Introduction to C

High Level Languages have been developed to facilitate easy programming of the computers by any ordinary person. One such HLL is the C Language. It is becoming so popular that the languages like BASIC, FORTRAN, and PASCAL are becoming obsolete. This is because C being a High Level language satisfies the varying requirements of programmers. It can be used for application program development as well as system program development.

History of C

The history of C starts with a language called BCPL (Beginners Combined Programming Language) developed by Martin Richards. In 1970, Mr. Ken Thompson, a System Engineer at AT & T Bell Laboratories, USA, wrote an earlier version of C language for the UNIX operating system. It is a modified version of the BCPL. Therefore to distinguish his language from BCPL, he called the language as B language, the first letter of BCPL. The B language was modified to a greater extent by Mr. Dennis Ritchie at Bell Labs. This modified version of B is called as the C language, the second letter of BCPL.

C was originally designed for UNIX operating system. But, later the whole UNIX operating system itself was almost rewritten in C language. C is a powerful, general purpose, procedure oriented, structured programming language.

About ANSI C Standard

For many years, the de facto standard for implementing the language has been the original C Reference Manual by Kernighan and Ritchie published in 1978. During these years, C has undergone many changes. Numerous different features and facilities have been developed and added. This has resulted in minor problems in portability. Consequently, the American National Standards Institute defined a standard for C, eliminating much uncertainty about the exact syntax of the language. This newcomer, called ANSI C, proclaims itself the standard version of the language. As such it will inevitably overtake, and eventually replace common C. ANSI C does incorporate a few improvements over the old common C. The main difference is in the grammar of the language. The form of function declarations has been changed making them rather more like Pascal procedures. The most important ANSI additions are:

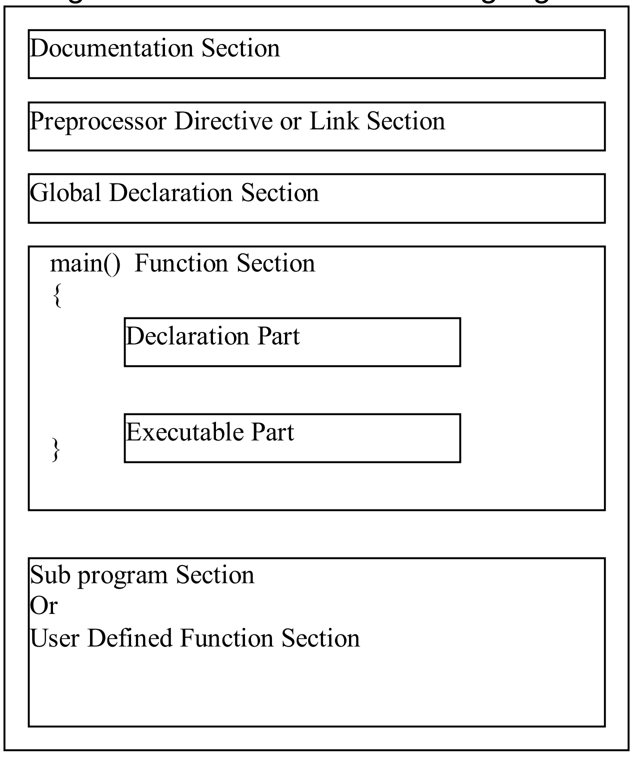
1. A new format for declaring and defining functions.
2. Facility for doing floating point computations.
3. Two new type qualifiers const and volatile and two new data type enum and void have been introduced.
4. The proprocessor includes some new macros.
5. A standard ANSI C library has been defined.

Important features of C Language

1. C is a system programming language which provides flexibility for writing compilers, operating systems, etc.
2. It can also be used for writing the application programs for scientific, engineering and business applications.
3. C is famous for its portability, meaning that program written in C for one computer can be easily loaded to another computer with little or no changes.
4. C supports variety of data types like integers, float point numbers, characters, etc.
5. C is a procedure oriented language which is most suited for structured programming practice.
6. It provides a rich set of built in functions
7. Programs written in C are found to execute faster than compared to other languages.

