UNIT - I
INTRODUCTION TO PROGRAMMING

SYLLABUS

Introduction to components of a computer system (disks, memory, processor, where a program is stored and executed, operating system, compilers, etc.)
Idea of Algorithm: steps to solve logical and numerical problems. Representation of Algorithm, Flowchart/ Pseudocode with examples.
From algorithms to programs; source code, variables (with data types) variables and memory locations, Syntax and Logical Errors in compilation, object and executable code.
Arithmetic expressions and precedence.

INTRODUCTION

- Humans use many languages for communication such as English, Hindi, Marathi, etc.
- In our day-to-day life, we come across computers many times. Being a student of computer science, the use of computers has become a primary need. Hence it is very important to understand the language that a computer understands. The computer is nothing but a machine hence it understands machine language unlike humans who understand human language.
- A computer is a machine which can be used as we want, and it will respond as per our instructions. Hence programming language is nothing but a set of instructions that we combine to make the machine perform a certain task.
- There are many programming languages we have in this world. Some are using C or C++ for device-level programming whereas some are using node, angulars, java etc. each language has a purpose and gives a specific solution that most other languages don't have.
- 'C' is a programming language. It is used to write the programs and execute them. There are mainly two types of languages, which are known high-level and low-level languages.
- 'C' language allows manipulation of bits and addresses the basic elements with which the computer functions.
- 'C' permits almost all the data type conversion. C supports a rich set of built-in operators. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators are used in programs to manipulate data and variables. They form a part of the mathematical or logical expression.

COMPONENT OF A COMPUTER SYSTEM

Q.1. Explain Components of a Computer System.

Ans. Components of a Computer System :

A computer is a programmable electronic machine that accepts instructions and data through input devices, manipulates the data according to instructions and provides the final result through the output device. Fig.(1) a conventional block diagram shows the components of a conventional computer.

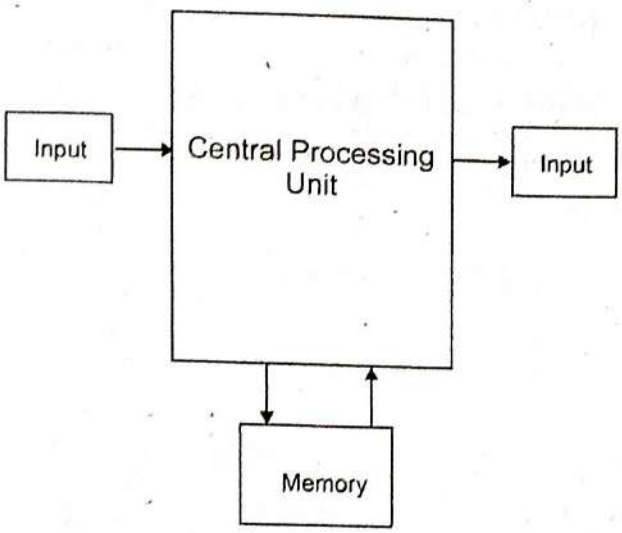


Fig.1. Block diagram of a computer system.

Fig.(2) shows a computer. In brief the various blocks/ components of a computer and summarized below.

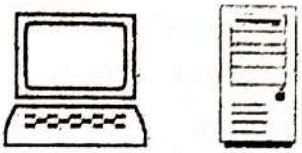


Fig.2. Computer as a single unit

(1) Input devices:

- An input device feeds instructions and data, which are stored in the computer's memory. Consequently, The computer reads data from the memory and manipulates it according to instructions.
- The various input devices to the central processor are keyboard, mouse, analog to digital converters, light pen, trackball, optical character reader, scanner, etc. Fig.(3) shows the input devices of a computer.

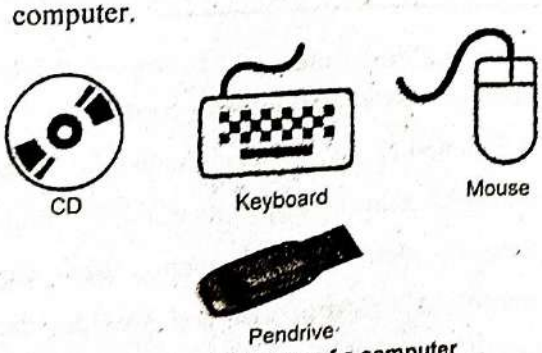


Fig.3. Input devices of a computer

- For example, a Pen drive and a compact disc can be used either as input or output devices.

(2) Output Devices:

- The processed data will be sent to the output device by the computer.
- The output device can be a monitor, printer, LED, seven-segment display, digital-to-analog converter, plotter, etc. Fig.(4) shows the different output devices of a computer.

Fig.4 Output devices of a computer

(3) Memory:

- Volatile semiconductor memory is used for storing instructions and data in the form of ones and zeros. The memory can be called a user's memory or read-write memory.
- The processor first reads the instructions and data then executes the instructions.
- One more memory device is used in the computer which is called 'read-only memory' (non-volatile memory), in which a program is installed permanently/fixed for providing certain operations.

(4) Central Processing Unit:

- The heart of the computer system is a central processing unit. It comprises arithmetic and logical unit, registers and control unit.
- The arithmetic and logical unit perform operations based on instructions.
- The control unit generates the timing and control signals for carrying out the operations within the processor. The registers are used for storing the result temporarily.

Q.2. Explain with suitable examples the different sections involved in the basic structure of any C program.

OR Explain structured programming in C.

Ans. Structure of 'C' program :

- The general basic structure of the C program is shown in Fig. below.

- The whole program is controlled within main ( ) along with left-brace denoted by "{"and right braces denoted by"}". If one needs to declare local variables and executable program structures are enclosed within "{" and"}" is called the body of the main function.
- The main( ) function can be preceded by documentation, preprocessor statements and global declarations.

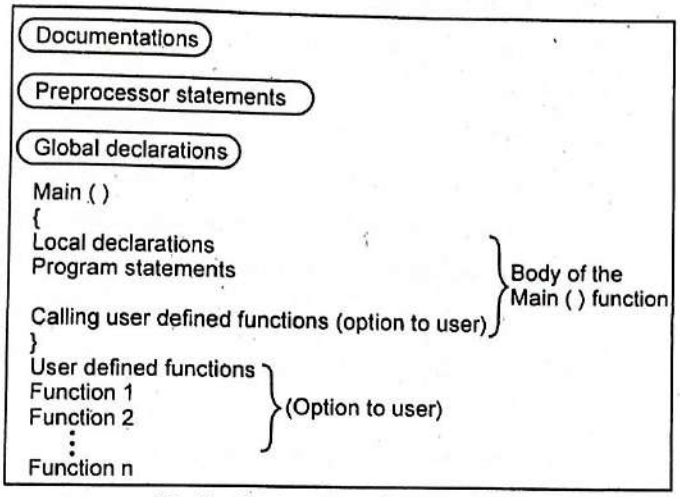


Fig. Basic structure of a C program

(i) Documentations:

The documentation section consist of a set of comment lines giving the name of the program, another name and other details, that the programmer would like to use later.

(ii) Preprocessor Statements:

- The preprocessor statements begin with the # symbol and are also called the preprocessor directive.
- These statements instruct the compiler to include C preprocessors such as header files and symbolic constants before compiling the C program.
- Some of the preprocessor statements are listed below.

```
#include <stdio . h>
#include <math . h>
#include <stdlib . h>
#include <conio . h>
```
} header files

```
#define PI 3.1412.
#define TRUE 1
#define FALSE 0
```
} Symbolic constants

(iii) Global Declarations:

- The variables that are declared before the main ( ) function, as well as user defined functions are called global variables.
- These global variables can be accessed by all the user-defined functions including main ( ) function.

(iv) The main( ) function:

- Each and every C program should contain only one main ( ).
- The C program execution starts with main ( ) function.
- No C program is executed without the main function.
- The main ( ) function should be written in small (lowercase) letters and it should not be terminated by a semicolon.
- main( ) executes user-defined program statements, library functions and user-defined functions and all these statements should be enclosed within left and right braces.

(v) Braces:

- Every C program should have a pair of curly braces ({, }).
- The left braces indicate the beginning of the main ( ) function and the right braces indicate the end of the main ( ) function.
- These braces can also be used to indicate the user-defined functions beginning and ending.
- These two braces can also be used in compound statements.

(vi) Local Declarations:

- The variable declaration is a part of the C program and all the variables are used in the main ( ) function should be declared in the local declaration section is called local variables.
- Not only variables, we can also declare arrays, functions, pointers etc.
- These variables can also be initialized with basic data types.
- **For example:**

```
main ( )
{
```

```
int sum = 0;
int x ;
float y ;
}
```

- Here, the variable sum is declared as an integer variable and it is initialized to zero.
- Other variables that are declared as int and float and these variables inside any function are called local variables.

(vii) Program Statements:

- These statements are building blocks of a program.
- They represent instructions to the computer to perform a specific task (operations).
- An instruction may contain input-output statements, arithmetic statements, control statements, simple assignment statements and any other statements and it also includes comments that are enclosed within /* and*/.
- The comment statements are not compiled and executed and each executable statement should be terminated with a semicolon.

(viii) User-defined Functions :

- These are subprograms, generally, a subprogram is a function and these functions are written by the user and are called user-defined functions.
- These functions are performed by user specific tasks and this also contains a set of program statements.
- They may be written before or after a main( ) function and called within a main( ) function. This is optional to the programmer.

Q.3. Explain the Process of Compiling a Program.

Ans. Process of Compiling a Program:

A program as C source code, compile it (or try to run it from our programming editor, which will typically compile and build automatically) to turn it into object code, and then build the program using a linker program to create an executable program that contains instructions written in machine code, which can be understood and run on the computer that you are working on.

Hold down the Alt key and press F9.

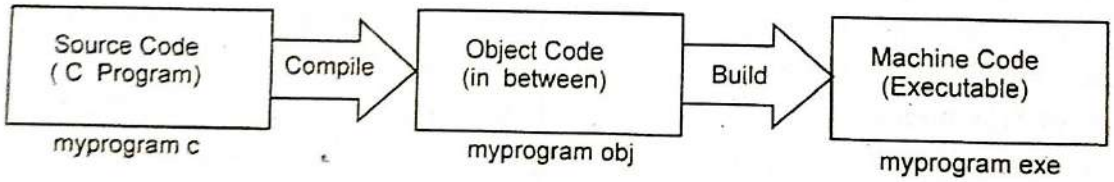


Fig. Compilation Process

- To compile a C program that we have typed in using C, C++ Editor, do one of the following:
- Click on the menu and choose Compile.

To run the program, do one of the following:

- Click on the menu and choose Run.
- Press Ctrl and F9 together.

Q.4. How a program is Compiled and Executed in C-language also mentioned steps in detail?

Ans.

Compiler converts a C program into an executable. Following four phases for a C program to become executable:

(1) C pre-processor: This program accepts C source files as input and produces a file that is passed to the compiler. The preprocessor is responsible for removing comments and interpreting special C language

preprocessor directives, which can be easily identified as they begin with the # symbol. These special preprocessor statements libraries include the following:

(i) **#include:** The preprocessor adds the information about the object code used in the body of the main function. These files are called header files.

(ii) **#define:** This directive assigns a symbolic name to a constant. Symbolic names are used to make programs more readable and maintainable.

(iii) **#ifdef:** This directive is called conditional compilation and allows the programmer to write programs that are adaptable to different operating environments.

(2) **C compiler:** This program translates the C language source code into the machine assembly language.

(3) **Assembler:** The assembler accepts the C compiler output and creates object code. If the program does not contain any external function calls, this code is directly executable.

(4) **Linker:** If a source file references library functions, which are defined in other source files, the linker combines these functions with the main() function to create an executable program file.

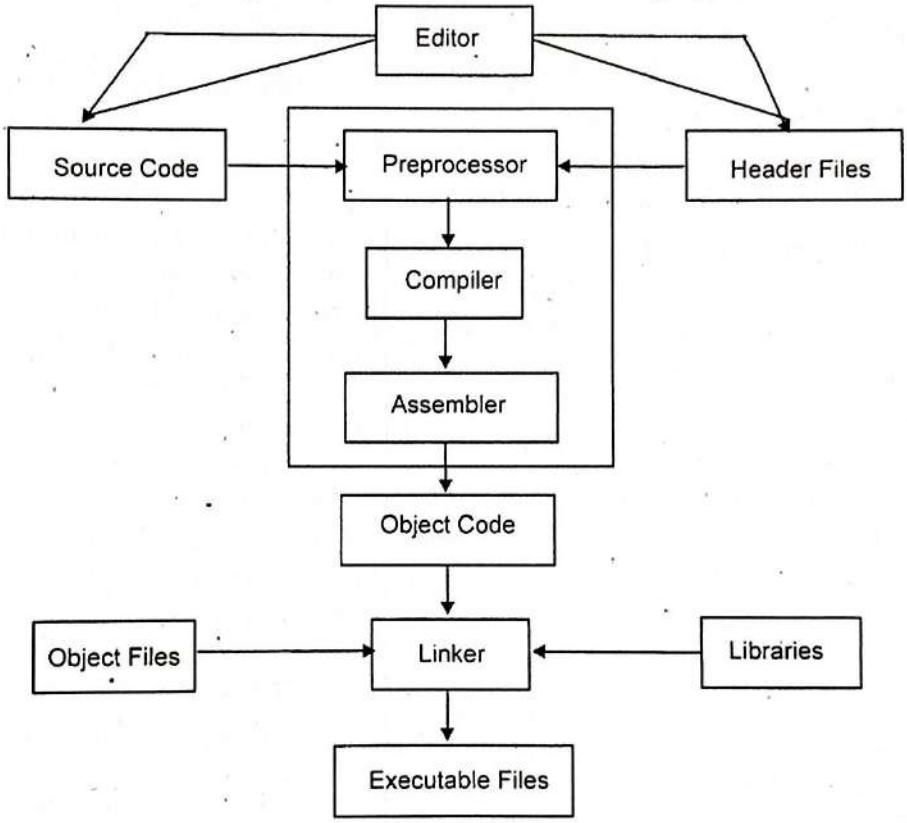


Fig. Stages of Compilation and Execution

Q.5. What are the different types of errors in C? Explain with examples.

Ans. Types of Errors:

Errors are the problems or faults that occur in the program. There are commonly five types of errors that exist in C programming:

(i) Syntax Error

(ii) Run-time Error

(iii) Linker Error

(iv) Logical Error

(v) Semantic Error

(i) **Syntax Error:**

- Syntax errors are also known as compilation errors as they occurred at the compilation time. These errors mainly occur due to mistakes while typing when the programmer does not follow the

grammar rule of the programming language. These errors can be easily corrected.

- Commonly occurred syntax errors are: using a variable without its declaration, missing the semicolon (;) at the end of the statement.

```
...
int a = 6,  b = 7;        /*Error: C is undefined */
c = a + b;
printf("%d", c);
```

(ii) Run-time Error / Execution Error:

- The errors exist during the execution time even after the successful compilation known as run-time errors. When the program is running, and it is not able to perform the operation is the main cause of the run-time error. The division by zero is a common example of run-time error. These errors are very difficult to find, as the compiler does not point out a these errors.

```
...
int a = 6,  b = 0, c;    /*Error :divided by zero*/
   c = a / b;
printf("%d", c);
```

(iii) Linker Error:

Linker errors are mainly generated when the executable file of the program is not created. This can happen either due to the wrong function prototyping or usage of the wrong header file.

```
#includee<studio.h>
                    /*Error :No such header file*/
#include<conio.h>
void main( )
{
 printf("enter first number:");
 ....
}
```

(iv) Logical Error:

A logical error is an error that leads to an undesired output. These errors produce incorrect output, but they are error-free, known as logical errors. The occurrence of these errors mainly depends upon the logical thinking of the developer.

```
...
int a = 6,  b = 7, avg;
c = a + b / 2;
                /*Error : braket is not used in formule*/
printf("%d", c);
```

(v) Semantic Error:

Semantic errors are the errors that occurr when the statements are not understandable by the compiler. The following can be the cases for the semantic error:

```
...
int a = 6,  b = 7, c;
a + b = c;              /*Error : Expression wrong*/
printf("%d", c);
```

Q.6. Explain Characteristics of C language.

Ans. Following are the list of characteristics that define the language:

(i) Small size.
(ii) Extensive use of function calls.
(iii) Loose typing.
(iv) Structured language.
(v) Low level (Bitwise) programming readily available.
(vi) Pointer implementation - extensive use of pointers for memory, array, structures and functions.
(vii) It has high-level constructs.
(viii) It can handle low-level activities.
(ix) It produces efficient programs.
(x) It can be compiled on a wide variety of computers.
(xi) C has now become a widely used professional language for various reasons.

Q.7. Write the importance of headers and their usage in C.

Ans. Importance of Headers:

Header files contain declaration information for functions or constants that are referred to in programs. They are used to keep source-file size to a minimum and to reduce the amount of redundant information that must be coded.

(i) # includes:

An include directive tells the preprocessor to include the contents of the specified file at the point in the program. Pathnames must either be enclosed by double quotes or angle brackets.

Example:

(a) # include <stdio.h>

(b) # include"mylib.h"

(c) # include"mine\include\mylib.h"

In example (a) above, the < > tells the preprocessor to search for the included file in a special known include directory or directories.

In example (b), the double quotes (" ") indicate that the current directory should be checked for the header file first. If it is not found, the special directory (or directories) should be checked.

The example (c) is similar, but the named relative directory \mine\include is checked for the header file mylib.h.

Relative paths can also be proceeded by the .\ or ..\ notation; absolute paths always begin with a \.

(ii) # defines:

ANSI C allows you to declare constants. The # define directive is used to tell the preprocessor to perform a search-and-replace operation.

Example:

1. # define Pi 3.14159
2. # define Tax-rate 0.0735

In the example above, the preprocessor will search through the source file and replace every instance of the token Pi with 3.14159

After performing the search and replace the operation, the preprocessor removes the # define line.

ALGORITHM

Q.8. What is an Algorithm? Explain in detail along with an example.

Ans. Algorithm:

- An algorithm is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. No matter what the input values may be, an algorithm terminates after executing a finite number of instructions.
- We represent an algorithm using a pseudo-language that is a combination of the constructs of a programming language together with informal English statements.
- The ordered set of instructions required to solve a problem is known as an *algorithm*.

Example:

- To understand this concept let's take the example of going to college. Consider ram wants to go to college on Monday. For this, he has to follow certain steps. It's not like he went to college at 4 am in a night dress.

  The steps that Ram will follow are:

  Step 1: Get up in the morning.

  Step 2: Get fresh and take all books for today's lecture.

  Step 3: Check if homework is completed.

  Step 4: Take a bath.

  Step 5: Get dressed and walk to college.
- To go to college Ram followed the above 5 steps. Now everyone doesn't need to follow the same 5 steps. There may be some other approaches for this task (going to college) to perform.
- An algorithm is nothing but a set of steps that we write to do a certain task.
- To call some set of steps an algorithm one should have the following properties:

  (i) An algorithm may or may not have input.

  (ii) It should be finite.

  (iii) It should have an output.
- These groups of steps are written into some programming language so that the computer can perform the same task that we have written as an algorithm.

Example:

Design an algorithm to add two numbers and display the result.

Algorithm:

Step 1 : START.

Step 2 : declare three integers a, b & c.

Step 3 : define values of a & b.

Step 4 : add values of a & b.

Step 5 : store output of step 4 to c.

Step 6 : print c.

Step 7 : STOP.

Q.9. Give the Characteristics of an Algorithm.

Ans. Algorithm:

An algorithm should have the following characteristics:

(i) **Unambiguous:** An Algorithm should be unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.

(ii) **Input:** An algorithm should have 0 or more well-defined inputs.

(iii) **Output:** An algorithm should have 1 or more well-defined outputs, and should match the desired output.

(iv) **Finiteness:** An Algorithm must terminate after a finite number of steps.

(v) **Feasibility:** An Algorithm should be feasible with the available resources.

(vi) **Independent:** An algorithm should have step-by-step directions, which should be independent of any programming code.

Q.10. Enlist the advantages and disadvantages of the algorithm.

Ans. Advantages:

(i) It is a step-wise representation of a solution to a given problem, which makes it easy to understand.

(ii) An algorithm uses a definite procedure.

(iii) It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.

(iv) Every step in an algorithm has its logical sequence so it is easy to debug.

(v) By using an algorithm, the problem is broken down into smaller pieces or steps hence, it is easier for a programmer to convert it into an actual program.

Disadvantages:

(i) Writing an algorithm takes a long time.

(ii) An Algorithm is not a computer program, it is rather a concept of how a program should be.

Q.11. Write an algorithm to find the largest number among three numbers with a flowchart.

Ans. Algorithms:

Step 1: Start

Step 2: Declare variables a,b and c.

Step 3: Read variables a,b and c.

Step 4: If a > b

If a > c

Display a is the largest number.

Else

Display c is the largest number.

Else

If b > c

Display b is the largest number.

Else

Display c is the greatest number.

Step 5: Stop.

Q.12. Write the algorithm. Find the factorial of a number.

Ans. Algorithm:

Step 1: Start

Step 2: Declare variables n, factorial and i.

Step 3: Initialize variables

factorial ← 1

I ← 1

Step 4: Read value of n

Step 5: Repeat the steps until i = n

factorial ← factorial

i ← i + 1

Step 6: Display factorial

Step 7: Stop.

Q.13. Find the Root of the quadratic equation $ax^2 + bx + c = 0$.

Ans. Algorithm:

Step 1: Start

Step 2: Declare variables a, b, c, D, x1, x2, rp and ip;

Step 3: Calculate discriminant

D ← b2 – 4ac

Step 4: If $D \geq 0$

$r1 \leftarrow (-b + \sqrt{D})/2a$

$r2 \leftarrow (-b - \sqrt{D})/2a$

Display r1 and r2 as roots.

Else

Calculate real part and imaginary part

$rp \leftarrow -b/2a$

$ip \leftarrow \sqrt{(-D)}\ /\ 2a$

Display rp+j(ip) and rp-j(ip) as roots

Step 5: Stop

Q.14. Develop an algorithm to calculate the area of a circle by taking radius as input.

Ans. Algorithm :

Step 1: Start

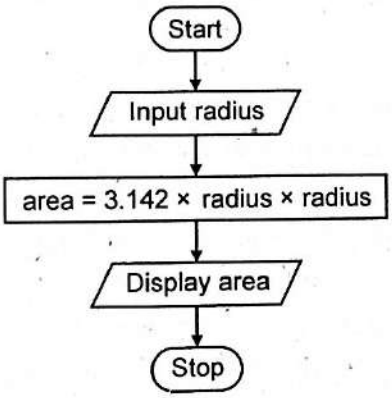
Step 2: Input value of radius

Step 3: Calculate area using the formula, area = 3.142 × radius × radius

Step 4: Display the value of area

Step 5: Stop

Flow chart:



Q.15. Develop an algorithm to find sum of first N positive numbers. i.e. 1 + 2 + 3 + ... + N.

Ans. Algorithm:

Step 1: Start

Step 2: Input value of N

Step 3: Assign 0 to Sum

Step 4: Assign 1 to Num

Step 5: Add Num to Sum and store result in Sum

i.e. Sum = Sum + Num

Step 6: Increment value of Num by 1.

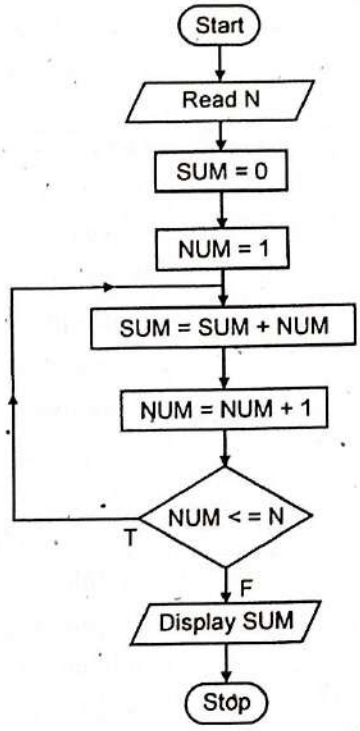
Step 7: If Num is <= N, then

goto step 5

Step 8: Display value of Sum

Step 9: Stop

4M



FLOW CHART

Q.16. Explain flowchart with symbols along with an example.

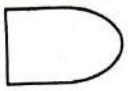
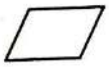
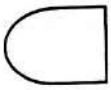
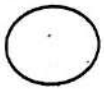
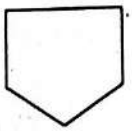
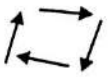
OR Give the guidelines of flowchart.

Ans. Flow Chart:

A flowchart is a diagrammatic representation of an algorithm. A flowchart can be helpful for both writing programs and explaining the program to others.

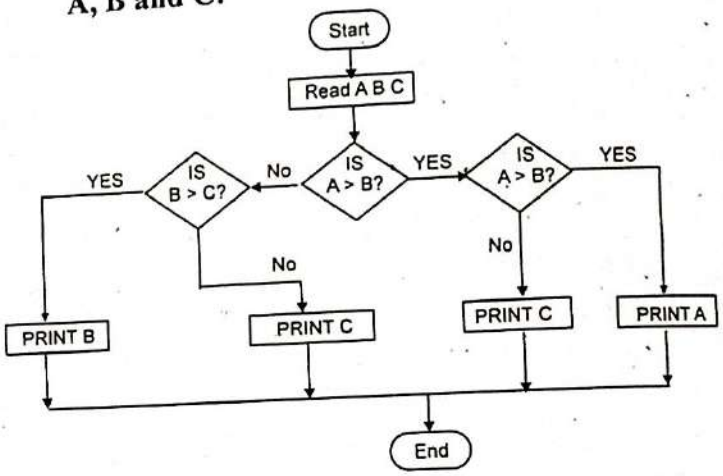
The flowchart shows the steps as boxes of various kinds and their order by connecting the boxes with arrows. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

Sr. No.	Symbol	Type of operation/C Code Example
(i)		**Terminator:** Indicates the beginning or end of a program flow in your diagram.
(ii)		**Process:** Indicates any processing function.

(iii)		**Decision:** Indicates a decision point between two or more paths in a flowchart.
(iv)		**Delay:** Indicates a delay in the process.
(v)		**Data:** Can represent any type of data in a flowchart.
(vi)		**Subroutine:** Indicates a predefined (named) process, such as a subroutine or a module.
(vii)		**Preparation:** Indicates a modification to a process, such as setting a switch or initializing a routine.
(viii)		**Display:** Indicates data that is displayed for people to read, such as data on a monitor or projector screen.
(ix)		**Connector:** Indicates an inspection point.
(x)		**Off-page connector:** Use this shape to create a cross-reference and type from a process on one page to a process on another page.
(xi)		**Document:** Indicates data that can be ready by people, such as printed output.
(xii)		**Multiple Documents:** Indicates multiple documents.
(xiii)		**Comment:** Indicate to add comments or clarification.
(xiv)		**Flow lines or arrows:** Indicates the sequence of steps ("flow of control")

Example:

Flowchart to find the largest of three numbers A, B and C:



Flowchart Guidelines:

The basic guidelines for drawing a flowchart with the above symbols are:

- In drawing a proper flowchart, all requirements should be listed in logical order.
- The flowchart should be neat, clean and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
- The flowchart is to be read left to right or top to bottom.
- A process symbol can have only one flow line coming out of it.
- For a decision symbol, only one flow line can enter it, but multiple lines can leave it to denote possible answers.

