MAJOR BASED ELECTIVE II
MICROPROCESSORN AND C PROGRAMMING

UNIT IV

Basic Structure of a C programs-Character set-C Tokens-Keywords and Identifiers-Constant-Variables - Data Types – Declarations –Assigning Value to Variables-Symbolic Constants– Operators and Expressions –Arithmetic Operators-Relational ,Logical and Assignment Operators-Increment and Decrement Operators-Conditional Operators, Bitwise and special Operators-Arithmetic expressions.

UNIT V

Data Input and Output –getchar ,putchar,scanf,printf,gets,puts functions –Decision Making and Branching- IF,IF-ELSE,ELSE – IF LADDER,SWITCH,BREAK,COUNTINE,GOTO-Decision Making and Looping- WHILE , DO – WHILE, FOR,NESTED LOOPS-Arrays(One,Two and Multi- Dimensional Arrays)-Declaration, Initialization of Arrays.

UNIT IV

History of C

- ALGOL was the first computer language to use a block structure.
- In 1967, Martin Richards developed a language called BCPL [Basic Combined Programming Language] primarily for writing system software.
- In 1970, Ken Thompson created a language using many features of BCPL and called it simply B.
- C was evolved from ALGOL, BCPL and B by Dennis Ritchie at Bell laboratories in 1972.
- C was many concepts from these languages and added the concept of data types and other powerful features.
- Since it was developed along with the UNIX operating system, It is strongly associated with UNIX.
- During 1970's C had evolved into what is now known as " Traditional C".
- To assure that the C language remains standard in 1983; American National Standards Institute [ANSI] appointed a technical committee to define a standard for C.
- The committee approved a version of C in 1989 which is now known as ANSI.

1960	ALGOL	International Group
1967	BCPL	Martin Richards
1970	B	Ken thompson
1972	Traditional C	Dennis Ritchie
1989	ANSI C	ANSI Committee
1990	ANSI / ISO C	ISO Committee

Importance of C :

- It is a robust language whose rich set of built – in – functions and operators can be used to write any complex program.
- The C compiler combines the capabilities of an assembly language with the features of a high – level language and therefore it is well suited for writing both system software and business package.
- It is many times faster then BASIC.
- Several standard functions are available which can be used for developing programs.
- C is highly portable this means that c program written for one computer can be run on another with little or no modifications.
- C language is well suited for structured programming thus required the user to think of a program in terms of function modules or blocks.
- Another important feature of c is its ability to extend itself.

Evaluation of the Structure of a C program

- Every "C" program must have one main() function section.
- 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 the closing braces. The program execution begins at the opening brace and ends at the closing brace.
- All statement in the declaration and executable parts end with semicolon.

Example :

```
DOCUMENTATION SECTION
/* Addition of 2 numbers */
LINK SECTION
#include <stdio.h>          //    Standard I/O File    //
#include <conio.h>               //     Console I/O File     //
MAIN() FUNCTION SECTION
void main()
{
DECLARATION PART
int a = 10;
int b = 12,c;
//EXECUTABLE PART
C = a+b;
Printf("Addition = %d"'c);
}
OUTPUT
     Additon = 22
```

Constants, Variables and Data Types :

CHARACTER SET :

- The characters in C are grouped into the following categories
  - i) Letters
  - ii) Digits
  - iii) Special Characters
  - iv) White Spaces

Letters :

★ Upper Case A……Z

★ Lower Case a……z

Digits :

★ All decimal digits 0…….9

Special Characters :

,	Comma	&	Ampersand
;	Semicolon	*	Asterisk
:	Colon	!	Exclamation mark
~	Tilde	‘	Apostrophe
/	Slash	_	Underscore

White Space :

★ Blank Space

★ Horizontal tab

★ Carriage return

★ New line

★ Form Feed

Trigraph Characters :

➢ Ansi C introduces the concept of "trigraph" sequence to provide a way to enter certain characters that are not available on some keyboards.

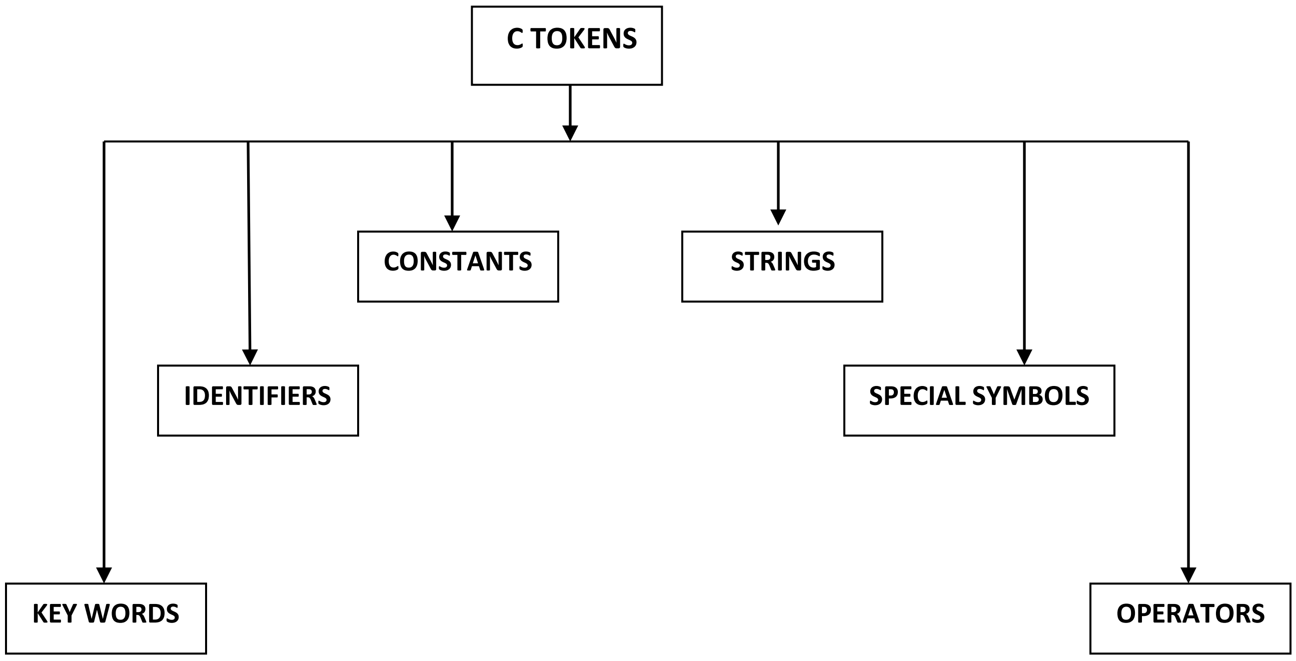
➢ Each trigraph sequence consists of three characters [two question mark followed by another character].

