Unit I

Programming Language

➢ What is programming language?

1. The computer doses not understand English so in order to interact with computer or to make a communication with the computer the programmer must know the language which the computer understands.
2. The information and instruction which are to be carried out by the computer are written in the form of a program.
3. Programming language is a medium to communicate with a computer.

➢ Types of Programming Languages

There are three types of programming languages.

1.1 Machine language

1.2 Assembly Language

1.3 High Level Language

1.1 Machine Language

1. Machine Language is simplest and first type of programming language.
2. Machine language was machine dependent.
3. A program written in machine language cannot be run on another type of computer without significant alterations.
4. Machine language is also called as the binary language; i-e, the language of 0 and 1.
5. Very few computer programs are actually written in machine language.
6. It is very difficult and very slow to write program using machine language.
7. Error removing and modification in program is very difficult.
8. There is no need of translator to execute machine code.
9. E.g. Binary language.

1.2 Assembly Language

1. Assembly languages are also called as low-level language.
2. It is easy to write a program in assembly language than machine language.
3. Assembly language uses English words to write the program which called as Mnemonics.
4. To execute assembly language program, assembler is used to translate assembly language code to machine code.
5. It is difficult to write large and complex program using assembly language.

1.3 High Level Language

2. High level language is easier to write program than assembly language.
3. High level language use number of keywords to write a program.
4. Library Function are provided by High Level Languages, makes easy to write program in less number of line of code.
5. High Level Languages are used to develop large and complex programs.
6. To execute High Level Language program, compiler or interpreter are used to translate High Level Language code to machine code.
7. E.g. FORTRON, COBOL, C, C++, JAVA etc.

➢ **What is translator?**

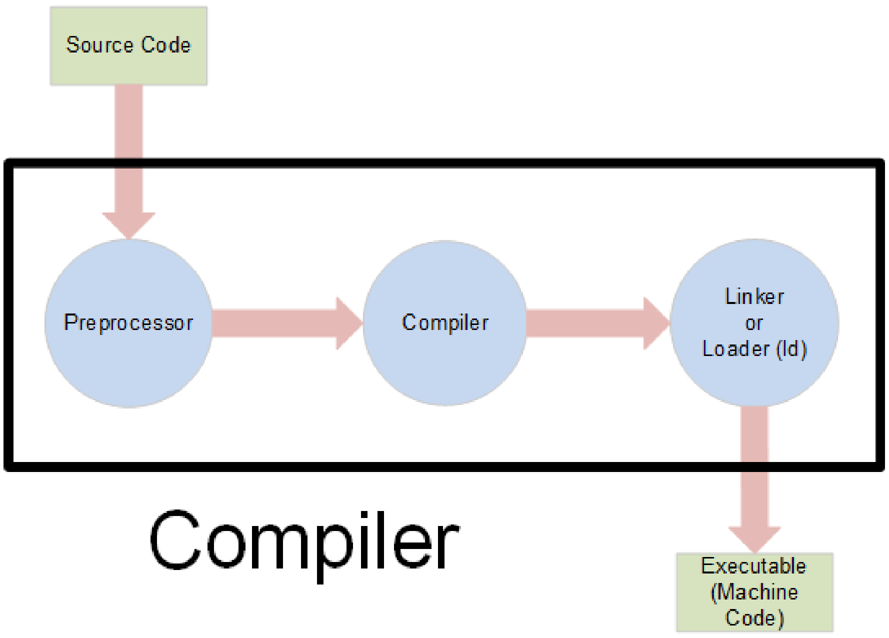
1. Computer programs are writer using English alphabets, special characters and digits.
2. These programs are not understood by computer (machine).
3. Computer understood only binary language i.e. 0's and 1's only.
4. Therefor to execute program on computer, translators are used to convert program code to machine code.
5. Example of translators are 1. Assembler, 2. Compiler, 3. Interpreter etc.

1.4 Compiler and Interpreter

A. Compiler

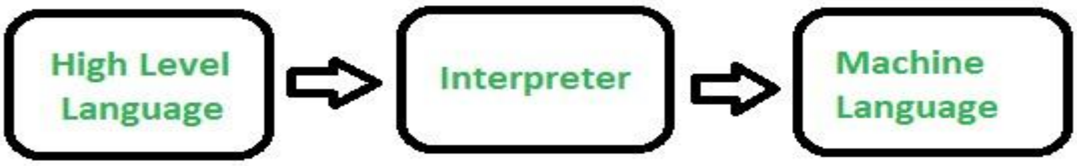
1. Compiler used to translate high level language code to machine code.
2. Compiler scans complete program as once and translate it as a whole into machine code.

3. Compiler has slow speed because it scans complete program as once and translate it as a whole into machine code.
4. Compiler generates all errors at a time, hence debugging is difficult than interpreter.
5. Compiler generate intermediate code which required linking hence compiler required more memory.
6. Compiler checks all kind of limits, ranges, errors etc.
7. Program language like C, C++ uses compiler.



B. Interpreter

1. Interpreter used to translate high level language code to machine code.
2. Interpreter reads line by line program and translate it into machine code.
3. Therefor Interpreter is faster than Compiler.
4. Interpreter generates single error at a time, hence debugging is easy than compiler.
5. Interpreter will not generate intermediate code hence Interpreter required less memory.
6. Program language like Python, Ruby uses Interpreter.



Difference between Compiler and Interpreter

Compiler	Interpreter
A compiler translates the entire source code in a single run.	An interpreter translates the entire source code line by line.
It consumes less time to translate source code to machine code	It consumes more time to translate source code to machine code
It consumes more time to execute machine code.	It consumes less time to execute machine code.
It is more efficient.	It is less efficient.
CPU utilization is more.	CPU utilization is less.
Both syntactic and semantic errors can be checked.	Only syntactic errors are checked.
The compiler is larger.	Interpreters are often smaller than compilers.
It is not flexible.	It is flexible.
The localization of errors is difficult.	The localization of error is easier than the compiler.
The compiler is used by the language such as C, C++.	An interpreter is used by languages such as Java.

Unit II

Introduction to Programming in C

2.1 History of C Programming Language

1. C is one of the high level language.
2. C programming language developed in 1972 by Dennis Ritchie at Bell Laboratories of AT & T (American Telephone & Telegram) in USA.
3. C was originally developed for UNIX operating system.
4. It was developed to overcome problem of previous language such as B, BPCL, etc.
5. UNIX operating system development was started in 1969 and its code was rewritten in c in the year 1972.

2.2 Application Areas

C programming is well known programming language. The applications of C language are as given below:

1. C Language used to develop application software.
2. I used in writing embedded software.
3. Firmware for various electronics, industrial and communication products which use micro controller.
4. It is also used to develop verification software, test code, simulation etc.
5. C also used for creating compiler of different languages.
6. C used to implement different operating system operations. UNIX kernel is completely developed using C.

2.3 Algorithms

1. An algorithm is set of well-defined instructions to solve problem.
2. It takes set of inputs and produce desired output.
3. It is not a complete program or code.
4. The instruction in algorithm can be implemented in any language with the same output.

➢ **Following are the characteristics of algorithm**

1. Input: Algorithm has some input values.
2. Output: Algorithm produce one or more output.
3. Un-ambiguity: Instruction in algorithm should be simple and un-ambiguous.
4. Finiteness: Algorithm should contain limited number of instructions and should be countable.
5. Effectiveness: Each instruction in algorithm affect overall process.
6. Language independent: Instruction in an algorithm can be implemented in any language with same output.