Q.16. Give advantages and disadvantages of flowchart.

Ans. Advantages of Flowchart:

(i) The flowchart is an excellent way of communicating the logic of a program.

(ii) It is easy and efficient to analyze the problem using a flowchart.

(iii) During the program development cycle, the flowchart plays the role of a guide or blueprint. Which makes the program development process easier.

(iv) After the successful development of a program, it needs continuous timely maintenance during its operation. The flowchart makes program or system maintenance easier.

(v) It helps the programmer to write the program code.

(vi) It is easy to convert the flowchart into any programming language code as it does not use any specific programming language concept.

Disadvantage of Flowchart:

(i) The flowchart can be complex when the logic of a program is quite complicated.

(ii) Drawing a flowchart is a time-consuming task.

(iii) Difficult to alter the flowchart. Sometimes, the designer needs to redraw the complete flowchart to change the logic of the flowchart or alter the flowchart.

(iv) Since it uses special sets of symbols for every action, it is quite tedious task to develop a flowchart as it requires special tools to draw the necessary symbols.

(v) In the case of the complex flowchart, other programmers might have a difficult time understanding the logic and process of the flowchart.

(vi) It is just a visualization of a program, it cannot function as an actual program.

KEYWORDS

Q.17. What is Keyword in C ? How many types of keywords are found in the C language?

Ans. Keyword in C:

In C we have 32 keywords, which have their predefined meaning and cannot be used as a variable name. These words are also known as "reserved words". It is good practice to avoid using these keywords as a variable name. All keywords in C are written in lower case.

C Keywords or Reserved Words:

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

Basics usage of these keywords:

(i) **goto:**

The goto statement is used to transfer control of the program to the specified label.

(ii) **int:**

The int keywords are used to declare integer type variables. For example.

(iii) **Short, long, signed and unsigned:**

The short, long, signed and unsigned keywords are type modifiers that alter the meaning of a base data type to yield a new type.

(iv) **return:**

The return keyword terminates the function and returns the value.

(v) **sizeof:**

The sizeof keyword evaluates the size of data (a variable or a constant).

(vi) **register:**

The register keyword creates register variables that are much faster than normal variables.

(vii) **static:**

The static keyword creates a static variable. The value of the static variables persists until the end of the program.

(viii) **struct:**

The struct keyword is used for declaring a structure. A structure can hold variables of different types under a single name.

(ix) **typedef:**

The typedef keyword is used to explicitly associate a type with an identifier.

(x) union:

A union is used for grouping different types of variables under a single name.

(xi) void:

The void keyword meaning is nothing or no value.

(xii) volatile:

The volatile keyword is used for creating volatile objects. A volatile object can be modified in an unspecified way by the hardware.

(xiii) Auto:

The auto keyword declares automatic variables.

(xiv) break and continue:

The break statement terminates the innermost loop immediately when it's encountered. It's also used to terminate the switch statement.

The continue statement skips the statements after it is inside the loop for the iteration.

(xv) switch, case and default:

The switch and case statement is used when a block of statements has to be executed among many blocks.

(xvi) char:

An identifier can be declared constant by using the const keyword.

(xvii) For, do....while and while:

These are types of loops in C programming.

(xviii) If and else:

In C programming, if and else are used to make decisions.

(xix) enum:

Enumeration types are declared in C programming using the keyword enum.

(xx) extern:

The extern keyword declares that a variable or a function has external linkage outside of the file it is declared.

Q.18. How characters are represented in the C language?

Ans. The character that can be used to form words, numbers and expressions depends upon the computer on which the program is run.

The two common standards for representing characters:

(1) ASCII (American standard code for information interchange).

(2) EBCDIC (Extended Binary coded decimal interchange code)

Character	ASCII	EBCDIC
A	65	193
1	49	241
+	43	78

C allows a character constant to be specified as a character enclosed in single quote marks (' ') which the compiler will then translate to a value in the code (ASCII or otherwise).

DATA TYPE

Q.19. Explain Data Type in C.

Ans. Data Type in C:

C data types are defined as the data storage format that a variable can store data to perform a specific operation. Data types are used to define a variable before use in a program. The size of variable, constant and array, is determined by data types.

Data Types:

There are four data types in the C language. They are,

Types	Data Types
Basic data types	int, char, float, double
Enumeration data type	enum
Derived data type	pointer, array, structure, union
Void data type	void

(i) Basic Data Types:

(A) Integer Data Type:

- The integer data type allows a variable to store numeric values.
- The "int" keyword is used to refer to integer data types.

- The storage size of int data type is 2 or 4 or 8 bytes.
- It varies depending upon the processor in the CPU that we use. If we are using a 16-bit processor, 2 bytes (16 bit) of memory will be allocated for an int data type.
- Likewise, 4 bytes a (32 bit) of memory for 32-bit processor and 8 bytes (64-bit) of memory for a 64-bit processor are allocated for int datatype.
- int (2 bytes) can store values from -32,768 to +32,767
- int (4 bytes) can store values from -2,147,483,648 to +2,147,483,647.
- If we want to use the integer value that crosses the above limit, you can go for "long int" and "long long int" for which the limits are very high.
- We can't store decimal values using int data type.
- If we use int data type to store decimal values, decimal values will be truncated and we will get only a whole number.
- In this case, float data type can be used to store decimal values in a variable.

(B) Character Data Type:

- Character data type allows a variable to store only one character.
- The storage size of the character data type is 1. We can store only one character using character data type.
- "Char" keyword is used to refer to the character data type.
- For example, 'A' can be stored using char data type. You can't store more than one character using char data type.

(C) Floating Point Data Type:

Floating point data type consists of 2 types. They are,

(i) Float:

- Float data type allows a variable to store decimal values.
- The storage size of float data type is 4. This also varies depending upon the processor in the CPU as "int" data type.
- We can use upto 6 digits after decimal using float data type.
- For example, 10.456789 can be stored in a variable using float data type.

(ii) Double:

- Double data type is also same as float data type which allows up-to 10 digits after decimal.
- The range for double datatype is from 1E–37 to 1E+37.

C Data Types / Storage Size	Range
char / 1	–127 to 127
int / 2	–32,767 to 32,767
float / 4	1E–37 to 1E+37 with six digits of precision
double / 8	1E–37 to 1E+37 with ten digits of precision
long double / 10	1E–37 to 1E+37 with ten digits of precision
long int / 4	–2,147,483,647 to 2,147,483,647
short int / 2	–32,767 to 32,767
unsigned short int / 2	0 to 65,535
signed short int / 2	–32,767 to 32,767
long long int / 8	– (2power(63) –1) to 2(power)63 –1
signed long int / 4	–2,147,483,647 to 2,147,483,647
unsigned long int / 4	0 to 4,294,967,295
unsigned long long int / 8	2(power)64 –1

(iii) Enumeration Data Type:

- Enumeration data type consists of named integer constants as a list.
- It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.

Syntax:

enum identifier [optional{ enumerator-list }];

Example:

```
enum month { Jan, Feb, Mar };
/* Jan, Feb and Mar variables will be assigned to
0, 1 and 2 respectively by default */
enum month { Jan = 1, Feb, Mar };
/* Feb and Mar variables will be assigned to 2
                    and 3 respectively by default */
enum month { Jan = 20, Feb, Mar };
/* Jan is assigned to 20. Feb and Mar variables
   will be assigned to 21 and 22 respectively by
                                default */
```

The above enum functionality can also be implemented by "#define" preprocessor directive as given below.

The above enum example is the same as given below:

```
#define Jan 20;
#define Feb 21;
#define Mar 22;
```

(iv) Derived Data Type:

Array, pointer, structure and union are called derived data types in the C language.

(v) Void Data Type:

- Void is an empty data type that has no value. This can be used in functions and pointers.
- Void data type use in function with simple call by value and call by reference example programs.

IDENTIFIERS

Q.18. What are Identifiers and how they are used in C? Explain.

Ans. Identifiers:

- The C language defines the names that are used to reference variables, functions, labels, and various other user-defined objects as identifiers.
- An identifier can vary from one to several characters. The first character must be a letter or an underscore with subsequent characters being letters, numbers, or underscore.

Example:

(a) int i, j,k;
float x,y,z;
char q;

Here i, j, k, x, y, z and q are identifiers.

(b) int money
double, account balance

Here, money and account balance are identifiers.

Identifiers Rules for Declaration:

(i) The first character must be a letter or underscore.
(ii) Characters in identifiers are limited up to 31 in the ANSI C compiler(it may vary).
(iii) Blank spaces and commas are not allowed within an identifier.
(iv) Identifiers should not start with a digit.
(v) Identifier name should not be the keyword.
(vi) Identifiers must be short and meaningful.

VARIABLES

Q.19. What are Variables? Explain different types of variables along with an example.

Ans. Variables:

A variable is a data name that may be used to store a data value. Variables are declared in three basic places: inside functions, in the definition of function parameters, and outside the functions. If they are declared inside functions, they are called local variables, if in the definition of functions then formal parameters and outside of all functions, then global variables.

(i) Global Variables:

- Global variables are known throughout the program. The variables hold their values throughout the execution of the program. Global variables are created by declaring them outside of any function.
- Global variables are defined above main () in the following way:

```
short number, sum;
int bignumber, bigsum;
char letter;
main()
{
.........
}
```

- It is also possible to pre-initialise global variables using the = operator for assignment.

Example:

```
float sum = 0.0;
int bigsum = 0;
char letter ='A';
main()
{

}
```

This is the same as:

```
float sum;
int bigsum;
char letter;
main()
{
sum=0.0;
bigsum=0;
letter='Aa';
}
```

is more efficient.

C also allows multiple assignment statements using =, for example: a = b = c = d = 3;

which is the same as, but more efficient than:

```
a =3;
b =3;
c =3;
d =3;
```

This kind of assignment is only possible if all the variable types in the statement are the same.

(ii) Local Variables:

- Variables that are declared inside a function are called "local variables". These variables are referred to as "automatic variables". Local variables may be referenced only by statements that are inside the block in which the variables are declared.
- Local variables exist only while the block of code in which they are declared is executing, i.e. a local variable is created upon entry into its block and destroyed upon exit.

Example:

Consider the following two functions:

```
void func1 (void)
    {
        int x;
        x = 10;
    }

    void func2 (void)
    {
        int x;
        x = -199;
    }
```

- The integer variable x is declared twice, once in func1() and once in func2(). The x in func1() has no bearing on or relationship to the x in func2(). This is because each x is only known to the code within the same block as the variable declaration.
- There are some rules for choosing variable names:

(i) A variable name can consist of capital letters A-Z, lowercase letters a-z digits 0-9 and the underscore character.

(ii) The first character must be a letter or underscore.

(iii) Blank spaces cannot be used in variable names.

(iv) Special characters like #, $ are not allowed.

(v) C keywords cannot be used as variable names.

(vi) Variable names are case sensitive.

(vii) Values of the variables can be numeric or alphabetic.

(viii) Variable type can be char, int, float, double or void.

Q.20. Explain how variables are declared.

Ans. Variables:

Variables are declared with the help of declaration variables. Declaration of variables does two things:

(a) It tells the compiler what the variable name is.

(b) It specifies what type of data the variable will hold.

Primary Type Declaration :

A variable can be used to store a value of any data type.

Syntax :

Data-type $v_1, v_2, ... v_n$;

$v_1, v_2, ... v_n$ are the names of variables.

Examples : int count;
int number, total;
double ratio;

User-Defined Type Declaration :

"Type definition" feature allows users to define an identifier that would represent an existing data type.

Syntax :

typedef type identifier;

where, type refers to an existing data type identifier refers to the "new" name given to the data type.

Examples : typedef int units;
typedef float marks;

Enumerated data type :

Enumerated **data type is user defined data type provided by ANSI standard. It is defined as follows:**

Enum identifier {value 1, value 2, ... Value n};

Example : enum day {Monday, Tuesday ... Sunday};

```
enum day week_st, week_end;
week_st = Monday;
week_end = Friday;
if(week_st = = Tuesday)
week_end = Saturday;
```

Q.21. What is difference between normal variable and address variable?

Ans.

Sr. No.	Normal variable	Pointer/Address variable
(i)	Normal variable carries the specified data.	Pointer variable carries the address of the specified data.
(ii)	Normal variable contains the value of variable either int or float.	Pointer variable contains the address of another variable.
(iii)	**Example :** int x = 10; where 'x' is the normal variable which carries 10.	**Example :** int x = 10; ptr p = & x; pointer variable is 'p' which carries address of the integer variable 'x'.

Q.22. How values are assigned to variables?

Ans.

- Value can be assigned to variables using the assignment operator = as follows :
  Variable_name = constant;
  Examples: initial_value = 0;
  final_value = 100;
- An assignment statement implies that the value of the variable on the left of the 'equal sign' is set equal to the value of the quantity on the right.
- It is possible to assign a value to a variable at the time the variable is declared. It takes the following form :
  data-type variable_name = constant;
  Example: char yes = 'x';
- The process of giving initial values to variables is called initialization. C permits the initialization of more than one variables in one statement using multiple assignment operators.
  Example: p = q = s = 0;
- Another way of giving values to variables is to input data through keyboard using the scanf function.

CONSTANTS

Q.23. What are Constants? Explain.

Ans. Constants:

- C Constants are also that like normal variables. But, the only difference is, that their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called literals. Constants may be belonging to any of the data types.
- We can define constants in a C program in the following ways.
  By "const" keyword

By "#define" preprocessor directive

When we try to change constant values after defining in C program, it will go through error.

Syntax:

const data_type variable_name;

(or) const data_type *variable_name;

Example:

int const a = 1;

const in a = 2;

Types of C Constant:

(i) Integer Constants.

(ii) Real or Floating Point Constants.

(iii) Octal and Hexadecimal Constants.

(iv) Character Constants.

(v) String Constants.

(vi) Backslash Character Constants.

Sr. No.	Constant Type	Data Type (Example)
(i)	Integer Constants	int (53, 762, -478 etc) unsigned int (5000u, 1000U etc) long int, long long int (483,647 2,147,483,680)
(ii)	Real or Floating Point Constants	float (10.456789) doule (600.123456789)
(iii)	Octal Constant	int (Example: 013 /*starts with 0 */)
(iv)	Hexadecimal Constant	int (Example: 0x90 /*starts with 0x*/)
(v)	Character Constants	char (Example: 'A', 'B', 'C')
(vi)	String Constants	char (Example: "ABCD", "Hai")

Rules For Constructing C Constant:

(i) **Integer Constants:**

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can either be positive or negative.
- No commas or blanks are allowed within an integer constant.
- If no sign precedes an integer constant, it is assumed to be positive.
- The allowable range for integer constants is 32768 to 32767.

(ii) **Real Constants:**

- A real constant must have at least one digit.
- It must have a decimal point.
- It could be either positive or negative.
- If no sign precedes an integer constant, it is assumed to be positive.
- No commas or blanks are allowed within a real constant.

(iii) **Character and String:**

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.
- The maximum length of a character constant is 1 character.
- String constants are enclosed within double-quotes.

(iv) **Backslash Character Constants:**

- Some characters have special meanings in the C language.
- They should be preceded by a backslash symbol to make use of their special function.
- Given below is the list of special characters and their purpose.

Backslash Character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab

\"	Double quote
\'	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark
\N	Octal constant (N is an octal constant)
\XN	Hexadecimal constant (N – hex.deml enst)

Example:

```
/* Program Using Const Keyword in C*/
#include <stdio.h>
void main()
{
const int height = 100;            /*int constant*/
const float number = 3.14;         /*Real constant*/
const char letter = 'A';           /*char constant*/
const char letter_sequence[10] = "ABC";
                                   /*string constant*/
const char backslash_char = '\?';
                                   /*special char enst*/
printf("value of height : %d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n",
letter_sequence);
printf("value of backslash_char : %c \n",
backslash_char);
}
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

Q.24. What is Manifest Constants?

Ans. Manifest Constants:

- The # define directive is used to tell the preprocessor to perform a search-and-replace operation.

  Example: # define Pi 3.14159
  # define Tax-rate 0.0735
- In the example above, the preprocessor will search through the source file and replace every instance of the token Pi with 3.14159
- After performing the search and replace operation, the preprocessor removes the # define a line.
- The following are two purposes for defining and using manifest constants:

(i) They improve source code readability.

(ii) They facilitate program maintenance.

Q.25. Difference between Constant and Variables in C language.

Ans.

Sr. No.	Constant	Variables
(i)	A constant does not change its value over time.	A variable, on the other hand, changes its value dependent on the equation.
(ii)	Constants are usually written in numbers.	Variable are specially written in letters or symbols.
(iii)	Constants usually represent the known values in an equation, expression or line of programming.	Variables, on the other hand, represent their unknown values.
(iv)	Constants are used in computer programming.	Variables also have their uses in computer programming and application.

OPERATORS

Q.26. What is an Operator? What are the different types of operators used in 'C'?

Ans. Operator:

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined to form expressions.
- Consider the expression A + B * 5, where, + * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression.

Types of C Operators:

Following is the list of operators used in the C language:

(i) Arithmetic operators,
(ii) Assignment operators,
(iii) Relational operators,
(iv) Logical operators,
(v) Bitwise operators,
(vi) Conditional operators (ternary operators),
(vii) Increment/decrement operators,
(viii) Special operators.

Q.22. Explain the Arithmetic Operators in C with an example.

Ans. Arithmetic Operators:

- C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs.

Operation	Symbol	Meaning
Add	+	Addition or unary plus.
Subtract	-	Subtraction or unary minus.
Multiply	*	Multiplication.
Division	/	Remainder lost.
Modulus	%	Gives the remainder on integer division, so 7 % 3 is 1.
--	Decrement	
++	Increment	

- As well as the standard arithmetic operators (+ - * /) found in most languages, C provides some more operators.

  Assignment is = *i.e.* *i*= 4; ch = `y';
- Increment ++, Decrement -- which are more efficient than their longhand equivalents.

  For example, X = X + 1 can be written as ++X or as X++. There is however a difference when they are used in an expression.
- The ++ operators can be either post-fixed or pre-fixed. A pre-increment operation such as ++a increments the value of a by 1 before a is used for computation, while a post-increment operation such as a++, uses the current value of 'a' in the calculation and then increments the value of a by 1. Consider the following:

  X = 10;
  Y = ++X;
- In this case, Y will be set to 11 because X is first incremented and then assigned to Y. However if the code had been written as

  X = 10;
  Y = X++;

  Y would have been set to 10 and then X incremented. In both cases, X is set to 11; the difference is when it happens.
- The % (modulus) operator only works with integers.

  Division / is for both integer and float division. So be careful. The answer to:x = 3 / 2 is 1 even if x is declared a float.

  Rule: If both arguments of / are integer then do integer division. So make sure we do this. The correct (for division) answer to the above is x = 3.0 / 2 or x= 3 / 2.0 or (better) x = 3.0 /2.0.

  There is also a convenient shorthand way to express computations in C. It is very common to have expressions like:i= i+ 3 or x = x * (y + 2)

This can be written in C (generally) in a *shorthand* form like this:

We can rewrite i = i + 3 as i + = 3 and x = x * (y + 2) as x *= y + 2.

Note: That x *= y + 2 means $x = x * (y + 2)$ and $x = x * y + 2$.

Q.28. Explain Assignment Operators.

Ans. Assignment Operators:

Assignment operators are used for assigning value to the variable. Like any other operator, C also supports Assignment Operator which is a binary operator that operates on any two operands. It has 2 values the right value and the left value. It has lower precedence than all available operators but has higher precedence than the comma operator.

Different List of Assignment Operators in C:

Following are is the list of Assignment operators in C as follows:

- **The simple assignment operator(=):** This operator Assigns values from the right operands to the left operand.
- **Add AND operator (– =):** This operator adds the right operand to the left operand and assigns the output to the left operand.
- **Subtract AND operator (–=):** This operator subtracts the right operand from the left operand and assigns the result to the left operand.
- **Multiply AND operator (*=):** This operator multiplies the right operand with the left operand and assigns the result to the left operand.
- **Divide AND operator (/=):** This operator divides the left operand with the right operand and assigns the result to the left operand.
- **Modulus AND operator (%=):** This operator takes modulus using two operands and assigns the result to the left operand.

There are many other assignment operators such as Left shift AND (<<=) operator, Right shift AND operator (>>=), Bitwise AND assignment operator (&=), Bitwise exclusive OR and assignment operator (^=), Bitwise inclusive OR and assignment operator(|=)

Example :

The following program illustrates the use of arithmetic operators :

```
/* a program demonstrates C arithmetic operators */
main()
{
    int x= 20, y = 10,z;
    clrscr();
    /* demonstration of + operator */
    z = x + y;
    printf("Addition is %d\n",z);
    /* demonstration of – operator */
    z = x – y;
    printf("Substraction is %d\n",z);
    /* demonstration of * operator */
    z = y * x;
    printf("Multiplication is %d\n",z);
    /* demonstration of / operator */
    z = x / y;
    printf("Division is %d\n",z);
    /* demonstratation modulus operator % */
    z = y %2;
    printf("Remainder is %d\n",z);
    getch();
}
```

Output:

Addition is 30
Substraction is 10
Multiplication is 200
Division is 2
Remainder is 0

Q.29. Explain Relational Operators.

Ans Relational Operators:

The relational operators are used to determine the relationship of one quantity to another. They always return 1 or 0 depending upon the outcome of the test. The relational operators are as follows:

Operator	Action
= =	equal to
! =	not equal to
<	Less than
<=	less than or equal to
>	Greater than
>=	Greater than or equal to

To test for equality is ==

If the values of x and y, are 1 and 2 respectively then the various expressions and their results are:

Expression	Result	Value
X != 2	False	0
X == 2	False	0
X == 1	True	1
Y != 3	True	1

Example :

The following example illustrates the use of relational operators :

```
int   x = 10;
      x > 5       /* it returns 1 */
      x > = 12    /* it returns 0 */
      x < 15      /* it returns 1 */
      x <= 10     /* it returns 1 */
      x==10       /* it returns 1 */
      x ! = 10    /* it returns 0 */
```

/*To check whether a given character is an uppercase alphabet or a lower case alphabet or digits or special symbols.*/

```
main()
{
char ch;clrscr();
printf("Enter any character");
scanf("%c",&ch);
if(ch>=65 && ch<=90)
{
      printf("It is an Uppercase alphabet");
}
else
{
if(ch>=97 && ch<=122)
{
printf("It is an Lowercase alphabet");
}
else
{
if(ch>=48 && ch<=57)
{
printf("It is Digit");
}
else
      {
            printf("It is Special Symbol");
                  }
            }
      }
getch();
}
```

Output:

```
Enter any character A
It is an Uppercase alphabet

Enter any character a
It is an Lowercase alphabet

Enter any character 7
It is Digit
Enter any character @
It is Special Symbol
```

Q.30. Explain the importance of Logical Operators, with a suitable C program.

Ans. Logical (Comparison)Operators:

Logical operators are usually used with conditional statements. The three basic logical operators are given below:

(i) && for logical AND,

(ii) || for logical OR and

(iii) ! for not.

The truth table for the logical operators is shown here using one's and Zeroes. (the idea of true and false underlies the concepts of relational and logical operators). In C true is any value other than zero and false is zero. Expressions that use relational or logical operators return zero for false and one for true.

P	Q	P && Q	P || Q	! P
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Example:

(i) x == 6 && y ==7

This while expression will be TRUE (1) if both x equals 6 and y equals 7, and FALSE (0) otherwise.

(ii) x < 5 || x >8

his whole expression will be TRUE (1) if either x is less than 5 or x is greater than 8 and FALSE (0) otherwise.

Example:

```
/* Program for Logical Operators in C/*
#include<stdio.h>
int main()
{
 int m=40, n =20;
 int o=20, p=30;
 if(m>n & m! =0)
 {
  Printf("&& Operator: Both conditions are true \n");
  }
 if (o>p||p!=20)
 {
 printf("\\Operator: Only one condition is true \n");
 }
 if (!m>n&& m!=0))
 {
 printf("! Operator : Both conditions are true \n");
 }
 else
 {
 Printf("!Operator : Both conditions are true."\
 "But, status is inverted as false\n");
 }
}
```

Output:

&& Operator : Both conditions are true.
|| Operator : Only one condition is true.
! Operator : Both conditions are true.
But, status is inverted as false.

In this program, operators (&&, || and !) are used to perform logical operations on the given expression.

&& Operator: "if clause" becomes true only when both conditions (m>n and m~=0) is true.

Else, it becomes false.

|| Operator: "if clause" becomes true when any one of the condition (o>p || p!=20) is true. It becomes false when none of the conditions is true.

! Operator: It is used to reverse the state of the operand.

If the conditions (m>n && m!=0) is true, true (1) is returned. This value is inverted by the "!" operator.

Q.31. Explain Bitwise Operators with the program.

Ans. Bitwise Operators:

- These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bitwise operators work on these bits.
- Bitwise operators in C language are & (Bitwise AND), | (Bitwise OR), ~ (Bitwise NOT), ^ (XOR), << (left shift) and >> (right shift).

The truth table for bitwise operations & bitwise operators:

x	y	x|y	x&y	x^y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Bitwise operators and their names in C language are as follows:

(i) & – Bitwise AND
(ii) | – Bitwise OR
(iii) ~ – Bitwise NOT
(iv) ^ – XOR
(v) << – Left Shift
(vi) >> – Right Shift

Example:

Consider x=40 and y=80. Binary form of these values are given below.

x = 00101000
y = 01010000

All bit wise operations for x and y are given below.

(i) x&y = 00000000 (binary) = 0 (decimal)

(ii) x|y = 01111000 (binary) = 120 (decimal)

(iii) ~x = 11111111111

```
{
    int m=5, n= 4;
    (m>n)? printf("m is greater than n that is
                                    %d>%d", m, n)
        : printf("n is greater than m that is
                                    %d> %d", n, m);
    return 0;
}
```

Output:
m is greater than n that is 5>4.

Q.33. Explain with suitable example increment and decrement operators.

Ans. increment and Decrement Operators:

Increment operators are used to increasing the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

Syntax:

Increment operator: ++var_name;(or) var_name++;

Decrement operator:– –var_name; (or) var_name– –;

Example:

Increment operator: + +i; i+ +

Decrement operator: – – i; i – –

Example:

/* Program for Increment Operators in C*/

In this program, the value of "i" is incremented one by one from 1 up to 9 using "i++" operator and the output is displayed as "1 2 3 4 5 6 7 8 9".

C

```
/*Example for increment operator*/
#include <stdio.h>
int main( )
{
    int i = 1;
    while (i < 10)
    {
    printf("%d", i);
    i++
    }
}
```

Output:
1 2 3 4 5 6 7 8 9

Example:

/* Program for the Decrement Operators in C*/

In this program, value of "1" is decremented one by one from 20 up to 11 using "i– –" operator is displayed as "20 19 18 17 16 15 14 13 12 11".

