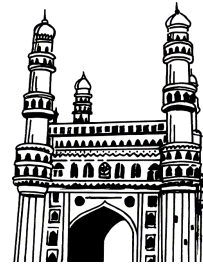


Rahul's ✓
Topper's Voice

**NEW
SYLLABUS**



B.C.A.

I Year I Sem

Latest 2024 Edition

PROGRAMMING IN C

- 👉 Study Manual
- 👉 Important Questions
- 👉 Solved Model Papers
- 👉 Previous Question Papers

- by -

WELL EXPERIENCED LECTURER

Price
199-00



Rahul Publications TM

Hyderabad. Cell : 9391018098, 9505799122

All disputes are subjects to Hyderabad Jurisdiction only

PROGRAMMING IN C

February - 2023

i - ii

July / August - 2023

iii - iv

STUDY MANUAL

Important Questions

V - VIII

Unit - I

1 - 56

Unit - II

57 - 126

Unit - III

127 - 164

Unit - IV

165 - 198

Unit - V

199 - 236

SOLVED MODEL PAPERS

Model Paper - I

237 - 238

Model Paper - II

239 - 240

Model Paper - III

241 - 242

July - 2021

243 - 246

December - 2019

247 - 250

FACULTY OF INFORMATICS
B.C.A I - Semester (CBCS) (Main & Backlog & One Time Chance) Examination
February - 2023
PROGRAMMING IN C

Time : 3 Hours]

[Max. Marks : 70

Note : I. Note : Answer all questions from Part-A and answer any five questions from Part-B. Choosing one question from each unit.

II. Missing data, if any, may be suitably assumed.

PART - A (10 × 2 = 20 Marks)

1. (a) What is Computer System?
(b) What is meant by Identifier?
(c) Describe about Break Statement.
(d) What is a Function?
(e) How to create an Array?
(f) What is sorting?
(g) What is R-value?
(h) Define a String.
(i) What is a Type Definition?
(j) What is a Stream?

PART - B (5 × 10 = 50 Marks)

Unit-I

2. (a) Explain about Computing Environments in detail.
(b) Explain about Decimal Number System with an example.

(OR)

3. (a) Explain about Input and Output Statements with examples.
(b) Discuss about Arithmetic Operators in brief.

Unit-II

4. (a) Explain about Logical Operators with an example.
(b) Discuss about for loop with an example.

(OR)

5. (a) Explain about User-defined Functions in detail.
(b) Discuss about Storage Classes in brief.

Unit-III

- 6. (a) Explain about Array Applications in detail.
- (b) Explain about Multidimensional Arrays with an example.

(OR)

- 7. (a) Discuss about Linear search in detail.
- (b) Explain about Bubble Sort with an example.
- 8. (a) Explain about Pointers for Inter-Function Communication.
- (b) Discuss about Memory Allocation Functions in detail.

(OR)

- 9. (a) Explain about Command-line Arguments.
- (b) Discuss about String Input / Output functions.

Unit-V

- 10. (a) Explain about Initialization of structures with examples.
- (b) Discuss about Self-referential Structures in brief.

(OR)

- 11. (a) Explain about Input and Output Streams in detail.
- (b) Discuss about Standard Library Input / Output functions.

FACULTY OF INFORMATICS
B.C.A. I-Semester (CBCS) Examination
July / August - 2023
PROGRAMMING IN C

Time : 3 Hours]

[Max. Marks : 70

Note : I. Answer all questions from Part-A and answer any five questions from Part-B.

Choosing one question from each unit.

II. Missing data, if any, may be suitably assumed.

PART - A (10 × 2 = 20 Marks)

1. (a) Write about Assembler and Loader.
(b) Define Machine Language.
(c) Write about Decision Making Statements.
(d) Define break, Continue and Goto.
(e) Write about Function.
(f) Write about Accessing the Address of a Variable
(g) Discuss about Searching and Sorting.
(h) Define String handling functions.
(i) What is Nested Structure and Union?
(j) Write about Reading and Writing a File.

PART - B (5 × 10 = 50 Marks)

UNIT - I

2. (a) Explain about Assemblers, Loaders and Linkers.
(b) Write briefly about parts of computers.
(OR)
3. (a) Discuss about Software development and Flowcharts.
(b) Describe briefly about High level Language and Machine Language.

UNIT - II

4. (a) Write about Data types in C.
(b) Write briefly about Decision making and Looping Control Structure with an example.
(OR)
5. (a) Write short notes on While loop and For Loop .
(b) Explain about break, continue and case control Structure with example.

UNIT - III

6. (a) How to access the address of a variables?
(b) Explain about Recursion Function with an example.
(OR)
7. (a) Explain about Types of Functions.
(b) Write about Declaration of Arguments and Return types in detail.

UNIT - IV

8. (a) How to declare an array and Handling an Array?
(b) Write the different types of Arrays with example.
(OR)
9. (a) Explain about Searching and Sorting.
(b) Write about String handling Functions.

UNIT - V

10. (a) Explain the differences between Strings and Unions.
(b) Discuss about Nested Structure.
(OR)
11. (a) Explain about Unions.
(b) Explain about File handling Functions.

Contents

Topic	Page No.
UNIT - I	
1.1 Introduction To Computers	1
1.1.1 Computer Systems	1
1.1.2 Computing Environments	2
1.1.3 Computer Languages	7
1.1.4 Creating And Running Programs	10
1.1.5 Software Development	12
1.1.6 Flow Charts	14
1.2 Number Systems	21
1.2.1 Binary	22
1.2.2 Octal	23
1.2.3 Decimal	27
1.2.4 Hexadecimal	28
1.3 Introduction To C Language	32
1.3.1 Background	32
1.3.2 C Programs	35
1.3.3 Identifiers	36
1.3.4 Data Types	38
1.3.5 Variables	40
1.3.6 Constants	41
1.3.7 Input / Output Statements	43
1.3.8 Arithmetic Operators	44
1.4 Expressions	46
1.4.1 Evaluating Expressions	46
1.4.2 Precedence And Associativity Of Operators	47
1.4.3 Type Conversions	49
➤ Short Question and Answers	51 - 53
➤ Choose the Correct Answer	54 - 54
➤ Fill in the blanks	55 - 55
➤ One Mark Answers	56 - 56

Topic	Page No.
UNIT - II	
2.1 Conditional Control Statements	57
2.1.1 Bitwise operators, Relational and Logical Operators	57
2.2 If, If-Else, Switch Statements and Examples	74
2.3 Looping Statements: While, Do-While and Examples	85
2.4 Continue, Break, Go to Statements	91
2.5 Functions	93
2.5.1 User defines functions	95
2.5.2 Inter function communication	100
2.5.3 Standard Functions	101
2.5.4 Parameters passing in C	102
2.5.5 Recursion–Recursive functions	104
2.6 Storage classes - Auto, Register, Static, Extern, scope rules	106
➤ Short Question and Answers	111 - 122
➤ Choose the Correct Answer	123 - 124
➤ Fill in the blanks	125 - 125
➤ One Mark Answers	126 - 126
UNIT - III	
3.1 Preprocessors – Preprocessor Commands	127
3.2 Arrays – Concepts using arrays in C	132
3.2.1 Inter function communication	135
3.2.2 Array Applications	137
3.4 Linear Search and Binary Search	145
3.5 Selection Sort and Binary Sort	153
➤ Short Question and Answers	157 - 161
➤ Choose the Correct Answer	162 - 162
➤ Fill in the blanks	163 - 163
➤ One Mark Answers	164 - 164

Topic	Page No.
UNIT - IV	
4.1 Pointers	165
4.1.1 Introduction	165
4.1.2 Pointers for Inter-Function Communication	167
4.1.3 Pointers to pointers	169
4.1.4 Compatibility	170
4.1.5 L-Value and R-Value	171
4.1.6 Arrays and Pointers	171
4.1.7 Pointer Arithmetic and arrays	172
4.1.8 Passing an Array to a Function	176
4.1.9 Memory Allocation Functions	176
4.1.10 Array of Pointers	178
4.1.11 Programming Application	178
4.1.12 Pointers to void	179
4.1.13 Pointers to Functions	181
4.1.14 Command line arguments	182
4.2 Strings	183
4.2.1 Concepts, C Strings	183
4.2.3 Arrays of Strings	185
4.2.4 String Manipulation Functions	187
➤ Short Question and Answers	190 - 194
➤ Choose the Correct Answer	195 - 196
➤ Fill in the blanks	197 - 197
➤ One Mark Answers	198 - 198
UNIT - V	
5.1 Structures - Definition, Initialization of structure, Accessing Structure	199
5.1.1 Nested structures	203
5.1.2 Array of Structures	207

Topic	Page No.
5.1.3 Structures and Functions	208
5.1.4 Pointers to Structures	211
5.1.5 Self Referential Structures	212
5.1.6 Unions.....	214
5.1.7 Type Definition (typedef)	217
5.1.8 Enumerated types	218
5.2 Input and Output.....	220
5.2.1 Introduction to files	220
5.2.2 Modes of Files	220
5.2.3 Streams, Standard Library Input/Output Functions, Character Input/ Output Functions	223
➤ Short Question and Answers	227 - 233
➤ Choose the Correct Answer	234 - 234
➤ Fill in the blanks	235 - 235
➤ One Mark Answers	236 - 236

Important Questions

UNIT - I

1. **How many types of computing Environments are there? Explain them with their advantages and disadvantages.**

Ans :

Refer Unit-I, Q.No. 2.

2. **Briefly describe about various Computer Languages with examples.**

Ans :

Refer Unit-I, Q.No. 3.

3. **What is SDLC?**

Ans :

Refer Unit-I, Q.No. 6.

4. **Explain the various stages of SDLC life cycle**

Ans :

Refer Unit-I, Q.No. 7.

5. **Design a flowchart for finding the largest among three numbers entered by the user.**

Ans :

Refer Unit-I, Q.No. 10.

6. **Draw a flowchart for calculating the area of a rectangle.**

Ans :

Refer Unit-I, Q.No. 14.

7. **How to declare and initialize a variable?**

Ans :

Refer Unit-I, Q.No. 38.

8. **Explain standard I/O functions in C**

Ans :

Refer Unit-I, Q.No. 40.

9. **Explain the types of type conversions in C with example?**

Ans :

Refer Unit-I, Q.No. 48.

UNIT - II

1. What is Operator? List out various types of Operators supported by C?

Ans :

Refer Unit-II, Q.No. 1.

2. Explain in detail Logical operators in C.

Ans :

Refer Unit-II, Q.No. 4.

3. Explain if-else-if ladder statement in C with an example.

Ans :

Refer Unit-II, Q.No. 10.

4. Explain in detail switch statement in C.

Ans :

Refer Unit-II, Q.No. 12.

5. What are the differences between If-else and switch statements.

Ans :

Refer Unit-II, Q.No. 13.

6. How can we use while loop in c programming?

Ans :

Refer Unit-II, Q.No. 16.

7. What is mean by nested loops?

Ans :

Refer Unit-II, Q.No. 19.

8. What is Function ? What are the advantages of functions ?

Ans :

Refer Unit-II, Q.No. 24.

9. List out various C Library functions.

Ans :

Refer Unit-II, Q.No. 28.

10. List out various standard function of C.

Ans :

Refer Unit-II, Q.No. 32.

UNIT - III

1. List out various preprocessor commands supported by C.

Ans :

Refer Unit-III, Q.No. 1.

2. What is array ? Explain its advantages.

Ans :

Refer Unit-III, Q.No. 3.

3. Explain the process of declaring and initializing an array ?

Ans :

Refer Unit-III, Q.No. 6.

4. Discuss in detail about Multi dimensional arrays in C.

Ans :

Refer Unit-III, Q.No. 11.

5. Write a C Programs to add two matrices using arrays.

Ans :

Refer Unit-III, Q.No. 12.

6. Write a C Program for multiplication of two matrices.

Ans :

Refer Unit-III, Q.No. 14.

UNIT - IV

1. Define pointer? Explain the process of declaring a pointer in c program.

Ans:

Refer Unit-IV, Q.No. 1.

2. How can we use arrays using pointers?

Ans:

Refer Unit-IV, Q.No. 9.

3. What is Dynamic Memory allocation? List and explain standard defined functions used for dynamic memory allocation.

Ans:

Refer Unit-IV, Q.No. 12.

4. **What is the use of Array of Pointers in C?**

Ans:

Refer Unit-IV, Q.No. 13.

5. **Explain the concept of Pointers to Functions.**

Ans:

Refer Unit-IV, Q.No. 16.

6. **What are string Input Output functions?**

Ans:

Refer Unit-IV, Q.No. 19.

UNIT - V

1. **How can we declare structures?**

Ans:

Refer Unit-V, Q.No. 2.

2. **Write a C Program to pass the entire Structure to Functions.**

Ans:

Refer Unit-V, Q.No. 8.

3. **What are self-referential structures? Write the syntax of self-referential structures.**

Ans:

Refer Unit-V, Q.No. 11.

4. **Define Union.**

Ans:

Refer Unit-V, Q.No. 13.

5. **Discuss the differences between Structures and Unions.**

Ans:

Refer Unit-V, Q.No. 14.

6. **What is the use of typedef in C programming?**

Ans:

Refer Unit-V, Q.No. 16.

7. **What is Enumerated types ? How can we declare enumerated types in C?**

Ans:

Refer Unit-V, Q.No. 18.

UNIT I

Introduction to Computers: Computer Systems, Computing Environments, Computer Languages, Creating and Running Programs, Software Development, Flow charts.

Number Systems: Binary, Octal, Decimal, Hexadecimal

Introduction to C Language - Background, C Programs, Identifiers, Data Types, Variables, Constants, Input / Output Statements

Arithmetic Operators and Expressions: Evaluating Expressions, Precedence and Associativity of Operators, Type Conversions.

1.1 INTRODUCTION TO COMPUTERS

1.1.1 Computer Systems

Q1. Define computer? Explain about the components of a computer system.

Ans:

A computer is an electronic device that can be programmed to accept data (input), process it and generate result (output). A computer along with additional hardware and software together is called a computer system. A computer system primarily comprises of a central processing unit, memory, input/output devices, and storage devices. All these components function together as a single unit to deliver the desired output. A computer system comes in various forms and sizes. It can vary from a high-end server to a personal desktop, laptop, tablet computer, or smartphone.

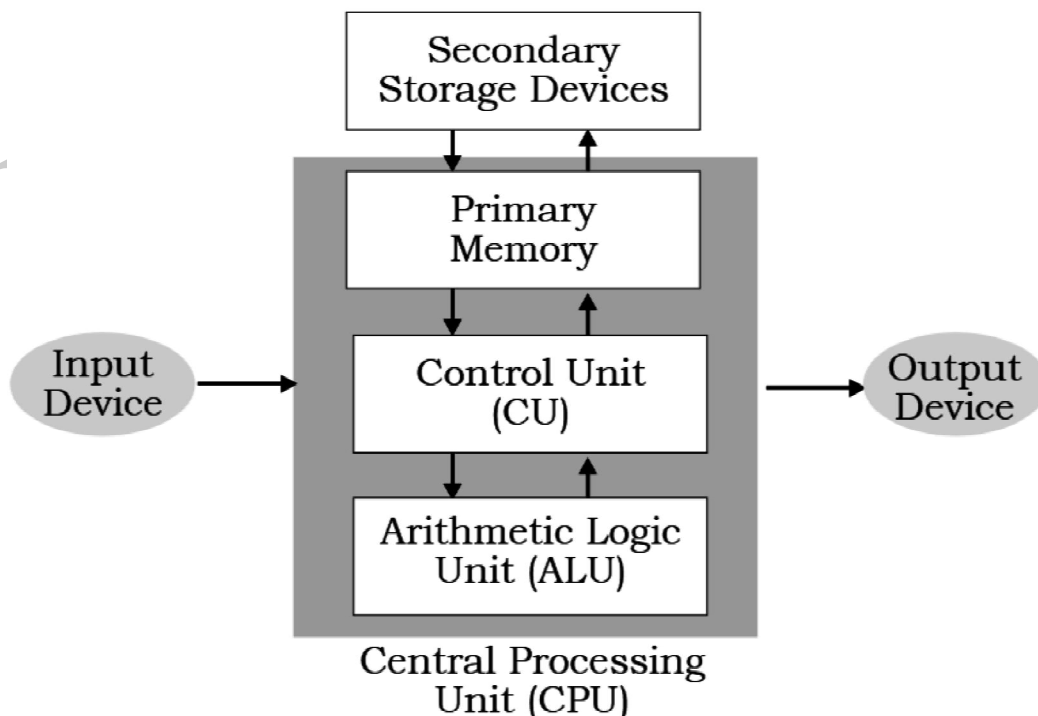


Fig.: Components of a Computer System

Central Processing Unit (CPU) It is the electronic circuitry of a computer that carries out the actual processing and is usually referred to as the brain of the computer. The CPU is given instructions and data through programs. The CPU then fetches the program and data from the memory and performs arithmetic and logical operations as per the given instructions and stores the result back to memory.

CPU has two main components - Arithmetic Logic Unit (ALU) and Control Unit (CU).

1. Arithmetic Logic UnitA

LU performs all the arithmetic and logic operations that need to be done as per the instruction in a program.

2. Control Unit

CU controls sequential instruction execution, interprets instructions and guides data flow through the computer's memory, ALU and input or output devices. CPU is also popularly known as microprocessor.

3. Input Devices

The devices through which control signals are sent to a computer are termed as input devices. These devices convert the input data into a digital form that is acceptable by the computer system. Some examples of input devices include keyboard, mouse, scanner, touch screen, etc.

4. Output Devices

The device that receives data from a computer system for display, physical production, etc., is called output device. It converts digital information into human understandable form. For example, monitor, projector, headphone, speaker, printer, etc.

1.1.2 Computing Environments

Q2. How many types of computing Environments are there? Explain them with their advantages and disadvantages.

Ans:

(Imp.)

Computing Environment

Computing Environment is a collection of computers, software, hardware, and networks that support the processing and exchange of electronic information used to support various types of computing solutions.

There are so many different types of computing environments today. They are

1. PersonalComputingEnvironment
2. Time-Sharing Environment
3. Client/Server Environment
4. Distributed Computing

1. Personal Computing environment

The personal computer is a small computer with a microprocessor, designed for use by an individual, used in the home, small businesses, etc. It is a complete system in itself and its convenient size, price and simple functions make it easy for the end-user to work on it without any intervention from computer operators.

Applications of Personal Computing

Today personal computing environment is used every where from home computer to Scientific simulation Applications

There are 4 types of Personal Computers available today:

- Desktop
- Laptop
- Tablet
- Smart Phone

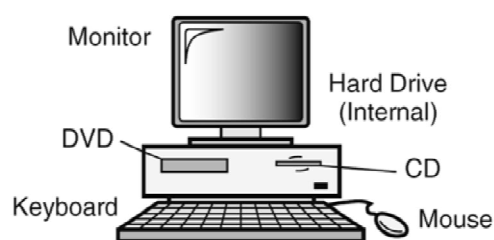


Fig.: Desktop computer

Advantages

- Lots of memory space
- easy to upgrade
- fast processors
- no battery
- large screen (depending on the monitor)
- cheaper than laptops and tablets
- the monitor, keyboard, mouse, etc

Disadvantages

- large footprint
- need keyboard, mouse and monitor
- big and heavy
- requires a separate monitor
- probably without wireless connection

2. Time Sharing Environment:

The concept of time sharing computing is to share the processing of the computer by allocating time slots to the users. In the time-sharing environment, all computing must be done by the central computer.

In this environment many users are connected to one or more computers, which are known as workstations. Each user is given a time slice of CPU time. Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. So that each user seems to be the sole user of the computer.

In this environment the output devices, auxiliary storage devices are shared by all the users. It needs a special operating system which is known as time sharing Operating System.

Examples of time sharing OS

- UNIX
- Windows NT etc

Applications of Time Sharing Environment

The bank's bankcard system, which allows hundreds of people to access the same program on the mainframe at the same time.

Advantages

- In time sharing systems many applications can run at the same time, which increases performance.
- All the software is stored at Central Computer and shared by all user, which can avoid duplication of software.
- Many users are connected with the single CPU at the same time, which makes the CPU busy always.

Disadvantages:

- The big disadvantages of time sharing systems is that it consumes much resources so it need special operating systems.
- Switching between tasks becomes sometimes sophisticated as there are lot of users and applications running which may hang up the system.
- So the time sharing systems should have high specifications of hardware.

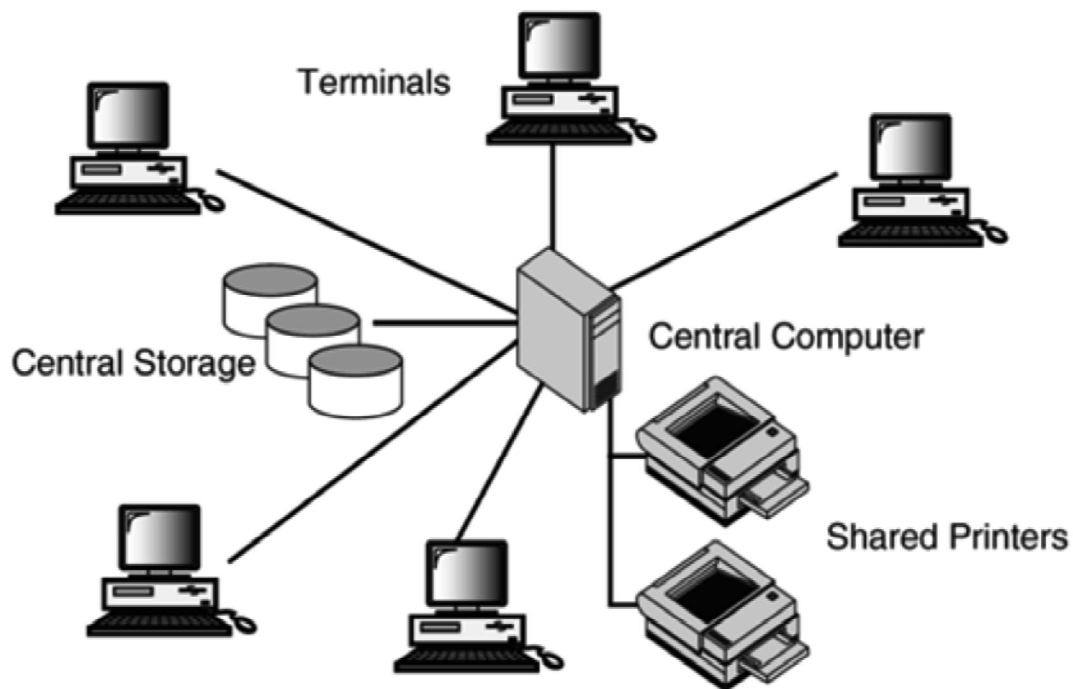


Fig.: Time sharing Environment

3. Client /Server Environment

Client /Server computing environment split the computing function between central computer and user computers. The client/server environment is divided into client and server.

A client is a machine which sends the requests to the server for processing. A Server is a Central machine which process the client's requests.

A client computer provides the interface , so that the user can work comfortably.A server computer provides high volume storage capacity and high processing capacity, so that it can process many user's requests at a time.

Applications

- Internet is the best example for client –server Computing, The web browser is the client, the user sends a request to the server , the web server process the request and send it back to the web browser (client).

Advantages

- A client-server architecture enables the roles and responsibilities of a computing system to be distributed among several independent computers.
- Greater ease of maintenance.
- All the data is stored on the servers, generally have much security controls than most clients. Servers can better control access and resources to guarantee that only those clients with the appropriate permissions may access and change data.
- Updates data are much easier to administrators than what would be possible under a P2P architecture.
- Many advanced clients server technologies are already available which were designed to ensure security, user friendly interfaces and ease of use.

Disadvantages

- Networks traffic blocking is one of the problems related to the client-server model. Number of simultaneous client requests to a given server increases, the server can become overloaded.
- Bandwidth decreases when more nodes are added to the network
- If one server fail, clients requests cannot be serverd.

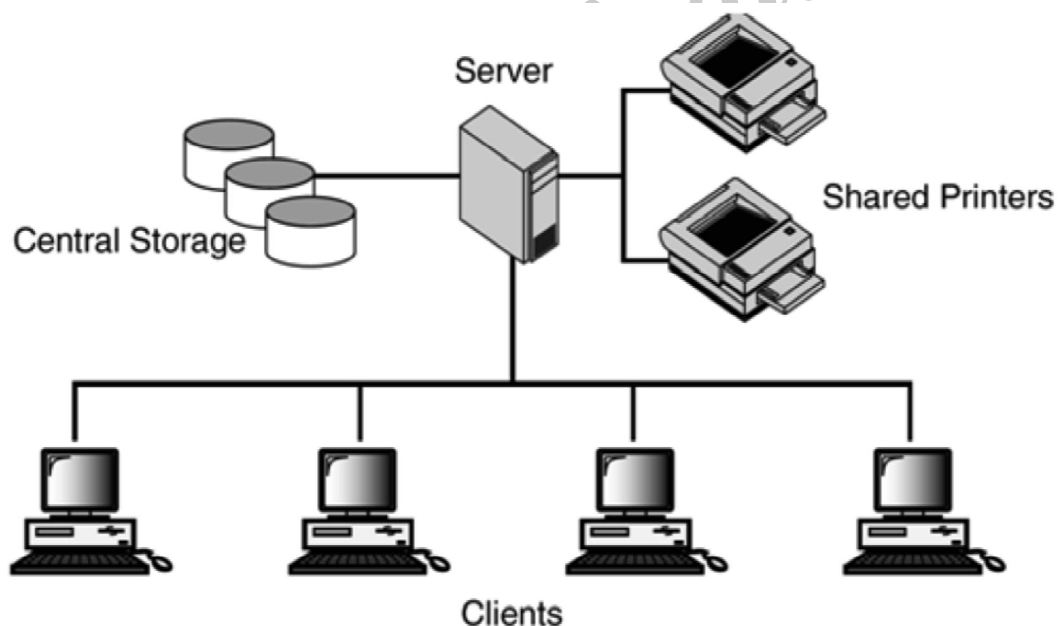


Fig.: Client server Environment

4. Distributed Computing

A distributed system is a group of several computers which do not share a memory and computers communicate with each other by sharing messages over the communication network.

Each computer has its own memory and runs its own operating system. The resources owned and controlled by a computer are said to be local to it. While the resources owned and controlled by other computers are said to be remote.

Applications

- Railway reservation system is the best example of distributed computing. The widely spread computers or devices does the single task of ticket booking.
- Telephone and Cellular networks
- Internet
- Wireless Sensor networks etc

Examples of Distributed operating Systems

- IRIX operating system.
- DYNIX operating system

Advantages

- **Sharing of Data:** The data can be shared among different users at different sites can reduce the redundancy of data.
- **Availability :** If one site fails in a distributed system, the remaining sites may be able to continue operating. Thus a failure of a site doesn't necessarily imply the shutdown of the System.
- **Reliability :** As data may exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible.

Disadvantages

- **Complexity :** Maintain proper coordination among the systems are very complicated.
- **Software Development Cost :** Implementing a distributed software is more costly issue
- **Security :** Controlling to access the data base is difficult, which arises the security issues.

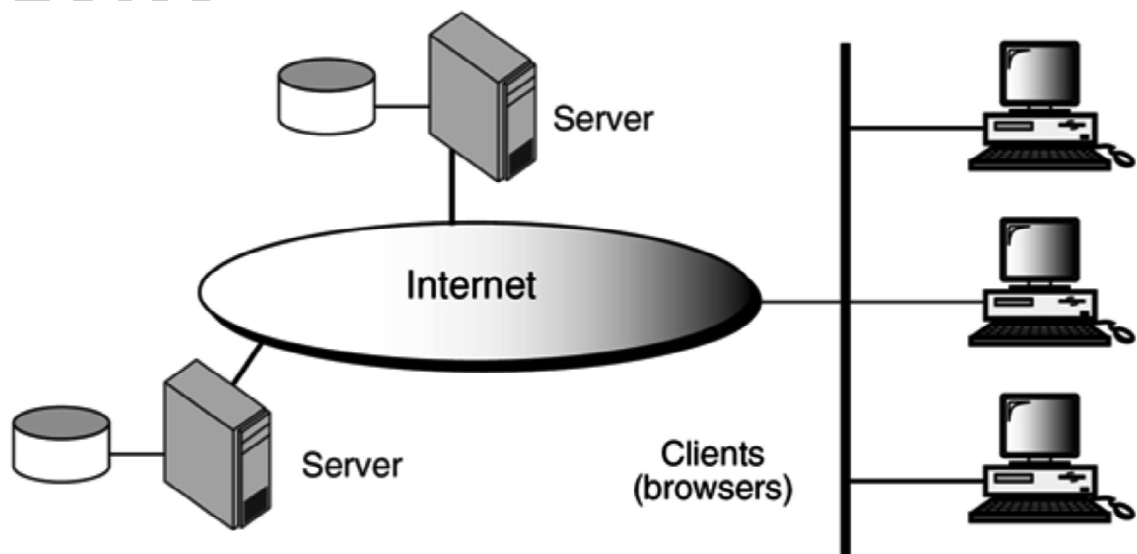


Fig.: Distributed Environment

1.1.3 Computer Languages

Q3. Briefly describe about various Computer Languages with examples.

Ans:

(Imp.)

To write a program for a computer, we must use a computer language. Over the years computer languages have evolved from machine language to natural languages.

The Computer languages are divided into 3 types

1. Machine Languages
2. Symbolic Languages
3. High Level Languages

1. Machine Languages

In the earliest days of computers, the only programming languages available were machine languages. It is a first generation programming language. It is the language based on binary digits (0's and 1's). The operations written in machine language can directly execute by a particular computer. A computer cannot understand instructions given to it in high-level languages or in English. It can only understand and execute instructions given in the form of machine language i.e. binary. This is the only language understood by the computer.

Advantages

- Machine language makes fast and efficient use of the computer
- The computer can understand instructions immediately
- It requires no translator to translate the code. It is directly understood by the computer

Disadvantages

- Machine dependent
- Programming is very difficult
- Difficult to understand
- Difficult to write bug free programs
- Difficult to isolate an error

Example : Addition of two numbers

```

2       → 0010
+ 3     → 0011
-----
5       ← 0101
-----
```

Example of Machine Language program for addition of 2 numbers

Location	Instruction Code
Hex	Binary
100	0010 0001 0000 0100
101	0001 0001 0000 0101
102	0011 0001 0000 0110
103	0111 0000 0000 0001
104	0000 0000 0101 0011
105	1111 1111 1111 1110
106	0000 0000 0000 0000

Instruction Code Hex	Instruction	Comments
2104	LDA 104	Load first operand into AC
1105	ADD 105	Add second operand to AC
3106	STA 106	Store sum in location 106
7001	HLT	Halt computer
0053	operand	83 decimal
FFFE	operand	-2 decimal
0000	operand	Store sum her

2. Symbolic / Assembly Languages

It is a second generation programming language. Assembly language was developed to overcome some of the many inconveniences of machine language. This is another type of low-level language in which operation codes and operands are given in the form of alphanumeric symbols instead of 0's and 1's. These alphanumeric symbols are known as mnemonic codes and can combine in a maximum of five-letter combinations e.g. ADD for addition, SUB for subtraction, START, LABEL etc. Because of this feature, assembly language is also known as 'Symbolic Programming Language.'

Assembly language is designed to be easily translated into machine language. Although blocks of data may be referred to by name instead of by their machine addresses. The instructions of the assembly language are converted to machine language by a language translator called assembler and then they are executed by the computer.

Advantages

- Assembly language is easier to understand and use as compared to machine language.
- It is easy to locate and correct errors.
- It is easily modified.

Disadvantages

- Like machine language, it is also machine dependent/specific.
- Since it is machine dependent, the programmer also needs to understand the hardware.

Example Assembly Language instructions

Addition of two numbers.

ORG	100 / Origin of program location 100	Comment
	LDAA	/Load operand from location A
	ADDB	/Add operation form location B
	STAC	/Store sum in location C
	HLT	/Halt computer
A,	DEC 83	/Decimaloperand
B,	DEC -2	/Decimal operand
C,	DEC 0	/Sum stored in location C
	END	

3. High Level Languages

These are known as third generation programming language. High-level languages allow us to write programs using instructions resembling everyday spoken language (for example: print, if, while) which are then translated into machine language to be executed.

Programs written in a high-level language need to be translated into machine language before they can be executed. Some programming languages use a compiler to perform this translation and others use an interpreter.

Advantages

- Easy to follow.
- Easy to understand
- Easy to modify and debug.
- Suitable for complex applications

Disadvantages

- It requires the translator program called Compiler or Interpreter.
- It runs programs slower with compare to low level languages

- Examples of high level languages
- C, C++ , JAVA, PASCAL ,COBOL , Etc.

Example C Program for addition of 2 numbers

```
#include <stdio.h>
int main()
{
    int a,b sum;
    printf("Enter two integers: ");
    scanf("%d %d",&a, &b );
    sum = a + b;
    printf("%d + %d = %d", a,b,sum);
    return 0;
}
```

Q4. Write the Comparisons between Machine level, Assembly, High level Languages.

