

**CHAPTER - I**

**INTRODUCTION TO**

**C - PROGRAMMING**

## INTRODUCTION TO C - PROGRAMMING

C is one of the most popular language today. C was an off-spring of the `Basic Combined Programming Language (BCPL) called B, developed in 1960's at Cambridge University. B language was modified by Dennis Ritchie and was implemented at Bell Laboratories in 1972 and new language was named C. Today, C is running under a number of operating systems including MS - DOS. This work also includes all essential features of ANSI C (American National Standard Institute - ANSI).

### 1.1.1 IMPORTANCE OF C

(i) It has rich set of built in functions and operators, so can be used to write any complex program.

(ii) Programs written in C are efficient and fast. This is due to variety of data types and powerful operators.

(iii) There are only thirtytwo keywords and its strength lies in "builtin" functions. Several standard functions are available which can be used for developing programs.

(iv) C is highly portable. This means that C -programs written for one computer can be run on another with little or no modification.

(v) C - language is suited for structural programming, thus requiring the user to think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. This modular structure make program debugging, testing and maintenance easier.

#### SAMPLE C PROGRAMS

```
/* Program # 1 - My first C program. */
# include "stdio.h"
main()
{
printf ("This is my frist C program");
}
```

When run, this program displays

" This is my first C program" on the display screen.

Let us examine the function of each line in the program.

1st line -

```
/* program #1 -My first C program. */
```

This is a comment.

In c, as in most other programming languages, you can enter a remark into a program's source code that is ignored by the compiler. The purpose of comment is to provide an explanation of what the program - or part of program - is doing. In more complex programs, comments are used to explain what each feature of the program is for and how it functions.

In C comment begins with a slash followed by an asterisk; that

is, the symbol pair `/*`. A comment is concluded by the same pair in reverse, `*/`.

Thus comments in program enhance its readability and understanding and can be inserted any where blank space can occur, but never in the middle of word.

The next line is,

```
# include "stdio.h"
```

The C language defines several files, called header files, that contain information either necessary or useful to a program. For this program, the file `stdio.h` is needed. `Stdio.h` refers to the standard I / O header file containing standard input and output functions.

The next line is,

```
main()
```

All c programs are composed of building blocks called functions. A program may consists of one or several functions. Each C function must have a name and the only function that any C program must have is the one called `main()`. The `main()` function is where program execution begins and, usually, ends. In technical terms, a C program begins with a call to `main()` and ends (in most cases) when `main()` returns.

The opening brace that follows `main()` marks the start of the `main()` function code. The next line in the program is `printf("This is my first C program.");` This line causes the message. "This is my first C- program." to be displayed on the screen. It does this by calling the standard `printf()` function.

### 1.1.2 CONSTANTS, VARIABLES AND DATA TYPES

A programming language is designed to help to process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information. The task of processing of data is accomplished by executing a sequence of precise instructions called a program. These instructions are formed using certain symbols and words according to some rigid rules known as syntax rules (or grammar). Like any other language, C has its own vocabulary and grammar.

Character set :

The character that can be used to form words, numbers and expressions depend upon the computer on which the program is run. The characters in C are grouped into the following categories :

1. Letters
2. Digits
3. Special characters
4. White spaces.

The complete character set is given in Table 1.1.1

Table 1.1.1

Letters	Digits
Upper case A.....Z	All digits 0 ..... 9
Lower case a.....z	
Special Characters	
, comma	& ampersand
. period	^ caret
; semicolon	* asterisk

:	colon	-	minus sign
?	question mark	+	plus sign
'	apostrophe	<	opening angle bracket or less than sign
"	quotation mark	>	closing angle bracket or greater than sign
!	exclamation mark		
	vertical bar	(	left paranthesis
/	slash	)	right paranthesis
\	backslash	[	left bracket
~	tilde	]	right bracket
_	underscore	{	left brace
\$	dollar sign	}	right brace
%	percent sign	#	number sign
	White spaces		
	Blank space		
	Horizontal tab		
	Carraige return		
	New line		
	Form Feed		

## 1.1.3

## C Tokens

In a passage of text, individual words and punctuation marks are called tokens. Similarly, in a C program the smallest individual units are known as C Tokens. C has six types of tokens as shown in Fig 1.1.1

C programs are written using these tokens and the syntax of the language.

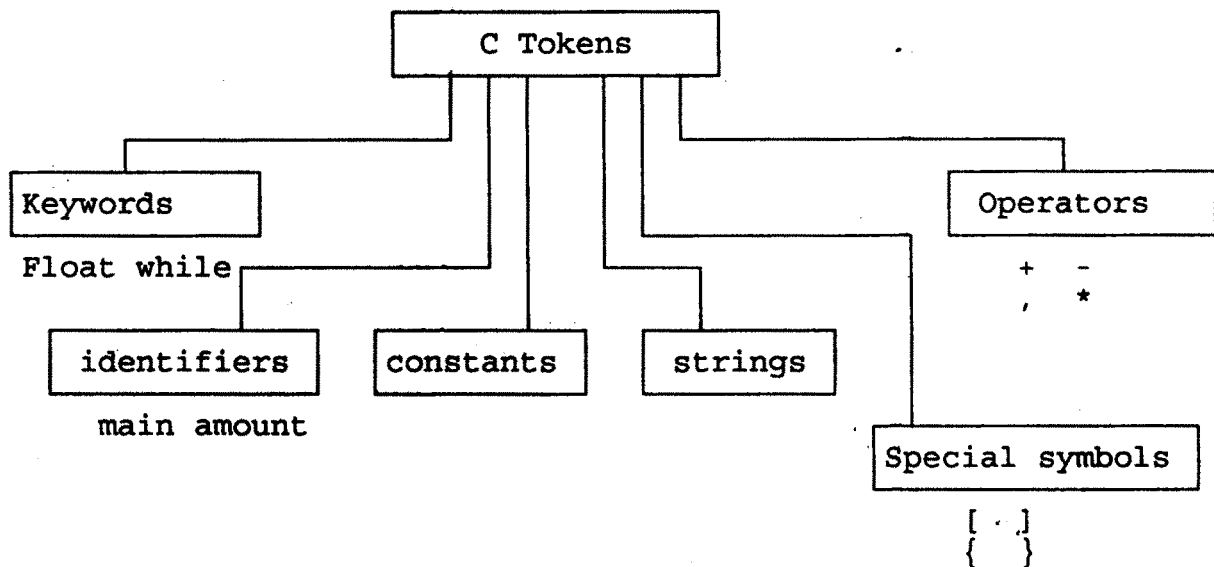


Fig 1.1.1

## Keywords and Identifiers

Every C word is classified as either a keyword or an identifier.