Write an algorithm to add two numbers entered by user

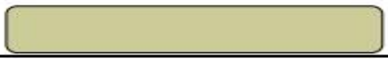
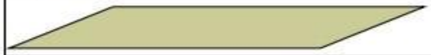
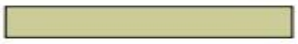
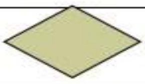
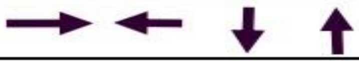
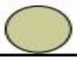
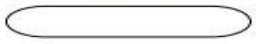
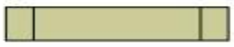
```
Step 1: Start
Step 2: Declare variable num1, num2 and sum.
Step 3: Read value num1 and num2
Step 4: Add num 1 and num2 and assign the result to sum
            Sum ← num1 + num2
Step 5:  Display sum
Step 6: Stop
```

Write an algorithm to find largest number among three numbers.

```
Step 1: Start
Step 2: Declare variables a, b and c
Step 3: Read value of a, b and c
Step 4:     if a > b and a > c
                    Display a is largest number
    else if b > c
                    Display b is largest number
    else
                    Display c is largest number
Step 5: Stop
```

2.4 Flowchart

1. Flowchart is a sequential, logical and stepwise diagrammatic representation of a program.
2. Flowchart uses simple geometric shapes and arrows to show the flow of program.
3. Flowchart can have only one start and stop symbol.
4. General flow of process is from top to bottom or from left to right.
5. Arrows should not cross each other.

Symbol	Meaning
	Start, Stop (begin, end)
	Input, Output
	Processing program instructions
	Decision
	Direction of flow
	Connector
	Loop
	Function symbol

2.5 Structure of C Programming

Structure of C programming language has six sections.

I. Documentation Section

1. In this section, information of program is written such as title of program, purpose of program, author name, date etc.
2. Documentation section is never execute, it also called as comment section.
3. C provides two types of comments i.e. single line comment and multiline comment.
4. Single line comment starts with // (double slash symbol)
5. Multiline comment starts with /* and ends with */.

II. Header (Link) Section

1. In this section, we use header files such as stdio.h, conio.h, string.h etc.

2. Header files are used to link function used in c program.
3. These file are used using # include statement.
4. Example: #include <conio.h>

III. Definition Section

1. In definition section we define constant values using # define
2. Example: #define PI 3.142

IV. Global Declaration section

1. The variable, which used in more than one function are called as global variable.
2. Global variable is declared in global declaration section i.e. outside of all functions.

V. Main Function Section

1. Every C has only one main function which starts with { and ends with }.
2. Main function has two sub sections.

A. Declaration part

i. In this part, variables are declared before used in program. ii.
Example: int a,b,c;

B. Execution part

i. In this section, execution statements and function call statements are used. ii.
Example: . if(), printf(), scanf(). Clrscr(), etc.

VI. Sub Program Section

1. In this section, user can define function which is called as user defined function.

➢ **The Character set**

1. Every language has it's own character set ex;- In English language there are 26 alphabets are used to create word, sentence, paragraph.
2. Similarly 'C' language has it's own character sets i.e. ASCII character set.
3. In C language character set, 256 characters are available.
4. Using these characters we write the C program.
5. C language uses four types of characters.
   i. Alphabets: Upper case 26 (A to Z) and lowercase 26 (a to z).

ii. Digits: 0 to 9 (10) iii. Special characters: , . / ' " ; : { } [ ] \ | ` ~ ! @ # $ % ^ & * ( ) _ - = + etc. iv. White spaces: These are used in string constant. E.g. /n /t /b // /" /r etc.

2.6 C Tokens

1. Token is smallest individual element of program.
2. For example, we cannot create sentence without word; similarly, we cannot create statement without token.
3. Therefore, we can say that tokens are basic component or building blocks to create program.

Classification of tokens in C

Tokens in C language can be divided into seven categories

1. Keywords	2. Identifiers
3. Strings	4. Operators
5. Constant	6. Special Characters

2.6.1 Keywords

1. Keywords are the words whose meaning has already been explained to the C compiler.
2. The keywords cannot be used as variable names.
3. There are 32 keywords used in C all keywords must be written in lower case.
4. List of keywords:

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

2.6.2 Identifiers

1. Identifiers are the names given to variables, functions, arrays constants, structures, unions and labels of statements.
2. The rules for naming identifiers are as follows −
3. Identifier names are unique.
4. Cannot use a keyword as identifiers.
5. Identifier has to begin with a letter or underscore (_).
6. It should not contain white space.
7. Special characters are not allowed.
8. Identifiers can consist of only letters, digits, or underscore.
9. Only 31 characters are significant.
10. They are case sensitive.

The following names are valid identifiers

X Names	y12 area	sum_1 text_rate	_tempera TABLE

2.6.3 String

Any group of characters defined between double quotation marks is a string constant. Eg. “Nandigram Institute of Information Technology”.

2.6.4 Constants

1. Constants are fixed values that never change during execution of program.
2. The types of constant can be integer constant, floating point constant, character constant, string constant, Boolean constant etc.

a) Integer constant

1. Any integer value using 0 to digits are integer constant.
2. ‘,’, ‘.’ Space ‘$’ or’ R’ sign are not allowed in integer constant e.g. 12,480 and $380 are illegal constants

b) Floating point constant

1. A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part.
2. You can represent floating point literals either in decimal form or exponential form.

e.g. 3.14159 and 314159E-5 are legal constants.

c) Character Constants

1. Character Constants are enclosed in single quotes, e.g., 'x'

d) String constant

1. Any group of characters defined between double quotation marks is a string constant.

Eg. “Nandigram Institute of Information Technology”.

e) #define preprocessor constant

1. We can define all basic type constants using #define preprocessor. Eg.

#define PI 3.142

2.6.5 Special Characters

Some of the special characters that are used in C programming are as follows

1. **Brackets[]** −used for array element
2. **Parentheses()** − are used for function calls and function parameters.
3. **Braces{}** −indicates the start and end of a block of code
4. **Comma (,)** −used to separate more than one variables or parameters in function.
5. **Semicolon(;)** − It is called as a statement terminator
6. **Asterisk (*)** − It is used to create a pointer variable.