Ans:

Feature	Machine	Assembly	High Level
Coding Form	0's and 1's	Mnemonic codes	Normal English
Dependency	Dependent	Dependent	Independent
Translator	Not Needed	Needed(Assembler)	Needed(Compiler)
Time of Execution	Less	Less	High
Languages	Only one	Intel 8085, X86 series...	C,C++,Java etc
Complexity	Difficult	Moderate	Easy
Memory Space	Less	Less	More

1.1.4 Creating And Running Programs

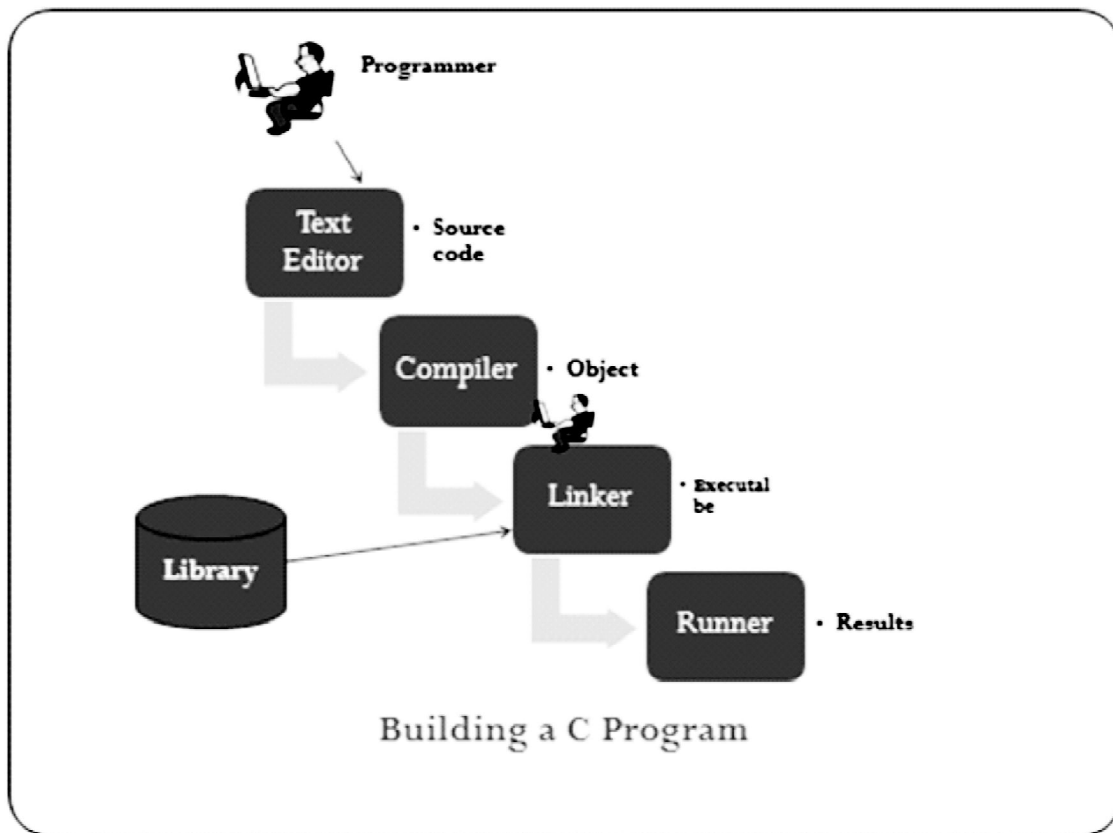
Q5. What are the steps involved in Creation and Running of a C Program? explain them.

Ans:

It is the job of programmer to write and test the program. This process is done in four steps. The following are the steps for creating and running programs:

- A. Writing and Editing the Program.
- B. Compiling the Program.
- C. Linking the Program with the required library modules.
- D. Executing the Program

The following picture shows the four step process of creating and running the C program



(A) Writing and Editing the Program

To write and Edit the programs we need to use a particular software. The software used to write programs is known as a text editor. A text editor helps us to enter, change and store character data. After the program is completed the program is saved in a file to disk with an extension of .C .

This file will be input to the compiler, it is known as source file. Every compiler comes with associated text editor.

(B) Compiling Programs

The code in a source file on the disk must be translated in to machine language. This is the job of compiler which translates code in source file stored on disk in to machine language. This translation process is known as compilation.

The entire high level program is converted into the executable machine code file. The Compiler which executes C programs is called as C Compiler.

Example Turbo C, Borland C, GC etc.,

The C compiler is actually divided into two separate programs:

- the preprocessor and
- the translator.

The preprocessor reads the source code and prepares it for the compiler. It scans the special instructions known as pre-processor commands. These commands tell the pre-processor to take for special code from libraries, make substitutions in the code. The result of pre-processing is called a translation unit.

After the preprocessor has prepared the code for compilation, the translator does the conversion of the program into machine language. The translator reads the translation unit and writes resulting object module to a file that can be combined with other precompiled units to form the final program. An object module is the code in machine language.

This module is not ready for execution because it does not have the required C and other functions included.

(C) Linking Programs

The Linker assembles all functions, the program's functions and system's functions into one executable program.

After the compiler has created all the object files, another program is called to bundle them into an executable program file. That program is called a linker and the process of bundling them into the executable is called linking.

It links all the object files by replacing the references to undefined symbols with the correct addresses. Each of these symbols can be defined in other object files or in libraries. If they are defined in libraries other than the standard library, you need to tell the linker about them.

The linker looks at all the object files you have told it to use.

C programs are made up of many pre-defined functions.

Example: printf () ,scanf(), main()... etc

Their code exists in the library. They must be attached to our program.

(D) Executing Programs

Once our program has been linked, it is ready for execution. To execute a program, we use operating system command, such as run to load the program in to main memory and execute it.

Getting program in to memory is the function of an Operating System programs called loader. Loader locates the executable program and reads it in to memory.

In a typical program execution, the program reads data for processing, either from user or from file. After the program processes the data, it prepares output. Data output can be to user's monitor or to a file. When program has executed, Operating System removes the program from memory.

1.1.5 Software Development

Q6. What is SDLC?

Ans:

(Imp.)

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

Q7. Explain the various stages of SDLC life cycle*Ans:***(Imp.)**

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.

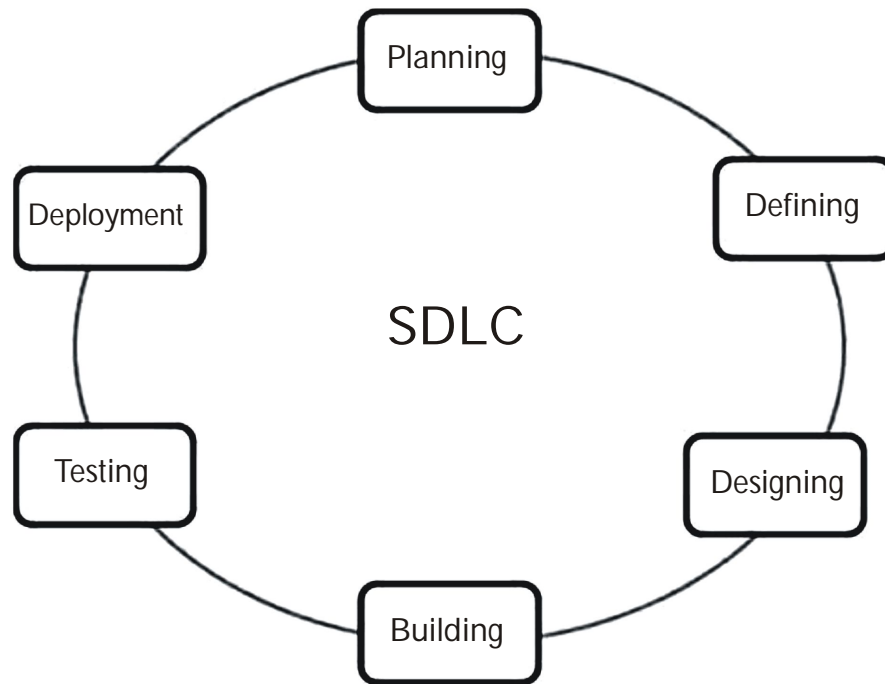


Fig.: System development Life Cycle

A typical Software Development Life Cycle consists of the following stages -

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry "

- Waterfall Model
- Iterative Model
- Spiral Model

1.1.6 Flow Charts**Q8. What is Flowchart ?**

Ans:

(Imp.)

It is a symbolic diagram of operations sequence, dataflow, control flow and processing logic in information processing.

The symbols used are simple and easy to learn.

It is a very helpful tool for programmers and beginners.

Purpose of a Flowchart

- Provides Communication.
- Provides an Overview.

- Shows all elements and their relation ships.
- Quick method of showing Program flow.
- Checks Program logic.
- Facilitates Coding.
- Provides Program revision.
- Provides Program documentation.

Q9. Explain the Various Symbols in a Flowchart ?

Ans :

The different flowchart symbols have different conventional meanings.

The various symbols used in Flowchart Designs are given below.

➤ **Terminal Symbol**

In the flowchart, it is represented with the help of a circle for denoting the start and stop symbol. The symbol given below is used to represent the terminal symbol.



➤ **Input/output Symbol**

The input symbol is used to represent the input data, and the output symbol is used to display the output operation. The symbol given below is used for representing the Input/output symbol.



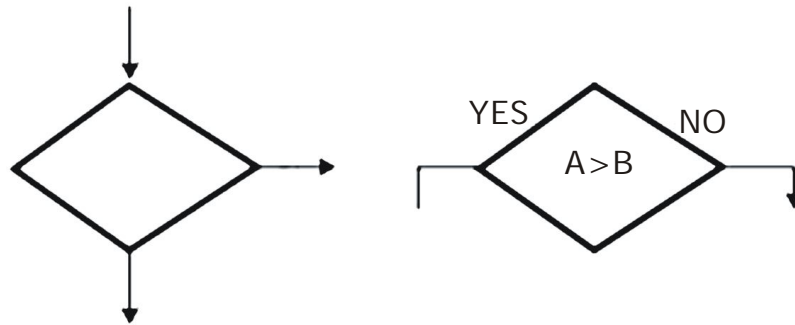
➤ **Processing Symbol**

It is represented in a flowchart with the help of a rectangle box used to represent the arithmetic and data movement instructions. The symbol given below is used to represent the processing symbol.



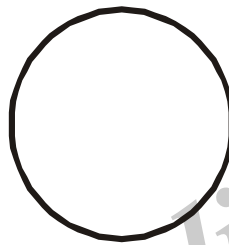
➤ **Decision Symbol**

Diamond symbol is used for represents decision-making statements. The symbol given below is used to represent the decision symbol.



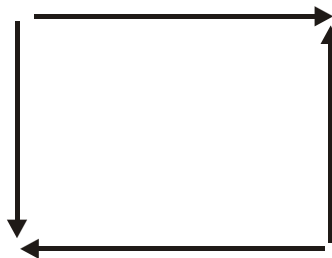
➤ **Connector Symbol**

The connector symbol is used if flows discontinued at some point and continued again at another place. The following symbol is the representation of the connector symbol.



➤ **Flow lines**

It represents the exact sequence in which instructions are executed. Arrows are used to represent the flow lines in a flowchart. The symbol given below is used for representing the flow lines:



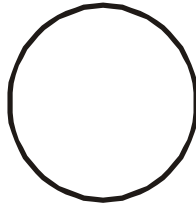
➤ **Hexagon symbol (Flat)**

It is used to create a preparation box containing the loop setting statement. The symbol given below is used for representing the Hexagon symbol.



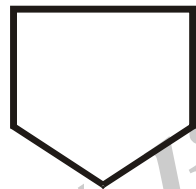
➤ **On-Page Reference Symbol**

This symbol contains a letter inside that indicates the flow continues on a matching symbol containing the same letters somewhere else on the same page. The symbol given below is used for representing the on-page reference symbol.



➤ **Off-Page Reference**

This symbol contains a letter inside indicating that the flow continues on a matching symbol containing the same letter somewhere else on a different page. The symbol given below is used to represent the off-page reference symbol.



➤ **Delay or Bottleneck**

This symbol is used for identifying a delay in a flowchart. The alternative name used for the delay is the bottleneck. The symbol given below is used to represent the delay or bottleneck symbol.



➤ **Document Symbol**

This symbol is used in a flowchart to indicate a document or report. The symbol given below is used to represent the document symbol.



➤ **Internal storage symbol**

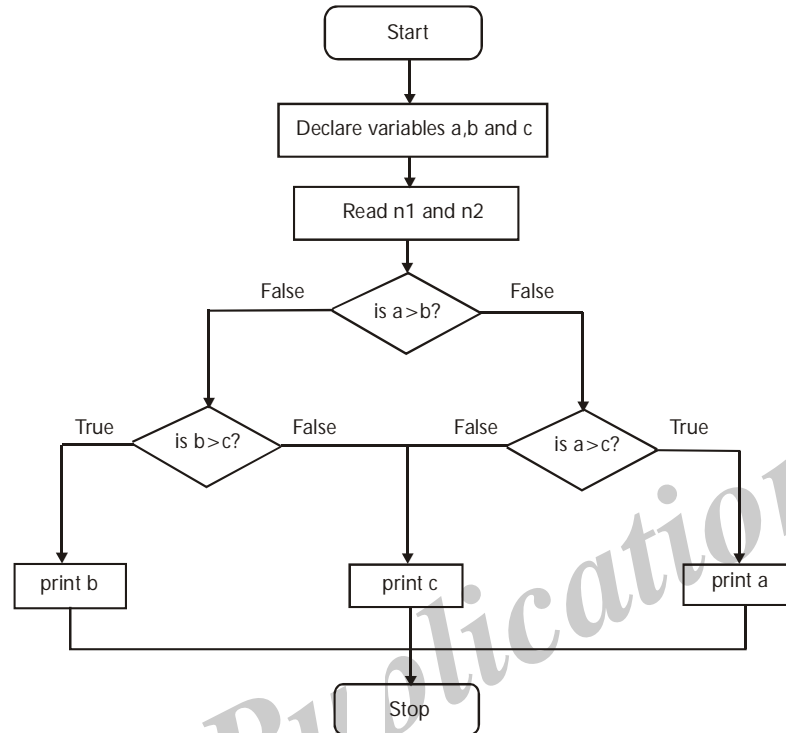
The symbol given below is used to represent the internal storage symbol.



Q10. Design a flowchart for finding the largest among three numbers entered by the user.

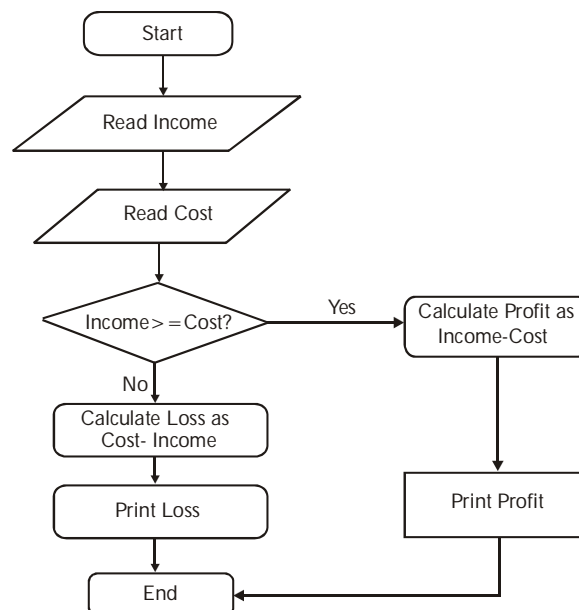
Ans.:

(Imp.)



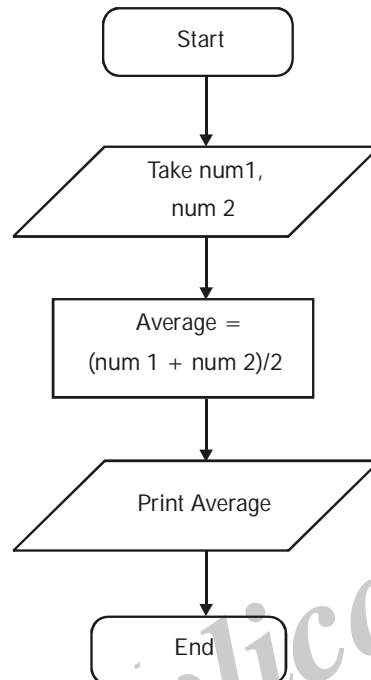
Q11. Draw a flowchart for calculating the profit and loss according to the value entered by the user.

Ans.:



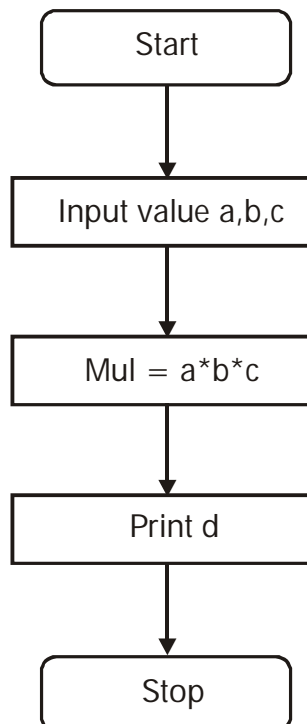
Q12. Draw a flowchart to calculate the average of two numbers.

Ans:



Q13. Draw a flowchart for the multiplication of three numbers entered by the user.

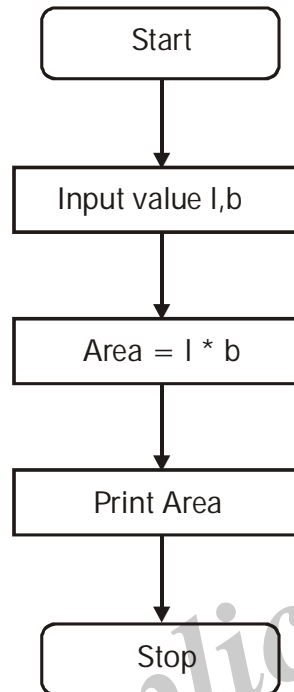
Ans :



Q14. Draw a flowchart for calculating the area of a rectangle.

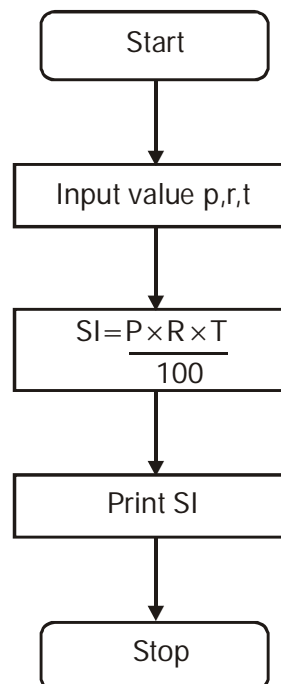
Ans:

(Imp.)



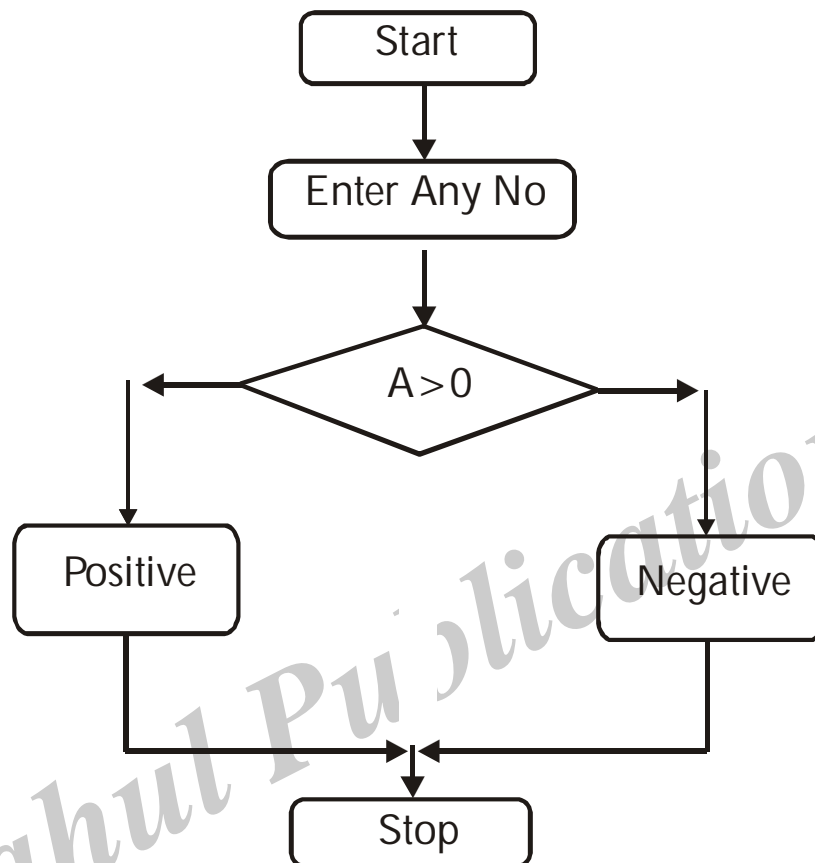
Q15. Draw a flowchart for calculating the Simple Interest according to the value entered by the user.

Ans:



Q16. Draw a flowchart for checking whether the number is positive or negative according to the number entered by the user.

Ans :



1.2 NUMBER SYSTEMS

Q17. What is number system?

Ans :

In a digital system, the system can understand only the optional number system. In these systems, digits symbols are used to represent different values, depending on the index from which it settled in the number system.

In simple terms, for representing the information, we use the number system in the digital system.

The digit value in the number system is calculated using:

1. The digit
2. The index, where the digit is present in the number.
3. Finally, the base numbers, the total number of digits available in the number system.

Q18. Explain about different types of number systems.

Ans:

Types of Number System

In the digital computer, there are various types of number systems used for representing information.

1. Binary Number System
2. Decimal Number System
3. Hexadecimal Number System
4. Octal Number System

1.2.1 Binary

Q19. What is Binary Number System?

Ans:

Generally, a binary number system is used in the digital computers. In this number system, it carries only two digits, either 0 or 1. There are two types of electronic pulses present in a binary number system. The first one is the absence of an electronic pulse representing '0' and second one is the presence of electronic pulse representing '1'. Each digit is known as a bit. A four-bit collection (1101) is known as a nibble, and a collection of eight bits (11001010) is known as a byte. The location of a digit in a binary number represents a specific power of the base (2) of the number system.

Characteristics:

1. It holds only two values, i.e., either 0 or 1.
2. It is also known as the base 2 number system.
3. The position of a digit represents the 0 power of the base(2). Example: 2^0
4. The position of the last digit represents the x power of the base(2). Example: 2^x , where x represents the last position, i.e., 1

Examples

$(10100)_2, (11011)_2, (11001)_2, (000101)_2, (011010)_2$.

Q20. Explain how to convert from binary to decimal with examples.

Ans:

Binary to other Number Systems

There are three conversions possible for binary number, i.e., binary to decimal, binary to octal, and binary to hexadecimal. The conversion process of a binary number to decimal differs from the remaining others. Let's take a detailed discussion on Binary Number System conversion.

Binary to Decimal Conversion

The process of converting binary to decimal is quite simple. The process starts from multiplying the bits of binary number with its corresponding positional weights. And lastly, we add all those products.

Let's take an example to understand how the conversion is done from binary to decimal.

Example 1: $(10110.001)_2$

We multiplied each bit of $(10110.001)_2$ with its respective positional weight, and last we add the products of all the bits with its weight.

$$(10110.001)_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$$

$$(10110.001)_2 = (1 \times 16) + (0 \times 8) + (1 \times 4) + (1 \times 2) + (0 \times 1) + (0 \times 1D\ 2) + (0 \times 1D\ 4) + (1 \times 1D\ 8)$$

$$(10110.001)_2 = 16 + 0 + 4 + 2 + 0 + 0 + 0 + 0.125$$

$$(10110.001)_2 = (22.125)_{10}$$

1.2.2 Octal

Q21. What is octal number system?

Ans :

The octal number system has base 8 (means it has only eight digits from 0 to 7). There are only eight possible digit values to represent a number. With the help of only three bits, an octal number is represented. Each set of bits has a distinct value between 0 and 7.

Below, we have described certain characteristics of the octal number system:

Characteristics

1. An octal number system carries eight digits starting from 0, 1, 2, 3, 4, 5, 6, and 7.
2. It is also known as the base 8 number system.
3. The position of a digit represents the 0 power of the base(8). Example: 8^0
4. The position of the last digit represents the x power of the base(8). Example: 8^x , where x represents the last position, i.e., 1

Number	Octal Number
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Examples

$$(273)_8, (5644)_8, (0.5365)_8, (1123)_8, (1223)_8.$$

Q22. How can we convert a binary number to octal and hexa decimal? Explain*Ans:***Binary to Octal Conversion**

The base numbers of binary and octal are 2 and 8, respectively. In a binary number, the pair of three bits is equal to one octal digit. There are only two steps to convert a binary number into an octal number which are as follows:

1. In the first step, we have to make the pairs of three bits on both sides of the binary point. If there will be one or two bits left in a pair of three bits pair, we add the required number of zeros on extreme sides.
2. In the second step, we write the octal digits corresponding to each pair.

Example 1: $(111110101011.0011)_2$

1. Firstly, we make pairs of three bits on both sides of the binary point.

111 110 101 011.001 1

On the right side of the binary point, the last pair has only one bit. To make it a complete pair of three bits, we added two zeros on the extreme side.

111 110 101 011.001 100

2. Then, we wrote the octal digits, which correspond to each pair.

$(111110101011.0011)_2 = (7653.14)_8$

Q23. Explain, how to convert a decimal number into octal and hexa decimal?*Ans:***Decimal to Octal Conversion**

For converting decimal to octal, there are two steps required to perform, which are as follows:

1. In the first step, we perform the division operation on the integer and the successive quotient with the base of octal(8).
2. Next, we perform the multiplication on the integer and the successive quotient with the base of octal(8).

Example 1: $(152.25)_{10}$ **Step 1:** Divide the number 152 and its successive quotients with base 8.

Operation	Quotient	Remainder
152 / 8	19	0
19 / 8	2	3
2 / 8	0	2

$(152)_{10} = (230)_8$

Step 2: Now perform the multiplication of 0.25 and successive fraction with base 8.

Operation	Result	Carry
0.25×8	0	2

$$(0.25)_{10} = (2)_8$$

So, the octal number of the decimal number 152.25 is 230.2

Decimal to hexadecimal conversion

For converting decimal to hexadecimal, there are two steps required to perform, which are as follows:

1. In the first step, we perform the division operation on the integer and the successive quotient with the base of hexadecimal (16).
2. Next, we perform the multiplication on the integer and the successive quotient with the base of hexadecimal (16).

Example 1: $(152.25)_{10}$

Step 1: Divide the number 152 and its successive quotients with base 8.

Operation	Quotient	Remainder
$152 / 16$	9	8
$9 / 16$	0	9

$$(152)_{10} = (98)_{16}$$

Step 2: Now perform the multiplication of 0.25 and successive fraction with base 16.

Operation	Result	Carry
0.25×16	0	4

$$(0.25)_{10} = (4)_{16}$$

So, the hexadecimal number of the decimal number 152.25 is 230.4.

Q24. Explain, how can we convert an octal number system to other number systems

Ans :

Octal to other Number System

Like binary and decimal, the octal number can also be converted into other number systems.

The process of converting octal to decimal differs from the remaining one. Let's start understanding how conversion is done.

Octal to Decimal Conversion

The process of converting octal to decimal is the same as binary to decimal. The process starts from multiplying the digits of octal numbers with its corresponding positional weights. And lastly, we add all those products.

Let's take an example to understand how the conversion is done from octal to decimal.

Example 1: $(152.25)_8$

Step 1: We multiply each digit of 152.25 with its respective positional weight, and last we add the products of all the bits with its weight.

$$(152.25)_8 = (1 \times 8^2) + (5 \times 8^1) + (2 \times 8^0) + (2 \times 8^{-1}) + (5 \times 8^{-2})$$

$$(152.25)_8 = 64 + 40 + 2 + (2 \times 1D\ 8) + (5 \times 1D\ 64)$$

$$(152.25)_8 = 64 + 40 + 2 + 0.25 + 0.078125$$

$$(152.25)_8 = 106.328125$$

So, the decimal number of the octal number 152.25 is 106.328125

Octal to Binary Conversion

The process of converting octal to binary is the reverse process of binary to octal. We write the three bits binary code of each octal number digit.

Example 1: $(152.25)_8$

We write the three-bit binary digit for 1, 5, 2, and 5.

$$(152.25)_8 = (001101010.010101)_2$$

So, the binary number of the octal number 152.25 is $(001101010.010101)_2$

Octal to hexadecimal conversion

For converting octal to hexadecimal, there are two steps required to perform, which are as follows:

1. In the first step, we will find the binary equivalent of number 25.
2. Next, we have to make the pairs of four bits on both sides of the binary point. If there will be one, two, or three bits left in a pair of four bits pair, we add the required number of zeros on extreme sides and write the hexadecimal digits corresponding to each pair.

Example 1: $(152.25)_8$

Step 1: We write the three-bit binary digit for 1, 5, 2, and 5.

$$(152.25)_8 = (001101010.010101)_2$$

So, the binary number of the octal number 152.25 is $(001101010.010101)_2$

Step 2:

1. Now, we make pairs of four bits on both sides of the binary point.

0 0110 1010.0101 01

On the left side of the binary point, the first pair has only one digit, and on the right side, the last pair has only two-digit. To make them complete pairs of four bits, add zeros on extreme sides.

0000 0110 1010.0101 0100

2. Now, we write the hexadecimal digits, which correspond to each pair.

$$(0000 \quad 0110 \quad 1010.0101 \quad 0100)_2 = (6A.54)_{16}$$

1.2.3 Decimal

Q25. What is decimal number system?

Ans:

Decimal Number System

The decimal numbers are used in our day to day life. The decimal number system contains ten digits from 0 to 9(base 10). Here, the successive place value or position, left to the decimal point holds units, tens, hundreds, thousands, and so on.

The position in the decimal number system specifies the power of the base (10). The 0 is the minimum value of the digit, and 9 is the maximum value of the digit. For example, the decimal number 2541 consist of the digit 1 in the unit position, 4 in the tens position, 5 in the hundreds position, and 2 in the thousand positions and the value will be written as:

$$(2 \times 1000) + (5 \times 100) + (4 \times 10) + (1 \times 1)$$

$$(2 \times 10^3) + (5 \times 10^2) + (4 \times 10^1) + (1 \times 10^0)$$

$$2000 + 500 + 40 + 1$$

$$2541$$

Q26. Explain how to convert decimal to other number system

Ans:

Decimal to other Number System

The decimal number can be an integer or floating-point integer. When the decimal number is a floating-point integer, then we convert both part (integer and fractional) of the decimal number in the isolated form(individually). There are the following steps that are used to convert the decimal number into a similar number of any base 'r'.

1. In the first step, we perform the division operation on integer and successive part with base 'r'. We will list down all the remainders till the quotient is zero. Then we find out the remainders in reverse order for getting the integer part of the equivalent number of base 'r'. In this, the least and most significant digits are denoted by the first and the last remainders.
2. In the next step, the multiplication operation is done with base 'r' of the fractional and successive fraction. The carries are noted until the result is zero or when the required number of the equivalent digit is obtained. For getting the fractional part of the equivalent number of base 'r', the normal sequence of carrying is considered.

Decimal to Binary Conversion

For converting decimal to binary, there are two steps required to perform, which are as follows:

1. In the first step, we perform the division operation on the integer and the successive quotient with the base of binary(2).
2. Next, we perform the multiplication on the integer and the successive quotient with the base of binary(2).

Example 1: $(152.25)_{10}$

Step 1: Divide the number 152 and its successive quotients with base 2.

Operation	Quotient	Remainder
152 / 2	76	0(LSB)
76 / 2	38	0
38 / 2	19	0
19 / 2	9	1
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1(MSB)

$$(152)_{10} = (10011000)_2$$

Step 2: Now, perform the multiplication of 0.27 and successive fraction with base 2.

Operation	Result	Carry
0.25×2	0.50	0
0.50×2	0	1

$$(0.25)_{10} = (.01)_2$$

1.2.4 Hexadecimal

Q27. What is hexa decimal number system?

Ans:

It is another technique to represent the number in the digital system called the hexadecimal number system. The number system has a base of 16 means there are total 16 symbols(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) used for representing a number. The single-bit representation of decimal values 10, 11, 12, 13, 14, and 15 are represented by A, B, C, D, E, and F. Only 4 bits are required for representing a number in a hexadecimal number. Each set of bits has a distinct value between 0 and

15. There are the following characteristics of the octal number system:

Characteristics

1. It has ten digits from 0 to 9 and 6 letters from A to F.
2. The letters from A to F defines numbers from 10 to 15.
3. It is also known as the base 16 number system.
4. In hexadecimal number, the position of a digit represents the 0 power of the base(16). Example: 16^0
5. In hexadecimal number, the position of the last digit represents the x power of the base(16). Example: 16^x , where x represents the last position, i.e., 1

Binary Number	Hexadecimal Number
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Examples

$(FAC2)_{16}$, $(564)_{16}$, $(0ABD5)_{16}$, $(1123)_{16}$,
 $(11F3)_{16}$.