All keywords have fixed meanings and these meanings cannot be changed. Keywords serve as basic building blocks for program statement. The list of all keywords in ANSI C are listed in Table

All keywords must be written in lowercases. Some compilers may use additional keywords that must be identified from the C manual.

Table 1.1.2

ANSI C KEYWORDS

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

Identifiers refer to the names of variable, functions and arrays. These are user - defined names and consists of a sequence of letters and digits, with a letter as a first character. Both uppercase and lowercase are permitted ,although lowercase letters are commonly used. The underscore character is also permitted in identifiers. It is usually used as a link between two words in long identifiers

Constants :

Constants in C refer to fixed values that do not change during the execution of a program. C supports several types of constants as in Fig. (1.1.2)



## Constants

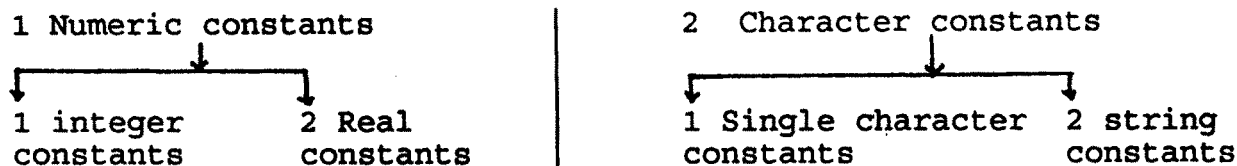


Fig 1.1.2 Basic types of C constants

### Integer

constants:

An integer constant refer to a sequence of digits. There are three types of integers, namely, decimal, octal and hexadecimal. Decimal integers consists of a set of digits, 0 through 9, preceded by an optional - or + sign. valid examples of decimal integer constants are,

123

-321

0

654321

+78

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

Examples,

15 750

20,000

\$ 1000

are illegal numbers.

Note that ANSI C supports unary plus which was not defined earlier.

An octal integer constants consists of any combination of digits from the set 0 through 7, with a leading 0 some examples of octal integers are

037

0

0435

A sequence of digits preceded by OX or ox is considered as hexadecimal integer. They may also include alphabets A through F or a through f. The letters A through F represent the number 10 through 15. Following are the examples of valid hex integers.

OX2

OX9F

OXabcd

OX

We rarely use octal and hexadecimal numbers in programming.

The largest integer value that can be stored is machine dependent. It is 32767 on 16 - bit machines and 2,147,483,647 on 32 - bit machines. It is also possible to store larger integer constants on these machines by appending qualifiers such as U, L and UL to the constants. for example :

56789U                    or 56789u (unsigned integer)

987612347UL            or 987612347ul (unsigned long integer)

9876543L                or 9876543l (long integer)

### Real Constants

Integer numbers are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices and so on. These quantities are represented by numbers containing fractional parts like 17.548 such numbers are called real (or floating point) constants. Further examples of real constants are

0.0083

-0.75

435.36

+247.0

These numbers are shown in decimal notation, having a whole number followed by a decimal point, or digits after the decimal point. That is,

215.

.95

-7.1

are valid real numbers.

A real number also can be expressed in exponential (or scientific) notation. For example, the value 215.65 may be written as  $2.1565 \times 10^2$  in exponential notation.  $e2$  means multiply by  $10^2$ . The general form is :

mantissa e exponent.

the mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer number with an optional plus or minus sign. The letter e separating the mantissa

and the exponent can be written in either lowercase or uppercase. Since exponent causes the decimal point to "float", this notation is said to represent a real number in floating point form. Examples of legal floating point constants are

0.65e4

12e-2

1.5e+5

3.18E+3

-1.2E-1

Embedded white space is not allowed.

Exponential notation is useful for representing numbers that are either very large or very small in magnitude. For example, 75000000000 may be written as 7.5e+10 or 75e9 similarly, -0.0000000386 is equivalent to -3.86e-8. Floating point constants are normally represented as double precision quantities. However, the suffixes f or F may be used to force single precision and l or L to extend double precision further.

Single character constants :

A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Examples of character constants are :

'5' 'x' 'j' ' '

character '5' is not the same as the number 5. The last constant is blank space.

Character constants have integer values known as ASCII values.

For example, the statement

```
printf ("%d", 'a');
```

would print the number 97, the ASCII value of letter a. Similarly, the statement `printf ("%c", '97');` would output the letter 'a'.

### String constants

A string constant is a sequence of characters enclosed in double quotes. The characters may be letters, numbers, special characters and blank space. Examples are

```
"Hellow !"
```

```
"1995"
```

```
"?...!"
```

```
"5+3"
```

```
"X"
```

Remember that a character constant (`'X'`) is not equivalent to the single character string constant (`"X"`).

### Backslash character constants:

C supports some special backslash character constants that are used in output functions. For example, the symbol `'\n'` stands for newline character.

### Variables :

A variable is a data name that may be used to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution.

A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program. Some examples of such names are

Average

Height

Class-strength

Variable name may consists of letters, digits and the underscore (`_`), character, subject to the following conditions.

1. They must begin with a letter some system permit underscore as the first character.
2. ANSI standard recognizes a length of 31 characters. However, the length should not be normally more than eight characters, since only the first eight characters are treated as significant by many compilers.
3. Uppercase and lowercase are significant. That is the variable Total is not the same as total or TOTAL.
4. The variable name should be not be a keyword.
5. White space is not allowed.

## DATA

## TYPES.

C language is rich in its data types. Storage representations and machine instructions to handle constants differ from machine to machine. The variety of data types available allow the programmer to select the type appropriate to the needs of the application as well as the machine. ANSI C supports four classes of data types :

1. Primary (or fundamental) data types.
2. User - defined data types.
3. Derived data types.
4. Empty data set.

All C compilers support four fundamental data types, namely integers (int), character (char), floating point (float), double - precision floating point (double). Many of them also offer extended data types such as long int and long double.