2.6.2 Variables

1. Variables are containers for storing data values, like numbers and characters.
2. In C, we can define variables according to the data type such as int, char, float, double, long etc.
3. Syntax:

type variableName = value; or *type variableNaame;*

4. *Example:*

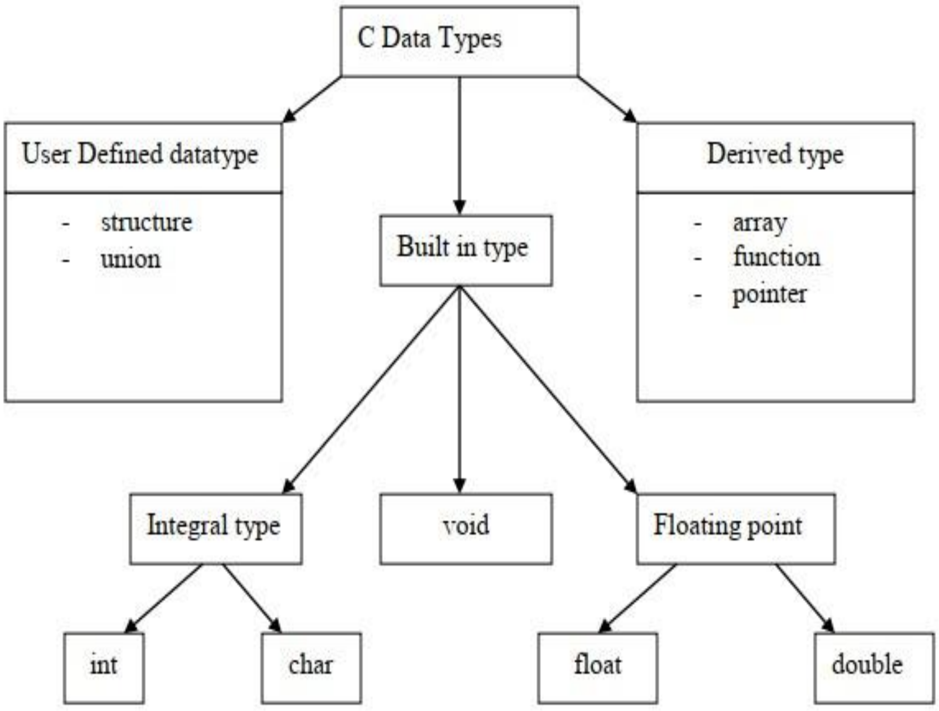
int age=19;

float weight;

2.6.3 Primary Data Types

1. Data types are used to define or declare the type of particular variable.
2. In C language there are three classes of data types.

1) Primary data type.
2) Derived data type.
3) User-defined data types.



1. Primary data type

1. It is also known as basic or built in data type.
2. Primary data type is divided into 3 type.

a. Integer data type:

1. It stores only integer values and cannot store decimal point values.
2. Integer data type contains char, unsigned char, int, unsigned int, long int and unsigned long int data types.
3. The range, size and format specifiers are shown in following table.

Data Type	Range	Size in Byte	Format String
char	-128 to 127 or -2^7 to 2^7-1	1	%c
unsigned char	0 to 255 or 0 to 2^8-1	1	%c
int	-32768 to 32767 or -2^{15} to 2^{15}-1	2	%i or %d
unsigned int	0 to 65535 or 0 to 2^{16}-1	2	%u
long int	-2147483648 to 2147483647 or -2^{31} to 2^{31}-1	4	%ld
unsigned long int	0 to 4294967295 or 0 to 2^{32}-1	4	%lu

b. Floating – point data type

1. A floating-point data type stores the value which has an integer part, a decimal point, a fractional part, and an exponent part.
2. Floating-point data type contains float, double and long double data types.
3. The range, size and format specifiers are shown in following table.

Data Type	**Range**	**Size in Byte**	**Format String**
Float	**3.4e-38 to 3.4e+38**	**4**	**%f**
Double	**1.7e-308 to 1.7e+308**	**8**	**%lf**
Long double	**3.4e-4932 to 1.1e+4932**	**10**	**%lf**

c. Void Type

1. The Void type has no value.
2. This is usually used to specify the type of functions.
3. The type of a function is said to be void when it does not return any value to the calling function.

2. Derived data type

1. Derived data type are derived from fundamental data type.
2. The derived data type are array, pointer and function.

3. User defined data types

1. The data types that are defined by the user are called as User defined data types.
2. User defined data types are structure and union.

2.6.4 Operators

1. An operator is a symbol that tells the compiler to perform mathematical or logical operations.
2. C language providing eight types of operators.

1. Logical Operators

Logical operator works on Boolean types of operands or conditions.

1. &&, || and ! are the logical operators.
2. Logical AND operator (&&) - If both the operands are true, then the condition becomes true. E.g. if((a>b) && (a>c)), if a is greater than both b and c then it returns true
3. Logical OR Operator (||) - If any of the two operands is true, then the condition becomes true.
   E.g. if((a>b) || (a>c)), if a is greater than b or c then it returns true.
4. Logical NOT operator (!): Is unary operator. If the condition is true it returns false and if the condition is false it returns true.
   E.g. if(!(a>b)), if a is greater than b then it returns false else it returns true.

2. Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language.

Assume A=10 and B=20.

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10

*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	It returns remainder after an integer division.	B % A = 0

3. Relational Operators

If we want to compare two values then Relational operators are used.

C supports six Relational operators
Assume A=10 and B=20.

Operator	Example
==	(A == B) is not true.
!=	(A != B) is true.
>	(A > B) is not true.
<	(A < B) is true.
>=	(A >= B) is not true.
<=	(A <= B) is true

4. Assignment Operator

The assignment operators are used to assign values or result of the expression to variables. ex:-

a=10;

5. Increment and Decrement operators

Prefix Increment and Decrement operators

In the prefix operators the values of variables is Increment and Decrement First and then assigned to the expression

++a and - - a

a=3 y=++a

y=4

Postfix Increment and Decrement operators

Postfix operators first assign the values to variables on left side and then increment the operand

a=5; y=a++; y=6

6. Special Operators

Below are some of the special operators that the C programming language offers.

Operators	**Description**
&	This is used to get the address of the variable. Example : &a will give address of a.
*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.

7. Bit-wise Operator

Bitwise operator works on bits and perform bit-by-bit operation.

Assume a=4 (0100) and b=3 (0011)

Operator	**Example**
&	(A & B) = 0, i.e., 0000
|	(A | B) = 7, i.e., 0111
^ XOR	(A ^ B) = 49, i.e., 0111
~ One;s compliments	(~A) = , i.e,. -0111101
<<	A << 2 = 16 i.e., 1 0000
>>	A >> 2 = 1 i.e., 0001

8. Conditional Operators

This is only one ternary operator in C programming language.

Syntax:

Expirations ? expression2 : expression3

Example: x=(a<b)? a:b;

it means that –

```
if (a>b)
        x=a
else
        x=b;
```

➢ Input and Output Functions

Input and Output Functions are used to read the data from the input devices and display the results on the screen.

In C programming, the input/output functions are classified in to two types:

1. Formatted I/O Function and
2. Unformatted I/O Function

2.7 Formatted I/O Function:

1. Formatted Input and Output functions used to read and write data with specific formatting.
2. These I/O functions supports all data types like int, float, char, and many more. 3. scanf() and printf() are commonly used formatted I/O functions

A. printf()

1. This function is used to display result on the screen.
2. It can be used to display any combination of numerical value or char or string value.
3. This function is defined in the **stdio.h** header file.
4. Syntax:

   printf (“format string”, v1, v2, . . . , vn);
5. For eg.

```
printf (“%f”, s);  printf (“\n
sum=%6.2f”, s);
printf (“\n %d factorial is %d”, k, kfact);
```

B. scanf()

1. This function is used to receive input from keyboard.
2. It can be used to receive any combination of numerical value or char or string value.
3. This function is defined in the **stdio.h** header file.
4. Syntax:

```
scanf (“format string”, &v1, &v2, . . . , &vn);
```

5. For eg.

```
scanf   (“%f”,   &s);      scanf
(“%c”,&ans);      scanf   (“%d
%d”,&k, &kfact);
```

//display the ascii value

```
#include<stdio.h>
#include<conio.h> void
main()
{
        int            y;
clrscr();
        prinf(“Enter     any     integer     value     ::     “);
        scanf(“%d”,&y);
         printf(“%c %d”,y,y);
        getch();
}
```

2.7 Unformatted Function: -

1. These functions are called unformatted I/O functions because it cannot use format specifiers.
2. The unformatted input/output functions only work with the character data type.
3. In case values of other data types are passed to these functions, they are treated as the character data.
4. getch(), getche(), getchar(), putch(), putchar() and clrscr() are commonly used unformatted functions provide by conio.h header file and gets() and puts() are the unformatted functions provided by stdio.h header file.

a. *getch()*

1. getch() function reads a single character from the keyboard.
2. The character entered by user doesn't display on the screen.
3. There is no need to press enter key from keyboard. As soon as any key pressed from keyboard getch() read a single character.
4. getch() is also used for hold the screen.
5. Example:

   ch=getch(); // where ch is a char variable.

b. *getche()*

1. getche() function reads a single character from the keyboard.
2. The character entered by user is displayed on the screen.
3. There is no need to press enter key from keyboard. As soon as any key pressed from keyboard getch() read a single character.
4. Example:

   ch=getche(); // where ch is a char variable.

c. *getchar()*

1. The getchar() function is used to read only a first single character from the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed. 2. Example:

   ch=getchar(); // where ch is a char variable.

d. *gets()*

1. gets() function reads a group of characters or strings from the keyboard.
2. These characters get stored in a character array.
3. This function allows us to write space-separated texts or strings.
4. Syntax: **gets(str);** // where str is a character string variable.
5. Example : **char str[15]; gets(str);**

e. *puts()*

1. puts() function is used to display a group of characters or strings which is already stored in a character array.
2. Syntax: puts(str); // where str is a string (array of characters)
3. Example :

   char str[20]=" Hello World"; puts(str);

f. *clrscr()*

1. clrscr() function is used to clear the monitor screen.
2. It has the following syntax : **clrscr();**

Difference between scanf() and gets()

1. The main difference between these two functions is that scanf() stops reading characters when it encounters a space, but gets() reads space as character too.
2. If you enter name as Shaikh Nisar using scanf() it will only read and store Shaikh and will leave the part after space. But gets() function will read it completely.

Unit III

Controlling Statement

3.1 Conditional / Decision Making Statements

1. The statements in a program are normally executed line by line from start to end of program.
2. Sometimes, a programmer needs a single statement or a group of statements to be execute according condition.
3. This will be done using conditional statements or decision making statements.
4. C Language provides following conditional or decision making statements. i. if statement ii. if …. else statement iii. Nested if statement iv. Ladder of else if statement

3.1.1 if Statement

1. The if statement is a powerful decision making statement used to control the flow of program.
2. if statement first of all check the condition and if condition is true, if block will be executed and if the condition is false, if block will not be executed and program control goes next to if block statement.
3. The general format of if statement is below:

```
if(test expression)
{
        Block of statements
}
```

1. After test expression of if statement, block of statements are written inside open and closed curly brackets {}.
2. Block of statements may contain single statement or group of statements.

3. If block of statements contains single statement, open and closed curly brackets {} are optional.

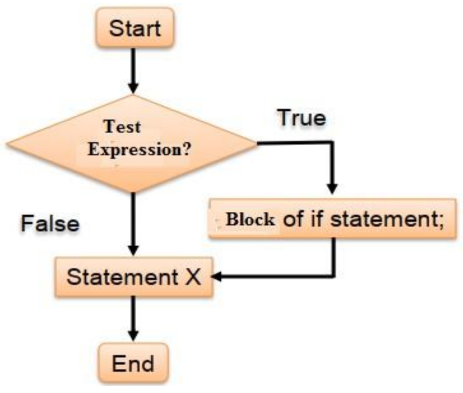


Fig.: Flowchart of simple if statement

4. Example:

```
if ( a>b)
{
        Printf(%d\n”,a);
}

Void main()
{
        int a,b;
        printf(“\nEnter the Number a & b”);
        scanf(“%d%d”,&a,&b);      if(a>b)
        printf(“A is grater number”);
        if(b>a) printf(“B is grater number”);
        getch();
} Note:
```

i. The test expression (condition) should not end with semicolon (;). ii. The test expression without expression arguments is not acceptable. iii. The test expression must be always enclosed within a pair of brackets ().

3.1.2 if….else Statement

1. The if – else statement is an extension of the simple if statement.
2. if statement first of all check the condition and if condition is true, true statements of block will be executed and if the condition is false, false statements of block will be executed and program control goes next to true and false statements of blocks.
3. In if…else statement every time true or false statements of block will be executed. It never happens that both blocks will be executed or will not be executed. At a time only one block will be executed.
4. The general format of if…else statement is below:

```
if (test expression)
{
        true statements block
} else
{
        false statements block
}
```

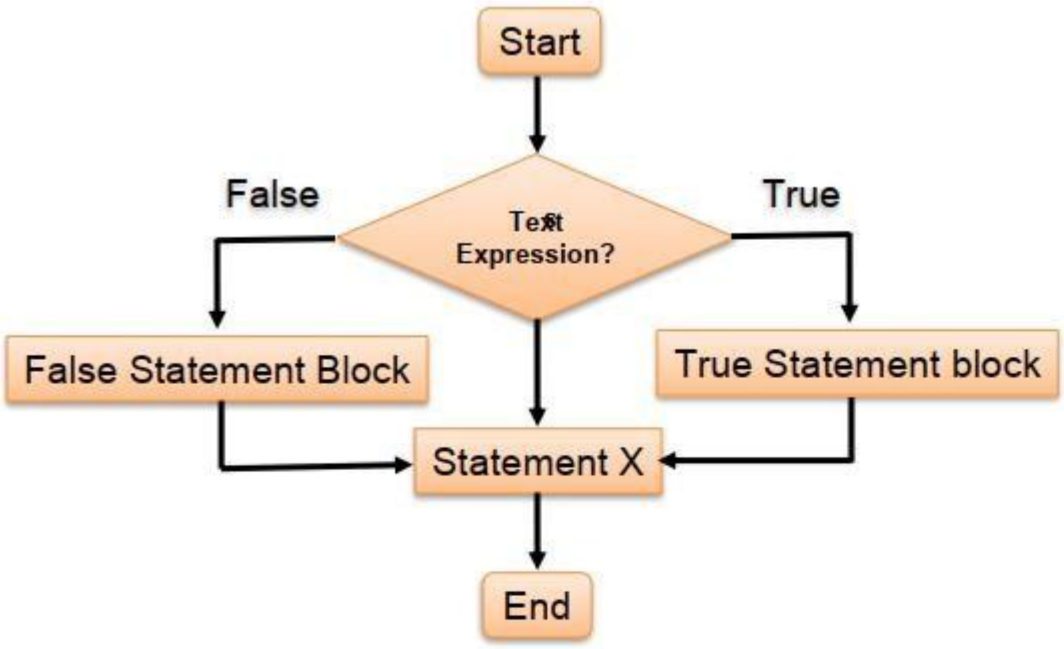


Fig.: Flowchart of if…else statement

//Write a program to find greater number within two numbers

```
void main()
{ int a,b;
        printf("\nEnter the Number a & b");
        scanf("%d%d",&a,&b);     if(a>b)
        printf("A is grater number");
        else printf("B is grater number");
        getch();
}
```

//Write a program to find given number is even or odd

```
void main()
{ int num;
        printf("\n Enter the any number");
        scanf("%d",&num); if(num%2==0)
        { printf("The given number is even number");
        } else
        {
                        printf("The given number is odd number");
        }
getch(); }
```

//Write a program to find given year is leap year or not

```
void main()
```

```
{    int  year;  printf("\nEnter  the
          year");
          scanf("%d",&year);
          if(year%4==0 && year%400==0 && year%100!=0)
          { printf("The given year is leap year");
          } else
          { printf("The given year is not leap year");
          } getch();
}
```

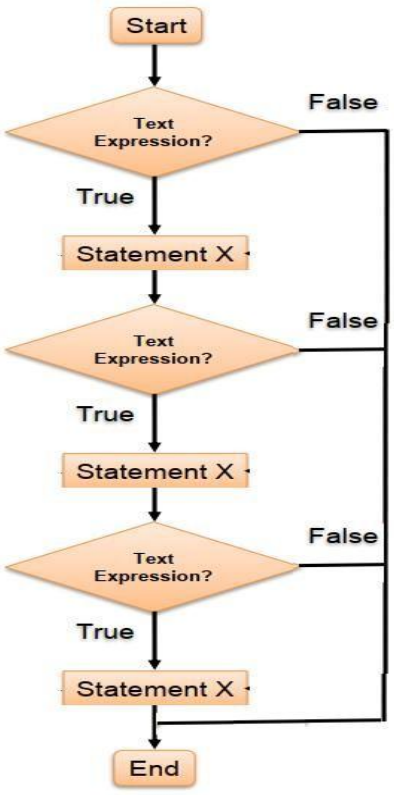
3.1.3 Nested if statement

1. In C programming we can use multiple if statements inside the body of other if statement block, this is called as nested if statement.
2. We can also use the multiple if or if…else statements inside the body of else statement block, this is called nested if … else statement.
3. General form of nested if statement is as below:

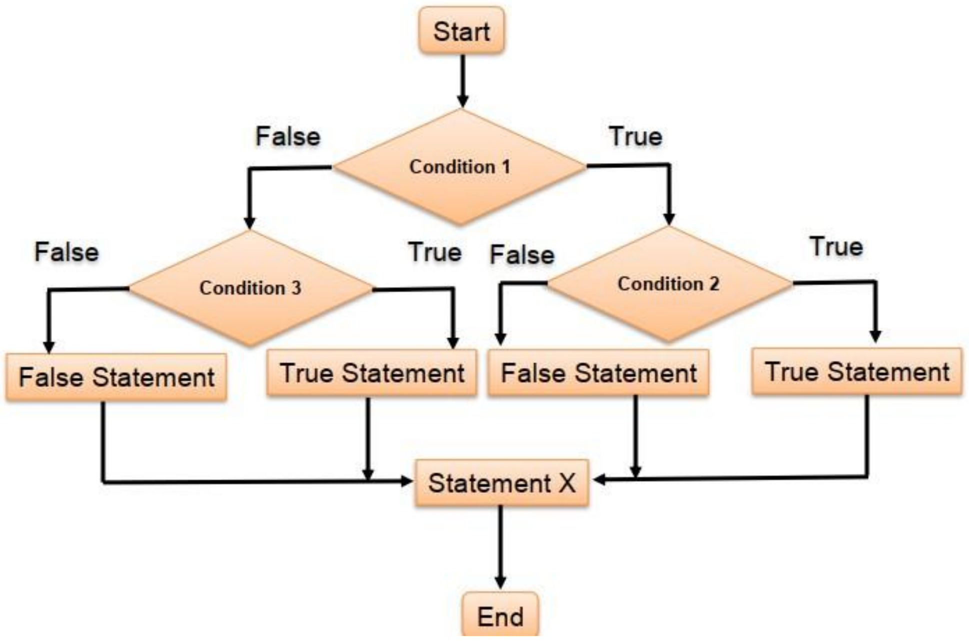
```
if(condition 1)
{
        Block of statements
        if(condition 2)
        {
                Block of statements
if(condition 3)
                {
                        Block of statements
                }
        }
}
```

4. General form of nested if…else statement is as below:

```
if(condtion 1)
{
        Block of statements
        if(condtion 2)
        {
                Block of statements
        }
        else
        {
                Block of statements
        }
} else
{
        Block        of        statements
if(condtion 3)
        {
                Block of statements
        }
        else
        {
                Block of statements
        }
}
```



Flowchart for nested if statements



Flow chart for nested if…else statement

//Write a program to print largest number among three numbers.

```
void main()
{
        int a,b,c;      printf("Enter    any
three numbers");
        scanf("%d %d %d", &a, &b, &c);
        if(a>b)
        { if(a>c)
                { printf("%d is grater ",a);
                } else
                { printf("%d is grater ",c);
                }
        } else
        { if(b>c)
                { printf("%d is grater ",b);
                } else
                { printf("%d is grater ",c);
                }
        }
}
```

//Write a program that display the class obtained by student if percentage of the student is input through the keyboard.

```
void main()
{ float per;
        printf("\nEnter the percentage");
        scanf("%f",&per); if(per<0 &&
        per>100)
        {    printf("Invalid    input");
                exit();
        }           if(per<40)
        printf("Fail");

        else
        { if(per<50) printf("Pass");

                else
                { if(per<60) printf("Second class class");

                        else printf("First class");

                }
        }
}
```

//Write a program find out entered key is upper case or lower case.

```
#include<stdio.h>
#include<conio.h> void
main()

{ char x; clrscr(); printf("\n Enter any char");
        scanf("%c",&x);
        if((x>=65&&x<=90)||(x>=97&&x<=122)
        )
        {

                if(x>=65&&x<=90)
                        printf("it is an upper case alphabet");
                else printf("It is lower case alphabet");

        } else printf("It is not alphabet");

        getch();
}
```

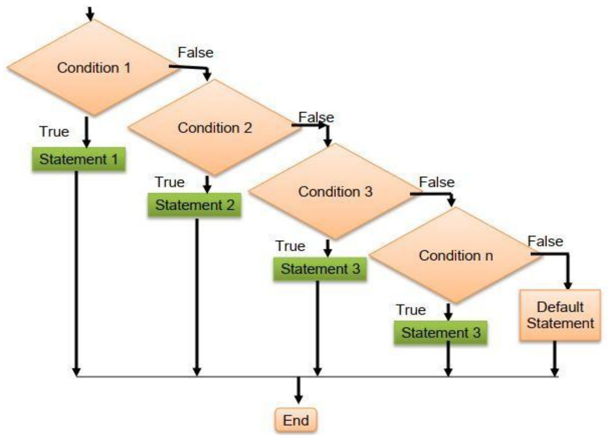
//Write a program find out Positive number and negative number

```
#include<stdio.h>
#include<conio.h> void
main()
{

        int num; printf("\n enter any
        number");
        scanf("%d",&num);
        if(num>0)
        { printf("given number is positive");
        } else
        { if(num<0)
                { printf("given num is negative");
                } else
                { printf("Zero");
                }
        } getch();
}
```

3.1.4 else if Ladder

1. if else if ladder in C programming is used to test a series of conditions sequentially.
2. Furthermore, if a condition is tested only when all previous if conditions in the if-else ladder are false.
3. If any of the conditional expressions evaluate to be true, the appropriate code block will be executed, and the entire if-else ladder will be terminated.
4. General form of nested if…else if ladder statement is as below:

```
if(condition 1)
{
            Block of statements
}  else  if(condition
2)
{
            Block of statements
}  else  if(condition
3)
{
            Block of statements
} else
{
            Default block of statements
}
```



Flow chart for if … else if ladder statement

//Write a program that display the class obtained by student if percentage of the student is input through the keyboard.

```
void main()
{  float  per;  printf("Enter  the
          value");
```

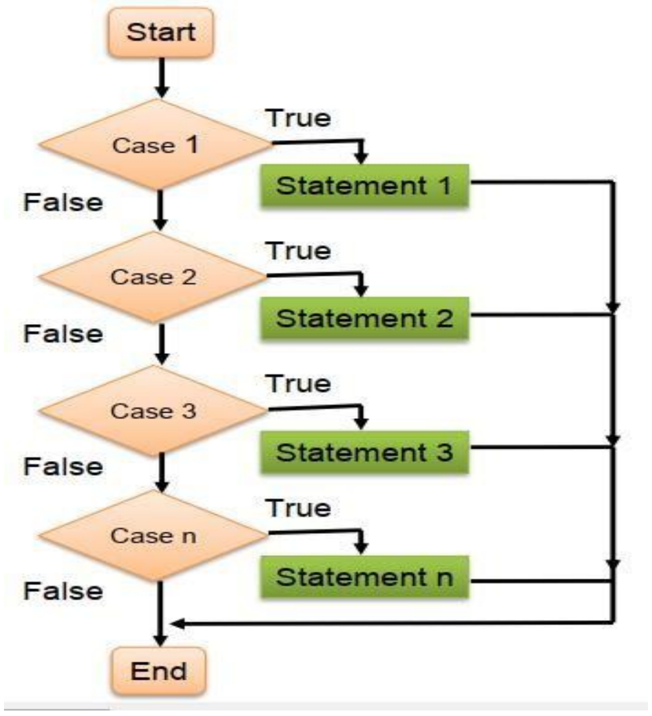
```
        scanf("%f",&per);
        if(per>80)
        { printf("grade is merit");
        }
        else if(per>70)
        { printf("Distinction");
        }
        else if(per>60)
        { printf("Grade A");
        }
        else if(per>50)
        {

                printf("Grade is B");
        }
        else if(per>=35)
        { printf("Grade is C");
        }
        else
        { printf("Failed");
        } getch();
}
```

3.1.5 switch … case Statement

1. Instead of writing nested if..else if statements, we can use the switch statement.
2. The switch statement selects one of many code blocks to be executed.
3. The switch expression is evaluated once.
4. The value of the expression is compared with the values of each case.
5. If there is a match, the associated block of code is executed.
6. The break and default statements are optional.
7. After every case if break statement is not used, after matching case value with expression value all remaining case block will be executed including default block.
8. The breaks statement is out of the switch block and stops the execution.

9. The default statement is specifies some code to run if there is no case match.
10. In a switch there can be either variable or expression
11. If it is a variable it must be either integer or character 12. If it is an expression it must be an arithmetic expression.
13. There can be any number of cases.
14. Every case should have an unique value.
15. These cases can be written in any sequence.
16. In a case there can be any number of statements.
17. It is possible to have the nested switch.
18. General format of switch …. case statement is as below:

```
switch(expression)
{
        case value 1:
                block of statements
                break;
        case value 2:
                block of statements
                break;
case value 3:
                block of statements
                break;
default:
                default block of statements
}
```



Flow chart for switch….case statement.

// Write a program that reads a number between 1 to 7 and display the day name

```
void main()
{

        int day;
        printf("Enter a number between 1 to 7 \n");
        scanf("%d",&day); switch(day)
        {   case   1:   printf("Monday\n");
                break;
                case 2: printf("Tuesday\n"); break;
                case 3: printf("Wednesday\n"); break;
                case 4: printf("Thursday\n"); break;
```

```
                    case 5:printf("Friday\n"); break;
                    case 6:printf("Saturday\n"); break;
                    case 7:printf("Sunday\n"); break;
                    default:printf("Monday\n");
            } getch();
        }
```

//Write a program add, sub, multi & division of two number

```
        void main()
        {   int   a,b,n,p;   float   c;   clrscr();
            xyz:printf("\nenter two number");
            scanf("%d%d",&a,&b);
            printf("\n 1 addition of two number"); printf("\n
            2 subtraction of two number"); printf("\n 3
            multiplication of two number"); printf("\n 4
            division of two number"); printf("\nenter your
            choice");
            scanf("%d",&n); switch(n)
            {
                case 1: c=a+b;
                    printf("\naddition of two number is%f",c); break;
                case 2: c+a-b; printf("\subtraction of two number
                    is%f",c); break;
                case 3: c+a*b; printf("\multiplication of two number
                    is%f",c); break;
                case 4: c+a-b; printf("\division of two number
                    is%f",c); break;
                default:
                    printf("\nentered choice incorrect");
            } printf("\nDo you want to continue
            (Y/N)...."); printf("\nEnter 6 for Yes and 7 for
            No");
            scanf("%d",&p); swatch(p)
            { case 6: goto xyz;
                case 7:exit();
            }

            getch();
        }
```

3.2 Looping Control Structures

In the C programming, sometimes we want to execute some part of the program again and again for some particular number of times. In such situations we use looping statements. C programming provides three types of looping statements

1. for loop
2. while loop
3. do….while loop

3.2.1for loop

1. When we exactly know how many times we want to execute loop, we use for loop instead of while and do…while loop.
2. for loop consist of three statements i.e. initialization statement, test expression / conditional statement and increment / decrement statement.
3. Syntax

```
for(initialization statement; test expression; increment / decrement) {
        Statements inside the body of loop
}
```

Example

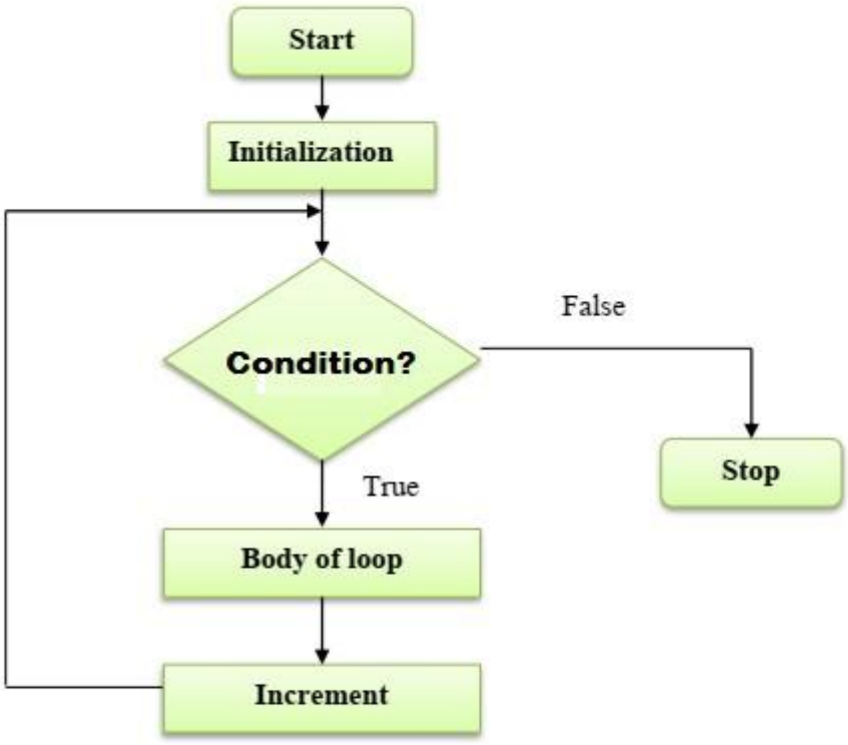
```
        for(int i=0; i<10; i++)
                printf("%d\t",i);
```

Output of above code is as:

```
0       1       2       3       4       5       6       7       8       9
```

➢ *How for loop works*

1. The initialization statement is executed only once. Variable should be initialized if already declared. If variable is not declared before initialization, variable should be declared and initialized.
2. Then test expression is evaluated. If the test expression is false, for loop will be terminated. If the test expression is true, the statements inside the body of loop will be executed. It is called as an iteration of loop.
3. After completion of one iteration, increment / decrement statement will executed and value of variable will be updated.
4. Again the test expression will be evaluated.
5. The loop will execute again and again until the test expression become false.
6.



