: this behaves in a manner similar to the getline() function for cin)  Same as read, it returns a string.

Example:

```
String ad = readLine("Enter your address: ");
writeLine("You live at " + ad);
```

Output:



```
********************************** RUNNING **********************************

Enter your address: 143, Henry's Street, Los Angeles
You live at 143, Henry's Street, Los Angeles


******************************** PROGRAM ENDED ******************************
```

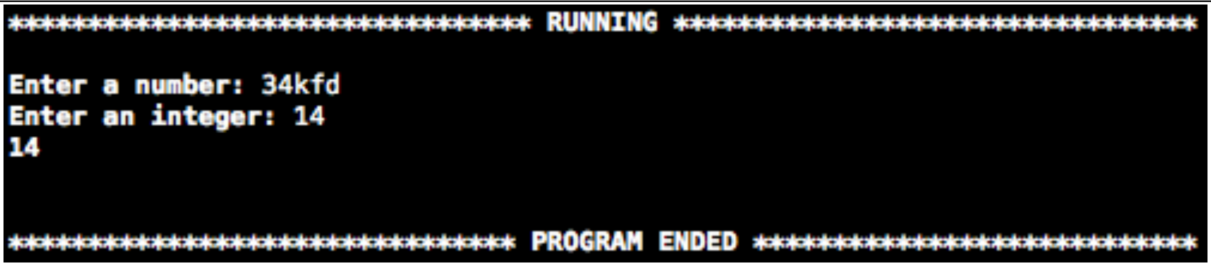
*readInteger()*

This function is used to read integers from the user. The function's aditional feature is that it keeps prompting the user("Enter an integer: ") to enter an integer as long as the user enters anything else.

Example:

```
Integer a = readInteger("Enter a number: ");
writeLine(a);
```

Output:



```
********************************** RUNNING **********************************

Enter a number: 34kfd
Enter an integer: 14
14


******************************** PROGRAM ENDED *****************************
```

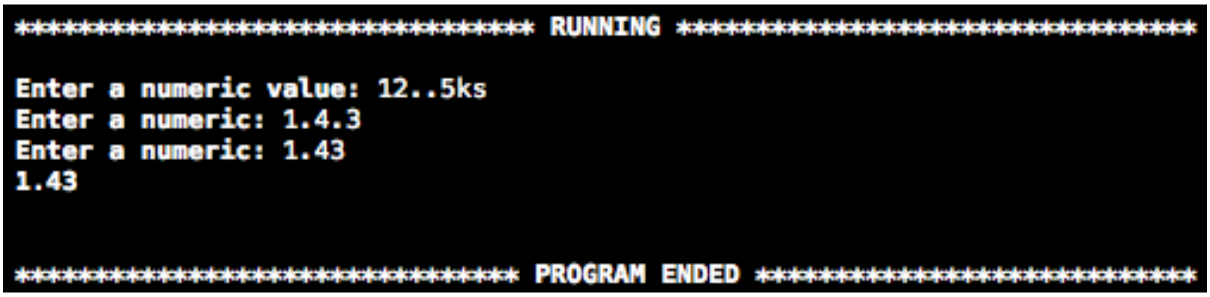
*readNumeric()*

This function is used to read numeric values from the user. The function’s aditional feature is that it keeps prompting the user(“Enter a Numeric: “) to enter an numeric as long as the user enters anything else.

Example:

```
Integer a = readNumeric("Enter a numeric value: ");
writeLine(a);
```

Output:



```
********************************** RUNNING **********************************

Enter a numeric value: 12..5ks
Enter a numeric: 1.4.3
Enter a numeric: 1.43
1.43


******************************** PROGRAM ENDED *****************************
```

# OPERATORS

The usual operators are offered, with some modified for readability.

Arithmetic Operators:

+ , - , * , / , % , ++ , --

The ++ and – operators are available in 2 forms, “post” and “pre”.