Fig 1.1.3

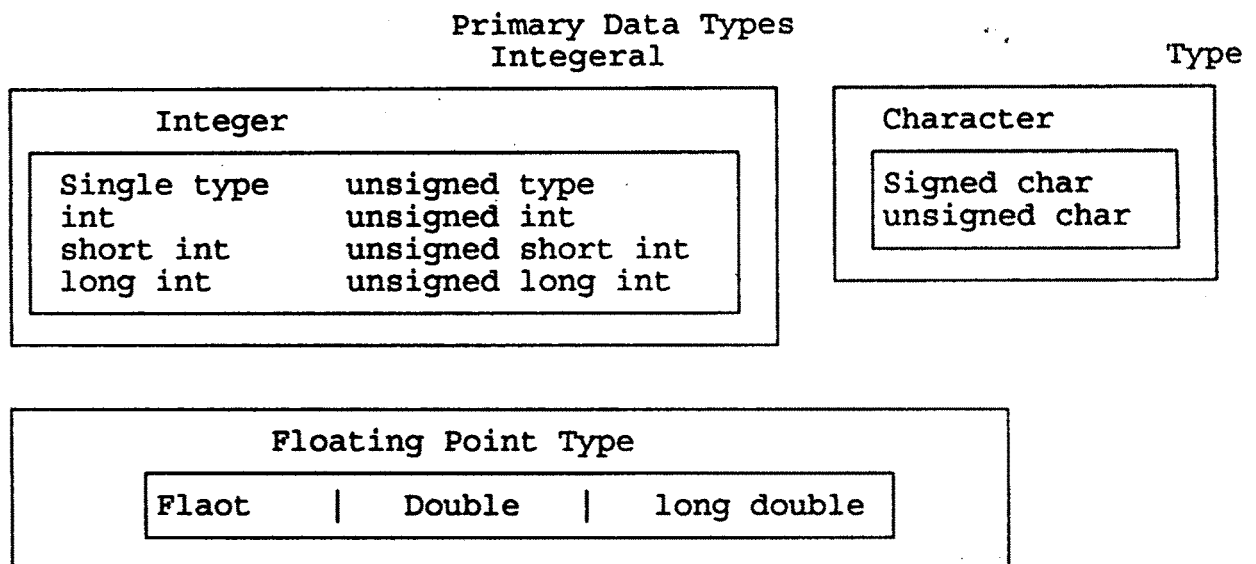


Table 1.1.3

Primary data types in C

Data Type	Range of values
Char	- 128 to 127
int	- 32,768 to 32,767
Float	3.4e -38 to 3.4e + 38
double	1.7 e - 308 to 1.7 e + 308

### Size and Range of Basic data types :

Various data types and the terminology used to describe them are given in Fig. 1.1.3.

The range of the basic four types are given in Table 1.1.3

**Integer types :-** Integers are whole numbers with a range of values supported by a particular machine. Generally, integers occupy one word of storage, and since the word sizes of machines vary the size of an integer that can be stored depends on the computer.

In order to provide some control over the range of numbers and storage space, C has three classes of integer storage, namely short int, int and long int, in both signed and unsigned forms. For example, short int represents fairly small integer values and requires half the amount of storage as a regular int number uses. Table 1.1.4 shows all the allowed combinations of basic types and qualifiers and their size and range on a 16 - bit machine.

Table 1.1.4, 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 or signed int	16	-32,768 to 32767
unsigned int short int or signed	16	0 to 65535
short int	8	-128 to 127



unsigned short int	8	0 to 255
long int or signed long int	32	-2,149,483,648 to 2,149,483,647

## 1.1.4

## Floating Point types

Floating point (or real) numbers are stored in 32 bit (with 6 digits of precision). Floating point numbers are defined in C by the keyword float. When the accuracy provided by a float number is not sufficient, the type double can be used to define the number. A double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. to extend precision further, we may use long double which uses 80 bits.

## Character type

:-

A single character can be defined as a character (char) type data. Characters are usually stored in 8 bits (one byte) of internal storage. The qualifier signed or unsigned may be explicitly applied to char.

## Declaration of variables :-

After designing suitable variable names, we must declare them to the compiler.

Declaration does two things :

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

Primary type declaration.

The syntax for declaring a variable is as follows

Data - type V1, V2, .....Vn:

Where V1, V2, .....Vn are the names of variables.

Some examples are,

```
int count;
```

```
float number, total;
```

```
double ratio;
```

Declaration of storage class :-

Variables in C can have not only data type but also storage class that provides information about their location and visibility.

There are four storage class specifiers as given in Table 1.1.5

Table 1.1.5

Storage class	Meaning
Auto	Local variable known to only to the function in which it is declared. Default is auto.
Static	Local variable which exists and retains its value even after the control is transformed to calling function.
Extern	Global variable known to all functions in the file.
Register	Local variable which is stored in the register.

Assigning values to variable :

Variables are created for use in program such as

```
Value = amount + inrate * amount ;
```

```
while (Year <= PERIOD)
```

```
{
```

```
....
```

```

.....
Year = Year + 1;
}

```

The result is stored in the variable value and called as target variable.

While all the variables are declared for their type, the variables that are used in expressions (on the right side of equal (=) sign of computational statement) must be assigned values before they are encountered in the program.

Similarly, the variable year and the symbolic constant PERIOD in the while statement must be assigned values before this statement is encountered.

#### 1.1.5 Overflow and underflow of data

Problem of data overflow occurs when the value of a variable is either too big or too small for the data type to hold. The largest value that a variable can hold also depends on the machine. Since floating point values are rounded off to the number of significant digits allowed (or specified), an overflow normally results in the largest possible real value, whereas an underflow results in zero.

Integers are always exact within the limits of the range of the integral data types used. However an overflow which is a serious problem may occur if the data type does not match the value of the constant. C does not provide any warning or indication of integer overflow. It simply gives incorrect results. We should

therefore exercise a greater care to define correct data types for handling the input / output values.

#### 1.1.6 Operators

C supports a rich set of operators.

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions.

C operators can be classified into a number of categories. They include :

1. Arithmetic operators.
2. Relational operators.
3. Logical operators.
4. Assignment operators.
5. Increment and decrement operators.
6. Conditional operators.
7. Bitwise operators.
8. Special operators.

#### Arithmetic Operators

:-

C provides all the basic arithmetic operators. They are listed in Table 1.1.6

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.

Table 1.1.6

## Arithmetic operators

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

Integer division truncates any fractional part. The modulo division produces the remainder of an integer division.

Note that C does not have an operator for exponentiation. Older versions of C does not support unary plus but ANSI C supports it.

Relational operators:-

We often compare two quantities, and depending on their relation, take certain decision. 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 meaning are shown in Table 1.1.7.

Table 1.1.7

## Relational Operators

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

Arithmetic operators have a higher priority over relational operators.

### Logical Operators

:-

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

& &	Meaning logical AND
	Meaning logical OR
!	Meaning logical Not

### Assignment Operator

:-

Assignment operators are used to assign the result of an expression to a variable. Usual assignment operator is '='. In addition, C has a set of 'Shorthand' assignment operators of the form illustrated in Table 1.1.8.