```
/*Example for decrement operator*/
int main( )
{
    int i = 20;
    while (i < 10)
    {
    printf("%d", i);
    i – –
    }
}
```

Output:
20 19 18 17 16 15 14 13 12 11

Q.34. Difference between Pre/Post increment and decrement Operators in C with example.

Ans. The difference between pre/post-increment and decrement operators in C programming language is as follows:

Operator	Operator/ Description
The increment operator (++i)	value of i is incremented before assigning it to the variable i.
Post increment operator (i++)	value of i is increment after assigning it to the variable i.
Pre decrement operator (– – i)	value of i is decremented before assigning it to the variable i.
Post decrement operator (i – –)	value of i is decremented after assigning it to variable i.

Example:

/* Program for Pre-Increment Operators in C*/

```
#include <stdio.h>
int main( )
{
    int i = 10
    while (++i < 5)
    {
    printf("%d", i);
}
    return 0;
}
```

Output:
1 2 3 4

- **Step 1:** In above program, value of "i" is incremented from 0 to 1 using pre-increment operator.
- **Step 2:** This incremented value "1" is compared with 5 in while expression.
- **Step 3:** Then, this incremented value "1" is assigned to the variable "i".
- Above 3 steps are continued until while expression becomes false and output is displayed as "1 2 3 4".

Example:

/*Program for Post-Increment Operators in C*/

```
#include <stdio.h>
int main( )
{
    int i = 1;
    while (i++ < 5)
    {
    printf("%d", i);
    }
    return 0;
}
```

Output:
1 2 3 4 5

- **Step 1:** In this program, value of i "0" is compared with 5 in while expression.
- **Step 2:** Then, value of "i" is incremented from 0 to 1 using post-increment operator.
- **Step 3:** Then, this incremented value "i" is assigned to the variable "i".
- Above 3 steps are continued until while expression becomes false and output is displayed as "1 2 3 4 5".

Example:

/*Program For Pre-Decrement Operators in C*/

```
#include <stdio.h>
int main( )
{
 int i = 10;
 while (--i < 5)
{
 printf("%d", i);
}
 return 0;
}
```

Output:
9 8 7 6

- **Step 1:** In above program, value of "I" is decremented from 10 to 9 using pre-decrement operator.
- **Step 2:** This decremented value "9" is compared with 5 in while expression.
- **Step 3:** Then, this decremented value "9" is assigned to the variable "I".
- Above 3 steps are continued until while expression becomes false and the output is displayed as "9, 8, 7, 6".

Example:

/*Program for Post-Decrement Operators in C*/

C

```
#include <stdio.h>
int main( )
{
    int i = 10;
    while (i-->5)
    {
    printf("%d", i);
    }
    return 0;
}
```

Output:
9 8 7 6 5

- **Step 1:** In this program, value of i "10" is compared with 5 in while expression.
- **Step 2:** Then, value of "i" is decremented from 10 to 9 using post-decrement operator.
- **Step 3:** Then, this decremented value "9" is assigned to the variable "i".
- Above 3 steps are continued until while expression becomes false and output is displayed as "9 8 7 6 5".

Q.35. What are Special Operators in C? Explain.

Ans. Special Operators:

The special operators are used for special functions in C programs. Some of the special operators available in C language are as follows:

(i) Ampersand(&):

Ampersand (&) also referred to as the address of the operator usually precedes the identifier name, which indicates the memory allocation (address) of the identifier.

(ii) Comma(,):

Comma (,) operator is used to link the related expressions together. Comma used expressions are linked from left to right and the value of the rightmost expression is the value of the combined expression. The comma operator has the lowest precedence of all operators.

Example:

Sum = (x = 12, y = 8, x + y);

The result will be sum = 20.

The comma operator is also used to separate variables during declaration.

Example:

int a, b, c;

(iii) Asterick (*):

Asterick(*) also referred to as an indirection operator usually precedes the identifier name, which identifies the creation of the pointer operator. It is also used as a unary operator.

(iv) Ellipsis (...):

Ellipsis (...) are three successive periods with no white space in between them. It is used in function prototypes to indicate that this function can have any number of arguments with varying types.

Example:

void fun (char c, int n, float f,...)

The above declaration indicates that fun () is a function that takes at least three arguments, a char, an int and a float in the order specified, but can have any number of additional arguments of any type.

(v) Hash (#):

Hash (#) also referred to as pound sign is used to indicate preprocessor directives.

(vi) Parenthesis (()):

Parenthesis () also referred as to function call operator is used to indicate the opening and closing of function prototypes, function calls, function parameters, etc., Parenthesis is also used to group expressions, and thereby changing the order of evaluation of expressions.

(vii) Semicolon (;)

Semicolon (;) is a statement terminator. It is used to end a C statement. All valid C statements must end with a semicolon, which the C compiler interprets as the end of the statement.

Example:

c = a + b; b = 5;

Q.36. Explain Sizeof Operator with example.

Ans. Sizeof Operator:

- When we need to incorporate the Sizeof some object into an expression and also for the code to be portable across different machines the Sizeof many operators will be useful. The Sizeof operator computes the Sizeof any object at compile time. This can be used for dynamic memory allocation.
- Sizeof can be applied to any data type, including primitive types such as integer and floating-point types, pointer types, or compound datatypes such as Structure, union, etc.

- **Step 1:** In this program, value of i "10" is compared with 5 in while expression.
- **Step 2:** Then, value of "i" is decremented from 10 to 9 using post-decrement operator.
- **Step 3:** Then, this decremented value "9" is assigned to the variable "i".
- Above 3 steps are continued until while expression becomes false and output is displayed as "9 8 7 6 5".

Q.35. What are Special Operators in C? Explain.

Ans. Special Operators:

The special operators are used for special functions in C programs. Some of the special operators available in C language are as follows:

(i) Ampersand(&):

Ampersand (&) also referred to as the address of the operator usually precedes the identifier name, which indicates the memory allocation (address) of the identifier.

(ii) Comma(,):

Comma (,) operator is used to link the related expressions together. Comma used expressions are linked from left to right and the value of the rightmost expression is the value of the combined expression. The comma operator has the lowest precedence of all operators.

Example:

Sum = (x = 12, y = 8, x + y);

The result will be sum = 20.

The comma operator is also used to separate variables during declaration.

Example:

int a, b, c;

(iii) Asterick (*):

Asterick(*) also referred to as an indirection operator usually precedes the identifier name, which identifies the creation of the pointer operator. It is also used as a unary operator.

(iv) Ellipsis (...):

Ellipsis (...) are three successive periods with no white space in between them. It is used in function prototypes to indicate that this function can have any number of arguments with varying types.

Example:

void fun (char c, int n, float f,...)

The above declaration indicates that fun () is a function that takes at least three arguments, a char, an int and a float in the order specified, but can have any number of additional arguments of any type.

(v) Hash (#):

Hash (#) also referred to as pound sign is used to indicate preprocessor directives.

(vi) Parenthesis (()):

Parenthesis () also referred as to function call operator is used to indicate the opening and closing of function prototypes, function calls, function parameters, etc., Parenthesis is also used to group expressions, and thereby changing the order of evaluation of expressions.

(vii) Semicolon (;)

Semicolon (;) is a statement terminator. It is used to end a C statement. All valid C statements must end with a semicolon, which the C compiler interprets as the end of the statement.

Example:

c = a + b; b = 5;

Q.36. Explain Sizeof Operator with example.

Ans. Sizeof Operator:

- When we need to incorporate the Sizeof some object into an expression and also for the code to be portable across different machines the Sizeof many operators will be useful. The Sizeof operator computes the Sizeof any object at compile time. This can be used for dynamic memory allocation.
- Sizeof can be applied to any data type, including primitive types such as integer and floating-point types, pointer types, or compound datatypes such as Structure, union, etc.

- **Usage**

  Sizeof() operator is used differently according to the operand type.

(i) When the operand is a Data Type. When Sizeof() is used with the data types such as int, float, char... etc it simply returns the amount of memory is allocated to that data type.

Example:

```
#include <stdio.h>
int main()
{
   printf("%lu\n", sizeof(char));
   printf("%lu\n", sizeof(int));
   printf("%lu\n", sizeof(float));
   printf("%lu", sizeof(double));
   return 0;
}
```

Output:
1
4
4
8

(ii) When the operand is an expression. When sizeof() is used with the expression, it returns size of the expression.

Example:

```
#include <stdio.h>
int main()
{
   int a = 0;
   double d = 10.21;
   printf("%lu", sizeof(a + d));
   return 0;
}
```

Output:
8

C OPERATOR PRECEDENCE

Q.37. Give the precedence table (with associativity) For all Arithmetic, Logical, Relational and Bitwise operators.

Ans. C operators in order of precedence are (highest to lowest). Their associativity indicates in what order operators of equal precedence in an expression are applied. The following table lists the precedence and associativity of C operators. Operators are listed top to bottom, in descending precedence.

Precedence	Operator	Description	Associa-tivity
1	++ – –	Suffix/postfix increment and decrement.	Left-to-right
	( )	Function call.	
	[ ]	Array subscripting.	
	.	Structure and union member access.	
	→	Structure and union member access through a pointer.	
	(type){list}	Compound literal	
2	++ – –	Prefix increment and decrement.	Right-to-left
	+ –	Unary plus and minus.	
	! ~	Logical NOT and Bitwise NOT.	
	(type)	Cast	
	*	Indirection (dereference)	
	&	Addressof	
	sizeof	Sizeof	
	_Alignof	Alignment requirement	
3	* / %	Multiplication, division, and remainder.	Left-to-right
4	+ –	Addition and subtraction.	

5	<< >>	Bitwise left shift and right shift.	
6	< <=	For relational operators < and ≤ respectively.	
	>>=	For relational operators > and ≥ respectively.	
7	== !=	For relational = and ≠ respectively.	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional.	Right-to-left
14	=	Simple assignment.	
	+= –=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder.	
	<<=, >>=	Assignment by bitwise left shift and right shift.	
	&=, ^=, |=	Assignment by bitwise AND, XOR, and OR.	
15	,	Comma	Left-to-right

- The operand of prefix ++ and – – can't be a typecast. This rule grammatically forbids some expressions that would be semantically invalid anyway. Some compilers ignore this rule and detect the invalidity semantically. The operand of sizeof can't be a typecast: the expression sizeof (int) * p is unambiguously interpreted as (sizeof(int)) * p, but not sizeof((int)*p).
- The expression in the middle of the conditional operator (between ? and :) is parsed as if parenthesized: its precedence relative to ?: is ignored.
- Assignment operators' left operands must be unary (level-2 non-cast) expressions. This rule grammatically forbids some expressions that would be semantically invalid anyway. Many compilers ignore this rule and detect the invalidity semantically.
- For example, e = a < d ? a++ : a = d is an expression that cannot be parsed because of this rule. However, many compilers ignore this rule and parse it as e = (((a < d) ? (a++) : a) = d), and then give an error because it is semantically invalid.

FILE AND TYPES OF I/O

Q.38. What is a File?

Ans. File : A file is a set of records that can be accessed through the set of library functions.

Different operations can be carried out on a file such as:

(i) Creation of a new file.
(ii) Opening an Existing file.
(iii) Reading from file.
(iv) Writing to a file.
(v) Moving to a specific location in a file.
(vi) Closing a file etc.

Q.39. Explain types of I/O available in C.

Ans. Types of Input/Output :

There are numerous library functions available for I/O. These can be classified into three broad categories :

(i) Console I/O functions :

Functions to receive input from the keyboard and write output to VDU (Visual Display Unit). It can be classified as formatted and unformatted.

(ii) **File I/O functions :**
Functions to perform I/O operations on a floppy disk or hard disk. They can be classified as high-level file I/O and low-level file I/O

(iii) **Port I/O functions :**
Functions to perform I/O operations in various ports.

CONSOLE I/O FUNCTIONS

Q.40.What do you mean by Console I/O functions?

Ans. Console I/O Functions :

- Functions used to receive input from a keyboard and write output to a visual display unit are called console I/O functions.
- The screen and keyboard together are called a console.
- Console I/O functions can be further classified into two categories-formatted and unformatted console I/O functions.
- The basic difference between them is that the formatted functions allow the input read from the keyboard or the output displayed on the visual display unit to be formatted as per our requirements.

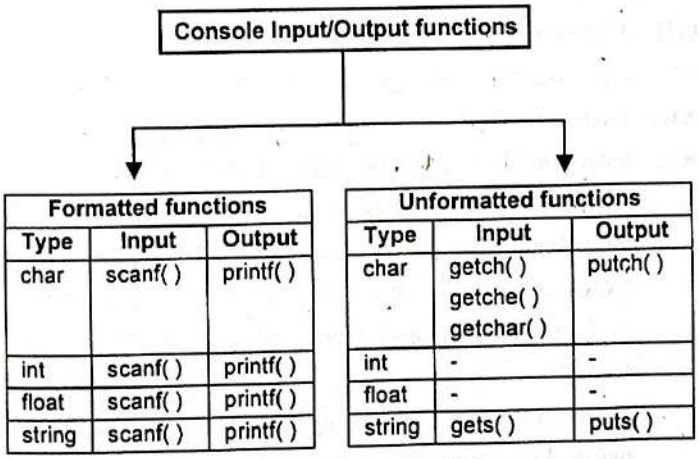


Fig. Classification of console I/O Functions

Q.41.Explain various formatted console I/O function in C.

Ans.

- printf() and scanf() functions are inbuilt library functions in the C programming language which are available in the C library by default. These functions are declared and related macros are defined in "stdio.h" which is a header file in C language.
- We have to include "stdio.h" file as shown in the below C program to make use of these printf() and scanf() library functions in C language.

(i) **Printf() Function in C Language:**

- In the C programming language, printf() function is used to print the ("character, string, float, integer, octal and hexadecimal values") onto the output screen.
- We use printf() function with %d format specifier to display the value of an integer variable.
- Similarly %c is used to display character, %f for float variable, %s for string variable, %lf for double and %x for hexadecimal variable.
- To generate a newline,we use "\n" in C printf() statement.

Note:

- C language is case sensitive. For example, printf() and scanf() are different from Printf() and Scanf(). All characters in printf() and scanf() functions must be in lower case.

Example:

```
/*Program for C Printf() Function*/
#include <stdio.h>
int main()
{
  char ch = 'A';
  char str[20] = "C programming";
  float flt = 10.234;
  int no = 150;
  double dbl = 20.123456;
  printf("Character is %c \n", ch);
  printf("String is %s \n" , str);
  printf("Float value is %f \n", flt);
  printf("Integer value is %d\n" , no);
  printf("Double value is %lf \n", dbl);
  printf("Octal value is %o \n", no);
  printf("Hexadecimal value is %x \n", no);
  return 0;
}
```

Output:
Character is A
String is C programming
Float value is 10.234000
Integer value is 150
Double value is 20.123456
Octal value is 226
Hexadecimal value is 96

(ii) Scanf Function in C Language:

- In the C programming language, scanf() function is used to read character, string, numeric data from keyboard.
- Consider below example program where user enters a character. This value is assigned to the variable "ch" and then displayed.
- Then, the user enters a string and this value is assigned to the variable "str" and then displayed.

Example:

/*Program for C Printf and Scanf Functions in C Programming Language*/

```
#include <stdio.h>
int main()
{
 char ch;
 char str[100];
 printf("Enter any character \n");
 scanf("%c", &ch);
 printf("Entered character is %c \n", ch);
 printf("Enter any string
                    (upto 100 character ) \n");
scanf("%s", &str);
printf("Entered string is %s \n", str);
}
```

Output:
Enter any character
a
Entered character is a
Enter any string (upto 100 character)
hai
Entered string is hai

Following are the tables of format specifier and escape sequence that can be needed for formatted I/O.

Format Specifier :

Code	Meaning
%c	Read a single character.
%d	Read a decimal integer.
%e	Read a floating point value.
%f	Read a floating point value.
%g	Rread a floating point value.
%h	Read a short integer.
%i	Read a decimal, hexadecimal or octal integer.
%o	Read an octal integer.
%s	Read a string.
%u	Read an unsigned decimal integer.
%x	Read a hexadecimal integer.
%[ ]	Read a string of word(s).

Escape Sequence :

Esc.Seq.	Purpose	Esc.Seq.	Purpose
\n	New line	\t	Tab
\b	Backspace	\r	Carriage return
\f	Form feed	\a	Alert
\'	Single quote	\"	Double quote

Q.42. Explain unformatted console I/O in C.

OR Explain,

(i) getch( ). (ii) getche( ). (iii) getchar( ).

Ans. Unformatted console I/O:

- Console I/O can be categorized in two types formatted and unformatted.
- The basic difference is that the formatted I/O functions allow the input read from the keyboard and displayed on-screen formatted as per our requirements.
- Unformatted I/O does not allow the output to be printed according to user requirement.
- The various unformatted I/O functions are

(i) getch( ) :

getch( ) returns the single characters as soon as it is typed. Unlike scanf( ) it does not wait for enter key to be Pressed.

(ii) getche( ) :

- getche( ) is similar to getch( ), the only addition is echo.

- getche( ) also returns the single character entered by user without waiting for the enter key.
- But in addition to this, it also displays a screen what letter was typed.
- Character 'e' in getche( ) means it echoes the character typed on screen.

(iii) **getchar( ) :**

- getchar( ) is similar to getche( ). It reads a single character and echos it on the screen.
- The only difference between getche( ) and getchar( ) is that the getchar( ) needs enter key to be pressed after entering a character.

(iv) **putch( ) and putchar( ) :**

putchar( ) and putch( ) outputs one character at a time.

Syntax :

The syntax for these functions are as follows.

```
putch(char);
putchar(char);
```

Example : putch ('z');
putchar(65);

(v) **gets( ):**

- gets( ) like scanf( ) receives a string from key board.
- The main disadvantage of scanf( ) is that it cannot accept multiple strings at a time.
- As soon as white space occurs scanf( ) terminate reading.
- To overcome the following disadvantage we use gets( ) function.

Syntax:

```
gets (char*[j]);
```

Example:

```
main ( )
{
char name [20];
printf(" enter name");
gets(name);
}
```

This code shares the name entered into array name.

(vi) **puts( ):**

- Puts works opposite to gets( ).
- It is similar to printf( ).
- The puts( ) function can output only one string at a time.
- puts( ) can print a multiword string but cannot print multiple strings.

Q.43. Difference between:

(i) scanf() and gets() (ii) priftf() and puts()

Ans. (i) scanf() and gets()

Sr. No.	Scanf()	gets()
(i)	It is used to read all types of data.	It is used to read-only string data.
(ii)	It is terminated by using white space.	It is terminated by entering the key or at end of the line.
(iii)	It cannot read the while space between two words of a string.	It is used to read complete strings with white space.
(iv)	It requires a format specifier to read formatted data.	It can read a single string at a time, it does not require a format specifier.
(v)	**Syntax:** scanf("list of format specifier", list of addresses of variable);	**Syntax:** gets(String_variable)
	Example: int x, y; scanff("%d%d"), &x, &y);	**Example:** char ch[10]; gets(ch);

(ii) priftf() and puts()

Sr. No.	Printf()	puts()
(i)	It is used to display all types of data and messages.	It is used display only string data and messages.
(ii)	It requires a format specifier to display formatted data	It does not require a format specifier to display string.

(iii)	It can display multiple data at a time by multiple format specifiers in one printf().	It is used to display only one string at a time.
(iv)	**Syntax:** printf("list of format specifier or message", list of variables);	**Syntax:** puts(variable);
	Example: int x, y; printf("%d%d");	**Example:** char ch[ ] ="Hello"; puts(ch);

Q.44.Explain various File operations in C language.

OR How files are handled in C programming? Explain various file handling operations.

Ans. File operations in C Language:

Different operations can be carried out on a file. These are :

(a) Creation of a new file.

(b) Opening an existing file.

(c) Reading from a file.

(d) Writing to a file.

(e) Moving to a specific location in a file (seeking).

(f) Closing a file.

Example:

```
/* Display contents of a file on screen. */
# include "stdio.h"
main( )
{
    FILE *fp ;
    char ch ;
    fp = fopen ("PR1. C", "r");
    while (1)
    {
        ch = fgetc(fp) ;
        if (ch == EOF)
    break ;
        printf("%c",ch);
    }
    fclose (fp);
}
```

On execution of this program it displays the contents of the file 'PR1.C' on the screen.

The file I/O functions in the stdio library are:

Function Name	Operations
fopen( )	Opens a text file.
fclose( )	Closes a text file.
feof( )	Detects end–of–file marker in a file.
fscanf( )	Reads formatted input from a file.
fprintf( )	Prints formatted output to a file.
fgets( )	Reads a string from a file.
fputs( )	Prints a string to a file.
fgetc( )	Reads a character from a file.
fputc( )	Prints a character to a file.

Q.45. What is fprintf( ) and fscanf( ) function?

OR List various types of I/O functions.

Ans.

- In C, files can be created to store the formatted data and it can be read using the functions fprintf( ) and fscanf( ).

(i) **fprintf( ):**

The fprintf( ) function is similar to printf( ), fprintf( ) writes data to a file whereas printf( ) writes on the screen.

Syntax:

fprintf(fptr, "format string", list of variables);

Here, fptr is a file pointer, format string specifies the data type of the variables and list of variables specifies the variables to be written to a file.

(ii) **fscanf( ):**

fscanf( ) used to returns the number of items that are successfully read like scanf.

Syntax:

fscanf(fptr, "format string", list of variables);

Here, fptr is a file pointer, format string specifies the data type of the variables to be read and list of variables specifies a list of address where the variables are to be stored after reading from file.

- In case the end-of-file is encountered during the read operation, fscanf( ) returns the value EOF.
- The fscanf( ) function, like scanf( ), assumes the field separator to be any white-space i.e., a blank character, a tab character or a new line.

FILE OPENING MODES

Q.46. Explain different File Opening Modes.

OR Write different File Opening Modes.

OR List and explain various modes in which a file can be opened using fopen( ) function

Ans. fopen( ):

- A data file must be opened before it can be created.
- This associates the file name with the buffer area (i.e. with the stream). It also specifies how the data file will be utilized, i.e., as a read-only file or a read-write file, in which both operations is permitted.

  Syntax :

  FILE *fp;

  fp = fopen ("filename", "mode");
- The first statement declares the variable fp as a "pointer to the data type FILE".
- The second statement opens the file named filename and assigns identification to the FILE type pointer fp.
- This pointer, which contains all the information about the file, is subsequently used as a communication link between the system and the program.
- The second statement also specifies the purpose of opening this file: The mode specifies whether the file is open for read, write or append. If fopen( ) fails to open a file, it returns NULL to the pointer.

  Example : FILE *fp1,*fp2;

  fp1 =fopen("abc.txt","r");

  fp2 =fopen("xyz.txt","w");

  The above statement will open a file abc.txt in read mode and xyz.txt in write mode. And also it returns the base address of the buffer where "abc.txt" and "xyz.txt" files reside, to the FILE pointer variable fp1 and fp2 respectively.

In C, a file can be open in different modes as mentioned below :

Mode	Description
r	Open a text file for reading.
w	Create a text file for writing. If the file exists, it is overwritten.
a	Open a text file in append mode. Text is added at the end of the file.
rb	Open a binary file for reading.
wb	Create a binary file for writing. If the file exists, it is overwritten.
ab	Open a binary file in append mode. Data is added at the end of the file.
r+	Open a text file for reading and writing.
w+	Create a text file for reading and writing. If the file exists, it is overwritten.
a+	Open a text file for reading and writing at the end.
r+b or rb+	Open binary file for reading and writing.
w+b or wb+	Create a binary file for reading and writing. If the file exists, it is overwritten.
a+b or ab+	Open a text file for reading and writing at the end

File opening modes for Low-level I/O:

Mode	Description
O_APPEND	Opens a file for appending.
O_CREATE	Creates a new file for writing (does not effect if file already exists).
O_RDONLY	Creates a new file for reading only.
O_RDWR	Creates a file for both reading and writing.
O_WRONLY	Creates a file for writing only.
O_BINARY	Creates a file in binary mode.
O_TEXT	Creates a file in text mode.

Q.47. Explain the syntax of fclose( ).

OR What is the syntax of fclose( )? Explain.

OR Explain in brief, the following the handling function :

fclose( )

Ans. fclose( ):

- The fclose( ) function is used for closing opened files.
- The only argument it accepts is the file pointer.

  Syntax : fclose(FILE*);

  Example : fclose(Fp);

  Where, Fp is file pointers which points to already opened file.
- If a program terminates, it automatically closes all opened files. But it is a good programming habit to close any file once it is no longer needed.
- This helps in better utilization of system resources and is very useful when working on numerous files simultaneously.

Q.48. Explain fgetc( ) and fputc( ) file handling function.

Ans.

(i) fgetc( ) :

- fgetc( ) is a character-oriented function. This function reads the single character from a given file and increments the file pointer position.
- On success, it returns the character after converting it to an int without sign extension. On error, it returns EOF .
- If successful, it returns the next requested object from the stream.
- Character values are returned as an unsigned char converted to an int. If the stream is at the end-of-file or a read error occurs, then it returns EOF.

Example :

```
/* program to show the use of fgetc( ) function*/
include<stdio.h>
main()
{
int c;          /*Character read from the file.*/
FILE *ptr;      /* Pointer to the file. FILE is a
                   structure defined in <stdio.h>*/
ptr = fopen("abc.txt","r"); /* Read one character
                   at a time checking for the End of File
                   EOF is defined in <stdio.h> as -1*/
while((c = fgetc(ptr)) != EOF)
{
printf("%c",c);         /* O/P the character to the
                                          screen */
}
fclose(ptr);            /*Close the file.*/
}
fputc( );
}
```

(ii) fputc( ) :

- This function writes a character to a specified file at the current file position and then increment the file position pointer.
- The putc( ) macro acts essentially identically to fputc(), but is a macro that expands in-line. It may evaluate stream more than once, so arguments given to putc() should not be expressions with potential side effects.

  Syntax : int fputc(int c, FILE *fp);
- The fputc function returns an integer representing the character written. If a write error occurs the error indicator is set and fputc returns EOF.

Example :

```
/*program to understand the use of fputc( )
function*/
#include<stdio.h>
main()
{
FILE *p;
char ch;
if((p==fopen("myfile.txt","w"))==NULL)
{
printf("this file does not exist\n");
exit( );
}
else
{
printf("enter the text");
while((ch==getchar())!=EOF)
fputc(ch,p);
}
fclose(p);
}
```

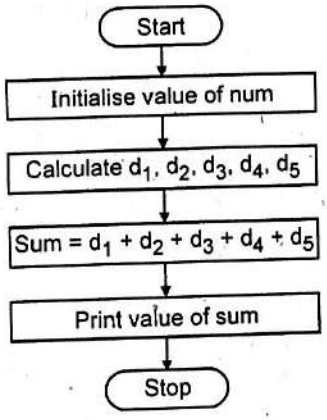
Q.49. Write a program to find sum of first and last digit of a given 3 digit number.

Ans.
```
#include<stdio.h>
#include<conio.h>
main()
{
int num;
clrscr();
printf("Enter three digit Number:");
                /* suppose entered number is 123*/
scanf("%d",&num);
printf("The sum of first and last digit is %
d",num%10+num/100);
getch();
}
```

Output :
Enter three digit Number: 123
The sum of first and last digit is 4

Q.50. Write a program to find sum of 5 digit number. Draw its flowchart.

Ans.
```
#include<stdio.h>
#include<conio.h>
main()
{
int num, d1, d2, d3, d4, d5, sum;
clrscr( );
printf("Enter five
digit number");
scanf("%d",
&num);
d1 = num%10 ;
num = num/10;
d2 = num%10;
num = num / 10;
d3 = num%10;
num =num/10;
d4 =num%10;
num=num/10;
d5=num%10;
num=num/10;
sum=d1 + d2 + d3 + d4 + d5;
printf("The sum of digits of given number is
%d",sum);
getch();
}
```



Output :
Enter five digit number 12345
The sum of digits of given number is 15

Q.51. Write a program to find sum of digits of a given 3 digit number.

Ans.
```
#include<stdio.h>
#include<conio.h>
main( )
{
int num,d1,d2,d3,sum;
clrscr();
printf("Enter three digit value:");
                /* suppose entered number is 123*/
scanf("%d",&num);
d1=num%10;                  /*d1=123%10=3*/
num= num/10;            /* num=123/10=12 */
d2=num%10;                  /* d2=12%10=2*/
num= num/10;            /* num=12/10=1*/
d3=num%10;                  /* d3=1%10=1*/
sum=d1+d2+d3;           /* sum= 3+ 2+1=6*/
printf("The sum of digits of given number is %d",
                                        sum);
getch();
}
```

Output :
Enter three digit value: 123
The sum of digits of given number is 6

Q.52. Write a program to find reverse of digits of a given 3 digit number.

Ans.
```
#include<stdio.h>
#include<conio.h>
main( )
{
int num,d1,d2,d3;
int d;
clrscr( );
printf("Enter three digit Number:");
                /* suppose entered number is 123*/
scanf("%d",&num);
d1=num%10;                  /*d1=123%10=3*/
num= num/10;            /* num=123/10=12 */
d2=num%10;                  /* d2=12%10=2*/
num= num/10;            /* num=12/10=1*/
d3=num%10;                  /* d3=1%10=1*/
d=d1*100+d2*10+d3;      /* d= 3*100+2*10+1
                            =300+20+1=321*/
```