Structure of the C Language

The general structure of the C language includes the following sections



Example Program

```
/* Sample C Program */
main()
{
        int a,b;

        clrscr();

        printf("Enter two numbers");
        scanf("%d%d", &a,&b);

        sum = a + b;

        printf("Sum = %d", sum);
}
```

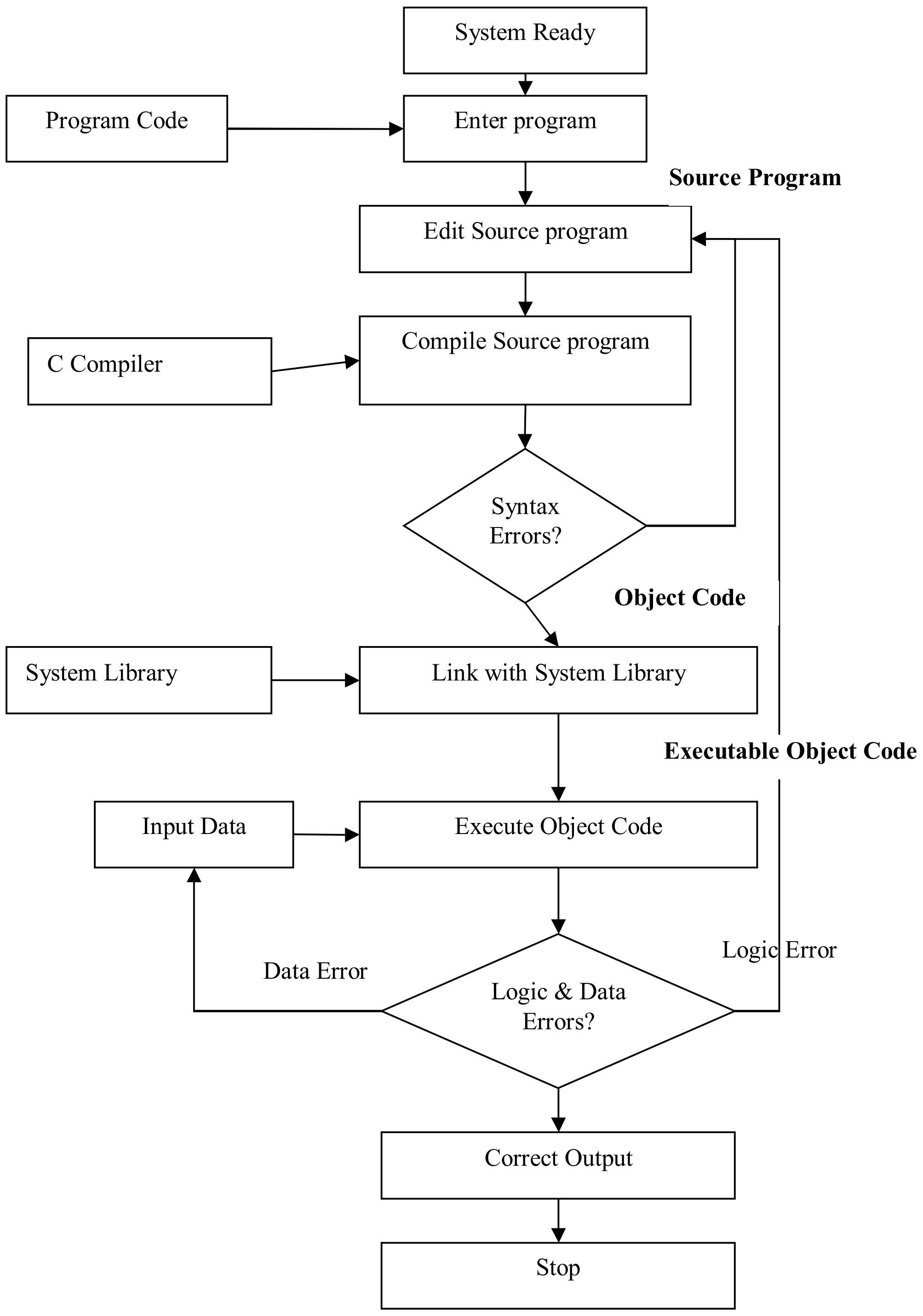
1. **The Documentation Section**: It consists of a set of comment lines giving the name of the program, the author and other details which the programmer would like to use later. The comments are enclosed in a C program using /* and */ .
2. **The Preprocessor directive or Link Section**: It provides instruction to the compiler to link some functions or do some processing prior to the execution of the program. It is also used to define symbolic constants of the program.
3. **Global Declaration Section**: There are some variables that are used in more than one function. Such variables are called global variables and are declared in this section that is outside of all other functions.
4. **main() function section** : Every C program must have one main() function. This section contains two parts, declaration part and executable part. The declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration part and executable parts must end with a semicolon.
5. **Sub program Section**: It contains all the user defined functions that are called in the main() function. User defined functions are generally placed immediately after the main function, although they may appear in any order.