Logical Operators:

== , != , > , < , >= ,<= , and (same as c++’s &&) , or (same as c++’s ||)

Assignment Operators:

= , += , -= , *= , /= , %=

All the above operators have their usual meanings, and are used the same way as in any other language.

# VARIABLES

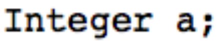
Variables store values of a particular datatype mentioned during the declartion of the variable. Variables of a particular datatype **cannot** be assigned a value which is not of the same datatype(except numeric – integer). The datatypes offered are as follows:

1. *Integer*
2. *Numeric*
3. *String*
4. *Char*
5. *Boolean*

1. *Integer*

   The Integer datatype is used to store integers like 2,3 etc.

   Example of declaration:

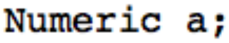
   

   ```
   Integer a;
   ```

2. *Numeric*

   The Numeric datatype stores numeric values with decimals like 3.14,1.43 etc.

   Example of declaration:

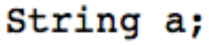
   ```
   Numeric a;
   ```

3. *String*

   The string datatype is used to store string values like “Prashant”, “hello” etc. Each character in the string can be accessed using the usual array notation(explained in *ARRAYS* section). The length of the string(as an integer) can be obtained by using the following syntax:

*a.length* (where a is a string)

Example of declaration:

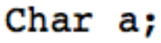


```
String a;
```

4. *Char*

This datatype stores characters like ‘a’, ‘k’.

Example of declaration:

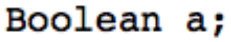


```
Char a;
```

5. *Boolean*

This datatype is fundamentally an integer, but stores 0 or 1. Keywords *true* and *false* can also be used in place of 1 and 0 respectively.

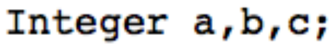
Example of declaration:



```
Boolean a;
```

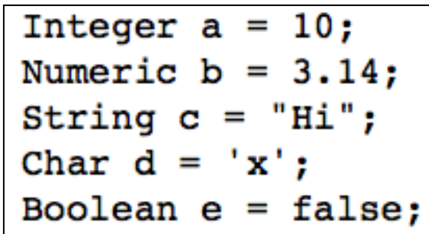
Multiple variables can also be declared in the same line.

Example:



```
Integer a,b,c;
```

Variables can also be initialized as shown:



```
Integer a = 10;
Numeric b = 3.14;
String c = "Hi";
Char d = 'x';
Boolean e = false;
```

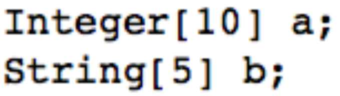
## ARRAYS

This language also offers the users to store multiple values of a particular data type in a variable.

The syntax for declaration is as follows:

*Data_type[Size]  identifier;*

Example:

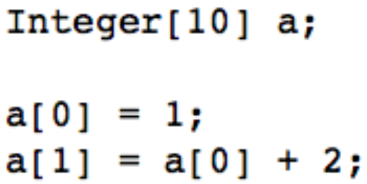


```
Integer[10] a;
String[5] b;
```

An elemet is accessed in the following way:

*Identifier[index]*

Exapmle:



```
Integer[10] a;

a[0] = 1;
a[1] = a[0] + 2;
```

# CONTROL STRUCTURES

The syntax is similar to most languages like c++, java.

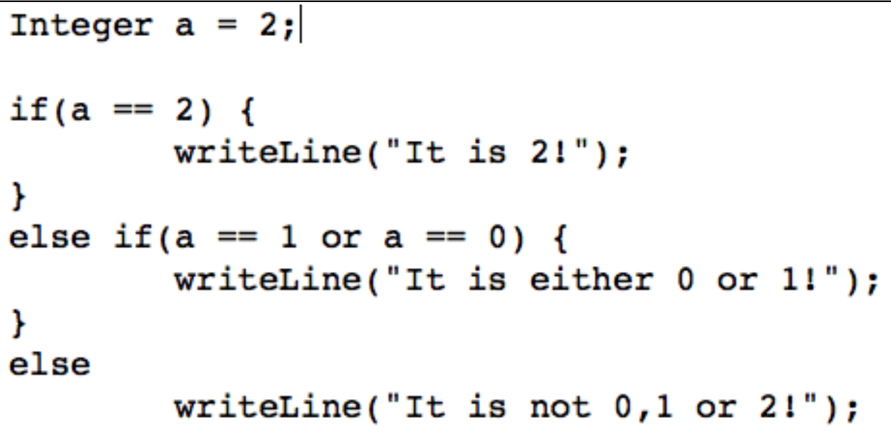
All these structures have their own scope ie. Variables declared inside the statement block of any of these control structures, **cannot** be accessed outside it.