```
printf("The reverse number is %d",d);
getch( );
}
```

Output :
Enter three digit Number:123
The reverse number is 321

Q.53. If a 7 digit number is input through the keyboard, write a program to reverse the number.

Ans.
```
#include<stdio.h>
#include<conio.h>
main( )
{
    long int num,d1,d2,d3,d4,d5,d6,d7;
    long int revnum;
    clrscr();
    printf("Enter seven digit number:");
    scanf("%ld",&num);
    d1=num%10;
    num= num/10;
    d2=num%10;
    num= num/10;
    d3=num%10;
    num= num/10;
    d4=num%10;
    num= num/10;
    d5=num%10;
    num= num/10;
    d6=num%10;
    num= num/10;
    d7=num%10;
    revnum=d1*1000000+d2*100000+d3*10000
                +d4*1000+d5*100+d6*10+d7;
    printf("The reverse number is %ld",revnum);
    getch();
}
```

Output :
Enter seven digit number: 1234123
The reverse number is 3214321

PROGRAM FOR PRACTICE:

Q.1. Write a program to swap two numbers.

Q.2. Write a C program to illustrate the difference between = and = = operators.

Q.3. Find the output of the following program.

Q.4. Find the output of following programs :

(i)
```
void main( )
{
int a ;
for(a=0;a<5;a++)
    {
    printf("hello ! ! \ n") ;
  a=a –1;
    }
}
```

(ii)
```
void main( )
{
    int i, j , k ;
    i = 2 ;
    i + + ;
    printf(" i = %d " , i);
    – – i ;
    printf(" \n i = %d", i);
    j = 2
    k – – i + j ;
    printf(" \n k = %d", k) ;
    k – – – ;
    printf("\n k = %d", k) ;
}
```

(iii)
```
void main( )
{
    if(1)
  printf (" C\n") ;
    else
    printf("C + + \ n") ;
    printf("oracle") ;
}
```

POINTS TO REMEMBER:

(1) A computer is a programmable electronic machine that accept instructions and data through input devices.

(2) The various components of computer such as input devices, output devices memory and central processing unit.

(3) C supports the various data types to store the values in a variable.

(4) Data types tells the compiler exact memory space requirement of the variable.

(5) Basic data types in C are integer, float (real) and character.

(6) Do not use the underscore as the first character of identifiers because many of the identifiers in the system library start with underscore.

(7) C supports various kinds of operators which includes arithmetic, relational, logical, assignment, increment and decrement, conditional and bitwise operators.

(8) It is an error if any space between the two symbols of the operators ==, !=, <= and >= is given.

(9) Input values are taken by using the scanf( ) function.

(10) The printf( ) function is used to print the messages and numerical values of the variables.

(11) All C programs are made up of one or more functions and they have a specific function called main( ).

(12) Comment can be inserted anywhere in a C program by using a pair as follows:
/* comment */

(13) For getting the output of a program, first compile and then execute the program using appropriate compiler.

UNIT - II
CONDITIONAL BRANCHING AND LOOPS, ARRAYS, BASIC ALGORITHM

SYLLABUS

(a) **Conditional Branching and Loops:** Writing and evaluation of conditionals and consequent branching Iteration and loops.

(b) **Arrays:** Arrays (1-D, 2-D), Character arrays and Strings.

(c) **Basic Algorithms:** Searching, Basic Sorting Algorithms (Bubble, Insertion and Selection). Finding roots of equations, notion of order of complexity through example programs (no formal definition required).

INTRODUCTION

C language possesses decision-making capabilities and supports the statements known as control or decision making statements. The purpose of control statements can be of three types: Selection, Repetition and Sequence. Selection is the type of control that is used to select the condition from the group of conditions. The selection tool is known as decision making. If else and switch case loops are used to select a condition at a time from the multiple choices and execute it.

Sometimes, the user may wish to repeat certain statements or instructions. With the help of initial and final condition, user can repeat the statements. Repetition and sequencing start with the initial condition and terminate when the final condition occurs. The controls used for repetition and sequencing are for and while loops. The 'C' statement is a single line instruction terminated by a semicolon. On the other hand, the loop can be a set of instructions repeatedly called.

C language provides a facility that enables the user to design a set of similar data types, called an array. Array is a collection of data that belongs to same data type. Array elements can be accessed by their position in the array called an index which starts with zero. An array is a collection of similar elements that share a common name.

CONDITIONAL AND BRANCHING LOOPS

Q.1. What is a Control Statement? What are the different types of Control Statements?

Ans. Control Statement:

The control statement deals with the various methods by that C can control the flow of logic in a program. Control statements can be classified as un-conditional and conditional branch statements and loop or iterative statements. The Branch type includes:

(i) **Un-conditional:**
- (a) goto
- (b) break
- (c) return
- (d) continue

(ii) **Conditional:**
- (a) if
- (b) if-else
- (c) Nested if
- (d) if-else-if ladder
- (e) switch case statement

(iii) **Loop Operations:**
- (a) for loop
- (b) while loop
- (c) do-while loop

Q.2. Explain Conditional Statement in detail with example.

Ans. Conditional Statement:

Sometimes we want a program to select an action

from two or more alternatives. This requires a deviation from the basic sequential order of statement execution. Such programs must contain two or more statements that might be executed but have some way to select only one of the listed options each time the program runs. This is known as conditional execution or conditional statement.

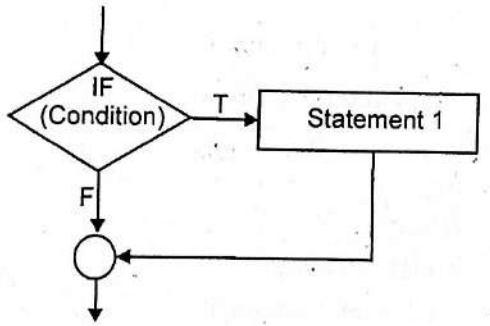
(i) If statement:

A statement or set of statements can be conditionally executed using if statement. Here, the logical condition is tested which, may be either true or false. If the logical test is true (non zero value) the statement that immediately follows is executed. If the logical condition is false the control transfers to the next executable statement.

Syntax:

```
if (condition)
    statement_to_execute_if_condition_is_true;
    or
if (condition)
{
    statement 1;
    statement 2;
    ........;
}
```

Flowchart :



Example:

```
main( )
{
int a,b,c,d;
float ratio;
printf("Enter four integer values\n);
scanf("%d%d%d", &a, &b, &c, &d);
if (c-d!=0)
{
  ratio = (float) (a + b)/ (float) ((c-d);
  print f("ratio = %f \n". ratio);
  }
}
```

(ii) If-else statement:

The if statement is used to execute only one action. If there are two statements to be executed alternatively, then an if-else statement is used. The if-else statement is a two-way branching.

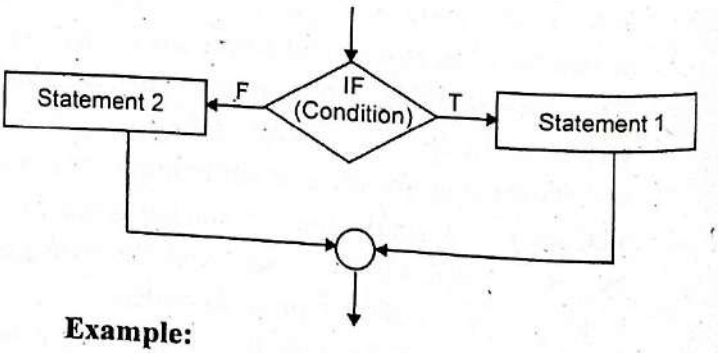
Syntax:

```
if (condition)
{
statement_to_execute_if_condition_is_true;
else
statement_to_execute_if_condition_is_false;
}
```

Where, the statement may be a single statement, a block, or nothing and the else statement is optional. The conditional statement produces a scalar result, i.e., an integer, character, or floating-point type.

It is important to remember that an if statement in C can execute only one statement on each branch (T or F). If we desire that multiple statements be executed on a branch, we must block them inside of a {and} pair to make them a single compound statement. Thus, the C code for the flowchart segment above would be:

Flowchart:



Example:

```
main()
{
    int num;
    printf("Enter a number : ");
    scanf("%d",&num);
    if (num % 2 == 0)
```

```
        printf(" Even Number")
            else
        printf(" Odd Number")
    }
```

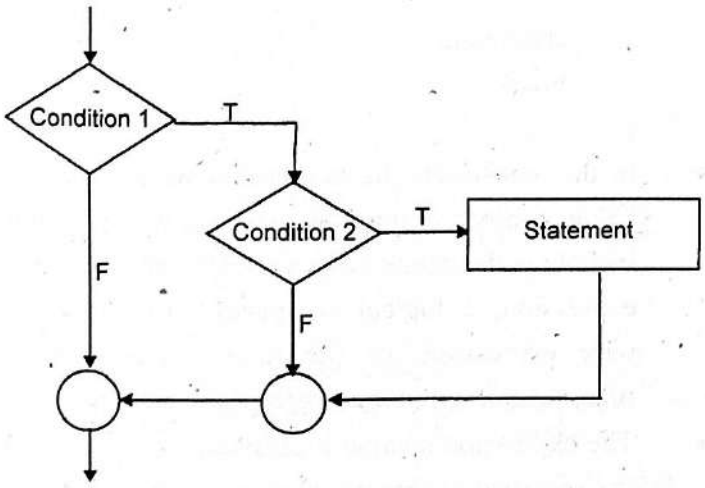
(iii) Nested if statement:

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, both C and C++ allow us to nested if statements within if statements, i.e., we can place an if statement inside another if statement.

Syntax:

```
if (condition)
{
  //Executes when condition 1 is true.
  if (condition 2)
  {
  //Execute when condition 2 is true
  }
}
```

Flowchart:



Example:

```
main()
{
      int num;
      printf(" Enter a number : ");
      scanf("%d",&num);
if( num > 0 )
{
  if( num % 2 == 0)
  printf("Even Number");
}
else
{
else
printf("Odd Number");
if( num < 0 )
printf("Negative Number");
else
print("Number is Zero")
      }
}
```

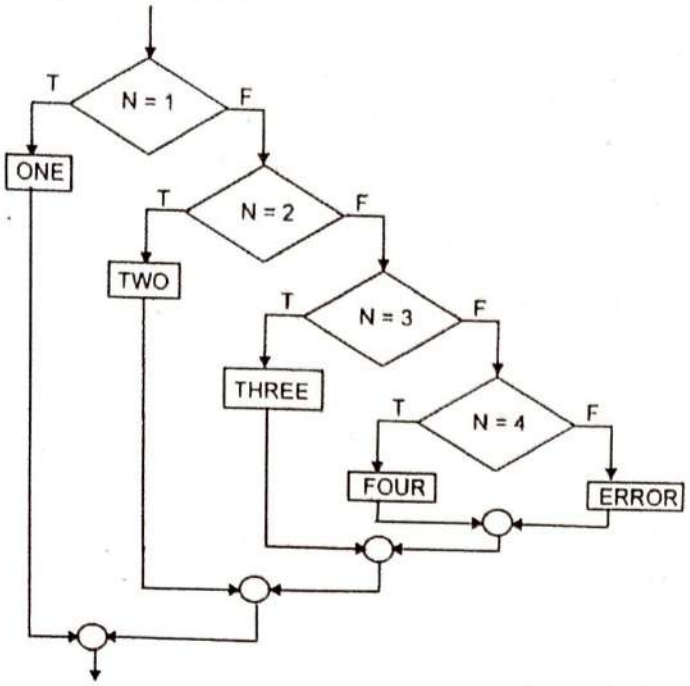
(iv) if-else-if Ladder:

When faced with a situation in which a program must select from many processing alternatives based on the value of a single variable, an analyst must expand his or her use of the basic selection structure beyond the standard two processing branches offered by the if statement to allow for multiple branches. One solution to this is to use an approach called nesting in which one (or both) branch(es) of a selection contain another selection. This approach is applied to each branch of an algorithm until enough additional branches have been created to handle each alternative.

Syntax:

```
if (expression)
      statement 1;
      else
      if(expression)
      statement 2;
..
..
else
statement 3;
```

Flow chart:



Example:

```
#include <stdio.h>
void main (void)
{
    int N;
    /* Menu Choice */
    printf ("MENU OF TERMS\n\n");
    printf ("1. Single\n");
    printf ("2. Double\n");
    printf ("3. Triple\n");
    printf ("4. Quadruple\n\n");
    printf ("Enter the number(1- 4): ");
    scanf ("%d", &N);
    if (N == 1)
    {
    printf ("one")
    else if (N == 2)
    {
        printf ("two");
        else if (N == 3)
        {
            printf ("three");
            else if (N == 4)
            printf ("four");
            {
                else
                printf ("ERROR");
                }
            }
        }
    }
}
```

(v) The Switch Statement:

The switch-case statement is used when an expression's value is to be checked against several values. If a match takes place, the appropriate action is taken. The general form of the switch case statement is:

Syntax:

```
switch (expression)
{
    case constant1:
    statement;
    break;
    case constant2:
    statement;
    break;
    default:
    statement;
    break;
}
```

- In this construct, the expression whose value is being compared may be any valid expression, including the value of a variable, an arithmetic expression, a logical comparison rarely, a bit-wise expression, or the return value from a function call, but not a floating-point expression.
- The expression's value is checked against each of the specified cases and when a match occurs, the statements following that case are executed. When a break statement is encountered, control proceeds to the end of the switch-case statement.
- The break statements inside the switch statement are optional. If the break statement is omitted, execution will continue on into the next case statements even though a match has already taken place until either a break or the end of the switch is reached.

- The keyword case may only be constants, they cannot be expressions. They may be integers or characters, but not floating-point numbers or character strings.
- Case constants may not be repeated within a switch statement.
- The last case is a special keyword default. The default statement is executed if no matches are found. The default is optional and if it is not present, no action takes place if all matches fail.

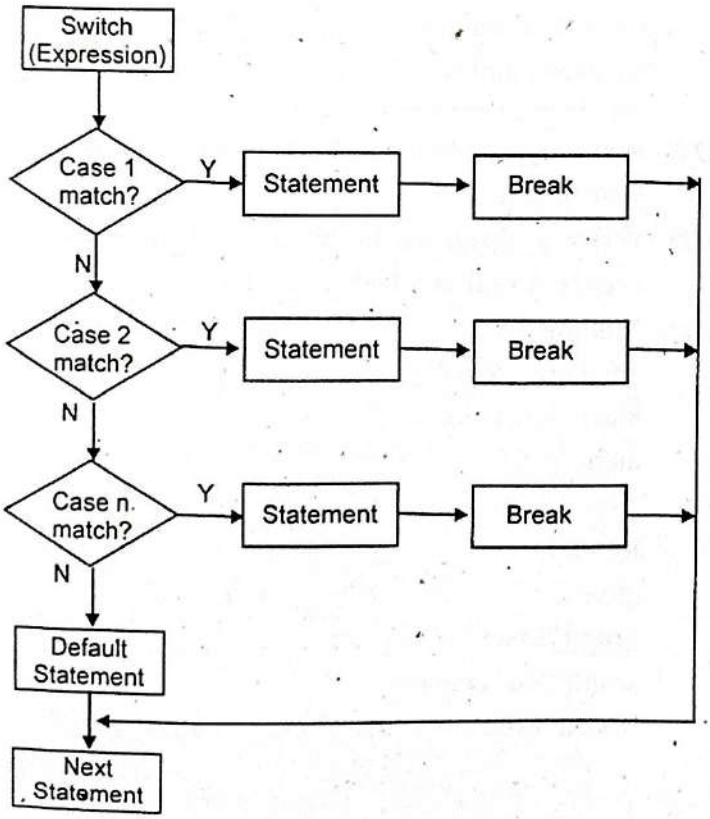
Three important things to know about the switch statement:

(i) The switch differs from if in that switch can only test for equality whereas if can evaluate any type of relational or logical expression.

(ii) No two case constants in the same switch can have identical values. But, a switch statement enclosed by an outer switch may have case constants and either same.

(iii) If character constants are used in the switch statement, they are automatically converted to integers.

Flowchart:



Example:

```
main()
{
        char gender;
        printf ("Enter Gender code:(M/F)");
        scanf ("%c", &gender);
        switch (gender)
{
        case 'M': printf (" Male");
                  break;
        case 'F': printf ("Female");
                  break;
        default : printf ("Wrong code");
}
}
```

We can also have null statements by just including a ";" or let the switch statement fall through by omitting any statements.

Q.3. Write a program to find out whether a given number is an odd number or even.

Ans.

```
#include<stdio.h>
#include<conio.h>
main( )
{
int num;
clrscr( );
printf("Enter a number to be checked for even/odd:
");
scanf("%d",&num);
if(num%2==0)
{
      printf("The entered number is EVEN\n");
}
else
{
      printf("The entered number is ODD\n");
}
getch( );
}
```

Output :

Enter a number to be checked for even/odd : 42
The entered number is EVEN

Q.4. Write a program to find out greatest number out of 3 numbers.

Ans. Method 1: Using if-else

```
#include<stdio.h>
#include<conio.h>
main( )
{
int a,b,c;
clrscr( );
printf("Enter 3  numbers");
scanf("%d %d  %d  ",&a, &b, &c);
if(a>b)
{
if(a>c)
printf("%d is greatest \n",a);
else
printf("%d is greatest \n ",c);
}
else
{
if(b>c)
printf("%d is greatest \n",b);
else
printf("%d is greatest \n ",c);
}
getch( );
}
```

Output :
Enter 3 numbers 22 10 9
22 is greatest

Method 2 : Using only if statement

```
#include<stdio.h>
#include<conio.h>
main( )
{
int a,b,c,max;
clrscr( );
printf("Enter 3  numbers  ");
scanf("%d %d  %d  ",&a, &b, &c);
max=a;
if(b>max)
max=b;
if(c>max)
max=c;
printf("Greatest Number is %d",max);
getch( );
}
```

Output:
Enter three numbers 22 10 9
Greatest number is 22

Method 3 : Using Conditional operator

```
#include<stdio.h>
#include<conio.h>
main()
{
int a,b,c,max;
clrscr();
printf("Enter three Numbers:. ");
scanf("%d %d  %d ",&a, &b, &c);
max=a;
max=b>max?b:max;
max=c>max?c:max;
printf("Greatest Number is %d",max);
getch();
}
```

Output :
Enter three numbers 22 10 9
Greatest number is 22

Q.5. Write a program to check given year is leap year or not.

OR Write a program in 'C' to test whether an accepted year is a leap year or not.

Ans. Method 1 :

```
#include<stdio.h>
#include<conio.h>
main( )
{
int year;
clrscr( );
printf("Enter the year: ");
scanf("%d",&year);
if(year%400 = =0 || (year%100 !=0 && year%4
                                    = = 0))
{
```

```
("Year %d is a leap year",year);
}
else
{
("Year %d is not a leap year",year);
}
getch( );
}
```

Output :
Enter the year 2004
Year 2004 is a leap year

Method 2 :

```
#include<stdio.h>
#include<conio.h>
main( )
{
    int year;
    clrscr();
    ("Enter the year: ");
    scanf("%d",&year);
    if(year%100 ==0)
    {
        if(year%400 ==0)
        ("Year is a leap year");
        else
        {
        ("Year  is not a leap year");
        }
    }
    else
    {
        if(year%4 ==0)
        ("Year is a leap year");
        else
        {
            ("Year  is not a leap year");
        }
    getch();
}
```

Output :
Enter the year : 2004
Year is a leap year

Q.6. Write a program to input character from the user and convert it into upper case letters.

Ans.

```
#include<stdio.h>
#include<conio.h>
main( )
{
    char ch;
    clrscr();
    printf("Enter any character");
    scanf("%c",&ch);
    if(ch>=97 && ch<=122)
    {
        ch=ch–32;
    }
    printf("The upper case letter is %c ",ch);
    getch();
}
```

Output :
Enter any character b
The upper case letter is B

Q.7. Write a program to find roots of a quadratic equation. Program must consider following cases:
(i) roots are real and equal
(ii) roots are real and unequal
(iii) roots are imaginary.

Ans.

```
#include<stdio.h>
#include<conio.h>
main( )
{
int a,b,c, term;
float root1, root2;
clrscr( );
printf("Enter the values of a,b,c:");
scanf("%d %d %d",&a,&b,&c);
term=(b*b)–(4*a*c);
root1=((–b)+sqrt(term))/(2*a);
root2=((–b)–sqrt(term))/(2*a);
if(term= =0)
    {
    printf("Roots are real and equal");
    root1=root2= –b/(2*a);
```

```
        printf("%f %f",root1,root2);
        }
    if(term>0)
        {
        printf("Roots are real and unequal");
        root1=((-b)+sqrt(term))/(2*a);
        root2=((-b)-sqrt(term))/(2*a);
        printf("%f %f",root1,root2);
        }
    if(term<0)
        {
        printf("Roots are real and imaginary");
        }
    getch();
    }
```

Output:
Enter the values of a, b, c : 2 3 4
Roots are real and unequal

Q.8. Write a program that finds a four-digit number AABB which is a perfect square. A and B represent different digits.

Ans.
```
#include<stdio.h>
#include<conio.h>
main( )
{
int num,d1,d2,d3,d4;
clrscr( );
printf("Enter four digit Number:");
scanf("%d",&num);
d1=num%10;
num= num/10;
d2=num%10;
num= num/10;
d3=num%10;
num= num/10;
d4 = num%10;
num=num/10;
if(d1==d2 && d3 == d4 && d2 ! = d3)
printf("The entered number is in the form of
                                    AABB");
else
printf("The entered number is not in the form of
                                    AABB");
getch( );
}
```

Output:
Enter four digit Number: 2222
The entered number is in the form of AABB
Enter four digit Number: 1223
The entered number is not in the form of AABB

Q.9. Write a program along with flowchart to find the value of y for the given value of x :

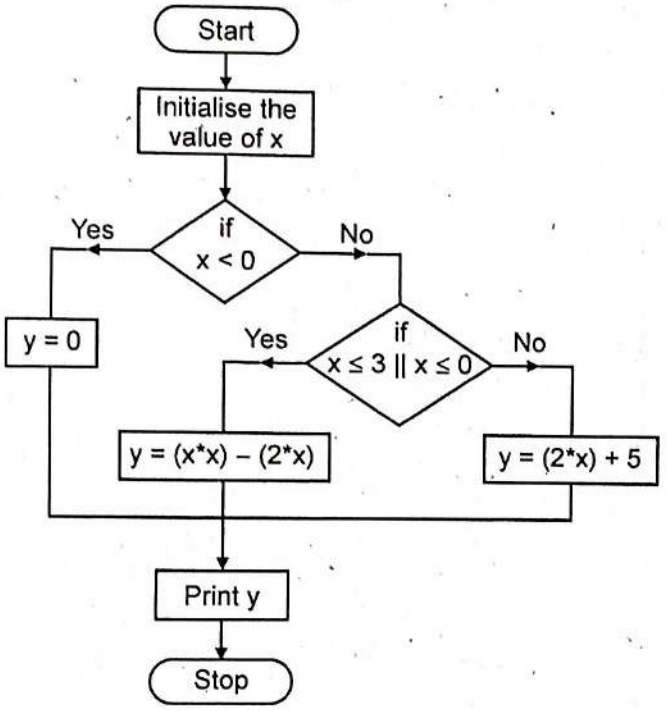
$y = x^2 - 2x$ when $0 <= x <= 3$
$= 2x + 5$ when $x > 3$
$= 0$ when $x < 0$

Ans.
```
#include<stdio.h>
#include<conio.h>
main( )
{
int x,y;
clrscr( );
printf("Enter the value for x:");
scanf("%d",&x);
printf("Calculation of value y");
if(x<0)
{
y=0;
printf("The value of y is %d\n",y);
}
else
{
if(x<=3||x<=0)
{
y=(x*x) - (2*x);
printf("The value of y is %d\n",y);
}
else
{
y=(2*x)+5;
printf("%d\n",y);
}
getch();
}
```

Output : Enter the value for x–2
Calculation of value y
The value of y is 0

Enter the value for x: 10
Calculation of value y
The value of y is 25

Flowchart :



Q.10. What is looping and iteration in C?

Ans. Looping is a powerful programming technique through which a group of statements is executed repeatedly until a certain specified condition is satisfied. Looping is also called repetition or iterative control mechanism.

C provides three types of loop control structures. They are:

(i) for statement
(ii) while statement
(iii) do-while statement

Q.11. Explain for loop with an example and flow chart.

Ans. For Loop:

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

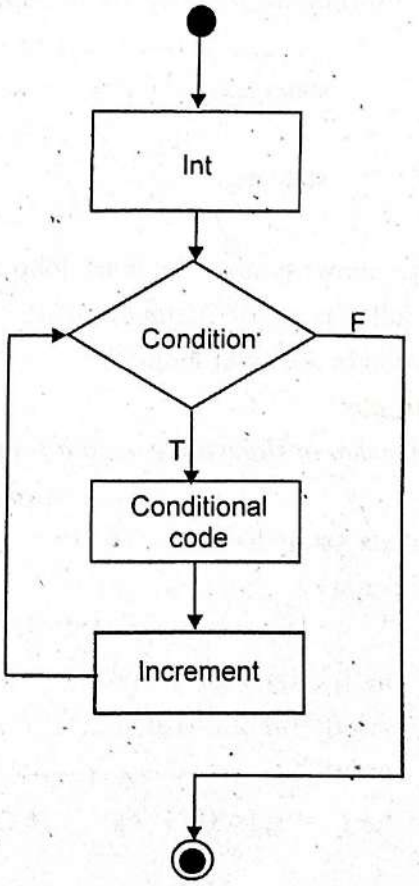
Syntax:

for (initialization; condition; increment)
statement;
{
}

The initialization is generally an assignment statement that is used to set the loop control variable.

- The condition is an expression (relational/ logical / arithmetic/bitwise ....) that determines when the loop exits.
- The increment defines how the loop control variable changes each time the loop is repeated. We must separate these three major sections by a semicolon.
- The for loop continues to execute as long as the condition is true. Once the condition becomes false, program execution resumes on the statement following the for.
- The comma (,) operator is used to extend the flexibility of the for a loop. It allows the general form to be modified as follows:

Flow chart:



Example:

/ Sum the squares of all the numbers between 1 to 5*/*

```
main()
{
int x, sum = 0;
for (x = 1; x <= 5; x ++)
{
    sum = sum + x * x;
}
    printf ("\n Sum of squares of all the
    numbers between 1 to 5 = %d ", sum);
}
```

Output:
Sum of squares of all the numbers between 1 to 5 = 55

Q.12. Explain Nested and Infinite For Loop with example.

Ans. Nested Loops: C programming language nested loops consist of one loop placed inside another loop.

