**BASeeCAL^2 Interpreter**

**System**

**Proponents:**

**Caronoñgan, Arturo III**

**Choi, Edward**

**Chua, Kelvin**

**Theory of Programming Languages (XPRGLAN)**

**Table of Contents:**

**Declaring the starting point of your program 3**

**Your first program 3**

**Write statement 4**

**Declaring Variables 4**

**Evaluating Expressions 5**

**Read statement 6**

**Controlling the program flow 6**

**Conditional statements 8**

**Loops 9**

**While statement 9**

**Do While statement 10**

**For statement 11**

**Functions 12**

**Simple function calls 12**

**Passing parameters to a function 13**

**assign and return statement 13**

**Recursion 14**

**Global Declarations 15**

**Constants 15**

**Arrays 15**

**setArrayValue statement 16**

**getArrayValue statement 16**

**Files 17**

**Catch statement 18**

**Declaring the starting point of your program**

Before the interpreter may begin, you will have to declare the starting point of your program. The starting point is determined by declaring the main() function. Declaring the main() function is demonstrated in the textbox below.

main()

{

// your code here

}

The contents of the function main() will be the set of statements that you would wish to have the interpreter interpret. You may declare a set of write statements to echo an output or make some statements to take input from the user. The following sections will demonstrate the capabilities of the interpreter and the programming language.

**Your first program**

Now that we have set the starting point, (through function main()), we may now construct our first program. The following textbox demonstrate a simple program.

main()

{

write (“Hello world”);

}

And the output will be a simple

Hello World

Now, there are several considerations that you will have to take when you will develop your program.

1. The programming language is case sensitive. It will not recognize “Main” as “main”
2. The programming language is line sensitive. It does not tolerate multiple statements in one line
3. Majority of the statements (except for some which will be explained) will require a ‘;’ at the end signifying the end of the statement.

**Write statement**

The “write” statement is declared as follows:

write(<statement>);

where <statement> = multiple statements.

Multiple statements may be made up of a string, a variable, or a newline character. The newline character is denoted by ‘\n’ which writes a newline at the end of the string.

The write statement imitates the “printf” statement in C, although written differently. The following code demonstrates how “write” is called.

main()

{

write (“Hello world\n”);

write (“My name is John\n”);

write (“Have a nice day!”);

}

Produces

Hello World

My Name is John

Have a nice day!

**Declaring variables**

Variables are addresses which may be used to store a value. A variable could either be any of the following:

1. Integer – A whole number, declared as an ‘int’
2. Float – Supports decimals and rational numbers, declared as a ‘float’
3. Character – A single character, declared as ‘char’
4. String – A set of characters, declared as ‘string’

To display the contents of a variable, you may use the “write” statement to do so. See the example code describing the difference between the variables and how they are displayed.

main()

{

int n = 3 / 2;

float f;

char ch = 'c';

string str = “john”;

f = 3 / 2;

write ("n = "+n+"\n");

write (“f = "+f+"\n");

write ("ch = "+ch+"\n");

write (" str = "+str+"\n");

}

Outputs

n = 1

f = 1.500000

ch = c

str = john

**Evaluating Expressions**

The interpreter is also able to evaluate arithmetic expressions. Take note of the following operations:

1. Addition – denoted by the ‘+’ character
2. Subtraction – denoted by the ‘-‘ character
3. Multiplication – denoted by the ‘\*’ character
4. Division – denoted by the ‘/’ character
5. Modulo – denoted by the ‘$’ character

Operator precedence is as follows:

1st priority: ( )

2nd priority: \*,/,$

3rd priority: +,-

Arithmetic expressions are mainly associated with variables or with conditions (which will be discussed later on). A sample of how arithmetic expressions are expressed in the code is displayed below

main()

{

int n;

n = 3 + 3 \* 3;

write(n);

}

The code will display

12

**Read statement**

The “read” statement is declared as follows:

read(<variable>);

where <variable> = valid variable.

The read statement is used to simulate the “scanf” operation from the C Language which takes input normally from the standard input.

The following code demonstrates the read statement being used:

main()

{

Int n;

write (“Enter a number: “);

read (n);

write (“You entered: “+n+”\n”);

}

Produces

Enter a number: 10

You entered: 10

**Controlling the program flow:**

Up until the current section, the interpreter executes the codes linearly, meaning that the lines are executed sequentially from top to bottom. However, we are able to control the program flow by telling the interpreter that there are some statements that should be executed or not if a specific condition is met. We are able to do those through conditional statements.

**Conditional Statements**

The first conditional statement is the “if” statement.

If (<condition>)

{

// Execute these codes

}

endif;

<condition> is a Boolean expression (meaning can be evaluated as true or not true) which involves two arithmetic conditions being compared with a certain operator. The operators are as follows:

1. Equal = Represented as ‘=’
2. Greater than = Represented as ‘>’
3. Greater than or equal to = Represented as ‘>=’
4. Less than = Represented as ‘<’
5. Less than or equal to = Represented as ‘<=’
6. Not equal to = Represented as ‘!=’

Multiple conditions are possible by using conjunctions. They are either:

1. OR = represented as ‘||’
2. AND = represented as ‘&&’

To demonstrate:

<condition1> || <condition2> = Met if EITHER condition1 is true OR condition2 is true

<condition1> && <condition2> = Met if BOTH condition1 AND condition2 is true

The if statement simulates that of the C language, and the endif (which is important to indicate the end of an if statement block) simulates that in Visual Basic.