*if – else*

Syntax:

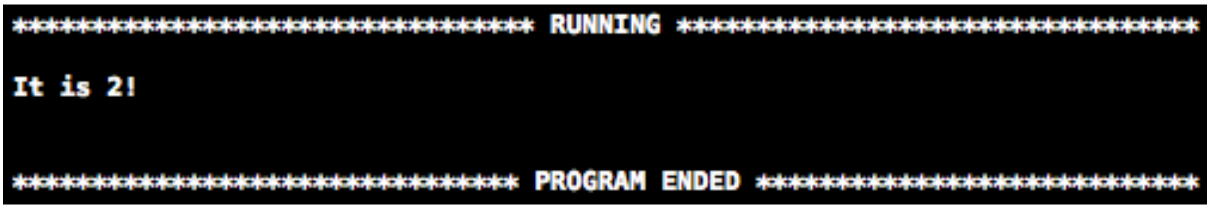
```
    if( condition ) {
        statements;
    }
    else if( condition ) {
        statements;
    }
    else {
        statements;
    }
```

Example:



```
Integer a = 2;

if(a == 2) {
        writeLine("It is 2!");
}
else if(a == 1 or a == 0) {
        writeLine("It is either 0 or 1!");
}
else
        writeLine("It is not 0,1 or 2!");
```

Output:



```
********************************** RUNNING **********************************

It is 2!

********************************* PROGRAM ENDED *****************************
```

*while*

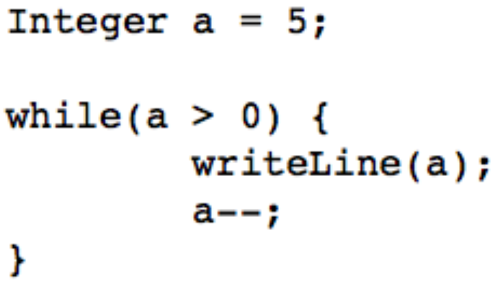
Syntax:

*while(condition) {*

*statements;*

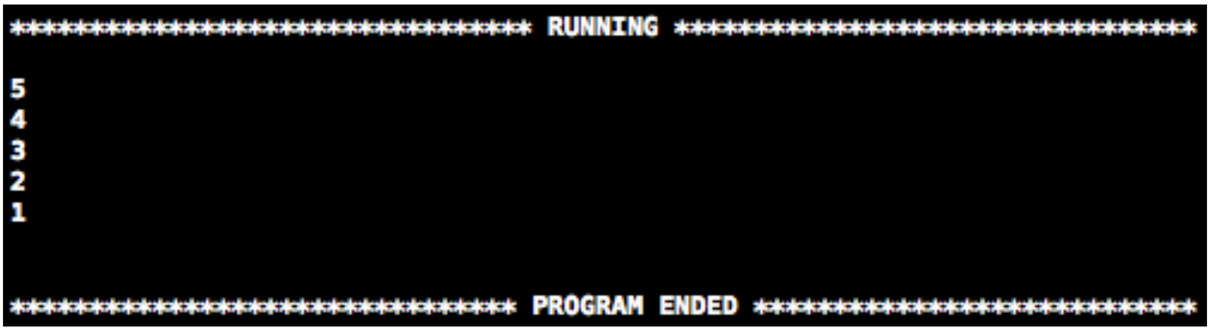
*}*

Example:



```
Integer a = 5;

while(a > 0) {
        writeLine(a);
        a--;
}
```