```
for (initialization; condition; operation)
{
    for (initialization; condition; operation)
    {
        statement;
    }
        statement;
}
```

In the above syntax, the inner loop runs through its full range of iterations for every single iteration of the outer loop.

Example:

*/*Program to show table of first four powers of numbers 1 to 9*/.*

```
#include <stdio.h>
void main()
{
    int i, j, k, temp;
    printf("I\tI^2\tI^3\tI^4 \n");
    printf("--------------------------\n");
    for ( i = 1; i< 10; i++)        /* Outer loop */
    {
    for (j = 1; j < 5; j++)
            /* 1st level of nesting*/
    {
        temp = 1;
        for(k = 0; k < j; k ++)
        temp = temp * I;
        printf ("%d\t", temp);
    }
    printf ("\n");
  }
}
```

Output:

I	I ^ 2	I ^ 3	I ^ 4
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561

The Infinite Loop:

A loop becomes an infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the for loop is required, we can make an endless loop by leaving the conditional expression empty.

Syntax:

```
for(;  ;)
```

Example:

```
#include<stdio.h>
int main( )
{
For (; ;)
{
printf("This loop will run forever.\n");
```

```
}
return 0;
}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the for(; ;) construct to signify an infinite loop.

Q.13. What is While Loop? Explain with an example. Write a program with a nested while loop.

Ans. While Loop:

- A while statement is useful to repeat a statement execution as long as a condition remains true or an error is detected. The while statement tests the condition before executing the statement.
- The condition can be any valid C languages expression including the value of a variable, a unary or binary expression, an arithmetic expression, or the return value from a function call.

Syntax:

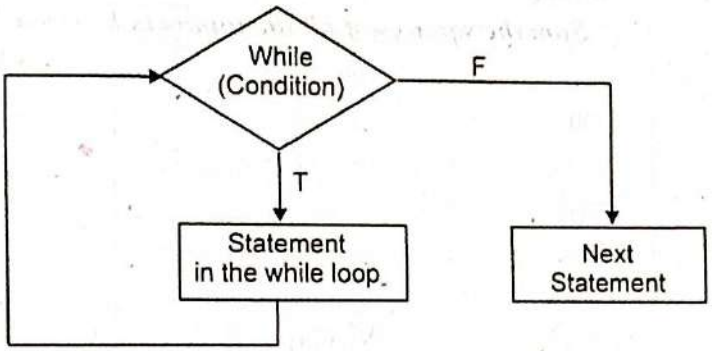
```
{
 while (expression)
      statement
 }
```

- The statement can be a simple or compound. A compound statement is a while statement appears as:

```
while (condition)
{
      statement 1;
      statement 2;
}
```

- With the if statement, it is important that no semicolon follow the closing parenthesis, otherwise the compiler will assume the loop body consists of a single null statement. This usually results in an infinite loop because the value of the condition will not change within the body of the loop.

Flow Chart:



- The while loop can accept expressions, not just conditions, the following are all legal:

```
while(x--);
while(x = x+1);
while(x += 5);
```

Using this type of expression, only when the result of x--, x=x+1, or x+=5, evaluates to 0 will the while condition fail and the loop is exited.

We can go further still and perform complete operations within the while expression:

```
while(i++ < 10);
```

The counts i up to 10.

```
while((ch = getchar()) !=`q') putchar(ch);
```

- This uses C standard library functions: getchar() to reads a character from the keyboard and putchar() to write a given char to the screen. The while loop will proceed to read from the keyboard and echo characters to the screen until a 'q' character is read.

Example:

```
main()
{
int j =1;
double degC, degF; clrscr ();
printf("\n Table of Celsius and Fahrenheit
degrees \n\n");
printf ("Celsius Degree \t Fahrenheit Degree \n")
degC = -20.0;
while (j <=6)
{
      degC = degC + 20.0;
      degF = (degC * 9.0/5.0) + 32.0;
      printf ("\n %7.2lf\t\ %7.2lf ", degC, degF);
                                        j++;
}
}
```

Output:

Table of Celsius and Fahrenheit degrees

Celsius Degree	Fahrenheit Degree
0.00	32.00
20.00	68.00
40.00	104.00
60.00	140.00
80.00	176.00
100.00	212.00

Q.14. Explain WHILE statement with proper syntax and example.

OR Write syntax used for while loop.

Ans. There are 3 types of looping statements available in the C language;

(i) While loop

(ii) Do-while loop

(iii) For loop

While Loop :

In while a looping statement allows several lines to represent until the condition is satisfied.

Syntax :

```
Initialization;
while(condition)
{
C Statement 1;
C Statement 2;
...
C Statement n;
Updation;
}
```

Where **while** is a keyword.

Example :

```
main()
{
int i;
clrscr();
i=1;
while(i<=3)
{
    printf("%d \n",i);
    i=i+1;
}
getch();
}
```

Output:

```
1
2
3
```

Q.15. Explain DO-WHILE statement with proper syntax and example.

OR Write syntax used for do-while loop.

Ans. Do WHILE loop : In DO WHILE loop first executes the statements then it checks the condition.

Syntax :

```
Initialization;
do
{
C Statement 1;
C Statement 2;
...
C Statement n;
Updation;
} while ( condition);
```

Where **do** and **while** are keywords.

Example :

```
main()
{
int i;
clrscr();
i=1;
do
{
    printf("%d \n",i);
    i=i+1;
    } while(i<1);
getch();
}
```

Output : 1

Q.16. Explain FOR statement with syntax and example.

OR Explain the role of the initialization, test and

update expression in a 'for' loop. When is each of them executed? Explain with an example.

Ans. For loop :

- In for, a looping statement allows several lines to represent until the condition is satisfied.

Syntax :

```
for(initialization ; condition ; increment/decrement)
{
C Statement 1;
C Statement 2;
...
C Statement n;
}
```

Where **for** is a keyword.

Example :

```
main( )
{
    int i;
    clrscr();
    for(i=0;i<=3;i=i+1)
    {
        printf("%d \n",i);
    }
    getch();
}
```

Output:

```
1
2
3
```

Explanation :

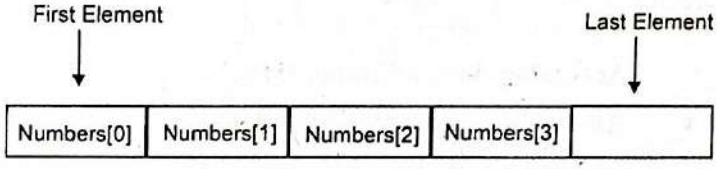
- In the above 'C' program, during the execution process the value zero is assigned to variable 'i', with the help of initialization. Then the condition gets a check.
- If the condition is true then logical statements written inside for loop are executed. In this case, when the condition becomes true it will print the value of 'i' as shown in the above program.
- Updation will work means either increment or decrement of the values according to the user's need.

C ARRAYS

Q.17. What is Array? Explain one-dimensional array with an example.

Ans. Array:

- C programming language provides a data structure called the array, which can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- 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. A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays:

- To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows:

  type arrayName [arraySize];

- This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and the type can be any valid C data type. For example, to declare a 10-element array called balance of type double, use this statement:

  double balance[10];

  Now balance is a variable array that is sufficient to hold upto 10 double numbers.

Initializing Arrays:

- We can initialize array in C either one by one or using a single statement as follows:

  double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};

- The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ].

Following is an example to assign a single element of the array:

If we omit the size of the array, an array which is just big enough to hold the initialization is created. Therefore, if we write:

double balance[ ] = {1000.0, 2.0, 3.4, 17.0, 50.0};

We will create the same array as we did in the previous example.

balance[4] = 50.0;

- The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th i.e. last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representation of the same array we discussed above:

	0	1	2	3	4
Balance	1000.0	2.0	3.0	7.0	50.0

Accessing Array Elements:

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example:

double salary = balance[9];

- The above statement will take the 10th element from the array and assign the value to the salary variable. Following is an example that will use all the above mentioned three concepts viz. declaration, assignment and accessing arrays:

Example:

```
#include <stdio.h>
int main ()
{
 int n[ 10 ];      /* n is an array of 10 integers */
 int i,j;
/* initialize elements of array n to 0 */
for ( i = 0; i < 10; i++ )
{
 n[ i ] = i + 100;
/* set element at location i to i + 100 */
}
/* output each array element's value */
for (j = 0; j < 10; j++ )
{
printf("Element[%d] = %d\n", j, n[j] );
}
return 0;
}
```

Output:
```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Q.18. Write a 'C' program to accept 5 integer numbers and display them using array.

Ans.
```
#include<stdio.h>
main( )
{
    int number[5];
    int i = 0;
    clrscr( );
    for(i = 0; i <= 4; i++)
    {
        printf("\n Enter number :");
        scanf("%d", &number[i]);
    }
    for(i = 0; i <= 4; i++)
    {
        printf("\n %d is stored at %d location
                                    of array"),
            number[i], i;
    }
}
```

Output :

```
Enter number : 80
Enter number : 68
Enter number : 54
Enter number : 35
Enter number : 97
80 is stored at 0 location of array
68 is stored at 1 location of array
54 is stored at 2 location of array
35 is stored at 3 location of array
97 is stored at 4 location of array
```

Q.19. Write a program in 'C' to accept 10 numbers in an array. Sort the array elements in ascending order.

Ans.

```
#include <stdio.h>
void main( )
{
    int arr[10];
    int a = 0, temp = 0, j = 0;
    for(i = 0; i <= 9; i++)
    {
        printf("Enter number :");
        scanf("%d", &arr[i]);
    }
    temp = arr[0];
    for(j = 0; j < = 9; j++)
    {
        if(arr[i + 1] < arr[i])
        {
            temp = arr[i];
            arr[i] = arr[i + 1];
            arr[i + 1] = temp;
        }
    }
    printf("\n Array in ascending order is :");
    for(i = 1; i <= 10; i++)
    {
        printf("\n %d", arr[i]);
    }
}
```

Output :

```
Enter number : 7
Enter number : 6
Enter number : 2
Enter number : 1
Enter number : 8
Array in ascending order is :
    1
    2
    6
    7
    8
```

Q.20. Write a program to sort elements of an array in alphabetical order.

Ans.

```
#include <stdio.h>
void main( )
{
    int arr['B', 'D', 'H' 'A', 'O' 'I', 'Z', 'X',
'M', 'P'];
    int i = 0, j = 0, temp = 0;
    for(i = 0; i <= 9; i++)
    {
        temp = arr[0];
        for(j = 0; j < = 9; j++)
        {
            if(arr[i + 1] < arr[i])
            {
                temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
        }
        printf("\n Array in alphabetical order
is :");
        for(i = 1; i <= 10; i++)
        {
            printf("\n %d", arr[i]);
        }
    }
```

Output:

```
Array in alphabetical order is:
A B D H I M O P X Z
```

MULTI-DIMENSIONAL ARRAYS

Q.21. Explain Multi-Dimensional Arrays.

Ans. Multi-Dimensional Arrays:

- C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

  type name[size1][size2]...[sizeN];

- For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array:

  int threedim[5][10][4];

TWO-DIMENSIONAL ARRAYS

Q.22. Explain the Two-Dimensional Array with an example.

Ans. Two-Dimensional Array

- The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x, y we would write something as follows:

  type arrayName [x][y];

- Where type can be any valid C data type and arrayName will be a valid C identifier. A two-dimensional array can be thick as a table which will have x number of rows and y number of columns. A 2-dimentional array a, which contains three rows and four columns can be shown as below:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0] [1]	a[0] [2]	a[0] [3]
Row 1	a[1][0]	a[1] [1]	a[1] [2]	a[1] [3]
Row 2	a[2][0]	a[2] [1]	a[2] [2]	a[2] [3]

- Thus, every element in array a is identified by an element name of the form a[i][j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

Initializing Two-Dimensional Arrays:

- Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
{0, 1, 2, 3} , /* initializers for row indexed by 0 */
{4, 5, 6, 7} , /* initializers for row indexed by 1 */
{8, 9, 10, 11} /* initializers for row indexed by 2
*/
};
```

- The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example:

  int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

Accessing Two-Dimensional Array Elements:

- An element in a 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example:

  int val = a[2][3];

- The above statement will take 4th element from the 3rd row of the array. We can verify it in the above diagram. Let us check the below program where we have used a nested loop to handle a two dimensional array:

Example:

```
#include <stdio.h>
int main ()
{
/* an array with 5 rows and 2 columns*/
int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
int i, j;
/* output each array element's value */
for ( i = 0; i < 5; i++ )
{
for ( j = 0; j < 2; j++ )
{
printf("a[%d][%d] = %d\n", i,j, a[i][j] );
}
}
return 0;
}
```

```
Output: a[0][0]: 0
        a[0][1]: 0
        a[1][0]: 1
        a[1][1]: 2
        a[2][0]: 2
        a[2][1]: 4
        a[3][0]: 3
        a[3][1]: 6
        a[4][0]: 4
        a[4][1]: 8
```

As explained above, we can have arrays with any number of dimensions, although it is likely that most of the arrays we create will be of one or two dimensions.

Q.23. Write a 'C' program for addition of two 3 × 3 matrix.

Ans.

```
#include<stdio.h>
#include<conio.h>
int main( )
{
    int mat1[3][3], mat2[3][3], mat3[3][3], i, j;
    clrscr( );
    printf("\n Enter values for first 3 × 3 matrix
                                        : \n");
    for(i = 0; i <= 2; i++)
    {
        for(j = 0; j <= 2; j++)
            scanf("%d", &mat1[i][j]);
    }
    printf("\n Enter values for second 3 × 3
                                    matrix : \n");
    for(i = 0; i <= 2; i++)
    {
        for(j = 0; j <= 2; j++)
        scanf("%d", &mat2[i][j]);
    }
    printf("\n Matrix 1 : \n");
    for(i = 0; i <= 2; i++)
    {
        for(j = 0; j <= 2; j++)
            printf("%d\t", mat1[i][j]);
            printf("\n");
    }
    printf("\n Matrix 2 : \n");
    for(i = 0; i <=2; i++)
    {
        for (j = 0; j <= 2; j++)
            printf("%d\t", mat2[i][j]);
            printf("\n");
    }
    for(i = 0; i <= 2; i++)
    {
    for(j = 0; j <= 2; j++)
        mat3[i][j] = mat1[i][j] + mat2[i][j];
    }
    printf("\n The new matrix after addition of
                                    the above two
                matrices is : \n");
    for(i = 0; i <= 2; i++)
    {
        for(j = 0; j <= 2; j++)
            printf("%d\t", mat3[i][j]);
            printf("\n");
    }
    getch( );
}
```

```
Output :
    Enter values for first 3 × 3 matrix :
     1  2  3  4  5  6  7  8  9
    Enter values for second 3 × 3 matrix :
     2  3  4  5  6  7  8  9  1
Matrix 1 :
    1  2  3
    4  5  6
    7  8  9
Matrix 2 :
    2  3  4
    5  6  7
    8  9  1
The new matrix after addition of the two
above matrices is :
    3   5   7
    9  11  13
   15  17  10
```

Q.24. Write a program to find transpose of any (3 × 3) matrix.

Ans.
```
#include<stdio.h>
#include<conio.h>
int main( )
{
    int mat[3][3], i, j, temp;
    clrscr( );
    printf("\n Enter values for 3 × 3 matrix : \n");
    for (i = 0; i <= 2; i ++)
    {
        for(j = 0; j <= 2; j ++)
        {
            scanf("%d", &mat[i][j]);
        }
    }
    printf("\n\n The matrix you entered is : \n");
    for(i = 0; i < 2; i ++)
    {
        for(j = 0; j <= 2; j ++)
        printf("%d\t", mat[i][j]);
        printf("\n");
    }
        for(j = 0; i <= 2; i ++)
        {
            for(j = i + 1; j <= 2; j ++)
            {
                temp = mat[i][j];
                mat[i][j] = mat[j][i];
                mat[j][i] = temp;
            }
        }
        printf("\n\n Transpose of the matrix is : \n");
        for(i = 0; i <= 2; i ++)
        {
            for(j = 0; j <= 2; j ++)
                printf("%d\t", mat[i][j]);
                printf("\n");
            }
            getch( );
            return 0;
        }
```

Output :

```
Enter values for first 3 × 3 matrix:
1  8  7
5  8  4
3  2  1
The matrix you entered is :
1  8  7
5  8  4
3  2  1
Transpose of the matrix is :
1  5  3
8  8  2
7  4  1
```

Q.25. Write a program to multiply two (2 × 2) matrices.

Ans.
```
#include<stdio.h>
#include<conio.h>
int main( )
{
    int mat1[2][2], mat2[2][2], mat3[2][2],
                                    i, j, k, sum;
    clrscr( );
    printf("\n Enter values for first 2 × 2 matrix : \n");
    for (i = 0; i < 2; i ++)
    {
        for(j = 0; j < 2; j++)
        scanf("%d", &mat1[i][j]);
    }
    printf("\n Enter values for second 2 × 2
                                    matrix : \n");
    for (i = 0; i < 2; i ++)
    {
        for(j = 0; j < 2; j ++)
            scanf("%d", &mat2[i][j]);
    }
printf("\n The first 2 × 2 matrix entered by you is
                                    : \m");
for(i = 0; i < 2; j ++)
{
    for(j = 0; j < 2; j ++)
        printf("%d\t", mat1[i][j]);
        printf("\n");
```

```
}
printf("\n The second 2 × 2 matrix entered by
                                you is : \n");
for(i = 0; i < 2; i ++)
    {
    for(j = 0; j < 2; j ++)
    printf("%d\t", mat3[i][j]);
    printf("\n");
}
for (i = 0; i < 2; i ++)
    for (j = 0; j < 2; j ++)
    {
        sum = 0;
        for (k = 0; k < 2; k ++)
        sum = sum + mat1[i][k] * mat2[k][j];
        mat3[i][j] = sum;
    }
}
printf("\n The product of the above two
                                matrices is : \n");
for(i = 0; i < 2; i ++)
{
    for(j = 0; j < 2; j ++)
    printf("%d\t", mat3[i][j]);
    printf("\n");
}
getch( );
return 0;
}
```

Q.26. What do you mean by two-dimensional array of characters? Explain.

Ans. Two-Dimensional Array of Characters :

- The two-dimensional character array also called a string array or an array of strings is initialized like single-dimensional arrays.
- **For example,** a C program segment may contain the following declaration :

```
Static char;
Color [6] [7] = {"red", "green", "blue", "white",
                                "black", "yellow"};
```

- In string manipulation, two-dimensional arrays can also be used, though they can be better represented by a pointer array.

Example :

This example program asks the user to type our name. When we do so, it checks the name against a master list to see if it is worthy of entry to the people.

```
# define FOUND 1
# define NOTFOUND 0
main( )
{
    char master lists [6] [10] = {"akash",
    "parag", "ram", "yogesh", "kunal", "raju",};
    int i, flag, a;
    char yourname [10];
    printf ("\n Enter your name");
    scanf ("%s", yourname);
    flag = NOTFOUND;
    for (i = 0 ; i <= 5 ; i++)
    {
        a = strcmp (& masterlist [i] [0],
                                yourname);
        if (a = = 0)
        {
            printf ("Welcome, you can enter
                                the palace");
            flag = FOUND;
            break;
        }
    }
if (flag = = NOTFOUND)
printf ("Sorry, you are a intruder");
}
```

Output :

```
Enter your name dinesh
Sorry, you are a intruder
Enter your name ram
Welcome, you can enter the palace
```

- Notice how the two-dimensional character array has been initialized. The order of the subscripts in the array declaration is important. The first subscript gives the number of names in the array, while the second subscript gives the length of each item in the array.

STRING

Q.27. How does string work in C programming? Explain in detail.

Ans.

- The string in the C programming language is a one-dimensional array of characters that is terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.
- The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello".

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

- If we follow the rule of array initialization then we can write the above statement as follows:

```
char greeting[] = "Hello";
```

Following is the memory presentation of the above-defined string in C/C++:

H	e	l	l	o	'\0'

We do not place the null character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above - mentioned string.

Example:

```
#include <stdio.h>
int main ()
{
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
printf("Greeting message: %s\n", greeting );
return 0;
}
```

Output:

Greeting message: Hello

C supports a wide range of functions that manipulate null-terminated strings:

S.No.	Function & Purpose
(i)	strcpy(s1, s2); Copies string s2 into string s1.
(ii)	strcat(s1, s2); Concatenates string s2 onto the end of string s1.
(iii)	strlen(s1); Returns the length of string s1.
(iv)	strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
(v)	strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1.
(vi)	strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.

Example:

```
#include <string.h>
int main ()
{
  char str1[12] = "Hello";
  char str2[12] = "World";
  char str3[12];
  int len ;
  /* copy str1 into str3 */
  strcpy(str3, str1);
  printf("strcpy( str3, str1) : %s\n", str3 );
  /* concatenates str1 and str2 */
  strcat( str1, str2);
  printf("strcat( str1, str2): %s\n", str1 );
  /* total lenghth of str1 after concatenation */
  len = strlen(str1);
  printf("strlen(str1) : %d\n", len );
  return 0;
}
```

Output:

strcpy(str3, str1): Hello

strcat(str1, str2): HelloWorld

strlen(str1) : 10

We can find a complete list of C string related functions in C Standard Library.

SEARCHING

Q.28. What is Searching?

Ans. Searching :

- Searching is a process of finding an element within the list of elements stored in any order randomly.
- Searching is divided into two categories:
  (i) Linear search.
  (ii) Binary search.

Q.29. Explain Linear Search.

Ans. Linear Search:

- In Linear search, users can access each element of an array one by one sequentially. Therefore it is also known as sequential search.
- A search is said to be successful if the user gets the desired element in the searching process.
- A search is said to be unsuccessful if all the elements are accessed and the desired element is not found.
- It is used for unsorted and unordered small list of elements.
- It has a time complexity of O(n), which means the time is linearly dependent on the number of elements, which is not bad, but not that good too.
- It has a very simple implementation.

Q30. Write a program to search a given element in an array of dimension 5 using linear search.

Ans.
```
main( )
{
int a[5],i,ele;
clrscr( );
printf("Enter five elements:");
/* Scanning five elements */
for(i=0;i<=4;i++)
scanf("%d",&a[i]);
printf("Enter the element to be search:");
scanf("%d",&ele);
/*Linear search*/
for(i=0;i<=4;i++)
{
if(ele==a[i])
{
printf("\n The element is present in the given
list");
break;
}
}
if(i==5)
printf("\n The element is not present in the given
list");
getch( );
}
```

Output:
```
Enter Five Elements: 23 55 67 78 28
Enter the element to be search: 55
The element is present in the given list.
```

Q.31. Explain Binary search.

Ans. Binary Search :

- Binary searching is the process of locating an element in the sorted list.
- The search starts by dividing the list into two parts.
- If the search element is less than the middle value, the top list will be searched, after finding the middle element of that list.
- The process continues until the element is found or the search in the top list is completed.
- The same process is continued for the bottom list until the element is found or the search in the bottom list is completed.
- If an element is found that must be the middle value.
- It is great to search through large sorted arrays.
- It has a time complexity of O(log n) which is a very good time complexity.
- It has a simple implementation.

Q.32. Write a program to search a given element in an array of dimension 10 using binary search.

OR Write a program to search a particular element in a list using the binary search technique.

Ans.
```
main( )
{
int array[10];
int i, j, n, temp, num;
```

```
int low,mid,high;
clrscr();
printf("Enter the value of the array\t");
scanf("%d",&n);
printf("Enter the elements one by one:\n");
for(i=0;i<n;i++)
{
scanf("%d",&array[i]);
}
printf("Enter the elements to be searched\n");
scanf("%d",&num);
low=1;
high=n;
do
{
mid=(low+high)/2;
if(num<array[mid])
high=mid -1;
else if(num>array[mid])
low=mid+1;
}
while(num!=array[mid]&&low<=high);
if(num==array[mid])
{
printf("\n The entered elements is present");
}
else
{
printf("Serach is failed \n");
}
getch();
}
```

Output : Enter the value of the array 5
Enter the elements one by one.
40
50
60
70
80
Enter the element to be searched 60
The entered element is present.

SORTING

Q.33. What is Sorting?

OR What do you mean by Sorting?

Ans. Sorting :

- The process of arranging data or records in a sequence is known as sorting. The data can be sorted in two ways such as ascending Sort and descending Sort.
- Ascending sort is a sorting technique in which the smallest data is placed at the first position and the largest data is placed at the last position.
- Descending sort is a sorting technique in which the largest data is placed in the first position and the smallest data is placed in the last position.

Q.34. Explain Selection Sort.

Ans. Selection Sort :

- The selection sort technique is based upon the extension of the minimum/ maximum technique.
- Selection sort finds the smallest element of an array and interchanges it with the element in the first position of the array.
- Let us consider there are N elements in the array. Then the process of selection sort is given as :

  Step 1 : Find the location of the smallest element in the list of N elements. Let loc is the location of the smallest number in an array.

  a[1],a[2].....a[n] and then interchange

  a[loc] ⇔ a[1] then

  Step 2 : Find the location of the smallest in the subject of n – 1 elements. a[2], a[3],.....a[n] and then interchange.

  a[loc] ⇔ a[2] then

  a[1], a[2] is sorted, since a[1] ≤ a[2]

  Step 3 : Find the location of the smallest in the sublist of (n – 2) elements.

  a[3], a[4].....a[n] and then interchange

  a[loc] ⇔ a[3] then

  a[1], a[2], a[3] is sorted since a[1] ≤ a[2] ≤ a[3]

  Repeat the process by reducing one element each from array.

  Step(n–1) : Find the location of the smallest of the element a[n – 1],

  a[n] and then interchange a[loc] and a[n–1].

a[1], a[2].....a[n] is sorted

Since, a[n–1] ≤ a[n]

Therefore a is sorted after (n–1) step.