Q28. Explain how to convert binary into hexa decimal.

Ans:

Binary to Hexadecimal Conversion

The base numbers of binary and hexadecimal are 2 and 16, respectively. In a binary number, the pair of four bits is equal to one hexadecimal digit. There are also only two steps to convert a binary number into a hexadecimal number which are as follows:

1. In the first step, we have to make the pairs of four bits on both sides of the binary point. If there will be one, two, or three bits left in a pair of four bits pair, we add the required number of zeros on extreme sides.
2. In the second step, we write the hexadecimal digits corresponding to each pair.

Example 1: $(10110101011.0011)_2$

1. Firstly, we make pairs of four bits on both sides of the binary point.

111 1010 1011.0011

On the left side of the binary point, the first pair has three bits. To make it a complete pair of four bits, add one zero on the extreme side.

0111 1010 1011.0011

2. Then, we write the hexadecimal digits, which correspond to each pair.

$(011110101011.0011)_2 = (7AB.3)_{16}$

Q29. Explain, how can we convert a hexa decimal number system to other number systems

Ans :

Hexa-decimal to other Number System

Like binary, decimal, and octal, hexadecimal numbers can also be converted into other number systems. The process of converting hexadecimal to decimal differs from the remaining one. Let's start understanding how conversion is done.

Hexa-decimal to Decimal Conversion

The process of converting hexadecimal to decimal is the same as binary to decimal. The process starts from multiplying the digits of hexadecimal numbers with its corresponding positional weights. And lastly, we add all those products.

Let's take an example to understand how the conversion is done from hexadecimal to decimal.

Example 1: $(152A.25)_{16}$

Step 1: We multiply each digit of 152A.25 with its respective positional weight, and last we add the products of all the bits with its weight.

$$(152A.25)_{16} = (1 \times 16^3) + (5 \times 16^2) + (2 \times 16^1) + (A \times 16^0) + (2 \times 16^{-1}) + (5 \times 16^{-2})$$

$$(152A.25)_{16} = (1 \times 4096) + (5 \times 256) + (2 \times 16) + (10 \times 1) + (2 \times 16^{-1}) + (5 \times 16^{-2})$$

$$(152A.25)_{16} = 4096 + 1280 + 32 + 10 + (2 \times 16^{-1}) + (5 \times 16^{-2})$$

$$(152A.25)_{16} = 5418 + 0.125 + 0.125$$

$$(152A.25)_{16} = 5418.14453125$$

So, the decimal number of the hexadecimal number 152A.25 is 5418.14453125

Hexadecimal to Binary Conversion

The process of converting hexadecimal to binary is the reverse process of binary to hexadecimal. We write the four bits binary code of each hexadecimal number digit.

Example 1: $(152A.25)_{16}$

We write the four-bit binary digit for 1, 5, A, 2, and 5.

$$(152A.25)_{16} = (0001 \ 0101 \ 0010 \ 1010.0010 \ 0101)_2$$

So, the binary number of the hexadecimal number 152A.25 is $(1010100101010.00100101)_2$

Hexadecimal to Octal Conversion

For converting hexadecimal to octal, there are two steps required to perform, which are as follows:

1. In the first step, we will find the binary equivalent of the hexadecimal number.
2. Next, we have to make the pairs of three bits on both sides of the binary point. If there will be one or two bits left in a pair of three bits pair, we add the required number of zeros on extreme sides and write the octal digits corresponding to each pair.

Example 1: $(152A.25)_{16}$

Step 1: We write the four-bit binary digit for 1, 5, 2, A, and 5.

$$(152A.25)_{16} = (0001 \ 0101 \ 0010 \ 1010.0010 \ 0101)_2$$

So, the binary number of hexadecimal number 152A.25 is $(0011010101010.010101)_2$

Step 2:

3. Then, we make pairs of three bits on both sides of the binary point.

001 010 100 101 010.001 001 010

4. Then, we write the octal digit, which corresponds to each pair.

$$(001010100101010.001001010)_2 = (12452.112)_8$$

So, the octal number of the hexadecimal number 152A.25 is 12452.112

Q30. Convert $(10A4.249)_{16}$ into binary, octal and decimal forms

Ans:

Base 16 to decimal calculation

$$(10A4.249)_{16} = (1 \times 16^3) + (0 \times 16^2) + (10 \times 16^1) + (4 \times 16^0) + (2 \times 16^{-1}) + (4 \times 16^{-2}) + (9 \times 16^{-3}) = (4260.142822265625)_{10}$$

Decimal to base 2 calculation

Multiply the number with the base raised to the power of decimals in result:

$$4260.142822265625 \times 2^5 = 136324.5703125$$

Divide by the base to get the digits from the remainders

Division	Quotient	Remainder (Digit)	Digit #
136324.5703125/2	68162	1	0
68162/2	34081	0	1
34081/2	17040	1	2
17040/2	8520	0	3
8520/2	4260	0	4
4260/2	2130	0	5
2130/2	1065	0	6
1065/2	532	1	7
532/2	266	0	8
266/2	133	0	9
133/2	66	1	10
66/2	33	0	11
33/2	16	1	12
16/2	8	0	13
8/2	4	0	14
4/2	2	0	15
2/2	1	0	16
1/2	0	1	17

$$= (1000010100100.001001001001)_2$$

Decimal to base 8 calculation

Multiply the number with the base raised to the power of decimals in result:

$$4260.142822265625 \times 8^5 = 139596360$$

Divide by the base to get the digits from the remainders

Division	Quotient	Remainder (Digit)	Digit #
139596360/8	17449545	0	0
17449545/8	2181193	1	1
2181193/8	272649	1	2
272649/8	34081	1	3
34081/8	4260	1	4
4260/8	532	4	5
532/8	66	4	6
66/8	8	2	7
8/8	1	0	8
1/8	0	1	9

$$= (10244.1111)_8$$

1.3 INTRODUCTION TO C LANGUAGE
1.3.1 Background

Q31. What Is C Language? Why C Is Called Mother Language?

Ans:

The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways :

1) C as a mother language

C language is considered as the mother language of all the modern programming languages because most of the compilers, JVMs, Kernels, etc. are written in C language, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

It provides the core concepts like the array, strings, functions, file handling, etc. that are being used in many languages like C++, Java, C#, etc.

2) C as a system programming language

A system programming language is used to create system software. C language is a system programming language because it can be used to do low-level programming (for example driver and kernel). It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

3) C as a procedural language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language specifies a series of steps for the program to solve the problem.

A procedural language breaks the program into functions, data structures, etc.

C is a procedural language. In C, variables and function prototypes must be declared before being used.

4) C as a structured programming language

A structured programming language is a subset of the procedural language. Structure means to break a program into parts or blocks so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

5) C as a mid-level programming language

C is considered as a middle-level language because it supports the feature of both low-level and high-level languages. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).

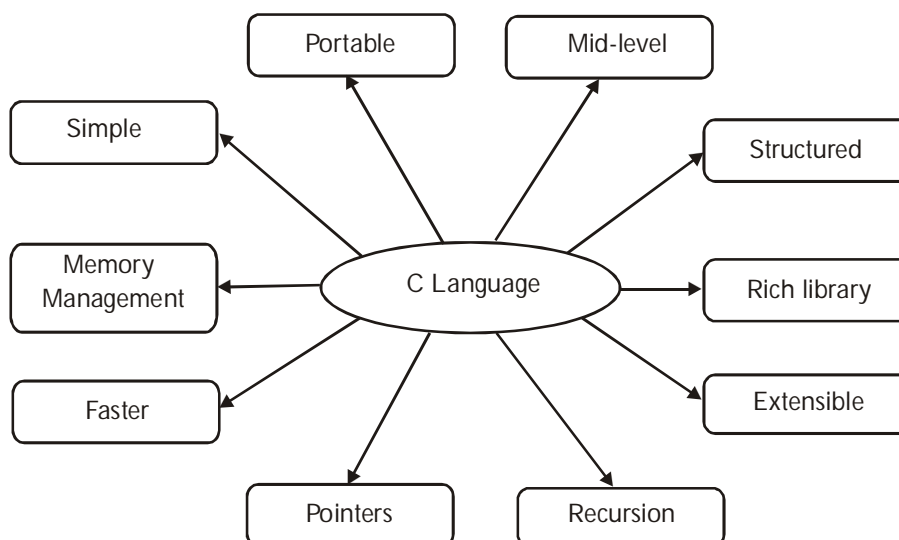
A Low-level language is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.

A High-Level language is not specific to one machine, i.e., machine independent. It is easy to understand.

Q32. Explain the features of C language.

Ans :

C is the widely used language. It provides many features that are given below



1) Simple

C is a simple language in the sense that it provides a structured approach (to break the problem into parts), the rich set of library functions, data types, etc.

2) Machine Independent or Portable

Unlike assembly language, c programs can be executed on different machines with some machine specific changes. Therefore, C is a machine independent language.

3) Mid-level programming language

Although, C is intended to do low-level programming. It is used to develop system applications such as kernel, driver, etc. It also supports the features of a high-level language. That is why it is known as mid-level language.

4) Structured programming language

C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify. Functions also provide code reusability.

5) Rich Library

C provides a lot of inbuilt functions that make the development fast.

6) Memory Management

It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the `free()` function.

7) Speed

The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

8) Pointer

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array, etc.

9) Recursion

In C, we can call the function within the function. It provides code reusability for every function. Recursion enables us to use the approach of backtracking.

10) Extensible

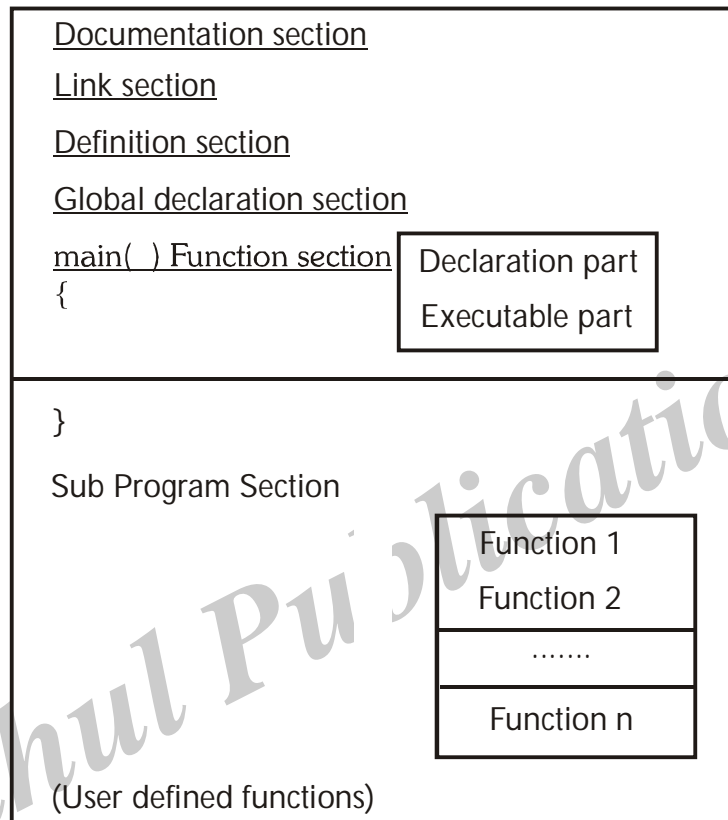
C language is extensible because it can easily adopt new features.

1.3.2 C Programs

Q33. What is the general structure of 'C' program and explain with example?

Ans:

Structure of C program



This section consists of a set of comment lines giving the name of the program, and other details. In which the programmer would like to user later.

Ex: `/*.....*/`

Link section: Link section provides instructions to the compiler to link functions from the system library.

Ex: `# include<stdio.h>`

`# include<conio.h>`

Definition section: Definition section defines all symbolic constants.

Ex: `# define A 10.`

Global declaration section

Some of the variables that are used in more than one function throughout the program are called global variables and declared outside of all the functions. This section declares all the user-defined functions.

Every C program must have one main () function section. This contains two parts.

(i) **Declaration part:** This part declares all the variables used in the executable part.

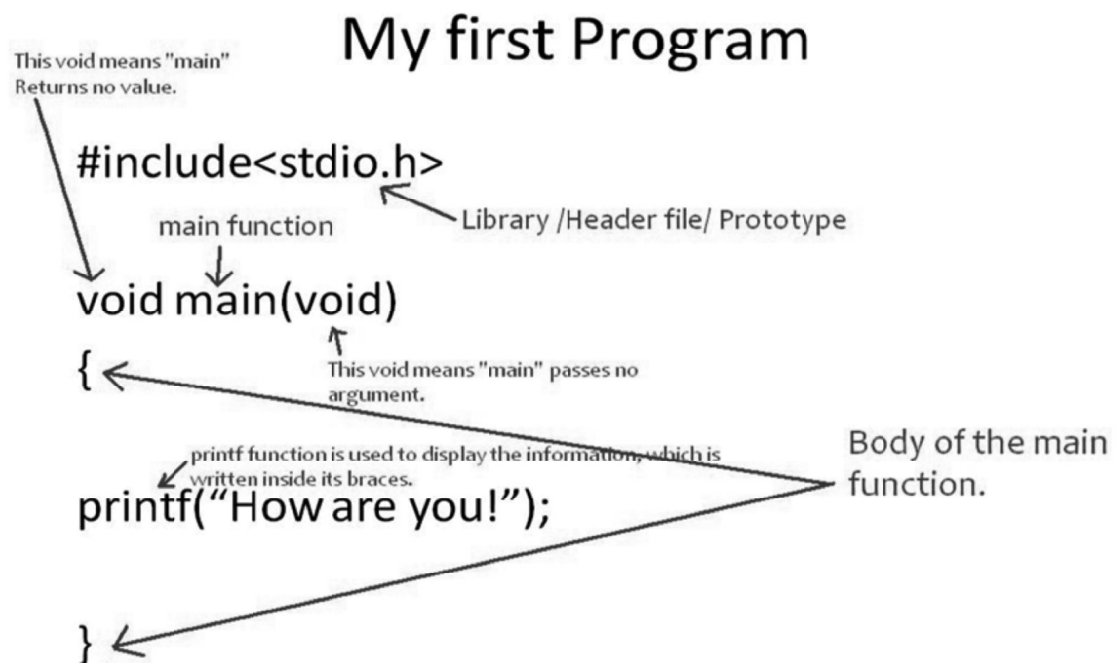
Ex: `inta,b;`

(ii) **Executable part:** This part contains at least one statement. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. All the statements in the declaration and executable parts end with a semicolon (;).

Sub program section

This section contains all the user-defined functions, that are called in the main () function. User-defined functions generally places immediately after the main() function, although they may appear in any order.

Ex:



1.3.3 Identifiers

Q34. What are the identifiers in C language. Give the rules to create identifiers

Ans:

In C language identifiers are the names given to variables, constants, functions and user-defined data. These identifiers are defined against a set of rules.

Rules for an Identifier

1. An Identifier can only have alphanumeric characters (a-z, A-Z, 0-9) and underscore (_).
2. The first character of an identifier can only contain alphabet (a-z, A-Z) or underscore (_).
3. Identifiers are also case sensitive in C. For example `name` and `Name` are two different identifiers in C.

4. Keywords are not allowed to be used as Identifiers.
5. No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.

Types of identifiers

- Internal identifier
- External identifier

Internal Identifier

If the identifier is not used in the external linkage, then it is known as an internal identifier. The internal identifiers can be local variables.

External Identifier

If the identifier is used in the external linkage, then it is known as an external identifier. The external identifiers can be function names, global variables.

Differences between Keyword and Identifier

Keyword	Identifier
Keyword is a pre-defined word	The identifier is a user - defined word
It must be written in a lowercase letter.	It can be written in both lowercase and uppercase letters.
Its meaning is pre-defined in and c compiler	Its meaning is not defined in the c compiler.
It is a combination of alphabetical characters.	It is a combination of alphanumeric characters.
It does not contain the underscore character.	Itc can contain the underscore character.

Let's understand through an example.

```
int main()
{
int a=10;
int A=20;
printf("Value of a is : %d",a);
printf("\nValue of A is :%d",A);
return 0;
}
```

Output

Value of a is : 10

Value of A is :20

The above output shows that the values of both the variables, 'a' and 'A' are different. Therefore, we conclude that the identifiers are case sensitive.

1.3.4 Data Types

Q35. What are known as data types

Ans:

Data type is the type of the data that are going to access within the program. C supports different data types. Each data type may have pre-defined memory requirement and storage representation. C supports 4 classes of data types.

1. Primary or (fundamental) data type(int, char, float, double)
2. User-defined data type(type def)
3. Derived data type(arrays, pointers, structures, unions)
4. Empty data type(void)- void type has no value.

1 byte = 8 bits (0's and 1's)

Q36. Explain data types in 'C'?

Ans:

1. Primary or (fundamental) data type

All C compilers support 4 fundamentals data types, they are

1. Integer (int)
2. Character(char)
3. Floating (float)
4. Double precision floating point(double)

1. Integer types

Integers are whole numbers with a range of values supported by a particular machine. Integers occupy one word of storage and since the word size of the machine vary. If we use 16 bit word length the size of an integer value is -32768 to +32767. In order to control over the range of numbers and storage space, C has 3 classes of integer storage, namely short, long, and unsigned.

2. Character type

Single character can be defined as a character (char) type data. Characters are usually stored in 8bits of internal storage. Two classes of char types are there.

Signed char, unsigned char.

Signed char(or) char 1 byte- -128 to +127 % c Unsigned char 1 byte 0 to 255 % c

3. Floating point types

Floating point (real) numbers are stored in 32 bits, with 6 digits of precision when accuracy provided by a float number is not sufficient

Float 4 bytes3, .4e-38 to 3.4e+38%f

4. Double precision type

Double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision no.s. Double type represents the same data type that float represents, but with a greater precision. To extend the precision further, we may use long double which uses 80 bits.

double 8 bytes 1.7e-308 to 1.7e+308% lf

long double 10 bytes 3.4e-4932 to 1.1e+4932% lf

Variable Type	Keyword	Bytes Required	Range	Format
Character (signed)	Char	1	-128 to + 127	%c
Integer (signed)	Int	2	-32768 to + 32767	%d
Float (signed)	Float	4	-3.4e38 to + 3.4e38	%f
Double	Double	8	-1.7e308 to + 1.7e308	%lf
Long integer (signed)	Long	4	2,147, 483, 648 to 2,147,438,647	%ld
Character (unsigned)	Unsigned char	1	0 to 225	%c
Integer (unsigned)	Unsigned int	2	0 to 65535	%u
Unsigned long Integer	unsigned long	4	0 to 4,294, 967,295	%lu
Long double	Long double	10	-1.7e932 to +1.7e932	%Lf

2. User defined data types:

C –supports a feature known as “type definition” that allows users define an identifier that would represents an existing type.

Ex: typedef data-type identifier;

Where data-type indicates the existing datatype

Identifier indicates the new name that is given to the data type.

Ex: typedef int marks;

Marks m1, m2, m3;

typedef can create a new data type ,it can rename the existing datatype. The main advantage of typedef is that we can create meaningful datatype names for increasing the readability of the program.

Another user-defined datatype is “enumerated data type(enum)”

Syntax : enum identifier {value1, value2,.....valuen}

Where identifier is user-defined datatype which is used to declare variables that can have one of the values enclosed within the braces. Value1 ,value2,.,valuen all these are known as enumeration constants.

Ex: enum identifier v1, v2,.....vn

V1=value3;

V2=value1;.....

Ex: enum day {Monday,Tuesday..... sunday};

```
Enum day week-f, week-end
```

```
Week-f = Monday
```

(or)

```
enum day{Monday...Sunday} week-f, week-end;
```

1.3.5 Variables

Q37. What is a variable? and write the rules for constructing variable?

Ans:

Variables

It is a data name that may be used to store a data value. It can't be changed during the execution of a program. A variable may take different values at different times during execution.

Rules

Variable names may consist of letters, digits and under score (_) character.

- First char must be an alphabet or an '-'
- Length of the variable can't exceed up to 8 characters, some C compilers can be recognized up to 31 characters.
- No , and no white space, special symbols allowed.
- Variable name should not be a keyword.
- Both upper & lower case letters are used.
 - Ex:- mark, sum1, tot-value, delhi are valid
 - Price\$, group one, char are invalid

Q38. How to declare and initialize a variable?

Ans:

(Imp.)

Declaration does two things

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

The declaration of variables must be done before they are used in the program.

The syntax for declaring a variable is as follows:

Data-type v1, v2, ..., vn;

v1, v2, ..., vn are the names of variables. Variables are separated by commas. A declaration statement must end with a semicolon. For example, valid declarations are:

```
int count;
```

```
int number, total;
```

```
double ratio;
```

The simplest declaration of a variable is shown in the following code fragment:

Ex:

```
main()/*.....Program Name.....*/  
{  
/*.....Declaration.....*/  
floatx,y;  
int code;  
shortint count;  
longint amount;  
double deviation;  
unsigned n;  
char c;  
/*.....Computation.....*/  
/*.....Program ends.....*/
```

Initialization of variable

Initialize a variable in c to assign it a starting value. Without this we can't get whatever happened to memory at that moment.

C does not initialize variables automatically. So if you do not initialize them properly, you can get unexpected results. Fortunately, C makes it easy to initialize variables when you declare them.

For Example :

```
int x=45;  
intmonth_lengths[] = {23,34,43,56,32,12,24};  
structrole = { "Hamlet", 7, FALSE, "Prince of Denmark ", "Kenneth Branagh"};
```

Note : The initialization of variable is a good process in programming.

1.3.6 Constants

Q39. Describe the different types of constants in C with example?

(OR)

What are the rules for creating C constants explain with example?

Ans :

Types of C Constants

1. Integer constants
2. Real constants
3. Character constants
4. String constants

String constants

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

Examples are:

"HELLO!"

"1979"

"welcome"

"? |"

"5+3"

"X"

Rules of Constructing Integer Constants

(a) an integer constant must have at least one digit.

(b) It must not have a decimal point.

(c) It can be either positive or negative.

(d) The allowable range for constants is -32768 to 32767

In fact the range of integer constants depends upon compiler.

For ex. 435

+786

-7000

Rules of Constructing Real Constants

(a) A real constants must have at least one digit

(b) it must have a decimal point.

(c) it could be either positive or negative.

(d) default sign is positive.

For ex. +325.34 426.0

In exponential form of representation, the real constants is represented in two parts. The part appearing before 'e' is called mantissa where as the part following 'e' is called exponent.

Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Ex. $+3.2e-5$

Rules of Constructing Character Constants

- (a) A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
- (b) The maximum length of character constant can be one character.

Ex : 'A'

1.3.7 Input / Output Statements**Q40. Explain standard I/O functions in C**

Ans :

(Imp.)

C Input output function

C programming language provides many of the built-in functions to read given input and write data on screen, printer or in any file.

scanf() and printf() functions

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i;
    printf("Enter a value");
    scanf("%d",&i);
    printf("\nYou entered: %d",i);
    getch();
}
```

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered.

Note

printf() function returns the number of characters printed by it, and scanf() returns the number of characters read by it.

```
int i = printf("study C ");
```

In this program i will get 12 as value, because studytonight has 12 characters.

getchar() & putchar() functions

The getchar() function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one characters. The putchar() function prints the character passed to it on the screen and returns the same character. This function puts only single character at a time. In case you want to display more than one characters, use putchar() method in the loop.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int c;
    printf("Enter a character");
    c=getchar();
    putchar(c);
    getch();
}
```

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered.

gets() & puts() functions

The gets() function reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF (end of file). The puts() function writes the string s and a trailing newline to stdout.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    charstr[100];
    printf("Enter a string");
    gets(str);
    puts(str);
    getch();
}
```

When you will compile the above code, it will ask you to enter a string. When you will enter the string, it will display the value you have entered.

1.3.8 Arithmetic Operators

Q41. What is an operator and List different categories of C operators based on their functionality? Give examples?

Ans:

Operators

An operator is a symbol performs certain mathematical or logical manipulations. Operators are used in programs to manipulate data variables.

C operators can be classified into a number of categories, they are

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

1. Arithmetic Operators

The arithmetic operators are

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

Here a and b are operands, assign values for a=14 and b=4 we have the following results

$$a - b = 10$$

$$a + b = 18 \quad a * b = 56$$

$$a / b = 3(\text{coefficient}) \quad a \% b = 2(\text{remainder})$$

2. Relational Operators:

Relational operators are used for comparing two quantities, and take certain decision. For example we may compare the age of two persons or the price of two items....these comparisons can be done with the help of relational operators.

An expression containing a relational operator is termed as a relational expression. The value of a relational expression is either one or zero. It is one if the specified relation is true and zero if the relation is false.

Ex:- $13 < 34$ (true) $23 > 35$ (false)

C supports 6 relational operators

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

Ex: $4.5 <= 10$ (true)

$6.5 < -10$ (false)

$10 < 4 + 12$ (true)

When arithmetic expression are used on either side of a relational operator, the arithmetic expression will be evaluated first and then the results compared, that means arithmetic operators have a higher priority over relational operators.

3. Logical Operator

C has 3 logical operators. The logical operators are used when we want to test more than one condition and make decisions.

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

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

Ex:- `a > b && x == 45`

This expression combines two or more relational expressions, is termed as a logical expression or a compound relational expression. The logical expression given above is true only if `a > b` is true and `x == 10` is true. if both of them are false the expression is false.

OP1	OP2	&&	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Some examples of logical expression

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

4. Assignment operator

These operators are used to assign the result of an expression to a variable. The usual assignment operator is `'='`

`V op = exp;`

Where `v` is a variable, `exp` is an expression and `op` is a C binary arithmetic operator. The operator `op=` is known as the shorthand assignment operator.

The assignment statement is `V op=exp;`

Ex: `X = X + (Y + 1);`

`a* = a; - - - - a = a*a;`

5. Increment and Decrement operator

`++` and `--` are increment and decrement operators in C. The operator `++` adds 1 to the operand, while `--` subtracts 1. both are unary operators.

`++m;` or `m++;` is equal to `m = m + 1` (`m++ = 1;`) `--- m;` or `m--;` is equal to `m = m - 1` (`m-- = 1;`)

We use the increment and decrement statements in for and while loops extensively.

Ex:- `m = 5;`

`Y = ++m;` the value of `y = 6` and `m = 6`.

Suppose if we write the above statement as `m = 5; y = m++;` the value of `y = 5` and `m = 6`.

A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

6. Conditional operator

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

`exp ? exp1 : exp2`

Where `exp1`, `exp2` and `exp3` are expressions,

The operator `?:` works as follows: `exp1` is evaluated first. If it is non-zero (true), then the

expression `exp2` is evaluated and becomes the value of the expression. If `exp1` is false, `exp3` is

evaluated and its value becomes the value of the expression.

Ex:- `a = 10; b = 45;`

`X = (a > b) ? a : b;`

o/p:- X value of b (45).

7. Bitwise Operator:

C supports a special operator known as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits, or shifting them right to left. Bitwise operators may not be applied to float or double.

Operator	Meaning
<code>&</code>	Bitwise AND
<code> </code>	Bitwise OR
<code>^</code>	Bitwise exclusive OR
<code><<</code>	Shift left
<code>>></code>	Shift right

8. Special operators

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

The comma operator: The comma operator is used to link the related expression together. A comma linked list of expressions is evaluated left to right and the value of right- most expression is the value of the combined expression.

For example, the statement `Value = (x=10, y=5, x+y);`

In for loops: `for (n=1 , m=10, n<=m; n++ , m++);`

The sizeof operator: The sizeof is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies. The operand may be variable, a constant or a data type qualifier.

`m = sizeof (sum);`

`n = sizeof (long int);`

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

1.4 EXPRESSIONS

1.4.1 Evaluating Expressions

Q42. How to evaluate C Expression ? Explain with the help of an example

Ans:

Expression evaluation

In c language expression evaluation is mainly depends on priority and associativity.

Priority

This represents the evaluation of expression starts from "what" operator.

Associativity

It represents which operator should be evaluated first if an expression is containing more than one operator with same priority.

Operator	Priority	Associativity
{ }, (), []	1	Left to right
++, --, !	2	Right to left
*, /, %	3	Left to right
+, -	4	Left to right
<, <=, >, >=, ==, !=	5	Left to right
& &	6	Left to right
	7	Left to right
?:	8	Right to left
=, +, -, *, /, %, =	9	Right to left

Example 1

$$\begin{array}{r}
 10 - 3 \% 8 + 6 / 4 \\
 \quad \boxed{} \\
 10 - 3 + 6 / 4 \\
 \quad \quad \boxed{} \\
 10 - 3 + 1 \\
 \quad \boxed{} \\
 7 + 1 \\
 \quad \boxed{} \\
 8
 \end{array}$$
Example 2

$$\begin{array}{r}
 17 - 8 / 4 * 2 + 3 - ++a \\
 \quad \quad \quad \boxed{} \\
 17 - 8 / 4 * 2 + 3 - 6 \\
 \quad \quad \boxed{} \\
 17 - 2 * 2 + 3 - 6 \\
 \quad \quad \boxed{} \\
 17 - 4 + 3 - 6 \\
 \quad \boxed{} \\
 13 + 4 - 6 \\
 \quad \boxed{} \\
 16 - 6 \\
 \quad \boxed{} \\
 10
 \end{array}$$
1.4.2 Precedence And Associativity Of Operators**Q43. Define precedence and associativity? Give an example?****(OR)**

Explain the hierarchy (priority) and associativity(clubbing)of operators in 'C' with example?

*Ans:***Operator Precedence**

Various relational operators have different priorities or precedence. If an arithmetic expression contains more operators then the execution will be performed according to their properties. The precedence is set for different operators in C.

Type of operator	Operators	Associativity
Unary operators	+, -, !, ++, --, type, size of	Right to left
Arithmetic operators	*, /, %, +, -	Left to right
Bit – manipulation operators	<<, >>	Left to right
Relational operators	>, <, >=, <=, ==, !=	Left to right
Logical operators	&&,	Left to right
Conditional operators	?, :	Left to right
Assignment operators	=, +=, -=, *=, /=, %=	Right to left

➤ Important note

Precedence rules decide the order in which different operators are applied. Associativity rule decides the order in which multiple occurrences of the same level operator are applied.

Hierarchy Of Operations In C

There are some operators which are given below with their mean. The higher the position of an operator is, higher is its priority.

Operator	Type
!	Logical NOT
*/ %	Arithmetic and modulus
+ -	Arithmetic
< > <= >=	Relational
== !=	Relational
&&	Logical AND
	Logical OR
=	Assignment

Associativity Of Operator

When an expression contains two operators of equal priority the tie between them is settled using the associativity of the operators.

Associativity can be of two types—Left to Right or Right to Left.

Left to Right associativity means that the left operand must be unambiguous. Unambiguous in what sense? It must not be involved in evaluation of any other sub-expression. Similarly, in case of Right to Left associativity the right operand must be unambiguous. Let us understand this with an example.

Consider the expression $a = 3 / 2 * 5$;

Here there is a tie between operators of same priority, that is between / and *. This tie is settled using the associativity of / and *. But both enjoy Left to Right associativity.

While executing an arithmetic statement, which has two or more operators, we may have some problem as to how exactly does it get executed.

Priority	Operators	Description
1st	*, /, %	multiplication, division, modular
2nd	+, -	addition, subtraction
3rd	=	Assignment

For Example

$i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8$ $i = 6 / 4 + 4 + 8 - 2 + 5 / 8$

$i = 1 + 4 / 4 + 8 - 2 + 5 / 8$ $i = 1 + 1 + 8 - 2 + 5 / 8$

$i = 1 + 1 + 8 - 2 + 0$ $i = 2 + 8 - 2 + 0$

$i = 10 - 2 + 0$ $i = 8 + 0$ $i = 8$

Step 5: $x = 10$

1.4.3 Type Conversions

Q47. What is type conversion?

Ans :

Type conversions: converting a variable value or a constant value temporarily from one datatype to other data type for the purpose of calculation is known as type conversion.

Q48. Explain the types of type conversions in C with example?

Ans :

(Imp.)

There are two types of conversions

1. automatic type conversion (or) implicit
2. casting a value (or) explicit

1. Implicit

In this higher data type can be converted into lower data type.

- Float value can be converted into integral value by removing the fractional part.
- Double value can be converted into float value by rounding of the digits.
- Long int can be converted into int value by removing higher order bits.

2. Explicit

In this type of conversion, the programmer can convert one data type to other datatype explicitly.

Syntax: (datatype) (expression)

Expression can be a constant or a variable

Ex: `y = (int) (a+b)`

`y= cos(double(x))`

`double a = 6.5`

`double b = 6.5`

`int result = (int) (a) + (int) (b)`

`result = 12` instead of 13.

`int a=10`

`float(a)->10.00000`

Rahul Publications

Short Question & Answers

1. Define computer?

Ans:

A computer is an electronic device that can be programmed to accept data (input), process it and generate result (output). A computer along with additional hardware and software together is called a computer system. A computer system primarily comprises of a central processing unit, memory, input/output devices, and storage devices. All these components function together as a single unit to deliver the desired output. A computer system comes in various forms and sizes. It can vary from a high-end server to a personal desktop, laptop, tablet computer, or smartphone.

2. What is SDLC?

Ans:

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

3. What is Flowchart ?

Ans:

It is a symbolic diagram of operations sequence, dataflow, control flow and processing logic in information processing.

The symbols used are simple and easy to learn.

It is a very helpful tool for programmers and beginners.

Purpose of a Flowchart

- Provides Communication.
- Provides an Overview.
- Shows all elements and their relationships.
- Quick method of showing Program flow.
- Checks Program logic.
- Facilitates Coding.

4. What is number system?

Ans :

In a digital system, the system can understand only the optional number system. In these systems, digits symbols are used to represent different values, depending on the index from which it settled in the number system.

In simple terms, for representing the information, we use the number system in the digital system.

The digit value in the number system is calculated using:

1. The digit
2. The index, where the digit is present in the number.
3. Finally, the base numbers, the total number of digits available in the number system.

5. What is Binary Number System?

Ans:

Generally, a binary number system is used in the digital computers. In this number system, it carries only two digits, either 0 or 1. There are two types of electronic pulses present in a binary number system. The first one is the absence of an electronic pulse representing '0' and second one is the presence of electronic pulse representing '1'. Each digit is

known as a bit. A four-bit collection (1101) is known as a nibble, and a collection of eight bits (11001010) is known as a byte. The location of a digit in a binary number represents a specific power of the base (2) of the number system.

6. What is decimal number system?

Ans:

Decimal Number System

The decimal numbers are used in our day to day life. The decimal number system contains ten digits from 0 to 9 (base 10). Here, the successive place value or position, left to the decimal point holds units, tens, hundreds, thousands, and so on.

The position in the decimal number system specifies the power of the base (10). The 0 is the minimum value of the digit, and 9 is the maximum value of the digit.

7. What is hexa decimal number system?

Ans:

It is another technique to represent the number in the digital system called the hexadecimal number system. The number system has a base of 16 means there are total 16 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) used for representing a number. The single-bit representation of decimal values 10, 11, 12, 13, 14, and 15 are represented by A, B, C, D, E, and F. Only 4 bits are required for representing a number in a hexadecimal number.

8. What Is C Language? Why C Is Called Mother Language?

Ans:

The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways :

1) C as a mother language

C language is considered as the mother language of all the modern programming languages because most of the compilers, JVMs, Kernels, etc. are written in C language, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

It provides the core concepts like the array, strings, functions, file handling, etc. that are being used in many languages like C++, Java, C#, etc.

2) C as a system programming language

A system programming language is used to create system software. C language is a system programming language because it can be used to do low-level programming (for example driver and kernel). It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

3) C as a procedural language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language specifies a series of steps for the program to solve the problem.

A procedural language breaks the program into functions, data structures, etc.

C is a procedural language. In C, variables and function prototypes must be declared before being used.

4) C as a structured programming language

A structured programming language is a subset of the procedural language. Structure means to break a program into parts or blocks so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

9. What is a variable? and write the rules for constructing variable?

Ans:

Variables

It is a data name that may be used to store a data value. It can't be changed during the execution of a program. A variable may take different values at different times during execution.

Rules

Variable names may consist of letters, digits and underscore (_) character.

- First char must be an alphabet or an '_'
- Length of the variable can't exceed up to 8 characters, some C compilers can be recognized up to 31 characters.
- No , and no white space, special symbols allowed.
- Variable name should not be a keyword.
- Both upper & lower case letters are used.
 - Ex:- mark, sum1, tot-value, delhi are valid
 - Price\$, group one, char are invalid

10. Explain the types of type conversions in C with example?

Ans :

There are two types of conversions

1. automatic type conversion (or) implicit
2. casting a value (or) explicit

1. Implicit

In this higher data type can be converted into lower data type.

- Float value can be converted into integral value by removing the fractional part.
- Double value can be converted into float value by rounding of the digits.

- Long int can be converted into int value by removing higher order bits.

2. Explicit

In this type of conversion, the programmer can convert one data type to other datatype explicitly.

Syntax: (datatype) (expression)

Expression can be a constant or a variable

Ex: y = (int) (a+b)

y = cos(double(x))

double a = 6.5

double b = 6.5

int result = (int) (a) + (int) (b)

result = 12 instead of 13.

int a = 10

float(a) -> 10.00000

Choose the Correct Answers

1. Which of the following converts a C program's source code into machine language: [a]
(a) compiler (b) Editor
(c) operating system (d) none of the above
2. The operating system manages [d]
(a) Memory (b) Processor
(c) Disk and I/O devices (d) All of the above
3. Which of the following is not a valid variable name declaration? [d]
(a) int __a3; (b) int __3a;
(c) int __A3; (d) None of the mentioned
4. Which of the following cannot be a variable name in C? [a]
(a) volatile (b) true
(c) friend (d) export
5. The correct order of evaluation for the expression "z = x + y * z / 4 % 2 - 1" [d]
(a) $- * / \% = + -$ (b) $- / * \% - + =$
(c) $- + = * \% /$ (d) $- * / \% + - =$
6. What is the size of an int data type? [d]
(a) 4 Bytes (b) 8 Bytes
(c) Depends on the system/compiler (d) Cannot be determined
7. Which of the datatypes have size that is variable? [b]
(a) int (b) struct
(c) float (d) double
8. Which of the following declaration is not supported by C? [a]
(a) String str; (b) char *str;
(c) float str = 3e2; (d) Both (a) and (c)
9. What is the value of x in this C code? [c]