A sample code showing how if statements work is shown below.

main()

{

int n;

n = 3 + 3 \* 3;

write(n);

if (n$2 == 0)

{

write (“EVEN\n”);

}

endif;

}

Will produce:

12

EVEN

The “else” statement is there to provide another set of statements to be executed should the first set of conditions be met. The format is shown below.

If (<condition>)

{

// Execute these codes

}

else\_if (<condition2>)

{

// Executed if <condition1> is not met but <condition2> is met

}

else

{

// Executed if neither <condition1> is met and <condition2> is not met

}

endif;

And example code is shown below

main()

{

int n;

n = 101;

write(n);

if (n$2 == 0 && n < 100)

{

write (“EVEN\n”);

}

else\_if (n$2 == 1 && n < 100)

{

write (“ODD\n”);

}

else

{

write (“TOO BIG\n”);

}

endif;

}

Output:

101

TOO BG

**Loops**

Like in other programming languages, it is possible to continuously execute a statement as long as a condition is met. This is what we would call a “Loop”. The interpreter is able to handle three different types of loops, namely the “while” loop, the “Do while” loop, and the “for” loop,

**I. While loop**

The format of the while loop is displayed below.

while (<condition>)

{

// Execute these codes as long as <condition> is true

}

The difference between the while loop and the “if” statement is that in an “If” statement, after executing the block, it exits the block. However, in the while loop, as long as the condition is true, it will continue executing the statements inside the block.

A demonstration is shown underneath.

main()

{

int n = 1;

while (n <= 5)

{

write (“n = “+n+”\n”);

n = n+1;

}

}

Produces:

n = 1

n = 2

n = 3

n = 4

n = 5

**II. Do-while loop**

The format of the do while loop is as follows:

do\_while (<condition>)

{

// Execute these codes as long as <condition> is true

}

The difference between the do\_while and the while is that in a while statement, it evaluates the condition first before executing the statement. In the do while statement, it executes the statements first then it evaluates the condition. If the condition is still met, it executes again. In short, the do\_while will execute the statements at least once, while in the while, it is not guaranteed that the statements will execute at least once.

You may observe the difference below.

main()

{

write ("This is do\_while vs. while\n");

int n = 10;

while (n < 10)

{

write ("This should not execute!\n");

n = n+1;

}

n = 10;

do\_while (n < 10)

{

write ("This should execute at least once\n");

n = n+1;

}

}

This will produce the following output:

This is do\_while vs. while

This should execute at least once

**III. For loop**

The format for the for loop is as follows

for (<valid variable assignment>;test <condition>;then <valid upate statement>)

{

// Executes as long as condition is true

}

The for loop statement simulates the for loop that you may see in C or in JAVA.

An example code is shown below.

main()

{

int i;

for (i = 1;test i <= 5;then i = i+1)

{

write(“I = “+i+” and “);

if (i$2 == 0)

{

write (“EVEN\n”);

}

else

{

write (“ODD\n”);

}

endif;

}

}

**Outputs:**

I = 1 and ODD

I = 2 and EVEN

I = 3 and ODD

I = 4 and EVEN

I = 5 and ODD

**Functions**

Functions are a set of statements that may or may not return a value. These are declared outside of main and may be called repeatedly.

To declare a function, you must use the following format:

function <return type> <validname> ( <parameters> )

A function’s return type may be any valid variable type (int, float, etc.) or void. Functions with a return type of void should not return anything while other return types should be returning a value.

A function may have any number of parameters, which are valid variables separated by commas. A function declaration is displayed below.

function void echo(int number)

{

write (“Echoing: “+number+”\n”);

}

functions must be declared on top of main.

function void echo()

{

write (“Hello world\n”);

}

main()

{

echo();

}

function void echo(int number)

{

write (“Echoing: “+number+”\n”);

}

main()

{

// Code

}

**I. Simple function calls**

To call a function, simply place the correct function name and the statements in that function will be executed. Observe the following block

The code will yield this result

Hello world

**II. Passing parameters to a function**

Aside from normally calling a function, it is also possible to pass parameters as demonstrated below.

function void sum(int a,int b)

{

int kal;

kal = a+b;

write ("The sum is: "+kal+"\n");

}

main()

{

sum(3,2);

}

The code will yield this result

The sum is: 5

Make sure that the parameters being passed match the data types of the respective parameters in the function, and make sure that the number of parameters being passed is equal to the number of parameters declared in the function.

**III. The assign statement and return statement.**

The “assign” keyword (unique in the interpreter) is used for functions that return a value. The syntax is as follows:

return <valid value>

assign(<variable>) = <function call>

This is used in conjunction with the return statement in a function which is in the following format.