All sections except the main() function may be absent when they are not required in any C program.

Executing a C program

Executing a C program involves a series of following steps.

1. Creating a program
2. Compiling the program
3. Linking the program with functions that are needed from the C library.
4. Executing the program



Important points to remember:

1. Every C program requires a main() function. Use of more than one main() is illegal. The place of main() is where the program execution begins.
2. The Execution of the function begins at the opening brace and ends at the closing brace.
3. C programs are written in lowercase letters. However uppercase letters may be used for symbolic names and constants.
4. All the words in a program line must be separated from each other by at least one space or a tab, or a punctuation mark.
5. Every statement must end with a semicolon.
6. All variables must be declared for their type before they are used in the program.
7. Compiler directives such as define and include are special instructions to the compiler, so they do not end with a semicolon.
8. When braces are used in the program make sure that the opening brace has corresponding ending brace.
9. C is a free form language and therefore proper form of indentation of various sections would improve the legibility of the program.
10. A comment can be inserted almost anywhere a space can appear. Use of appropriate comments in proper places increases the readability of the program and helps in debugging an testing.

LEXICAL ELEMENTS OF THE C LANGUAGE

Like any other High level language, C provides a collection of basic building blocks, symbolic words called lexical elements of the language. Each lexical element may be a symbol, operator, symbolic name or word, a special character, a label, expression, reserve word, etc. All these lexical elements are arranged to form the statements using the syntax rules of the C language. Following are the lexical elements of the C language.

C Character Set

Every language has its own character set. The character set of the C language consists of basic symbols of the language. A character indicates any English alphabet, digit or special symbol including arithmetic operators. The C language character set includes

1. Letter, Uppercase A ….. Z, Lower case a….z
2. Digits, Decimal digits 0….9.
3. Special Characters, such as comma, period. semicolon; colon: question mark?, apostrophe‘ quotation mark “ Exclamation mark ! vertical bar | slash / backslash \ tilde ~ underscore _ dollar $ percent % hash # ampersand & caret ^ asterisk * minus – plus + <, >, (,), [,], {, }
4. White spaces such as blank space, horizontal tab, carriage return, new line and form feed.

C Tokens

In a passage of text, individual words and punctuation marks are called tokens. Similarly, in C program, the smallest individual units are known as C tokens. C has following tokens

1. Keywords or Reserve words such as float, int, etc
2. Constants such 1, 15,5 etc
3. Identifiers such name, amount etc
4. Operators such as +, -, * etc
5. Special Symbols such as :, ;, [,] etc and special characters

Keywords

Key words or Reserve words of the C language are the words whose meaning is already defined and explained to the C language compiler. Therefore Reserve words can not be used as identifiers or variable names. They should only be used to carry the pre-defined meaning. For example int is a reserve word. It indicates the data type of the variable as integer. Therefore it is reserved to carry the specific meaning. Any attempt to use it other than the intended purpose will generate a compile time error. C language has 32 keywords. Following are some of them

auto	break	case	char	const	continue	default
do	double	else	enum	extern	float	for
goto	if	int	long	register	return	short
signed	sizeof	static	struct	switch	typedef	union
unsigned	void	volatile	while			

Constants

A constant can be defined as a value or a quantity which does not change during the execution of a program. Meaning and value of the constant remains unchanged throughout the execution of the program. These are also called as literals. C supports following types of constant.