```
#include <stdio.h>
void main( )
{
int x = 5*9/3+9
}
```


(a) 3.75 (b) Depends on compiler
(c) 24 (d) 3
10. Relational operators cannot be used on: [a]
(a) structure (b) long
(c) strings (d) float

Fill in the blanks

1. A c program is basically a collection of _____
2. C language is well suited for _____ programming.
3. A _____ character instructs the computer to move the control to the next line.
4. C program execution begins from _____
5. Local variable which exists and retains its value even after the control is transferred to the calling function is, _____ storage class.
6. The operator "++" is know as _____ operator.
7. The _____ operator can be used to determine the length of array and structures.
8. The standard mathematical functions are included in the _____ header file.
9. _____ function can be used to read a single character.
10. Logical and is performed with _____ operator.

ANSWERS

1. functions
2. Structure
3. newline
4. main ()
5. static
6. Increment
7. size of
8. math.h
9. getch()
10. &&

One Mark Answers

1. Personal Computing environment.

Ans:

The personal computer is a small computer with a microprocessor, designed for use by an individual, used in the home, small businesses, etc.

2. Client /Server Environment.

Ans:

Client /Server computing environment split the computing function between central computer and user computers. The client/server environment is divided into client and server.

3. High Level Languages.

Ans:

These are known as third generation programming language. High-level languages allow us to write programs using instructions resembling everyday spoken language.

4. Binary Number System.

Ans:

Generally, a binary number system is used in the digital computers. In this number system, it carries only two digits, either 0 or 1.

5. Octal number system.

Ans :

There are only eight possible digit values to represent a number.

6. C Language.

Ans :

C programming is considered as the base for other programming languages, that is why it is known as mother language.

7. Identifiers in C language.

Ans:

In C language identifiers are the names given to variables, constants, functions and user-defined data. These identifiers are defined against a set of rules.

8. Data types

Ans:

Data type is the type of the data that are going to access within the program.

9. Operators

Ans:

An operator is a symbol performs certain mathematical or logical manipulations.

UNIT II

Conditional Control Statements: Bitwise Operators, Relational and Logical Operators, If, IfElse, Switch-Statement and Examples. Loop Control Statements: For, While, Do-While and Examples. Continue, Break and Goto statements

Functions: Function Basics, User-defined Functions, Inter Function Communication, Standard Functions, Methods of Parameter Passing.

Recursion- Recursive Functions.

Storage Classes: Auto, Register, Static, Extern, Scope Rules, and Type Qualifiers

2.1 CONDITIONAL CONTROL STATEMENTS

2.1.1 Bitwise operators, Relational and Logical Operators

Q1. What is Operator? List out various types of Operators supported by C?

Ans :

(Imp.)

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Ternary or Conditional Operators
6. Assignment Operator
7. Misc Operator

1. Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values

The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then–

Operator	Description	Example
+	Adds two operands.	A + B = 30
–	Subtracts second operand from the first.	A – B = – 10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
–	Decrement operator decreases the integer value by one.	A – = 9

2. Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

The following table shows all the relational operators supported by C. Assume variable A hold 10 and variable B holds 20 then

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	

3. Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

4. Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for $\&$, $|$, and \wedge is as follows -

p	q	$p \& q$	$p q$	$p \wedge q$
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then -

Operator	Description	Example
$\&$	Binary AND Operator copies a bit to the result if it exists in both operands.	$(A \& B) = 12$, i.e., 0000 1100
$ $	Binary OR Operator copies a bit if it exists in either operand.	$(A B) = 61$, i.e., 00111101
\wedge	Binary XOR Operator copies the bit if it is set in one operand but not both.	$(A \wedge B) = 49$, i.e., 00110001
\sim	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	$(\sim A) = \sim(60)$, i.e., - 0111101
$<<$	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	$A << 2 = 240$ i.e., 11110000
$>>$	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	$A >> 2 = 15$ i.e., 00001111

5. Ternary (or) Conditional Operators

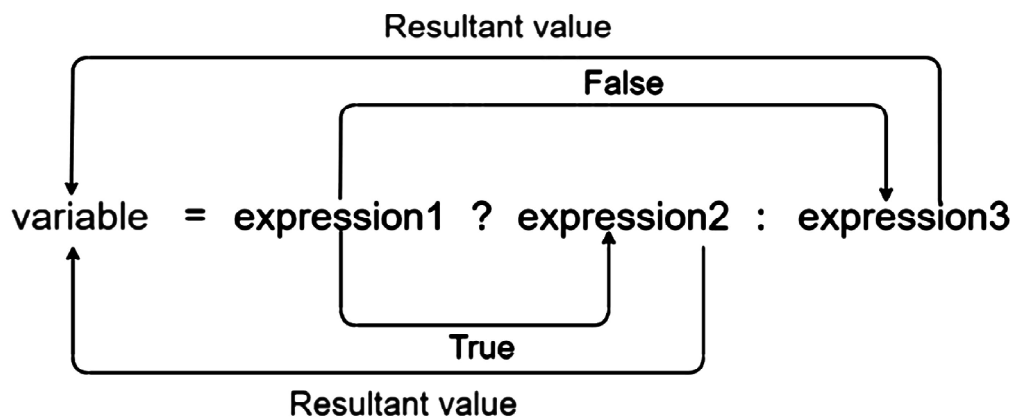
The conditional operator is also known as a ternary operator. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and '∴'.

As conditional operator works on three operands, so it is also known as the ternary operator.

The behavior of the conditional operator is similar to the 'if-else' statement as 'if-else' statement is also a decision-making statement.

Syntax of a conditional operator

Expression1? expression2: expression3;



- In the above syntax, the expression1 is a Boolean condition that can be either true or false value.
- If the expression1 results into a true value, then the expression2 will execute.
- The expression2 is said to be true only when it returns a non-zero value.
- If the expression1 returns false value then the expression3 will execute.
- The expression3 is said to be false only when it returns zero value.

6. Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

The following table lists the assignment operators supported by the C language –

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A

<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
=	Bitwise inclusive OR and assignment operator.	C = 2 is same as C = C 2

7. Misc Operators → sizeof & ternary

Besides the operators discussed above, there are a few other important operators including sizeof and ?: supported by the C Language.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
?:	Conditional Expression.	If Condition is true ? then value X: otherwise valueY

Q2. Explain in detail Bitwise operators in C.

Ans :

The bitwise operators are the operators used to perform the operations on the data at the bit-level. When we perform the bitwise operations, then it is also known as bit-level programming. It consists of two digits, either 0 or 1. It is mainly used in numerical computations to make the calculations faster.

We have different types of bitwise operators in the C programming language. The following is the list of the bitwise operators:

Operator	Description
&	Bitwise AND operator
	Bitwise OR operator
^	Bitwise exclusive OR operator
~	One's complement operator (unary operator)
<<	Left shift operator
>>	Right shift operator

Let's look at the truth table of the bitwise operators.

X	Y	X & Y	X Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	1

(i) Bitwise AND operator

Bitwise AND operator is denoted by the single ampersand sign (&). Two integer operands are written on both sides of the (&) operator. If the corresponding bits of both the operands are 1, then the output of the bitwise AND operation is 1; otherwise, the output would be 0.

For example,

We have two variables a and b.

a = 6;

b = 4;

The binary representation of the above two variables are given below:

a = 0110

b = 0100

When we apply the bitwise AND operation in the above two variables, i.e., a & b, the output would be:

Result = 0100

As we can observe from the above result that bits of both the variables are compared one by one. If the bit of both the variables is 1 then the output would be 1, otherwise 0.

Let's understand the bitwise AND operator through the program.

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

```
int main()
```

```
{
```

```
int a=6, b=14; // variable declarations
```

```
printf("The output of the Bitwise AND operator a&b is %d",a&b);
```

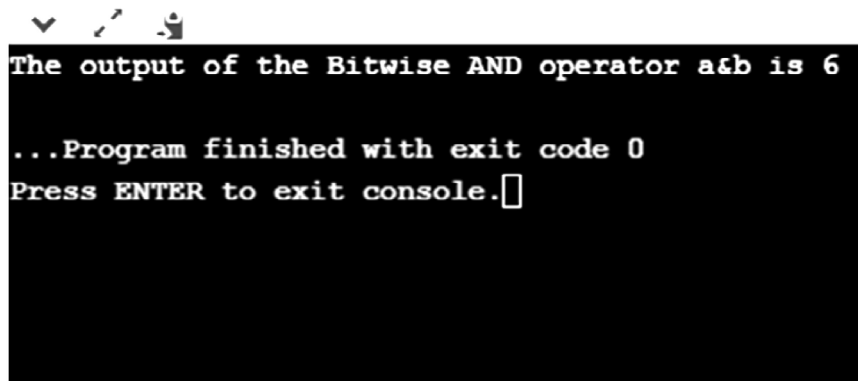
```
return 0;
```

```
}
```

In the above code, we have created two variables, i.e., 'a' and 'b'. The values of 'a' and 'b' are 6 and 14 respectively. The binary value of 'a' and 'b' are 0110 and 1110, respectively. When we apply the AND operator between these two variables,

a AND b = 0110 & 1110 = 0110

Output



```
The output of the Bitwise AND operator a&b is 6
...Program finished with exit code 0
Press ENTER to exit console.
```

(ii) Bitwise OR operator

The bitwise OR operator is represented by a single vertical sign ($|$). Two integer operands are written on both sides of the ($|$) symbol. If the bit value of any of the operand is 1, then the output would be 1, otherwise 0.

For example,

We consider two variables,

a = 23;

b = 10;

The binary representation of the above two variables would be:

a = 0001 0111

b = 0000 1010

When we apply the bitwise OR operator in the above two variables, i.e., $a|b$, then the output would be:

Result = 0001 1111

As we can observe from the above result that the bits of both the operands are compared one by one; if the value of either bit is 1, then the output would be 1 otherwise 0.

Let's understand the bitwise OR operator through a program.

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

```
int main()
```

```
{
```

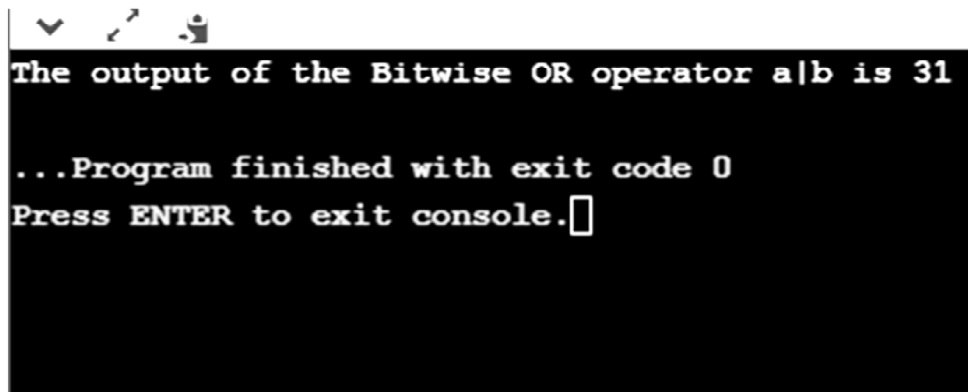
```
int a=23,b=10; // variable declarations
```

```
printf("The output of the Bitwise OR operator a|b is %d",a|b);
```

```
return 0;
```

```
}
```

Output



```
The output of the Bitwise OR operator a|b is 31

...Program finished with exit code 0
Press ENTER to exit console.
```

(iii) Bitwise exclusive OR operator

Bitwise exclusive OR operator is denoted by (^) symbol. Two operands are written on both sides of the exclusive OR operator. If the corresponding bit of any of the operand is 1 then the output would be 1, otherwise 0.

For example,

We consider two variables a and b,

a = 12;

b = 10;

The binary representation of the above two variables would be:

a = 0000 1100

b = 0000 1010

When we apply the bitwise exclusive OR operator in the above two variables ($a \wedge b$), then the result would be:

Result = 0000 1110

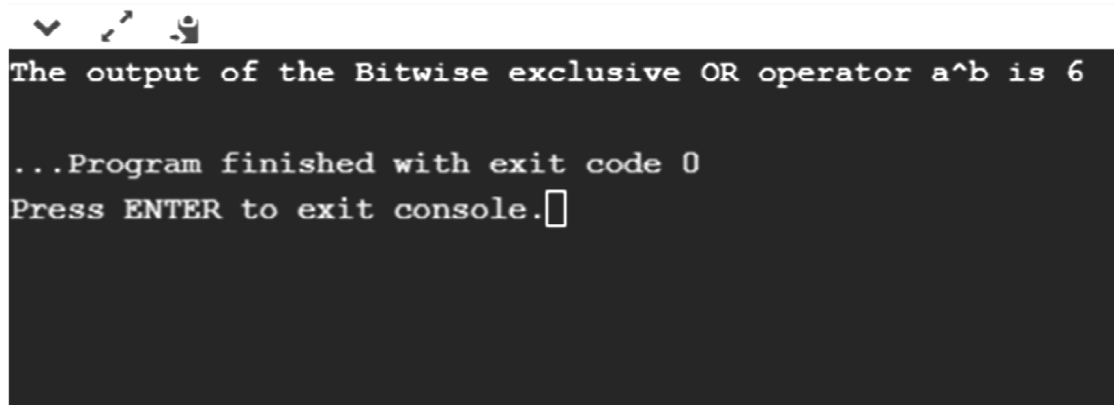
As we can observe from the above result that the bits of both the operands are compared one by one; if the corresponding bit value of any of the operand is 1, then the output would be 1 otherwise 0.

Let's understand the bitwise exclusive OR operator through a program.

```
#include <stdio.h>

int main()
{
    int a=12,b=10; // variable declarations
    printf("The output of the Bitwise exclusive OR operator  $a \wedge b$  is %d",a^b);
    return 0;
}
```

Output



```
✓ ↗ 🧑  
The output of the Bitwise exclusive OR operator a^b is 6  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

(iv) Bitwise shift operators

Two types of bitwise shift operators exist in C programming. The bitwise shift operators will shift the bits either on the left-side or right-side. Therefore, we can say that the bitwise shift operator is divided into two categories:

- Left-shift operator
- Right-shift operator

(v) Left-shift operator

It is an operator that shifts the number of bits to the left-side.

Syntax of the left-shift operator is given below:

Operand << n

Where,

Operand is an integer expression on which we apply the left-shift operation.

n is the number of bits to be shifted.

In the case of Left-shift operator, 'n' bits will be shifted on the left-side. The 'n' bits on the left side will be popped out, and 'n' bits on the right-side are filled with 0.

For example,

Suppose we have a statement:

```
int a = 5;
```

The binary representation of 'a' is given below:

```
a = 0101
```

If we want to left-shift the above representation by 2, then the statement would be:

```
a << 2;
```

```
0101<<2 = 00010100
```

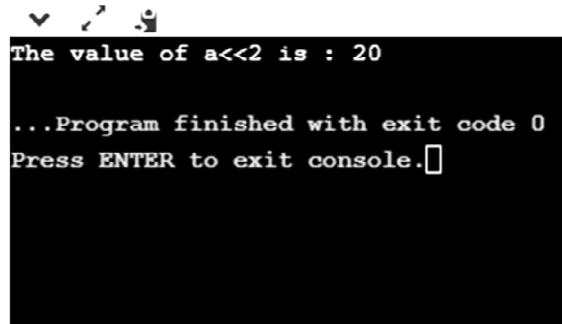
Let's understand through a program.

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

```

int a=5; // variable initialization
printf("The value of a<<2 is : %d ", a<<2);
return 0;
}
Output

```



```

The value of a<<2 is : 20

...Program finished with exit code 0
Press ENTER to exit console.

```

Right-shift operator

It is an operator that shifts the number of bits to the right side.

Syntax of the right-shift operator is given below:

Operand >> n;

Where,

Operand is an integer expression on which we apply the right-shift operation.

N is the number of bits to be shifted.

In the case of the right-shift operator, 'n' bits will be shifted on the right-side. The 'n' bits on the right-side will be popped out, and 'n' bits on the left-side are filled with 0.

For example,

Suppose we have a statement,

```
int a = 7;
```

The binary representation of the above variable would be:

a = 0111

If we want to right-shift the above representation by 2, then the statement would be:

```
a>>2;
```

0000 0111 >> 2 = 0000 0001

Let's understand through a program.

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

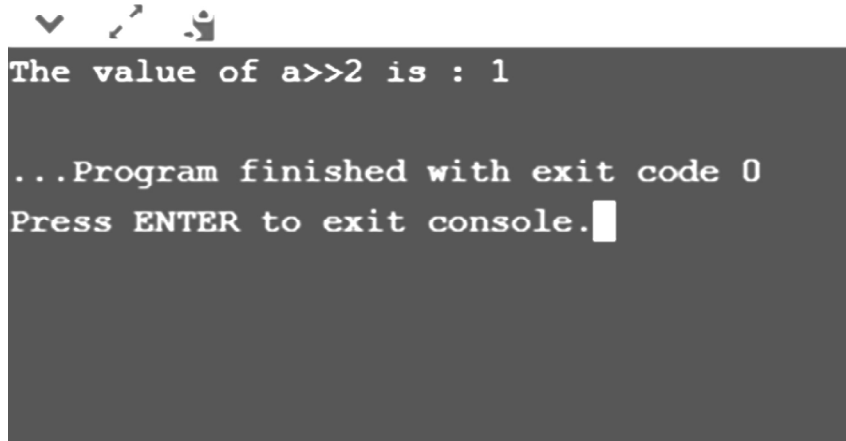
```
int main()
```

```
{
```

```
int a=7; // variable initialization
```

```
printf("The value of a>>2 is : %d ", a>>2);
```

```
return 0;  
}  
Output
```



```
The value of a>>2 is : 1  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Q3. List and explain all the relational operators in C.

Ans :

Relational Operators are the operators used to create a relationship and compare the values of two operands.

For example, there are two numbers, 5 and 15, and we can get the greatest number using the greater than operator (>) that returns 15 as the greatest or larger number to the 5.

Types of Relational Operators

Following are the various types of relational operators in C.

(i) Equal To Operator (==)

It is used to compare both operands and returns 1 if both are equal or the same, and 0 represents the operands that are not equal.

Syntax

1. Opr1 == Opr2

Let's create a program to use the double equal to operator (==) for comparing the operands value in C.

```
#include <stdio.h>  
#include <math.h>  
  
int main ()  
{  
    int a = 5;  
    int b = 10;
```

```
// Use Equal To Operator
printf ("a == b : %d", (a == b));
if (a == b)
printf ("\n %d is equal to %d", a, b);
else
printf("\n %d is not equal to %d",a,b);
int x = 5;
int y = 5;
// Use Equal To Operator
printf (" \n x == y : %d", (x == y));
if (a == b)
printf (" \n %d is equal to %d", x, y);
else
printf ("\n %d is not equal to %d", x, y);
return 0;
}
```

Output

a == b : 0

5 is not equal to 10

x == y : 1

5 is not equal to 5

(ii) Not Equal To Operator (!=)

The Not Equal To Operator is the opposite of the Equal To Operator and is represented as the (!=) operator. The Not Equal To Operator compares two operands and returns 1 if both operands are not the same; otherwise, it returns 0.

Syntax:

Opr1 != Opr2;

Let's create a simple program to use the Not equal to (!=) operator for comparing the values of variable in C.

```
#include <stdio.h>
#include <math.h>
int main ()
{
    int a = 5;
    int b = 10;
```

```
// Use Not Equal To (!=) Operator
printf (" a != b : %d", (a != b));
if (a != b)
    printf ("\n %d is equal to %d", a, b);
else
    printf (" \n %d is not equal to %d", a, b);
int x = 5;
int y = 5;
// Use Not Equal To (!=) Operator
printf (" \n x != y : %d", (x != y));
if (a != b)
    printf (" \n %d is equal to %d", x, y);
else
    printf ("\n %d is not equal to %d", x, y);
return 0;
}
```

Output

a != b : 1

5 is equal to 10

x != y : 0

5 is equal to 5

(iii) Less than Operator (<)

It is used to check whether the value of the left operand is less than the right operand, and if the statement is true, the operator is known as the Less than Operator.

Syntax:

A < B;

Here the operand A is less than operand B.

Let's create a program to use the less-than operator (<) to compare the operand value in C.

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

```
int main ()
```

```
{
```

```
    int num1, num2;
```

```
    printf (" Enter the value of num1: ");
```

```
    scanf (" %d", &num1);
```

```
printf (" \n Enter the value of num2: ");
scanf (" %d", &num2);
// use less than operator (<)
if (num1 < num2)
{
    printf (" \n The value of num1 is less than num2.");
}
else
{
    printf (" \n The value of num2 is less than num1.");
}
return 0;
}
```

Output:

Enter the value of num1: 45

Enter the value of num2: 89

The value of num1 is less than num2.

(iv) Greater than Operator (>)

The operator checks the value of the left operand is greater than the right operand, and if the statement is true, the operator is said to be the Greater Than Operator.

Syntax:

A > B;

Here, operand A is greater than operand B.

Let's create a program to use the greater than operator (>) to compare the operand value in C.

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

```
int main ()
```

```
{
```

```
    int num1, num2;
```

```
    printf (" Enter the value of num1: ");
```

```
    scanf (" %d", &num1);
```

```
    printf (" \n Enter the value of num2: ");
```

```
    scanf (" %d", &num2);
```

```
    // use greater than operator (>)
```

```
    if (num1 > num2)
```



```
{
    printf (" \n The value of num1 is greater than num2.");
}
else
{
    printf (" \n The value of num2 is greater than num1.");
}
return 0;
}
```

Output

Enter the value of num1: 20

Enter the value of num2: 30

The value of num2 is greater than num1.

(v) Less than Equal To Operator (<=)

The operator checks whether the value of the left operand is less than or equal to the right operand, and if the statement is true, the operator is said to be the Less than Equal To Operator.

Syntax:

A <= B;

Here, operand A is less than or equal to operand B. So, it is a Less than Equal to Operator.

Let's create a program to use the Less than Equal to operator (<=) to compare the operand value in C.

Program5.c

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

```
int main ()
```

```
{
```

```
    int num1, num2;
```

```
    printf (" Enter the value of num1: ");
```

```
    scanf (" %d", &num1);
```

```
    printf (" \n Enter the value of num2: ");
```

```
    scanf (" %d", &num2);
```

```
    // use less than equal to operator (<=)
```

```
    if (num1 < num2)
```

```
    {
```

```
        printf (" \n The value of num1 is less than num2.");
```

```
    }
```

```
    else if (num1 <= num2 )
    {
        printf (" \n The value of num1 is equal to num2.");
    }
    else
    {
        printf (" \n The value of num2 is less than num1.");
    }
    return 0;
}
```

Output

Enter the value of num1: 45

Enter the value of num2: 45

The value of num1 is equal to num2.

(vi) Greater than Equal To Operator (>=)

The operator checks whether the left operand's value is greater than or equal to the right operand. If the statement is true, the operator is said to be the Greater than Equal to Operator.

Syntax:

A >= B;

Here, operand A is greater than equal to the right operand B. So, it is the Greater than Equal To operator.

Let's create a program to use the Greater than Equal To operator to compare the operand value in C.

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

```
int main ()
```

```
{
```

```
    int num1, num2;
```

```
    printf (" Enter the value of num1: ");
```

```
    scanf ("%d", &num1);
```

```
    printf (" \n Enter the value of num2: ");
```

```
    scanf ("%d", &num2);
```

```
    // use greater than equal to operator (>=)
```

```
    if (num1 > num2)
```

```
{
```

```

printf (" \n The value of num1 is greater than num2.");
}
else if (num1 >= num2 ) // greater than operator (>=)
{
    printf (" \n The value of num1 is equal to num2.");
}
else
{
    printf (" \n The value of num2 is greater than num1.");
}
return 0;
}

```

Output

Enter the value of num1: 28

Enter the value of num2: 79

The value of num2 is greater than num1.

Q4. Explain in detail Logical operators in C.

Ans :

(Imp.)

These operators are used to perform logical operations on the given expressions.

- There are 3 logical operators in C language. They are, logical AND (&&), logical OR (||) and logical NOT (!).

Operators	Example/Description
&& (logical AND)	(x>5)&&(y<5) It returns true when both conditions are true
(logical OR)	(x>=10) (y>=10) It returns true when at-least one of the condition is true
! (logical NOT)	!((x>5)&&(y<5)) It reverses the state of the operand " $((x>5) \&\& (y<5))$ " If " $((x>5) \&\& (y<5))$ " is true, logical NOT operator makes it false

Example:

```

#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. " \n  
    "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

2.2 IF, IF-ELSE, SWITCH STATEMENTS AND EXAMPLES

Q5. What is the use of if-else statements in C ? List out various types of if statements.

Ans :

The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.

There are the following variants of if statement in C language.

- If statement
- If-else statement
- If else-if ladder
- Nested if

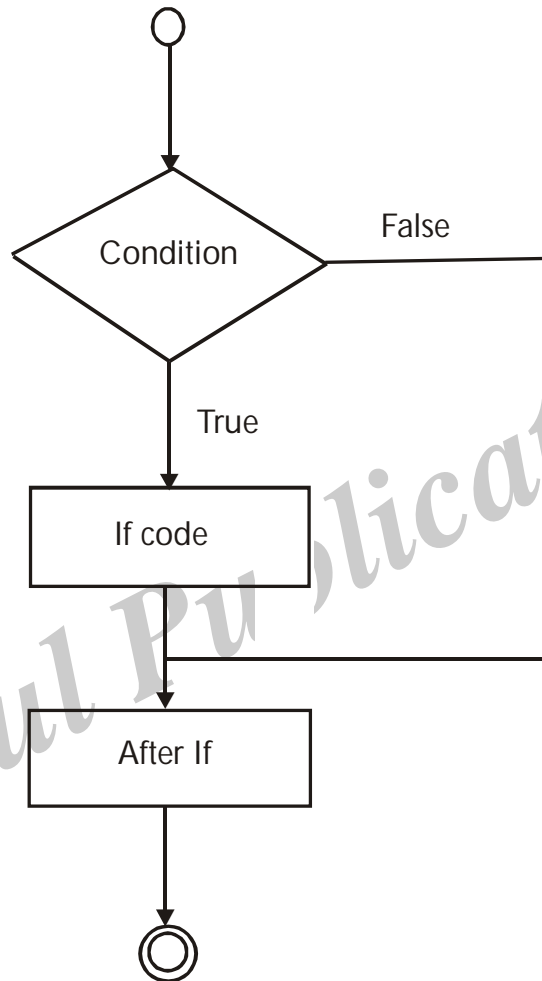
Q6. Explain in detail If statement with an example.