Trigraph Sequence	Translation
?? =	# number sign
?? (	[left bracket

?? <	{ left brace
?? /	\ back slash
?? \	^ caret

C Tokens :

- In a C Program the smallest individual units are known a C Tokens.
- C has six types of C Tokens.



Key Words :

- Every C word is classified as either a keyword or an identifier.
- All keywords have fixed meanings and these meanings cannot be changed.
- All keywords must be written in lower case.
  - Eg : int, float, char, double, if, do, else, static, main, scanf, printf.

Identifiers :

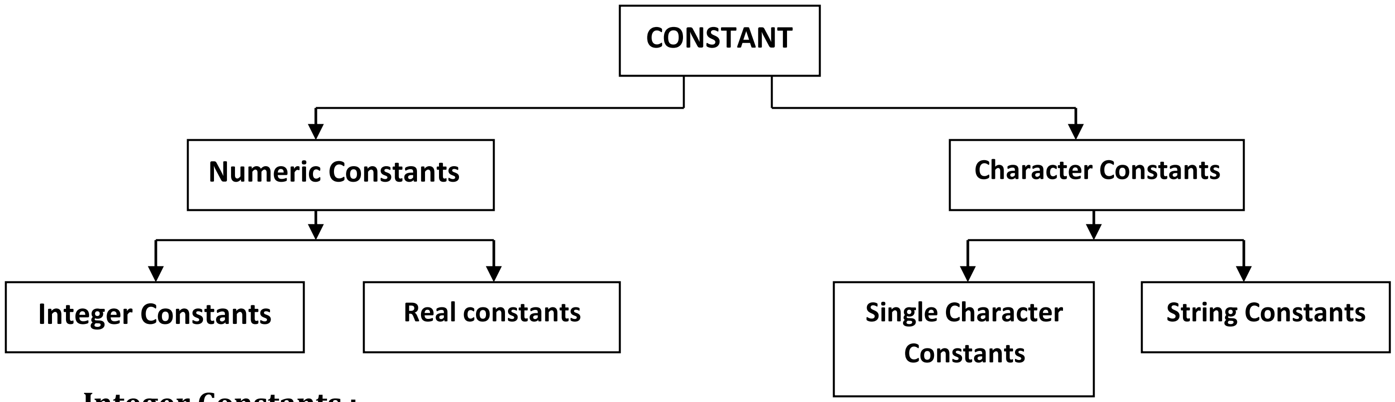
- Identifiers refer to the names of variables, function and arrays.
- These are user-defined names and consists of sequence of letters and digits, with a letter as a first character.
- Both uppercase and lowercase letters are permitted. The underscore characters is also permitted in identifiers.

Rules for Identifiers :

1) First character must be a letter or underscore.
2) Only first 31 characters are significant.
3) Cannot use keyword.
4) White spaces are not allowed.

Constants :

- Constants in C refers tp fixed values that do not change during the execution of a program.
- C supprot several types of constants.



Integer Constants :

- An integer constants refers to a sequence of digits. There are three types of integers namely Decimal Integer, Octal Integer and Hexadecimal Integer.
- Decimal Integers consists of a set of digits 0 through 9 preceeded by optional – (or) + sign.

  Eg : 123, -321, 0...

- An Octal Integer constant consist of any combination of digits from 0 to 7 with leading zero.

  Eg : 037, 0435..,

- A sequence of digits preceded by OX or ox is considered as hexadecimal integer. They also include alphabets A to F or a to f. The letter A to F represents the numbers from 10 to 15.

  Eg : ox2, OXA1

Real Constants :

- A number with decimal point is called real or floating point.

  Eg : 12.4, 0.25.
- A real number may also be expressed in exponential or scientific notation.
- The general form is **"mantissa e exponent"** the mantissa is either a real or an integer. The exponent is an integer number with an optional + or _ sign.

  Eg: 0.65e4, 1.4e+2.

Single Character Constants :

- A single character constants contains a single character enclosed within a pair of single quote mark.

  Eg : '5' , 'A'

String Constants :

- A string constant is a sequence of characters enclosed in double quotes.

  Eg: "Hai"

VARIABLES :

- A variables is a data name that may be used to store a value.
- A variable may take different values at different times during execution.

Rules for naming a variable :

- They must begin with an alphabet.
- Some system permits underscore as the first character.
- It should not be a keyword.
- White space is not allowed.

Valid => Sum, +1, S-value,

Invalid => % sum, 5+0+, int.

DATA TYPES:

- C supports three classes of data types.
  - i) Primary [or] Fundamental [or] Built-in data types.

ii) Derived data types.

iii) User – Defined data types.

Primary Data Types:

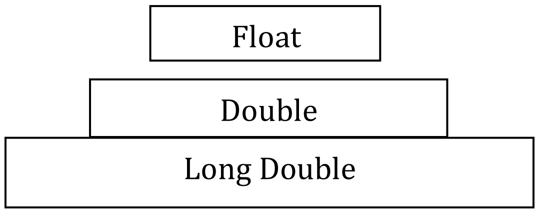
- C Supports four fundamental data types, They are
  - i) Integer
  - ii) Floating Point
  - iii) Character
  - iv) Void.

Integer types:

- Integers are whole number C has three classes of Integer storage namely,
  - ★ Short int –> 1 byte –> (range) -128 to +127
  - ★ int –> 2 byte –> (range) -32768 to +32767
  - ★ Long int –> 4 byte –> (range) -2,147,483,648 to +2,147,483,647.