Output:



```
********************************* RUNNING *********************************

5
4
3
2
1


******************************** PROGRAM ENDED ****************************
```

*for*

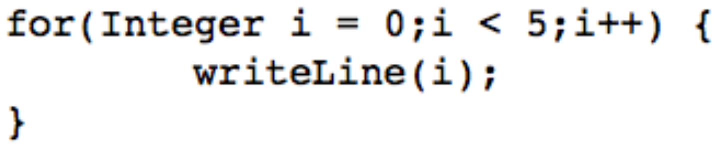
Syntax:

*for(initialization;condition;increment/decrement) {*

*statements;*

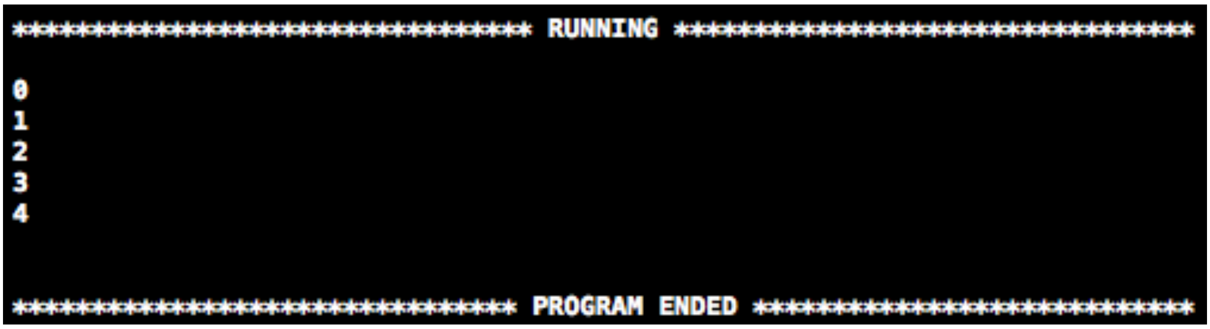
*}*

Example:



```
for(Integer i = 0;i < 5;i++) {
        writeLine(i);
}
```

Output:



```
********************************* RUNNING *********************************

0
1
2
3
4


******************************** PROGRAM ENDED ****************************
```

# FUNCTIONS

This language supports the modular approach by allowing the user to define his/her own function.

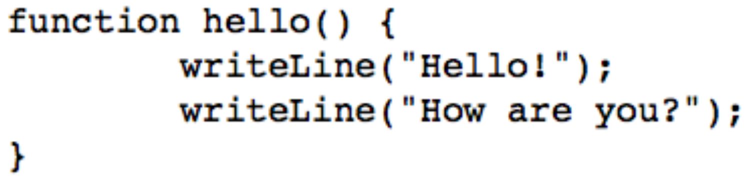
**DECLARATION:**

The keyword *function* is used before the declaration of a user defined function.

Syntax:

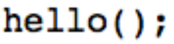
```
        function function_name(Parameters) {
            statements
        }
```

Example:



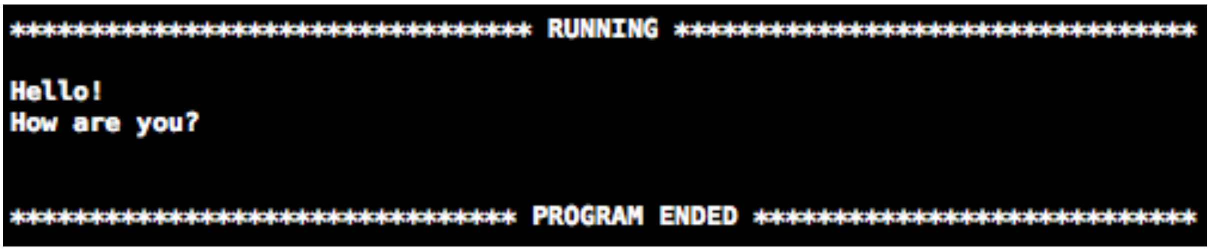
```
function hello() {
        writeLine("Hello!");
        writeLine("How are you?");
}
```