Ans :

The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions. The syntax of the if statement is given below.

```
if(expression)
{
//code to be executed
}
```

Flowchart of if statement in C



Let's see a simple example of C language if statement.

```
#include <stdio.h>

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

```
}  
return 0;  
}
```

Output

Enter a number:4
4 is even number
enter a number:5

Q7. Write a C program to find the largest of given three numbers.

Ans .:

```
#include <stdio.h>  
int main()  
{  
    int a, b, c;  
    printf("Enter three numbers?");  
    scanf("%d %d %d",&a,&b,&c);  
    if(a>b && a>c)  
    {  
        printf("%d is largest",a);  
    }  
    if(b>a && b > c)  
    {  
        printf("%d is largest",b);  
    }  
    if(c>a && c>b)  
    {  
        printf("%d is largest",c);  
    }  
    if(a == b && a == c)  
    {  
        printf("All are equal");  
    }  
}
```

Output

Enter three numbers?
12 23 34
34 is largest

Q8. Explain if – else statement with an example.

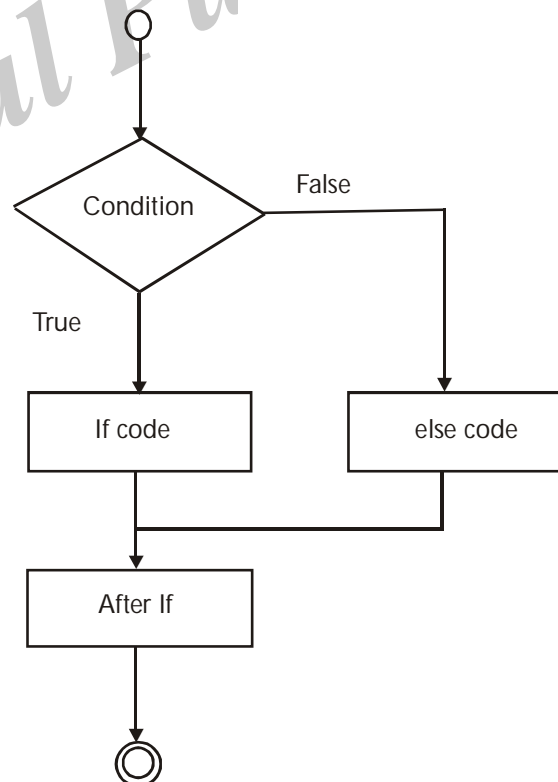
Ans :

If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simultaneously. Using if-else statement is always preferable since it always invokes an otherwise case with every if condition. The syntax of the if-else statement is given below.

```
if(expression)
{
//code to be executed if condition is true
}
else
{
//code to be executed if condition is false
}
```

Flowchart of the if-else statement in C



Let's see the simple example to check whether a number is even or odd using if-else statement in C language.

```
#include <stdio.h>

int main(){
    int number=0;
    printf("enter a number:");
    scanf("%d",&number);
    if(number%2==0){
        printf("%d is even number",number);
    }
    else{
        printf("%d is odd number",number);
    }
    return 0;
}
```

Output

```
enter a number:4
4 is even number
enter a number:5
5 is odd number
```

Q9. Write a C program to check whether a person is eligible to vote or not?

Ans :

```
#include <stdio.h>

int main()
{
    int age;
    printf("Enter your age?");
    scanf("%d",&age);
    if(age >= 18)
    {
        printf("You are eligible to vote...");
    }
    else
    {
```



```
        printf("Sorry ... you can't vote");  
    }  
}
```

Output:

```
Enter your age?18  
You are eligible to vote...  
Enter your age?13  
Sorry ... you can't vote
```

Q10. Explain if-else-if ladder statement in C with an example.*Ans :***(Imp.)**

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

```
if(condition1)  
{  
    //code to be executed if condition1 is true  
}  
else if(condition2)  
{  
    //code to be executed if condition2 is true  
}  
else if(condition3)  
{  
    //code to be executed if condition3 is true  
}  
...  
Else  
{  
    //code to be executed if all the conditions are false  
}
```

Flowchart of else-if ladder statement in C

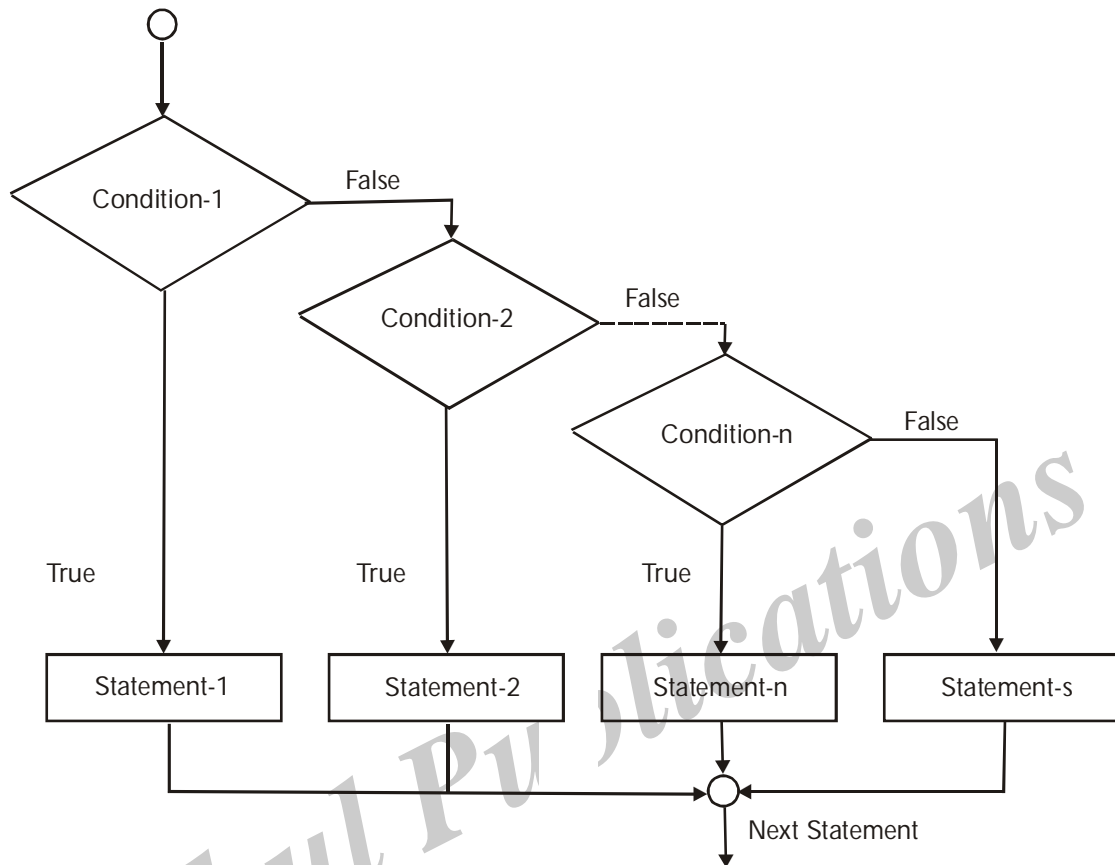


Fig.: else-if ladder

The example of an if-else-if statement in C language is given below.

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

```
int main(){
```

```
int number=0;
```

```
printf("enter a number:");
```

```
scanf("%d",&number);
```

```
if(number==10){
```

```
printf("number is equals to 10");
```

```
}
```

```
else if(number==50){
```

```
printf("number is equal to 50");
```

```
}
```

```
else if(number==100){
```

```
printf("number is equal to 100");
}
else{
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

Q11. Write a C program to calculate the grade of a student according to the marks.

Ans :

```
#include <stdio.h>
int main()
{
    int marks;
    printf("Enter your marks?");
    scanf("%d",&marks);
    if(marks > 85 && marks <= 100)
    {
        printf("Congrats ! you scored grade A ...");
    }
    else if (marks > 60 && marks <= 85)
    {
        printf("You scored grade B + ...");
    }
    else if (marks > 40 && marks <= 60)
    {
        printf("You scored grade B ...");
    }
    else if (marks > 30 && marks <= 40)
    {
```

```
        printf("You scored grade C ...");
    }
    else
    {
        printf("Sorry you are fail ...");
    }
}
```

Output

Enter your marks?10

Sorry you are fail ...

Enter your marks?40

You scored grade C ...

Enter your marks?90

Congrats ! you scored grade A ...

Q12. Explain in detail switch statement in C.

Ans :

(Imp.)

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

The syntax of switch statement in c language is given below:

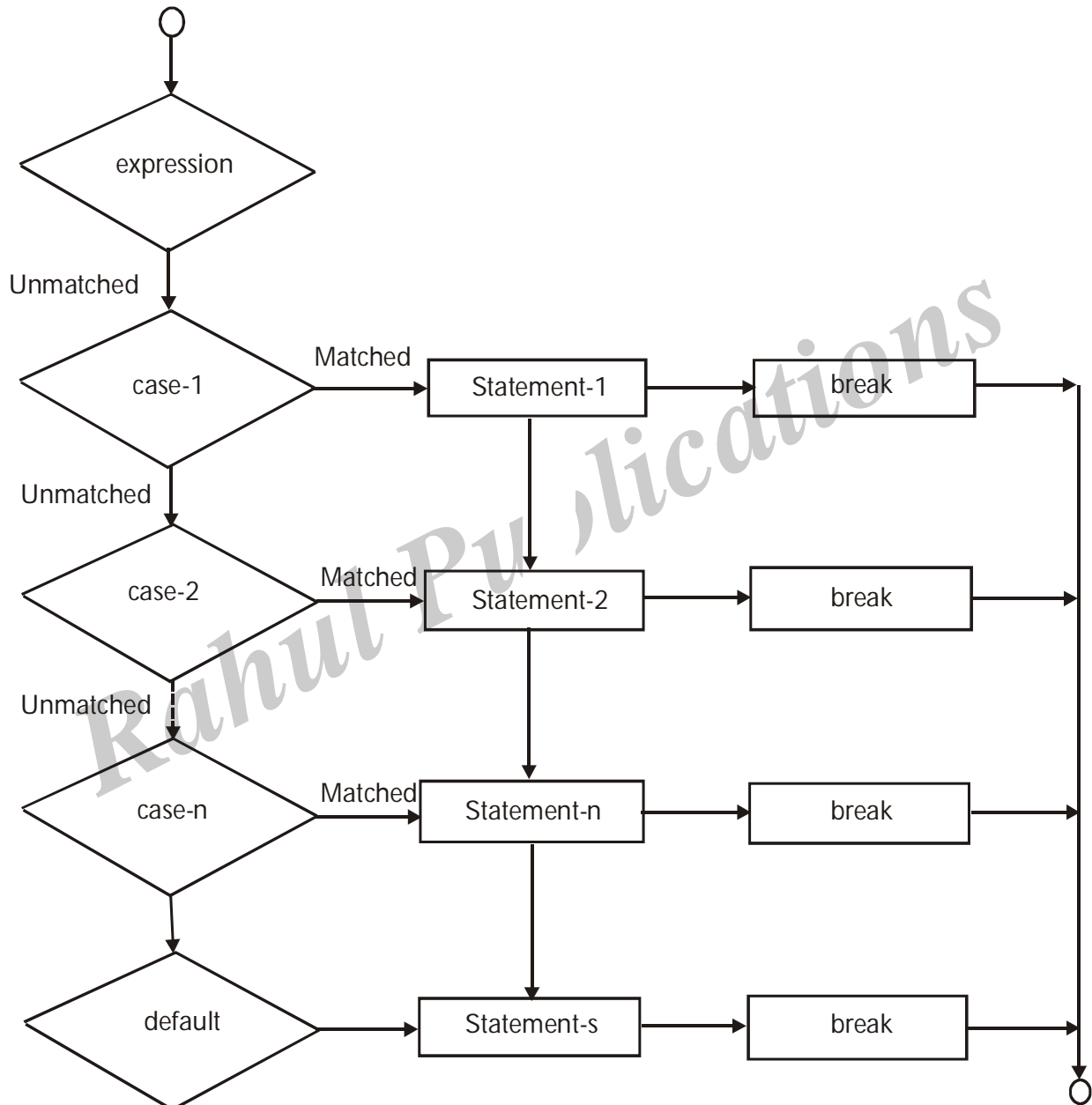
```
switch(expression)
{
    case value1:
        //code to be executed;
        break; //optional
    case value2:
        //code to be executed;
        break; //optional
    .....
    default:
        code to be executed if all cases are not matched;
}
```

Rules for switch statement in C language

- 1) The switch expression must be of an integer or character type.
- 2) The case value must be an integer or character constant.

- 3) The case value can be used only inside the switch statement.
- 4) The break statement in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as fall through the state of C switch statement.

Flowchart of switch statement in C



Functioning of switch case statement

First, the integer expression specified in the switch statement is evaluated. This value is then matched one by one with the constant values given in the different cases. If a match is found, then all the statements specified in that case are executed along with the all the cases present after that case including the default

statement. No two cases can have similar values. If the matched case contains a break statement, then all the cases present after that will be skipped, and the control comes out of the switch. Otherwise, all the cases following the matched case will be executed.

Let's see a simple example of c language switch statement.

```
#include<stdio.h>

int main()
{
    int number=0;
    printf("enter a number:");
    scanf("%d",&number);
    switch(number){
    case 10:
        printf("number is equals to 10");
        break;
    case 50:
        printf("number is equal to 50");
        break;
    case 100:
        printf("number is equal to 100");
        break;
    default:
        printf("number is not equal to 10, 50 or 100");
    }
    return 0;
}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

Q13. What are the differences between If-else and switch statements.

Ans :

(Imp.)

	If-else	switch
Definition	Depending on the condition in the 'if' statement, 'if' and 'else' blocks are executed.	The user will decide which statement is to be executed.
Expression	It contains either logical or equality expression.	It contains a single expression which can be either a character or integer variable.
Evaluation	It evaluates all types of data, such as integer, floating-point, character or Boolean.	It evaluates either an integer, or character.
Sequence of execution	First, the condition is checked. If the condition is true then 'if' block is executed otherwise 'else' block	It executes one case after another till the break keyword is not found, or the default statement is executed.
Default execution	If the condition is not true, then by default, else block will be executed.	If the value does not match with any case, then by default, default statement is executed.
Editing	Editing is not easy in the 'if-else' statement.	Cases in a switch statement are easy to maintain and modify. Therefore, we can say that the removal or editing of any case will not interrupt the execution of other cases.
Speed	If there are multiple choices implemented through 'if-else', then the speed of the execution will be slow.	If we have multiple choices then the switch statement is the best option as the speed of the execution will be much higher than 'if-else'.

2.3 LOOPING STATEMENTS: WHILE, DO-WHILE AND EXAMPLES

Q14. What is mean by loop? Discuss various types of loops supported by C Language.

Ans :

(Imp.)

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times.

For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1. It provides code reusability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

There are three types of loops in C language that is given below:

1. do while
2. while
3. for

(i) do-while loop in C

The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

The syntax of do-while loop in c language is given below:

```
Do
{
//code to be executed
}while(condition);
```

(ii) While loop in C

The while loop in c is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.

The syntax of while loop in c language is given below:

```
while(condition)
{
//code to be executed
}
```

(iii) for loop in C

The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.

The syntax of for loop in c language is given below:

```
for(initialization;condition;incr/decr)
{
//code to be executed
}
```

Q15. Explain In detail Do-While loop in C.

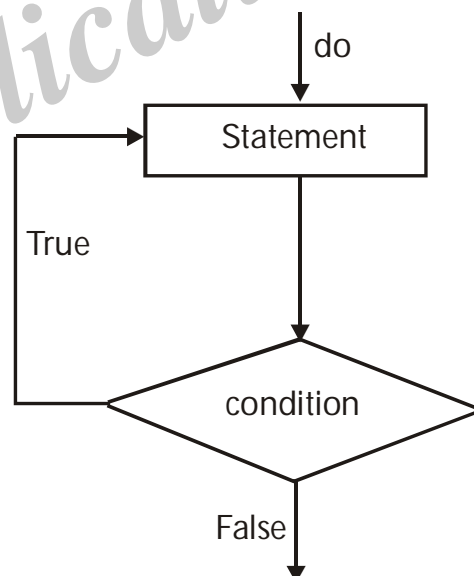
Ans :

The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

do while loop syntax

```
do
{
//code to be executed
}while(condition);
```

Flowchart of do while loop



Example:

```
#include <stdio.h>

int main()
{
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
```



```

do
{
printf("%d \n", (number*i));
i++;
}while(i <= 10);
return 0;
}

```

Output

Enter a number: 5

5
10
15
20
25
30
35
40
45
50

Q16. How can we use while loop in c programming?

Ans : (Imp.)

While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

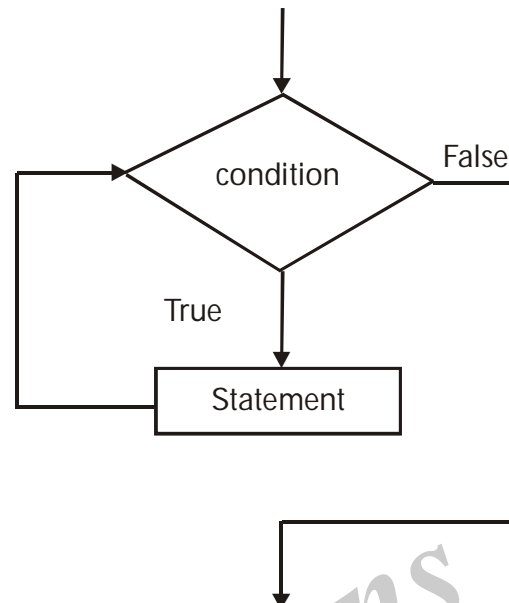
Syntax

```

while(condition)
{
//code to be executed
}

```

Flowchart

**Properties of while loop**

- A conditional expression is used to check the condition. The statements defined inside the while loop will repeatedly execute until the given condition fails.
- The condition will be true if it returns 0. The condition will be false if it returns any non-zero number.
- In while loop, the condition expression is compulsory.
- Running a while loop without a body is possible.
- We can have more than one conditional expression in while loop.
- If the loop body contains only one statement, then the braces are optional.

Example:

```

#include <stdio.h>

int main(){
int i=1;
while(i <= 10){
printf("%d \n", i);
i++;
}
}

```

```

}
return 0;
}

```

Output

```

1
2
3
4
5
6
7
8
9
10

```

Q17. Write a c programming to print a table for the given number using while loop.

Ans :

```

#include<stdio.h>
int main()
{
int i=1,number=0,b=9;
printf("Enter a number: ");
scanf("%d",&number);
while(i <= 10)
{
printf("%d \n",(number*i));
i++;
}
return 0;
}

```

Output

```

Enter a number: 50
50
100

```

```

150
200
250
300
350
400
450
500

```

Q18. How can we use for loop in C Programming?

Ans :

The for loop in C language is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

Syntax

```

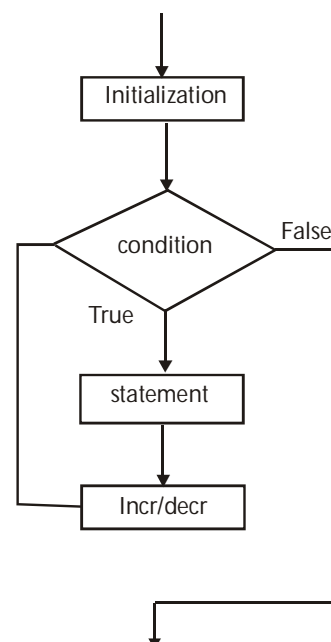
for (Expression 1; Expression 2;
Expression 3)

```

```

{
//code to be executed
}

```

Flow chart

Properties of Expression 1

- The expression represents the initialization of the loop variable.
- We can initialize more than one variable in Expression 1.
- Expression 1 is optional.
- In C, we can not declare the variables in Expression 1. However, It can be an exception in some compilers.

Properties of Expression 2

- Expression 2 is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.
- Expression 2 can have more than one condition. However, the loop will iterate until the last condition becomes false. Other conditions will be treated as statements.
- Expression 2 is optional.
- Expression 2 can perform the task of expression 1 and expression 3. That is, we can initialize the variable as well as update the loop variable in expression 2 itself.
- We can pass zero or non-zero value in expression 2. However, in C, any non-zero value is true, and zero is false by default.

Properties of Expression 3

- Expression 3 is used to update the loop variable.
- We can update more than one variable at the same time.
- Expression 3 is optional.

Example:

```
#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<=4;i++)
    {
        printf("%d ",i);
    }
}
```

Output

0 1 2 3 4

Q19. What is mean by nested loops?

Ans : (Imp.)

C supports nesting of loops in C. Nesting of loops is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.

Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define 'while' loop inside a 'for' loop.

Syntax of Nested loop

```
Outer_loop
{
    Inner_loop
    {
        // inner loop statements.
    }
    // outer loop statements.
}
```

Outer_loop and Inner_loop are the valid loops that can be a 'for' loop, 'while' loop or 'do-while' loop.

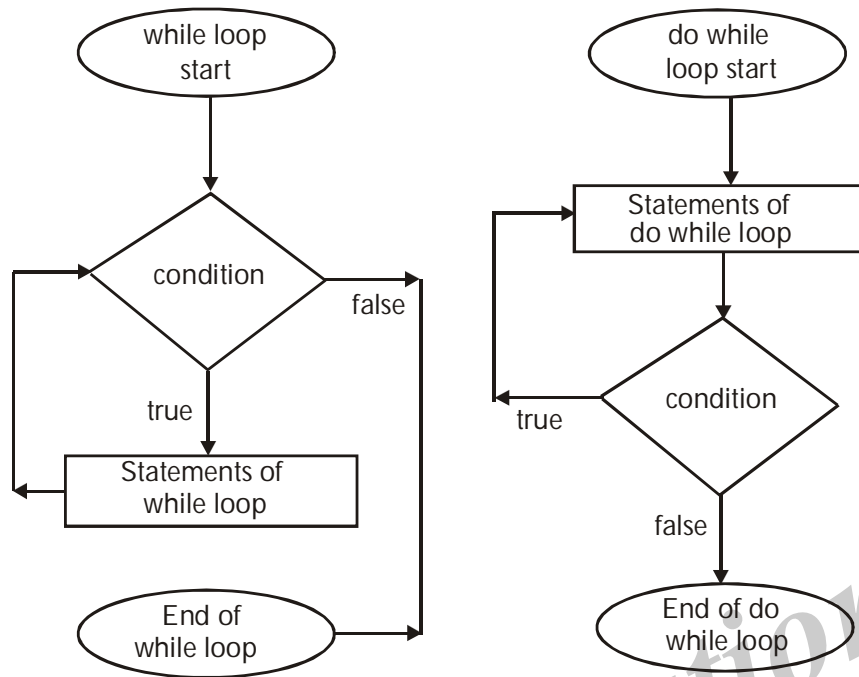
Example:

```
#include <stdio.h>
int main()
{
    int n;// variable declaration
    printf("Enter the value of n :");
    // Displaying the n tables.
    for(int i=1;i<=n;i++) // outer loop
    {
        for(int j=1;j<=10;j++) // inner loop
        {
            printf("%d\t",i*j); // printing the value.
        }
        printf("\n");
    }
}
```

Q20. What are the differences between while loop and do-while loop.

Ans :

S.No.	while loop	do-while loop
1.	While the loop is an entry control loop because firstly, the condition is checked, then the loop's body is executed.	The do-while loop is an exit control loop because in this, first of all, the body of the loop is executed then the condition is checked true or false.
2.	The statement of while loop may not be executed at all.	The statement of the do-while loop must be executed at least once.
3.	The while loop terminates when the condition becomes false.	As long as the condition is true, the compiler keeps executing the loop in the do-while loop.
4.	In a while loop, the test condition variable must be initialized first to check the test condition in the loop.	In a do-while loop, the variable of test condition initialized in the loop also.
5.	In a while loop, at the end of the condition, there is no semicolon. Syntax: while (condition)	In this, at the end of the condition, there is a semicolon. Syntax: while (condition);
6.	While loop is not used for creating menu-driven programs.	It is mostly used for creating menu-driven programs because at least one time; the loop is executed whether the condition is true or false.
7.	In a while loop, the number of executions depends on the condition defined in the while block.	In a do-while loop, irrespective of the condition mentioned, a minimum of 1 execution occurs.
8.	Syntax of while loop: while (condition) { Block of statements; } Statement-x;	Syntax of do-while loop: do { statements; } while (condition); Statement-x;
9.	Program of while loop: Program of while loop: #include #include Void main() { inti; clrscr(); i = 1; while(i <= 10) { printf("hello"); i = i + 1; } getch(); }	Program of do-while loop: #include #include Void main() { inti; clrscr(); i = 1; do { printf("hello"); i = i + 1; } while(i <= 10); getch(); }
10.	Flowchart of while loop:	Flowchart of do-while loop:



2.4 CONTINUE, BREAK, GO TO STATEMENTS

Q21. What is the use of break statement?

Ans :

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

Syntax:

break;

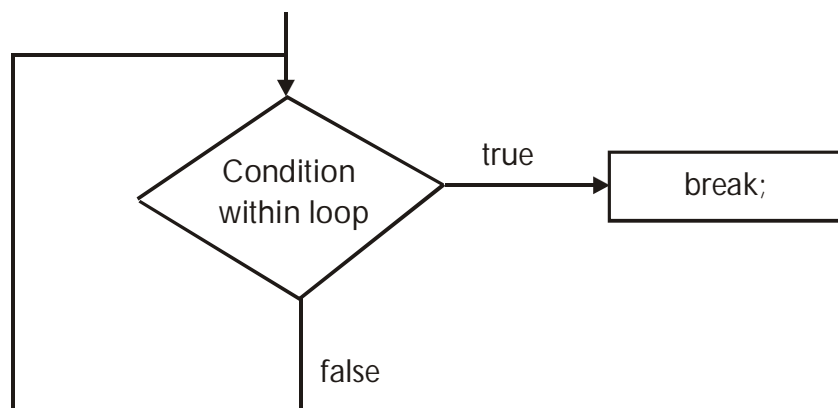


Fig.: Flowchart of break statement

Example

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
    int i;
    for(i = 0; i<10; i++)
    {
        printf("%d ",i);
        if(i == 5)
            break;
    }
    printf("came outside of loop i =%d",i);
}
```

Output

0 1 2 3 4 5 came outside of loop i = 5

Q22. What is the use of continue statement in C?*Ans :*

The continue statement in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax:

```
//loop statements
continue;
// some lines of the code which is to be
skipped
```

Example:

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

```
void main ()
{
    int i = 0;
    while(i!=10)
    {
        printf("%d", i);
        continue;
        i++;
    }
}
```

Output

infinite loop

Q23. Explain in detail go-to statement with an example.*Ans :*

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statement can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement. However, using goto is avoided these days since it makes the program less readable and complicated.

Syntax:

```
label:
//some part of the code;
goto label;
```

Example:

```
#include <stdio.h>
int main()
{
    int num,i=1;
```

```
printf("Enter the number whose table you
want to print?");
scanf("%d",&num);
table:
printf("%d x %d=%d\n",num,i,num*i);
i++;
if(i<=10)
goto table;
}
```

Output:

How to find Nth Highest Salary in SQL

Enter the number whose table you want to print?10

10 x 1 = 10

10 x 2 = 20

10 x 3 = 30

10 x 4 = 40

10 x 5 = 50

10 x 6 = 60

10 x 7 = 70

10 x 8 = 80

10 x 9 = 90

10 x 10 = 100

2.5 FUNCTIONS

Q24. What is Function ? what are the advantages of functions ?

Ans :

(Imp.)

In c, we can divide a large program into the basic building blocks known as function. 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. In other words, we can say that the collection of functions creates a program. The

function is also known as procedure or subroutine in other programming languages.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always a overhead in a C program.

Q25. What are the major aspects of functions?

Ans :

There are three aspects of a C function.

- **Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
- **Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.
- **Function definition** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

S.No.	C function aspects	Syntax
1.	Function declaration	return_type function_name (argument list);
2.	Function call	function_name (argument_list)
3.	Function definition	return_type function_name (argument list) {function body;}

The syntax of creating function in c language is given below:

```
return_type function_name(data_type parameter...)
{
    //code to be executed
}
```

Q26. How can we define a function in C ? Explain with an example ?

Ans :

The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list ) {
    body of the function
}
```

A function definition in C programming consists of a function header and a function body. Here are all the parts of a function.

- **Return Type:** A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

Example:

```
/* function returning the max between two numbers */
int max(int num1, int num2) {
    /* local variable declaration */
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```


Function Declarations

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts -

return_type function_name(parameter list);

For the above defined function max(), the function declaration is as follows-

int max(int num1, int num2);

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration.

int max(int, int);

Function declaration is required when you define a function in one source file and you call that function in another file.

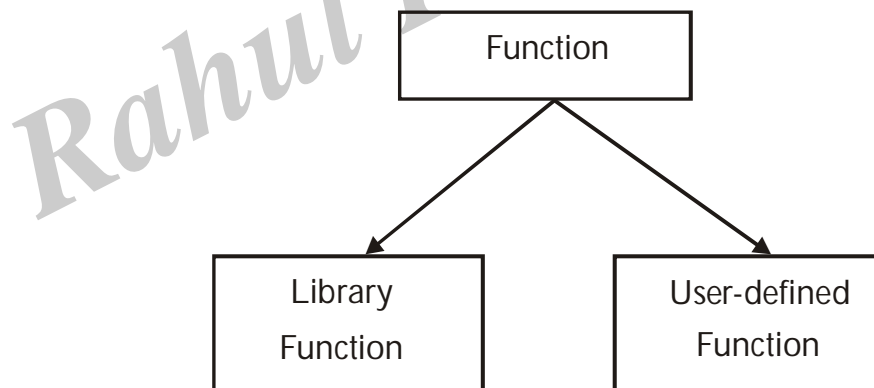
2.5.1 User defines functions

Q27. List out various types of functions supported by C ?

Ans :

There are two types of functions in C programming:

1. **Library Functions:** are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
2. **User-defined functions:** are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.



Q28. List out various C Library functions.

Ans :

(Imp.)

Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations. For example, printf is a library function used to print on the console. The library functions are created by the designers of compilers. All C standard library functions are defined inside the different header files saved with the extension .h. We need to include these header files in our program to make use of the library functions defined in such header files. For example, To use the library functions such as printf/scanf we need to include stdio.h in our program which is a header file that contains all the library functions regarding standard input/output.

The list of mostly used header files is given in the following table.

S.No.	Header file	Description
1.	stdio.h	This is a standard input/output header file. It contains all the library functions regarding standard input/output.
2.	conio.h	This is a console input/output header file.
3.	string.h	It contains all string related library functions like gets(), puts(), etc.
4.	stdlib.h	This header file contains all the general library functions like malloc(), calloc(), exit(), etc.
5.	math.h	This header file contains all the math operations related functions like sqrt(), pow(), etc.
6.	time.h	This header file contains all the time-related functions.
7.	ctype.h	This header file contains all character handling functions.
8.	stdarg.h	Variable argument functions are defined in this header file.
9.	signal.h	All the signal handling functions are defined in this header file.
10.	setjmp.h	This file contains all the jump functions.
11.	locale.h	This file contains locale functions.
12.	errno.h	This file contains error handling functions.
13.	assert.h	This file contains diagnostics functions.

Q29. How can we declare user defined functions in C? what are the advantages of user defined functions ?

Ans :

we can also create functions as per our need. Such functions created by the user are known as user-defined functions.

Example:

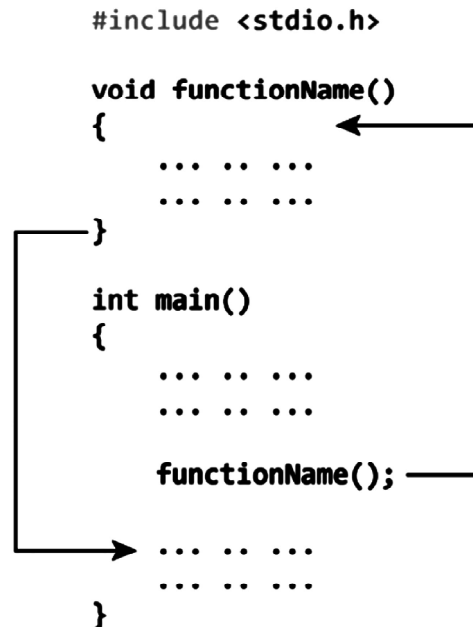
```
#include <stdio.h>
void functionName()
{
    ... ..
    ... ..
}
int main()
{
    ... ..
    ... ..
    functionName();
    ... ..
    ... ..
}
```

The execution of a C program begins from the `main()` function.

When the compiler encounters `functionName();`, control of the program jumps to `void functionName()`

And, the compiler starts executing the codes inside `functionName()`.

The control of the program jumps back to the `main()` function once code inside the function definition is executed.



Advantages of user-defined function

1. The program will be easier to understand, maintain and debug.
2. Reusable codes that can be used in other programs
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Q30. Explain various types of user defined functions with an example program ? (or) what are the aspects function calling?

Ans :

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

1. function without arguments and without return value
2. function without arguments and with return value
3. function with arguments and without return value
4. function with arguments and with return value

Function with no argument and no return value : When a function has no arguments, it does not receive any data from the calling function. Similarly when it does not return a value, the calling function does not receive any data from the called function.

Syntax :

Function declaration : void function();

Function call : function();

Function definition :

```
void function()
{
    statements;
}
```

Example:

```
#include<stdio.h>
void printName();
void main ()
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf("Rakesh");
}
```

Output

Hello Rakesh

Function with arguments but no return value : When a function has arguments, it receive any data from the calling function but it returns no values.

Syntax :

Function declaration : void function (int);

Function call : function(x);

Function definition:

```
void function( int x )
{
    statements;
}
```

Example:

```
#include<stdio.h>
void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum
of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
void sum(int a, int b)
{
    printf("\nThe sum is %d", a+b);
}
```

Output

Going to calculate the sum of two numbers:

Enter two numbers 10

24

The sum is 34

Function with no arguments but returns a value : There could be occasions where we may need to design functions that may not take any arguments but returns a value to the calling function. A example for this is getchar function it has no parameters but it returns an integer an integer type data that represents a character.

