OPERATOR

An operator is a symbol that specifies an operation to be performed.

1. Arithmetic Operator
2. Relational Operator
3. Conditional Operator
4. Logical Operator
5. Increment/Decrement Operator
6. Assignment Operator
7. Comma Operator
8. Size of Operator
9. Bitwise Operator
10. Scope resolution operator

Arithmetic Operators

Arithmetic operations are the basic and common operations performed using any computer programming. Normally, these operators are considered as basic operators and known as numeric operators as they require two variables to be evaluated.

OPERATOR	MEANING
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo(remainder of an integer division)

Thus there are few implicit conversion rules

OPERAND1	OPERAND2	RESULT
int	int	int
int	float	float
float	int	float
float	float	float
float	double	double
double	float	double
int	double	double
double	int	double

double	double	double

Arithmetic operators as per precedence:

L-R	Left to Right
( )	For grouping the variables
*, /, %	Multiplication, division and modulo
+ & -	Addition and subtraction

Program to find the area of the square

```
void main( )
{
  float a,area;
  clrscr( );
  cin>>a;
 area=a*a;
  cout<<"Area="<<area;
  getch( );
}

OUTPUT
5.1
Area=26.01
```

Program to find the area of the circle

```
main()
{
  float r,area;
  clrscr( );
  cin>>r;
  area=3.14*r*r;
  cout<<"Area="<<area;
  getch();
}

OUTPUT:

2.5
Area=19.642857
```

Relational Operator

Relational operators compare values to see if they are equal or if one of them is greater than the other and so on.

OPERATOR	MEANING
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to

!=	Not equal to

The relational operators are represented in the following syntax:

Expression1 relational_operator expression2

It returns 1 if the condition is true. It returns 0 if the condition is false.

Conditional Operator

The conditional operators ? and : are sometimes called ternary operators since they take three operands. Their general form is

Expression1 ? expression2: expression3

If the expression1 is true (that is, if its value is non zero), then the value returned would be expression2 otherwise the value returned will be expression3.

Program:

```
main()
{
  int a,b,big;
  clrscr( );
  cin>>a>>b;
  big=(a>b)?a:b;
        cout<<"Big="<<big;
  getch();
}
```

```
OUTPUT:
      56
      89
      Big=89
```

Logical Operator

OPERATOR	MEANING
&&	Logical and
||	Logical or
!	Not

(i) logical and

A compound expression is true when two conditions (expressions) are true.

(ii) logical or

A compound expression is true when any condition (expression) is true.

(iii) logical not

A logical expression can be changed from false to true or from true to false with the negation operator !.

Program

```
main()
{
  int m1,m2,m3;
  char result;
  cin>>m1>>m2>>m3;
  result=(m1>=40 && m2>=40 && m3>=40)?'p':'f';
  cout<<"Result="<<result;
  getch( );
}
```

OUTPUT

```
56
89
34
Result=f
```

Increment / Decrement operator

C++ offers two special operators ++ and -- called increment and decrement operators, respectively. These are ***'unary'*** operators since they operate on only one operand. The operand has to be a variable and not a constant.

Increment operator

There are two types of increment operators: ***prefix increment*** and ***postfix increment***. In prefix incrementer, first it is incremented and then the operations are performed. On the other hand, in the postfix incrementer, first the operations are performed and then it is incremented. However, the result of the incremented value will be the same in both the cases.

Assignment operator

An assignment operator is used to assign back to a variable, a modified value of the present holding.

1. **+=**: Value of LHS variable will be added to the value of RHS and assign it back to the variable in LHS.
2. **-=**: Value of RHS variable will be subtracted from the value of LHS and assign it back to the variable in LHS.
3. ***=**: Value of LHS variable will be multiplied by the value of RHS and assign it back to the variable in LHS.
4. **/=**: Value of LHS variable will be divided by the value of RHS and assign it back to the variable in LHS.
5. **%=**: The remainder will be stored back to the LHS after integer division is carried out between the LHS and the RHS variable.

The comma operator

The comma operator (**,**) permits two different expressions to appear in situations where only one expression would ordinarily be used. The comma operator separates the expressions. The following program illustrates the use of comma operator.

```
main()
{
 int a,b,c;
 clrscr( );
 c=(a=10,b=20,a+b);
 cout<<"Sum="<<c;
 getch();
}
OUTPUT
Sum=30
```

Sizeof operator

The **sizeof** operator returns the number of bytes the operand occupies in memory. The operand may be a variable, a constant or a data type qualifier.

Bitwise operator

Unlike some other languages, C++ provides and supports a full complement of bitwise operators. These are used for manipulation of data at bit level. The term bitwise operation refers to the testing, setting or shifting of the actual bits in a byte or word.

(i) Bitwise logical operator

bitwise **AND** bitwise **OR** bitwise **exclusive OR**

Bitwise AND

The bitwise **AND** will be carried out by the notation **&**. To generate a 1 bit in the result, bitwise **AND** need a one in both numbers.

Bitwise OR

The bitwise **OR** operations are similar to the bitwise **AND** and the result is 1 if any one of the bit value is 1. The symbol **|** represents the bitwise **OR**.

Bitwise exclusive OR

The bitwise **exclusive OR** will be carried out by the notation **^**. To generate a 1 bit in the result, a bitwise **exclusive OR** needs a one in either number but not both.

Op1	Op2	Op1&op2	Op1|op2	Op1^op2
0	0	0	0	0

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

(ii) Bitwise shift operator

The shift operations take binary patterns and shift the bits to the left or right, keeping the same number of bits by dropping shifted bits off the end and filling in with zeros from the other end. C++ provides two types of shift operations, left shift and right shift.

(i) ***Left shift:*** The **<<** operator is used for left shifting.

(ii) ***Right shift:*** The right shift **>>** operator is used for right shifting.

(iii) One's complement operator

The complement operator **~** switches all the bits in a binary pattern, that is, all the zeros become ones and all the ones become zeros.

1. **Scope Resolution operator:**

This operator is used to access the global variable's value.

Program:

```
#include<iostream.h>
#include<conio.h>
int m=10;
void main()
{
 int m=5;
 clrscr();
 cout<<m<<endl;
 cout<<::m<<endl;
 getch();
}

output:
5
10
```