COMSATS University Islamabad

Abbottabad Campus

3/20/2022

PROGRAMMING FUNDAMENTALS

ASSIGNMENT 01

NAME:

SYED SHAH HUSSAIN BADSHAH
(FA21-BSE-172)

SUBMITTED TO:

SIR YASHER ALI

Table of Contents

OPERATORS IN C LANGUAGE: 2
RELATIONAL OPERATORS: 2
- EQUAL TO OPERATOR (==): 2
- NOT EQUAL TO OPERATOR (!=): 2
- LESS THAN OPERATOR (<): 2
- GREATER THAN OPERATOR (>): 3
- LESS THAN EQUAL TO OPERATOR (<=): 3
- GREATER THAN EQUAL TO OPERATOR (>=): 3

LOGICAL OPERATORS: 4
LOGICAL AND: 5
LOGICAL OR: 5
LOGICAL NOT: 5
- BITWISE AND OPERATOR (&): 6
- BITWISE OR OPERATOR (|): 6
- BITWISE XOR OPERATOR(^): 6
- BITWISE COMPLEMENT(~): 6
- BITWISE SHIFT OPERATORS(>>,<<): 7

UNARY OPERATORS: 8
- UNARY MINUS (-): 8
- UNARY PLUS (+): 9

UNARY INCREMENT OPERATOR: 9
- PRE INCREMENT: 9
- POST INCREMENT: 10

UNARY DECREMENT OPERATOR: 11
- PRE DECREMENT: 11
- POST DECREMENT: 11

UNARY SIZE OF () OPERATOR: 12
LOGICAL NOT (!): 13
ADDRESS OF OPERATOR (&): 13
ARITHMITICAL OPERATORS: 14
INPUT: 16
OUTPUT: 16
- Scanf() INPUT STATEMENT: 17
- CHAR: 18
- INT: 18
- FLOAT: 18
- DOUBLE: 18

EXAMPLE: 19

QUESTION 02:

OPERATORS IN C LANGUAGE:

- Unary operators.
- Bitwise operators.
- Logical operators.
- Relational operators.

RELATIONAL OPERATORS:

The relational operators checks the relationship between two operands.

➢ EQUAL TO OPERATOR (==):

It is used to compare both operands and returns 1 if both are equal or the same, and 0 represents the operands that are not equal.

➢ NOT EQUAL TO OPERATOR (!=):

The Not Equal To Operator is the opposite of the Equal To Operator and is represented as the (!=) operator. The Not Equal To Operator compares two operands and returns 1 if both operands are not the same; otherwise, it returns 0.

➢ LESS THAN OPERATOR (<):

It is used to check whether the value of the left operand is less than the right operand, and if the statement is true, the operator is known as the Less than Operator.

➢ GREATER THAN OPERATOR (>):

The operator checks the value of the left operand is greater than the right operand, and if the statement is true, the operator is said to be the Greater Than Operator.

➢ LESS THAN EQUAL TO OPERATOR (<=):

The operator checks whether the value of the left operand is less than or equal to the right operand, and if the statement is true, the operator is said to be the Less than Equal To Operator.

➢ GREATER THAN EQUAL TO OPERATOR (>=):

The operator checks whether the left operand's value is greater than or equal to the right operand. If the statement is true, the operator is said to be the Greater than Equal to Operator.

EXAMPLE:

```c
#include<stdio.h>
int main()
{
    int a;
    int b;

    printf("Enter first Integer:  ");
    scanf("%d",&a);
    printf("Enter second Integer:  ");
    scanf("%d",&b);

    printf("%d<%d=%d\n",a,b,a<b);
    printf("%d<=%d=%d\n",a,b,a<=b);
    printf("%d>%d=%d\n",a,b,a>b);
    printf("%d<=%d=%d\n",a,b,a<=b);
    printf("%d==%d=%d\n",a,b,a==b);
    printf("%d!=%d=%d\n",a,b,a!=b);

    return 0;
}
```

OUTPUT:

```
Enter first Integer:  44
Enter second Integer:  55
44<55=1
44<=55=1
44>55=0
44<=55=1
44==55=0
44!=55=1

--------------------------------
Process exited after 13.98 seconds with return value 0
Press any key to continue . . .
```

LOGICAL OPERATORS:

There are three logical operators:

- Logical AND.
- Logical OR.
- Logical NOT.