Syntax :

Function declaration : int function();

Function call : function();

Function definition :

```
int function()
{
    statements;
    return x;
}
```

Example:

```
#include<stdio.h>

int sum();
void main()
{
    printf("Going to calculate the area of the
    square\n");
    float area = square();
    printf("The area of the square: %f\n",
    area);
}
int square()
{
    float side;
    printf("Enter the length of the side
    in meters: ");
    scanf("%f",&side);
    return side * side;
}
```

Output

Going to calculate the area of the square

Enter the length of the side in meters: 10

The area of the square: 100.000000

Function with arguments and return value

When a function is capable to accept arguments and returns a value then it is said to be functions with arguments and return value.

Syntax :

Function declaration :int function (int);

Function call : function(x);

Function definition:

```
int function( int x )
{
    statements;
    return x;
}
```

Example:

Program to check whether a number is even or odd

```
#include<stdio.h>
int even_odd(int);
void main()
{
    int n,flag=0;
    printf("\nGoing to check whether a
    number is even or odd");
    printf("\nEnter the number: ");
    scanf("%d",&n);
    flag = even_odd(n);
    if(flag == 0)
    {
        printf("\nThe number is odd");
    }
    else
    {
        printf("\nThe number is even");
    }
}
int even_odd(int n)
{
    if(n%2 == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Output

Going to check whether a number is even or odd

Enter the number: 100

The number is even

2.5.2 Inter function communication

Q31. What is inter function communication? Explain in detail.

Ans : (Imp.)

When a function gets executed in the program, the execution control is transferred from calling a function to called function and executes function definition, and finally comes back to the calling function. In this process, both calling and called functions have to communicate with each other to exchange information. The process of exchanging information between calling and called functions is called inter-function communication.

In C, the inter function communication is classified as follows...

- Downward Communication
- Upward Communication
- Bi-directional Communication

Downward Communication

In this type of inter function communication, the data is transferred from calling function to called function but not from called function to calling function. The functions with parameters and without return value are considered under downward communication. In the case of downward communication, the execution control jumps from calling function to called function along with parameters and executes the function definition, and finally comes back to the calling function without any return value. For example consider the following program...

Example Program

```
#include<stdio.h>
#include<conio.h>
void main(){
int num1, num2 ;
void addition(int, int) ; // function declaration
clrscr() ;
num1 = 10 ;
num2 = 20 ;
```

```
printf("\nBefore swap: num1 = %d, num2
= %d", num1, num2) ;
```

```
addition(num1, num2) ; // calling function
```

```
getch() ;
```

```
}
```

```
void addition(int a, int b) // called function
```

```
{
```

```
printf("SUM = %d", a+b) ;
```

```
}
```

Output:

Before swap: num1 = 10, num2 = 20

SUM = 30

Process returned 0(0×0) ececution time:
0.047 s

Press any key to continue

Upward Communication

In this type of inter-function communication, the data is transferred from called function to calling-function but not from calling-function to called-function. The functions without parameters and with return value are considered under upward communication. In the case of upward communication, the execution control jumps from calling-function to called-function without parameters and executes the function definition, and finally comes back to the calling function along with a return value. For example, consider the following program...

Example Program

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

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

```
void main(){
```

```
int result ;
```

```
int addition() ; // function declaration
```

```
clrscr() ;
```

```
result = addition() ; // calling function
```

```
printf("SUM = %d", result) ;
```

```

getch() ;
}

int addition() // called function
{
    int num1, num2 ;

    num1 = 10;
    num2 = 20;

    return (num1 + num2) ;
}

```

Output:

```

SUM = 30

Process returned 0 (0x0) execution time :
0.047 s

Press any key to continue.

```

Bi - Directional Communication

In this type of inter-function communication, the data is transferred from calling-function to called function and also from called function to calling-function. The functions with parameters and with return value are considered under bi-directional communication. In the case of bi-directional communication, the execution control jumps from calling-function to called function along with parameters and executes the function definition and finally comes back to the calling function along with a return value. For example, consider the following program...

Example Program

```

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

void main(){
    int num1, num2, result ;

    int addition(int, int) ; // function declaration

    clrscr() ;

```

```

    num1 = 10 ;
    num2 = 20 ;

    result = addition(num1, num2) ; // calling
    function

    printf("SUM = %d", result) ;

    getch() ;
}

int addition(int a, int b) // called function
{
    return (a + b) ;
}

```

Output:

```

SUM = 30

Process returned 0 (0x0) execution time :
0.047 s

Press any key to continue.

```

2.5.3 Standard Functions**Q32. List out various standard function of C.***Ans :***(Imp.)**

The standard functions are built-in functions. In C programming language, the standard functions are declared in header files and defined in .dll files. In simple words, the standard functions can be defined as "the ready made functions defined by the system to make coding more easy". The standard functions are also called as library functions or pre-defined functions.

In C when we use standard functions, we must include the respective header file using #include statement. For example, the function printf() is defined in header file stdio.h (Standard Input Output header file). When we use printf() in our program, we must include stdio.h header file using # include <stdio.h> statement.

C Programming Language provides the following header files with standard functions.

Header File	Purpose	Example Functions
stdio.h	Provides functions to perform standard I/O operations	printf(), scanf()
conio.h	Provides functions to perform console I/O operations	clrscr(), getch()
math.h	Provides functions to perform mathematical operations	sqrt(), pow()
string.h	Provides functions to handle string data values	strlen(), strcpy()
stdlib.h	Provides functions to perform general functions/td>	calloc(), malloc()
time.h	Provides functions to perform operations on time and date	time(), localtime()
ctype.h	Provides functions to perform - testing and mapping of character data values	isalpha(), islower()
setjmp.h	Provides functions that are used in function calls	setjump(), longjump()
signal.h	Provides functions to handle signals during program execution	signal(), raise()
assert.h	Provides Macro that is used to verify assumptions made by the program	assert()
locale.h	Defines the location specific settings such as date formats and currency symbols	setlocale()
stdarg.h	Used to get the arguments in a function if the arguments are not specified by the function	va_start(), va_end(), va_arg()
errno.h	Provides macros to handle the system calls	Error, errno
graphics.h	Provides functions to draw graphics.	circle(), rectangle()
float.h	Provides constants related to floating point data values	
stddef.h	Defines various variable types	
limits.h	Defines the maximum and minimum values of various variable types like char, int and long	

2.5.4 Parameters passing in C

Q33. What is mean by parameters ? How can we pass parameters in functions?

Ans :

When a function gets executed in the program, the execution control is transferred from calling-function to called function and executes function definition, and finally comes back to the calling function. When the execution control is transferred from calling-function to called-function it may carry one or number of data values. These data values are called as parameters.

In C, there are two types of parameters and they are as follows...

- Actual Parameters
- Formal Parameters

The actual parameters are the parameters that are specified in calling function. The formal parameters are the parameters that are declared at called function. When a function gets executed, the copy of actual parameter values are copied into formal parameters.

In C Programming Language, there are two methods to pass parameters from calling function to called function and they are as follows -

- Call by Value
- Call by Reference

Call by Value

In call by value parameter passing method, the copy of actual parameter values are copied to formal parameters and these formal parameters are used in called function. The changes made on the formal parameters does not effect the values of actual parameters. That means, after the execution control comes back to the calling function, the actual parameter values remains same. For example consider the following program...

Example Program

```
#include<stdio.h>
#include<conio.h>
void main(){
int num1, num2 ;
void swap(int,int) ; // function declaration
clrscr() ;
    num1 = 10 ;
    num2 = 20 ;
printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
swap(num1, num2) ; // calling function
printf("\nAfter swap: num1 = %d\nnum2 = %d", num1, num2);
getch() ;
}
void swap(int a, int b) // called function
{
int temp ;
    temp = a ;
    a = b ;
    b = temp ;
}
```

Output:

Before swap: num1 = 10, num2 = 20

After swap: num1 = 10

num2 = 20

Process returned 0 (0x0) execution time : 0.047 s

Press any key to continue.

In the above example program, the variables num1 and num2 are called actual parameters and the variables a and b are called formal parameters. The value of num1 is copied into a and the value of num2 is copied into b. The changes made on variables a and b does not effect the values of num1 and num2.

Call by Reference

In Call by Reference parameter passing method, the memory location address of the actual parameters is copied to formal parameters. This address is used to access the memory locations of the actual parameters in called function. In this method of parameter passing, the formal parameters must be pointer variables.

That means in call by reference parameter passing method, the address of the actual parameters is passed to the called function and is recieved by the formal parameters (pointers). Whenever we use these formal parameters in called function, they directly access the memory locations of actual parameters. So the changes made on the formal parameters effects the values of actual parameters. For example consider the following program...

Example Program

```
#include<stdio.h>
#include<conio.h>
void main(){
int num1, num2 ;
void swap(int *,int *) ; // function declaration
clrscr() ;
    num1 = 10 ;
    num2 = 20 ;
```

```

printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
swap(&num1, &num2) ; // calling function
printf("\nAfter swap: num1 = %d, num2 = %d", num1, num2);
getch() ;
}

void swap(int *a, int *b) // called function
{
    int temp ;
    temp = *a ;
    *a = *b ;
    *b = temp ;
}

```

Output:

Before swap: num1 = 10, num2 = 20

After swap: num1 = 20, num2 = 10

Process returned 0 (0x0) execution time : 0.047 s

Press any key to continue.

Q34. What are the differences between call by value and call by reference.

Ans :

S.No.	Call by value	Call by reference
1.	A copy of the value is passed into the function	An address of value is passed into the function
2.	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3.	Actual and formal arguments are created at the different memory location.	Actual and formal arguments are created at the same memory location

2.5.5 Recursion–Recursive functions**Q35. What is recursion?**

Ans :

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.

Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

Generally, iterative solutions are more efficient than recursion since function call is always overhead. Any problem that can be solved recursively, can also be solved iteratively.

In the following example, recursion is used to calculate the factorial of a number.

```
#include <stdio.h>

int fact (int);

int main()
{
    int n,f;

    printf("Enter the number whose
    factorial you want to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("factorial = %d",f);
}

int fact(int n)
{
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}
```

Output

Enter the number whose factorial you want to calculate?5

factorial = 120

We can understand the above program of the recursive method call by the figure given below:

```
return 5 * factorial(4) = 120
└─ return 4 * factorial(3) = 24
    └─ return 3 * factorial(2) = 6
        └─ return 2 * factorial(1) = 2
            └─ return 1 * factorial(0) = 1
```

1 * 2 * 3 * 4 * 5 = 120

Fig.: Recursion

Q36. What is recursion function?

Ans :

A recursive function performs the tasks by dividing it into the subtasks. There is a termination condition defined in the function which is satisfied by some specific subtask. After this, the recursion stops and the final result is returned from the function.

The case at which the function doesn't recur is called the base case whereas the instances where the function keeps calling itself to perform a subtask, is called the recursive case. All the recursive functions can be written using this format.

Pseudocode for writing any recursive function is given below.

```
if (test_for_base)
{
    return some_value;
}
else if (test_for_another_base)
{
    return some_another_value;
}
else
{
    // Statements;
    recursive call;
}
```

Let's see an example to find the nth term of the Fibonacci series.

```
#include<stdio.h>

int fibonacci(int);

void main ()
{
    int n,f;
    printf("Enter the value of n?");
    scanf("%d",&n);
    f = fibonacci(n);
    printf("%d",f);
}

int fibonacci (int n)
{
    if (n==0)
    {
        return 0;
    }
    else if (n == 1)
    {
        return 1;
    }
    else
    {
        return fibonacci(n-1)+fibonacci
        (n-2);
    }
}
```

Output

Enter the value of n?12

144

2.6 STORAGE CLASSES - AUTO, REGISTER, STATIC, EXTERN, SCOPE RULES

Q37. List out various types of storage classes in C.

Ans :

(Imp.)

In C programming language, storage classes are used to define things like storage location (whether RAM or REGISTER), scope, lifetime and the default value of a variable.

In the C programming language, the memory of variables is allocated either in computer memory (RAM) or CPU Registers. The allocation of memory depends on storage classes.

In C programming language, there are FOUR storage classes and they are as follows

1. auto storage class
2. extern storage class
3. static storage class
4. register storage class

1. auto storage class

The default storage class of all local variables (variables declared inside block or function) is auto storage class. Variable of auto storage class has the following properties...

Property	Description
Keyword	auto
Storage	Computer Memory (RAM)
Default Value	Garbage Value
Scope	Local to the block in which the variable is defined
Life time	Till the control remains within the block in which variable is defined

Example:

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

int main()
{
    inti;
```

```

    auto char c;
    float f;

    printf("i = %d\tc = %c\tf = %f",i,c,f);

    return 0;
}

```

2. External storage class

The default storage class of all global variables (variables declared outside function) is external storage class. Variable of external storage class has the following properties...

Property	Description
Property	Description
Keyword	extern
Storage	Computer Memory (RAM)
Default Value	Zero
Scope	Global to the program (i.e., Throughout the program)
Life time	As long as the program's execution does not comes to end

Example

```

#include <stdio.h>
#include <conio.h>
inti;    //By default it is extern variable
int main(){
    printf("%d",i);
    return 0;
}

```

3. Static storage class

The static storage class is used to create variables that hold value beyond its scope until the end of the program. The static variable allows to initialize only once and can be modified any number of times. Variable of static storage class has the following properties...

Property	Description
Keyword	static
Storage	Computer Memory (RAM)
Default Value	Zero
Scope	Local to the block in which the variable is defined
Life time	The value of the persists between different function calls (i.e., Initialization is done only once)

Example

```
#include <stdio.h>

#include <conio.h>

static int a;

int main(){

    printf("%d",a);

    return 0;

}
```

4. Register storage class

The register storage class is used to specify the memory of the variable that has to be allocated in CPU Registers. The memory of the register variable is allocated in CPU Register but not in Computer memory (RAM). The register variables enable faster accessibility compared to other storage class variables. As the number of registers inside the CPU is very less we can use very less number of register variables. Variable of register storage class has the following properties.

Property	Description
Keyword	register
Storage	CPU Register
Default Value	Garbage Value
Scope	Local to the block in which the variable is defined
Life time	Till the control remains within the block in which variable is defined

Example

```
#include <stdio.h>

#include <conio.h>

int main(){

    register int a,b;

    scanf("%d%d",&a,&b);

    printf("%d %d",a,b);

}
```

Q38. Discuss all the properties of storage classes in C.*Ans :*

The following table provides detailed properties of all storage classes...

Storage Class	Keyword	Memory Location	Default Value	Scope	Life Time
Automatic	auto	Computer Memory (RAM)	Garbage Value	Local to the block in which the variable has defined	Till the control remains within the block in which variable is defined
External	extern	Computer Memory (RAM)	Zero	Global to the program (i.e., Throughout the program)	As long as the program's execution does not come to end
Static	static	Computer Memory (RAM)	Zero	Local to the block in which the variable has defined	The value of the persists between different function calls (i.e., Initialization is done only once)
Register	register	CPU Register	Garbage Value	Local to the block in which the variable has defined	Till the control remains within the block in which variable is defined

Q39. What are the use of type Qualifiers in C.*Ans :*

In C programming language, type qualifiers are the keywords used to modify the properties of variables. Using type qualifiers, we can change the properties of variables. The C programming language provides two type qualifiers and they are as follows.

- const
- volatile

const type qualifier in C

The const type qualifier is used to create constant variables. When a variable is created with const keyword, the value of that variable can't be changed once it is defined. That means once a value is assigned to a constant variable, that value is fixed and cannot be changed throughout the program.

The keyword const is used at the time of variable declaration. We use the following syntax to create constant variable using const keyword.

When a variable is created with const keyword it becomes a constant variable. The value of the constant variable can't be changed once it is defined. The following program generates error message because we try to change the value of constant variable x.

Example Program

```
#include <stdio.h>

#include <conio.h>

void main(){
```

```
inti = 9 ;  
constint x = 10 ;  
clrscr() ;  
i = 15 ;  
x = 100 ; // creates an error  
printf("i = %d\nx = %d", i, x ) ;  
}
```

volatile type qualifier in C

The `volatile` type qualifier is used to create variables whose values can't be changed in the program explicitly but can be changed by any external device or hardware.

For example, the variable which is used to store system clock is defined as a volatile variable. The value of this variable is not changed explicitly in the program but is changed by the clock routine of the operating system.

Short Question & Answers

1. Explain Ternary Operator.

Ans :

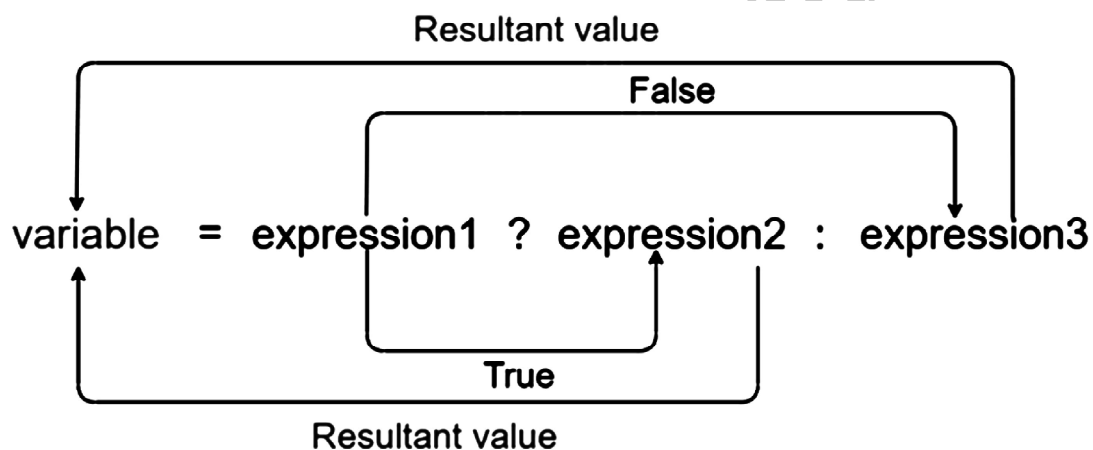
The conditional operator is also known as a ternary operator. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and ':'.

As conditional operator works on three operands, so it is also known as the ternary operator.

The behavior of the conditional operator is similar to the 'if-else' statement as 'if-else' statement is also a decision-making statement.

Syntax of a conditional operator

Expression1? expression2: expression3;



- In the above syntax, the expression1 is a Boolean condition that can be either true or false value.
- If the expression1 results into a true value, then the expression2 will execute.
- The expression2 is said to be true only when it returns a non-zero value.
- If the expression1 returns false value then the expression3 will execute.
- The expression3 is said to be false only when it returns zero value.

2. What is the use of if-else statements in C ? List out various types of if statements.

Ans :

The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.

There are the following variants of if statement in C language.

- If statement
- If-else statement
- If else-if ladder
- Nested if

3. Draw the flow chart of if-else-if-ladder statement in C ?

Ans :

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

```
if(condition1)
{
    //code to be executed if condition1 is true
}
else if(condition2)
{
    //code to be executed if condition2 is true
}
else if(condition3)
{
    //code to be executed if condition3 is true
}
...
Else
{
    //code to be executed if all the conditions are false
}
```

Flowchart of else-if ladder statement in C

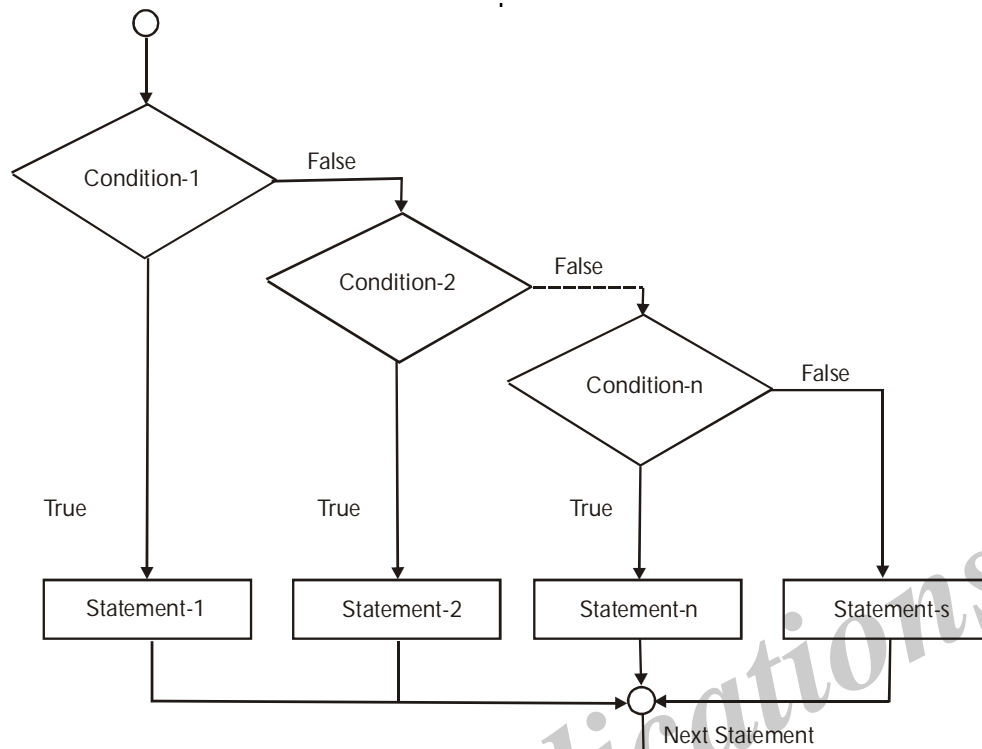


Fig.: else-if ladder

4. Explain in detail switch statement in C.

Ans :

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

The syntax of switch statement in c language is given below:

```

switch(expression)
{
    case value1:
        //code to be executed;
        break; //optional
    case value2:
        //code to be executed;
        break; //optional
    .....
    default:
        code to be executed if all cases are not matched;
}
  
```

Rules for switch statement in C language

- 1) The switch expression must be of an integer or character type.
- 2) The case value must be an integer or character constant.
- 3) The case value can be used only inside the switch statement.
- 4) The break statement in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as fall through the state of C switch statement.

5. What are the differences between If-else and switch statements.

Ans :

	If-else	switch
Definition	Depending on the condition in the 'if' statement, 'if' and 'else' blocks are executed.	The user will decide which statement is to be executed.
Expression	It contains either logical or equality expression.	It contains a single expression which can be either a character or integer variable.
Evaluation	It evaluates all types of data, such as integer, floating-point, character or Boolean.	It evaluates either an integer, or character.
Sequence of execution	First, the condition is checked. If the condition is true then 'if' block is executed otherwise 'else' block	It executes one case after another till the break keyword is not found, or the default statement is executed.
Default execution	If the condition is not true, then by default, else block will be executed.	If the value does not match with any case, then by default, default statement is executed.
Editing	Editing is not easy in the 'if-else' statement.	Cases in a switch statement are easy to maintain and modify. Therefore, we can say that the removal or editing of any case will not interrupt the execution of other cases.
Speed	If there are multiple choices implemented through 'if-else', then the speed of the execution will be slow.	If we have multiple choices then the switch statement is the best option as the speed of the execution will be much higher than 'if-else'.

6. What is a Loop ? What are the advantages of a Loop.

Ans :

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times.

For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1. It provides code reusability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

7. What is mean by nested loops?*Ans :*

C supports nesting of loops in C. Nesting of loops is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.

Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define 'while' loop inside a 'for' loop.

Syntax of Nested loop

Outer_loop

{

Inner_loop

{

// inner loop statements.

}

// outer loop statements.

}

Outer_loop and Inner_loop are the valid loops that can be a 'for' loop, 'while' loop or 'do-while' loop.

8. What are the differences between while loop and do-while loop.*Ans :*

S.No.	while loop	do-while loop
1.	While the loop is an entry control loop because firstly, the condition is checked, then the loop's body is executed.	The do-while loop is an exit control loop because in this, first of all, the body of the loop is executed then the condition is checked true or false.
2.	The statement of while loop may not be executed at all.	The statement of the do-while loop must be executed at least once.
3.	The while loop terminates when the condition becomes false.	As long as the condition is true, the compiler keeps executing the loop in the do-while loop.
4.	In a while loop, the test condition variable must be initialized first to check the test condition in the loop.	In a do-while loop, the variable of test condition initialized in the loop also.
5.	In a while loop, at the end of the condition, there is no semicolon. Syntax: while (condition)	In this, at the end of the condition, there is a semicolon. Syntax: while (condition);

6.	While loop is not used for creating menu-driven programs.	It is mostly used for creating menu-driven programs because at least one time; the loop is executed whether the condition is true or false.
7.	In a while loop, the number of executions depends on the condition defined in the while block.	In a do-while loop, irrespective of the condition mentioned, a minimum of 1 execution occurs.
8.	Syntax of while loop:while (condition) { Block of statements; } Statement-x;	Syntax of do-while loop: do { statements; } while (condition); Statement-x;
9.	Program of while loop: Program of while loop: #include #include Void main() { inti; clrscr(); i = 1;while(i<=10) { printf("hello"); i = i + 1; } getch(); }	Program of do-while loop: #include #include Void main() { inti; clrscr(); i = 1; do { printf("hello"); i = i + 1; } while(i<=10); getch(); }
10.	Flowchart of while loop:	Flowchart of do-while loop:

9. What is the use of break statement?

Ans :

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

Syntax:

break;

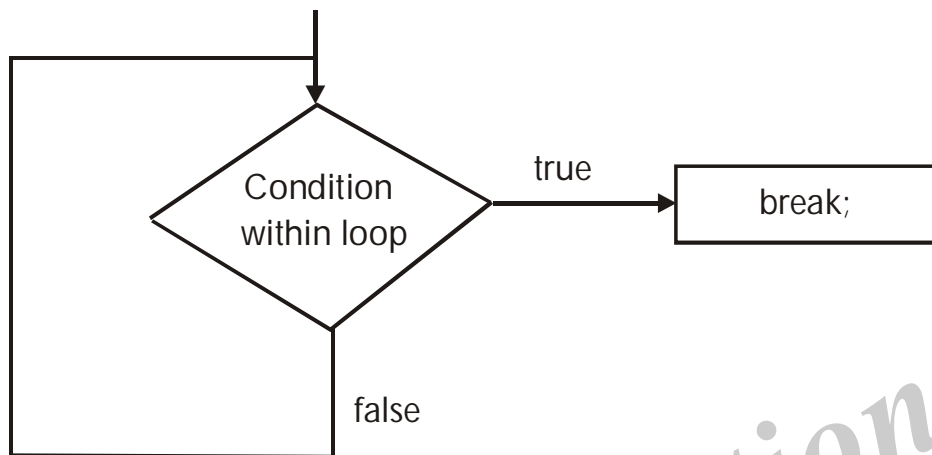


Fig.: Flowchart of break statement

Example

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
    int i;
    for(i = 0; i<10; i++)
    {
        printf("%d ",i);
        if(i == 5)
            break;
    }
    printf("came outside of loop i =%d",i);
}
```

Output

0 1 2 3 4 5 came outside of loop i = 5

10. What is Function ? what are the advantages of functions ?

Ans :

In c, we can divide a large program into the basic building blocks known as function. 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. In other words, we can say that the collection of functions creates a program. The function is also known as procedure or subroutine in other programming languages.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always a overhead in a C program.

11. How can we define a function in C ? Explain with an example ?

Ans :

The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list ) {  
    body of the function  
}
```

A function definition in C programming consists of a function header and a function body. Here are all the parts of a function.

- **Return Type :** A function may return a value. The `return_type` is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the `return_type` is the keyword `void`.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

Example:

```
/* function returning the max between two numbers */  
int max(int num1, int num2) {  
    /* local variable declaration */
```



```

int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}

```

12. List out various C Library functions.

Ans :

Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations. For example, printf is a library function used to print on the console. The library functions are created by the designers of compilers. All C standard library functions are defined inside the different header files saved with the extension .h. We need to include these header files in our program to make use of the library functions defined in such header files. For example, To use the library functions such as printf/scanf we need to include stdio.h in our program which is a header file that contains all the library functions regarding standard input/output.

The list of mostly used header files is given in the following table.

S.No.	Header file	Description
1.	stdio.h	This is a standard input/output header file. It contains all the library functions regarding standard input/output.
2.	conio.h	This is a console input/output header file.
3.	string.h	It contains all string related library functions like gets(), puts(), etc.
4.	stdlib.h	This header file contains all the general library functions like malloc(), calloc(), exit(), etc.
5.	math.h	This header file contains all the math operations related functions like sqrt(), pow(), etc.
6.	time.h	This header file contains all the time-related functions.
7.	ctype.h	This header file contains all character handling functions.
8.	stdarg.h	Variable argument functions are defined in this header file.
9.	signal.h	All the signal handling functions are defined in this header file.
10.	setjmp.h	This file contains all the jump functions.
11.	locale.h	This file contains locale functions.
12.	errno.h	This file contains error handling functions.
13.	assert.h	This file contains diagnostics functions.

13. What are the differences between call by value and call by reference.*Ans :*

S.No.	Call by value	Call by reference
1.	A copy of the value is passed into the function	An address of value is passed into the function
2.	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3.	Actual and formal arguments are created at the different memory location.	Actual and formal arguments are created at the same memory location

14. What is recursion?*Ans :*

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.

Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

Generally, iterative solutions are more efficient than recursion since function call is always overhead. Any problem that can be solved recursively, can also be solved iteratively.

In the following example, recursion is used to calculate the factorial of a number.

```
#include <stdio.h>
int fact (int);
int main()
{
    int n,f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("factorial = %d",f);
}
int fact(int n)
{
    if (n==0)
    {
```

```

        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}

```

Output

Enter the number whose factorial you want to calculate?5

factorial = 120

We can understand the above program of the recursive method call by the figure given below:

```

return 5 * factorial(4) = 120
└─ return 4 * factorial(3) = 24
    └─ return 3 * factorial(2) = 6
        └─ return 2 * factorial(1) = 2
            └─ return 1 * factorial(0) = 1

```

1 * 2 * 3 * 4 * 5 = 120

Fig.: Recursion

15. What is recursion function?

Ans :

A recursive function performs the tasks by dividing it into the subtasks. There is a termination condition defined in the function which is satisfied by some specific subtask. After this, the recursion stops and the final result is returned from the function.

The case at which the function doesn't recur is called the base case whereas the instances where the function keeps calling itself to perform a subtask, is called the recursive case. All the recursive functions can be written using this format.

Pseudocode for writing any recursive function is given below.

```

if (test_for_base)
{
    return some_value;
}
else if (test_for_another_base)
{
    return some_another_value;
}
else
{
    // Statements;
    recursive_call;
}

```

Let's see an example to find the nth term of the Fibonacci series.

```

#include<stdio.h>

int fibonacci(int);

void main ()
{
    int n,f;

    printf("Enter the value of n?");

    scanf("%d",&n);

    f = fibonacci(n);

    printf("%d",f);
}

int fibonacci (int n)
{
    if (n==0)
    {
        return 0;
    }
}

```

```

else if (n == 1)
{
    return 1;
}
else
{
    return fibonacci(n-1)+fibonacci (n-2);
}
}

```

Output

Enter the value of n?12

144

16. Discuss all the properties of storage classes in C.

Ans :

The following table provides detailed properties of all storage classes...

Storage Class	Keyword	Memory Location	Default Value	Scope	Life Time
Automatic	auto	Computer Memory (RAM)	Garbage Value	Local to the block in which the variable has defined	Till the control remains within the block in which variable is defined
External	extern	Computer Memory (RAM)	Zero	Global to the program (i.e., Throughout the program)	As long as the program's execution does not come to end
Static	static	Computer Memory (RAM)	Zero	Local to the block in which the variable has defined	The value of the persists between different function calls (i.e., Initialization is done only once)
Register	register	CPU Register	Garbage Value	Local to the block in which the variable has defined	Till the control remains within the block in which variable is defined

Choose the Correct Answers

1. ++ operator is called as _____. [a]
(a) Increment (b) Addition
(c) Increase (d) Adding
2. _____ operator. [a]
(a) Bitwise (b) Logical
(c) Ternary (d) Conditional
3. Conditional operator is also called as _____. [b]
(a) Bitwise (b) Ternary
(c) Logical (d) Relational
4. _____ statement is used to perform two operations for a single condition. [a]
(a) if - else (b) Nested - if
(c) else - if (d) for loop
5. _____ statement in switch case is not must. [d]
(a) switch (b) default
(c) case (d) break
6. _____ contains a single expression which can be either a character (or) integer variable. [b]
(a) if - else (b) switch
(c) for loop (d) None
7. _____ provides code reusability. [d]
(a) Operators (b) Keywords
(c) Statements (d) Loop
8. A loop with in athor loop is called as _____. [d]
(a) while - loop (b) do - while
(c) for - loop (d) nested loop

9. A large program divided into basic build blocks is known as _____. [c]
- (a) Operator (b) Functions
(c) Loop (d) Statements
10. Functions are divided into _____ categories. [a]
- (a) 2 (b) 3
(c) 4 (d) 5

Rahul Publications

Fill in the Blanks

1. Built in functions are also called as _____.
2. Functions are divided into library functions and _____.
3. In C, the inter function communication is classified into _____ types.
4. _____ provides functions to perform standard I/o operations.
5. _____ and _____ are two types of parameters.
6. _____ function performs the tasks by dividing into the sub-tasks.
7. _____ A copy of the value is passed into the function.
8. The default storage class of all local variables is _____ storage class.
9. _____ storage class is used to specify the memory of the variable that has to be allotted in CPU register.
10. A function can call it self is called _____.

