CHAPTER - 2

Operators and Expressions

Introduction:

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate the data and variables. C language is rich in built-in operators and classified into a number of categories. They include:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and decrement operators
- Conditional operators
- Bitwise Operators
- Special Operators

An expression is a sequence of operands and operators that reduces to a single value.

Ex: 10+15

is an expression whose value is 25. The value can be any type other than void.

Arithmetic Operators:

C provides all the basic arithmetic operators. They are

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

The operators +,-,* and / all work the same way as they do in other languages. These can operate on any built-in data type allowed in C. The unary minus operator, in effect, multiplies its single operand by -1. Therefore, a number preceded by a minus sign changes its sign.

Integer Arithmetic:

When both the operands in a single arithmetic expression such as a+b are integers, the expression is called an integer expression, and the operation is called integer arithmetic. This mode of expression always yields an integer value. The largest integer value depends on the machine, as pointed out earlier

Example:

If a and b are integers then for a=14 and b=4

We have the following results: a – b=10

a + b = 18

a * b = 56

a / b = 3

a% b = 2

During integer division, if both the operands are of the same sign, the result is truncated towards zero. If one of them is negative, the direction of truncation is implementation dependent. That is,

6/7=0 and -6/-7=0

but -6/7 may be zero or -1 (Machine dependent)

Similarly, during modulo division, the sign of the result is always the sign of the first operand(the dividend)

That is -14 % 3 =-2

-14 % -3= -2

14 % -3=2

Real Arithmetic: An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation. Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result.

If x, y and z are floats, then we will have: x = 6.0 / 7.0 = 0.857143

y = 1.0 / 3.0 = 0.333333

z = - 2.0 / 3.0 = - 0.666667

The operator % cannot be used with real operands.

Mixed-mode Arithmetic: when one of the operands is real and the other is integer, the expression is cal. led a mixed-mode arithmetic expression. If either operand is of the real type, then only the real operation is performed and the result is always a real number. Thus 15/10.0 = 1.5 whereas 15/10=1

Relational Operators:

If we want to compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help relational operators.An expression such as

a<b or 1<20

containing a relational operator is termed as relational expression. The value of relational expression is termed as either one or zero. It is one if the specified relation is true and zero if the condition is false.

Ex: 10<20 is true

but 20<10 is false

C supports 6 relational operators in all. They are:

Operator	Meaning
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
= =	is equal to
!=	is not equal to

A simple relational expression contains only one relational operator and has the following form:

ae-1 relational operator ae-2

ae-1 and ae-2 are arithmetic expressions, which may be simple constants, variables or combination of them. Given below are some examples of simple relational expressions and their values:

4.5	<=	10	TRUE	
4.5	<	10	FALSE	
-35	>=	0	FALSE	
10	<	7+5	TRUE	
a+b	==	c+d	TRUE	only if the sum of values of a and b is equal to the sum c & d.

When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared. That is, arithmetic operators have a higher priority over relational operators. Relational expressions are used in decision statements such as, if and while to decide the course of action of a running program.

Logical Operator:

In addition to the relational operators . C has the following three logical operators.

&& logical AND

|| logical OR

! logical NOT

The logical operators && and || are used when we want to test more than one condition and make decisions.

Ex: a>b && x == 10

Theabove expression is termed as a logical expression or a compound relational expression which combines two or more relational expressions. Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table shown below. The logical expression given above is true only if a>b is true and x==10 is true. If either (or both) of them are false, the expression is false.

Truth Table

Op-1	op-2	Value of the expression	
		Op-1 && op-2	op-1 || op-2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Ex:

1. If(age>55 && salary <1000)
2. If (number<0 || number>100)

Precedence of relational operators and logical operators:

Ex: (a>b *5) &&(x<y+6)

In the above example, the expressions within the parentheses are evaluated first. The arithmetic operations are carried out before the relational operations. Thus b*5 is calculated and after that a is compared with it. Similarly y+6 is evaluated first and then x is compared with it.

Assignment Operator:

Assignment operators are used to assign the result of an expression to a variable. We have seen the usual assignment operator, '='. In addition, C has a set of 'shorthand' assignment operators of the form.

v op = exp;

Where v is a variable, exp is an expression and op is a C binary arithmetic operator. The operator op= is known as the shorthand assignment operator.

The assignment statement v op = exp;

Is equivalent to v = v op (exp); with v evaluated only once.

Ex: x += y+1;

This is same as x = x + (y+1);

The shorthand operator += means 'add y+1 to x' or 'increment x by y+1'.