1. Integer Constants

An integer constant refers to a sequence of digits. There three types of integer constants, namely decimal, octal and hexadecimal.

Decimal integer constant consists of set of digits from 0 to 9 preceded by an optional + or – sign.

Ex: 123, -321, 0, 4567, + 78

Embedded spaces, commas and non-digit characters are not permitted between digits.

An octal integer constant consists of any combination of digits from 0 to 7 with a leading 0(zero). Ex : 037, 0435, 0567

A sequence of digits preceded by 0x or 0X is considered as hexadecimal digit. They may also include alphabets A to F or a to f representing numbers from 10 to 15.

The largest integer value that can be stored is machine dependant; It is 32767 for 16 bit computers. It is also possible to store larger integer constants by appending qualifiers such U, L and UL to the constants.

2. Floating point Constants or Real Constants

The quantities that are represented by numbers with fractional part are called floating point numbers. Ex: 0.567, -0.76, 56.78, +247.60. These numbers are shown in decimal notation , having a whole number followed by a decimal point and the fractional part. It is possible to omit digits before the decimal point or digits after the decimal point. Ex: 215., .95, -.76 or +.5

A real number or a floating point number can also expressed as in exponential notation. Ex: 2.15E2. The general form of exponential notation is

mantissa **e** exponent or mantissa **E** exponent

The mantissa is either a real number or an integer. The exponent is an integer with an optional plus or minus sign. Embedded white is not allowed in this notation.

3. Single Character constants

A single character constant contains a any valid character enclosed within a pair of single quote marks. Ex: '5', 'A', ';' ' '. The character constants have integer values associated with them known as ASCII values. For ex: A is having the ASCII value of 65.

4. String constants

A string constant is a sequence of characters enclosed in double quotes. The characters may be alphabets, numbers special characters and blank space. Ex: "Hello", "2002", "Wel Come", "5+3"

5. Backslash character constants or Escape sequence characters.

C supports some special backslash character constants that are used in output functions. For ex: '\n' stands for new line characters. Each one of them represents a single character even though it consists of two characters.

\a	Audible alert or bell	\b	back space
\f	form feed	\n	new line
\r	carriage return	\t	horizontal tab
\v	vertical tab	\'	single quote
\"	double quote	\?	Question mark
\\	backslash	\0	null

Identifiers

An identifier is a sequence of letters, digits and an underscore. Identifiers are used to identify or name program elements such as variables, function names, etc. Identifiers give unique names to various elements of the program. Some identifiers are reserved as special to the C language. They are called keywords.

Variables

A variable is a data name that may be used to store data value. A value or a quantity which may vary during the program execution can be called as a variable. Each variable has a specific memory location in memory unit, where numerical values or characters can be stored. A variable is represented by a symbolic name. Thus variable name refers to the location of the memory in which a particular data can be stored. Variables names are also called as identifiers since they identify the varying quantities.

For Ex : sum = a+b. In this equation sum, a and b are the identifiers or variable names representing the numbers stored in the memory locations.

Rules to be followed for constructing the Variable names (identifiers)

1. They must begin with a letter and underscore is considered as a letter.
2. It must consist of single letter or sequence of letters, digits or underscore character.
3. Uppercase and lowercase are significant. For ex: Sum, SUM and sum are three distinct variables.
4. Keywords are not allowed in variable names.
5. Special characters except the underscore are not allowed.
6. White space is also not allowed.
7. The variable name must be 8 characters long. But some recent compilers like ANSI C supports 32 characters for the variable names and first 8 characters are significant in most compilers.

Data Types

Data refers to any information which is to be stored in a computer. For example, marks of a student, salary of a person, name of a person etc. These data may be of different types. Computer allocates memory to a variable depending on the data that is to be stored in the variable. So it becomes necessary to define the type of the data which is to be stored in a variable while declaring a variable. The different data types supported by C are as follows:

int Data Type

Integer refers to a whole number with a range of values supported by a particular machine. Generally integers occupy one word of storage i.e 16 bits or 2 bytes. So its value can range from -32768 to +32767.