Selection Sort

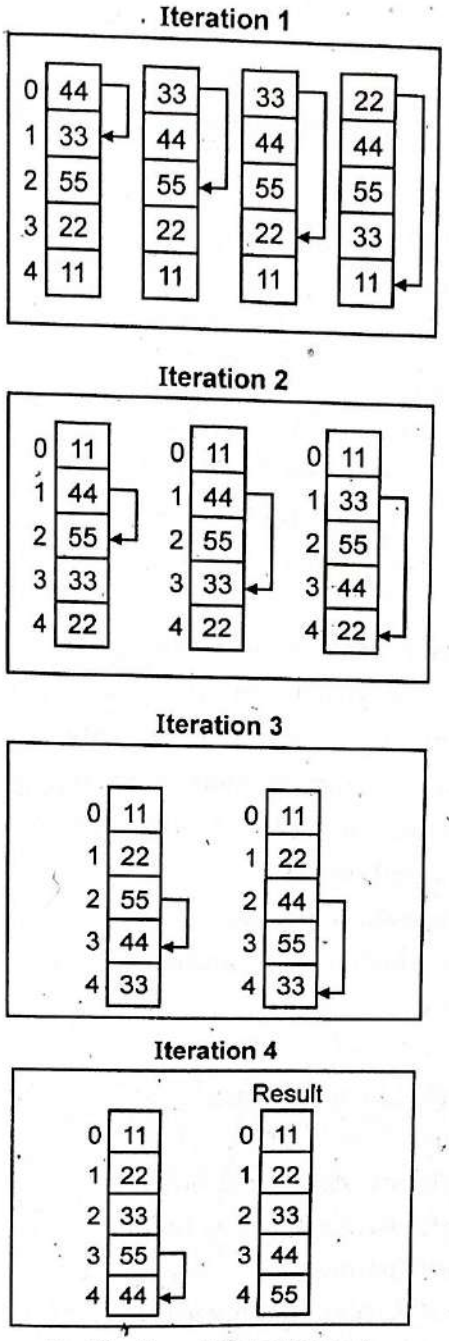


Fig. Working of Selection sort

Time complexity of Selection Sort:

Average case time complexity : $O(n^2)$

Worst-case time complexity: $O(n^2)$

Best case time complexity: $O(n^2)$

Q.35. Write a program to sort an array of dimension 5 in ascending order by using the selection sort method.

OR Write a C program to sort 20 elements stored in an array, in ascending order.

Ans.
```
main( )
{
int a[5],i,j,temp;
clrscr( );
printf("Enter Five Elements:");
/* Scanning five elements*/
for(i=0;i<=4;i++)
{
scanf("%d",&a[i]);
}
/*Selection sort */
for(i=0;i<=3;i++)
{
for(j=i+1;j<=4;j++)
{
if(a[i]>a[j])
{
temp = a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
printf("The sorted elements in ascending order:");
for(i=0;i<=4;i++)
printf("\t%d",a[i]);
getch();
}
```

Output:
Enter Five Elements : 44 33 55 22 11
The sorted elements in ascending order:
11 22 33 44 55

Q.36. Write a program to sort an array of dimension 5 in descending order by using the selection sort method.

Ans.
```
main( )
{
int a[5],i,j,temp;
clrscr();
printf("Enter Five Elements:");
/* Scanning five elements*/
for(i=0;i<4;i++)
```

```
scanf("%d",&a[i]);
/*Selection sort */
for(i=0;i<=4;i++)
{
for(j=i+1;j<=4;j++)
{
if(a[i]<a[j])
{
temp = a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
printf("The sorted elements in descending order:");
for(i=0;i<=4;i++)
printf("\t%d",a[i]);
getch();
}
```

Output:
Enter Five Elements : 11 22 33 44 55
The sorted elements in ascending order:
55 44 33 22 11

Q.37. Write a program to arrange 10 numbers read from the keyboard in descending order.

OR Write a 'C' program to sort 10 elements stored in an array in descending order.

Ans.

```
main ( )
{
int a[10],i,j,temp;
clrscr( );
printf("Enter Five Elements:");
/* Scanning ten elements*/
For(i=0;i<9;i ++)
scanf("%d" & a[i]);
/*Selection sort*/
for(i=0; i<=8; ++)
{
for (j=i+1; j<=9j++)
{
If (a[i]<a[i])
{
temp = a[i];
a[i]=a[i];
a[i]=temp;
}
}
}
printf("The sorted elements in descending order:");
for(i=0; i<=9; i++)
printf("\t%d",a[i];
getch( );
}
```

Output :
Enter ten elements :
11 22 33 44 55 66 77 88 99 100
The sorted elements in descending order :
100 99 88 77 66 55 44 33 22 11

Q.38. Write a program to sort a given array in the order specified by the user. Mention the sorting method used and explain it.

OR Write a program that sorts the given array according to the following menu :
(i) Ascending order.
(ii) Descending order.
State which sorting method you have used.

Ans.

```
main()
{
int a[5],i,j,temp,choice;
clrscr();
printf("\n1. Ascending order");
printf("\n2. Descending order");
printf("\n3.Exit");
printf("Enter your choice");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter Five Elements:");
for(i=0;i<=4;i++)
{
scanf("%d",&a[i]);
}
```

```
/*Selection sort */
for(i=0;i<=4;i++)
{
  for(j=i+1;j<=4;j++)
  {
   if(a[i]>a[j])
   {
    temp = a[i];
    a[i]=a[j];
    a[j]=temp;
   }
  }
}
printf("The sorted elements in ascending order:");
for(i=0;i<=4;i++)
printf("\t%d",a[i]);
break;
case 2:
printf("Enter Five Elements:");
/* Scanning five elements */
for(i=0;i<=4;i++)
scanf("%d",&a[i]);
/*Selection sort */
for(i=0;i<=4;i++)
{
      for(j=i+1;j<=4;j++)
      {
      if(a[i]<a[j])
      {
      temp = a[i];
      a[i]=a[j];
      a[j]=temp;
      }
    }
  }
  printf("The sorted elements in descending order:");
  for(i=0;i<=4;i++)
  printf("\t%d",a[i]);
  break;
  case 3: exit( );
  }
        getch( );
  }
```

Output:

1. Ascending order
2. Descending order
3. Exit

Enter your choice 1
Enter Five Elements:
44 33 55 22 11
The sorted elements in ascending order:
11 22 33 44 55

In selection sort, the first element gets compared with the second element. If the second element is less than the first element then interchanging takes place between these two elements otherwise array of elements remain as it is. Then first element gets compared with third element and same procedure is repeated until we get sorted elements.

Q.39. Explain the working of Bubble Sort.

Ans. Bubble Sort :

- In bubble sort, each element is compared with its adjacent element.
- If the first element is larger than the second one, then the position of the elements is interchanged, otherwise it is not changed.
- Then next element is compared with its adjacent elements and the same process is repeated for all the elements in the array.
- Let us consider there are N elements in the array. The process of bubble sort works as :

  Step 1 : First compare the 1st and 2nd elements of an array. If $1^{st} < 2^{nd}$ then compare the 2nd with 3rd.

  Step 2 : If $2^{nd} > 3^{rd}$ Then interchange the value of 2nd and 3rd.

  Step 3 : Now, compare the value of 3rd (which has the value of 2nd) with 4th.

  Step 4 : Similarly compare until the N–1 element is compared with N^{th} element.

  Step 5 : Now, the highest value element is reached in the Nth place.

  Step 6 : Now elements are compared until the N–1 element.

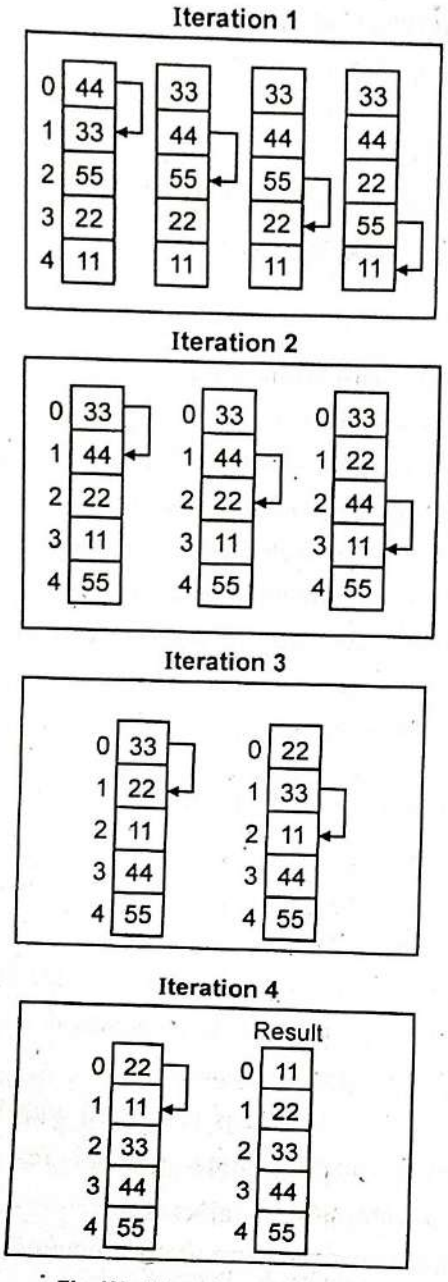


Fig. Working of bubble sort

Q.40. Explain time complexity and space complexity of bubble sort.

Ans.

(i) Time Complexity:

Case	Time Complexity
Best Case	O(n)
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

(a) Best Case Complexity:

It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of bubble sort is O(n).

(b) Average Case Complexity:

It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of bubble sort is $O(n^2)$.

(c) Worst Case Complexity:

It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of bubble is $O(n^2)$.

(ii) Space Complexity:

Space Complexity	O(1)
Stable	YES

- The space complexity of bubble sort is O(1). It is because, in bubble sort, an extra variable is required for swapping.
- The space complexity of optimized bubble sort is O(2). It is because two extra variables are required in optimized bubble sort.

Q.41. Write a program to sort an array of dimension 5 in ascending order by using the bubble sort method.

OR Write a program to sort the elements using bubble sort. Also, show the result of each pass.

OR Write a program to sort the elements of an array using bubble sort.

Ans.

```
main( )
{
int a[5],i,j,temp;
clrscr();
printf("Enter Five Elements:");
/* Scanning ten elements*/
for(i=0;i<=4;i++)
scanf("%d",&a[i]);
/*Bubble sort */
for(j=0;j<=3;j++)
{
for(i=0;i<=3-j;i++)
{
if(a[i]>a[i+1])
{
```

```
temp = a[i];
a[i]=a[i+1];
a[i+1]=temp;
}
}
}
printf("The sorted elements in ascending order:");
for(i=0;i<=4;i++)
printf("\t%d",a[i]);
getch();
}
```

Output :
Enter Five Elements: 44 33 55 22 11
The sorted elements in descending order:
11 22 33 44 55

Q.42. Write a program to sort an array of dimension 5 in descending order by using the bubble sort method.

Ans.

```
main( )
{
int a[5],i,j,temp;
clrscr();
printf("Enter Five Elements:");
/* Scanning ten elements*/
for(i=0;i<=4;i++)
scanf("%d",&a[i]);
/*Bubble sort */
for(j=0;j<=3;j++)
{
for(i=0;i<=3-j;i++)
{
if(a[i]<a[i+1])
{
temp = a[i];
a[i]=a[i+1];
a[i+1]=temp;
}
}
}
printf("The sorted elements in descending order:");
for(i=0;i<=4;i++)
printf("\t%d",a[i]);
getch();
}
```

Output :
Enter Five Elements: 1 2 3 4 5
The sorted elements in descending order:
5 4 3 2 1

Q.43. Explain Insertion Sort.

Ans. Insertion Sort :

- The insertion sort insert each element in the proper place. There are n elements in the array and we place each element of array at the proper place in the previously sorted element.
- Let us consider there are N elements in the array a [ ]. Then the process of inserting each element at the proper place works as :

Step 1 : a[1] is already sorted because of only one element.

Step 2 : a[2] is inserted before or after a[1] so a[1] and a[2] is sorted

Step 3 : a[3] is inserted before a[1], in between a[1] and a[2] or after a[2]. So a[1], a[2] and a[3] is sorted and so on.

Step 4 : a[N] is inserted into its proper place in the array.

a[1], a[2].....a[N-1]
so a[1], a[2].....a[N] is sorted.

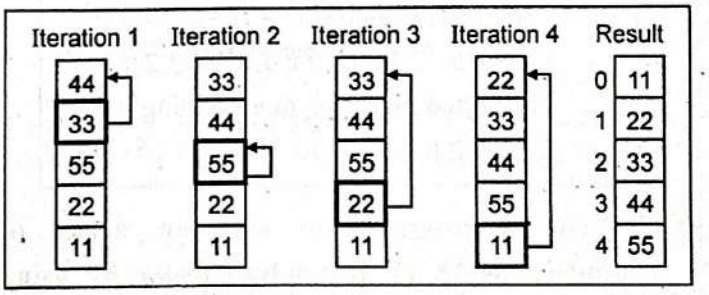


Fig. Working of insertion sort

Time complexity of Insertion Sort:

Best case time complexity: $O(n^2)$

Average and worst-case time complexity: $O(n^2)$

Q.44. Write a program to sort an array of dimension 15 in ascending order by using the insertion sort method.

Ans.
```
main( )
{
int a[15],i,j,temp;
clrscr();
printf("Enter Fifteen Elements:");
/* Scanning fifteen elements*/
for(i=0;i<=14;i++)
scanf("%d",&a[i]);
/*Insertion sort */
for(i=1;i<=14;i++)
{
for(j=0;j<=i-1;j++)
{
if(a[i]<a[j])
{
temp = a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
/*Display elements*/
printf("The sorted elements in ascending order:");
for(i=0;i<=14;i++)
printf("\t%d",a[i]);
getch();
}
```

Output:
```
Enter Five Elements:
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
The sorted elements in ascending order:
1 2 3 4 5 6 7 8 9 1 11 12 13 14 15
```

Q.45. Write a program to sort an array of dimensions 15 in descending order by using insertion sort method.

Ans.
```
main( )
{
int a[15],i,j,temp;
clrscr();
printf("Enter Fifteen Elements:");
/* Scanning fifteen elements*/
for(i=0;i<=14;i++)
scanf("%d",&a[i]);
/*Insertion sort */
for(i=1;i<=14;i++)
{
for(j=0;j<=i-1;j++)
{
if(a[i]>a[j])
{
temp = a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
/*Display elements*/
printf("The sorted elements in descending order:");
for(i=0;i<=14;i++)
printf("\t%d",a[i]);
getch();
}
```

Output:
```
Enter Five Elements:
1 2 3 4 5 6 7 8 9 1 11 12 13 14 15
The sorted elements in ascending order:
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

PROGRAMS FOR PRACTICE:

Q.1. Write a program to generate the following output using loop structure.

1 2 3 4
1 2 3

Q.2. Write a program to read the age of 100 persons and count the number of persons in the age group 50 to 60. Use for and continue statements.

Q.3. Write a program to read three integer values from the keyboard and displays the output stating that they are the sides of a right-angled triangle.

Q.4. Write a C program to print sum of diagonal elements of a 3 × 3 matrix.

Q.5. Write a C program to add two matrices and store the result in third matrix.

POINTS TO REMEMBER :

(1) C provides a powerful syntax for condition checking that is control structures using decision making and looping or repeating some part of code.

(2) The statements like if, if-else and else-if ladder are used for decision making and to execute some part of code depending on trueness or falseness of condition.

(3) The if statement is used to check some conditions and then execute some part of code.

(4) The if statement has other ways of using if-else and else-if ladder.

(5) The switch-case is a statement that can be used to select from a number of options any one option or a case that matches the expression used with switch.

(6) Every switch statement has a default case which is executed when no other option or case matches with the expression value used with a switch statement.

(7) Every case within the switch statement must be ended with a break statement, to execute only one case at a time.

(8) Cases can be arranged in any order.

(9) For repeatedly executing the same code of the program many times C provides various loop control structures like while, do-while and for.

(10) In for loop we need initialization, condition checking and increment or decrement statements.

(11) The basic difference between for loop and a while loop is that in for loop we can guess the number of executions of the loop body whereas in while loop the loop body statements are executed until the condition used with a while does not become false.

(12) In the do-while loop body executes at least one time irrespective of the condition outcome (i.e. true or false).

(13) The continue statement is used inside the loop body for the continuation of the code in some special cases.

(14) String is not a data type in C, but it is considered a data structure stored in an array.

(15) '\0' marks the "end of string".

(16) Using an array of pointers to string is more efficient than using two-dimensional array to store strings.

(17) Name of string contains the base address to the first element of string.

(18) Searching is a process of finding an element within the list of elements stored in any order on randomly.

(19) Searching is divided into two categories.

(i) Linear search.

(ii) Binary search.

(20) In Linear search, the user can access each element of an array one by one sequentially. Therefore it is also known as sequential search.

(21) Binary searching is the process of locating an element in the sorted list.

(22) The process of arranging data or records in a sequence is known as sorting. The data can be sorted in two ways such as ascending Sort and descending Sort.

(23) The selection sort technique is based upon the extension of the minimum/ maximum technique.

(24) Selection sort finds the smallest element of an array and interchanges it with the element in the first position of the array.

(25) In bubble sort, each element is compared with its adjacent element.

(26) The insertion sort insert each element in proper place. There are n elements in the array and we place each element of the array at proper place in the previously sorted element.

UNIT - III
FUNCTION AND RECURSION

SYLLABUS

(a) **Function:** Functions (including using built-in libraries), Parameter passing in functions, Call by value, Passing arrays to functions: Idea of call by reference.

(b) **Recursion:** Recursion, as a different way of solving problems. Example programs, Such as finding factorial.

INTRODUCTION

A function is a group of statements that together perform a task. Every C program has at least one function which is main( ), and all the most trivial programs can define additional functions.

Function in C plays a major role because it supports and implements a modular approach to programming in C coding. By declaring and using various functions in C program it becomes more structured and systematic. This makes the task of removing errors easier and it also makes programs readable.

This chapter introduces various concepts of function definition, usage and applications of function in C.

FUNCTION

Q.1. What is a Function?

Ans. Function:

A function is a group of statements that have been given a name. This allows us to break our program into manageable pieces and reuse our code. The function contains the set of programming statements enclosed by { }. A function can be called multiple times to provide reusability and modularity to the C program.

Q.2. What are the advantages of a Function?

Ans. Advantages of Functions:

(i) The function makes the lengthy and complex program easy and in short forms. It means a large program can be sub-divided into self-contained and convenient small modules having a unique name.

(ii) The length of the source program can be reduced by using functions by using it at different places in the program according to the user's requirement.

(iii) By using the function, memory space can be properly utilized. Also, less memory is required to run the program if a function is used.

(iv) They also allow more than one person to work on one program.

(v) The function increases the execution speed of the program and makes the programming simple.

(vi) By using the function, the portability of the program is very easy.

(vii) It removes the redundancy (occurrence of duplication of programs) i.e. avoids the repetition and saves time and space.

(viii) Debugging (removing error) becomes very easier and fast using the function sub-programming.

(ix) Functions are more flexible than library functions.

(x) Testing (verification and validation) is very easy by using functions.

(xi) However, function call is always overhead in a C program.

Q.3. Explain different elements of User-Defined Functions.

Ans. User-Defined Functions:

- Every 'C' program has at least one function which is the main function, but a program can have any number of functions. The main () function in C is a starting point of a program.
- In 'C' programming, functions are divided into two types:
  (i) Library functions
  (ii) User-defined functions
- The difference between the library and user-defined functions in C is that we do not need to write a code for a library function. It is already present inside the header file which we always include at the beginning of a program. We just have to type the name of a function and use it along with the proper syntax. Printf and scanf are examples of a library functions.
- Whereas, a user-defined function is a type of function in which we have to write a body of a function and call the function whenever we require the function to perform some operation in our program.
- A user-defined function in C is always written by the user, but later it can be a part of 'C' library. It is a major advantage of 'C' programming.

C programming functions are divided into three activities such as,
(i) Function declaration
(ii) Function definition
(iii) Function call

(i) Function Declaration:

- Function declaration means writing a name of a program. It is a compulsory part of using functions in code. In a function declaration, we just specify the name of a function that we are going to use in our program like a variable declaration. We cannot use a function unless it is declared in a program. A function declaration is also called a "Function prototype."
- The function declarations (called prototype) are usually done above the main () function and take the general form:
  return_data_type function_name (data_type arguments);
- The return_data_type is the data type of the value function returned to the calling statement.
- The function_name is followed by parentheses
- Arguments names with their data type declarations optionally are placed inside the parentheses.
- We consider the following program that shows how to declare an addnum function to calculate the sum of two integers.

```
#include <stdio.h>
/*Function declaration*/
int add(int a,b);
/*End of Function declaration*/
int main( ) { }
```

- Keep in mind that a function does not necessarily return a value. In this case, the keyword void is used.
- For example, the output_message function declaration indicates that the function does not return a value: void output_message();

(ii) Function Definition:

Function definition means just writing the body of a function. A body of a function consists of statements that are going to perform a specific task. A function body consists of a single or a block of statements. It is also a mandatory part of a function.

```
int add(int a,int b)        /*function body*/
{
  int c;
  c=a+b;
  return c;
}
```

(iii) Function Call:

A function call means calling a function whenever it is required in a program. Whenever we call a function, it performs an operation for

which it was designed. A function call is an optional part of a program.

result = add(4,5);

Here, is the complete code:

```
#include <stdio.h>
int add(int a, int b);     //function declaration
int main()
{
   int a=10,b=20;
   int c=add(10,20);     //function call
   printf("Addition:%d\n",c);
   getch();
}
   int add(int a,int b)   //function body
{
  int c;
  c=a+b;
  return c;
}
```

Output:
Addition: 30

Q.4. What are Parameters/ Arguments in function?

Ans.

- Arguments and parameters are the variables used in a program and a function. Variables used in the calling function are called arguments. These are written within the parentheses followed by the name of the function. They are also called *actual parameters*, as they are accepted in the main program (or calling function).
- Variables used in the function definition (called function) are called parameters. They are also referred to as *formal parameters* because they are not the accepted values. They receive values from the calling function.
- Parameters must be written within the parentheses followed by the name of the function, in the function definition.

User-Defined Data Types:

- Data type that are derived from an existing data type is called user-defined data types (UDT). We can create a customized data types to extend the built-in types that are already available.
- typedef and enum are used to create user-defined data types. Advantage of user-defined data type is that it increases the program readability.
- C allows the programmer to create custom data types in different ways:

(i) 'typedef':

'typedef' allows users to define the identifier which would represent an existing data type. This defined data type can be used to declare variables.

Example:

```
typedef int integer;
int main()
{
integer num1 = 1, num 2= 2;
      /*num1 and num2 are declared as 'int'*/
printf("%d %d, num1, num2);
return 0;
}
```

Output:
1, 2

(ii) 'enum'

"enum" is the keyword and "day" is the user-defined data type that is used to declare the variables within the curly braces, mainly used to assign names to integral parts. We can also explicitly assign the enumeration constants.

Example:

```
enum day {monday, tuesday, wednesday,
thursday, friday, saturday};
enum State {Working = 1, Failed = 0};
                    /* explicit assignment*/
int main( )
{
   enum day today;
   today = friday;
   printf("Today is day %d \n",today);
   printf("%d, %d, ", Working, Failed);
   return 0;
}
```

Output:
Today is day 4
1, 0

(iii) Structures:

Example:

Structure datatype in C language allows us to group together relative data of different types. The keyword "struct" is use followed by structure the tag (name) to define the structure.

The closing bracket in the structure type declaration must be followed by a semicolon.

Example:

```
struct student
{
   char name[10];
   int R_no;
   float percent;
};
int main(){
struct student s1;
printf("Enter name, roll number and percentage of
student\n");
 scanf("%s %d %f",s1.name, &s1.R_no,
                                  &s1.percent);
printf("\n***Student details***\n");
printf("Name= %s\nRoll no.= %d\nPercentage=
%f",s1.name,s1.R_no,s1.percent);
return 0;
}
```

Output:

```
Enter name, roll number and percentage
of student
hellocodies 1234 98.9
***Student details***
Name= hellocodies
Roll no.= 1234
Percentage= 98.900002
```

(iv) Unions:

Union stores different data types in the same memory location with only one member assigned memory and value at any given time. The memory union get assigned is the largest of all members the union contain.

Example:

```
union example
{
int X;
float Y;
long double Z;
};
 int main(){
 union example Data;
 printf("size of union is of LONG DOUBLE =
 %u\n",sizeof(Data));
 Data.X=100;     /* 'X' got memory and value*/
 Data.Y=98.99;  /* 'X' cleared & memory freed
                                    for 'Y'*/
 printf("Now 'X contain garbage value = %d
 \n",Data.X);
 printf("Y = %lf",Data.Y);
 return 0;
}
```

Output:

```
size of union is of LONG DOUBLE = 12
Now 'X contain garbage value = 1120271073
Y = 98.989998
```

Q.5. Write the output of following program. Take suitable input.

```
#include<stdio.h>
void large( )
{
      int a, b;
      printf("Enter values of a and b:");
      scanf("%d%d, &a, &b);
      if(a < b)
      printf("large = %d", b);
      else
      printf("large = %d", a);
}
main( )
{
      large( );
      large( );
}
```

Output :

```
Enter values of a and b:
      2     =   3
      large =   3
```

Q.6. What will be the output of the following programs?

i)
```
#include<stdio.h>
void display( );
int main( )
{
    printf("\n Hello");
    display( );
    return 0;
}
void display( )
{
    printf("\n World");
    main( );
}
```

Output:
Both the messages will get printed indefinitely.

(ii)
```
#include<stdio.h>
int check(int);
int main( )
{
    int i = 45, c;
    c = check(i);
    printf("\n %d", c);
    return 0;
}
int check(int ch)
{
    if(ch >= 45)
    return(100);
    else
    return(10 * 10);
}
```

Output:
100

(iii)
```
#include<stdio.h>
float circle(int);
int main( )
{
    float area;
    int radius = 1;
    area = circle(radius);
    printf("\n %f", area);
    return 0;
}
float circle(int r)
{
    float a;
    a = 3.14 * r * r;
    return(a);
}
```

Output:
3.000000

Q.7. Point out the errors, if any, in the following programs :

(i)
```
#include<stdio.h>
int addmult(int, int);
int main( )
{
    int i = 3, j = 4, k, l;
    k = addmult(i, j);
    l = addmult(i,j);
    printf("\n %d %d", k, l);
    return 0;
}
int addmult(int ii, int jj)
{
    int kk, ll;
    kk = ii + jj;
    ll = ii * jj;
    return(kk, ll);
}
```

Error:
A function cannot return 2 values at a time.

(ii)
```
#include<stdio.h>
int main( )
{
    float a = 15.5;
    char ch = 'c';
    print it(a, ch);
    return0;
}
printit(a, ch)
{
    printf("\n %f  %c", a, ch);
}
```

Error:
a and ch should be declared as float and char respectively in the function printit( ).

(iii)
```
#include<stdio.h>
void message( );
int main( )
{
    message( );
    message( );
    return 0;
}
void message( );
{
printf("\n C is a programming language");
}
```

Error :
Semicolon should not be present immediately after message( ) in the function definition.

Q.8. Write a function to calculate the factorial value of any integer entered through the keyboard.

Ans.
```
#include<stdio.h>
#include <conio.h>
long fact(int);
int main( )
{
    int num;
    long factorial;
    clrscr( );
    printf("Enter a number=");
    scanf("%d", &num);
    factorial = fact(num);
    printf("\n Factorial of %d = %ld", num,
                                factorial);
    getch( );
    return 0;
}
long fact(int num)
{
    int i;
    long factorial = 1;
    for(i = 1;  i <= num; i++)
    factorial = factorial * i;
    return(factorial);
}
```

Output:
Enter a number = 4
Factorial of 4 = 24

Q.9. Write a function to determine whether the year is a leap year or not.

Ans.
```
#include<stdio.h>
#include<conio.h>
void leap year(int);
int main( )
{
    int year;
    clrscr( );
    printf("Enter year:");
    scanf("%d", &year);
    leapyear(year);
    getch( );
    return 0;
}
void leapyear(int year)
{
if(year % 4 ==0 && year % 100 != 0 || year %
                                    400 = 0)
    printf("%d is leap year", year);
else
    printf("%d is not a leap year", year);
}
```

Output:
Enter year : 2004
2004 is leap year

Q.10. Write a function to obtain the prime factors of a number.

Ans.
```
#include <stdio.h>
#include <conio.h>
void prime(int);
int main( )
{
    int num;
    clrscr( );
    printf("Enter number:");
    scanf("%d", &num);
    prime(num);
```

```
        getch( );
        return 0;
}
void prime(int num)
{
        int i = 2;
        printf("Prime factors of %d are", num);
        while(num != 1)
        {
        if(num % i == 0)
        printf("%d", i);
        else
        {
                i++;
                continue;
        }
        num = num / i;
        }
}
```

Output:
Enter number : 24
Prime factors of 24 are 2, 2, 2 and 3

Q.11. Write a function to find largest of two numbers.

Ans.

```
#include<stdio.h>
void large(int a, int b)
{
        if(a < b)
            printf("Large is = %d," a);
        else
            printf("Large is = %d", b);
}
main( )
{
        int i, j;
        printf("Enter the value of a and b");
        scanf("%d %d", &j, &i);
        large(j, i);
        printf("Enter the value of a and b");
        scanf("%d  %d", &j, &i);
        large(j, i);
}
```

Output:
Enter the value of a and b 3 5
Large is = 5
Enter the value of a and b 12 10
Large is = 12

Q.12. Write a program to display equivalent binary number of given number using function.

Ans.

```
#include<stdio.h>
void main( )
{
        void bin(void);
        clrscr( );
        bin( );
        getch( );
}
void bin( )
{
        int n = 0, rem = 0, arr[5];
        int i = 0, j = 0;
        printf("\n Enter number:");
        scanf("%d", &n);
        printf("\n Binary equivalent of %d = ", n);
        while(n!= 0)
        {
                rem = n%2;
                arr[i] = rem;
                i++;
                n = n/2;
        }
        i--;
        for(j = i; j >= 0; j--)
        {
        printf("%d", arr[j]);
        }
}
```

Output:
Enter number : 10
Binary equivalent of 10 = 1010

Q.13. Write a program to print square series of a number till the position entered by the user.

Ans. #include<stdio.h>

```
void sqr(int);
main( )
{
        int pos = 0;
        printf("\n Enter last position");
        scanf("%d", &pos);
        sqr(pos);
        getch ( );
}
void sqr(int x)
{
        int i = 1;
        for(i = 1;  i <= x; i++)
        {
        printf("%d \ t", i * i);
        }
}
```

Output:
Enter last position 5
1 4 9 16 25

Q.14. Write a function to calculate sum of series 1 + 2 + 3 + ... + n. Input value of n and display the result.

Ans.
```
#include<stdio·h>
#include<conio·h>
main( )
{
        int i = 1, n;
        x = 0;
        clrscr( );
        printf("Enter the value of n");
        scanf("%d", &n);
        while(i<=n)
        {
          x = x + 1;
          i = i + 1;
        }
        printf("The value is %d", x);
        getch( );
}
```

Output:
Enter the value of n 5
The value is 15

Q.15. Write a program using function arm( ) to find whether entered no. is Armstrong or not.

$153 = 1^3 + 5^3 + 3^3$

Ans.
```
#include<stdio·h>
#include<conio·h>
void arm(int);
int main( )
{
        int num;
        clrscr( );
        printf("Enter  number :");
        scanf("%d", &num);
        arm(num);
        getch( );
        return 0;
}
void arm(int num)
        {
                int a,b,c,d;
                a = n/100;
                b = ((n/10) % 10);
                c = n % 10;
                d = a * a * a + b * b * b + c * c * c;
                if(n = = d)
                printf(" The Given Number is
                                Armstrong number");
                else
                printf("The Given Number is Not
                                Armstrong number");
                getch( );
        }
```

Output:
Enter number: 15 3
The Given Number is Armstrong number

Q.16. Write a program to find product of two numbers using function.

Ans.
```
#include<stdio.h>
int prod(int);
int main( )
{
        int p1, p2, p3;
        product=prod(p1, p2);
        printf("\n%d",product);
        return 0;
```

```
}
int prod(int p1, int p2)
{
    int p;
    p = p1 * p2;
    return(p);
}
```

Output:
12

Q.17. Print the following pattern using functions

@ @ @ @
@ @ @
@ @
@

Ans.

```
#include<stdio.h>
void pattern(int);
main( )
{
    int row = 0;
    clrscr( );
    printf("Enter maximum rows:");
    scanf("%d", &row);
    pattern(row);
    getch( );
}
void pattern(int r)
{
    int i = 0, j = 0;
    for(i = 1; i <= r; i++)
    {
        printf("\n");
        for(j = r; j >= i; j--)
        {
            printf("@");
        }
    }
}
```

PARAMETER PASSING IN FUNCTION

Q.18. Explain different parameter passing techniques with example.

OR Write a program using the function to swap two integers using the pass by reference concept.

OR Giving an example for each, discuss the call by value and call by reference method of parameter passing.

OR What are the parameters mechanisms in function?

Ans.

(i) Call by Value :

- When a function is called by passing values or variables to the called function, such function calls are called "call by value".
- Call by value is a function that make changes in function but can't pass it back to the main program.

Example :

```
main( )
{
int a=10, b=20;
printf("The values of a and b before swap: %d %d", a, b);
swapvalue(a,b);
        /*Here values of the variable is passing*/
printf("The values of a and b after swap: %d %d", a, b);
getch( );
}
swapvalue(int x, int y)
        /*Here x and y are formal variables*/
{
int temp;
temp=x;
x=y;
y=temp;
printf("%d %d",x,y);
}
```

Output:
The values of a and b before swap: 10 20
The values of a and b after swap: 10 20

- In above program, first the values 10 and 20 get assigns to variable 'a' and 'b' as shown below.

Variable Name →	a	b
Value →	10	20

- The swapvalue( )function is passing the values of variables a and b (actual arguments) to the

variable 'x' and 'y' (formal argument). So that the values of variable a and b gets copied to variables x and y.

- The moment when programmer declares temp variable it will contain garbage value as shown in following figure as a programmer is not assigning any value to temp variable.

Variable Name →	x	y	temp
Value →	10	20	Garbage value

- The statement temp=x; assigns the value of x to the variable temp and value of variable x remains same as given below.

Variable Name →	x	y	temp
Value →	10	20	10

Similarly the statement x=y; assigns the value of y to the variable x and value of y i.e. 20 remains same as given below

Variable Name →	x	y	temp
Value →	20	20	10

Also the statement y=temp; assigns the value of temp i.e.10 to the variable y as given below

Variable Name →	x	y	temp
Value →	20	10	10

- After this with the help of printf( ) function, programmer prints the value of variables x and y i.e. 20 and 10 respectively but these changes will not get reflected in the main( ) function as it is the concept of call by value.
- It means when execution control goes back to the main function, the changes that happens in called function will not get reflected in the calling function.
- Therefore the values of a and b remain same i.e. 10 and 20. So if programmer wants changes reflected in calling function, the concept of call by reference is useful.

(ii) Call by Reference :

- At the time of call, instead of passing variables or values if the reference of variable is passed then the function call is called as call by reference.
- In case of call by reference address of the variable is passed and therefore whatever changes that happened in function interchange get reflected in the address location and therefore gets reflected in the original function call in main.
- In other words, address is passed by using a symbol and the value is accessed by using symbol*.

Example :

```
main( )
{
int a=10, b=20;
printf("The values of a and b before swap: %d
                                        %d", a, b);

swapaddress(&a,&b);
    /*Here addresses of the variable is passing*/
printf("The values of a and b after swap %d %d", a,
b);
getch( );
}
swap(int*x, int*y)
/*Pointer variable get the address of the variables*/
{
int temp;
temp=*x;
*x=*y;
            /*Here changing the values a=10 by 20*/
*y=temp;
        /*Here changing the values b=20 by 10*/
}
```

Output:
The values of a and b before swap : 10 20
The values of a and b after swap : 20 10

- In above program, first the values 10 and 20 gets assigns to variable 'a' and 'b' as shown below.

Variable Name →	a	b
Value →	10	20
Address →	2000	3000

- The swapaddress( ) function is passing the addresses of variables a and b (actual arguments) to the variable 'x' and 'y' (formal argument). So that the addresses of variable a and b i.e. 2000 and 3000 respectively gets copied to variables x and y.
- The moment when programmer declare temp variable it will contain garbage value as shown in

following figure as programmer not setting any value to temp variable.

x	y	temp
2000	3000	Garbage Value

- The statement temp=*x; assigns the value of x means value at 2000 i.e. 10 to the variable temp as given below.

x	y	temp
2000	3000	10

- Similarly the statement *x=*y; assigns the value at y i.e. value at 3000 i.e. 20 to the value at 2000. Also the statement *y=t assigns the value of temp variable i.e. 10 to the *y(value at 3000) as given below

	a	b	
20	~~10~~	~~20~~	10
	2000	3000	

Now, the values of a and b gets change i.e. a=20 and b=10.

Q.19. Distinguish between call by reference and call by value.

OR What is the difference between call by value and call by reference? Give example.

Ans.

Sr. No.	Call by value	Call by reference
(1)	In call by value, both the actual and formal parameters will be created in different memory locations.	In call by reference if parameters are called by reference both will be created at the same location.
(2)	In call by value method, a compiler get a copy of the variable and thus changes made in the value in function will not be reflected back to the called function.	In call by reference method, the compiler didn't get any copy, but actually it works on the original copy and thus changes will be reflected back.
(3)	Using call by value, programmer can return only one value at a time.	Using call by reference, programmer can return more than one value at a time.
(4)	Call by value is less efficient than call by reference.	Call by reference works more efficiently than call by value.

(5)	**Example:**
	main()
	{
	int a=10, b=20 ;
	printf("The values of a and b before swap: %d %d",a, b);
	swapvalue(a,b);
	printf(" The values of a and b after swap:%d %d", a,b);
	getch();
	}
	swapvalue(int x, int y)
	{
	int temp;
	temp=x;
	x=y;
	y=temp;
	printf("%d %d",x,y);
	}
	Output:
	The values of a and b before swap : 10 20
	The values of a and b after swap : 20 10
		Example:
		main()
		{
		int a=10, b=20 ;
		printf("The values of a and b before swap: %d %d",a, b);
		swapaddress(&a,&b);
		printf("The values of a and b after swap %d %d", a,b);
		getch();
		}
		swapaddress(int *x, int *y)
		{
		int temp;
		temp=*x;
		*x=*y;
		*y=temp;
		}
		Output:
		The values of a and b before swap : 10 20
		The values of a and b after swap : 20 10

Q.20. Justify whether TRUE or FALSE.

In 'C' parameters are passed by reference.

Ans. True, The term pass by reference in C language means passing the address (memory location) of a variable to a function, rather than its value. Since the address is passed to a function called whatever modification is done by this called function, to the data (variable) against that address, gets reflected in the calling function. It is easily implemented in C parameter pass by reference.

Q.21. Explain return statements in C.

Ans.

- The return causes an immediate exit from a

function to the point from where the function is called. It may also be used to return one value per call.

- All functions, except those of type void, return a value. If no return statement is present, most compilers will cause a 0 to be returned. The return statement can be any one of the types as shown below:

  (i) return;
  (ii) return();
  (iii) return(constant);
  (iv) return(variable);
  (v) return(expression);
  (vi) return (conditional expression);
  (vii) return(function);

- The first and second return statements do not return any value and are just equal to the closing brace function. If the function reaches the end without using a return statement, the control is simply transferred back to the calling portion of the program without returning any value. The presence of an empty return statement is recommended in such situations.
- The third return statement returns a constant function.

  Example: if (x <= 10)
  return (1);

- The fourth return statement returns a variable to the calling function.

  Example: if (x <= 10)
  return (x);

- The fifth return statement returns a value depending upon the expression specified inside the parenthesis.

  Example:
  return (a + b * c);

- The sixth return statement returns a value depending upon the result of the conditional expression specified inside the parenthesis.

  Example:
  return (a >b ?1 : 2);

- The last return statement calls the function specified inside the parenthesis and collects the result obtained from that function and returns it to the calling function.

  Example:
  return (pow (4, 2));

  The parenthesis used around the expression or value used in a return statement is optional.

DIFFERENT CATEGORY OF FUNCTION

Q.22. What are a different Categories of Function? Explain with example.

Ans. Categories of Function:

We have some other types of functions where the arguments and the return value may be present or absent. Such functions can be categorized into:

(i) Functions with no arguments and no return value.
(ii) Functions with arguments and no return value.
(iii) Functions with no arguments but a return value.
(iv) Functions with arguments and return value.

(i) Functions with no arguments and no return value:

- Here, the called functions do not receive any data from the calling function and, it does not return any data to the calling function.
- Hence, there is no data transfer between the calling function and the called function.

Example:

This program illustrates the function with no arguments and no return value.

```
# include <stdio.h>
void read_name ();
main ()
{
    read_name ();
}
void read_value ()          /*No return value */
{
    char name [10];
    printf ("Enter your name: ");
    scanf ("%s", name);
    printf ("\nYour name is %s: ", name);
}
```

Output:
Enter your name: Herbert
Your name is: Herbert

(ii) Functions with arguments and no return value:

- Here, the called function receives the data from the calling function. The arguments and parameters should match in number, data type and order. But, the called function does not return and value back to the calling function. Instead, it prints the data in its scope only.
- It is one-way data communication between the calling function and the called function.

Example:

This program illustrates the function with arguments but has no return value.

```
#include <stdio.h>
void maximum (x, y);
main()
{
    int x, y;
    printf ("Enter the values of x and y: ");
    scanf ("%d %d", &x, &y);
    maximum (x, y);
}
void maximum (int p, int q)
        /* function to compute the maximum */
{
    if (p > q)
    else return;
}
printf ("\n maximum = %d", p);
printf ("\n maximum = %d" q);
```

Output:
Enter the values of x and y: 14 10
maximum = 14

(iii) Function with no arguments but return value:

- Here, the called function does not receive any data from the calling function. It manages with its local data to carry out the specified task.
- However, after the specified processing the called function returns the computed data to the calling function. It is also a one-way data communication between the calling function and the called function.

Example:

This program illustrates the function with no arguments but has a return value.

```
#include <stdio.h>
float total ();
main ()
{
    float sum; sum = total ();
    printf ("Sum = %f\n", sum);
}
float total ()
{
    float x, y; x = 20.0;
    y = 10.0;
    return (x + y);
}
```

Output:
Sum = 30.000000

(iv) Function with arguments and return value:

- When a function has arguments it receives data from the calling function and does some process and then returns the result to the called function.
- In this way, the main() function will have control over the function.

Example:

This program illustrates the function with arguments and return value. (Program to find the sum of first N natural number)

```
#include <stdio.h>
int sum (int x);
void main ()
{
    /* function prototype*/
    int n;
    printf ("Enter the limit: ");
    scanf ("%d", &n);
    printf ("sum of first %d natural numbers is:
                                %d", n, sum(n));
}
int sum(int x)
```

```
{
    int i, result = 0
    for (i=1; i<= x; i++)
    result += i; return (result);
}
```

Output:
Enter the limit: 5
Sum of first 5 natural numbers is: 15

The main() is the calling function which calls the function sum(). The function sum() receives a single argument. Note that the called function (i.e., sum ()) receives its data from the calling function (i.e., main()). The return statement is employed in this function to return the sum of the n natural numbers entered from the standard input device and the result is displayed from the main() function to the standard output device. Note that int is used before the function name sum() instead of void since it returns the value of type int to the called function.

Q.23. What are important things one should consider while calling a function?

Ans.

- Parentheses are compulsory after the function name.
- The function name in the function call and the function definition must be the same.
- The type, number, and sequence of actual and formal arguments must be the same.
- A semicolon must be used at the end of the statement when a function is called.
- The number of arguments should be equal to the number of parameters.
- There must be a one-to-one mapping between arguments and parameters. i.e. they should be in the same order and should have the same data type.
- Same variables can be used as arguments and parameters.
- The data types of the parameters must match or be closely compatible. It is very risky to call a function using an actual parameter that is floating-point data if the formal parameter was declared as an integer data type. We can pass a float to a double, but should not do the opposite. We can also pass a short int to an int, etc. But we should not pass any type of integer to any type of floating-point data or do the opposite.

Q.24. What is the Nested function? Explain with example.

Ans. Nested Functions:

When one or more functions are utilized under a particular function, it is known as nesting function in C programming language.

```
int main( )
{    /*main function in C*/
return_type function_name();
/*Call function under the main function*/
return 0;
}
return_type function_name()
{
/*user defined function*/
/*statements of function*/
}
/*nesting function in C language*/
```

Example:

```
#include <stdio.h>
void func1();
void func2();
void main()
{
    printf ("\n Inside main function");
    func1();
    printf ("\n Again inside main function");
}
void func1()
{
    printf ("\n Inside function 1");
    func2();
    printf ("\n Again inside function 1");
}
void func2()
{
    printf ("\n Inside function 2");
}
```

Output:

Inside main function Inside function 1
Inside function 2 Again inside function 1
Again inside main function

Uses two functions func1() and func2() other than the main() function. The main() function calls the function func1() and the function func1() calls the function func2().

Example:

Find cube value using the nested function

```
#include <stdio.h>
#include <stdlib.h>
int cube(int a)          /*cube function*/
{
return a*a*a;
}
void display()       /*Function to display*/
{
int i;
for(i=1; i<=10; i++)
{
printf("cube value of %d is %d\n",i,cube(i));
} /*cube function called inside the display function*/
}
int main()
{
display();
getch();
return 0;
}
```

Output:

cube value of 1 is 1
cube value of 2 is 8
cube value of 3 is 27
cube value of 4 is 64
cube value of 5 is 125
cube value of 6 is 216
cube value of 7 is 343
cube value of 8 is 512
cube value of 9 is 729
cube value of 10 is 1000

PASSING ARRAY TO FUNCTION

Q.25.Explain function with an array in detail.

Ans. Function with Array:

Generally, the C convention is the call-by-value. An exception is where arrays are passed as arguments. We can pass either an entire array as an argument or its elements.

Arrays in Functions:

- An array name represents the address of the first element in that array. So, arrays are passed to functions as pointers (a pointer is a variable, which holds the address of other variables).
- When an array is passed to a function, only the address is passed, not the entire array. When a function is called with just the array name, a pointer to the array is passed. In C, an array name without an index is a pointer to the first element. This means that the parameter declaration must be of a compatible pointer type. There are three ways to declare a parameter which, is to receive an array pointer:

Example:

void display (int num[10]);

in which case the compiler automatically converts it to a pointer.

As an unsized array:

void display (int num[ ]);

which is also converted to a pointer.

As a Pointer:

void display (int *num);

which, is allowed because any pointer may be indexed with [ ] as if it were an array.

An array element used as an argument is treated like any other simple variable, as shown in the example below:

Example:

```
main ()
{
    ...
    display (t [a]);
    ...
}
```

```
void display (int num)
{
      printf ("%d", num);
}
```

C Function to Sort the Array:

```
<include<stdio.h>
void Bubble_Sort(int[]);
void main( )
{
   int arr[10] = {10, 9, 7, 101, 23, 44, 12, 78, 34,
   23};
   Bubble_Sort(arr);
}
void Bubble_Sort(int a[]) /*array a[] points to array*/
{
   int i, j, temp;
      for (i = 0; i<10; i++)
      {
         if(a[j] < a[i])
         {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
         }
      }
}
printf("Printing Sorted Element  List...\n");
for (i = 0; i<10; i++)
  {
      Printf(%d\n", a[i]);
  }
}
```

Output:

```
7
9
10
12
23
23
34
44
78
101
```

RECURSION

Q.26. What is Recursion? Explain in detail with an example.

Ans. Recursion :

A programming technique in which a function calls itself. It is one of the most effective techniques in programming.

Example :

Consider evaluation of factorials of a given number :

```
factorial(num)
{
int fact;
if(num= =1)
return(1);
else
fact=num * factorial (num –1);
return(fact);
}
```

Recursion Process :

- Assume n=3.
  since the value of n is not 1, the statement
  fact=num * factorial(num –1);
  will be executed with n=3 that is,
  fact=3 * factorial(2); will be evaluated.
- Now fact includes a call to factorial with num=2. This call will return the value : 2 * factorial(1) Once again, factorial is called with num=1. This time the function returns 1. The sequence of execution is :

  Fact = 3 * factorial(2)
  = 3 * 2 * factorial(1)
  = 3 * 2 * 1
  = 6.
- Recursion functions can be effectively used to solve problems where the solution is expressed in terms of successively applying the same solution to subsets of the problem.
- When one writes recursive functions, one must have an if statement somewhere to force the function to return without the recursive call being executed. Otherwise, the function will never return.

Advantages of Recursive Function :

(i) Although most of the time a problem can be solved without recursion, in some situations in programming, it is must to use recursion.

(ii) The recursion is very flexible in a data structures like stacks, queues, linked lists and quick sort.

(iii) Using recursion, the length of the program can be reduced.

Disadvantages of Recursive Function :

(i) It requires extra storage space. The recursive calls and automatic variables are stored on the stack. For every recursive calls separate memory is allocated to automatic variables with the same name.

(ii) If the programmer forgets to specify the exit condition in the recursive function, the program will execute out of memory. In such a situation user has to press ctrl + break to pause and stop the function.

(iii) The recursion function is not efficient in execution speed and time.

(iv) If possible, try to solve a problem with iteration instead of recursion.

Q.27. Write a program to generate the Fibonacci series for a given number using a recursive function.

Ans.
```
#include <stdio.h>
int fibonacci(int i)
{
if(i == 0) {
return 0;
}
if(i == 1)
{
return 1;
}
return fibonacci(i-1) + fibonacci(i-2);
}
int main() {
int i;
for (i = 0; i< 10; i++)
{
printf("%d\t\n", fibonacci(i));
}
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

0
1
1
2
3
5
8
13
21
34

Q.28. Write a program to obtain the sum of the first ten terms of the following series using recursion $x - x^3/3! + x^5/5! .....$

Ans.
```
float sum,term,nr;
int dr,p,n;
main( )
{
float x;
fsine(float,int);
float s;
clrscr( );
printf("Enter the value of x and n");
scanf("%f %d",&x,&n);
x=x*3.1415/180;
sum=0;
p=3;
nr=x;
dr=1;
s=fsine(x,n);
printf("sin(x)=%f",sum);
getch( );
}
float fsine(float x, int n)
{
```

```
float value;
if(n==0)
return sum;
else
{
term=nr/dr;
sum=sum+term;
nr=nr*(-x)*x;
dr=dr*(p-1)*p;
p=p+2;
value=fsine(x,n-1);
return(value);
}
}
```

Output:

```
Enter the value of x and n
1
2
six(x) = 0.017452
```

PROGRAMS FOR PRACTICE:

Q.1. Write a function that can be called to find the largest element of an m by n matrix.

Q.2. Write a program to find the factorial of a positive integer using recursion.

Q.3. Write a recursive function of find the sum of two non-negative number.

Q.4. Write a program to swap the two numbers using functions without using third variable.

Q.5. Write a program to find factorial of a number using function.

POINTS TO REMEMBER:

(1) Function is a group of statements that have been given a name.

(2) A function can be called multiple times to provide reusability and modularity to the C program.

(3) Function makes the lengthy and complex program easy and in short forms.

(4) Function increases the execution speed of the program and makes the programming simple.

(5) In C programming, functions are divided into two types:
Library function and user-defined function.

(6) User defined function divided into 3 activities such as function declaration. Function definition, function call:
- Function declaration means writing a name of program. It is a compulsory part for using function in code.
- Function definition means just writing the body of function.
- Function call means calling a function whenever it is required in a program.

(7) When a function is called by passing values to the called function such function calls are called as call by value.

(8) At the time of call, instead of passing variables or values if the reference of variable is passed then the function call is called as call by reference.

(9) When one or more functions are utilized under a particular function, it is known as nesting function in C programming language.

(10) In C programming, we can pass on entire array to function.

(11) Recursion is the process of repeating items in a self-similar way.

(12) If a program allows calling a function inside the same function, then it is called a recursive call of the function.

UNIT - IV
STRUCTURE AND POINTERS

SYLLABUS

(A) **Structure:** Structure, Defining Structures and Array of Structures.

(B) **Pointers:** Idea of pointers, Defining pointers, Use of Pointers in self-referential structures, notion of linked list (no implementation).

(A) STRUCTURE

INTRODUCTION

A structure is a convenient tool for handling a group of logically related data items. For example, it can be used to represent a set of attributes, such as student_name, roll_number and marks. The concept of a structure is analogous to that of 'record' in many other languages. More examples of such structures are:

Time	:	Seconds, minutes, hours.
Date	:	Day, month, year.
Book	:	Author, title, price, year.
City	:	Name, country, population.
Address	:	Name, door-number, street, city
Inventory	:	item, stock, value.
Customer	:	Name, telephone, city, category.

Structure helps to organize complex data in a more meaningful way. It is a powerful concept that we may often need to use in our program design.

DEFINING STRUCTURE

Q.1. What is Structure? Explain in detail.

OR Define Structure. Explain the syntax of structure with an example.

Ans. Structure:

- A structure is a user-defined data type. Using a structure we can define a new type of data considerably more complex than the types we have been using.
- A structure is a collection of variables referenced under one name, providing a convenient means of keeping related information together. The variables which make up a structure are called structure elements.
- In the following, the C keyword "struct" tells the compiler that a structure template is being defined.
- Dot (.) operator is used to access the data of the memory variable in the structure. When the programmer declares the variable, the memory will be occupied.

Syntax:

```
struct <structure name>
{
    structure element 1;
    structure element 2;
    ........
    structure element n;
};
struct <structure name> variable 1,
                variable 2 ............ variable n;
```

where struct is a keyword.

Example:

```
main()
{
struct Data
{
int i;
char ch;
};
struct Data d1;
clrscr();
d1.a=100;                    /*Assigning values*/
d1.ch='P';
printf("%d %c",d1.a,d1.ch);
getch();
}
```

Output: 100 P

Features:

(i) Structure can store dissimilar data types.

(ii) The structure variables are not stored contiguously.

Q.2. Define a structure within a structure for student data with the field as name, age, year, data-of-birth and branch.

Ans.
```
struct student
{
    char name [25];
    int age;
    int date of birth;
};
struct student info
{
    int year ;
    char branch [20];
    struct students;
};
```

Q.3. What is the necessity of using Structure? Write a 'C' program using structure. And find out the total number of bytes required to store the structure you have considered in your program.

Ans. Necessity of Structure:

- The basic necessity of using structure is to store collection of dissimilar data types.
- The sizeof() operator gives the size of the variable or a structure that how many bytes are occupied by the memory.
- The following program shows the total number of bytes required to store the structure that we declared.

```
main()
{
struct Data
{
int a;
char ch;
};
struct Data d1;
clrscr();
printf("\n%d",sizeof(d1));
printf("\n%d",sizeof(struct Data));
getch();
}
```

Output:
3
3

- In the above program, the name of the structure is Data having integer variable and character variable occupying 2 bytes and 1 byte respectively. Therefore the total number of bytes required to store the given structure is 3 bytes.