Flow chart for 'for' loop

//Write a program to print addition of 1 to 10 numbers

```
#include<stdio.h>
#include<conio.h>
Void main()
{ int sum=0; clrscr(); for(int
        i=1;i<=10;i++)
                sum+=i;
        printf("/nAddition of 1 to 10 numbers is %d",sum);
        getch();
}
```

//Write program to print all numbers from 1 to 200 divisible by 3,4 and 5.

```
#include<stdio.h>
#include<conio.h>
Void main()
{ clrscr();
        printf("All numbers from 1 to 200 divisible by 3,4 and 5\n");
        for(int i=1;i<=100;i++)
        {
                if(i%3==0 && i%4==0 && i%5==0)
                        printf("%d/t",i);
        }
        getch();
}
```

//Write program to print table of 1 to 20

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        clrscr();
        for(int i=1;i<=10;i++)
        {
                for(int j=1;j<=20;j++)
                printf("%4d",(i*j));
                printf("\n");
        }
        getch();
}
```

3.2.2 While loop

1. while loop is one of the entry control loop.
2. In this loop the condition is given at the top.
3. This loop check the condition first and if condition is true, body of loop will be executed.
4. After completion of one iteration (loop), again condition will checked and if condition is true, body of loop will be executed.
5. Body of loop will be execute again and again, until condition become false.

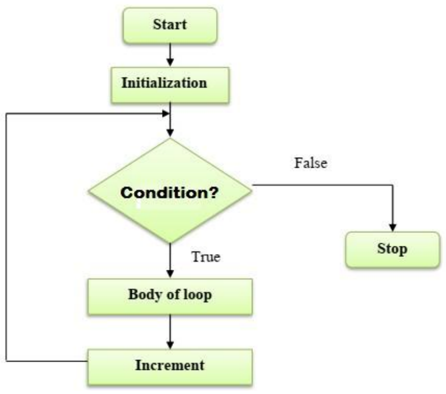
Syntax:

```
while(condition)
{
        Body of loop

}
```

Example:

//Write a program to print all even numbers from 1 to 100.

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

void main()
{
        int         i=2;
        clrscr();
        while(i<=100)
        {
                printf("%3d",i);
                i=i+2;
        }
        getch();
}
```



Flow-chart for while loop

3.2.3 Do….while loop

1. do….while loop is the exit control loop.
2. In this loop the condition is given at the end of loop.
3. This loop executes the body of loop without checking the condition for first iteration.
4. Since the test condition is execute at the bottom of the loop, the body of the loop is always executed at least once.
5. After completion of one iteration (loop), condition will be checked and if condition is true, body of loop will be executed for second time.
6. Body of loop will be executing again and again, until condition become false.
7. At the end of while condition in do…while loop, semicolon is compulsory.

Syntax:

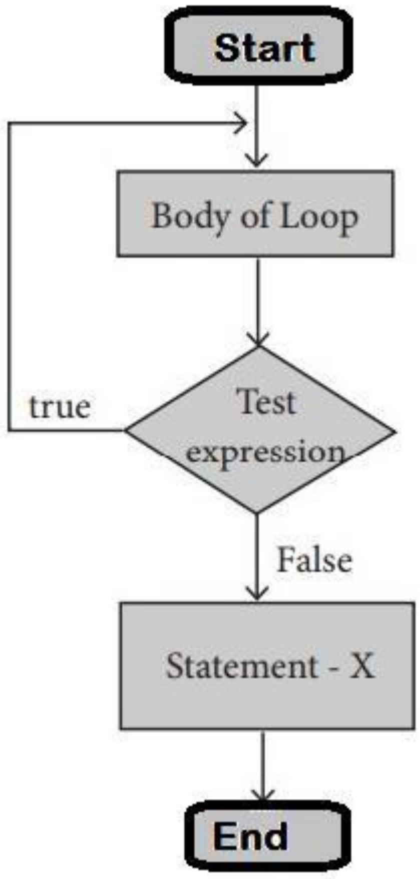
```
        do
        {

        Bod
y of loop

        } while(condition);
```

Example:

//Write a program to print all odd numbers from 1 to 100.

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

```
void main()
{
        int          i=1;
        clrscr();
        do
        {
                printf("%3d",i);
                i=i+2;
        }          while(i<=100)
        getch();
}
```



Flow chart for do….while loop

//Write program for Fibonacci series

```
#include<conio.h>
#include<stdio.h> void
main()
{
        int i=1,n,f,f1,f2;
        clrscr();
        printf("Enter Number of Fibonacci Values Needed : ");
        scanf("%d",&n);
f=0;            f1=1;
f2=1;           do      {
i++;
printf("%d\n",f);
f1=f2;                  f2=f;

f=f1+f2;        }
while(i<=n);
        getch();
}
```

OUTPUT:

```
Enter Number of Fibonacci Values Needed : 10

0
1
1
2
3
5
8
13
21
34
```

//Write program to addition of numbers.

```
#include        <stdio.h>
#include<conio.h> void
main()
{
        int number, sum = 0;
        clrscr();
        do  {

         printf("Enter  a  number:  (To  stop  enter  0)  ");
                scanf("%lf", &number);
                sum += number;
        }
        while(number != 0); printf("Sum
        = %.d",sum);
        getch();
}
```

Difference between while and do…while loop.

while loop	do … while loop
This is entry control loop	This is exit control loop.
Condition is at the top	Condition is at the bottom
No necessity of brackets if there is single statement in body	Brackets are compulsory even if there is a single statement.
There is no semicolon at the end of while.	The semicolon is compulsory at the end dowhile.
It is not compulsory to execute body of loop at least once.	It is compulsory to execute body of loop at least once.

while(n<=10)
{
printf("%d\n",n);
n++;
}
	do {
	printf("%d\n",n);
	n++;
	}while(n<=100);

3.3 A. Break Statement

1. The break statement is used for two different purposes. In switch … case statement, break statement terminate the switch block.
2. It is also used in looping statement. When break is encountered inside any loop, control automatically passes to the first statement after the loop.

➢ **Ex:-** switch()

```
{   case 1: block of
        statements. break;
        case 2: block of
        statements.

        break; default:

        block of statements.

}
```

//Write a program to print 1 to 10 numbers up to number divisible by 2 and 3.

```
void main()
{
        int i;
        for(i=1;i<=10;i++)
        {
                if(i%2==0 && i%3==0)
                {

                        break;
                } printf("%d",i);
        } getch();
}
```

3.3 B. Continue Statement

1. This statement should be used only within the loop.
2. After execution of continue statement inside the loop, remaining statements of loop will not be execute and next iteration will be start to execute.

Example

//Write a program to print all number from 1 to 100 except number divisible by 2 and
3.

```
Void main()
```

```
{
        int i;
        for(i=1;i<=20;i++)
        {
                if(i%2==0 && i%3==0)
                { continue;
                } printf("\n%d",i);
        } getch();
}
```

In above example, if condition is true when i =6,12 and 18 and continue statement will be execute. Therefore printf("\n%d",i) statement will not execute after continue statement execution. The above program will not print the numbers 6, 12 and 18.

3.3 C. goto Statement

1. This statement transfers the control from one statement to other statement in the program, which may not be in the sequence.
2. The general form of the goto statement is

```
goto label; Statement
1

Statement          2
Statement  n  label:
Statement         x1
Statement x2

Statement xn
```

Above example is called as forward jump.

```
label : Statement 1
Statement 2

Statement  n  go  to
label;     Statement
x1

Statement x2

Statement xn
```

This is called as backward jump.

3. goto statements requires a label to identify where to jump.
4. Label names follow the rules of variable name.
5. After label name colon (:) is compulsory.
6. After goto label semicolon (;) is used.

Example

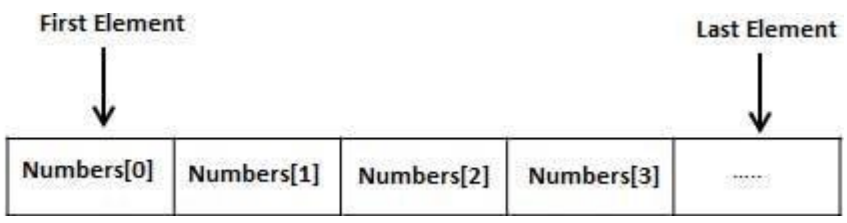
```
#include<stdio.h>

#include<conio.h>
#include<process.h> void
main ()

{

        int a,b;

        clrscr();

        xyz:printf("enter the number");
        scanf("%d",&a); if(a%2==0)

        { printf("given num is even ");

        }

        else

        {

                printf("\nthe given num is odd");

        }    printf("\nif    you    want    to
        continue"); printf("\n Enter 6 to yes
        7    to    no");    scanf("%d",&b);
        if(b==6) goto xyz; if(b==7) exit(0);

}
```

Unit IV

Array and Structure

4.1 Array

1. Arrays used to store multiple values in a single variable, instead of declaring separate variables for each value.
2. An array is a group of similar type of data elements stored in contiguous memory location. These data element addressed by common name.
3. Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.



4. A specific element in an array is accessed by an index.
5. Index of first element of array is 0 and index of last element is array size-1.

4.2 Array declaration, initialization

1. An array is declared by writing the data types, followed by the name, followed by the size in brackets.
2. Syntax: data type array_name[size];

3. Example:
```
int n[12];
```
4. Here declares the n as an array to contain a maximum of 12 integer constants.
5. The c language treats character strings simply as arrays of characters.
6. The size in a character string represents the maximum number of characters that the string can hold.
7. Example:

```
char n[20];
```

8. Here declare the variable n as a character array variable that can hold a maximum of 20 characters. Suppose we read the following string constant into the string variable name.
   “Nandigram Institute”
9. Each character of the string is treated as an element of the array name and is stored in the memory as follow.

n[0] n[1] n[2] n[3] n[4] n[5] n[6] n[7] ….. …..n[20]

‘N’	‘a’	‘n’	‘d’	‘i’	‘g’	‘r’	‘a’	‘m’	‘ ‘	‘I’	‘n’	‘s’	‘t’	‘i’	‘t’	‘u’	‘t’	‘e’	‘/n’

10. When the compiler finds a character string. It terminates it with an additional null character. The element n[20] holds the null character ‘\0’.
11. When declaring character arrays, we must allow one extra element space for the null terminator.

➢ **Types of Arrays**

There are three types of arrays

1. One dimensional array
2. Two dimensional array
3. Multi dimensional array

4.3. One dimensional array

1. One-dimensional arrays, are arrays with only one dimension or a single row.
2. Generally, values in list of same data types are stored as single dimensional arrays.

➢ **Declaration of one dimensional array**

1. An array is declared by writing the data types, followed by the name, followed by the size in brackets.
2. Syntax:
   data type array_name[size];
3. example:
   int n[12];

➢ **Initialization of one dimensional array**

1. We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.
2. Syntax:

   Data type array-name[size]={list of values};

   The values in the list are separated by commas

3. Ex-

4. Here declare the variable number as an array of size 3 and will assign zero to each elements.
5. If the number of values in the list is less than the number of elements will be set to zero automatically.

   float total[5]={0.0,15.75,-10.0};

6. Here initialize the first three elements to 0.0, 15.75 and -10.0 and the remaining two elements to 0.0.
7. We also declare the array as variant. In that no need to enter the size of array.

   Ex int a[]={1,3,4};

8. That is called as variant and in that we can store the number of element.
9. Similar, We can also declare the character array but we must end the string with '\0'

   ex- char name[]={'b','h','o','s','l','e','\0'}

➢ **Entering data into an array**

```
int marks[30];
for ( i = 0 ; i <= 29 ; i++ )
{ printf ( "\nEnter marks " ) ; scanf
        ( "%d", &marks[i] ) ;
}
```

The 'for' loop causes the process of asking and receiving a student's marks from the user to be repeated 30 times.

- **Reading Data from an Array**

Suppose we want to read the above array of 30 student's marks and calculate the average of marks.

```
int sum=0, avg;
for ( i = 0 ; i <= 29 ; i++ )
{
            sum=sum+marks[i];
    }
    Avg=sum/30;
    printf ( "\nAverage of marks = %d ", avg ) ;
```

4.4. Two dimensional array

1. Two-dimensional array is array with two dimensions or with row and column.
2. Generally, values in table of same data types are stored as two dimensional arrays.

- **Declaration of two-dimensional array**

1. An array is declared by writing the data types, followed by the name, followed by the row-size in first brackets and column-size in second brackets.
2. Syntax: data type array_name[row-size][colum-size];
3. example:

   int n[12][10];

- **Initialization of one dimensional array**

1. We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.
2. Syntax:

   Data type array-name[row-size][column-size]={{list of values in first row}, {list of values in second row}, ... , {list of values in row-size th row}} ;

   The values in the list are separated by commas

3. Ex- int number[3][4]={{0,0,0,0}, {1,2,5,4}, {5,9,8,7}};
4. Here declare the two dimensional array number as an array of row-size 3 and columsize 4 will assign value as shown in { } each elements.
5. We also declare the array as variant. In that no need to enter the size of array.

Ex int a[ ][ ] = { {0,0,0,0}, {1,2,5,4}, {5,9,8,7} };

6. That is called as variant and in that we can store the number of element.

- **Entering data into an two dimensional array**

```
int marks[30][4]; for ( i =
0 ; i <= 29 ; i++ )
{
        for(int j=0; j<4; j++}
        { printf ( "\nEnter marks " ) ; scanf
                ( "%d", &marks[i] ) ;
        }
```