Declaration: int variable name;

Ex: int qty;

The above declaration will allocate a memory location which can store only integers, both positive and negative. If we try to store a fractional value in this location, the fractional data will be lost.

float Data Type

A floating point number consists of sequence of one or more digits of decimal number system along with embedded decimal point and fractional part if any. Computer allocates 32 bits, i.e. 4 bytes of memory for storing float type of variables. These numbers are stored with 6 digits of precision for fractional part.

Declaration: float variableName; Ex : float amount;

double Data Type

It is similar to the float type. It is used whenever the accuracy required to represent the number is more. In others words variables declared of type double can store floating point numbers with number of significant digits is roughly twice or double than that of float type. It uses 64 bits i.e. 8 bytes of memory giving a precision of 14 decimal digits.

Declaration : double variableName;

char Data Type

A single character can be defined as char data type. These are stored usually as 8 bits i.e 1 byte of memory.

Declaration : char variableName ; Ex: char pass;

String refers to a series of characters. Strings are declared as array of char types.

Ex: *char name[20];* will reserve a memory location to store upto 20 characters.

Further, applying qualifiers to the above primary data types yield additional data types. A qualifier alters the characteristics of the data type, such as its sign or size. There are two types of qualifiers namely, sign qualifiers and size qualifiers.

signed and unsigned are the sign qualifiers

short and long are the size qualifiers.

Size and range of data types on a 16 bit Machine

Type	Size (bits)	Range
char or signed char	8	-128 to 127
unsigned char	8	0 to 255
int	16	-32768 to 32767
unsigned int	16	0 to 65535
short int or signed short int	8	-128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	-2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float	32	3.4e-38 to 3.4e+38
double	64	1.7e-308 to 1.7e+308
long double	80	3.4 e-4932 to 1.1e+4932

Assigning Values to Variables

Values can be assigned to variables using assignment operator “=”.

Syntax:

```
Variablename = value;
Ex:        a = 10;      intialvalue = 0;
```

It is also possible to assign a value to a variable at the time the variable is declared. The process of giving initial value to the variable is called initialization of the variable.

Ex:

```
int a=10, b=5;
float x=10.5, y=1.2e-9
```

Declaring a variable as constant

We may want the value of the certain variable to remain constant during the execution of the program. We can achieve this by declaring the variable with const qualifier at the time of initialization.

Ex;

const int tax_rate = 0.30;

The above statement tells the compiler that value of variable must not be modified during the execution of the program. Any attempt change the value will generate a compile time error.

Declaring a variable as volatile

Declaring the variable volatile qualifier tells the compiler explicitly that the variable's value may be changed at any time by some external source and the compiler has to check the value of the variable each time it is encountered.

Ex;

volatile int date;

Defining Symbolic Constants

We often use certain unique constants in a program. These constants may appear repeatedly in number of places in a program. Such constants can be defined and its value can be substituted during the preprocessing stage itself.

Syntax:

```
#define symbolic-name  value of constant
```

ex:

```
#define PI 3.14
#define MAX 100
```

Operators and Expression in C

An operator is a symbol which acts on operands to produce certain result as output. For example in the expression a+b; + is an operator, a and b are operands. The operators are fundamental to any mathematical computations.

Operators can be classified as follows:

1. Based on the number of operands the operator acts upon:
   a. Unary operators: acts on a single operand. For example: unary minus(-5, -20, etc), address of operator (&a)
   b. Binary operators: acts on two operands. Ex: +, -, %, /, *, etc
   c. Ternary operator: acts on three operands. The symbol ?: is called ternary operator in C language. Usage: big= a>b?a:b; i.e if a>b, then big=a else big=b.
2. Based on the functions
   a. Arithmetic operators
   b. Relational operators
   c. Logical Operators
   d. Increment and Decrement operators
   e. Assignment operators
   f. Bitwise operators
   g. Conditional Operators
   h. Special operators

Arithmetic operators:

C provides all the basic arithmetic operators. 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.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo division

Integer division truncates any fractional part. The modulo division produces the remainder of an integer division and can be used only on integer operands.