Floating Point Type :

- In C floating point numbers are defined by the keyword float. The storage capacity of float is 4 bytes or 32 bits, The range of float data type is from 3.4e – 38 to 3.4e + 38.
- When the accuracy of float is not sufficient the type double can be used. Double data type uses 64 bits.
- To extend the accuracy we may use long double which uses 80 bits.



Character Data Type:

- In C character are defined by the keyword char.
- Characters are usually stored in 8 bits or 1 byte.
- It may be signed or unsigned

- Signed char values from -128 to 127 unsigned char values from 0 to 255.

Void Type :

- It has no values.
- This is usually used to specify the type of function.
- The type of function is said to be void, when it does not return any value to the calling function.

Declaration of Variables:

- Declaration does two things
  - i) It tells the name of the variable to the compiler.
  - ii) It specifies what type of data the variable will hold.
- The variable must be declared before they are used.
- There are two types of declaration.
  - Primary type declaration.
  - User _ defined type declaration.

Primary Type Declaration :

- A variable can be used to store the value of any data type

Syntax :

```
Datatype v1,v2,.....vn;
```

- Where v1,v2 are the names of variable they are separated by commas.

  Eg : float a, xy;

User - Defined data type Declaration :

C supports user defined data types like typedef, enum, structures, union etc...

Assigning Value to Variables:

- The variable must be declared before assigning value to the variable.

Syntax:

```
Variable Name  = Value;
```

- The equal to sign is used to assign value to variable.

  Eg: a=10.0;

  xy=25.0;

Symbolic Constant:

- A symbolic constant is a name given to numeric value or a constant value or string value.
- Symbolic constant are also known as constant identifers.

Syntax:

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

Eg: #define PI=3.1415

#define MAX 500

Rules:

- Symbolic constant name written in uppercase.
- It must not end with semicolon.
- It must begin with #define.

Operators :

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulation.
- Operators are used in programs to manipulate data and variables.
- C operators can be classified into a number of categories.
- They include,

  i) Arithmetic operators

  ii) Relational operators

  iii) Logical operators

  iv) Assignment operators

  v) Increment and Decrement operators

  vi) Conditional operators

vii) Bitwise operators

viii) Special operators

Arithmetic Operators :

- C provides all the basic arithmetic operators. They can operate on any built –in data type allowed in C.

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

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

<u>Integer Arithmetic :</u>

- When both the operands in a single arithmetic expression such as a+b are integers, the expression is called as an integer expression and the operation is called ARITHMETIC..,

<u>Eg :</u> a-b = 10

a+b = 18

Relational Operators :

- C supports six relational operators in all

Operator	Meaning
<	Less than
<=	Less than or equal to

>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

- A simple relational expression contains only one relational operator and takes the following forms.

ae – 1 relational operator ae -2

Where,

ae – 1 and ae – 2 are arithmetic expressions which may be simple constants, variables or combination of them.

Logical Operators :

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

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

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

  Eg : a>b && x == 10;

- An expression which combines two or more relational expressions, is termed as a logical expression.

Eg : if (age > 55 && salary < 1000).

Assignment Operators :

- Assignment operators are used to assign the result of in expression to a variable. In addition, C has a set of SHORTHAND assignment operators of the form,

$\boxed{\text{v op = exp}}$

Where, **v** is a variable, **exp** is an expression and **op** is a C binary arithmetic operator.

- The assignment operator statement v op = exp; is equivalent to v = v op (exp);

Eg : x+ = y+1;

Same as x = x+(y+1);

Increment and Decrement Operator :

++ and –

- The operator ++ adds 1 to the operand, while – subtracts 1 from the operand. Both are unary operators and takes the following form.

++m ; or m++;

--m; or m--;

i) ++m is equivalent to m=m+1

ii) --m is equivalent to m=m-1

m=5; y=m++

then the value of y would be 5 and m would be 6.

- A prefix operator ++m first adds 1 to the operand and then the result is assigned to the variable on left.

- A postfix operator m++ first assign the value to the variables an left and then increases 1 to the operand.

Conditional Operator :

- A ternary operator pair "?" is available to construct conditional expressions of the form.

```
exp1?  exp2 :  exp3;
```

Where, exp1, exp2 and exp3 are expressions.

- The operator ?: works as follows :
  - i) exp1 is evaluated first. If it is non zero (true), then the expression exp2 is evaluated and becomes the value of the expressions.
  - ii) If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.

<u>Eg :</u> a = 10; b = 15;

x = (a>b) ? a : b ;

Bitwise Operator :

- 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 operator may not be applied to float or double.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift Left
>>	Shift Right

Special Operators :

- C supports some special operators of interest such as comma operator, size of operator, pointer operators (& and *) and member selection operator (. and ->).

Comma Operator :

- The comma operator can be used to link the related expressions together.

Eg: value = (x=10, y=5, x+y);

Sizeof Operator :

- The sizeof is a compile time operator and when used, with an operand, it returns the number of bytes the operand occupies.

Eg : m = sizeof (sum);

n = sizeof (long int);

k = sizeof (235L);

Arithmetic Expressions:

- An arithmetic expressions is a combination of variables constants and operators arranged as per the syntax of the language.

Evaluation of Expressions :

- Expressions are evaluated using an assignment statement of the form.

Variable = expressions

- Variable is any valid C variable name. The expression is evaluated first and the result then replaces the previous value of the variable on the left hand side.

Eg : x = a * b - c;

y = b / c * a;

z = a – b / c + d;

Program :

```
Main()
{
        Float  a, b ,c, x, y, z ;
        a = 9;  b =12; c = 3;
        x = a – b / 3 \ = c * z - 1
        y = a – b / (3 + c) *  (z - 1);
        z = a - (b / (3 + c) * z) - 1;
        Printf("X = %f  \n", x);
        Printf("Y = %f  \n", y);
        Printf("Z = %f  \n", z);
}
```

Precedence of arithmetic operator :

- An arithmetic expression without parenthesis 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.

  i) High Priority * ? %

  ii) Low Priority + -

  ★ *X = a – b / 3 + c * 2 – 1*

  When a = 9 b = 12 and c = 3 the statement becomes,

  ★ *X = 9 – 12 / 3 + 3 * 2 – 1 and is evaluated as follows.*

  Step 1 : x = 9 – 4 + 3 * 2 – 1

  Step 2 : x = 9 – 4 + 6 – 1

  Step 3 : x = 5 + 6 – 1

  Step 4 : x = 11 – 1

  Step 5 : x = 10

Rules for evaluation of Expression:

- First, parenthesized sub expression from left to right are evaluated.
- If parenthesis are nested, the evaluation begins with the innermost sub-expression.
- The precedence rule is applied in determining the order of application of operators in evaluating sub expression.
- The associativity rule is applied when two or more operators of the same precedence level appear in a sub expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.

- When parenthesis is used, the expressions with in parentheses assume highest priority.

Type Conversion in Expression :

- There are two types of conversions, they are as follows :
  - i) Implicit type conversion
  - ii) Explicit type conversion

Implicit Type Conversion :

- C permits mixing of constants and variables of different types in an expression.
- This automatic conversion is known as implicit type conversion.
- If the operands are of different types, the "lower" type is automatically converted to the "higher" type before the operation proceeds.

<u>Rules :</u>

- All short and char are automatically converted into int; then
  1) If one of the operands is long double and the other will be converted to long double.
  2) Else, if one of the operands is double, the other will be converted and the result in double.
  3) Else,. If one of the operands is float, the other will be converted 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 results will be unsigned long int.
  5) Else, if one of the operands is long int and the other is unsigned int, then the unsigned int can be converted into long int.

Explicit Conversion :

- There are instances when we want to force a type conversion in a way that is different from the automatic conversion is known as explicit conversion.
- The process of such a local conversion is known as Explicit Conversion or CASTING A VALUE.

Syntax :

```
(type-name) expression
```

Example	Action
X = (int) 7.5	7.5 is converted to integer by truncation.
A = (int)21.3 / (int)4.5	Evaluated as 21/4 and the result would be 5.

Built-in Functions:

- The C standard library provides numerous built-in functions that your program can call. For example, function **strcat()** to concatenate two strings, function **memcpy()** to copy one memory location to another location and many more functions.

- A function is known with various names like a method or a sub-routine or a procedure etc.These functions can perform task such like string handling, mathematical computations, input/output processing, memory allocation and several other operating system services.

- These functions are included in the c programm by adding their header file in the starting of the c program.
- Some of the header files are:

Math.h	- Defines mathematical functions.
String.h	- Defines string manipulation functions.
Time.h	- Containg time and date handling functions.

UNIT V

Data Input and Output Control Statements

Output Operations:

★ Unformated I/P and O/P Operations

★ Formated I/P and O/P Operations

Unformated I/P and O/P Operations:

Reading a Character :

- ★ Reading a single character can be done by using the function get char()
- ★ The getchar() takes the following form
    - ▪ Variable-name = getchar();

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

Eg : char name

Name = getchar();

Writing a Character :

- ➢ A putchar() is used for a single character to the terminal.
- ➢ The putchar() takes the following form.

Putchar (variable-name);

Where, variable-name is a type char variable containing a character.

Eg : answer = 'Y';

Purchar (answer);

Example :

```
# include <stdeio.h>
# incluce<conio.h>
Main()
{
        Char alphabet;
        Printf("Enter an alphabet");
        Putchar ("\n");
        Alphabet = getchar();
        If  (is lower (alphabet))
                Putchar (to upper (alphabet));
        Else
```

```
                    Putchar(to lower (alphabet));
        }
```

Ouput:

```
Enter an alphabet:  b
B
```

Formatted I/P and O/P Operations:

Formatted Input :

- Formatted input refers to an input data that has been arranged in a particular format.
- The general form of scanf() is

  scanf("control string", &arg1, &arg2,.....& argn)

- The control string specifies the field format in whch the data is to be entered and the arguments arg2, arg2,.....argn specifies the address of locations wh3re the data is stored.
- Control string and arguments are separated by commas.

Inputting Integer Numbers :

- The field specification for reading an integer number is %wd
- W is an integer number that specifies the field width of the number to be read and d known as datatype.

  Eg : scanf("% 2d %sd", & num1, & num2);

  50 31426

- The value 50 is assigned to num1 and 31426 to num2. Suppose the input data is as follows 31426   50
- The variable num1 will be assigned 31 because of %2d abd num2 will be assigned 426.
- The value 50 that is unread will be assigned to the first variable in the scanf next call.

Inputting Character Strings:

- Follwoing are the specifications for reading character strings

  %ws (or) %wc

Reading mixed datatype:

- It is possible to use one scanf statement to input a data line containing mixed mode data.

<u>Eg:</u> scanf(“%d %c %f %s”, &count, &code, &ration; name);

15 p1.575 coffee

Rules for scanf

- Each variable to be read must have field specification.
- For each field specification there must be a variable address of proper type.
- Any non-white space character used in the format string must have a matching character in the user input.
- Never end the format string with whitespaces. It is a fatal error!
- <u>The scanf reads until :</u>
  - A white space character is found in a numeric specification (or)
  - The maximum number characters have been read (or)
  - An error is detected (or)
  - The end of file is reached.

Scanf code format

Code	meaning
%c	read a single character
%d	read a decimal integer
%e, %f, %q	read a floating point value
%h	read a short integer
%i	read a decimal hexadecimal or octal
%o	read a octal integer
%s	read a string
%u	read a unsigned decimal integer
%x	read a hexadecimal integer

Formatted Output :

➢ The general form of printf statement is printf("control string", arg1, arg2,.... arg n);

Output of Integer Numbers:

➢ The format specification for printing an integer no is %wd, where w specifies the minimum field width for the output d specifies that the value to be printed is an integer.

Format	Output
Printf ("% d", 9876)	$\boxed{9}\boxed{8}\boxed{7}\boxed{6}$
Printf ("% 6d", 9876)	$\boxed{}\boxed{}\boxed{9}\boxed{8}\boxed{7}\boxed{6}$
Printf (""% -6d", 9876)	$\boxed{9}\boxed{8}\boxed{7}\boxed{6}\boxed{}\boxed{}$

Output of Real Numbers:

➢ The output of a real number may be displayed in decimal notation using the following format specification.

% wpf

- The integer w indicates the minimum number of positions that are to be used for the display of the value and the integer p indicates the number of digits to be displayed after the decimal point.
- It can also display a real number in exponential notation by using the specification.

% wpe

Format	Output
Printf ("% 7.4f ", y)	$\boxed{9}\boxed{8}\boxed{.}\boxed{7}\boxed{6}\boxed{5}\boxed{4}$
Printf ("% 7.2f ", y)	$\boxed{}\boxed{}\boxed{9}\boxed{8}\boxed{.}\boxed{7}\boxed{6}$
printf ("% -7.2f ", y)	$\boxed{9}\boxed{8}\boxed{.}\boxed{7}\boxed{6}\boxed{}\boxed{}$
printf ("% f ", y)	$\boxed{9}\boxed{8}\boxed{.}\boxed{7}\boxed{6}\boxed{5}\boxed{4}$
printf ("%f 10.2e", y)	$\boxed{}\boxed{}\boxed{9}\boxed{.}\boxed{8}\boxed{8}\boxed{e}\boxed{+}\boxed{0}\boxed{1}$

Printing A Single Character:

- A single character cab ne displayed in a desired position using the format %wc. The character will be displayed right-justified in the field of w columns. We can make the display left-justified by placing a minus sign before the integer w. The default value of 2 is 1.

Printing of strings :

- The format specification for outputting string is similar to that of real numbers.

  %wps

  Where, w specifies that field width for display and p instructs that only the first p characters of the string are to be displayed. The display is right justified.

Specification   Output

%s

N	E	W	D	E	L	H	I		1	1	0	0	0	1				

% 20s

			N	E	W	D	E	L	H	I		1	1	0	0	0	1	

% 20.10s

									N	E	W	D	E	L	H	I

% -20.10s

N	E	W	D	E	L	H	I											

Mixed Data Output:

- It is permitted to mix data type in one printf statement for example the statement of the type.

Printf (“%d %f %s %c”, a, b, c, d);

Decision making and Branching :

- The statement which is used to change (or) alter the flow of execution is called by the name control statement.
- There are two types of control statement.
  1) Selection (or) Branching and decision making.
  2) Looping (or) Iteration.

Selection (or) Branching :

- This is further classified into two types.
  - If
  - Switch

If :

- The different form of if statement is as follows

- Simple if statement
- If .....else statement
- Nested if .....elst statement
- Eles if .... ladder.

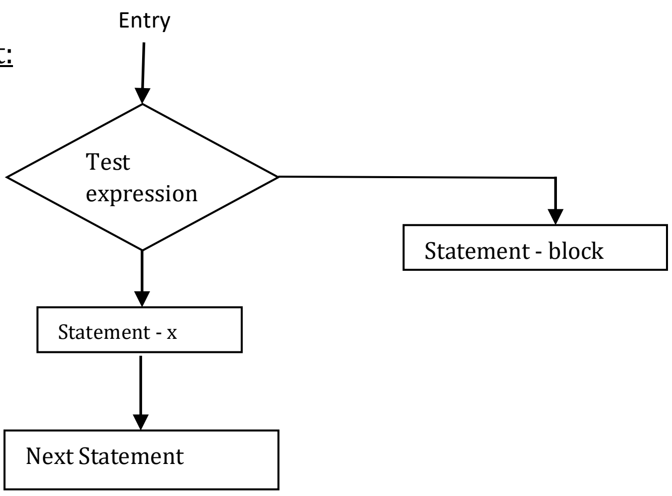
Simple if Statement :

- The General form of a if statement

```
If (test expression)
{
      Statement-block ;
}
      Statement – x;
```

- The statement – block may be a single statement of 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 statement - x.

Flow chart:



Example:

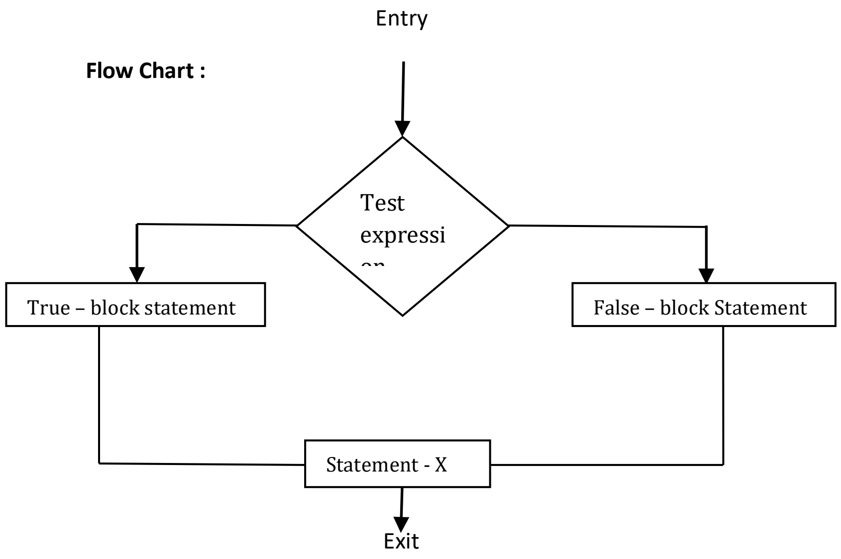
```
# include <stdio.h>
# include <conio.h>
Void main()
{
Int a,b;
Clrscr();
Printf (“Enter the No. ”);
Scanf (“%d %d”, &a, &b);
If (a>b)
{
Printf (“a is greater”);
}
Getch();
}
```

The If ......Else Statement :

- The if....else statement is an extension of the simple if statement.
- The General form is

```
If (test expression)
{
        True – block statement;
}
Else
        {
        Flase – block statement;
        }
Statement – x;
```

Flow Chart :



- The Test expression is true, then the true – block statement, immediately following the if statement are executer; otherwise the false – block statement are executed in either case, either true or false – block will be executed not both in both the cases, the control is transferred subsequently to the statement –x.
- <u>Example :</u>

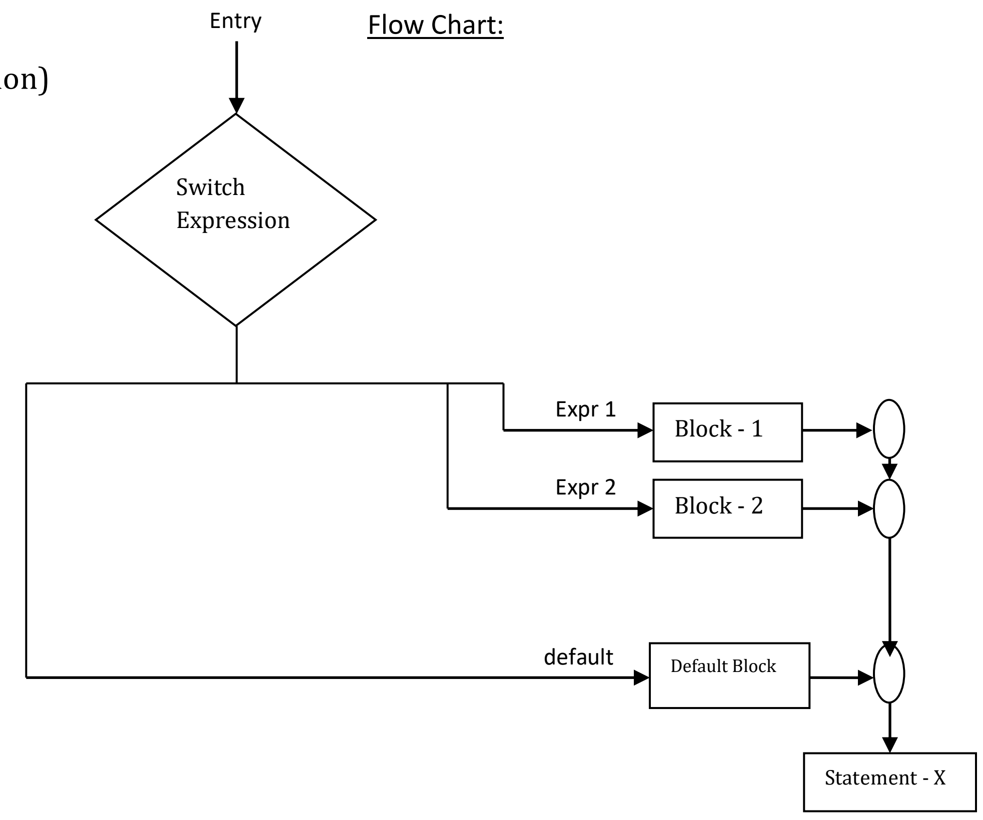
```
        # include <stdio.h>
        #include <conio.h>
Void main()
{
Int a,b;
Clrscr();
Printf("Enter the Number");
Scanf("%d D", &a, &b);
If (a>b)
{
        Printf(" a is greater");
}
Else
{
        Printf("b is greater");
}
Printf("Process is Completed");
Getch();
}
```

The Switch Statement :

- C has a built in multi way decision statement known a 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.
- <u>General Syntax:</u>

<u>Flow Chart:</u>

```
    Switch (expression)
{
case value-1;
block -1;
break;
case value -2;
block -2;
break 2;
......
......
default ;
default-block
break;
}
statement –x;
```



- Expression is an integer expression or characters value-1, value-2 are constants or constant expressions are known as <u>case labels</u>. Each of these values should be a unique within a switch.
- Statement block-1, block-2..... are statement lists and may contain zero or more statements. There is no need to put bracer around these blocks. Note that case labels end with a count.
- When the switch is executed the value of the expression is successfully compared against the value, value-1, value-2,,....... if a case is found whose value matches with

the value of the expressoion then the block of statements that follows, the case are executed.

- The break statement at the end of each block signals the end of a particular case and causes an exit from the switch sttatement transferring the control to the statement-x following the switch.

<u>Example :</u>

```
# include <stdio.h>
#include<conio.h>
Voidmain()
{
 Int res a, b ,c;
 Clrscr ();
 Printf (“Enter the a, b, value \n”);
 Scanf(“ %d %d”, &a, &b);
 Switch();
 {
        Case 1:
          C = a + b;
               Break;
        Csse 2:
               C = a – b;
               Break;
        Case 3:
               C = a * b;
               Break;
        Case 4:
               C = a / b;
               Break;
        Default:
               Printf (“Exit”);
        }
Printf (“The Result is %d”,c);
getch ();
}
```

Rules for Switch Statement:

- The switch expression must be an integral type.
- Case labels must be constants.
- Case labels must be unique. No 2 labels can have the same value. It must end with colon.

- The break statement transfers the control out of the switch statement.

GoTo Statement:

- The go to statements is used to branch unconditionally from one point to another in the program. The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be follows by a colon.

<u>The G.F.:</u>

<u>Forward jump</u>

```
        Go to label;
        ------- ---
        --- - --- --
        Label;
Statement;
```

<u>Backward jump</u>

```
    Label 1;
Statement ;
---- - - -
---- - - -
Goto label;
```

- The label : can be anywhere in the program either before or after the fotolabel; statement.
- <u>Ex :</u>

```
#include <stdio.h>
# include<conio.h>
Void main()
{
 Int i,x;
 Printf (“ goto example program \n”);
 Scanf (“%d”, & x);
 For (i = 0; i < 100; i = i + 2)
```

```
  {
    If ( i==50)
            Goto x;
   Printf ("i  = %d \n",');
 }
Printf("End");
getch();
}
```

Decision Making (or) Looping:

- The loop in a program consists of two parts one is body of the loop another one is control statement
- Any looping statement includes the following steps.
  1) Initialization of a condition variable.
  2) Test the control statement
  3) Executing the body of the loop depending on the condition.
  4) Updating the condition variable.
- There are two types of looping.
  - i) Entry – controlled loop
  - ii) Exit – controlled loop.

Entry controlled loop:

- In the entry controlled loop the condition are tested before the start of the loop execution.

Exit controlled loop:

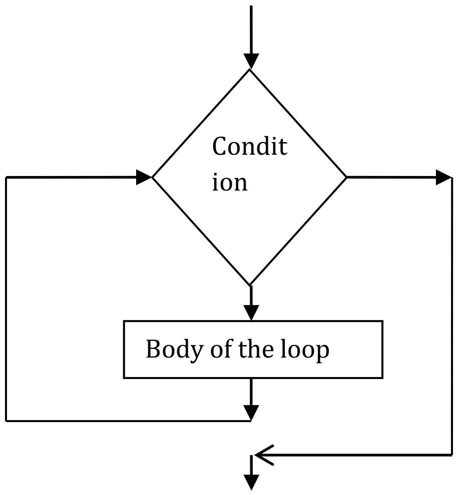
- In the exit – controlled loop the test is performed of the end of the body of the loop. Therefore body is executed unconditionally for the first time.
- The 'C' language provides for three constructs for the performing looping operations. They are,
  1. While Statement
  2. Do _while Statement
  3. For loop Statement.

The While Statement:

G.F:

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

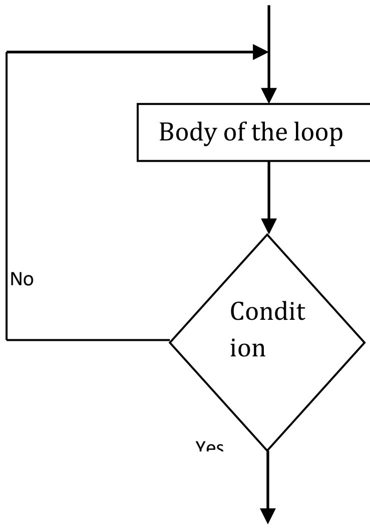
- The while is an entry-controlled loop statement. The best condition is evaluated and if the condition is true then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.
- This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop.



Example:

```
# include <stdio.h>
#include ?<conio.h>
void main()
{
        int i, sum;
        Clrscr ();
        i=1;
        Sum = 0;
        While (i<=10)
        {
                Sum = sum+1;
                i++
        }
        Printf("The sum of the numbers is =%d",sum);
        getch();
}
```

Do - While Statement:



Syntax:

```
Do
        {
        Body of the loop;
        } While (test-condition);
```

- The program proceeds to evaluate the body of the loop first. At the end of the loop the test-condition in the while statement is evaluated. If the condition is true the program continue to evaluate the body of the loop once again. The process continues as long as the condition is true when the condition become false the loop will be terminated.

Ex :

```
# include <stdio.h>
# include <conio.h>
void main()
{
         int i, sum;
        Clrscr();
        i = 1;
        Sum = 0;
        Do
        {
                Sum = sum + 1;
                i++;
        }
        While ( i < = 10);
        Printf ("The sum of the no is =%d", sum);
        getch ();
}
```

For Loop:

Syntax:

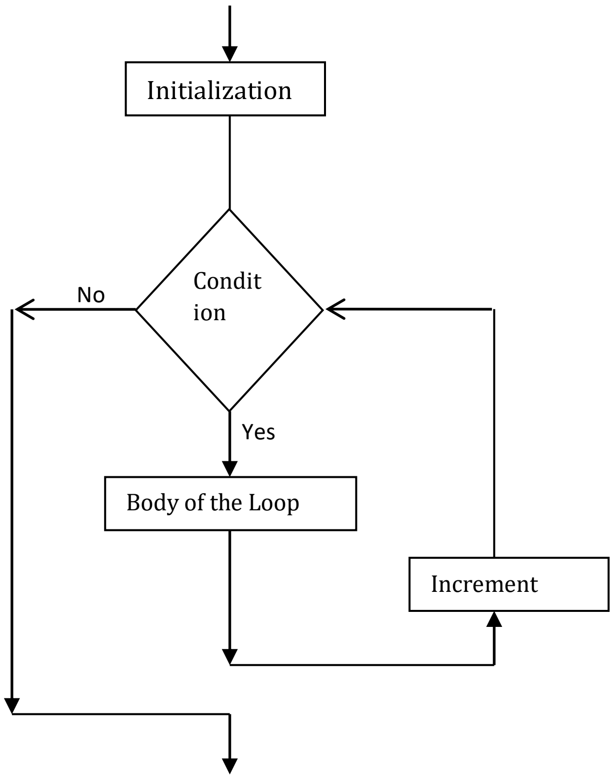
```
For (initialization; test-condition; increment)

 -- - - - -- ----

--------------

body of the loop
```

Flow chart :



Example:

```
# include <stdio.h>
# include <conio.h>
void main()
{
 int i, sum;
Clrscr();
i = 1;
Sum = 0;
for ( i = 1; i < = 10; i + +)
{
        Sum = sum + i;
}
Printf ("The sum of the no is = % d", sum);
getch ();
}
```

Difference between while and do-while statement:

While	Do-While
This is the top tested loop [Entry Controlled]	This is the bottom tested loop [Exit Controlled]
This condition is first tested. If the condition is true then the block is executed until the condition is false	If executes the body once after it check the condition. If it is true the body executed until condition becomes false.
Loop is not executed. If the condition is false	Loop is executed at least once even though the condition is false.
It has no termination	It has termination

Break Statement :

➢ Break statement is used to terminate the loop

Syntax:

break:

Eg :

```
# include <stdio.h>
# include <conio.h>
void main()
{
 int i;
for (i = 1; i < = 10; i + +)
{
        for (i == 6)
        break;
        printf("%d" , i);
}
getch ();
}
```

The Continue Statement:

Syntax :

```
        continue;
```

Eg :

```
        # include <stdio.h>
        # include <conio.h>
        void main()
        {
         int i, n,sum = 0;
        Clrscr();
        for ( i = 1; i < = 5; i + +)
        {
                Printf (“The sum of the no is = % d”, sum);
                Scanf  (“%d”, &n);
                If (n < 0)
                        continue;
                Else
                        Sum = sum + n;
        }
        Printf (“The sum of the no is = % d”, sum);
        getch ();
        }
```

Break	Continue
Break statement takes the control to the outside of the loop	Continue statement takes the control to the beginning of the loop
It is also used in switch statement	This can be used only in loop statement
Always associated with if condition in loops	This is also associated with if condition in loops

Arrays:

Definition :

- An array is a group of related data items that share a common name.