The statements are demonstrated in the following page.

function void drawNumbers(int n,int max)

{

int temp;

if (n <= max)

{

write ("n = "+n+"\n");

temp = n+1;

drawNumbers(temp,max);

}

endif;

}

main ()

{

drawNumbers(1,5);

}

function int function()

{

return 5+5;

}

main()

{

int n;

assign(n) = function();

write (“n = “+n+”\n”);

}

This will produce the following as a result:

10

**IV. Recursion**

The process of recursion is calling the function within itself. A recursive call needs a condition that will prevent an infinite loop. Recursion is demonstrated in the block below:

That will provide the following as output

n = 1

n = 2

n = 3

n = 4

n = 5

**Global Declarations**

Global declarations are variables which are shared by other functions and the main function. As normally, variables are only within the scope of the specific function using them, global declarations are used throughout the program. The three global declarations could either be a constant, an array, or a file.

All global declarations should be declared outside the main function and before all other function declarations.

**I. Constants**

To declare a constant, the format is shown below:

#<constant> as <value>;

A common example is using “PI” in solving for the circumference. Refer to the example below:

#PI as 3.14159;

main()

{

write (“The value of PI is:”+PI+”\n”);

}

That will produce the following as a result:

The value of PI is 3.14159

The difference between a constant and a variable (aside from its scope) is that once you set it’s value, it may never be changed. The interpreter’s error recovery scheme is able to detect a constant’s value being modified and it disregards that statement.

**II. Arrays**

Arrays are a homogeneous set of data. By homogenous, it means that it is comprised of similar types of data (int, char, float, etc.)

To declare an array, it is done through the following:

array <data\_type> <arrayname>[<size>];

An example array declaration can be seen below:

array int arrays[5];

main ()

{

int i,address;

for (i = 0;test i < 5;then i = i+1;)

{

setArrayValue(arrays[i],i);

}

}

array int arrays[10]; // an array of 10 elements

In the programming language, arrays require two predefined functions to use them. They are the setArrayValue and the getArrayValue statements

**a. setArrayValue statement**

The setArrayValue is a predefined function used for assigning an array with values at their respective indexes. It is used with the following declaration:

getArrayValue(<valid Array>[<valid Index>],<valid expression>]);

setArrayValue(<valid Array>[<valid Index>],<valid expression>]);

An example of the function being used can be displayed in the following code:

That will mean that the contents of the array “arrays” will be {0,1,2,3,4}

**b. getArrayValue statement**

The getArrayValue is a predefined function used for getting values from an array from a respective index. It is used with the following declaration.

A demonstration of the two functions (setArrayValue and getArrayValue) working together is shown in the next page.

array int arrays[5];

main ()

{

int i,address;

for (i = 0;test i < 5;then i = i+1;)

{

setArrayValue(arrays[i],i+1);

}

for (i = 0;test i < 5;then i = i+1;)

{

getArrayValue(address[i],address);

write(“value at index “+i+” is “+address+”\n”);

}

}

This code will produce the following output

value at index 0 is 1

value at index 1 is 2

value at index 2 is 3

value at index 3 is 4

value at index 4 is 5

**III. Files**

The interpreter is also able to generate files in text format through four predefined functions: fOpen, fWrite, and fClose.

A file declaration is done using the keyword “file”

fWrite(<filename>,<statements>);

fOpen (<filename>,<string>);

file <filename>;

fOpen is the function that is used to open a specific file. The declaration is as follows:

fWrite is the function that is used to write contents into a file. The declaration is as follows:

fClose is the function that is used to close the file and to write it into memory. The declaration is as follows:

catch

{

// execute these statements If a runtime error occurs

}

Hello world

fClose(<filename>);

Using the following statements and file declaration, a sample program which will create a new text file called “check.txt” with “Hello world” written into it is shown below:

file pFile;

main()

{

fopen(pFile,"check.txt");

fwrite(pFile,"Hello world\n");

fclose(pFile);

}

This should produce the file “check.txt” in the directory of the interpreter with the following written inside it:

**Catch statement**

The catch statement slightly imitates that of the ones found in JAVA, C++, and other programming languages that support exception handling. The Catch statement’s main aim is to allow the interpreter to recover from a runtime error (IE: Division by 0, Invalid input, invalid array index, etc.)

The catch statement is only declared with the following

To demonstrate the catch statement, refer to the code being displayed in the next page

main()

{

float a,b,c;

a = 4;

b = 0;

c = a/b;

write ("The quotient is: "+c+"\n");

write ("The end of the program!\n");

}

Normally, upon executing that block of code, we will receive the following runtime error notification.

Recovering from error

The quotient is: 2

The end of the program!

UNCAUGHT EXCEPTION REPORTED!

DIVISION\_BY\_ZERO\_EXCEPTION reported

Unable to perform complete arithmetic operation

Invalid operation encountered. Division by '0' cannot be completed

Exit signal obtained

With a catch block placed:

main()

{

float a,b,c;

a = 4;

b = 0;

c = a/b;

write ("The quotient is: "+c+"\n");

catch

{

write (“Recovering from error: “);

b = 2;

c = a/b;

write (“The quotient is: “+c+”\n”);

}

write ("The end of the program!\n");

}

That enables the program to recover from the error at runtime: