Unit – 2

Operators

An operator is a symbol that tells to the compiler what operation is performed on the operands. C language is rich in built-in operators.

For example, '+' is an operator used for addition, as shown below:

c = a + b;

Here, '+' is the operator known as the addition operator and 'a' and 'b' are operands. The addition operator tells the compiler to add both of the operands 'a' and 'b'.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operator
- Shorthand Operator
- Comma Operator
- Increment and Decrement Operator
- Bitwise Operators
- Ternary or Conditional Operators

Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values.

Operators	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Example:

Suppose, a=20, b=3

Operand 1	Operand 2	Expression	Result
a	b	a+b	23
a	b	a-b	17
a	b	A*b	60
a	b	a/b	6
a	b	a%b	2

Program: WAP to perform arithmetic operations

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a=20, b=3;
        clrscr();
        printf("Addition is =%d", a+b);
        printf("\nSubtraction is =%d", a-b);
        printf("\nMultiplication is =%d", a*b);
        printf("\nDivision is =%d", a/b);
        printf("\nModulus is =%d", a%b);
        getch();
}
```

Output:

Addition is=23

Subtraction is =17

Multiplication is=60

Division=6

Modulus is=2

Relational Operators:

These types of operators check the relationship between two of the available operands. In case the relation happens to be true, then it returns to 1. But in case this relation turns out to be false, then it returns to the value 0. We use the relational operators in loops and in the cases of decision making.

Types of Relational Operators

If P=5 and Q=3, then:

Meaning	Operator	Details	Example
Equal to	= =	(P == Q) is not true	4 == 5 gets evaluated to 0 4 == 4 gets evaluated to 1
Not equal to	!=	(P != Q) is true	4 != 5 gets evaluated to 1 4 != 4 gets evaluated to 0
Less than	<	(P < Q) is not true	4 < 2 gets evaluated to 0 4 < 6 gets evaluated to 1
Greater than	>	(P > Q) is true	4 > 2 gets evaluated to 1 4 > 9 gets evaluated to 0
Less than or equal to	<=	(P <= Q) is not true	4 <= 2 gets evaluated to 0 4 <= 6 gets evaluated to 1 4 <= 4 gets evaluated to 1
Greater than or equal to	>=	(P >=Q) is true	4 >= 2 It gets evaluated to 1 4 >= 9 It gets evaluated to 0 4 <= 4 gets evaluated to 1

Here, 0 means false and 1 means true.

Program of Relational Operators

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

```
int main()
{
int p = 9;
int b = 4;
printf(“p > q: %d \n”, p > q);
printf(“p >= q: %d \n”, p >= q);
printf(“p <= q: %d \n”, p <= q);
printf(“p < q: %d \n”, p < q);
printf(“p == q: %d \n”, p == q);
printf(“p != q: %d \n”, p != q);
}
```

The output generated here would be:

p > q: 1

p >= q: 1

p <= q: 0

p < q: 0

p == q: 0

p != q: 1

Logical Operators

Logical operators perform logical operations on a given expression by joining two or more expressions or conditions. It can be used in various relational and conditional expressions. This operator is based on Boolean values to logically check the condition, and if the conditions are true, it returns 1. Otherwise, it returns 0 (False).

Types of Logical operator:

Operator	**Meaning**
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical AND Operator

The logical AND operator is represented as the '&&' double ampersand symbol. It checks the condition of two or more operands by combining in an expression, and if all the conditions are true, the logical AND operator returns the Boolean value true or 1. Else it returns false or 0.

Syntax

(condition1 && condition2)

There are two conditions in the above syntax, condition1 and condition2, and in between the double (&&) ampersand symbol. If both the conditions are true, the logical AND operator returns Boolean value 1 or true. Otherwise, it returns false.

Truth table of the Logical AND (&&) operator

A	B	A && B
1	1	1
1	0	0
0	1	0
0	0	0

Program to demonstrate the Logical AND Operator in C

```
#include <stdio.h>
#include <conio.h>
int main ()
{
   int n = 20;
   clrscr();
   printf (" %d \n", (n == 20 && n >= 8));
   printf (" %d \n", (n >= 1 && n >= 20));
   printf (" %d \n", (n == 10 && n >= 0));
   printf (" %d \n", (n >= 20 && n <= 40));
  getch();
}
```

Output

```
1
1
0
1
```

Logical OR Operator

The logical OR operator is represented as the '||' double pipe symbol. It checks the condition of two or more operands by combining in an expression, and if all the conditions are false, the logical OR operator returns the Boolean value false or 0. Else it returns true or 1.

Syntax

(condition1 || condition2)

There are two conditions in the above syntax, condition1 and condition2, and in between the double (||) pipe symbol. If both the conditions are false, the logical OR operator returns Boolean value 0 or false. Otherwise, it returns true.

Truth table of the Logical OR (||) operator

A	B	A && B

1	1	1
1	0	1
0	1	1
0	0	0

Program to demonstrate the Logical OR Operator in C

```
#include <stdio.h>
#include <conio.h>
int main ()
{
   int n = 20;
   clrscr();
   printf (" %d \n", (n == 20 || n >= 8));
   printf (" %d \n", (n >= 1 || n >= 20));
   printf (" %d \n", (n == 10 || n >= 0));
   printf (" %d \n", (n > 30 || n <= 40));
  getch();
}
```

Output

```
1
1
1
0
```

Logical NOT operator

The logical NOT operator is represented as the '!' symbol, which is used to reverse the result of any given expression or condition. If the result of an expression is non-zero or true, the

result will be reversed as zero or false value. Similarly, if the condition's result is false or 0, the NOT operator reverses the result and returns 1 or true.

For example, suppose the user enters a non-zero value is 5, the logical NOT (!) operator returns the 0 or false Boolean value. And if the user enters a zero (0) value, the operator returns the true Boolean value or 1.

Syntax of the logical NOT operator

! (condition);

Here, the '!' symbol represents the logical NOT operator, which inverses the result of the given condition.

The truth table of the logical NOT operator:

Following is the truth table of the logical not operator in C

Condition	!(condition)
1	0
0	1

Program to use the logical NOT operator in C

```
#include <stdio.h>
#include <conio.h>
void main ()
{
   int x = 5;
   clrscr();
   printf (" The  value is = %d", ! (x == 5));
   printf (" \n The  value = %d", ! (x != 5));
   printf (" \n The value = %d", ! (x >= 3));
   printf (" \n The value = %d", ! (x < 3));
  getch();
    }
```

Output:

```
The value = 0
The value = 1
The value = 0
The value = 1
```

Assignment Operator

The assignment operator is used to assign the value, variable and function to another variable.

A = 5; // use Assignment symbol to assign 5 to the operand A

B = A; // Assign operand A to the B

B = &A; // Assign the address of operand A to the variable B

A = 20 \ 10 * 2 + 5; // assign equation to the variable A

Simple Assignment Operator (=):

It is the operator used to assign the right side operand or variable to the left side variable.

Syntax

int a = 5;

or

int b = a;

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
   int n1, n2, c, x, y;
   clrscr();
   n1 = 5;
   n2 = n1;
   c = n1 + n2;
   x = 20 / 4 * 2 + 5;
   printf (" \n The value of n1: %d", n1);
```

```
    printf (" \n The value of n2: %d", n2);
    printf (" \n The value of c: %d", c);
    printf (" \n The value of x: %d", x);
    getch();
}
```

Output

```
The value of n1: 5
The value of n2: 5
The value of c: 10
The value of x: 15
```

Shorthand Assignment Operators

Plus and Assign Operator (+=):

The operator is used to add the left side operand to the left operand and then assign results to the left operand.

Syntax

A += B;

Or

A = A + B;

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
      int n1, n2, c;
clrscr();
n1 = 5;
    n2 = 10;
    n2 += n1;
    printf (" \n The value of n1: %d", n1);
    printf (" \n The value of n2: %d", n2);
```

```
getch();
}
```

Output

```
The value of a: 5
 The value of b: 15
```

Subtract and Assign Operator (-=):

The operator is used to subtract the left operand with the right operand and then assigns the result to the left operand.

Syntax

```
A -= B;
Or
A = A - B;
```

Program

```
#include <stdio.h>
#include <conio.h>
int main ()
{
   int n1, n2, c;
clrscr();
   n1 = 5;
   n2 = 10;
   n2 -= n1;
   printf (" \n The value of n1: %d", n1);
   printf (" \n The value of n2: %d", n2);

getch();
}
```

Output

```
The value of n1: 5
The value of n2: 5
```

Multiply and Assign Operator (*=)

The operator is used to multiply the left operand with the right operand and then assign result to the left operand.

Syntax

A *= B;

Or

A = A * B;

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
   int n1, n2, c;
clrscr();
n1 = 5;
   n2 = 10;
   n2 *= n1;
   printf (" \n The value of n1: %d", n1);
   printf (" \n The value of n2: %d", n2);

getch();
}
```

Output

```
The value of n1: 5
 The value of n2: 50
```

Divide and Assign Operator (/=):

An operator is used between the left and right operands, which divides the first number by the second number to return the result in the left operand.

Syntax

A /= B;
Or
A = A / B;

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
   int n1, n2, c;
clrscr();
n1 = 5;
   n2 = 10;
   n2 /= n1;
   printf (" \n The value of n1: %d", n1);
   printf (" \n The value of n2: %d", n2);
getch();
}
```

Output

```
The value of n1: 5
 The value of n2: 2
```

Modulus and Assign Operator (%=):

An operator used between the left operand and the right operand divides the first number (n1) by the second number (n2) and returns the remainder in the left operand.

Syntax

A %= B;
Or
A = A % B;

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
   int n1, n2, c;
   clrscr();
   printf (" Enter the value of n1: ");
   scanf ("%d", &n1);
   printf (" \n Enter the value of n2: ");
   scanf ("%d", &n2);
   n1 %= n2;
   printf (" \n The modulus value of n1: %d", n1);
getch();
}
```

Output

```
Enter the value of n1: 23

 Enter the value of n2: 5

 The modulus value of n2: 3
```

sizeof() operator

The **sizeof()** operator is commonly used in C. It determines the size of the expression or the data type specified in the number of char-sized storage units. The **sizeof()** operator contains a single operand which can be either an expression or a data typecast where the cast is data type enclosed within parenthesis.

```
#include <stdio.h>
#include<conio.h>
void main()
{
  clrscr();
     printf("\nsize of the character data type is %d",sizeof(char));
    printf("\nsize of the integer data type is %d",sizeof(int));
    printf("\nsize of the floating data type is %d",sizeof(float));
```

```
    printf("\nsize of the double data type is %d",sizeof(double));

getch();
}
```

Output:

size of the character data type is 1

size of the integer data type is 2

size of the float data type is 4

size of the double data type is 8

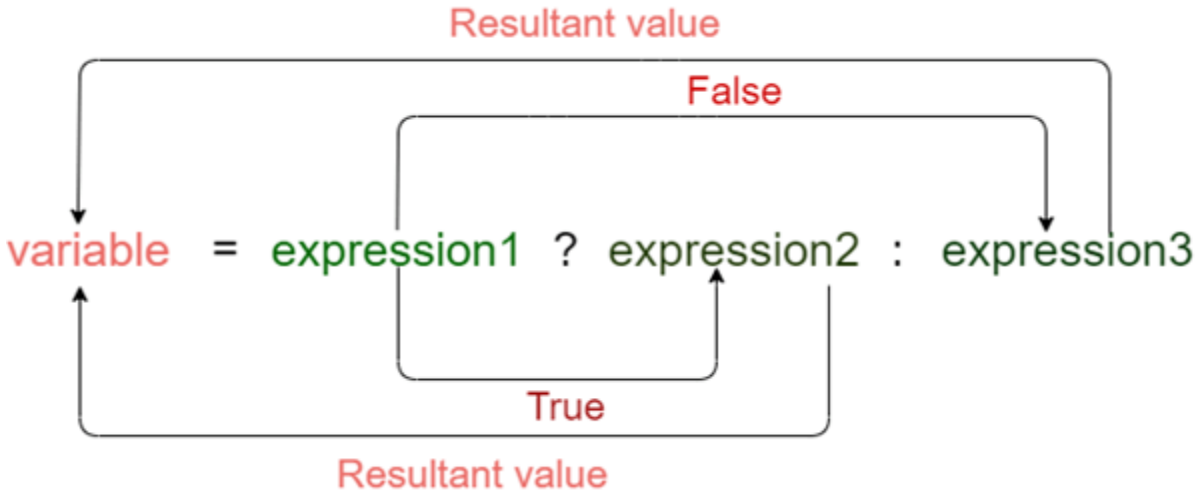
Conditional Operator

The conditional operator is also known as a **ternary operator**. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and ':'.

As conditional operator works on three operands, so it is also known as the ternary operator.

Syntax of a conditional operator

Expression1? expression2: expression3;



Meaning of the above syntax.

- In the above syntax, the expression1 is a Boolean condition that can be either true or false value.
- If the expression1 results into a true value, then the expression2 will execute.
- The expression2 is said to be true only when it returns a non-zero value.
- If the expression1 returns false value then the expression3 will execute.
- The expression3 is said to be false only when it returns zero value.

Program:

```
#include <stdio.h>
#include<conio.h>
    void main()
    {
            int a,b,c;
            clrscr();
            printf("Enter two numbers =");
            scanf("%d%d",&a,&b);
            c= a>b ? a : b;
            printf("The greater number is =%d",c);
            getch();
}
```

Program:

```
#include <stdio.h>
#include<conio.h>
    void main()
    {
            int a,b,c;
            clrscr();
            printf("Enter two numbers =");
            scanf("%d%d",&a,&b);
            c= a<b ? a : b;
            printf("The smaller number is =%d",c);
            getch();
}
```

Program:

```
#include <stdio.h>
#include<conio.h>
    void main()
    {
       int age;
       clrscr();
       printf("Enter your age");
       scanf("%d",&age);
       (age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting"));
    getch();
    }
```

Comma Operator

The comma sign is used for mainly two different purposes in the C language – as an operator and as a separator.

The Comma as a Separator

When we want to declare multiple numbers of variables in any program and provide different arguments in any function, we use the comma in the form of a separator in the program.

For instance,

int x, y, z;

In the statement mentioned above, the comma acts as a separator and informs the compiler that the x, y, and z variables are three different types of variables.

The Comma as an Operator

We use the comma in the form of an operator when we want to assign multiple numbers of values to any variable in a program with the help of a comma.

For example,

int a = (40, 50, 60, 70, 80, 90);

The value of a will be equivalent to 90. It is because the values 40, 50, 60, 70, 80 and 90 are enclosed in the form of braces () and these braces have a higher priority, as compared to the equal to (=) assignment operator.

Increment and Decrement Operators

Increment Operator

Increment Operators are the unary operators used to increment or add 1 to the operand value. The Increment operand is denoted by the double plus symbol (++). It has two types, Pre Increment and Post Increment Operators.

Pre-increment Operator

The pre-increment operator is used to increase the original value of the operand by 1 before assigning it to the expression.

Syntax

X = ++A;

In the above syntax, the value of operand 'A' is increased by 1, and then a new value is assigned to the variable 'B'.

Example 1: Program to use the pre-increment operator in C

```
#include <stdio.h>
#include <conio.h>
void main ()
{
int x, y, z;
printf (" Input the value of X: ");
scanf (" %d", &x);
printf ("  Input the value of Y: ");
scanf (" %d", &y);
printf (" Input the value of Z: ");
scanf (" %d", &z);
++x;
++y;
++z;
printf (" \n The updated value of the X: %d ", x);
printf (" \n The updated value of the Y: %d ", y);
printf (" \n The updated value of the Z: %d ", z);
getch();
```

}

Output

```
Input the value of X: 10
Input the value of Y: 15
Input the value of Z: 20

The updated value of the X: 11
The updated value of the Y: 16
The updated value of the Z: 21
```

Post increment Operator

The post-increment operator is used to increment the original value of the operand by 1 after assigning it to the expression.

Syntax

1. X = A++;

   In the above syntax, the value of operand 'A' is assigned to the variable 'X'. After that, the value of variable 'A' is incremented by 1.

Example 2: Program to use the post-increment operator in C

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

```
void main ()
{
int x, y, z, a, b, c;
printf (" Input the value of X: ");
scanf (" %d", &x);
printf (" Input the value of Y: ");
scanf (" %d", &y);
printf (" Input the value of Z: ");
scanf (" %d", &z);
a = x++;
b = y++;
c = z++;
printf (" \n The original value of a: %d", a);
printf (" \n The original value of b: %d", b);
printf (" \n The original value of c: %d", c);
printf (" \n\n The updated value of the X: %d ", x);
printf (" \n The updated value of the Y: %d ", y);
printf (" \n The updated value of the Z: %d ", z);
getch();
}
```

Output

```
Input the value of X: 10
Input the value of Y: 15
Input the value of Z: 20

The original value of a: 10
The original value of b: 15
The original value of c: 20

The updated value of the X: 11
The updated value of the Y: 16
The updated value of the Z: 21
```

Decrement Operator

Decrement Operator is the unary operator, which is used to decrease the original value of the operand by 1. The decrement operator is represented as the double minus symbol (--). It has two types, Pre Decrement and Post Decrement operators.

Pre Decrement Operator

The Pre Decrement Operator decreases the operand value by 1 before assigning it to the mathematical expression. In other words, the original value of the operand is first decreases, and then a new value is assigned to the other variable.

Syntax

B = --A;

In the above syntax, the value of operand 'A' is decreased by 1, and then a new value is assigned to the variable 'B'.

Example 3: Program to demonstrate the pre decrement operator in C

```
#include <stdio.h>
#include <conio.h>
void main ()
{
int x, y, z;
printf (" Input the value of X: ");
scanf (" %d", &x);
printf (" \n Input the value of Y: ");
scanf (" %d", &y);
printf ("n Input the value of Z: ");
scanf (" %d", &z);
--x;
--y;
--z;
printf (" \n The updated value of the X: %d ", x);
printf (" \n The updated value of the Y: %d ", y);
printf (" \n The updated value of the Z: %d ", z);
```

```
getch();
}
```

Output

```
Input the value of X: 5
Input the value of Y: 6
Input the value of Z: 7

The updated value of the X: 6
The updated value of the Y: 7
The updated value of the Z: 8
```

Post decrement Operator:

Post decrement operator is used to decrease the original value of the operand by 1 after assigning to the expression.

Syntax

1. B = A--;

In the above syntax, the value of operand 'A' is assigned to the variable 'B', and then the value of A is decreased by 1.

Example 4: Program to use the post decrement operator in C

```
#include <stdio.h>
#include <conio.h>
void main ()
{
int x, y, z, a, b, c;
printf (" Input the value of X: ");
scanf (" %d", &x);
printf (" Input the value of Y: ");
scanf (" %d", &y);
printf (" Input the value of Z: ");
scanf (" %d", &z);
a = x--;
```

```
b = y--;
c = z--;
printf (" \n The original value of a: %d", a);
printf (" \n The original value of b: %d", b);
printf (" \n The original value of c: %d", c);
printf (" \n\n The updated value of the X: %d ", x);
printf (" \n The updated value of the Y: %d ", y);
printf (" \n The updated value of the Z: %d ", z);
getch();
}
```

Output

```
Input the value of X: 6
Input the value of Y: 12
Input the value of Z: 18

The original value of a: 6
The original value of b: 12
The original value of c: 18

The updated value of the X: 5
The updated value of the Y: 11
The updated value of the Z: 17
```

Bitwise Operator in C

The bitwise operators are the operators used to perform the operations on the data at the bit-level. When we perform the bitwise operations, then it is also known as bit-level programming. It consists of two digits, either 0 or 1. It is mainly used in numerical computations to make the calculations faster.

We have different types of bitwise operators in the C programming language. The following is the list of the bitwise operators:

Operator	Meaning of operator
&	Bitwise AND operator

|	Bitwise OR operator
^	Bitwise exclusive OR (XOR) operator
~	One's complement operator or Bitwise Not
<<	Left shift operator
>>	Right shift operator

Bitwise AND operator

Bitwise AND operator is denoted by the single ampersand sign (&). Two integer operands are written on both sides of the (&) operator. If the corresponding bits of both the operands are 1, then the output of the bitwise AND operation is 1; otherwise, the output would be 0.

For example,

We have two variables a and b.

a =6;

b=4;

The binary representation of the above two variables are given below:

a = 00000000 00000110

b = 00000000 00000100

When we apply the bitwise AND operation in the above two variables, i.e., a&b, the output would be:

Result = 0100

As we can observe from the above result that bits of both the variables are compared one by one. If the bit of both the variables is 1 then the output would be 1, otherwise 0.

Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a=6, b=14,c;
```

```
  clrscr();
  c=a&b;
  printf("The output of the Bitwise AND operator is %d",c);
getch();
}
```

Output:

The output of the Bitwise AND operator is 6

Bitwise OR operator

The bitwise OR operator is represented by a single vertical sign (|). Two integer operands are written on both sides of the (|) symbol. If the bit value of any of the operand is 1, then the output would be 1, otherwise 0.

For example,

We consider two variables,

a = 23;

b = 10;

The binary representation of the above two variables would be:

a = 00000000 00010111

b = 00000000 00001010

When we apply the bitwise OR operator in the above two variables, i.e., a|b , then the output would be:

Result = 00000000 00011111

Program:

```
#include <stdio.h>
int main()
{
   int a=23,b=10, c;
  clrscr();
  c=a|b;
```

```
   printf("The output of the Bitwise OR operator is %d",c);
getch();
}
```

Output:

The output of the Bitwise OR operator is 31

Bitwise exclusive OR (XOR) operator

Bitwise exclusive OR operator is denoted by (^) symbol. Two operands are written on both sides of the exclusive OR operator. If the corresponding bit of any of the operand is 1 then the output would be 1, otherwise 0.

For example,

We consider two variables a and b,

a = 12;

b = 10;

The binary representation of the above two variables would be:

a = 00000000 00001100

b = 00000000 00001010

When we apply the bitwise exclusive OR operator in the above two variables (a^b), then the result would be:

Result = 00000000 00001110

Program

```
#include <stdio.h>
#include <stdio.h>
void main()
{
  int a=12,b=10, c;
  clrscr();
```

```
  c=a^b;
  printf("The output of the Bitwise XOR operator is %d",c);
 getch();
}
```

Output

The output of the Bitwise XOR operator is 31

Bitwise complement operator

The bitwise complement operator is also known as one's complement operator. It is represented by the symbol tilde (~). It takes only one operand or variable and performs complement operation on an operand. When we apply the complement operation on any bits, then 0 becomes 1 and 1 becomes 0.

For example,

If we have a variable named 'a',

a = 8;

The binary representation of the above variable is given below:

a = 1000

When we apply the bitwise complement operator to the operand, then the output would be:

Result = 0111

Program

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a=8,b;
  clrscr();
  b=~a;
  printf("The output of the Bitwise NOT operator is %d",b);
```

```
getch();
}
```

Output

The output of the Bitwise NOT operator is -9

Bitwise shift operators

Two types of bitwise shift operators exist in C programming. The bitwise shift operators will shift the bits either on the left-side or right-side. Therefore, we can say that the bitwise shift operator is divided into two categories:

- Left-shift operator
- Right-shift operator

Left-shift operator

It is an operator that shifts the number of bits to the left-side.

Syntax of the left-shift operator is given below:

1. Operand << n

Where,

Operand is an integer expression on which we apply the left-shift operation.

n is the number of bits to be shifted.

For example,

Suppose we have a statement:
int a = 5;
The binary representation of 'a' is given below:
a = 00000000 00000101
If we want to leftshift the above representation by 2, then the statement would be:
a << 2;
00000000 00000101<<2 = 00000000 00010100

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

```
#include <conio.h>
void main()
{
   int a=5,b; // variable initialization
   clrscr();
   b=a<<2;
   printf("The value of a<<2 is : %d ", b);
getch();
}
```

Output

The value of a<<2 is : 20

Right-shift operator

It is an operator that shifts the number of bits to the right side.

Syntax of the right-shift operator is given below:

Operand >> n;

Where,

Operand is an integer expression on which we apply the right-shift operation.

N is the number of bits to be shifted.

For example,

Suppose we have a statement,
int a = 7;
The binary representation of the above variable would be:
a = 00000000 00000111
If we want to rightshift the above representation by 2, then the statement would be:
a>>2;
00000000 00000111 >> 2 = 00000000 00000001

Program

```
#include <stdio.h>
int main()
{
    int a=7,b;
   clrscr();
b=a>>2;
    printf("The value of a>>2 is : %d ",b);
getch();
}
```

Output

The value of a>>2 is : 1

What is a Flowchart?

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as “flowcharting”.

Rules or guidelines of Flow chart:

The various Rules or Guidelines for drawing the flowchart are given below–

- Only conventional flowchart symbols should be used.
- Proper use of names and variables in the flowchart.
- If the flowchart becomes large and complex, use connector symbols.
- Flowcharts should have start and stop points.

Flowchart symbols:

The different flowchart symbols have different conventional meanings.

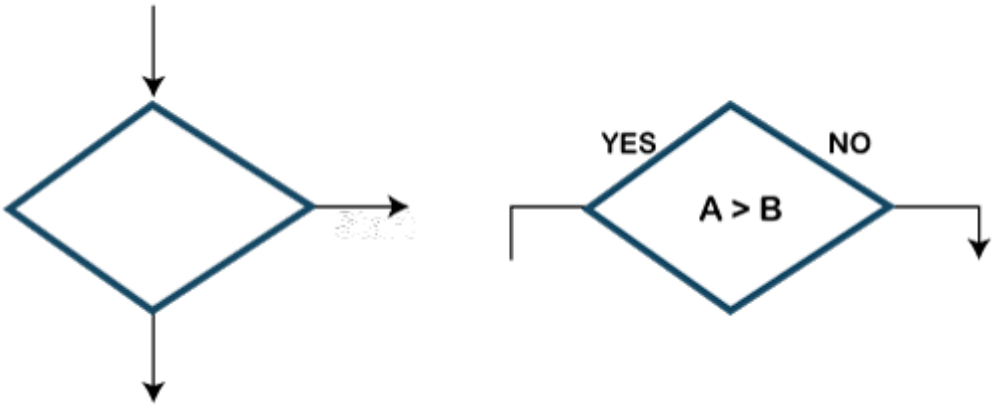
The various symbols used in Flowchart Designs are given below.

- **Terminal Symbol:** In the flowchart, it is represented with the help of a circle for denoting the start and stop symbol. The symbol given below is used to represent the terminal symbol.

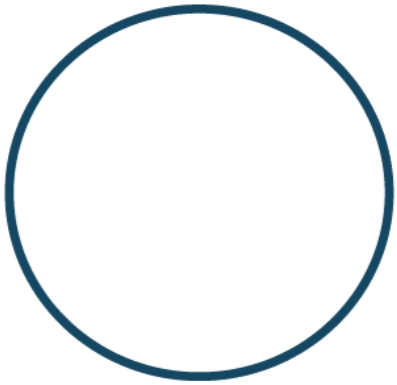
- **Input/output Symbol:** The input symbol is used to represent the input data, and the output symbol is used to display the output operation. The symbol given below is used for representing the Input/output symbol.

- **Processing Symbol:**It is represented in a flowchart with the help of a rectangle box used to represent the arithmetic and data movement instructions. The symbol given below is used to represent the processing symbol.

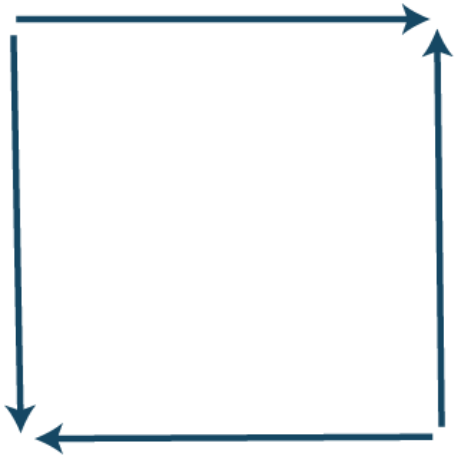
- **Decision Symbol:** Diamond symbol is used for represents decision-making statements. The symbol given below is used to represent the decision symbol.



- **Connector Symbol:**The connector symbol is used if flows discontinued at some point and continued again at another place. The following symbol is the representation of the connector symbol.



- **Flow lines:** It represents the exact sequence in which instructions are executed. Arrows are used to represent the flow lines in a flowchart. The symbol given below is used for representing the flow lines:



Advantages of Flowchart in C:

Following are the various advantages of flowchart:

- **Communication:** A flowchart is a better way of communicating the logic of a program.
- **Synthesis:** Flowchart is used as working models in designing new programs and software systems.
- **Efficient Coding:** Flowcharts act as a guide for a programmer in writing the actual code in a high-level language.

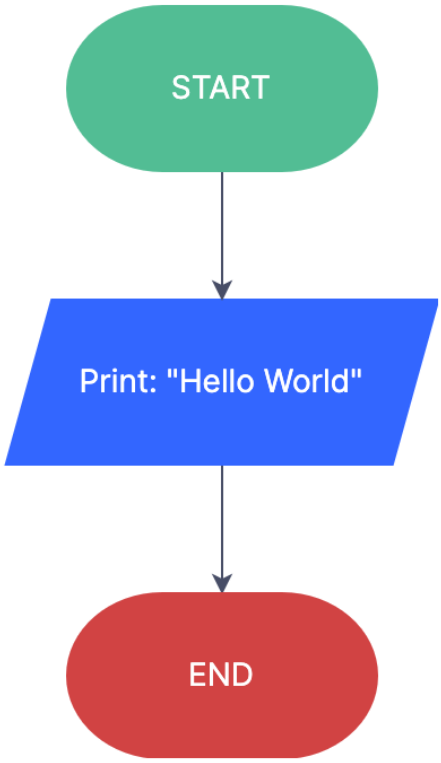
- **Proper Debugging:** Flowcharts help in the debugging process.
- **Effective Analysis:** Effective analysis of logical programs can be easily done with the help of a related flowchart.
- **Proper Documentation:** Flowchart provides better and proper documentation. It consists of various activities such as collecting, organizing, storing, and maintaining all related program records.
- **Testing:** A flowchart helps in the testing process.
- **Efficient program maintenance:** The maintenance of the program becomes easy with the help of a flowchart.

Disadvantages of Flowchart in C:

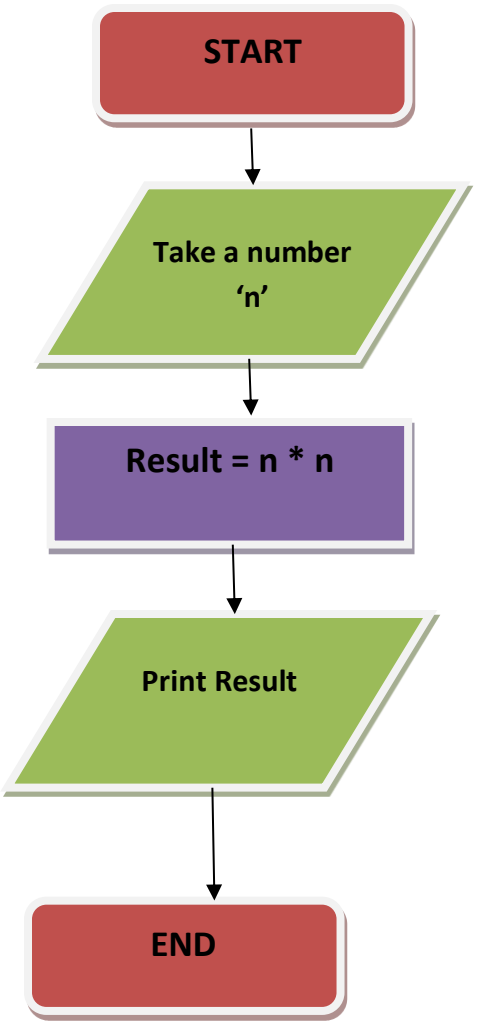
Following are the various disadvantages of flowchart:

- **Time-consuming:** Designing **a** flowchart is a very time-consuming process.
- **Complex:** It isn't easy to draw a flowchart for large and complex programs.
- **There is no standard** in the flowchart; there is no standard to determine the quantity of detail.
- **Difficult to modify:** It is very difficult to modify the existing flowchart.

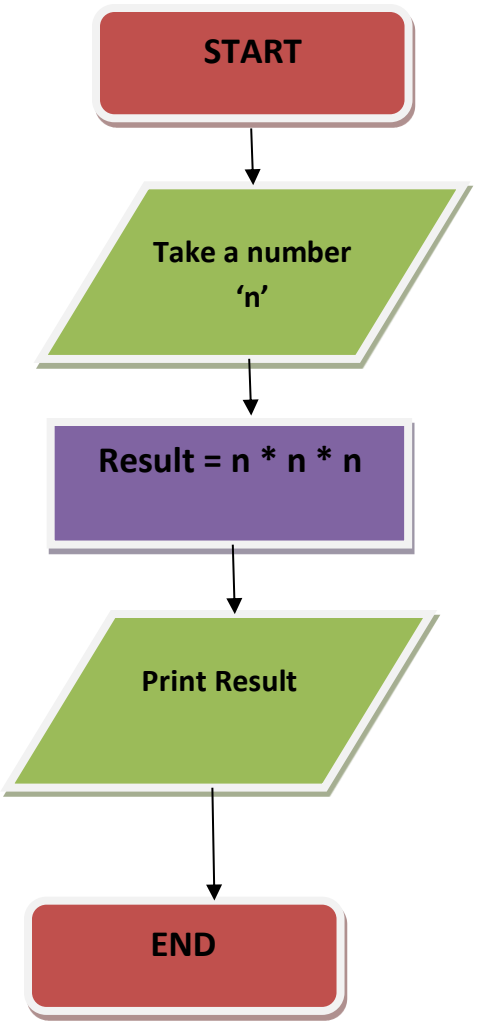
1. **Draw a flowchart to print "Hello World" Message**


START
Print: "Hello World"
END

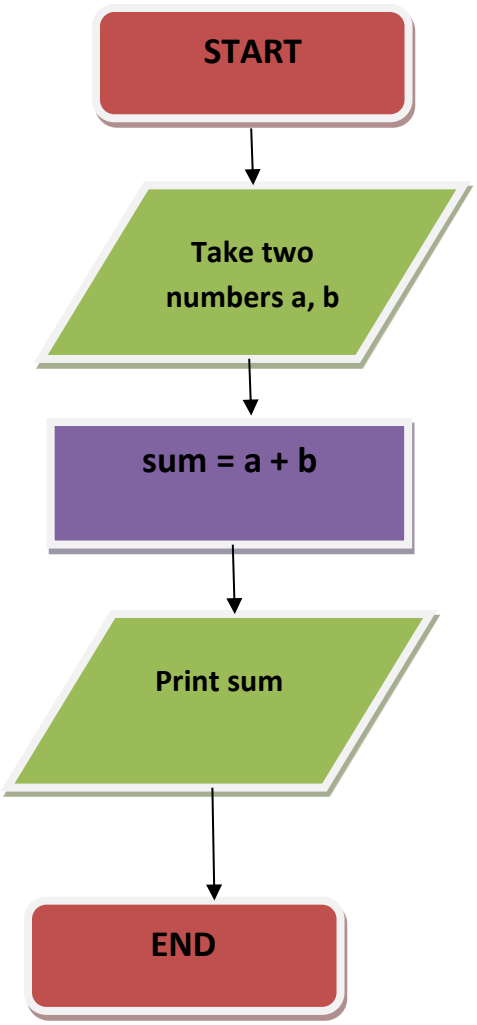
2. Draw a flowchart to print the square of a given number



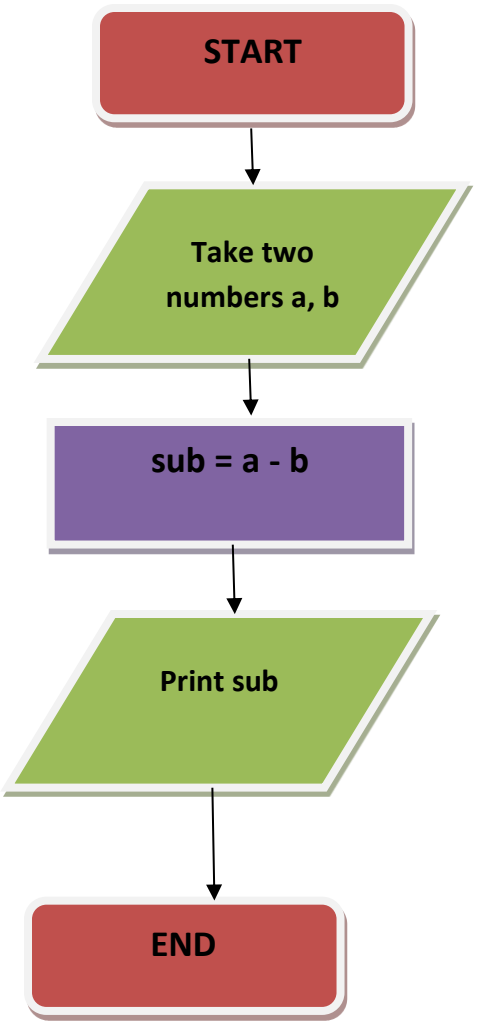
3. **Draw a flowchart to print the cube of a given number**



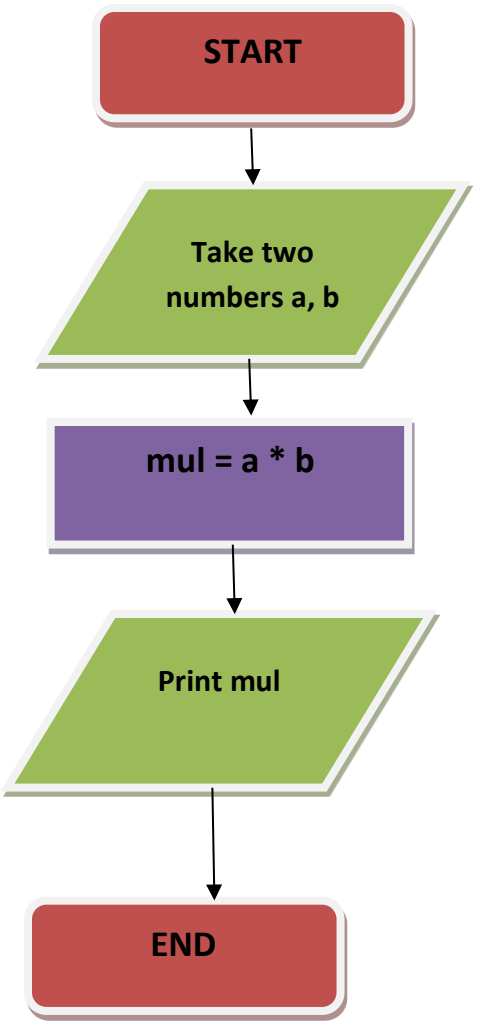
4. Draw a flowchart to print the addition of two numbers



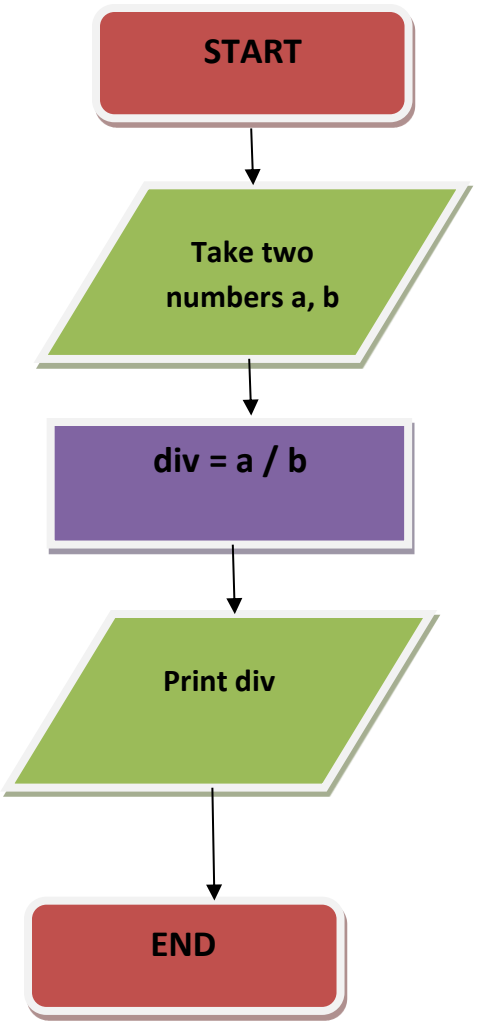
5. **Draw a flowchart to print the subtraction of two numbers**



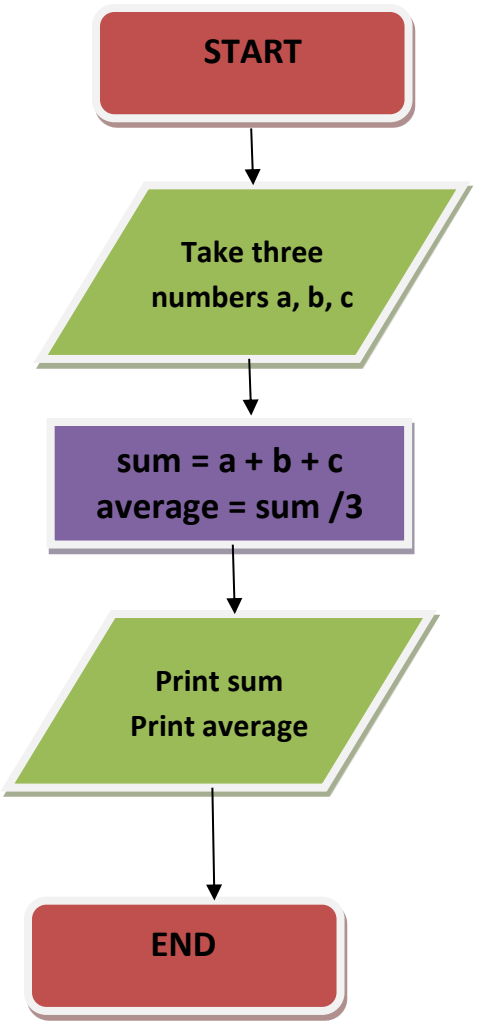
6. **Draw a flowchart to print the multiplication of two numbers**



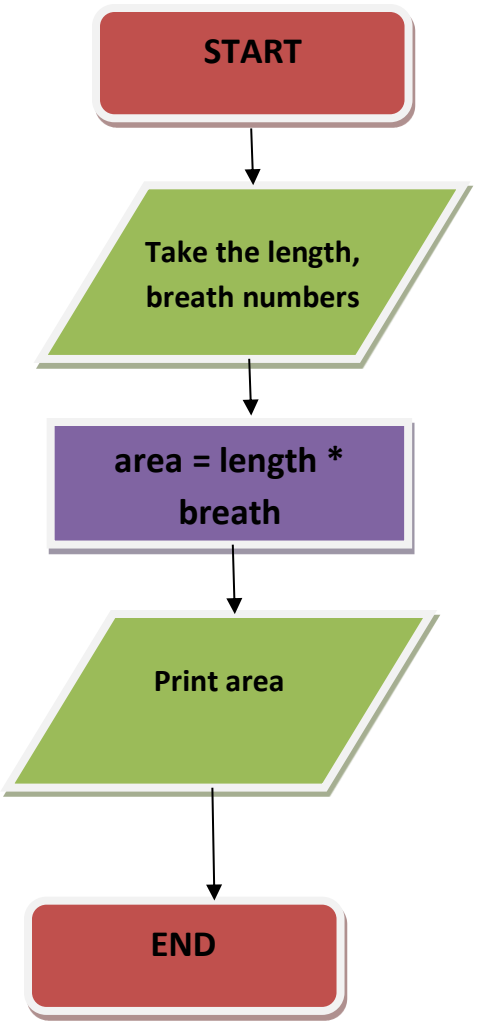
7. **Draw a flowchart to print the division of two numbers**



8. **Draw a flowchart to print the average of three numbers**



9. Draw a flowchart to calculate the area of rectangle



10. Draw a flowchart to calculate the perimeter of rectangle