Table 1.1.8

### Shorthand Assignment Operators

Statement with Simple assignment operator.	Statement with shorthand operator
<hr/>	<hr/>
<hr/>	<hr/>
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.

#### Increment And Decrement Operators:

C has two very useful operators not generally found in other languages. The increment and decrement operators are :

`++` and `--`

the operator `++` adds 1 to the operand while `--` subtract 1. Both are unary operators and take the following form :

`++ 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 ;`)

#### Conditional operators :-

A ternary operator pair `"? :"` is available in C to construct conditional expressions of the form

`expression 1 ? expression 2 : expression 3 ;`

The operator `? :` works as follows.

`expression 1` is evaluated first. If it is nonzero (true), then the `expression 2` is evaluated and becomes the value of the expression.

#### Bitwise operators :-

C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. These

are used for testing the bits, or shifting them right to left.

Bitwise operators may not be applied to float or double.

Table 1.1.9 lists bitwise operators and their meanings.

**Special operators :-**

C supports some special operators of interest such as comma operator, sizeof operators, pointer operators and member selection operators.

Table 1.1.9 Bitwise operator.

Operators	Meaning
&	bitwise AND
! ^	bitwise or bitwise exclusive OR
< <	Shift left
> >	Shift right
~	ones complement

#### 1.1.6(a) Precedence of arithmetic operators :-

An arithmetic expression without parantheses will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C.

High priority                      \*, /, %

Low priority                        +, -

The basic evaluation procedure includes two left to right passes through the expression. During the first pass, the high priority operators (if any) are applied as they are encountered. During the second pass, the low priority operators (if any) are applied as they are encountered.



### 1.1.6(b)                      Some                      computational                      problems.

When expression include real values, then it is important to take necessary precautions to guard against certain computational errors. We know that the computer gives approximate values for real numbers and the errors due to such approximations may lead to serious problems. For example, consider the following statements :

```
a = 1.0 / 3.0 ;
```

```
b = a * 3.0 ;
```

We know that (1.0 / 3.0) is equal to 1. But there is no guarantee that the value of b computed in a program will give 1.

Another problem is division by zero. On most computers, any attempts to divide a number by zero will result in abnormal termination of the program. In some cases such a division may produce meaningless results. Care should be taken to test the denominator that is likely to assume zero value and avoid any division by zero.

The third problem is to avoid overflow or underflow errors. It is our responsibility to guarantee that operands are of correct type and range and result may not produce any overflow or underflow.

### 1.1.6(c)                      Type                      conversions                      in                      expressions                      :

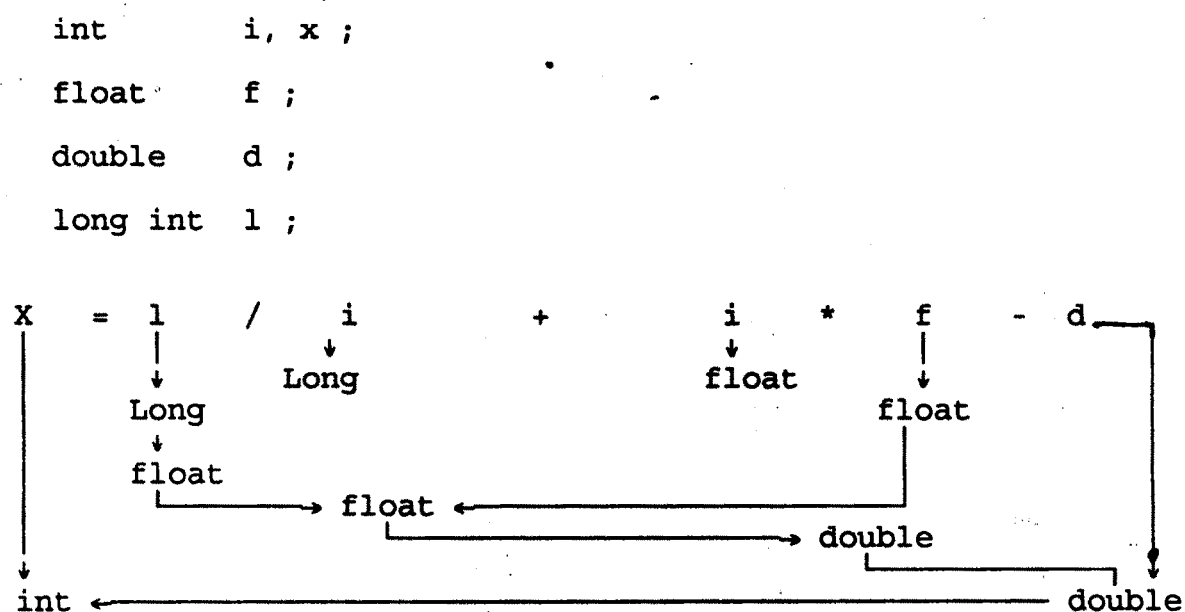
Automatic type conversion.

C permit mixing of constants and variables of different types in an expression, but during evaluation it adheres to very strict rules of type conversion. We know that the computer considers one

operator at a time, involving two operands.

If the operands are of different types, the 'lower' type is automatically converted to the 'higher' type before the operation proceeds. The result is of higher type. A typical type conversion process is illustrated in Fig. 1.1.4

Fig. 1.1.4



Given below is the sequence of the rules that are applied while evaluating expressions.

All short and char are automatically converted to int; then .

1. if one of the operand is long double, the other will be converted to long double and the result will be long double;
2. else, if one of the operands is double, the other will be converted to double and the result will be double;
3. else, if one of the operands is float, the other will convert-

ed to float and the result will be float;

4. else, if one of the operands is unsigned long int, the other will be converted to unsigned long int and the result will be unsigned long int;

5. else, if one of the operands is long int and the other is unsigned int, then :

a) if unsigned int can be converted to long int, the unsigned int operand will be converted as such and the result will be long int.

b) else, both operands will be converted to unsigned long int and the result will be unsigned long int;

6. else if one of operands is unsigned int, the other will be converted to unsigned int and the result will be unsigned int.

7. else, if one of the operand is long int, the other will be converted to long int and the result will be long int;

The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However, the following changes are introduced during the final assignment.

1. float to int causes truncation of the fractional part.

2. double to float causes rounding of digits.

3. long int to int causes dropping of the excess higher order bits.

### 1.1.6(d) Casting a value

C performs type conversion automatically. However, there are instances when we want to force a type conversion in a way that is different from the automatic conversion.