The above function is called as follows:



```
hello();
```

Output:



```
********************************** RUNNING **********************************

Hello!
How are you?


******************************** PROGRAM ENDED ******************************
```

Functions can also have parameters. They are declared as follows:

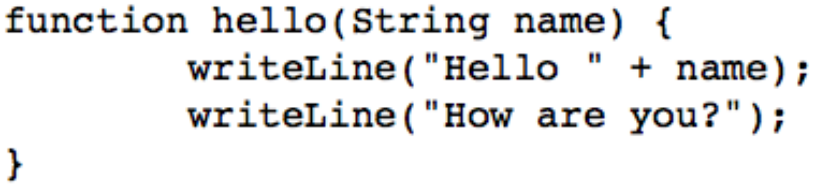
*function function_name(dt p_1, dt p_2 ..) {*

*Statements;*

*}*

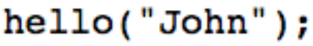
where *dt* stands for datatype and p_i stands for the identifier for the ith parameter.

Example:



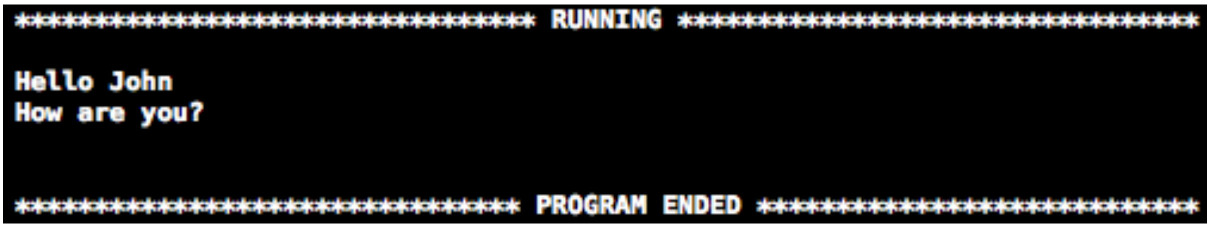
```
function hello(String name) {
        writeLine("Hello " + name);
        writeLine("How are you?");
}
```

The above function is called as follows:



```
hello("John");
```

Output:



```
********************************** RUNNING **********************************

Hello John
How are you?


******************************** PROGRAM ENDED *****************************
```

*return* Statement

The *return* keyword is used to return a value from the function. Note that, unlike languages like c++, the user **does not** specify the return type during the function declaration. Hence, while using a function, the user is assumed to know what he is doing, and knows what the function returns.

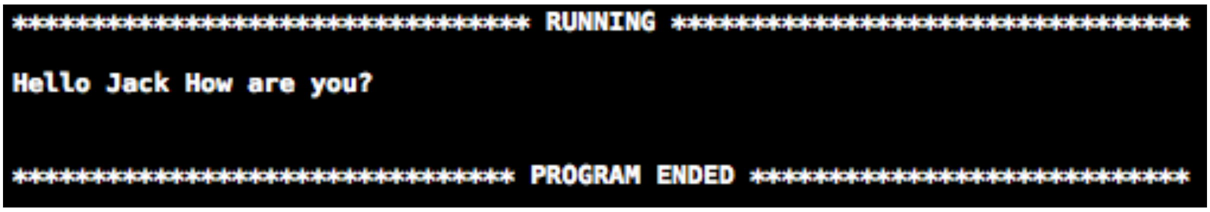
Example:

```
function hello(String name) {
        return "Hello " + name + " How are you? ";
}
```

The above function could be used as follows:

```
String t = hello("Jack");
writeLine(t);
```

Output:



```
********************************** RUNNING **********************************

Hello Jack How are you?


******************************** PROGRAM ENDED *****************************
```

