X

X is an objected oriented interpreted language. This phase of the project aims at development of procedural aspect of the language. This phase of the project assume that there is only one file of program to execute.

Symbols

Symbols are the user defined words given as variable names or function names.

A symbol is the string of characters containing letters (A to Z, a to z) numbers (0 to 9) or underscore

A symbol should always start with a letter

For example 'hello_world' is valid and '12_Hello', '_var' are invalid

A symbol should never ends with an underscore. That is 'hello_' is invalid

A symbol is case sensitive. That is 'HELLO' and 'hello' are different symbols

Keywords

Keywords are reserved words used by the language. These should not be used as a symbol. That is, it should not be used as a variable name or function name. The keyword defined for this language are

if

else

elif

while

break

continue

true

false

null

fun

return

global

Operators

An operation defines a statement that that has a result value. For eg a statement 5+3. This has a result value equal to 8. This also can be defined a 5+3 returns a value 8

Example

a=20

b=30

c = a+b

The statement a+b returns 50 that is assigned to variable c

Assignment Operators

= This is an assinment opetrator. This assigns a value to a variable.

Example

a =10

The variable (also called as reference variable) a is assigned a value 10.

a = b

The variable a is assigned the value in the variable b

Arithmetic Operators

This operators includes

+ This is an addition operation. This is used to add 2 integers or 2 strings. Addition of two integers gives the result of mathematical addition

Addition of two strings causes the concatenation of two strings. That is c = “abcd” + “efGH” gives c the value “abcdefGH”

- This is a subtraction operation. This is used to subtract 2 integers. Subtraction of two integers gives the result of mathematical subtraction.

/ This is a division operation. This is used to divide 2 integers. Division of two integers gives the result of mathematical division.

***** This is a multiplication operation. This is used to divide 2 integers. Division of two integers gives the result of mathematical division.

% This is a modulo operation. This is used to find reminer 2 integers. Reminder of two integers gives the result as reminder of mathematical division.

Comparison operators

== If two values are equal, this opetation returns True. If they are not equal, this operation returns false.

For example

a+b == c

If the result of opetaion a+b is equal to the value in c, then this operation returns True, else False.

a == b

If the value of a is equal to the value of b, then this operation returns True, else it returns False.

a == 10

If the value of a is equal to 10, then this operation returns True, else it returns False.

!= Similar to the above case. In this case if the value at left side is not equal to the value at right side, then this operation returns True, else it returns False.

< Similar to the above case. In this case if the value at left side is less than the value at right side, then this operation returns True, else it returns False.

<= Similar to the above case. In this case if the value at left side is less than or equal to the value at right side, then this operation returns True, else it returns False.

> Similar to the above case. In this case if the value at left side greater the value at right side, then this operation returns True, else it returns False.

>= Similar to the above case. In this case if the value at left side is greater than or equal to the value at right side, then this operation returns True, else it returns False.

Logical

A logical operator is generally used in loops and conditions. But practically it can be used anywhere a programmer wants. A logical operator returns true or false.

Definition of true and false are explained in section TODO

and – This operation returns true if both the operands are nonzero integers, else returns false.

a	b	return
false (0)	false(0)	false(0)
false (0)	true(1) or any non zero integer	false(0)
true(1) or any non zero integer	false (0)	false(0)
true(1) or any non zero integer	true(1) or any non zero integer	true(1)

or – This operation returns true if any of the operands are nonzero integers, else returns false.

a	b	return
false (0)	false(0)	false(0)
false (0)	true(1) or any non zero integer	true(1)
true(1) or any non zero integer	false (0)	true(1)
true(1) or any non zero integer	true(1) or any non zero integer	true(1)

not – This operation represent complement of an operand. This returns false if the operand is a nonzero integer. This returns true if the operand is 0

a	return
true(1) or any non zero integer	false(0)
false (0)	true(1)

The operation statement, except assignment, can be simple as 'a+b' or can be complicated as 'a+b-c*d and f ' The order of processing the operation can be decided by the operator precedence. Follow the operator precedence of 'C language'

The opetations can also be grouped by brackets '(' and ')'. The operation inside the brackets are processed first before the operations outside. For example in 'a*(c+d)' 'c+d' is prcessed and the result of which is processed with 'a'. In the case of nested brackets, the operation lies inside is processed first, before processing the outer brackets. For example (a+(b-(c*d))). In this case c*d is processed first.