The process of a local conversion is known as casting a value.

The general form of a cast is :

(type name) expression.

Where type name is one of the standard C data types. The expression may be a constant, variable or an expression.

### 1.1.6(e) Operator Precedence And Associativity

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. There are distinct levels of precedence and an operator may belong to one of the same precedence are evaluated either from left to right or right to left, depending on the level. This is known as the associativity property of an operator.

Table 1.1.10.

Summary of c operators

Operator	Description	Associativity	Rank
()	Function call	left to right	1
[]	array element reference		
-----			
+	unary plus	right to left	2
-	unary minus		
++	increment		
--	Decrement		

Operator	Description	Associativity	Rank
!	logical negative		
~	ones complement		
*	pointer reference (indirection)		
&	address		
sizeof	size of an object		
(type)	type cast (conversion)		
-----			
*	Multiplication	left to right	3
/	Division		
%	modulus		
-----			
+	addition	left to right	4
-	subtraction		
-----			
< <	left shift	left to right	5
> >	right shift		
-----			
<	less than	left to right	6
< =	less than or equal to		
>	greater than		
> =	greater than or equal to		
-----			
= =	Equality	left to right	7
! =	Inequality		
-----			
&	Bit wise AND	left to right	8
^	Bit wise XOR	left to right	9
	Bit wise OR	left to right	10
& &	Logical AND	left to right	11
	Logical OR	left to right	12
?	conditional expression	right to left	13
-----			
=	Assignment	right to left	14
*= /= %=	operators		
-----			
+= -= &=			
=   =			
-----			
<<= >>=			
-----			
,	comma operator	left to right	15

Table 1.1.10 provides a complete list of operators, their precedence levels and their rules of association.

The groups are listed in the order of decreasing precedence (rank 1 indicates the highest precedence level and 15 the lowest).

Mathematical functions :-

Table 1.1.11 Math functions.

Function	Meaning
<b>Trigonometric</b>	
acos(x)	Arc cosine of x
asin(x)	Arc sin of x
atan(x)	Arc tangent of x
atan2(x,y)	Arc tangent of x/y
cos(x)	cosine of x
sin(x)	sin of x
tan(x)	tangent of x
<b>Hyperbolic</b>	
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sin of x
tanh(x)	Hyperbolic tangent of x
<b>Other functions</b>	
ceil(x)	X rounded upto the nearest integer
exp(x)	e to the power X ( $e^x$ )
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, $x > 0$
log10(x)	Base of 10 log of x, $x > 0$
pow(x,y)	X to the power y ( $x^y$ )
sqrt(x)	square root of x, $x \geq 0$

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.

Mathematical functions such as cos, sqrt, log etc. are frequently used in analysis of real life problems. There are systems that have a more comprehensive math library and one should consult the reference manual to find out which functions are available. Table (1.1.11) lists some standard math functions.

### 1.1.7 Managing Input and output Operations

#### Reading a character

Reading a single character can be done by using the function getchar. (This can also be done with the help of scanf function).

The getchar takes the following form :

```
variable - name = getchar ();
```

variable - name is a valid C name that has been declared as char type.

Like getchar, there is an analogous function putchar for writing characters one at a time to the terminal. It takes the form as shown below :

```
putchar (variable - name);
```

Where variable - name is a type char variable containing a character. This statement displays the character contained in the variable - name at the terminal.

Commonly used scanf format code is given in table 1.1.12

Table

1.1.12

#### Scan format codes

%c	read a single character
%d	read a decimal integer
%e	read a floating point value
%f	read a floating point value

<code>%g</code>	read a floating point value
<code>%h</code>	read a short integer
<code>%i</code>	read a decimal, hexadecimal or octal integer
<code>%o</code>	read an octal integer
<code>%s</code>	read a string
<code>%n</code>	read an unsigned decimal integer
<code>%x</code>	read a hexadecimal integer
<code>% [...]</code>	read a string of words

The following letters may be used as prefix for certain characters.

<code>h</code>	for short integers
<code>l</code>	for long integers or double
<code>L</code>	for long double

Commonly used printf format codes are given in Table 1.1.13

Table 1.1.13      Printf Format Codes

Code	Meaning
<code>%c</code>	print a single character
<code>%d</code>	print a decimal integer
<code>%e</code>	print a floating point value in exponent form
<code>%f</code>	print a floating point without exponent
<code>%g</code>	print a floating point value either e type or f type
<code>%i</code>	print a single decimal integer
<code>%o</code>	print an octal integer, without leading zero
<code>%s</code>	print a string
<code>%n</code>	print an unsigned decimal integer
<code>%x</code>	print a hexadecimal integer without leading ox.

The following letters may be used as prefix for certain conversion characters

<code>h</code>	for short integers,
<code>l</code>	for long integers or double
<code>L</code>	for long double



### 1.1.8 Decision making and branching

C program is a set of statements which are normally executed sequentially in the order in which they appear. This happens when no options or no repetitions of certain calculations are necessary. However in practice, we have a number of situations where we may have to change the order of execution of statement based on certain conditions, or repeat a group of statements until certain specified conditions are met. This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly.

C language possesses such decision making capabilities and supports the following statements known as control or decision making statements.

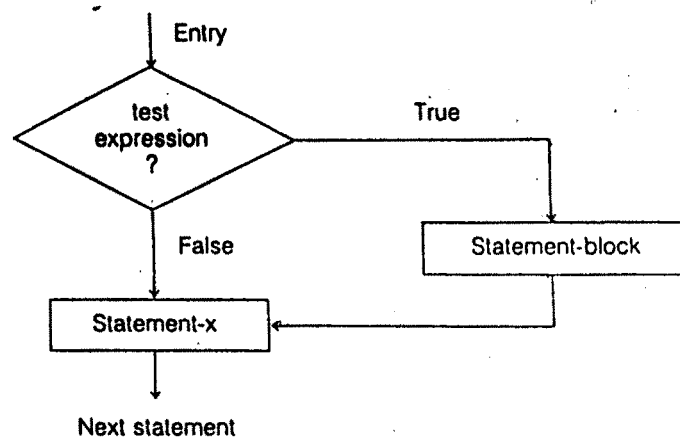
1. if statements
2. switch statement
3. conditional operator statement
4. goto statement

#### 1:Decision making if statement

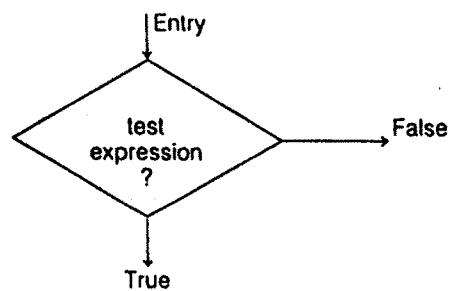
The if statement is used to control the flow of execution of statements. It is basically a two-way decision statement and is used in conjunction with an expression. It takes the following form

```
if(test expression)
```

It allows the computer to evaluate the expression first and



**Fig.1. 1.6**  
Flowchart of simple if control



**Fig.1.1.5**

Flow chart of if-statement

then, depending on whether the value of expression (relation or condition) is "true"(nonzero) or "false"(zero), it transfers the control to a particular statement. This type of program has two paths to follow, one for the true condition and the other for the false condition as shown in Fig 1.1.5

The if statement is implemented in different form depending on the complexity of conditions to be tested.

