OPERATORS

An Operator specifies an operation to be performed that yields a value. The variables, constants can be joined by various operators to form an expression. An operand is a data item on which an operator acts. Some operators require two operands, while others act upon only one operand. C includes a large number of operators that fall under several different categories, which are as :-

1. Arithmetic operators
2. Assignment operator
3. Increment & decrement operator
4. Relational operator
5. Logical operator
6. Conditional operator
7. Sizeof operator
8. Bitwise operator

Conditional Operator

It is a ternary operator(? And :) which requires three expressions as operands. This is written as –

TestExpression ? Expression1 : Expression2

Firstly the TestExpression is evaluated.

i. If TestExpression is True (Non-Zero) , then expression1 is evaluated and it becomes the value of the overall conditional expression.
ii. If TestExpression is False(Zero) , then expression2 is evaluated and it become the value of overall conditional expression.

For example, consider this conditional expression –

a > b ? a : b

Here first the expression a > b is evaluated, if the value is true then the value of variable ‘a’ becomes the value of conditional expression otherwise the value of ‘b’ becomes the value of conditional expression.

Suppose a=5 and b=8, then

Max = a > b ? a : b;

First the expression ‘a>b’ is evaluated, since it is False so the value of ‘b’ becomes the value of conditional expression and it is assigned to variable ‘Max’.

Example : a<b ? printf(“a is smaller”) : printf(“b is smaller”);

Since a=5 and b=8 , therefore the expression a<b is True, so the first printf function is executed.

SizeOf Operator

SizeOf operator is an unary operator. This operator gives the size of its operand in terms of bytes. The operand can be a variable, constant or any datatype (int, float, etc.). For example sizeof(int) gives the bytes occupied by datatype i.e. 2.

Bitwise Operators

Bitwise operators are used for operations on individual bits. Bitwise operators operate on integers only. The bitwise operators are as –

Bitwise operator	Meaning
&	Bitwise AND
|	Bitwise OR
~	One's complement
<<	Left shift
>>	Right shift
^	Bitwise XOR

Here all the operators are binary , except the One's complement operator , which is unary.

Bitwise AND (&)

It is a binary operator and requires two operands. These operands are compared bitwise i.e. all the corresponding bits in both operands are compared. The resulting bit is 1, only when the bits in both operands are 1, otherwise it is 0.

Boolean table

Bit of operand1	Bit of operand2	Resulting Bit
0	0	0
0	1	0
1	0	0
1	1	1

Let us take a= 0110 and b=1011 are two integer variables. The result after performing bitwise AND operation is -

a 0110

b 1011

a&b 0010

Bitwise OR (|)

It is a binary operator and requires two operands. These operands are compared bitwise i.e. all the corresponding bits in both operands are compared. The resulting bit is 0, only when the bits in both operands are 0, otherwise it is 1.

Boolean table

Bit of operand1	Bit of operand2	Resulting Bit
0	0	0
0	1	1
1	0	1
1	1	1

Let us take a= 0110 and b=1011 are two integer variables. The result after performing bitwise OR operation is -

a 0110

b 1011

a | b 1111

Bitwise XOR (^)

It is a binary operator and requires two operands. The corresponding bits of both operands are compared and the resulting bit is 1, if bits of both operands have different value, otherwise it is 0.

Boolean table

Bit of operand1	Bit of operand2	Resulting Bit
0	0	0
0	1	1
1	0	1
1	1	0

Let us take a= 0110 and b=1011 are two integer variables. The result after performing bitwise OR operation is -

a 0110

b 1011

a ^ b 1101

One's Complement

It is a unary operator and requires only one operand. It negates the value of the bit. If the bit of the operand is 1 then the resulting bit is 0 and if the bit of the operand is 0 then the resulting bit is 1.

Boolean table

Bit of operand	Resulting Bit
0	1
1	0

Let us take a= 0110 an integer variables. The result after performing One's Complement operation is -

a 0110

~a 1001

Bitwise Left Shift (<<)

This operator is used for shifting the bits left. It requires two operands.the left operand is the operand whose bits are shifted and the right operand indicates the number of bits to be shifted. On shifting the bits

left, an equal number of bit positions on the right are vacated. These positions are filled in with 0 bits. Consider an integer variable a= 0001 0011 0100 0110 , then the result of a << 4 is

0011 0100 0110 **0000**

On shifting all bits to the left by 4 , the leftmost 4 bits are lost while the rightmost 4 bit positions become empty which are filled with 0 bits as shown above.

Bitwise Right Shift (>>)

This operator is similar to the left shift operator, except that it shifts the bits to the right side. On shifting the bits right, an equal number of bit positions on the left are vacated. These positions are filled in with 0 bits in unsigned integers. Consider an integer variable a= 0001 0011 0100 0110 , then the result of a >> 4 is

0000 0001 0011 0100

On shifting all bits to the right by 4 , the rightmost 4 bits are lost while the leftmost 4 bit positions become empty which are filled with 0 bits as shown above.

In right shift if the first operand is a signed integer, then the result is compiler dependent. Some compilers follow logical shift while others may follow arithmetic shift.

In logical shift , the vacated bits are always filled with zeros.

In arithmetic shift, the vacated bits are filled with the value of the leftmost bit in the initial pattern.