Statement

A program consist of multiple statements. A statement ends with a semicolon ';' or a newline character (This means that if there is only one statement in a line, then the semicolon is optional), except in cases such as loop, if condition, function declaration. Or the statements coming inside a brackets '(' ')' of loop and conditions.

If more than one statement is in a single line, it is separated by semicolon ';'

For examlple. a=b+c ; c=d+e

All additional spaces in the program are discarded.

Every statement should be with in one line.

For example

'A = b+c' is valid

where as

a = b+

c

is not valid.

If a statement has to be in multilines, it has to be with in brackets '( )', '[ ]' or '{ }'

For example

A = (b +

c)

is valid

Assignments

Assignemt statement assignes a value of the operation statemtn defined in section 'Operators' to a variable.

Syntax,

a = <operation>;

Conditions and loops

Defined in section 'Conditions'

Function definition

Defined in section 'Functions'

Function call

Defined in section 'Functions'

Return statement

Defined in section 'Functions'

< I might have missed something >

Functions

Function is a block of code statement. Every code is written inside a function.

Syntax

```
fun function_name(var1, var2, .., varN){

<Code block>

return variable; < Return is an optional statement, if  not return statement is there, the function returns Null >

}
```

Any function can call any other function execute a particular set of code. The return statement gives the result of the function to the caller.

Example.

```
fun function_1(){
        Statement 1;
        ….
        Var = function_2()
        ….
}


fun function_2(){
        Statement A;
        ….
        C = 5+3
        Return C
}
```

The variable returned by function_2() (that is C) will get assigned to the variable 'Var' in function function_1()

Calling a function

The function is called by entering Function name followed by an opening bracket '(' , then variable names (if any, separated by commas) and closing bracket ')'

<u>syntax</u>

func_name(var1, var2, …etc)

It the called function returs a variable it can be assigned to a variable in the called function .

syntax

a = func_name(var1, var2, ...etc)

The file starts with the execution of **start**(argList) function

Condition

For condition if , else, elif is supported.

Syntax

if (<Condition>){

<if condition is true or a non zero integer, execute this>

}elif (<Condition 2>){

<if condition 2 is true or a non zero integer, execute this>

<This part is optional, can occur 0 or more times >

}else{

< If all the above condition fails, execute this>

<This part is optional, can occur 0 or more times >

}

Loops

While loop is supported for phase 1 for looping.

Syntax

while(< condition >}{

<till condition is True or a non zero integer or till It encounters break statement, execute this>

}

A 'break' statement in a loop will exit from the loop. And starts executing from the first statement outside the loop

A 'continue' statement stops the stop the loop for that iteration and will begin from the start of the loop for the next iteration.

NOTE : For loop is reserved for later phase for iteration

NOTE : There can be nested conditions and loop operations. That is, a while can contain an if and an if can contain a while. That can go on in anyway any number of times.

Variables.

A variable does not have a type definition. It means, a variable assume the type of the data associated to it.

e.g.

a=10 (variable a is now of type integer)

a = “MYname” (variable a is now of string type)

Variable scope.

A variable name defined inside a function has a scope and life inside the function. IT means that a variable declared inside a function cannot be accessed outside of it. A global variable defined outside the function has the access and life throughout all the function in the program.

To access the global variable inside a function, the keyword global is used. For eg. If the variable ‘gvar’ is defined as a global variable, To access inside a function a statement ‘global gvar’ has to be declared inside the function to access the variable. If this statement is not found, ‘gvar’ is considered a local variable.