ANSWERS

1. Standard functions
2. User - defined - functions
3. 3
4. <stdio.h>
5. Actual, Formal
6. Recursive
7. Call-by-value
8. Auto
9. Register
10. Nested function

One Mark Answers

1. Define Operator

Ans :

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.

2. While loop?

Ans :

While the loop is an entry control loop because firstly, the condition is checked, then the loop's body is executed.

3. List any four advantages of functions.

Ans :

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always a overhead in a C program.

4. What are the differences between call by value and call by reference.

Ans :

S.No.	Call by value	Call by reference
1.	A copy of the value is passed into the function	An address of value is passed into the function
2.	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3.	Actual and formal arguments are created at the different memory location.	Actual and formal arguments are created at the same memory location

5. Explain Ternary Operator.

Ans :

The conditional operator is also known as a ternary operator. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and '.:'

As conditional operator works on three operands, so it is also known as the ternary operator.

The behavior of the conditional operator is similar to the 'if-else' statement as 'if-else' statement is also a decision-making statement.

Syntax of a conditional operator

Expression1? expression2: expression3;

UNIT III

Preprocessors: Preprocessor Commands.

Arrays - Concepts, Using Arrays in C, Inter - Function Communication, Array Applications, Two - Dimensional Arrays, Multidimensional Arrays, Linear and Binary Search, Selection and Bubble Sort

3.1 PREPROCESSORS – PREPROCESSOR COMMANDS

Q1. What is the purpose of using preprocessors in C Programming?

(OR)

List out various preprocessor commands supported by C.

Ans :

(Imp.)

In C programming language, preprocessor directive is a step performed before the actual source code compilation. It is not part of the compilation. Preprocessor directives in C programming language are used to define and replace tokens in the text and also used to insert the contents of other files into the source file.

When we try to compile a program, preprocessor commands are executed first and then the program gets compiled.

Every preprocessor command begins with # symbol. We can also create preprocessor commands with parameters.

Following are the preprocessor commands in C programming language.

#define

#define is used to create symbolic constants (known as macros) in C programming language. This preprocessor command can also be used with parameterized macros.

Example

```
#include <stdio.h>
#include <conio.h>
#define PI 3.14
#define SQR(x) x*x           //Parameterized Macro

void main(){
    double radius, area ;
    clrscr() ;

    printf("Enter the radius: ");
    scanf("%ld",&radius);
```

```
area = PI * SQR(radius) ;  
printf("area = %ld",area);  
getch();  
}
```

Output

#undef

#undef is used to destroy a macro that was already created using #define.

#ifdef

#ifdef returns TRUE if the macro is defined and returns FALSE if the macro is not defined.

#ifndef

#ifndef returns TRUE if the specified macro is not defined otherwise returns FALSE.

#if

#if uses the value of specified macro for conditional compilation.

#else

#else is an alternative for #if.

#elif

#elif is a #else followed by #if in one statement.

#endif

#endif is used to terminate preprocessor conditional macro.

#include

#include is used to insert specific header file into C program.

#error

#error is used to print error message on stderr.

#pragma

#pragma is used to issue a special command to the compiler.

In C programming language, there are some pre-defined macros and they are as follows...

1. **__DATE__** : The current date as characters in "MMM DD YYYY" format.
2. **__TIME__** : The current time as characters in "HH : MM : SS" format.
3. **__FILE__** : This contains the current file name.
4. **__LINE__** : This contains the current line number.
5. **__STDC__** : Defines 1 when compiler compiles with ANSI Standards.

Q2. Explain various types of preprocessor directives.*Ans :*

The #define Preprocessor Directive as Function

The #define preprocessor directive can be used as inline function which is replaced at compile time. The definition of macro accepts an argument and that argument is replaced by the actual argument found in program.

Example of #define preprocessor as function

```
#include <stdio.h>
#define SQUARE(x)    x*x
void main()
{
    int num;
    printf("Enter any number : ");
    scanf("%d",&num);
    printf("\nThe square is :
           %d",SQUARE(num));
}
```

Output

Enter any number : 5

The square is : 25

The #include Preprocessor Directive

The #include preprocessor directive is used to include another source file in our source code. This is commonly used to include header files in our program.

Syntax of #include preprocessor directive

#include "filename.h"

or

#include <filename.h>

The source file to be added must be enclosed within double quotes (" ") or angle brackets (< >) . If the file is enclosed in double quotes the search will be done in current directory. If the file is enclosed in angle brackets the search will be done in standard directories (include directory) where the libraries are stored.

The #if-#else-#endif Preprocessor Directive

The #if preprocessor directive takes condition in parenthesis, if condition is true, then the statements given between #if and #else will get execute. If condition is false, then statements given between #else and #endif will get execute.

Syntax of #if-#else-#endif Preprocessor Directive

#if(condition)

#else

#endif

Example of #if-#else-#endif Preprocessor Directive

```
#include <stdio.h>
#define MAX 45
void main()
{
    #if MAX > 40
        printf("Yes, MAX is greater then 40.");
    #else
        printf("No, MAX is not greater then 40.");
    #endif
}
```

Output

Yes, MAX is Greater then 40.

The #elif Preprocessor Directive

The #elif preprocessor directive is similar to if-else ladder statement. It is used for checking multiple conditions, if the first condition will not satisfy, compiler will jump to #else block and check other condition is true or not and so on.

Syntax of #if-#elif-#else-#endif Preprocessor Directive

```

    #if(condition)
        -----
        -----
    #elif(condition)
        -----
        -----
    #elif(condition)
        -----
        -----
    #else
        -----
        -----
    #endif

```

Example of #if-#elif-#else-#endif Preprocessor Directive

```

#include<stdio.h>
#define MKS 65
void main()
{
    #if MKS >= 78
        printf("\nGrade A");
    #elif MKS >= 50
        printf("\nGrade B");
    #elif MKS >= 25
        printf("\nGrade C");
    #else
        printf("\nGrade D");
    #endif
}

```

Output

Grade B

The #ifdef Preprocessor Directive

The #ifdef preprocessor directive is used to check whether the macro-name is previously defined or not, if defined then the statements given between #ifdef and #else will get execute.

Syntax of #ifdef preprocessor directive

```

#ifdef macro-name
    -----
    -----
#else
    -----
    -----
#endif

```

Example of #ifdef preprocessor directive

```

#include<stdio.h>
#define MKS 65
void main()
{
    #ifdef MONTH
        printf("\nMONTH is defined.");
    #else
        printf("\nMONTH is not defined.");
    #endif
}

```

Output

MONTH is not defined.

The #ifndef Preprocessor Directive

The #ifndef preprocessor directive is used to check whether the macro-name is previously defined or not, if not defined then the statements given between #ifndef and #else will get execute.

Syntax of #ifndef preprocessor directive

```

#ifndef macro-name
    -----
    -----
#else
    -----
    -----
#endif

```

Example of #ifndef preprocessor directive

```

#include<stdio.h>
#define MKS 65

```

```

void main()
{
    #ifndef MAX
        printf("\nMAX is not defined.");
    #else
        printf("\nMAX is defined.");
    #endif
}

```

Output

MAX is not defined.

Note

The #ifdef and #ifndef can not use the #elif preprocessor directive.

The #line Preprocessor Directive

The #line preprocessor directive is used to overwrite the default value of pre-defined macro-names `_LINE_` and `_FILE_`. By default `_LINE_` displays the current line number and `_FILE_` displays the current filename.

Example of #line preprocessor directive 1:

```

#include <stdio.h>
void main()
{
    printf("\nLine number %d", _LINE__);
    printf("\nFile name %s", _FILE__);
}

```

Output

Line number : 4
File name : LineDemo.c

Note :

We have saved the above code in LineDemo.c file.

Example of #line preprocessor directive 2:

```

#include <stdio.h>
#line 25 Demo.c
void main()

```

```

{
    printf("\nLine number %d", _LINE__);
    printf("\nFile name %s", _FILE__);
}

```

Output

Line number : 25
File name : Demo.c

The above code is still in LineDemo.c file but we have overwritten the line number and the file name using #line directive.

The #error Preprocessor Directive

The #error preprocessor directive is used to force the compile to stop compiling the source code. In other words we can manually force the compiler to stop compiling and give fatal error.

Example of #error preprocessor directive 1:

```

#include <stdio.h>
void main()
{
    #ifndef MAX
        printf("\nMAX is not defined.");
        //Statement 1
    #else
        printf("\nMAX is defined.");
        //Statement 2
    #endif
    printf("\nEnd of program.");
    //Statement 3
}

```

Output

MAX is not defined.
End of program.

Example of #error preprocessor directive 2:

```

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

```

```

#ifndef MAX
    #error MAX is not defined.
    //Statement 1
#else
    printf("\nMAX is defined.");
    //Statement 2
#endif
printf("\nEnd of program.");
//Statement 3
}

```

3.2 ARRAYS – CONCEPTS USING ARRAYS IN C

Q3. What is array ? Explain its advantages.

Ans : (Imp.)

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

C array is beneficial if you have to store similar elements. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

By using the array, we can access the elements easily. Only a few lines of code are required to access the elements of the array.

Properties of Array

The array contains the following properties.

- Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.

- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Advantages of Arrays

1. **Code Optimization:** Less code to access the data.
2. **Ease of traversing:** By using the for loop, we can retrieve the elements of an array easily.
3. **Ease of sorting:** To sort the elements of the array, we need a few lines of code only.
4. **Random Access:** We can access any element randomly using the array.

Disadvantage of C Array

Fixed Size

Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically.

Q4. What is the need of array ?

Ans :

In computer programming, the most of the cases requires to store the large number of data of similar type. To store such amount of data, we need to define a large number of variables. It would be very difficult to remember names of all the variables while writing the programs. Instead of naming all the variables with a different name, it is better to define an array and store all the elements into it.

Following example illustrates, how array can be useful in writing code for a particular problem.

In order to illustrate the importance of array, we have created two programs, one is without using array and other involves the use of array to store marks.

Program without array

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

```
void main ()
```

```
{
```

```
    int marks_1 = 56, marks_2 = 78,
    marks_3 = 88, marks_4 = 76, marks_5 = 56,
    marks_6 = 89;
```

```

float avg = (marks_1 + marks_2 +
marks_3 + marks_4 + marks_5 +marks_6) /
6;

printf(avg);
}

```

Program by using array

```

#include <stdio.h>

void main ()
{
    int marks[6] = {56,78,88,76,56,89};
    int i;
    float avg;
    for (i=0; i<6; i++ )
    {
        avg = avg + marks[i];
    }
    printf(avg);
}

```

Q5. How memory allocation is done in arrays ?

Ans :

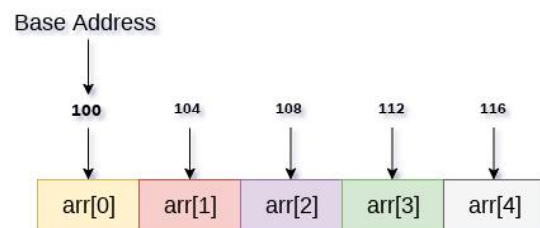
Memory Allocation of the array

All the data elements of an array are stored at contiguous locations in the main memory. The name of the array represents the base address or the address of first element in the main memory. Each element of the array is represented by a proper indexing.

The indexing of the array can be defined in three ways.

1. **0 (zero - based indexing):** The first element of the array will be arr[0].
2. **1 (one - based indexing):** The first element of the array will be arr[1].
3. **n (n - based indexing):** The first element of the array can reside at any random index number.

In the following image, we have shown the memory allocation of an array arr of size 5. The array follows 0-based indexing approach. The base address of the array is 100th byte. This will be the address of arr[0]. Here, the size of int is 4 bytes therefore each element will take 4 bytes in the memory.



int arr[5]

In 0 based indexing, If the size of an array is n then the maximum index number, an element can have is n-1. However, it will be n if we use 1 based indexing.

Q6. Explain the process of declaring and initializing an array ?

Ans :

(Imp.)

Declaration of C Array

We can declare an array in the c language in the following way.

```
data_type array_name[array_size];
```

Now, let us see the example to declare the array.

```
int marks[5];
```

Here, int is the *data_type*, marks are the *array_name*, and 5 is the *array_size*.

Initialization of C Array

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

```

marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;

```

80	60	70	85	75
----	----	----	----	----

marks[0] marks[1] marks[2] marks[3] marks[4]

Initialization of Array

example

```
#include<stdio.h>

int main(){
    int i=0;
    int marks[5]; //declaration of array
    marks[0]=80; //initialization of array
    marks[1]=60;
    marks[2]=70;
    marks[3]=85;
    marks[4]=75;
    //traversal of array
    for(i=0;i<5;i++){
        printf("%d \n",marks[i]);
    } //end of for loop
    return 0;
}
```

Output

80
60
70
85
75

We can also define an initialize an array by using following method also.

Declaration with Initialization

We can initialize the c array at the time of declaration. Let's see the code.

```
int marks[5]={20,30,40,50,60}
```

In such case, there is no requirement to define the size. So it may also be written as the following code.

```
int marks[]={20,30,40,50,60};
```

Let's see the C program to declare and initialize the array in C.

```
#include<stdio.h>

int main(){
    int i=0;
    int marks[5]={20,30,40,50,60}; //
    declaration and initialization of array
    //traversal of array
    for(i=0;i<5;i++){
        printf("%d \n",marks[i]);
    }
    return 0;
}
```

Output

20
30
40
50
60

Q7. Write a C program for sorting of an elements using array .

Ans:

```
#include<stdio.h>

void main ()
{
    int i, j,temp;
    int a[10] = { 10, 9, 7, 101, 23, 44, 12,
                  78, 34, 23};
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
            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

Printing sorted elements lis...

7
9
10
12
23
23
34
44
78
101

3.2.1 Inter function communication

Q8. Explain the process of passing an array to function.

Ans :

In C, there are various general problems which requires passing more than one variable of the same type to a function. For example, consider a function which sorts the 10 elements in ascending order. Such a function requires 10 numbers to be passed as the actual parameters from the main function. Here, instead of declaring 10 different numbers and then passing into the function, we can declare and initialize an array and pass that into the function. This will resolve all the complexity since the function will now work for any number of values.

As we know that the array_name contains the address of the first element. Here, we must notice that we need to pass only the name of the array in the function which is intended to accept an array. The array defined as the formal parameter will automatically refer to the array specified by the array name defined as an actual parameter.

Consider the following syntax to pass an array to the function.

functionname(arrayname)//passing array

There are 3 ways to declare the function which is intended to receive an array as an argument.

First way

return_type function(type arrayname[])

Declaring blank subscript notation [] is the widely used technique.

Second way

return_type function (type arrayname [SIZE])

Optionally, we can define size in subscript notation [].

Third way

return_type function(type *arrayname)

Example

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

```
int minarray(int arr[],int size){
```

```
int min=arr[0];
```

```
int i=0;
```

```
for(i=1;i<size;i++){
```

```
if(min>arr[i]){
```

```
min=arr[i];
```

```
}
```

```
}//end of for
```

```
return min;
```

```
}//end of function
```

```
int main(){
```

```
int i=0,min=0;
```

```
int numbers[]={4,5,7,3,8,9};//declaration of array
```

```

        min=minarray(numbers,6);//passing array
with size
printf("minimum number is %d \n",min);
return 0;
}

```

Output

minimum number is 3

Returning array from the function

As we know that, a function can not return more than one value. However, if we try to write the return statement as return a, b, c; to return three values (a,b,c), the function will return the last mentioned value which is c in our case. In some problems, we may need to return multiple values from a function. In such cases, an array is returned from the function.

Returning an array is similar to passing the array into the function. The name of the array is returned from the function. To make a function returning an array, the following syntax is used.

```

int * Function_name()
{
//some statements;
return array_type;
}

```

To store the array returned from the function, we can define a pointer which points to that array. We can traverse the array by increasing that pointer since pointer initially points to the base address of the array. Consider the following example that contains a function returning the sorted array.

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

```

int* Bubble_Sort(int[]);
void main ()
{
    int arr[10] = { 10, 9, 7, 101, 23, 44, 12,
                    78, 34, 23};

    int *p = Bubble_Sort(arr), i;
    printf("printing sorted elements ...\n");
    for(i=0;i<10;i++)
    {

```

```

        printf("%d\n",*(p+i));
    }
}

int* Bubble_Sort(int a[])
//array a[] points to arr.
{
    int i, j,temp;
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
            if(a[j] < a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    return a;
}

```

Output

Printing Sorted Element List ...

```

7
9
10
12
23
23
34
44
78
101

```

3.2.2 Array Applications

Q9. List out the applications of Arrays.

Ans :

In c programming language, arrays are used in wide range of applications. Few of them are as follows...

➤ **Arrays are used to Store List of values**

In c programming language, single dimensional arrays are used to store list of values of same datatype. In other words, single dimensional arrays are used to store a row of values. In single dimensional array data is stored in linear form.

➤ **Arrays are used to Perform Matrix Operations**

We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.

➤ **Arrays are used to implement Search Algorithms**

We use single dimensional arrays to implement search algorithms like ...

1. Linear Search
2. Binary Search

➤ **Arrays are used to implement Sorting Algorithms**

We use single dimensional arrays to implement sorting algorithms like ...

1. Insertion Sort
2. Bubble Sort
3. Selection Sort
4. Quick Sort
5. Merge Sort, etc.,

➤ **Arrays are used to implement Datastructures**

We use single dimensional arrays to implement datastructures like...

1. **Stack Using Arrays**
2. **Queue Using Arrays**

➤ Arrays are also used to implement CPU Scheduling Algorithms

Two dimensional Array, Multi-dimensional Array

In c programming language, arrays are classified into **two types**. They are as follows...

1. Single Dimensional Array / One Dimensional Array
2. Two Dimensional Array
3. Multi Dimensional Array

1. Single Dimensional Array

In c programming language, single dimensional arrays are used to store list of values of same datatype. In other words, single dimensional arrays are used to store a row of values. In single dimensional array, data is stored in linear form. Single dimensional arrays are also called as one-dimensional arrays, Linear Arrays or simply 1-D Arrays.

We can declare an single dimensional array in the c language in the following way.

```
data_type array_name[array_size];
```

Now, let us see the example to declare the array.

```
int marks[5];
```

2. Two dimensional arrays

- At times we need to store the data in form of tables or matrices. For this, we can use the two dimensional arrays.
- This array is specified by using two subscripts where one subscript is denoted as the row and the other as the column.
- It is also viewed as an array of arrays.

Syntax

```
Data type arratname[][];
```

Example

```
Int A[][];
```

3. Multi Dimensional Array in C

An array which is declared as more than one size is called as multi dimensional array. The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Q10. How can we declare two dimensional arrays in C?

Ans :

Two dimensional arrays

- At times we need to store the data in form of tables or matrices. For this, we can use the two dimensional arrays.
- This array is specified by using two subscripts where one subscript is denoted as the row and the other as the column.
- It is also viewed as an array of arrays.

	Columns		
Rows	score [0] [0]	score [0] [1]	score[0] [2]
	score[1] [0]	score[1] [1]	score[1] [2]
	score[2] [0]	score[2] [1]	score[2] [2]

Fig: Two-dimensional array

Above figure shows how the rows and columns are addressed in a two- dimensional array. The first subscript denotes the row and the second subscript denotes the column.

i) Declaration of a two-dimensional array

The declaration of the array will tell the compiler as to which array is being used for the program.

Syntax

data_type array_name [row_size] [column_size] ;

- The declaration of the rows and columns is compulsory for a two-dimensional array.

- We cannot replace the row size with the column size and the column size to row size. The proper sequence has to be maintained.

ii) Initialization of two-dimensional array

A two-dimensional array is initialized in the same way as the one-dimensional array.

Example: Initialization of 2D array

```
int score [3] [2] = {50, 60, 70, 95, 3, 36};
```

The initialization of the array is done row-by-row.

The above can also be written in the following manner:

```
int score [3] [2] = {{50, 60} , {70,95} ,{3,36}};
```

The elements will be seen in the following format once they are initialized.

```
score[0][0] = 50 score [0][1] = 60
```

```
score[1][0] = 70 score [1][1] = 95
```

```
score[2][0] = 3 score [2][1] = 36
```

The above example has been divided into three rows and two columns.

iii) Accessing the elements

- The elements of this array are stored in a continuous memory location.
- The last subscript varies rapidly as compared to the first one.
- Two for loops required for scanning the elements of the two-dimensional array. The first for will loop for each row and second for will loop for each column for every row.

Example: Simple program for printing the elements of 2D array.

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

```
void main()
```

```
{
```

```
int score[3][2]= {10,20,30,40,50,60};
```

```
int i,j;
```

```
for(i=0;i<3;i++)
```

```
{
```

```

        printf("\n");
        for(j=0;j<2;j++)
            printf("%d\t",score[i][j]);
    }
}

```

Output:

```

10    20
30    40
50    60

```

iv) Passing 2D array as parameters to functions

There are three ways of passing the parameters to the functions.

a) Passing individual elements: They are done in the same manner as that of the one-dimensional array.

b) Passing a row

- A row can be passed by indexing the array name with the number of the row.
- When a single row is sent to the called function, it is received as a one-dimensional array.

Example: Passing a row

```

void main() // Calling function
{
    int score [2][3] = {{10,20,30},
                        {40, 50, 60}};

    func (score [10]);
}

void func (int score[ ]) // Called function
{
    int i;
    for (i=0;i<5;i++)
        printf ("%d", score [i] * 10);
}

```

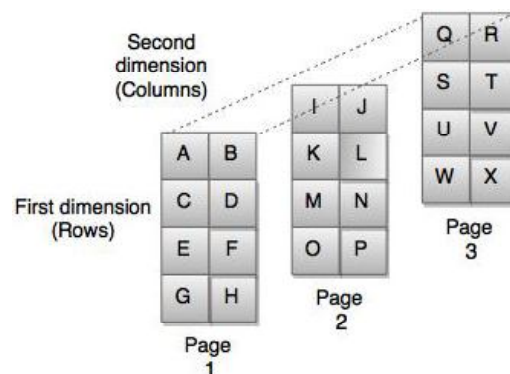
c) Passing the entire 2D array

We use the array name as the actual parameter for passing a 2D array to a function.

But the parameter in the called function should denote that the array has two dimensions.

Q11. Discuss in detail about Multi dimensional arrays in C.*Ans :***(Imp.)****Multidimensional array**

- In simple terms it is called an array of arrays.
- We have 'n' number of indexes in this array.
- It is specified by using 'n' number of indices.
- The requirement of the memory increases with the number of indices that it uses.
- But, if we talk practically we would not use more than three indices.
- These arrays are declared and initialized in the same manner as that of one and two-dimensional arrays.

**Fig.: Multidimensional Array****Example: Write a program to read and display a 2*2*2 array.**

```

#include <stdio.h>

void main()
{
    int arr[3][3][3],i,j,k;
    printf("\n Enter the elements for the array:");
    for(i=0;i<2;i++)

```

```
{
    for(j=0;j<2;j++)
    {
        for(k=0;k<2;k++)
        {
            printf("\n array [%d][%d][%d] = ",i,j,k);
            scanf("%d",&arr[i][j][k]);
        }
    }
}
printf("\n The matrix is:");
for(i=0;i<2;i++)
{
    printf("\n\n");
    for(j=0;j<2;j++)
    {
        printf("\n");
        for(k=0;k<2;k++)
            printf("\t array[%d][%d][%d]=%d",i,j,k, arr[i][j][k]);
    }
}
```

Output

Enter the elements for the array: 10,20,30,40,50,60,70,80

array [0][0][0] = 10

array [0][0][1] = 20

array [0][1][0] = 30

array [0][1][1] = 40

array [1][0][0] = 50

array [1][0][1] = 60

array [1][1][0] = 70

array [1][1][1] = 80

The matrix is:

array[0][0][0]=10

array[0][0][1]=20

array[0][1][0]=30

array[0][1][1]=40

array[1][0][0]=50

array[1][0][1]=60

array[1][1][0]=70

array[1][1][1]=80

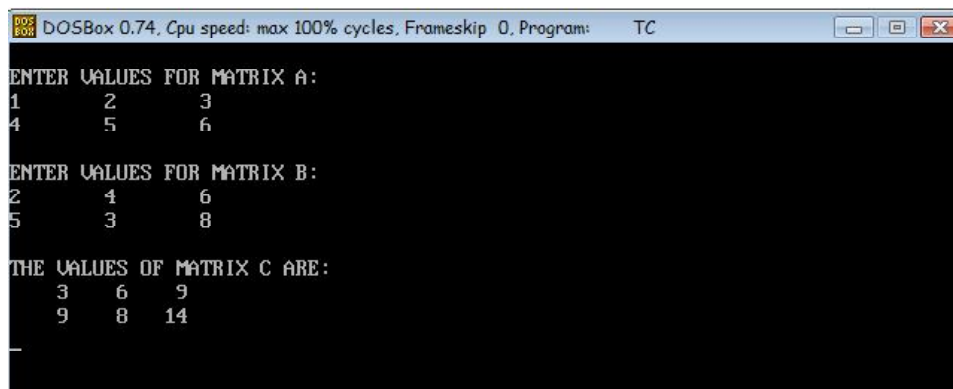
Q12. Write a C Programs to add two matrices using arrays.

Ans :

(Imp.)

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[2][3],b[2][3],c[2][3],i,j;
    clrscr();
    printf("\nENTER VALUES FOR MATRIX A:\n");
    for(i=0;i<2;i++)
        for(j=0;j<3;j++)
            scanf("%d",&a[i][j]);
    printf("\nENTER VALUES FOR MATRIX B:\n");
    for(i=0;i<2;i++)
        for(j=0;j<3;j++)
            scanf("%d",&b[i][j]);
    for(i=0;i<2;i++)
        for(j=0;j<3;j++)
            c[i][j]=a[i][j]+b[i][j];
    printf("\nTHE VALUES OF MATRIX C ARE:\n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
            printf("%5d",c[i][j]);
        printf("\n");
    }
    getch();
}
```

Output



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

ENTER VALUES FOR MATRIX A:
1      2      3
4      5      6

ENTER VALUES FOR MATRIX B:
2      4      6
5      3      8

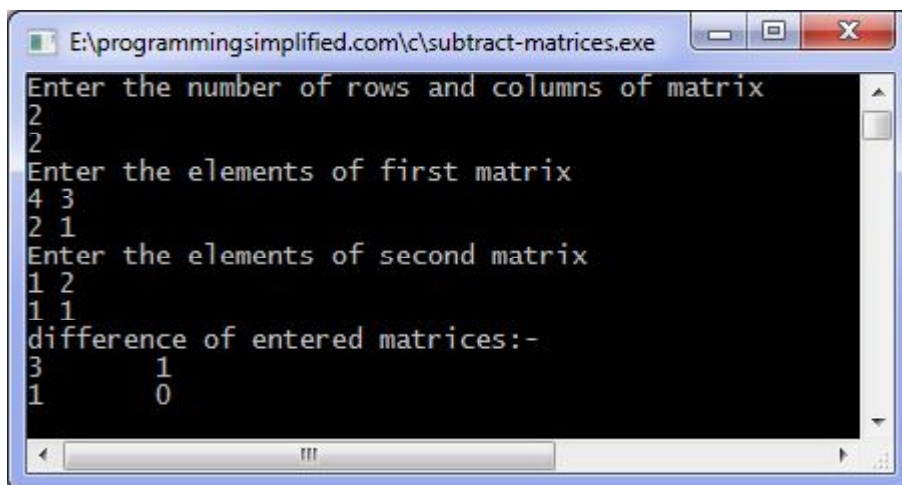
THE VALUES OF MATRIX C ARE:
3      6      9
9      8     14
```

Q13. Write a C Program for subtraction of two matrices.

Ans :

```
#include <stdio.h>
int main()
{
    int m, n, c, d, first[10][10], second[10][10], difference [10] [10];
    printf("Enter the number of rows and columns of matrix\n");
    scanf("%d%d", &m, &n);
    printf("Enter the elements of first matrix\n");
    for (c = 0; c < m; c++)
        for (d = 0 ; d < n; d++)
            scanf("%d", &first[c][d]);
    printf("Enter the elements of second matrix\n");
    for (c = 0; c < m; c++)
        for (d = 0; d < n; d++)
            scanf("%d", &second[c][d]);
    printf("Difference of entered matrices:-\n");
    for (c = 0; c < m; c++) {
        for (d = 0; d < n; d++) {
            difference[c][d] = first[c][d] - second[c][d];
            printf("%d\t", difference[c][d]);
        }
        printf("\n");
    }
    return 0;
}
```

Output



```
E:\programmingsimplified.com\c\subtract-matrices.exe
Enter the number of rows and columns of matrix
2
2
Enter the elements of first matrix
4 3
2 1
Enter the elements of second matrix
1 2
1 1
difference of entered matrices:-
3      1
1      0
```


Q14. Write a C Program for multiplication of two matrices.

Ans :

(Imp.)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
    system("cls"); printf("enter the number of row=");
    scanf("%d",&r); printf("enter the number of column=");
    scanf("%d",&c);
    printf("enter the first matrix element=\n");
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("enter the second matrix element=\n");
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            scanf("%d",&b[i][j]);
        }
    }
    printf("multiply of the matrix=\n");
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            mul[i][j]=0;
            for(k=0;k<c;k++)
            {
                mul[i][j] += a[i][k]*b[k][j];
            }
        }
    }
```

```

}

//for printing result
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
    {
        printf("%d\t",mul[i][j]);
    }
    printf("\n");
}
return 0;
}

```

Output

```

enter the number of row=3
enter the number of column=3
enter the first matrix element=
1 1 1
2 2 2
3 3 3
enter the second matrix element=
1 1 1
2 2 2
3 3 3
multiply of the matrix=
6 6 6
12 12 12
18 18 18

```

$$\text{Matrix 1} \begin{Bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{Bmatrix} \quad \text{Matrix 2} \begin{Bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{Bmatrix}$$

$$\begin{matrix} \text{Matrix 1} \\ * \\ \text{Matrix 2} \end{matrix} \begin{Bmatrix} 1*1+1*2+1*3 & 1*1+1*2+1*3 & 1*1+1*2+1*3 \\ 2*1+2*2+2*3 & 2*1+2*2+2*3 & 2*1+2*2+2*3 \\ 3*1+3*2+3*3 & 3*1+3*2+3*3 & 3*1+3*2+3*3 \end{Bmatrix}$$

$$\begin{matrix} \text{Matrix 1} \\ * \\ \text{Matrix 2} \end{matrix} \begin{Bmatrix} 6 & 6 & 6 \\ 12 & 12 & 12 \\ 18 & 18 & 18 \end{Bmatrix}$$

3.4 LINEAR SEARCH AND BINARY SEARCH

Q15. What is searching ? Explain various searching methods.

Ans :

Searching is the process of finding some particular element in the list. If the element is present in the list, then the process is called successful and the process returns the location of that element, otherwise the search is called unsuccessful.

There are two popular search methods that are widely used in order to search some item into the list. However, choice of the algorithm depends upon the arrangement of the list.

- Linear Search
- Binary Search

Q16. What is linear search? Write a linear search algorithm.

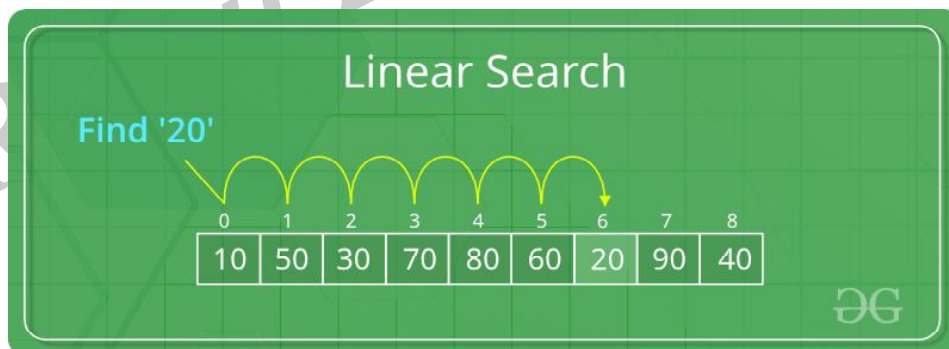
Ans :

(Imp.)

Linear search is the simplest search algorithm and often called sequential search. In this type of searching, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match found then location of the item is returned otherwise the algorithm return NULL.

A simple approach is to do a **linear search**, i.e

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
- If x matches with an element, return the index.
- If x doesn't match with any of elements, return -1.



Linear search is mostly used to search an unordered list in which the items are not sorted. The algorithm of linear search is given as follows.