LOGICAL AND:

If both the operands are non-zero, then the condition becomes true.

LOGICAL OR:

If any of the two operands is non-zero, then the condition becomes true.

LOGICAL NOT:

It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

EXAMPLE:

```c
#include<stdio.h>
int main()
{
    int x,y;

    printf("Enter first Number");
    scanf("%d",&x);

    printf("Enter second number");
    scanf("%d",&y);

    printf("\n %d&&%d=%d",x,y,x&&y);
    printf("\n %d||%d=%d",x,y,x||y);

    return 0;

}
```

```
Enter first Number77
Enter second number88

 77&&88=1
 77||88=1
--------------------------------
```

BITWISE OPERATORS:

➢ BITWISE AND OPERATOR (&):

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

➢ BITWISE OR OPERATOR (|):

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

➢ BITWISE XOR OPERATOR(^):

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

➢ BITWISE COMPLEMENT(~):

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

➢ BITWISE SHIFT OPERATORS(>>,<<):

There are two shift operators in C programming:

- Right shift operator
- Left shift operator.

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by the left shift operator are filled with 0. The symbol of the left shift operator is <<.

EXAMPLE:

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("\nBITWISE AND VALUE IF a=12 and b=25 = %d", a&b);
    printf("\nBITWISE OR VALUE IF a=12 and b=25 = %d", a|b);
    printf("\nBITWISE XOR VALUE IF a=12 and b=25 = %d", a^b);
    printf("\nBITWISE COMPLEMENT VALUE IF a=12 = %d", ~a);

    return 0;
}
```

OUTPUT:

```
BITWISE AND VALUE IF a=12 and b=25 = 8
BITWISE OR VALUE IF a=12 and b=25 = 29
BITWISE XOR VALUE IF a=12 and b=25 = 21
BITWISE COMPLEMENT VALUE IF a=12 = -13
--------------------------------
```

UNARY OPERATORS:

Unary operators is an operator used to operate single operand to return a new value. In other words, it is an operator have equal value of an operand or expression value. In unary operator operators have equal priority from left to right side associativity.

➢ UNARY MINUS (-):

The minus operator changes the sign of its arguments. A positive number becomes negative and a negative number becomes positive.

EXMAPLE :

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=2;
    int b= -(a);

    int n1=10;
    int n2=-20;

    printf("The Value of a:  %d\n",a);
    printf("The value of b:  %d\n",b);

    printf("The Value of -n1:  %d\n",-n1);
    printf("The Value of -n2:  %d\n",-n2);

    return 0;
}
```

OUTPUT:

```
The Value of a:  2
The value of b:  -2
The Value of -n1:  -10
The Value of -n2:  20
```

➢ UNARY PLUS (+):

The unary plus operator is represented as "+" symbol and it does not change to the operand value.

EXAMPLE :

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=10;
    int b=(-10);

    printf("The Value of a:  %d\n");
    printf("The Value of b:  %d\n");
    return 0;

}
```

OUTPUT:

```
The Value of a:  8262576
The Value of b:  -1774192080
```

UNARY INCREMENT OPERATOR:

➢ PRE INCREMENT:

This operator is represented by (++a) and is also known as pre increments operator. Which means value is increment by 1 before using operand to the expression.

➢ POST INCREMENT:

The operator is represented by (a++) and is known as post increment operator which means the value of a is incremented by 1 after assigning the original value to the expression.

EXAMPLE:

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int x,y,a,b;
    a=10;
    x=++a;

    printf("PRE INCREMENT OPERATOR");
    printf("\n The Value of x is:  %d",x);
    printf("\n The Value of a is:  %d",a);

    b=20;
    y=b++;

    printf("\n\n POST ENCREMENT OPERATOR");
    printf("\n The VAlue of y is  %d",y);

    printf("\n The Vlaue of b is  %d", b);
    return 0;

}
```