- A) Simple if statement
- B) if\_\_\_\_\_else statement
- C) Nested if\_\_\_\_\_else statement
- D) elseif statement

A) Simple if statement-The general form of a simple if statement is

```
if(test expression)
{statement -block;
}
```

statement-x;

The statement block statement may be single statement or a group of statements. If the test expression is true, the statement block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x. When the condition is true both the statement block and the statement-x are executed in sequence.

This is illustrated in Fig 1.1.6

Fig 1.1.6. Flowchart of simple if control.

B) The if\_\_\_\_else statement

The general form is,

```
if(test expression)
{
True block statement(s)
}
else
{False block statement}
statement-x;
```

If the test expression is true ,then the true block statement(s) following if statement are executed ; otherwise, the false-block statement(s) are executed.

C) When series of decisions are involved,we may use more than one if\_\_\_\_else statements in nested form.

D) elseif-There is another way of putting ifs together ,when multipath decision are involved.A multipath decision is a chain of ifs in which the statement associated with each else is an if.

## 2. The switch statement

We have seen that when one of the many alternatives is to be selected, we can design program using if statements to control the selection. However, the *complexity* of such a program increases. When the alternatives increases.The program becomes difficult to read and follow.Fortunately, c has a built in multiway decision statement known as Switch. The switch statement tests the

value of a given variable(or expression) against a list of case values and when a match is found ,a block of statements associated with that case is executed. The general form of the switch statement is as shown below:

```
switch(expression)
```

```
{
```

```
case value-1:
```

```
block-1
```

```
break;
```

```
case value-2:
```

```
block-2
```

```
break;
```

```
_____
```

```
_____
```

```
default:
```

```
default block
```

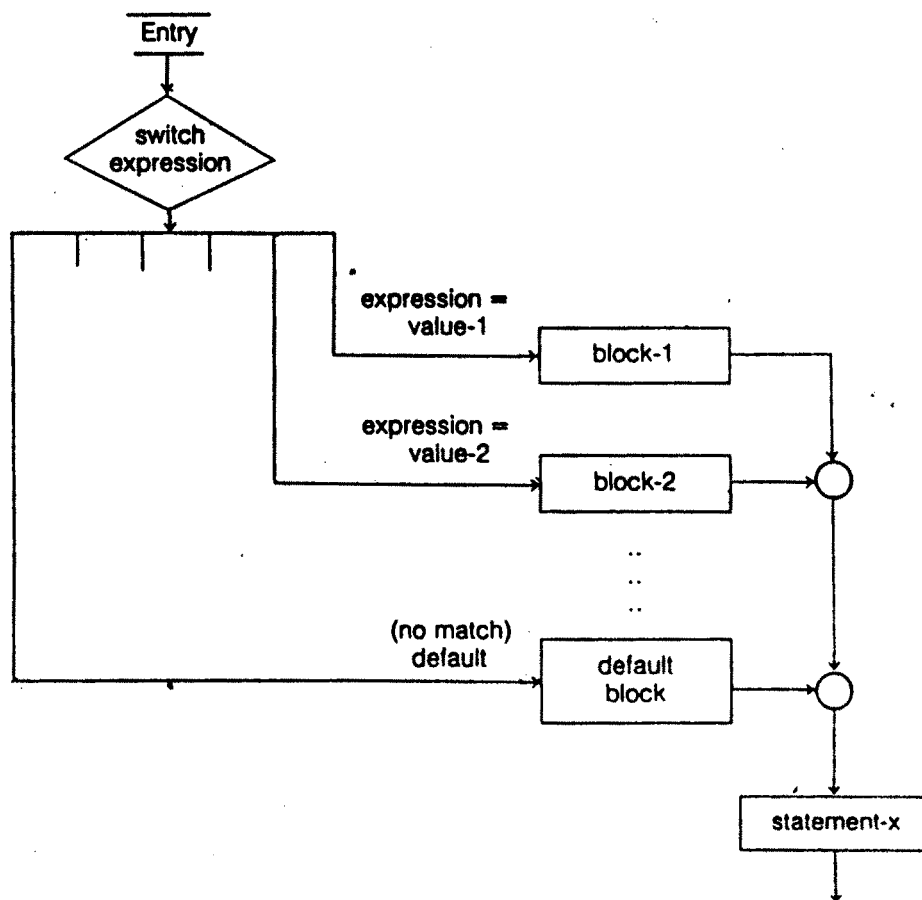
```
break;
```

```
}
```

```
statement-x;
```

Here, value-1, value-2,\_\_\_- are constants or constant expressions and block-1, block-2,\_\_\_-are statement list and may contain zero or more statements.

The selection process of switch is illustrated in the flowchart shown in the figure 1.1.7.



**Fig.1.1.7**  
**Flowchart of switch statement**

## C) The ?: operator

The c language has an operator, useful for two-way decisions

For example, the segment

```
if(x<0)
```

```
flag = 0;else
```

```
flag = 1;
```


can be written as, `flag =(x<0)?0:1;`

## D) GoTo staement

C supports the goto statement to branch unconditionally from one point to another in the program.

The general form of goto and lable statement(any valid variable name) is shown below:

<pre>goto lable: _____ ----- lable: statement; forword jump statement; Forword jump</pre>	<pre>lable: statement ----- ----- goto lable;</pre>
---	---



## 1.1.9 Decision making and looping

In c it is possible to execute a segment of a program repeatedly by introducing a counter and later testing it using the if statement. While this method is quite satisfactory for all practical purposes, we need to initialize and increment a counter and test

its value at an appropriate place in the program for the completion of the loop.

In looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied. A program loop therefore consists of two segments, one known as the body of the loop and the other known as the control statement. The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

Depending on the position of the control statement in the loop, a control structure may be classified either as the entry controlled loop or as the exit controlled loop. The flowcharts in Fig 1.1.8 (a) and (b) illustrates these structures. In entry controlled loop, the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed. In the case of an exit controlled loop, the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time.

The test condition should be carefully stated in order to perform the desired number of loop executions. It is assumed that the test condition will eventually transfer the control out of the loop. In case, due to some reason it doesnot do so, the control sets up an infinite loop and the body is executed over and over again.



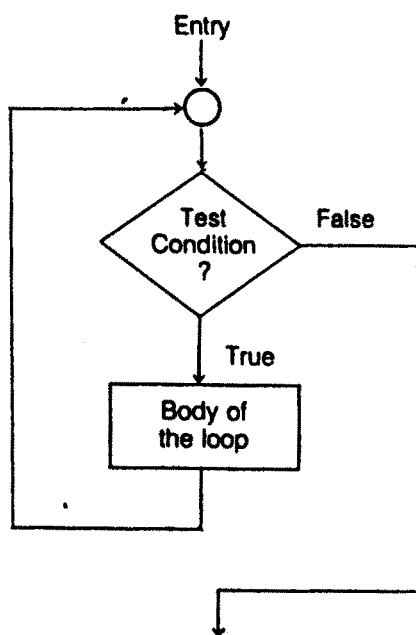


Fig.1.1.8 (a)  
Entry controlled loop

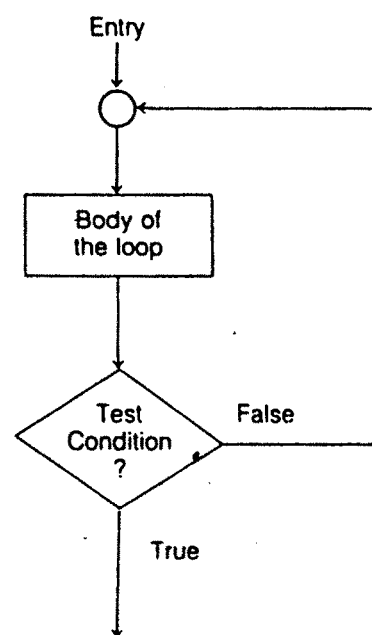


Fig.1.1.8(b)  
Exit controlled loop

A looping process , in general, would include the following four steps:

1. Setting and initialization of a counter.
2. Execution of statements in the loop.
3. Test for a specified condition for execution of the loop
4. Incrementing the counter.

The test may be either to determine whether the loop has been repeated the specified number of times or to determine whether a particular condition has been met.

The c language provides three loop constructs for performing loop operations. They are :

1. While statement.
2. Do statement.
3. For statement.

1. While statement: The basic format of while statement is

```
while(test expression)
```

```
{
body of the loop
}
```

The while is entry controlled loop in which the test condition is evaluated first and if it is true the body of the loop is executed. This process continues until the test condition becomes false and the control is transferred out of loop. On exit, the program continues with the statement immediately after the body of the loop.

## 2. Do statement:

on some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the do statements . This takes the form:

```
do{
body of the loop}
while(test condition);
```

On reaching the do statement , the program proceeds to evaluate the body of loop first. At the end of the loop, the test-condition in the while statement is evaluated until test condition becomes false,the loop will be terminated and the control goes to the statement that appears after while statement.

Thus while loop is exit control loop.

## For statement:

The for loop is another entry controlled loop.

The general form of the for loop is:

```
for(initialization;test condition;increment)
{
body of the loop
}
```

The execution of the for statement is as follows:

1. Initialization of the control variable is done first , using assignment statements with loop control variables.
2. The value of the control variable is tested using the test condition. The test condition is relational expression, that

determines when the loop will exit . If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.

3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating last statement in the loop. Now the control variable is incremented using an assignment statement and the new value of the control variable is again tested to see whether it satisfies the loop condition . If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

#### 1.1.10 Jumps in loops

Loop performs a set of operations repeatedly until the control variable fails to satisfy the test condition . The number of times a loop is repeated is decided in advance and the test condition is written to achieve this. Sometimes, when executing a loop it becomes desirable to skip a part of the loop or to leave the loop as soon as a certain condition occurs. C permits a jump from one statement to another within a loop as well as a jump out of loop.

#### 1.1.11 Arrays

Arrays are helpful in storing and retrieving data of homogeneous type. An array describes a contiguously allocated nonempty set of objects with the same basic type. Hence, many number of same type

of variables can be grouped together with a common name as an array.

### **Array Declaration**

Arrays also should be declared as an basic type with subscripted numbers given within square brackets . By declaring an array, the specified number of locations are fixed in the memory. Naming an array follows the same rules as that of a variable. There is no limit for array dimensioning in c language. The limit can only be the computers main memory .

#### **SYNTAX:**

```
type var[n];/*One dimentional array*/
```

type -variable type, var -variable name, n -a positive constant.

#### **1.1.12(a) :-**

**Strings/ Character array and initialization.**

An array of characters can be declared and hence a chain of characters called as string can be stored in that array.

Arrays can be initialized at the time of declaration as the other variable. example,

```
int reg_no[5] = {10,20,30,40,50};
```

```
char city [3] = {'m','a','\0'};
```

```
char city [7] = "madras"
```

\0 inidicates end of string character.

#### **1.1.12 (b) Rules to initialize array**

1.Array can be initialized only with constants.

2.Numeric array automatically initialized with 0.

- 3.The whole array can be referenced by the array name.
- 4.Fewer initializers than the specified size are allowed.
- 5.Too many initializers more than specified size, gives erroneous result.
- 6.The middle elements can not be initialized.
- 7.If all the elements should be initialized with a specific number,the repetition of the data can not be avoided.
- 8.Arrays can be initialized without mentioning the number of elements.

1.1.13

**C-Pointers:**

C provides the appreciating feature of data manipulation with the adress of the variables and hence the execution time is very much reduced. Such concept is possible with spcial data type called pointer. Pointer is variable which holds the address of another variable. This allows indirect access to the objects.

A varible can be declared as a pointer variable and can point to the starting byte address of any data type variable.

1.1.13 (a) Declaring a pointer variable:

To declare and refer a pointer type variable, c provides two special unary operators & and \*. A pointer variable can be declared in the same way as the other variables, but an asterisk symbol should precede the variable name.

Example:

char \*c; Here c is a pointer variable pointing to a character type variable.Hence a pointer variable can be declared with the

type of what it is pointing to, with the asterisk symbol preceding the variable name.