E.g:

A single variable can be used to store n values of the same category.

int salary[10];

- Salary is a integer variable, that can store 10 values in its location starting from salary[0] to salary[10].

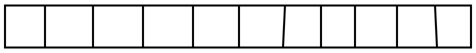
Declaration Syntax :

Datatype varname [size];

Datatype	-	datatype of the array variable
Varname	-	variable name
Size	-	also called as subscript (or) index.

- Individual values in the array are called “elements”.
- Array’s can be of any variable type.
- Array elements are stored in continuous memory locations

  A[0] [1] [2] [3] [4]..................



- Array index always starts from zero(0).
- Array elements can be accessed using the position of the element in the array.

  Arr[i] refers to the ith element in the array arr.
- Index should always be an integer or an expression that gives integers.
- Index cannot be negative.
- Index must be given within square brackets after the array name.

One – Dimensional Array:

- A list of items stored in single variable using only one index (or subscript) is called single-subscripted variable or one-dimensional array.

Initialize an Array:

Syntax:

datatype var nm[size] = {val 1, val 2,..... val n};

val 1, val 2,....val n - initial values for the corresponding array elements.

E.g:

1. static int a[5] = {3,7,9,10,11};
2. char col[3] = {‘R’ , ‘E’ , ‘D’};
3. int dig[3] = {7,9,10,13};
4. int y[5] = {7,8};
5. int z[ ] = {7, 8, 10, 11};

Eg 1:	Eg 2:	Eg 3:	Eg 4:	Eg 5:
a[0] =3	Col[0] = ‘R’	dig[0] = 7	y[0] = 7	z[0] = 7
a[1] =7	Col[1] = ‘E’	dig[1] = 9	y[1] = 8	z[1] = 9
a[2] =9	Col[2] = ‘D’	dig[2] = 10	y[2] = 0	z[2] = 10
a[3] =10			y[3] = 0	z[3] = 11
a[4] =11			y[4] = 0	

- In the above example,
  'a' is a static array.
  All the others are external arrays.
- All individual elements that are not assigned explicit initial values are set to '0'.
- If the size of an array is undefined, the compiler will count the number of initializes and substitute that number for the size of the array.
- In a character array, each characters are assigned within a set of single quotes (' ').
- To assign string to a variable, unsized char array is used.
  str[0] = 'B', str [1] = 'H', str [2] = 'A', str [3] = 'R', str [4] = 'A', str [5] = 'T', str [6] = 'H', str [7] = '\o'.
- The array size will be assigned automatically.
- This will include a provision for the null character '\0', which is automatically added at the end of every string.
- Unsized array in other data types are also allowed.

Accessing Array Element:

- Single operations involving entire arrays are not permitted in C.
- If a and b are similar arrays, assignment operations, comparison operations etc must be carried out on an element basis.
- Usually a loop is used to process one-dimensional array.
- The number of passes through the loop will be equal to the number of array elements to be processed.

E.g: Sum of n numbers:

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int i, sum , a[15], n;
        float avg;
        clrscr();
        sum =0;
        printf("Enter n:");
        scanf(" % d" , & n);
        for (i = 0 ; i < n ; i++)
        {
                scanf("%d" , & a[i]);
                sum = sum + a[i];
        }
        printf("sum = %d" , sum);
        getch();
}
```

Passing Arrays to Function:

- ➢ Arrays can be passed to functions.
- ➢ To pass an array to a function the array name must appear by itself, without brackets or subscript as an actual, assignment within the function call.

  E.g:
  ```
  int a[12];
  y = max(a);
  ```
- ➢ The corresponding formal argument is written in the same manner, though it must be declared as an array within the formal argument declaration.
  ```
  int max(int a1[ ])
  {
  ---------
  ----------
  }
  ```
- ➢ When declaring a one-dimensional array as formal argument, the array name is written with a pair of square brackets, the size of the array is not specified within the formal argument declaration.

E.g:
```
        int max(int [ ]);
```
- ➢ When the function is called, the values of all elements of the array are passed to the corresponding elements of the array in the called function.

E.g:
```
//To fine MEAN of n values:
#include <stdio.h>
#include<conio.h>
{
        float val[10]; int n; float m;
        float mean (float [ ]; int);
        printf("\n Enter no of values to enter:");
        scanf("%d", &n);
        printf("Enter the Values:");
        for ( i = 0 ; i < n ; i++)
        scanf("%f", &val[i]);
        m = mean(val, n);
        printf("\n The MEAN of the number is %d", m);
        getch();
```

```
}
float mean(float arr[ ], int n)
{
        int i;
        float sum = 0;
        for (i = 0; i < n ; i++)
        sum += arr[i];
        return (sum / (float) n);
}
```

Multi - Dimensional Array:

- Applications involving matrix requires two-dimensional arrays to store and process information.
- Similarly a 'N' dimensional array can also be created.
- For every dimension, one more square bracket is included.
- The limit for dimensions is determined by the compiler.

Declaration:

Type arr_nm[s1] [s2] [s3]....[sn];

Type	- data type of the array.
Arr_nm	- array variable name,
[s1]...[sn]	– maximum size of each array location.

A 'm X n' 2-dimensional array can be thought of as a table of values having 'm' rows and 'n' columns.

E.g:

```
int a1[3][3];
A1[3][3] = {{1,1,1},{3,4,5),{6,7,8}};
```

- To initialize the two dimensional array, inside a pair of curly braces, data's are inserted for every individual rows separately.
- To access the elements of a two dimensional array, nested for loop is the best solution.

```
for (i = 0; i<3;i++) {
for(j=0;j<3;j++) {
printf("%d" , a1[i][j]);
}
printf("\n");
}
```

In char datatype, two dimension arrays are used to store more than one string values.

```
char names[10] [20];
names [10] [20] ={"INDIA' , "BHARATH", "GANDHIJI" , "Nethaji"};
here
```

names[0] => INDIA
names[1] => BHARATH.. and so on.