Reference Variable

Stores the reference (similar to address) of the variables. All variables need to be stored in any reference for operation

For e.g. in a=10 A memory location is made for the integer 10 and the reference ‘a’. The address of integer is stored in reference a.

After this if a statement comes like a = “MYname”, A memory location for the string is defined and its address is stored in the reference ‘a’

Memory structure for reference variable

TYPE_REF (4 bytes)
Address (4 bytes)
Extension (4 bytes)
Extension (4 bytes)

Null

Null is a data that represent nothing. It means that a variable stores nothing.

Memory structure for int variable

TYPE_NULL (4 bytes)

Int

Stores an integer type of length 4 bytes. Also stores other information. This can store +ve and –ve integers. Binary value of the +ve integers are stored and –ve integers are stored in 2's complement method. The memory structure is done considering its OOPs extension in future, where we consider all data types as objects.

Syntax

A=12

The above statement creates an integer 12 in the memory and assigns it to the reference A

B = -12

All the preceeding 0's should be ignores, i.e. a = 001 is equal to a = 1

Memory structure for int variable

TYPE_INT (4 bytes)
Value (4 bytes)
Extension (4 bytes)
Extension (4 bytes)

Integer is also used to represent the boolean value. If the value is 0, it is considered as false. For all other values it is considered as true.

String

Stores the string of ascii charecters for now. Expandable to Unicode later.

Memory structure for string variable

TYPE_STRING (4 bytes)
Address of 1st char (4 bytes)
Length (4 bytes)
Extension (4 bytes)

Memory structure for nth character

Character code
Address of n+1th char (4 bytes)

The string is stored as list that can be helpful for fast manipulation of strings.

Syntax

A string is represented by the value in a single quotes or on a double quotes.

A = 'Hello' represents a string Hello

A = "Hello" represents a string Hello

The above statement creates a string 'Hello' in the memory and assigns it to the reference A

List

List of either integer, string or list itself. Internally it contains only the list of references.

Memory structure for List variable

TYPE_LIST (4 bytes)
Address of 1st reference (4 bytes)
Length (4 bytes)
Extension (4 bytes)

Memory structure for n^{th} reference

Character code
Address of $n+1^{th}$ reference (4 bytes)

Syntax

A = [12, 13, -22, 'Hello']

The above statement creates the list of references and assigns to the List reference.

To access the nth element the syntax A[n] is used.

For example A[0] gives the value 12, A[1] is 12 , A[2] is -22 and A[3] is 'Hello'

For any other integer, the operation should give a runtime error

True and False

True and False are the keywords defined by the language. This is internally replaced by integer 1 and 0 respectively.

Examples

5 == True

This statement returns False (Since True is replace by 1)

1 == True

This statement returns True (Since True is replace by 1)

not 5

This statement returns False (That is 0)

not 0

This statement returns True (That is 1)

Built in functions.

These are the built in functions that are provided by the language

len()

This function will give the length of the string and list vaiable. For any other variable, it should give a run time error.

For examlple

A = [12,23,"Hello"]

len(A) gives the value 4

A = "Hello"

len(A) gives the value 5

str()

This function gives the string representation of a variable

A=12

str(A) will returns the string of length 2 with charecters '12'

A = [12,23,"Hello"]

str(A) will returns the string of charecters '[12,23,"Hello"]'

A = null

str(A) returns 'null'

print(string_value)

This function will print the given string value in the output console.

For example

print("Hello") will print Hello in the output console.

Print(12) will give a runtime error since 12 is not a string.

To print integer value, you can use

print (str(12))

println(string_value)

Same as print(). Except this function also prints a newline at the end of the string_value

Comments

A comment is ignored by the compiler and is used to improve the readability of the program.

All the words in a line after the character '#' is treated as commend. The '#' inside a string should not be considered as the start of the comment.

#This is a comment.

A = 12 # This charecters after the hash character is comment.

A = "hello#2" #The charecters after the 2nd hash is comment.

#Another comment

#Yet another comment