OUTPUT:

```
PRE INCREMENT OPERATOR
 The Value of x is:  11
 The Value of a is:  11

 POST ENCREMENT OPERATOR
 The VAlue of y is  20
 The Vlaue of b is  21
--------------------------------
```

UNARY DECREMENT OPERATOR:

It is used to decrement the value of the variable by 1. The decrement can be done in two ways:

➢ PRE DECREMENT:

In this method the operator precedes (--a). The value of operand will be altered before it is used.

➢ POST DECREMENT:

In this method the operator follows the operand (a--). The value of operand will be altered after it is used.

EXAMPLE :

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int x,y,a,b;
    a=20;
    x=--a;

    printf("PRE DECREMENT OPERATOR");

printf("\n The Value of x is %d",x);
    printf("\n The Value of a is %d",a);

    b=30;
    y=b--;

    printf("\n\n POST DECREMENT OPEARTOR");
    printf("\n The Value of y is %d",y);

    printf("\n The Value of b is %d",b);

    return 0;


}
```

OUTPUT:

```
PRE DECREMENT OPERATOR
 The Value of x is 19
 The Value of a is 19

 POST DECREMENT OPEARTOR
 The Value of y is 30
 The Value of b is 29
--------------------------------
```

UNARY SIZE OF () OPERATOR:

This operator returns the size of its operand, in bytes. The *sizeof* operator always precedes its operand.

EXAMPLE:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int x;
    float y;
    char ch;
    double z;


    printf("The size of int (x) variable is:  %d\n",sizeof(x));
    printf("The size of int (y) variable is:  %d\n",sizeof(y));
    printf("The size of int (ch) variable is:  %d\n",sizeof(ch));
    printf("The size of int (z) variable is:  %d\n",sizeof(z));
    return 0;


}
```

OUTPUT:

```
The size of int (x) variable is:  4
The size of int (y) variable is:  4
The size of int (ch) variable is:  1
The size of int (z) variable is:  8

--------------------------------
```

LOGICAL NOT (!):

It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

EXAMPLE;

```c
#include<stdio.h>
#include<stdbool.h>
int main()
{
    bool a=true;
    bool b;

    b=!a;

    printf("The boolean value of a is:  %d\n",a);
    printf("The boolean value of b is:  %d\n",b);

    bool c=0;
    bool d=!c;

    printf("\n The Boolean value of c is:  %d",c);
    printf("\n The Boolean value of d is:  %d",d);

    return 0;

}
```

OUTPUT:

```
The boolean value of a is:  1
The boolean value of b is:  0

 The Boolean value of c is:  0
 The Boolean value of d is:  1
--------------------------------
```

ADDRESS OF OPERATOR (&):

It gives an address of a variable. It is used to return the memory address of a variable. These addresses returned by the address-of operator are known as pointers because they "point" to the variable in memory.

EXAMPLE:

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=13;
    int b;

    b= &a;

    printf("The value of variable is:  %d\n",a);
    printf("The address of variable b is:  %d\n",b);

    return 0;

}
```

OUTPUT:

```
The value of variable is:  13
The address of variable b is:  6487576
```

ARITHMITICAL OPERATORS:

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	Division
%	remainder after division (modulo division)

EXAMPLE:

```c
#include<stdio.h>
int main()
{
    int a=9;
    int b=4;
    int c;

    c=a+b;
    printf("a+b = %d\n",c);

    c=a-b;
    printf("a-b = %d\n",c);

    c=a*b;
    printf("a*b = %d\n",c);

    c=a/b;
    printf("a/b = %d\n",c);

    c=a%b;
    printf("Reminder when divided by b:  %d\n",c);

    return 0;
}
```

OUTPUT:

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Reminder when divided by b:  1

--------------------------------
```

QUESTION 03:

INPUT:

Input means to provide the program with some data to be used in the program.

OUTPUT:

Output means to display the data on the screen.

The C programming language provides standard library functions to read any given input and to display data on the console.
The functions used for standard input and output are present in the stdio.h header file. Hence to use the functions we need to include the stdio.h header file in our program.

```
#include<stdio.h>
```

Following are the functions used for standard input and output:

➢ Printf() OUTPUT STATEMENT:

printf() function shows output.

The printf() function is the most used function in c language. This function is defined in the stdio.h header file and is used to show output on the console.

This function is used to print a simple text sentence or value of any variable which can be of int, char, float, or any other datatype.

EXAMPLE;

```c
#include<stdio.h>
int main()
{
    printf("THIS IS MY FIRST PROGRAMMING FUNDAMENTAL ASSIGNMENT");

    return 0;
}
```

Output:

```
THIS IS MY FIRST PROGRAMMING FUNDAMENTAL ASSIGNMENT
--------------------------------
Process exited after 0.02541 seconds with return value 0
Press any key to continue . . .
```

➢ Scanf() INPUT STATEMENT:

Scanf() function takes input:

If we have to take an integer value input from the user we have to define an integer variable and then use a scanf() function.

```c
#include<stdio.h>
int main()
{
    int user_input;

    printf("Please enter a number:  ");
    scanf("%d",&user_input);

    printf("YOU ENTERED:  %d",user_input);

    return 0;
}
```

```
Please enter a number:  45
YOU ENTERED:  45
--------------------------------
Process exited after 1.713 seconds with return value 0
Press any key to continue . . .
```

In the above code example, we have used %d format specifier to inform the scanf() function that user input will be of type integer.

QUESTION 01:

Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one:

➢ CHAR:

The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

➢ INT:

As the name suggests, an int variable is used to store an integer.

➢ FLOAT:

It is used to store decimal numbers (numbers with floating point value) with single precision.

➢ DOUBLE:

It is used to store decimal numbers (numbers with floating point value) with double precision.

EXAMPLE:

```c
#include <stdio.h>
int main()
{
    int a = 1;
    char b = 'G';
    double c = 3.14;
    printf("Hello World!\n");


    printf("Hello! I am a character. My value is %c and "
           "my size is %lu byte.\n",
           b, sizeof(char));


    printf("Hello! I am an integer. My value is %d and "
           "my size is %lu  bytes.\n",
           a, sizeof(int));


    printf("Hello! I am a double floating point variable."
           " My value is %lf and my size is %lu bytes.\n",
           c, sizeof(double));

    printf("Bye! See you soon. :)\n");

    return 0;
}
```

OUTPUT:

```
Hello World!
Hello! I am a character. My value is G and my size is 1 byte.
Hello! I am an integer. My value is 1 and my size is 4  bytes.
Hello! I am a double floating point variable. My value is 3.140000 and my size is 8 bytes.
Bye! See you soon. :)

--------------------------------
```

C Data types / storage Size	Range
char / 1	–127 to 127
int / 2	–32,767 to 32,767
float / 4	1E–37 to 1E+37 with six digits of precision
double / 8	1E–37 to 1E+37 with ten digits of precision
long double / 10	1E–37 to 1E+37 with ten digits of precision
long int / 4	–2,147,483,647 to 2,147,483,647
short int / 2	–32,767 to 32,767
unsigned short int / 2	0 to 65,535
signed short int / 2	–32,767 to 32,767
long long int / 8	–(2power(63) –1) to 2(power)63 –1
signed long int / 4	–2,147,483,647 to 2,147,483,647
unsigned long int / 4	0 to 4,294,967,295
unsigned long long int / 8	2(power)64 –1

THE END:)