Q.4. Define a structure student that consists of three fields: student name, roll number and percentage. Write a C program to read the information about 80 students from the keyboard and display the same on the screen.

Ans.
```
#include <stdio.h>
#include <conio.h>
main( )
{
int i;
struct student
{
char name[20];
int rollno;
float percentage;
}s[80];
clrscr( );
printf("Enter the student information:");
for(i=0;i<=79;i++)
{
scanf("%s %d %f",&s[i].name,&s[i].rollno,&s[i]
                                    .percentage);
}
printf("The information about student is:");
for(i=0;i<=79;i++)
{
printf("\t %s \t %d \t %f", s[i].name, s[i].age,
                                    s[i].phone);
}
getch( );
}
```

Q.5. Define a structure called cricket game that stores the following information:

player_name

team_name

batting_average

player_age

Using cricket game, declare an array players with 100 elements and write a program to read information about all the 100 players and display a team-wise list containing names of players with their batting average and age.

Ans.
```
#include <stdio.h>
#include <conio.h>
main( )
{
int i;
struct cricket
{
char playername[50];
char teamname[50];
float batavg;
int age;
}c[100];
clrscr( );
printf("Enter information for teams:");
for(i=0;i<=99;i++)
{
scanf("%s %s %f %d", c[i].playername,
        c[i].teamname,&c[i].batavg,&c[i].age);
}
printf("The team details:");
for(i=0;i<=99;i++)
printf("\t%s \t%s \t%f \t%d\n", c[i]. playername,
c[i].teamname, c[i].batavg,c[i].age);
getch( );
}
```

Q.6. Give different ways to declare structures. When does the compiler know to reserve space in memory for members of the structure?

Ans. Different ways of declaring a structure is as follows:

(i)
```
struct foo {
short a;
int b;
float c;
};
```

(ii)
```
typedef struct baz {
short a;
int b;
float c;
} baz;
```

- Structures can be embedded.
- Structure within another structure is called an embedded structure.
- Embedded structures can be declared in 2 ways:

(a) Structure may be completely defined within another structure.

Example:
```
struct emp
{
int empno;
char emp_name[30];
int salary;
struct date_join
{
    int day;
    int month;
    int year;
};
};
```

Memory is allocated for the members of the structure at the time of structure variable declaration.

(b) There may be a separate structure, the embedded structure declared first and the other structure next.

Example:
```
struct emp
{
    int empno;
    char emp_name[30];
    int salary;
    struct date_join
};
```

```
struct date_join
{
    int day;
    int month;
    int year;
};
```

Q.7. How structure elements are stored in memory? State at least two features that are not possible with an array but could be accomplished with structure. Illustrate your answer with an example.

Ans. Like array elements structure elements are not stored in a contiguous manner, they are stored in a random fashion where the memory locations are free. Two features that are not possible with an array but could be accomplished with the structure are:

(i) In structure, variable can store elements of dissimilar data types in a single entity whereas an array allows only to store elements of similar data types.

(ii) Linked list is a very common type of data structure. It is used to store similar types of data in memory. Arrays are also used for storing similar types of data in memory, but the array occupies contiguous memory for storing data elements.

For example:

```
#include <stdio.h>
#include <conio.h>
struct student
{
 int id;
 char *name;
 float percentage;
} student1, student2, student3;
int main()
{
 struct student st;
 student1.id=1;
 student2.name = "Angelina";
 student3.percentage = 90.5;
 printf(" id is: %d \n", student1.id);
 printf("Name is: %s \n", student2.name);
 printf("Percentage is: %f \n",
                    student3.percentage);
 getch();
 return 0;
}
```

Output:
Id is 1.
Name is: Angelina
Percentage is: 90.50000

Q.8. Write a program to implement a structure of employees-details which includes Name, Birth date and salary.

Ans.

```
#include <stdio.h>
void main( )
{
struct employee_details
{
char name[20];
char birthdate[20];
int salary;
};
struct employee_details emp;
clrscr();
printf("Enter Name:");
scanf("%s",&emp.name);
printf("\nDate of birth dd/mm/yy:");
scanf("%s",&emp.birthdate);
printf("\nEnter salary:");
scanf("%d",&emp.salary);
printf("Employee %s was born on
    %s",emp.name,emp.birthdate);
printf("and his salary is %d",emp.salary);
getch();
}
```

Output:
Enter Name:Amit
Date of Birth dd/mm/yy:10/10/13
Enter salary:40000
Employee Amit was born on 10/10/13 and his salary is 40000

ARRAY OF STRUCTURES

Q.9. Write a short note on Array of Structure.

Ans. Array of structure:

- An array is a collection of the same type of elements. Structure is a collection of different types of elements.
- To store more than one record in the memory, use an array of structures. This array gets stored continuously in the memory. Due to continuous storage, access to this array becomes easy.

  An array of structures is declared in the usual way:

  struct personal_data my_struct_array[100];
- The members of the structures in the array are then accessed by statements such as the following:

  The value of a member of a structure in an array can be assigned to another variable, or the value of a variable can be assigned to a member.
- For example, the following code assigns the number 1974 to the year_of_birth member of the fourth element of my_struct_array:

  my_struct_array[3].year_of_birth = 1974;

  (Like all other arrays in C, struct arrays start their numbering at zero.)
- The following code assigns the value of the year_of_birth member of the fourth element of my_struct_array to the variable yob:

  yob = my_struct_array[3].year_of_birth;
- Finally, the following example assigns the values of all the members of the second element of my_struct_array, namely my_struct_array[1], to the third element, so my_struct_array[2] takes the overall value of my_struct_array[1].

  my_struct_array[2] = my_struct_array[1];

Example:

```
#include <stdio.h>
struct
{
    char initial; int age; int grade;
}
kids[12];
int main()
{
    int index;
    for (index = 0 ; index < 12 ; index++)
    {
        kids[index].initial = 'A' + index;
        kids[index].age = 16;
        kids[index].grade = 84;
}
kids[3].age = kids[5].age = 17; kids[2].grade
        =kids[6].grade = 92; kids[4].grade = 57;
kids[10]=kids[4]; /* Structure assignment
                        for kids[10]=kids[4]; */
{
    (index = 0 ; index < 12 ;index++)
    printf("%c is %d years old and got a grade
            of %d\n", kids[index].initial,
        kids[index]. age, kids[index].grade);
}
return 0;
}
```

Output:

```
A is 16 years old and got a grade of 84
B is 16 years old and got a grade of 84
C is 16 years old and got a grade of 92
D is 17 years old and got a grade of 84
E is 16 years old and got a grade of 57
F is 17 years old and got a grade of 84
G is 16 years old and got a grade of 92
H is 16 years old and got a grade of 84
I is 16 years old and got a grade of 84
J is 16 years old and got a grade of 84
K is 16 years old and got a grade of 57
L is 16 years old and got a grade of 84
```

Q.10. Declare a structure add with fields add1, add2, city and state. Use this structure to declare another structure student having fields Roll no, name and address. Declare an array of structure student with 10 elements.

Ans.

```
#include <stdio.h>
{
struct add
```

```
{
int add1[20];
int add2[20];
char city[20];
char state[20];
struct student;
};
struct student
{
int rollno;
char name[20];
char address[30];
};
struct student stud[10];
getch();
}
```

Q.11. A blood bank stores the information about the donors as follows:
Name, Address, Age, Weight, Blood group.
Write a program using a structure that print the records of donor with age less than 30 and blood group equal to 'O+' in the list of 1000 donors.

Ans.

```
#include <stdio.h>
void main()
{
struct donor
{
char name[20];
int age;
char address[30];
char bloodgroup[2];
};
int i;
struct donor d[1000];
clrscr();
printf("Name\tAge\tAddress\tBlood Group\n");
for(i=0;i<1000;i++)
{
scanf("%s%d%s%s",&d[i].name,&d[i].age,&d[i].
                    address,&d[i].bloodgroup);
}
for(i=0;i<1000;i++)
{
if(d[i].age<30&&strcmp(d[i].bloodgroup,
                                    "O+")==0)
{
printf("%s\t%d\t%s\t%s",d[i].name,d[i].age,d[i].
                    address,d[i].bloodgroup);
}
}
getch();
}
```

Output:

Name	Age	Address	Blood Group
Vijay	25	Nagpur	O+
Amit	34	Nagpur	A+
Vivek	45	Nagpur	B+
Vijay	25	Nagpur	O+

Q.12. Declare an array of structure consisting of name and phone number of 10 elements. Write a C function getname() which receives a pointer to the above array, size and phone number as arguments. It should search the phone number in the array and return a pointer to the corresponding name.

Ans.

```
#include <stdio.h>
#include <conio.h>
struct info
{
char name[10];
int phone;
};
struct info person[10];
void main()
{
int i,n,ph;
struct info(*st_ptr);
st_ptr=&person;
clrscr();
printf("Enter the size of array:");
scanf("%d",&n);
printf("\nEnter the name and phone number:");
```

```
for(i=0;i<n;i++)
{
scanf("%s %d", & person[i].name, &
                                person[i]. phone);
}
printf("\nEnter the phone number to be searched:");
scanf("%d",&ph);
getname(st_ptr,n,ph);
getch();
}
getname(struct info *p,int size,int phno)
{
int j=0;
for(j=0;j<size;j++)
{
if(phno==p->phone)
{
printf("Name is:%s",p->name);
}
else
{
printf("Not found");
}
}
}
```

Output:
Enter the size of array:2
Enter name and phone number: Amit 1234
Vijay 4567
Enter the phone number to be searched:1234
Name is: Amit.

PASSING STRUCTURE TO FUNCTION

Q.13. Write a short note on passing structure to function in brief with a suitable example.

Ans.

- When we pass a member of a structure to a function, we are passing the value of that member to a function.
- Structure can be passed to the function using following ways:

(i) By passing individual members of the structure.

Example:

```
main( )
{
    struct book
    {
        char name[25];
        int callno;
    };
    struct book b1={"Let us c",101};
    display(b1.name, b1.callno);
    getch( );
}
display(char *s, int n)
{
    printf("\n%s d",s,n);
}
```

Output:
Let us c 101

(ii) By passing the entire structure variable.

Example:

```
struct book
{
char bname[20];
};
main( )
{
    struct book b1 = {"Let us C"};
    display(b1);
    getch( );
}
display(struct book b1)
{
    printf("The book name is %s",b1.name);
}
```

Output:
The book name is Let us C

Example:

```
#include <stdio.h>
struct struct_type
{
 int a, b;
 char ch;
};
```

```
void f1(struct struct_type parm)
{
 printf("%d", parm.a);
}
int main(void)
{
 struct struct_type arg;
 arg.a = 1000;
 f1(arg);
 return 0;
}
```

Output:
1000.

Q.14. Define a structure called time with hour, min and seconds. Write a function time update that update the time by 1 second. Ask the current time from the user and pass that structure as an argument to the time update function and also write a function to display time.

Ans.

```
#include <stdio.h>
struct time
{
int hour;
int min;
int sec;
};
struct time timeupdate(struct time);
void main()
{
struct time currtime;
struct time newtime;
clrscr();
printf("Enter the time:");
scanf("%d:%d:%d",&currtime.hour,&currtime.
                        min, &currtime.sec);
newtime=timeupdate(currtime);
printf("\nNewtime:%d:%d:%d",newtime.hour,ne
                        wtime.min,newtime.sec);
getch();
}
struct time timeupdate(struct time t)
{
t.sec=t.sec+1;
if(t.sec==60)
{
t.min=t.min+1;
t.sec=0;
}
if(t.min==60)
{
t.hour=t.hour+1;
t.min=0;
t.sec=0;
}
return t;
}
```

Output:
Enter the time:12:10:01
New time: 12:10:02

Q.15. Can we pass a structure to a function by value or by reference method?

OR What are different ways to pass structure information in function?

Ans. Yes, we can pass a structure to a function by value as well as reference as follows:

Examples:

(i) Structs are passed by value, just like primitives.

```
void func(struct Foo foo)
{
    foo.x = 56;
    foo.array[3]=55;
}
void main( )
{
    struct Foo f;
    f.x = 54;
    f.array[3]=9;
    func(f);
}
```

(ii) Structs are passed by reference.

```
void func(struct Foo* p_foo)
{
```

```
        p_foo->x = 56;
        p_foo->array[3]=55;
    }
    void main()
    {
        struct Foo f;
        f.x = 54;
        f.array[3]=9;
        func(&f);
    }
```

Q.16. Write a C program to illustrate the method of sending an entire structure as a parameter to a function.

Ans.

```
struct book
{
char bname[20];
};
main()
{
struct book b1={"Let us C"};
display(b1);
getch();
}
display(struct book b1)
{
printf("The book name is %s",b1.bname);
}
```

Output:
The book name is Let us C

Q.17. Write a program showing an example of pointers to structure.

Ans.

```
#include <stdio.h>
#include <conio.h>
int main( )
{
 struct st
 {
  int id;
  char *name;
  char *address;
 };
 struct st employee, *stptr;
 stptr = &employee;
 stptr->id = 1;
 stptr->name = "Angelina";
 stptr->address ="Rohini,Delhi";
 printf("Employee Information:
 Id=%d\nName=%s\nAddress=%s\n",
        stptr->id, stptr->name, stptr->address);
 getch( );
 return 0;
}
```

Output:
Employee Information:
Id=1
Name=Angelina
Address=Rohini, Delhi.

Q.18. Write the difference between array and structure.

Ans. Differences between arrays and structures:

Sr. No.	Arrays	Structures
(1)	An array is a single entity representing a collection of data items of the same data types.	A structure is a single entity representing a collection of data items of different data types.
(2)	Individual entries in an array is called elements.	Individual entries in a structure is called members.
(3)	An array declaration reserves enough memory space for its elements.	The structure definition reserves enough memory space for its members.
(4)	There is no keyword to represent arrays but the square braces [] preceding the variable name tells us that we are dealing with arrays.	The keyword struct tells us that we can dealing with structures.
(5)	Initialization of elements can be done during array declaration.	Initialization of members can be done only during structure definition.

(6)	The elements of an array are stored in sequence of memory locations.	The members of a structure is not stored in sequence of memory locations.
(7)	The array elements are accessed by its followed by the square braces [] within which the index is placed.	The members of a structure are accessed by the dot operator.
(8)	Its general format is data type variable name [size];	Its general format is: struct <structname> { data_type structure member 1; data_type structure member2; . . . data_type structure memberN; } structure variable;
(9)	**Example:** int sum [100];	**Example:** struct student { char studname[25]; int rollno; } stud1;

(B) POINTERS

INTRODUCTION

A pointer is a derived data type in C language. A pointer is nothing but a variable that contains an address of a location in memory. One can declare the variable ptr is a pointer to an integer as,

int *ptr;

Pointers are a very useful and important feature of the C language. Pointers are used to reduce the complexity of programs. Pointers are very useful for the fast execution of programs.

DEFINING POINTERS

Q.19.Explain Pointer.

OR What do you mean by Pointers?

Ans. Pointer:

- A pointer is a derived data type in C language. A pointer is a programming language data type whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address.
- By making use of a pointer, one can access and change the contents of the memory location.
- A pointer is a variable that contains the address in memory of another variable. One can have a pointer to any variable type.
- The unary or monadic or address of operator '**&**' gives the **"address of a variable"**.
- The indirection or dereference operator or Value at address '*' gives the **"value of an object pointed to by a pointer"**.
- **Example:** y = *p

Q.20.How pointers are declared and initialized?

Ans.

- A pointer variable contains the memory location of another variable.
- One begins the declaration of a pointer by specifying the type of data stored in the location identified by the pointer.
- The asterisk tells the compiler that you are creating a pointer variable.
- Finally one gives the name of the pointer variable. The pointer declaration syntax is as shown below:

  type * variable name;
- **Example:**

  int *ptr;

  float *string;
- A pointer can be assigned the address of an ordinary variable.
- Consider the effect of the following code:

  int x =1, y = 2;

  int *ip; ***/*declaration of integer pointer*/***

```
ip = &x;   /*ip stores the address of x means it is
                                pointer initialization*/
y = *ip;
x=ip;
*ip=3;
```

Assume that variable x resides at memory location 100, y at 200 and ip at 1000. A pointer is a variable and thus its values need to be stored somewhere.

```
int x = 1, y = 2;
int *ip;
ip= &x;
```

	x	y	ip
	1	2	100
	100	200	1000

y = *ip;

	x	y	ip
	1	1	100
	100	200	1000

x = ip;

	x	y	ip
	100	1	100
	100	200	1000

*ip = 3

	x	y	ip
	3	1	100
	100	200	1000

- Now the assignments x = 1 and y = 2 load these values into the variables, ip declared to be a pointer to an integer and is assigned to the address of x(&x). So ip gets loaded with the value 100 as shown in the above diagram.
- Next y gets assigned to the contents of ip. In this example ip currently points to memory location 100, the location of x. So y gets assigned to the values of x which is 1.
- Now assignment of the current value of ip to x is carried out. The value of ip at this instant is 100. Therefore x will hold 100.
- Finally *ip=3 means the assignment of a value to the contents of a pointer (*ip).

POINTER VARIABLE

Q.21. Define Pointer Variable.

OR What is the function of the Pointer Variable?

Ans. Pointer Variable:

- A variable that holds the address of another variable is called a pointer variable.
- As a pointer is a variable, its value is also stored in another memory location. Any variable declared in a C program has two components:

(i) **Address of the Variable:**
To print the address of a variable, the 'Address of'(&) operator is used.

(ii) **Value stored in the Variable:**
To print the value at address, the 'Value at address'(*) operator is used.
The Address of and Value at address operators both together are known as pointer operators.

- **Example:**
int i = 10;
The above declaration tells the C compiler for:

(i) Allocation of space in memory to store the value.
(ii) Assigning the name x with this memory location.
(iii) Storing the value 10 at this location.

- It can be represented with the following figure:

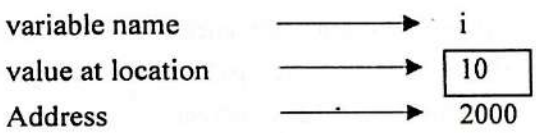
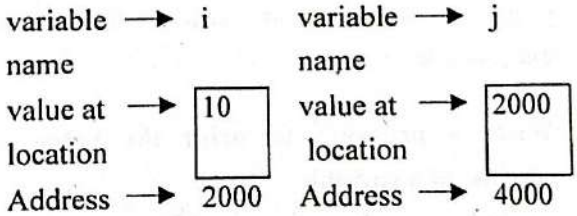


Fig. Representation of a variable

In above figure, the address 2000 is assumed, it may be some other address. Let the address of variable i i.e.2000 be assigned to variable j and let us assume that the address of variable j is 4000. In other words, one can say that variable 'j' points to x. So variable j is called 'pointer variable'.

- It can be shown diagrammatically as below:



As it is directly seen that variable j is holding the address of variable i. Therefore the declaration of variable j in any C program would be int *j;

Q.22. What are the uses of Pointer Variable?

OR What is the use of pointers in 'C'?

OR What are the advantages of using Pointer?

Ans. Advantages:

- A pointer enables us to access a variable that is defined outside the function.
- Pointers are more efficient in handling the data tables.
- Pointers reduce the length and complexity of program.
- They increase the execution speed.
- The use of pointer array to character string result in saving of data storage space in memory.

Uses:

- Pointers are more efficient in handling arrays and data tables.
- Pointers can be used to pass information back and forward between a function and its reference point.
- Pointers can be used to return multiple values from a function via function parameters.
- Pointers permit references to functions and thereby allow the passing of functions as parameters to other functions.
- Pointers provide an alternate way to access individual array elements.
- For saving the storage space by using the pointer arrays for character strings.
- Dynamic data structures such as structures, linked lists, stacks, queues and trees can be easily manipulated using pointers.
- Pointers are used for reducing the size and complexity of programs, for fast execution of programs and in turn reduce the program execution time.

Q.23. Write a program to print the value and address of a variable.

Ans.
```
#include <stdio.h>
#include <conio.h>
main()
{
 int i=10;
 clrscr();
 printf("\nThe value of i=%d",i);
 printf("\nThe address of i=%d",&i);
 printf("\nThe value of i=%d",*(&i));
 getch();
}
```

Output:
The value of i = 10
The address of i = 2000
The value of i = 10

Q.24. Write a program which shows the use of pointer operators.

Ans.
```
#include <stdio.h>
#include <conio.h>
main()
{
 int i=10;
 int *j,**k;
 clrscr();
 printf("\nThe value of i=%d",i);
 printf("\nThe address of i=%d",&i);
 printf("\nThe value of i=%d",*(&i));
 j=&i;
 printf("\nThe address of j=%d",&j);
 printf("\nThe value of j=%d",j);
 printf("\nThe value of j=%d",*(&j));
 k=&j;
 printf("\n%d \n%d \n%d \n%d",k,&k,*k,*&k);
 getch();
}
```

Output:
The value of i = 10
The address of i = 2000
The value of i = 10
The address of j = 4000
The value of j = 2000
The value of j = 2000
4000
6000
2000
4000

POINTER OPERATOR

Q.25. Explain different types of pointer operators in 'C'.

Ans. There are two types of pointer operators:

(i) Ampersand (&) operator,

(ii) Asterisk (*) operator.

(i) Ampersand (&) Operator:

- The address operator is used to access the address of a variable.
- The operator '&' immediately preceding a variable return the address of a variable associated with it.
- For example, the statement
  P = &quantity;
  would assign the address 5000 to the variable P.
- This operator is also known as address of.
- The '&' operator can be used only with a simple variable or an array element.

Example:

```
main( )
{
  char a;
  int x;
  float p, q;
  a = 'A';
  x = 125;
  p = 10.25, q = 18.76;
  printf("% c is stored at addr % u\n", a, &a);
  printf("%d is stored at addr % u\n", a, &x);
  printf("%f is stored at addr % u\n", p, &p);
  printf("%f is stored at addr % u\n", q, &q);
  getch ( );
}
```

Output:

```
A is stored at addr 4436
125 is stored at addr 4434
10.250000 is stored at addr 4442
18.760000 is stored at addr 4438
```

(ii) Asterisk (*) Operator:

- To access the value of the variable using the pointer, the unary operator '*' (asterisk), usually known as the indirection operator is used.
- Indirection operation is also known as dereferencing operator.
- Consider the following statements:
  int quantity, *p, n;
  quantity = 179;
  p = &quantity;
  n = *p;
- *p returns the value of the variable quantity, because p is the address of quantity. Thus, the value of n would be 179 when the operator '*' is placed before a pointer variable in an expression, the pointer returns the value of the variable of which the pointer value is the address.

Example:

```
main()
{
  int a = 8;
  int b;
  int *pa;
  pa = &a;
  printf("value of a %d", a);
  printf("value of a through pa %d", *pa);
  b = *pa;
  printf("value of b %d", b);
  getch ( );
}
```

Output:

```
value of a 8
value of a through pa 8
value of b 8
```

SELF REFERENTIAL STRUCTURES

Q.26. Explain Self-Referential Structure with its different types.

Ans. Self-Referential Structures:

- Self-Referential structures are those structures that have one or more pointers that point to the same type of structure, as their member.

Self-Referential Structures

```
struct node{
        int data1;
        char data2;
        struct node*  link;
};
```

- In other words, structures pointing to the same type of structures are self-referential.

Example:

```
struct node
{
    int data1;
    char data2;
    struct node* link;
};
int main()
{
    struct node ob;
    return 0;
}
```

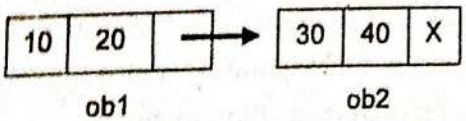
- In the above example 'link' is a pointer to a structure of type 'node'. Hence, the structure 'node' is a self-referential structure with 'link' as the referencing pointer.
- An important point to consider is that the pointer should be initialized properly before accessing, as by default it contains garbage value.

Two types of Self-Referential Structures:

Following are the two types of self referential structure:

(a) Self-Referential Structure with Single Link:

- These structures can have only one self-pointer as their member.
- The following example will show us how to connect the objects of a self-referential structure with a single link and access the corresponding data members.
- The connection formed is shown in the following figure.

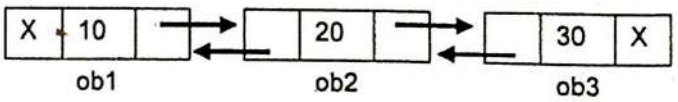



```
#include <stdio.h>
struct node
{
    int data1;
    char data2;
    struct node* link;
};
int main()
{
    struct node ob1;        /*Node1 Initialization*/
    ob1.link = NULL;
    ob1.data1 = 10;
    ob1.data2 = 20;
    struct node ob2;        /*Node2 Initialization */
    ob2.link = NULL;
    ob2.data1 = 30;
    ob2.data2 = 40;
    /* Linking ob1 and ob2*/
    ob1.link = &ob2;
    /*Accessing data members of ob2 using ob1*/
    printf("%d", ob1.link->data1);
    printf("\n%d", ob1.link->data2);
    return 0;
}
```

Output:
30
40

(b) Self-Referential Structure with Multiple Links:

- Self-referential structures with multiple links can have more than one self-pointers.
- Many complicated data structures can be easily constructed using these structures.
- Such structures can easily connect to more than one node at a time. The following example shows one such structure with more than one link.
- The connections made in the above example can be understood using the following figure.



Output:
10 20 30
10 20 30
10 20 30

- In the above example we can see that 'ob1', 'ob2' and 'ob3' are three objects of the self-referential structure 'node'. They are connected using their links in such a way that any of them can easily access each other's data.
- This is the beauty of the self-referential structures. The connections can be manipulated according to the requirements of the programmer.

Q.27. Give the applications of self-referential structures.

Ans. Applications:

Self-referential structures are very useful in creation of other complex data structures like:

(i) Linked Lists
(ii) Stacks
(iii) Queues
(iv) Trees
(v) Graphs etc.

PROGRAMS FOR PRACTICE:

Q.1. Write a program to accept the marks of 3 subjects along with roll no. of 10 students and print the same using array of structures.

Q.2. Write a program to swap two numbers using pointers.

Q.3. Write a program using a structure called student structure that contains information about students (roll_no, name, branch, age). Ask the user for information of 10 different students and display the same information in ascending order of age.

Q.4. Write a C program to accept the name and salaries of 30 employees. Find the highest salary, lowest salary and average salary.

Q.5. Write a function large( ) to find the largest element from an array of sizes using pointer.

POINTS TO REMEMBER:

(1) A structure is usually used when we wish to store dissimilar data together.
(2) Structure elements can be accessed through a structure variable using a dot(.) operator.
(3) Structure elements can be accessed through a pointer to a structure using the arrow(->) operator.
(4) All elements of one structure variable can be assigned to another structure variable using the assignment(=) operator.
(5) It is possible to pass a structure variable to a function either by value or by address.
(6) It is possible to create an array of structures.
(7) The pointers pointing to a structure are known as structure pointers.
*(8) A pointer variable is declared by giving it a type and a name (e.g int *ptr) where the asterisk tells the compiler that the variable named ptr is a pointer variable and the type tells the compiler what type the pointer is to point to.*
(9) Once a variable is declared, we can get its address by preceding its name with the unary operator, as in &k and is called as address of the operator.
(10) Pointers can be used to stimulate 'call by reference' to functions.
(11) Self-referential structures are those structures that have one or more pointers that point to the same type of structure as their member.