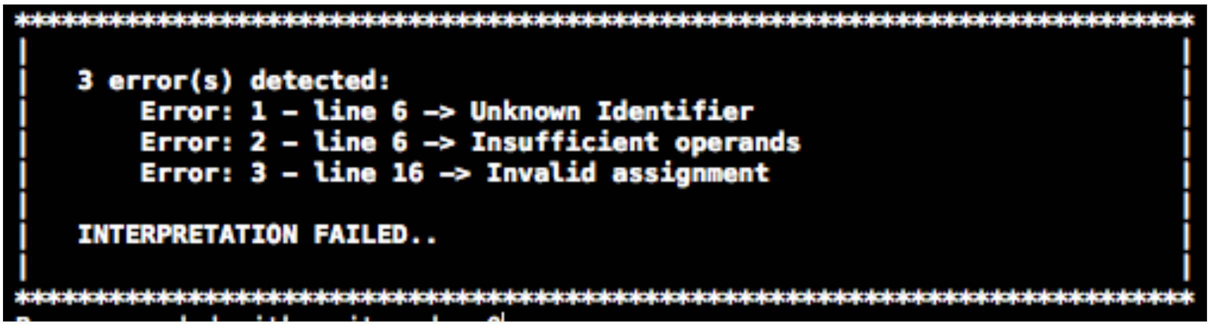
The function could also be called like this:

```
writeLine(hello("Jack"));
```

Note that the functions have their own scope too, ie. Variables declared within the function, or as parameters, are pushed into a variable “stack”. Once the function comes to an end, the variables are removed, or “pop”ed out. Hence calls like these:

```
function hello(String name) {
        return "Hello " + name + " How are you? ";
}

name = "Prashant";
```

would flag an error as follows:



```
*************************************************************************
|                                                                       |
|   3 error(s) detected:                                                |
|       Error: 1 - line 6 -> Unknown Identifier                         |
|       Error: 2 - line 6 -> Insufficient operands                      |
|       Error: 3 - line 16 -> Invalid assignment                        |
|                                                                       |
|   INTERPRETATION FAILED..                                             |
|                                                                       |
*************************************************************************
```