Precedence of Operators

If more than one operators are involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence.
Operator precedence determines which operator is performed first in an expression with more than one operator with different precedence. So, whenever an expression contains more than one operator, the operator with a higher precedence is evaluated first.
In C, precedence of arithmetic operators(*, %, /, +, -) is higher than relational operators(==, !=, >, <, >=, <=) and precedence of relational operator is higher than logical operators(&&, || and !).
The precedence of commonly used arithmetic operators is shown below in table –

Priority	Operators	Description
1st	• / %	Multiplication, division , Modular division
2nd	+ -	Addition , Subtraction
3rd	=	Assignment

For example , in the expression –

2 + 3 * 5

Multiplication will be executed before addition since multiplication operator has higher precedence than the addition operator.

Example : Determine the hierarchy of operations and evaluate the following expression, assuming that I ia an integer variable :
I = 2*3/4+4/4+8-2+5/8
Stepwise evaluation of this expression is shown below :
I = 2*3/4+4/4+8-2+5/8
I = 6/4+4/4+8-2+5/8 operation : *
I = 1+4/4+8-2+5/8 operation : /
I = 1+1+8-2+5/8 operation : /
I = 1+1+8-2+0 operation :/
I = 2+8-2+0 operation : +
I = 10-2+0 operation : +
I = 8 +0 operation : -
I = 8 operation : +

Associativity of Operators

When an expression contains two operators of equal precedence the tie between them is settled using the associativity of the operators. Associativity of operators indicate the order in which they execute. Associativity can be of two types – Left to Right or Right to Left.
Associativity of the operators within same group is same.All the operators either associate from left to right or from right to left. Consider the expression –
5 + 16 / 2 * 4
Since / and * operators associate from left to right so / will be evaluated before * and the correct result is 37.

Here is the list of C operators in order of precedence (highest to lowest) and their associativity.

Operator	Description	Associativity
() [ ] -> .	Parentheses or function call Brackets (array subscript) Arrow operator Dot opeartor	Left to right
+ - ++ -- ! ~ * & sizeof	Unary plus/minus Increment / decrement Logical NOT One's Complement Indirection Address operator SizeOf operator	Right to Left
* / %	Multiplication Division Modulus division	Left to Right
+ -	Addition subtraction	Left to Right
<< >>	Left shift Right Shift	Left to Right
< <= > >=	Less than Less than equal to Greater than Greater than equal to	Left to Right
== !=	Equal to Not Equal to	Left to Right
&	Bitwise AND	Left to right
^	Bitwise XOR	Left to right

|	Bitwise OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional operator	Right to Left
=	Assignment Operator	Right to Left
,	Comma Operator	Left to right

Basic Input / Output statements in C

There are three main functions of any program – it takes data as input, processes this data and gives the output. The input operation involves movement of data from an input device to computer memory, while in output operation the data moves from computer memory to the output device. The input output operation is performed through a set of library functions that are supplied with every C compiler. The set of library functions that performs input-output operations is known as standard I/O Library.

There are several header files that provide necessary information in support of the various library functions. These header files are entered in the program using the #include directive at the beginning of the program. For example, if a program uses any function from the standard I/O library, then it should include the header file stdio.h as –

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

Reading Input Data

C provides the scanf() library function for entering input data. This function can take all types of values(numeric, character , string) as input. The scanf() function can be written as-

```
scanf( “control string” , address1, address2,……);
```

This function should have at least two parameters. First parameter is a control string, which contains conversion specification characters. It should be within double quotes. The conversion specification characters may be one or more ; it depends on the number of variables we want to input. The other parameters are addresses of variables. In scanf() function at least one address should be present.

Some examples of scanf() function are as –

```
#include<stdio.h>
Main()
{
Int marks;
scanf(“%d”, &marks);
}
```

In this example, the control string contains only one conversion specification character %d, which implies that one integer value should be entered as input.

```
#include<stdio.h>
main()
{
int basic, hr;
scanf(“%d%d”,&basic,&hr);
```

}
Here the control string has two conversion specifiers implying that two integer values should be entered. These values are stored in the variables basic and hr.

Writing Output Data

Output data can be written from computer memory to the standard output device (monitor) using printf() library function. With this function all types of values (numeric, character or string) can be written as output. The printf() function can be written as –

printf("control string", variable1, variable2,…..);

In this function the control string contains conversion specification characters and text. It should be enclosed within double quotes. The name of variables should not preceded by an ampersand (&) sign. If the control string does not contain any conversion specification, then the variable names are not specified.someexamples of printf() function are as –

Example 1 :-

```
#include<stdio.h>
main()
{
printf("C is excellent");
}
```

Output :

C is excellent

Here control string has only text and no conversion specification character, hence the output is only text.

Example 2:-

```
#include<stdio.h>
main()
{
int age;
printf("Enter your age:");
scanf("%d", &age);
}
```

Here also printf() does not contain any conversion specification character and is used to display a message that tells the user to enter his age.

Example 3 :-

```
#include<stdio.h>
main()
{
int basic=2000;
…….
printf("%d", basic);
……..
}
```

In this example control string contains a conversion specification character %d, which implies that an integer value will be displayed. The variable basic has that integer value which will be displayed as output.