For y = 2, the above statement becomes x += 3;

and when this statement is executed, 3 is added to x. If the old value of x is, say 5, then the new value of x is 8. Some of shorthand assignment operators are:

Statement with simple Assignment operator	Statement with shorthand operator
a = a + 1	a + = 1
a = a - 1	a - = 1
a = a *(n +1)	a * = (n+1)

a = a / (n+ 1)	a / = (n + 1)
a = a %b	a % = b

The use of shorthand assignment operators has three advantages:

1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
2. The statement is more concise and easier to read.
3. The statement is more efficient.

```
#include<stdio.h>
#define A  2
int main()
{
        int a;
        a=A;
        printf("%d \n",a);
        a*=a;
        printf ("%d \n",a);
        return0;
}
```

Output: 2

4

Increment and Decrement operators:-

The increment operator ++ and decrement operator -- are unary operators with the same precedence as the unary -, and they all associate from right to left. Both ++ and -- can be applied to variables, but no to constants or expressions. They can occur in either prefix or postfix position, with possibly different effects occurring. These are usually used with integer data type.

syn: ++variable or --variable or variable++ or variable—

The operator ++ adds 1 to the operand, while – subtracts 1.

Ex: ++m; or m++;

--m; or m--;

++m; is equivalent to m = m+1; (or m+=1;)

--m; is equivalent to m = m-1; (or m-=1;)

We use the increment and decrement statements in for and while extensively.

Ex: m=5;

y=++m;

In this case, the value of y and m would be 6. Suppose, if we rewrite the above statements as

m=5;

y=m++;

then the value of y would be 5 and m would 6. A prefix operator first adds to 1 to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

Similar is the case, when we use ++(or--) in the subscripted variables. That is, the statement a[i++]=10;

is equivalent to a[i]=10;

i=i+1;

The increment and decrement operators can be used in complex statements.

Ex: m=n++ -j+10;

Old value of n is used in evaluating the expression. n is incremented after the evaluation.

Conditional operator:

An operator called ternary operator pair "?:" is available in C to construct conditional expressions of the form.

exp1? exp2: exp3;

where exp1,exp2, and exp3 are expressions.

The operator ?; works as follows: exp1 is evaluated first. If it is nonzero(true), then the expression exp2 is evaluated and becomes the value of the expression. If exp1 is false,

exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expressions(either exp2 or exp3) is evaluated.

Ex: x=3; y=15;

z=(x>y)?x:y;

In this example, z will be assigned the value of y. This can be achieved using the if..else statements as follows:

```
If (x>y)
    z=x;
else
  z=b;
```

Ex:
```
main ()
{
    int a,b;
    clrscr ();
    printf (“Enter the values of a and b”);
    scanf (“%d%d”,&a,&b);
    (a>b)?printf(“%d is bigger”,a):printf (“%d is bigger”,b);
    Getch ();
}
```

Ex:
```
main ()
{
    int a,b,c,d;
    clrscr ();
    a=15;b=10;
    c=++a – b;
    printf (“The values of a, b, c are %d , %d , %d”,a,b,c);
    d=b++ +a;
    printf (“The values of a, b, d are %d , %d , %d”,a,b,d);
```

```
        printf ("a / b = %d",a/b);
        printf ("a %% b = %d",a%b);
        printf ("a * = b = %d",a*=b);
        printf (" %d",(c>d)?1:0);
        printf (" %d",(c<d)?1:0);
}
```

Output: The values of a, b, c are 16, 10, 6

The values of a, b, d are 16, 11, 26

a/b = 1 a%b = 5 a *= b = 176 0 1

Bitwise operators:

All data items are stored in the computer's memory as sequence of bits (i.e. 0s and 1s) and some applications need the manipulation of bits. C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double.

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	One's Complement

The result of bitwise AND operator is 1 when both the bits are 1, otherwise it is 0. Similarly the result of bitwise OR is one if any one of the bit is 1, otherwise it is 0. The result of bitwise XOR is 1 when the bits are different. The bitwise complement operator reverses the state of each bit. i.e. if the bit is zero, complement of it will 1 and vice versa.

Ex: int a =5, b=3;

Then equivalent binary representation of a = 5 = 0000 0101 and b=3= 0000 0011

Therefore

a = 0000 0101

b = 0000 0011

a& b = 0000 0001 = 1

a | b = 0000 0111 = 7

a ^ b = 0000 0110 = 6

~a = 1111 1010

a << 1 = 0000 1010 = 10 equivalent to multiplying a by 2

a >> 1 = 0000 0010 = 2 equivalent to dividing a by 2

Special Operators:

1) Comma Operator

C has some special operators. The comma operator is one among them. This operator can be used to link the related expressions together. A comma-linked list of expressions is evaluated left to right and the value of right-most expression is the value of the combined expression.

For example, the statement

Value=(a=2, b=6 ,a+b);

First assigns the value 2 to a, then assigns 6 to b, and finally assigns 8(i.e. 2+6) to value. The comma operator has the lowest precedence of all operators, hence the parentheses are necessary.