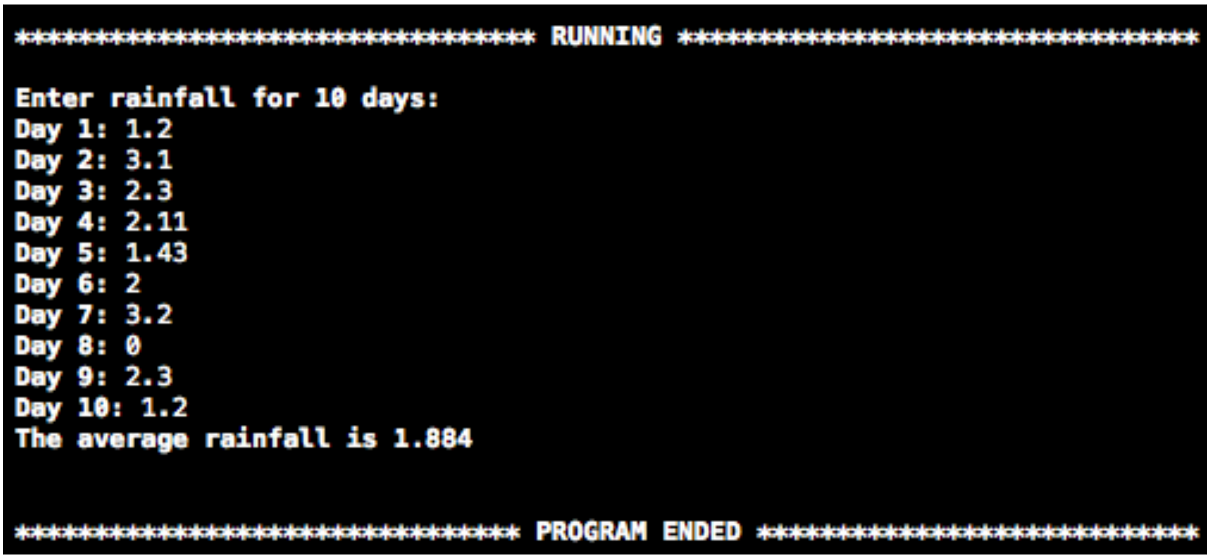
# SAMPLE PROGRAM 1

*Accept the rainfall for 10 days, display the average.*

Program:

```
Numeric[10] rainfall;
writeLine("Enter rainfall for 10 days: ");
for(Integer i = 0;i < 10;i++) {
        rainfall[i] = readNumeric("Day " + (i + 1) +": ");
}

Numeric sum = 0,average;

for(Integer i = 0;i < 10;i++) {
        sum+=rainfall[i];
}

average = sum / 10;

writeLine("The average rainfall is " + average);
```

Output:



```
********************************* RUNNING *********************************

Enter rainfall for 10 days:
Day 1: 1.2
Day 2: 3.1
Day 3: 2.3
Day 4: 2.11
Day 5: 1.43
Day 6: 2
Day 7: 3.2
Day 8: 0
Day 9: 2.3
Day 10: 1.2
The average rainfall is 1.884


******************************** PROGRAM ENDED ****************************
```

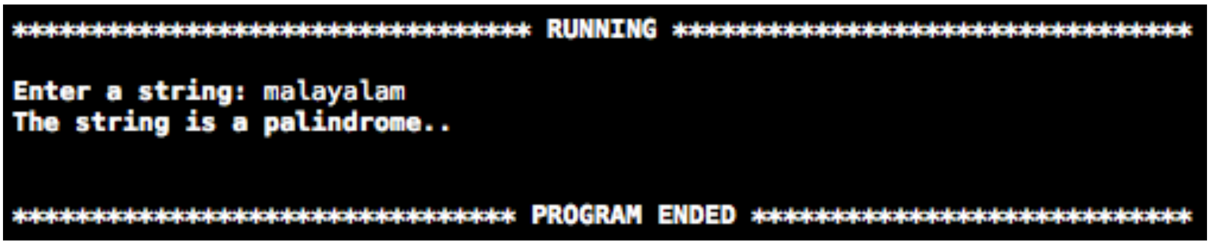
## SAMPLE PROGRAM 2

*Accept a string and display whether it is a palindrome.*

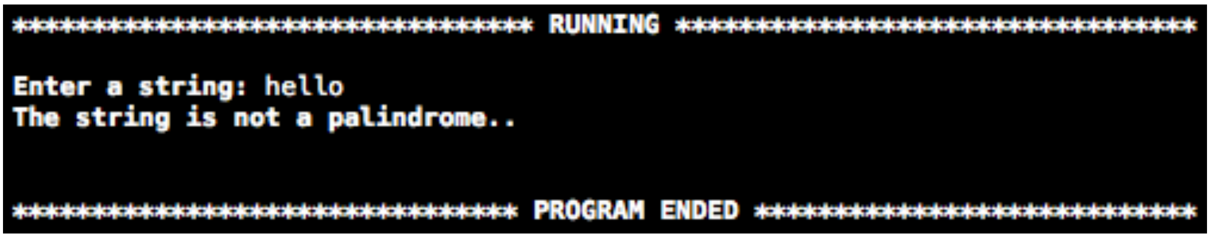
Program:

```
function isPalindrome(String s) {
        for(Integer i = 0;i < s.length/2;i++) {
                if(s[i] != s[s.length - i - 1]) return 0;
        }
        return 1;
}

String s = read("Enter a string: ");
if(isPalindrome(s)) {
        writeLine("The string is a palindrome..");
}
else {
        writeLine("The string is not a palindrome..");
}
```

Output:



```
********************************* RUNNING *********************************

Enter a string: malayalam
The string is a palindrome..


******************************** PROGRAM ENDED ****************************
```



```
********************************* RUNNING *********************************

Enter a string: hello
The string is not a palindrome..


******************************** PROGRAM ENDED ****************************
```

**************************************************************