Examples:

1) If a and b are integers and a =14	2) -14 / 3 = -4

and b = 3, then a + b = 14 + 3 = 17 a – b = 14 – 3 = 11 a * b = 14 * 3 = 42 a / b = 14 / 3 = 4 a % b = 14 % 3 = 2	-14 / -3 = 4 14 / -3 = -4 -14 % 3 = -2 -14 % -3 = -2 14 % -3 = 2 2 % 3 = 2 2 / 3 = 0

Here a and b are variables and are known as operands. The modulo divison operator % cannot be used on floating point data.

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.

Modes of expression:

There are three different modes of expressions.

1. Integer Arithmetic
2. Real Arithmetic
3. Mixed-mode Arithmetic

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 called 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 where as 15/10=1

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, in 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 are 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

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.

The general syntax is:

```
++variable or --variable or variable++ or variable—
```

Some examples are

```
++count -kk index++ unit_one--
```

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

```
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. Example

```
m=n++ -j+10;
```

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

Relational operators:

We often compare two quantities and depending on their relation, to take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators.

C supports six relational operators in all. These operators and their meanings are shown below Relational Operators

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 operators:

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.

Example: a>b && x == 10

An expression of this kind which combines two or more relational expression is termed as a logical expression or a compound relational expression. 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

Some examples of the usage of logical expressions are:

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

Relational and logical expressions:

We have seen that float or integer quantities may be connected by relational operators to yield an answer which is true of false.

For example the expression,

Marks>=60

would have an answer true if marks is greater than or equal to 60 and false if marks is less than 60. The result of the comparison (marks>=60) is called a logical quantity. C provides a facility to combine such logical quantities by logical operators to logical expressions. These logical expressions are useful in translating intricate problem statements.

Example : A university has the following rules for a student to qualify for a degree with Physics as the main subject and Mathematics as the subsidiary subject:

(i) He should get 50 percent or more in Physics and 40 percent or more in Mathematics.
(ii) If he gets less than 50 percent in Physics he should get 50 percent or more in Mathematics. He should get atleast 40 percent in Physics.
(iii) If he gets less than 40 percent in Mathematics and 60 percent or more in Physics he is allowed to reappear in an examination in Mathematics to qualify.
(iv) In all the other cases he is declared to have failed.

```
/*This program implements above rules*/
include<stdio.h>
main()
{
         int roll_no, physics_marks, chem_marks;
        while(scanf("%d %d %d", &roll_no, &physics_marks, &chem_marks)!=EOF)
        {
                If(((chem_marks>=50    &&    (physics_marks>=35))    ||    ((chem_marks>=40))    &&
                (physics_marks>=50)))
                        printf("%d %d %d  Pass\n", roll_no, chem_marks, physics_marks);
                else
                  if (( chem_marks>=60)) && physics_marks<35))
                        printf("%d %d %d Repeat Physics\n", roll_no,physics_marks,chem_marks);
                  else
                        printf("%d %d %d Failed \n", roll_no, physics  _marks, chem_marks);
                }
                }/*End while*/
}/*End main*/
```

Precedence of relational operators and logical operators:

Example:

(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 .

In general within parentheses:

1. The unary operations, namely, -,++,--,! (logical not) are performed first.
2. Arithmetic operations are performed next as per their precedence.
3. After that the relational operations in each sub expressions are performed, each sub expression will be a zero or non _ zero. If it is zero it is taken as false else it is taken as true.
4. These logical values are now operated on by the logical operators.
5. Next the logical operation && is performed next.
6. Lastly the evaluated expression is assigned to a variable name as per the assignment operator.

The 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.

For example, consider the following statements

```
x=3;
y=15;
z=(x>y)?x:y;
```

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

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

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 are 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.
Some examples are given below:

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. Its syntax is as follows:

sizeof(operand);

Example : 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

Example program:

```
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", sizeofch));
        getch();
}
```

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
+ - ++ -- ! ~ & sizeof(type)	Unary plus Unary minus Increment Decrement Logical negation One's Complement Address of type cast conversion	2	R – L
* / %	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+=(x0>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. Other wise, the a/b 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.