Ex: In for loops:

for(a=1, b=10;a<=b; a++, b++)

In while loops:

while(c=getchar(), C!='10')

Exchanging values:

t=x, x=y, y=t;

2) Sizeof operator

The sizeof operator returns the memory size (in bytes) of the operand. The operand may be a constant, variable or a data type. It is normally used to determine the size of arrays and structures.

syn: sizeof(operand);

Ex: x = sizeof(int); will return 2

y = sizeof(10.2) will return 4 since a float point constant requires 4 bytes of memory

z = sizeof(x) will return 2

```
Ex:   main()
      {
            int x;
            double y;
            char ch='y';
            clrscr();
            printf("Size of x = %d\n", sizeof(x));
            printf("Size of y = %d\n", sizeof(double));
            printf("Size of char = %d\n", sizeof(ch));
            getch();
      }
```

Output: size of x = 2

size of y = 8

size of char = 1

Arithmetic Expressions:

An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language. Expressions are evaluated using an assignment statement of the form

Variable = expression;

The table below shows the algebraic expression and C language expression

Algebraic expression	**C expression**
a b-c	a*b-c
(m + n) (x + y)	(m + n) *(x + y)
(a b)/c	a*b/c
$3x^2+2x+1$	3*x*x+2*x+1

Variable is any valid C identifier. When the statement is encountered, the expression is evaluated first and the result then replaces the precious value of the variable on the left-hand side. All variables used in the expression must be assigned values before evaluation is attempted.

```
x = a * b - c;
y = b / c * a;
z = a – b / c + d;
```

The blank space around an operator is optional and adds only to improve readability. When these statements are used in a program, the variables a, b, c and d must be defined before they are used in the expressions.

Ex:

```
main ()
{
    Float a,b,c,x,y,z;
    a=9;b=12;c=3;
    x=a-b/3+c*2-1;
    y=a-b/(3+c)*(2-1);
    z=a-(b/(3+c)*2)-1;
    printf (“x = %f”,x);
    printf (“y = %f”,y);
    printf (“z = %f”,z);
}
```

Output: x = 10.000000 y = 7.000000 z = 4.000000

Arithmetic operator's precedence:-

In a program the value of any expression is calculated by executing one arithmetic operation at a time. The order in which the arithmetic operations are executed in an expression is based on the rules of precedence of operators.

The precedence of operators is :

Unary (-)	FIRST
Multiplication(*) Division(/) and (%)	SECOND
Addition(+) and Subtraction(-)	LAST

For example, in the integer expression –a *b/c+d the unary- is done first, the result –a is multiplied by b, the product is divided by c(integer division) and d is added to it. The answer is thus: -ab/c+d

All the expressions are evaluated from left to right. All the unary negations are done first. After completing this, the expression is scanned from left to right; now all *, / and % operations are executed in the order of their appearance. Finally all the additions and subtractions are done starting from the left of the expression.

For example, consider the expression: Z=a + b* c

Initially b*c is evaluated and then the resultant is added with a. Suppose, if want to add a with b first, then it should be enclosed with parenthesis, is shown below

Z = (a + b) * c

Use of parentheses:

Parentheses are used if the order of operations governed by the precedence rules is to overridden.In the expression with a single pair of parentheses the expression inside the parentheses is evaluated FIRST. Within the parentheses the evaluation is governed by the precedence rules.

For example, in the expression: a * b/(c+d * k/m+k)+a

the expression within the parentheses is evaluated first giving: c+dk/m+k

After this the expression is evaluated from left to right using again the rules of precedence giving ab/c+dk/m+k +a

If an expression has many pairs of parentheses then the expression in the innermost pair is evaluated first, the next innermost and so on till all parentheses are removed. After this the operator precedence rules are used in evaluating the rest of the expression.

((x * y)+z/(n*p+j)+x)/y+z

xy, np+j will be evaluated first.

In the next scan xy+z/np+j +x will be evaluated.

In the final scan the expression evaluated would be:

(xy+ z/np+j+x)/y +z

Precedence and Associativity of operators:

Each operator in C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated. The operator at the higher level of precedence are evaluated first.

The operators of the same precedence are evaluated either from left to right or from right to left depending on the level. This is known as the associativity property of an operator.

Operator	Description	Level	Associativity
( ) [ ]	Parenthesis Array index	1	L – R
+ - ++ -- !	Unary plus Unary minus Increment Decrement Logical negation	2	R – L

~ & sizeof(type)	One's Complement Address of type cast conversion		
* / %	Multiplication Division Modulus	3	L- R
+ -	Addition Subtraction	4	L – R
<< >>	Left Shift Right Shift	5	L – R
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	L – R
= = ! =	is equal to Not equal to	7	L – R
&	Bitwise AND	8	L – R
^	Bitwise XOR	9	L – R
|	Bitwise OR	10	L – R
&&	Logical AND	11	L – R
| |	Logical OR	12	L – R
? :	Conditional Operator	13	R – L
=, +=, -=, *=, /=, %=	Assignment operator Short hand assignement	14	R – L
,	Comma operator	15	R – L

Evaluation of expressions involving all the above type of operators:

The following expression includes operators from six different precedence groups.