They can also be initialized with the address of the variable.

Example:

```
int i, *j = &i;
```

Here j is a pointer variable initialized with the address of i.

#### 1.1.14

#### C- Files:

In the previous discussion, data stored in all these variables are only temporary. When the program execution is over, all the entered data will be lost and for the subsequent execution again the data should be fed. "In real time environments, the data fed should be stored permanently for subsequent processing". c provides the new data type FILE and file operations, through which the data can be stored in a secondary storage device and hence enables permanent storage.

#### FILE DATA TYPES:

This data type name should always be given in capital letters. This type expects the variables to be pointer variables. Example:

```
FILE *ptr *, *fopen();
```

Here ptr is a FILE type pointer variable fopen() is a function, which returns a FILE type pointer.

FILE OPEN WITH fopen(). To open file the function fopen() is used.

This function returns a pointer to a file. The usage is as follows:

Syntax:

```
file_pointer = fopen("datafile","mode");
```

Before this assignment, `file_pointer` and `fopen()` should be declared as `FILE` pointer type variables. `file_pointer` this is the logical name given to the data file and throughout the program, the data file (physical file) is referred by this file pointer. `data file` is a file name in which the data is stored or retrieved. This is the physical file name for the data in the secondary storage device.

`mode` is file in `c` and can be opened in various ways. This mode decides the read/write operations with data file .

## 1.2 FILE OPEN MODES IN `fopen()`:

Consider the syntax shown above for file open modes.

1.2.1 `w`(write) mode : C compiler checks for the existence of data file specified , If not available, it creates a file in the name of "data file". If the "data file" exists, the old data is deleted and new fresh data is written in the file. If for some reason, a file cannot be created (may be secondary storage is full), `NULL` is returned to `file_pointer`. Example:

```
ptr = fopen("pay.dat","w"); file = fopen("Inventry","w");
```

Legal values for `mode` are shown in Table 1.2.1 given below



Table 1.2.1

MODE	MEANING
"r"	Open a text file for reading
"w"	Creates a text file for writting
"a"	Append to a text file
"rb"	Open a binary file for reading
"wb"	Create a binary file for reading
"ab"	Append to binary file
"r+"	Open a text file for read/write
"w+"	Create a text file for read/write
"a+"	Append a text file for read/write
"r+b"	Open a binary file for read/write
"w+b"	Create a binary file for read/write
"a+b"	Append a binary file for read/write

### 1.2.2 FILE CLOSE WITH fclose():

All the files that are opened should be closed after all input and output operations with the file, to prevent the data from getting corrupted.

Syntax: `fclose(file_pointer);` Where `file_poiter` is returned value of `fopen()`. After this `fclose`, no I/O with the file can take place. Further access and storage, the file has to be opened again.

### 1.2.3 GRAPHICS FEATURES IN C - LANGUAGE :

ANSI C does not define any text screen or graphics functions

because of capabilities of diverse hardware environments. But Turbo C version 1.5 and higher support extensive screen and graphics support facilities.

#### 1.2.4

#### MODES:

Basically there are two modes namely, text mode and graphics mode. In graphics mode any type of figures can be displayed, captured and animated.

#### 1.2.5 GRAPHICS MODE :

In this mode it is possible to display text as well as graphical figures. The basic element of the graphics is picture element or pixel. The monitor type can be monochrome, bga, cga, ega etc. To execute these functions <graphics.h> file should be included in the c program.

1.2.5 (a) `initgraph(int driver, int mode, char path);`

Whenever any graphics figure has to be drawn this `initgraph()` function should be used to initialize the graphics mode on the video.

Example:

```
int driver, mode;
```

```
driver = 1;
```

```
mode = g;
```

```
initgraph(&driver, &mode, " ");
```

1.2.5(b) `putpixel(int x, int y, int color);`

This function illuminates the pixel represented by x and y coordinates in the represented by color.

### 1.2.6 Application of C - program

C - language has many applications as\_ mathematical, Scientific, Engineering and commerical. Application software is programs written to solve specific problems. These application software is of two types:

1. General purpose.
2. Specific purpose software.

Based on the effective utilization of the system, the specific purpose software applications are divided into two types. (a) scientific applications and (b) administrative / business applications.

Scientific applications involve large number of calulations and less input, output of data.Science laboratories of educational institutions,industries dealing with many research oriented projects, space technology etc. have benefitted by the use of c-language.

### 1.2.7 Some examples of C-programs:

Fig 1.9 : e.g of array

```
#include<stdio.h>

main()
{
int i[10];
int j;
/* Array initialization*/
float p[10] = {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.10};
```

```

for(j = 0;j<=10;j++)
{
i[j] = j; /*This reserves 10 integers*/
printf("i[j]=%d p[j]=%f",i[j],p[j]);
}}

```

Fig 1.10- e.g TO CALCULATE PEREODIC TIME OF ASTABLE MULTIVIBRATOR

```

#include<stdio.h>
#include<math.h>
#define A 0.693
#define C 0.01
main()
{
int r1,r2,c; /*r1,r2,c represent resistances in ohm and capacity
in microfarad*/
int r;
float t; /* t is periodic time*/
for(r2 = 1;r2 < 50;r2 = r2+1)
{
r1 = 10;
r = r1+(2*r2);
t = A*r*c;
printf("r1=%d r2=%d t=%f",r1,r2,t);
}
getch();}

```

Fig 1.11 e.g- TO CALCULATE MODULATION INDEX

```
#include<stdio.h>
#include<math.h>
#define MAX 100
main()
{
    int i;
    float m,n,mod;
    float vmax, vmin;
    printf("Enter values of vmax and vmin");
    for(i = 0;i < 2;i = i+1)
    {
        scanf("\n %f",&vmax);
        scanf("\n %f",&vmin);
    }
    for(i = 0;i < 2;i = i+1)
    {
        m = (vmax-vmin);
        n = (vmax+vmin);
        mod = m/n;
    }
    printf("\n vmax=%f vmin=%f mod=%f",vmax,vmin,mod);
}
```

## CHAPTER - I

- (1) E. Balagurusamy.

Programming in ANSIC, (2nd Ed.), (P. nos (1-3), (18-29), (47 - 67), (85 - 87), (99 - 119), (129 - 134), (142), (159 - 162), (280 - 284), (309 - 312) ) Tata Mc. Graw - Hill Publishing Company Limited. (1994).

- (2) J. Jayasri.

The 'C' Language trainer with C graphics and ( ++, (P.nos. (192 - 195), (198), (200) ) Wiley Eastern Limited (1993).