Consider variables x, y , z as integer variables.

```
Z+=(x>0 && x<=10) ? ++x : x/y;
```

The statement begins by evaluating the complex expression

```
(x>0  &&  x<=10)
```

If this expression is true, the expression ++x is evaluated. Otherwise, the x/y is evaluated. Finally, the assignment operation(+=) is carried out, causing the value of c to be increased by the value of the conditional expression.

If for example x, y, and z have the values 1,2,3 respectively, then the value of the conditional expression will be 2(because the expression ++a will be evaluated), and the value of z will increase to 5(z=3+2). On the other hand, if x,y and z have the values 50,10,20 respectively, then the value of the conditional expression will be 5(because the expression x/y will be evaluated) and the value of z will increase to 25(z=20+5).

Mathematical Functions:

In c there are no built-in mathematical functions. The table below shows the mathematical functions.

Function	Meaning
Trigonometric	
1. acos(x)	Arc cosine of x
2. asin(x)	Arc sine of x
3. atan(x)	Arc tan of x
4. atan2(x,y)	Arc tangent of x/y
5. cos(x)	cosine of x
6. sin(x)	sine of x
7. tan(x)	tangent(x)
Hyperbolic	

1. cosh(x)	Hyperbolic cosine of x
2. sinh(x)	Hyperbolic sine of x
3. tanh(x)	Hyperbolic tangent of x

Other functions

ceil(x)	x rounded up to the nearest integer
exp(x)	e to the power x
fabs(x)	Absolute value of x
floor(x)	x rounded down to the nearest integer
fmod(x,y)	Remainder of x/y
log(x)	Natural log of x, x>0
log 10 (x)	Base 10 log of x, x>0
pow(x,y)	x to the power y
sqrt(x)	Square root of x, x>=0

Note:

1. x and y should be declared as double
2. In trigonometric and hyperbolic functions, x and y are in radians
3. All the functions return a double.

Preprocessor directives:

There are different preprocessor directives. The table below shows the preprocessor directives.

Directive	**Function**
#define	defines a macro substitution
#undef	Undefines a macro
#include	Specifies the files to be included.
#ifdef	Tests for a macro definition
#endif	Specifies the end of #if
#ifndef	Tests whether a macro is not defined
#if	Tests a compile-time condition
#else	Specifies alternatives when #if tests fails

#define statement:

One of the uses of the #define statement is for assigning symbolic names to program constants.

syn: #define TRUE 1

defines the name TRUE and makes it equivalent to the value 1. The name TRUE can subsequently be used anywhere in the program where the constant 1 could be used. Whenever this name appears, its defined value of 1 will be automatically substituted into the program by the preprocessor. For example, we might have the following C statement that uses the defined name TRUE

count=TRUE;

This statement would assign the value of TRUE to count.

The preprocessor statement #define FALSE 0

would define the name FALSE and would make its subsequent use in the program equivalent to specifying the value 0. Therefore, the statement

count=FALSE;

would assign the value of FALSE to count.

```
#define TRUE  1
#define  FALSE 0
/*Function to determine if an integer is even*/
main()
{
    int  number,ans;
    printf(“enter the number:”);
    scanf(“%d” , &number);
    if (number%2==0)
    {
       ans=TRUE;
```

```
        else
            ans=FALSE;
        }
}
```

Ex: #define COUNT 100

#define SUBJECTS 6

#define PI 3.1415

#define CAPITAL “BANGALORE”

#include statement:

C preprocessor enables you to collect all your definitions into a separate file and then include them in your program using the #include statement.

The general statement for this preprocessor directive is:

#include “filename”

The #include directive can take the form

#include<filename>

without double quotation marks. In this case, the file searched only in the standard directories.

#include <stdio.h>

#include “SYNTAX.C”

#include “STAT.C”

#include “TEST.C”

Header files:

C language offers simpler way to simplify the use of library functions to the greatest extent possible. This is done by placing the required library function declarations in special source files, called header files. Most C compilers include several header files, each of which contains declarations that are functionally related. stdio.h is a header file containing declarations for input/ouput routines; math.h contains declarations for certain mathematical functions and so on. The header files also contain other information related to the use of the library functions, such as symbolic constant definitions.

The required header files must be merged with the source program during the compilation process. This is accomplished by placing one or more #include statements at the beginning of the source program.

The other header files are:

<ctype.h>	character testing and conversion functions
<stdlib.h>	utility functions such as string conversion routines , memory allocation routines, random number generator, etc
<string.h>	String manipulations functions
<time.h>	time manipulation functions