Algorithm

- LINEAR_SEARCH(A, N, VAL)
- **Step 1:** [INITIALIZE] SET POS = -1
- **Step 2:** [INITIALIZE] SET I = 1
- **Step 3:** Repeat Step 4 while I <= N
- **Step 4:** IF A[I] = VAL

SET POS = I

PRINT POS

Go to Step 6

[END OF IF]

SET I = I + 1

[END OF LOOP]

➤ **Step 5:** IF POS = -1

PRINT " VALUE IS NOT PRESENTIN THE ARRAY "

[END OF IF]

➤ **Step 6:** EXIT

Q17. Write a c program to implement linear search mechanism.

Ans :

```
#include<stdio.h>
void main ()
{
    int a[10] = {10, 23, 40, 1, 2, 0, 14, 13, 50, 9};
    int item, i,flag;
    printf("\nEnter Item which is to be searched\n");
    scanf("%d",&item);
    for (i = 0; i < 10; i++)
    {
        if(a[i] == item)
        {
            flag = i+1;
            break;
        }
        else
            flag = 0;
    }
    if(flag != 0)
    {
        printf("\nItem found at location %d\n",flag);
    }
    else
    {
        printf("\nItem not found\n");
    }
}
```

Output

Enter Item which is to be searched

20

Item not found

Enter Item which is to be searched

23

Item found at location 2

Q18. Write a Binary search algorithm.

Ans :

Binary search is the search technique which works efficiently on the sorted lists. Hence, in order to search an element into some list by using binary search technique, we must ensure that the list is sorted.

Binary search follows divide and conquer approach in which, the list is divided into two halves and the item is compared with the middle element of the list. If the match is found then, the location of middle element is returned otherwise, we search into either of the halves depending upon the result produced through the match.

Binary search algorithm is given below.

BINARY_SEARCH(A, lower_bound, upper_bound, VAL)

Step 1: [INITIALIZE] SET BEG = lower_bound

END = upper_bound, POS = -1

Step 2: Repeat Steps 3 and 4 while BEG ≤ END

Step 3: SET MID = (BEG + END)/2

Step 4: IF A[MID] = VAL

SET POS = MID

PRINT POS

Go to Step 6

ELSE IF A[MID] > VAL

SET END = MID - 1

ELSE

SET BEG = MID + 1

[END OF IF]

[END OF LOOP]

Step 5: IF POS = -1

PRINT "VALUE IS NOT PRESENT IN THE ARRAY"

[END OF IF]

Step 6: EXIT

Example

Let us consider an array $arr = \{1, 5, 7, 8, 13, 19, 20, 23, 29\}$. Find the location of the item 23 in the array.

In 1st step

1. $BEG = 0$
2. $END = 8$
3. $MID = 4$
4. $a[mid] = a[4] = 13 < 23$, therefore

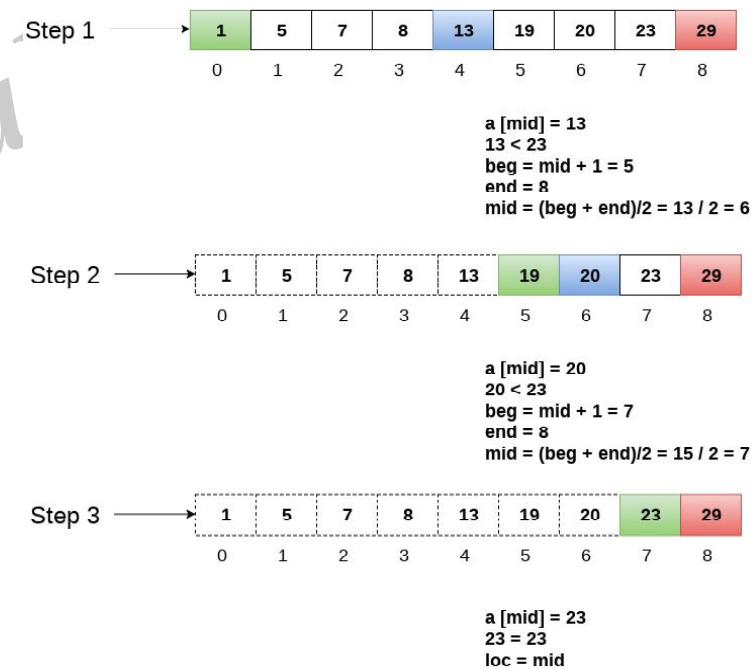
In Second step

1. $Beg = mid + 1 = 5$
2. $End = 8$
3. $mid = 13/2 = 6$
4. $a[mid] = a[6] = 20 < 23$, therefore;

in third step

1. $beg = mid + 1 = 7$
2. $End = 8$
3. $mid = 15/2 = 7$
4. $a[mid] = a[7]$
5. $a[7] = 23 = \text{item};$
6. therefore, set location = mid;
7. The location of the item will be 7.

Item to be searched = 23



Return location 7

Q19. Explain Binary search using an example.

Ans :

(Imp.)

A binary search is a search in which the middle element is calculated to check whether it is smaller or larger than the element which is to be searched. The main advantage of using binary search is that it does not scan each element in the list. Instead of scanning each element, it performs the searching to the half of the list. So, the binary search takes less time to search an element as compared to a linear search.

The one **pre-requisite of binary search** is that an array should be in sorted order, whereas the linear search works on both sorted and unsorted array. The binary search algorithm is based on the divide and conquer technique, which means that it will divide the array recursively.

There are three cases used in the binary search:

Case 1: $\text{data} < a[\text{mid}]$ then $\text{left} = \text{mid} + 1$.

Case 2: $\text{data} > a[\text{mid}]$ then $\text{right} = \text{mid} - 1$

Case 3: `data = a[mid]` // element is found

In the above case, 'a' is the name of the array, mid is the index of the element calculated recursively, data is the element that is to be searched, left denotes the left element of the array and right denotes the element that occur on the right side of the array.

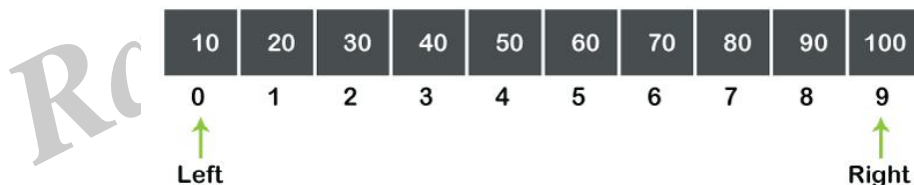
Let's understand the working of binary search through an example.

Suppose we have an array of 10 size which is indexed from 0 to 9 as shown in the below figure:

We want to search for 70 element from the above array.

Step 1

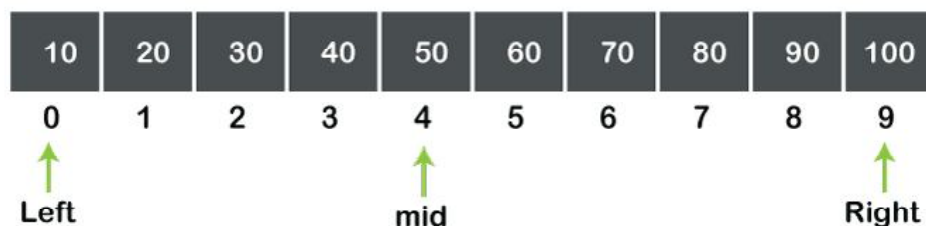
First, we calculate the middle element of an array. We consider two variables, i.e., left and right. Initially, left = 0 and right = 9 as shown in the below figure:



The middle element value can be calculated as:

$$\text{mid} = \frac{\text{left} + \text{right}}{2}$$

Therefore, $\text{mid} = 4$ and $a[\text{mid}] = 50$. The element to be searched is 70, so $a[\text{mid}]$ is not equal to data. The case 2 is satisfied, i.e., $\text{data} > a[\text{mid}]$.

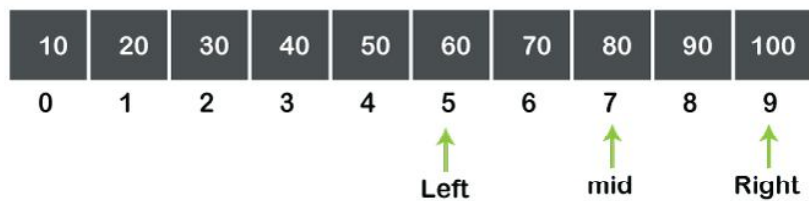


Step 2

As $\text{data} > a[\text{mid}]$, so the value of left is incremented by $\text{mid} + 1$, i.e., $\text{left} = \text{mid} + 1$. The value of mid is 4, so the value of left becomes 5. Now, we have got a subarray as shown in the below figure:



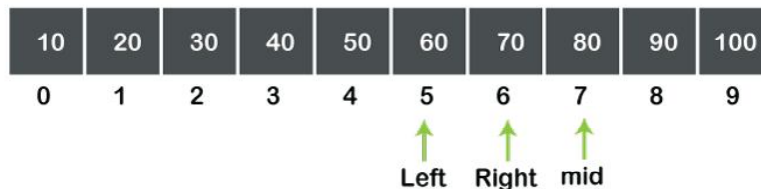
Now again, the mid-value is calculated by using the above formula, and the value of mid becomes 7. Now, the mid can be represented as:



In the above figure, we can observe that $a[\text{mid}] > \text{data}$, so again, the value of mid will be calculated in the next step.

Step 3

As $a[\text{mid}] > \text{data}$, the value of right is decremented by $\text{mid} - 1$. The value of mid is 7, so the value of right becomes 6. The array can be represented as:



The value of mid will be calculated again. The values of left and right are 5 and 6, respectively. Therefore, the value of mid is 5. Now the mid can be represented in an array as shown below:



In the above figure, we can observe that $a[\text{mid}] < \text{data}$.

Step 4

As $a[\text{mid}] < \text{data}$, the left value is incremented by $\text{mid} + 1$. The value of mid is 5, so the value of left becomes 6.

Now the value of mid is calculated again by using the formula which we have already discussed. The values of left and right are 6 and 6 respectively, so the value of mid becomes 6 as shown in the below figure:



We can observe in the above figure that $a[mid] = \text{data}$. Therefore, the search is completed, and the element is found successfully.

Q20. Write a C program to search an element using binary search.

Ans .:

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

```
int binarySearch(int[], int, int, int);
```

```
void main ()
```

```
{
```

```
    int arr[10] = {16, 19, 20, 23, 45, 56, 78, 90, 96, 100};
```

```
    int item, location=-1;
```

```
    printf("Enter the item which you want to search ");
```

```
    scanf("%d",&item);
```

```
    location = binarySearch(arr, 0, 9, item);
```

```
    if(location != -1)
```

```
    {
```

```
        printf("Item found at location %d",location);
```

```
    }
```

```
    else
```

```
    {
```

```
        printf("Item not found");
```

```
    }
```

```
}
```

```
int binarySearch(int a[], int beg, int end, int item)
```

```
{
```

```
    int mid;
```

```
    if(end >= beg)
```

```
    {
```

```
        mid = (beg + end)/2;
```

```
        if(a[mid] == item)
```

```
        {
```

```
            return mid+1;
```

```
        }
```

```
    else if(a[mid] < item)
```

```
    {
```

```

        return binarySearch (a,mid+1, end, item);
    }
    else
    {
        return binarySearch(a,beg,mid-1,item);
    }
}
return -1;
}

```

Output

Enter the item which you want to search

19

Item found at location 2

Q21. What are the differences between linear search and binary search.

Ans :

(Imp.)

Basis of comparison	Linear search	Binary search
Definition	The linear search starts searching from the first element and compares each element with a searched element till the element is not found.	It finds the position of the searched element by finding the middle element of the array.
Sorted data	In a linear search, the elements don't need to be arranged in sorted order.	The pre-condition for the binary search is that the elements must be arranged in a sorted order.
Implementation	The linear search can be implemented on any linear data structure such as an array, linked list, etc.	The implementation of binary search is limited as it can be implemented only on those data structures that have two-way traversal.
Approach	It is based on the sequential approach.	It is based on the divide and conquer approach.
Size	It is preferable for the small-sized data sets.	It is preferable for the large-size data sets.
Efficiency	It is less efficient in the case of large-size data sets.	It is more efficient in the case of large-size data sets.
Worst-case scenario	In a linear search, the worst- case scenario for finding the element is $O(n)$.	In a binary search, the worst- case scenario for finding the element is $O(\log_2 n)$.
Best-case scenario	In a linear search, the best-case scenario for finding the first element in the list is $O(1)$.	In a binary search, the best-case scenario for finding the first element in the list is $O(1)$.
Dimensional array	It can be implemented on both a single and multidimensional array.	It can be implemented only on a multidimensional array.

3.5 SELECTION SORT AND BINARY SORT

Q22. What is sorting ? List out basic sorting methods available in C.

Ans :

Arranging the data in ascending or descending order is known as **sorting**.

Sorting is very important from the point of view of our practical life.

The best example of sorting can be phone numbers in our phones. If, they are not maintained in an alphabetical order we would not be able to search any number effectively.

Sorting is the process of arranging the elements of an array so that they can be placed either in ascending or descending order. For example, consider an array $A = \{A_1, A_2, A_3, A_4, \dots, A_n\}$, the array is called to be in ascending order if element of A are arranged like $A_1 > A_2 > A_3 > A_4 > A_5 > \dots > A_n$.

Consider an array;

`int A[10] = { 5, 4, 10, 2, 30, 45, 34, 14, 18, 9 };`

The Array sorted in ascending order will be given as;

`A[] = { 2, 4, 5, 9, 10, 14, 18, 30, 34, 45 }`

Many methods are used for sorting, such as:

1. Bubble sort
2. Selection sort
3. Insertion sort
4. Quick sort
5. Merge sort
6. Heap sort
7. Radix sort
8. Shell sort

Generally a sort is classified as **internal** only if the data which is being sorted is in main memory.

It can be **external**, if the data is being sorted in the auxiliary storage.

The user will decide which sorting method can be used depending on the following conditions:

- a) Time required by a programmer for coding a particular sorting program.
- b) The machine time required for running the program.
- c) Space required by the program.

Q23. Write an algorithm to implement Selection sorting.

Ans :

(Imp.)

Selection Sorting

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array.

First, find the smallest element of the array and place it on the first position. Then, find the second smallest element of the array and place it on the second position. The process continues until we get the sorted array.

The array with n elements is sorted by using n-1 pass of selection sort algorithm.

- In 1st pass, smallest element of the array is to be found along with its index **pos**. then, swap $A[0]$ and $A[pos]$. Thus $A[0]$ is sorted, we now have n -1 elements which are to be sorted.
- In 2nd pass, position pos of the smallest element present in the sub-array $A[n-1]$ is found. Then, swap, $A[1]$ and $A[pos]$. Thus $A[0]$ and $A[1]$ are sorted, we now left with n-2 unsorted elements.
- In n-1th pass, position pos of the smaller element between $A[n-1]$ and $A[n-2]$ is to be found. Then, swap, $A[pos]$ and $A[n-1]$.

Therefore, by following the above explained process, the elements $A[0], A[1], A[2], \dots, A[n-1]$ are sorted.

Example

Consider the following array with 6 elements. Sort the elements of the array by using selection sort.

A = {10, 2, 3, 90, 43, 56}.

Pass	Pos	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
1	1	2	10	3	90	43	56
2	2	2	3	10	90	43	56
3	2	2	3	10	90	43	56
4	4	2	3	10	43	90	56
5	5	2	3	10	43	56	90

Sorted A = {2, 3, 10, 43, 56, 90}

Algorithm

SELECTION SORT(ARR, N)

- **Step 1:** Repeat Steps 2 and 3 for K = 1 to N-1
- **Step 2:** CALL SMALLEST(ARR, K, N, POS)
- **Step 3:** SWAP A[K] with ARR[POS]
- [END OF LOOP]
- **Step 4:** EXIT

SMALLEST (ARR, K, N, POS)

- **Step 1:** [INITIALIZE] SET SMALL = ARR[K]
- **Step 2:** [INITIALIZE] SET POS = K
- **Step 3:** Repeat for J = K+1 to N -1
- IF SMALL > ARR[J]
- SET SMALL = ARR[J]
- SET POS = J
- [END OF IF]
- [END OF LOOP]
- **Step 4:** RETURN POS

Q24. Write a C Program to implement selection sorting.

Ans :

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

```
int smallest(int[],int,int);
```

```
void main ()
```

```
{
```

```
    int a[10] = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
```

```
    int i,j,k,pos,temp;
```

```
    for(i=0;i<10;i++)
```

```

    {
        pos = smallest(a,10,i);
        temp = a[i];
        a[i]=a[pos];
        a[pos] = temp;
    }
    printf("\nprinting sorted elements...\n");
    for(i=0;i<10;i++)
    {
        printf("%d\n",a[i]);
    }
}

int smallest(int a[], int n, int i)
{
    int small,pos,j;
    small = a[i];
    pos = i;
    for(j=i+1;j<10;j++)
    {
        if(a[j]<small)
        {
            small = a[j];
            pos=j;
        }
    }
    return pos;
}

```

Output:

printing sorted elements...

7
9
10
12
23
23
34
44
78
101

Q25. Write an algorithm to implement bubble sorting.

Ans :

(Imp.)

Bubble Sorting

In Bubble sort, Each element of the array is compared with its adjacent element. The algorithm processes the list in passes. A list with n elements requires n-1 passes for sorting. Consider an array A of n elements whose elements are to be sorted by using Bubble sort. The algorithm processes like following.

1. In Pass 1, A[0] is compared with A[1], A[1] is compared with A[2], A[2] is compared with A[3] and so on. At the end of pass 1, the largest element of the list is placed at the highest index of the list.
2. In Pass 2, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of Pass 2 the second largest element of the list is placed at the second highest index of the list.
3. In pass n-1, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of this pass. The smallest element of the list is placed at the first index of the list.

Algorithm

- **Step 1:** Repeat Step 2 For i = 0 to N-1
- **Step 2:** Repeat For J = i + 1 to N - 1
- **Step 3:** IF A[J] > A[i]
SWAP A[J] and A[i]
[END OF INNER LOOP]
[END OF OUTER LOOP]
- **Step 4:** EXIT

C Program

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

```
void main ()
```

```
{
```

```
    int i, j,temp;
```

```
    int a[10] = { 10, 9, 7, 101, 23, 44,
12, 78, 34, 23};
```

```
for(i = 0; i<10; i++)
{
    for(j = i+1; j<10; j++)
    {
        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

Printing Sorted Element List . . .

7
9
10
12
23
34
34
44
78
101

Short Question and Answers

1. Discuss #ifdef preprocessor directive.

Ans :

The #ifdef preprocessor directive is used to check whether the macro-name is previously defined or not, if defined then the statements given between #ifdef and #else will get execute.

Syntax of #ifdef preprocessor directive

```
#ifdef macro-name
```

```
-----
```

```
-----
```

```
#else
```

```
-----
```

```
-----
```

```
#endif
```

Example of #ifdef preprocessor directive

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

```
#define MKS 65
```

```
void main()
```

```
{
```

```
    #ifdef MONTH
```

```
        printf("\nMONTH is defined.");
```

```
    #else
```

```
        printf("\nMONTH is not defined.");
```

```
    #endif
```

```
}
```

Output

```
MONTH is not defined.
```

2. What is #ifndef preprocessor directive?

Ans :

The #ifndef preprocessor directive is used to check whether the macro-name is previously defined or not, if not defined then the statements given between #ifndef and #else will get execute.

Syntax of #ifndef preprocessor directive

```
#ifndef macro-name
```

```
-----
```

```
-----
```

```
#else
```

```
-----
```

```
-----
```

```
#endif
```

Example of #ifndef preprocessor directive

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

```
#define MKS 65
```

```
void main()
```

```
{
```

```
    #ifndef MAX
```

```
        printf("\nMAX is not defined.");
```

```
    #else
```

```
        printf("\nMAX is defined.");
```

```
    #endif
```

```
}
```

Output

```
MAX is not defined.
```

3. List any four advantages of an Array.

Ans :

Advantages of Arrays

- i) **Code Optimization:** Less code to the access the data.
- ii) **Ease of traversing:** By using the for loop, we can retrieve the elements of an array easily.
- iii) **Ease of sorting:** To sort the elements of the array, we need a few lines of code only.
- iv) **Random Access:** We can access any element randomly using the array.

4. How memory allocation is done in arrays ?

Ans :

Memory Allocation of the array

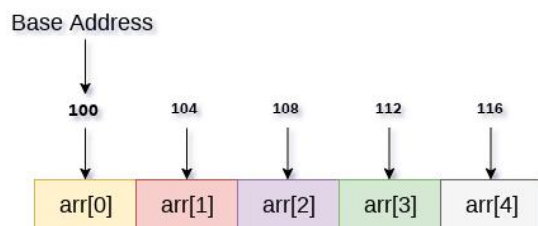
All the data elements of an array are stored at contiguous locations in the main memory. The

name of the array represents the base address or the address of first element in the main memory. Each element of the array is represented by a proper indexing.

The indexing of the array can be defined in three ways.

- i) **0 (zero - based indexing):** The first element of the array will be arr[0].
- ii) **1 (one - based indexing):** The first element of the array will be arr[1].
- iii) **n (n - based indexing):** The first element of the array can reside at any random index number.

In the following image, we have shown the memory allocation of an array arr of size 5. The array follows 0-based indexing approach. The base address of the array is 100th byte. This will be the address of arr[0]. Here, the size of int is 4 bytes therefore each element will take 4 bytes in the memory.



int arr[5]

In 0 based indexing, If the size of an array is n then the maximum index number, an element can have is n-1. However, it will be n if we use 1 based indexing.

5. Write a C program for sorting of an elements using array .

Ans:

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

```
void main ()
```

```
{
```

```
    int i, j,temp;
```

```
    int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
```

```
for(i = 0; i<10; i++)
{
    for(j = i+1; j<10; j++)
    {
        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

Printing sorted elements lis...

7
9
10
12
23
23
34
44
78
101

6. What is multi Dimensional Array.

Ans :

An array which is declared as more than one size is called as multi dimensional array. The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

7. What is linear search.*Ans :*

Linear search is the simplest search algorithm and often called sequential search. In this type of searching, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match found then location of the item is returned otherwise the algorithm return NULL.

A simple approach is to do a **linear search**, i.e

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
- If x matches with an element, return the index.
- If x doesn't match with any of elements, return -1.

8. Write a Binary search algorithm.*Ans :*

Binary search is the search technique which works efficiently on the sorted lists. Hence, in order to search an element into some list by using binary search technique, we must ensure that the list is sorted.

Binary search follows divide and conquer approach in which, the list is divided into two halves and the item is compared with the middle element of the list. If the match is found then, the location of middle element is returned otherwise, we search into either of the halves depending upon the result produced through the match.

Binary search algorithm is given below.

BINARY_SEARCH(A, lower_bound, upper_bound, VAL)

Step 1: [INITIALIZE] SET BEG = lower_bound
END = upper_bound, POS = - 1

Step 2: Repeat Steps 3 and 4 while BEG <=END

Step 3: SET MID = (BEG + END)/2

Step 4: IF A[MID] = VAL
SET POS = MID
PRINT POS
Go to Step 6
ELSE IF A[MID] > VAL
SET END = MID - 1
ELSE
SET BEG = MID + 1
[END OF IF]
[END OF LOOP]

Step 5: IF POS = -1
PRINT "VALUE IS NOT PRESENT IN THE ARRAY"
[END OF IF]

Step 6: EXIT

9. What are the differences between linear search and binary search.

Ans :

Basis of comparison	Linear search	Binary search
Definition	The linear search starts searching from the first element and compares each element with a searched element till the element is not found.	It finds the position of the searched element by finding the middle element of the array.
Sorted data	In a linear search, the elements don't need to be arranged in sorted order.	The pre-condition for the binary search is that the elements must be arranged in a sorted order.
Implementation	The linear search can be implemented on any linear data structure such as an array, linked list, etc.	The implementation of binary search is limited as it can be implemented only on those data structures that have two-way traversal.
Approach	It is based on the sequential approach.	It is based on the divide and conquer approach.
Size	It is preferable for the small-sized data sets.	It is preferable for the large-size data sets.
Efficiency	It is less efficient in the case of large-size data sets.	It is more efficient in the case of large-size data sets.
Worst-case scenario	In a linear search, the worst-case scenario for finding the element is $O(n)$.	In a binary search, the worst-case scenario for finding the element is $O(\log_2 n)$.
Best-case scenario	In a linear search, the best-case scenario for finding the first element in the list is $O(1)$.	In a binary search, the best-case scenario for finding the first element in the list is $O(1)$.
Dimensional array	It can be implemented on both a single and multidimensional array.	It can be implemented only on a multidimensional array.

10. What is selection sorting ?

Ans :

Selection Sorting

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array.

First, find the smallest element of the array and place it on the first position. Then, find the second smallest element of the array and place it on the second position. The process continues until we get the sorted array.

The array with n elements is sorted by using $n-1$ pass of selection sort algorithm.

- In 1st pass, smallest element of the array is to be found along with its index **pos**. then, swap $A[0]$ and $A[pos]$. Thus $A[0]$ is sorted, we now have $n-1$ elements which are to be sorted.
- In 2nd pass, position **pos** of the smallest element present in the sub-array $A[n-1]$ is found. Then, swap, $A[1]$ and $A[pos]$. Thus $A[0]$ and $A[1]$ are sorted, we now left with $n-2$ unsorted elements.

- In n-1th pass, position pos of the smaller element between A[n-1] and A[n-2] is to be found. Then, swap, A[pos] and A[n-1].

Therefore, by following the above explained process, the elements A[0], A[1], A[2],..., A[n-1] are sorted.

11. What is bubble sorting.

Ans :

Bubble Sorting

In Bubble sort, Each element of the array is compared with its adjacent element. The algorithm processes the list in passes. A list with n elements requires n-1 passes for sorting. Consider an array A of n elements whose elements are to be sorted by using Bubble sort. The algorithm processes like following.

1. In Pass 1, A[0] is compared with A[1], A[1] is compared with A[2], A[2] is compared with A[3] and so on. At the end of pass 1, the largest element of the list is placed at the highest index of the list.
2. In Pass 2, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of Pass 2 the second largest element of the list is placed at the second highest index of the list.
3. In pass n-1, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of this pass. The smallest element of the list is placed at the first index of the list.

Algorithm

- **Step 1:** Repeat Step 2 For i = 0 to N-1
- **Step 2:** Repeat For J = i + 1 to N - 1
- **Step 3:** IF A[J] > A[i]
SWAP A[J] and A[i]
[END OF INNER LOOP]
[END OF OUTER LOOP]
- **Step 4:** EXIT

Choose the Correct Answers

1. _____ is used to create symbolic constants in c. [a]
(a) # define (b) Constant
(c) Keyword (d) Variable
2. _____ is used to insert a specific header file into c program. [c]
(a) # undef (b) # if
(c) # include (d) # elif
3. _____ Pre processor is used to check whether the macro name is previously defined (or) not [b]
(a) # undef (b) # ifndef
(c) # include (d) None
4. _____ is the collection of similar data type elements. [b]
(a) Loop (b) Array
(c) Pointer (d) Storage
5. Arrays are used for _____. [d]
(a) Matrix (b) Operations
(c) Sorting Algorithms (d) All of the above
6. Array of Arrays are known as _____. [c]
(a) Structured array (b) 2D Array
(c) Multi dimensional Array (d) None
7. _____ is the process of finding some particular element in the list [a]
(a) Searching (b) Sorting
(c) Merging (d) None
8. Binary search follows _____ approach. [b]
(a) Merging (b) Divide and conquer
(c) Both (d) None
9. _____ Search is less efficient in the case of large size data sets. [a]
(a) Linear (b) Binary
(c) Simple (d) None
10. _____ Uses the value of specified macro for conditional compilation. [c]
(a) # else (b) # error
(c) # if (d) # undef

Fill in the blanks

1. _____ search is a sequential search.
2. _____ is the process of finding some particular element.
3. _____ search is preferable for the large-size data sets.
4. In _____ sorting, the smallest value among the unsorted elements.
5. In _____ sorting, each element of the array is compared with it's adjacent element.
6. _____ is used terminate processor conditional macro.
7. Each element of an array carries _____ size.
8. Single dimensional arrays used to represent _____ and _____ search algorithms.
9. _____ is the major disadvantages of array.
10. Array with in other array is known as _____.

ANSWERS

1. Linear
2. Searching
3. Binary search
4. Selection sort
5. Bubble sort
6. # endif
7. Same
8. Linear, Binary
9. Fixed size
10. Nested Array

One Mark Answers

1. **# define**

Ans :

define is used to create symbolic constants in c.

2. **Array**

Ans :

Array is the collection of similar datatype elements. Which are stored in contiguous format.

3. **What is searching?**

Ans :

Searching is the process of finding some particular element in the list.

4. **What is sorting?**

Ans :

Arranging the data in ascending (or) descending order is known as sorting.

5. **Binary Search**

Ans :

It finds the position of the search element by finding the middle element of the array.

UNIT IV

Pointers - Introduction, Pointers for Inter-Function Communication, Pointers to Pointers, Compatibility, L-value and R-value, Arrays and Pointers, Pointer Arithmetic and Arrays, Passing an Array to a Function, Memory Allocation Functions, Array of Pointers, Programming Applications, Pointers to void, Pointers to Functions, Command-line Arguments.

Strings - Concepts, C Strings, String Input/Output Functions, Arrays of Strings, String Manipulation Functions.

4.1 POINTERS

4.1.1 Introduction

Q1. Define pointer? Explain the process of declaring a pointer in c program.

Ans :

(Imp.)

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

```
int n = 10;
```

```
int* p = &n; // Variable p of type pointer is pointing to the address of the variable  
n of type integer.
```

Declaring a pointer

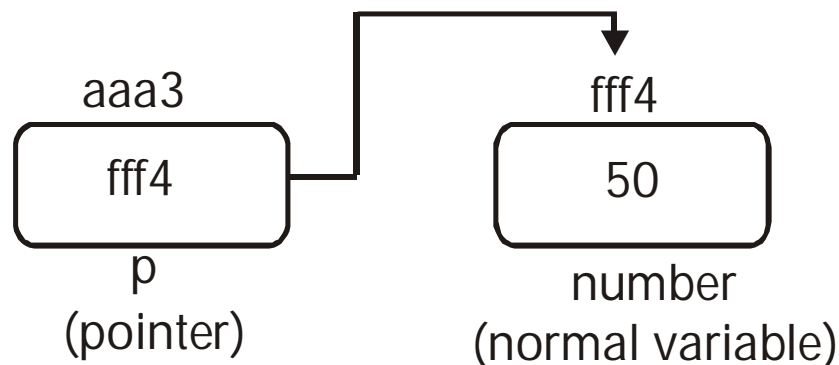
The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

```
int *a;//pointer to int
```

```
char *c;//pointer to char
```

Example

An example of using pointers to print the address and value is given below.



As you can see in the above figure, pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of * (indirection operator), we can print the value of pointer variable p.

Let's see the pointer example as explained for the above figure.

```
#include<stdio.h>

int main(){
    int number=50;
    int *p;

    p=&number;//stores the address of
    number variable
    printf("Address of p variable is %x \n",p);

    // p contains the address of the number
    therefore printing p gives the address of
    number.

    printf("Value of p variable is %d \n",*p);

    // As we know that *is used to
    dereference a pointer therefore if we print *p, we
    will get the value stored at the address
    contained by p.

    return 0;
}
```

Output

Address of number variable is fff4

Address of p variable is fff4

Value of p variable is 50

Q2. What are the advantages and usages of pointers ?

Ans:

Advantages of pointer

- 1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.
- 2) We can return multiple values from a function using the pointer.

- 3) It makes you able to access any memory location in the computer's memory.

Usage of pointer

There are many applications of pointers in c language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where the pointer is used.

2) Arrays, Functions, and Structures

Pointers in c language are widely used in arrays, functions, and structures. It reduces the code and improves the performance.

Q3. Write a c program for swapping of two numbers using pointers ?

Ans:

```
#include<stdio.h>
int main(){
    int a=10,b=20,*p1=&a,*p2=&b;
    printf("Before swap: *p1=%d *p2=%d",*p1,*p2);

    *p1=*p1+*p2;
    *p2=*p1-*p2;
    *p1=*p1-*p2;

    printf("\nAfter swap: *p1=%d *p2=%d",*p1,*p2);

    return 0;
}
```

Output

Before swap: *p1=10 *p2=20

After swap: *p1=20 *p2=10

Q4. What are the uses of

1. Address of operator
2. Null pointer

Ans:

Address of (&) Operator

The address of operator '&' returns the address of a variable. But, we need to use %u to display the address of a variable.

```
#include <stdio.h>

int main(){
    int number=50;
    printf("value of number is %d, address
    of number is %u",number,&number);
    return 0;
}
```

Output

value of number is 50, address of number is fff4

NULL Pointer

A pointer that is not assigned any value but NULL is known as the NULL pointer. If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value. It will provide a better approach.

```
int *p=NULL;
```

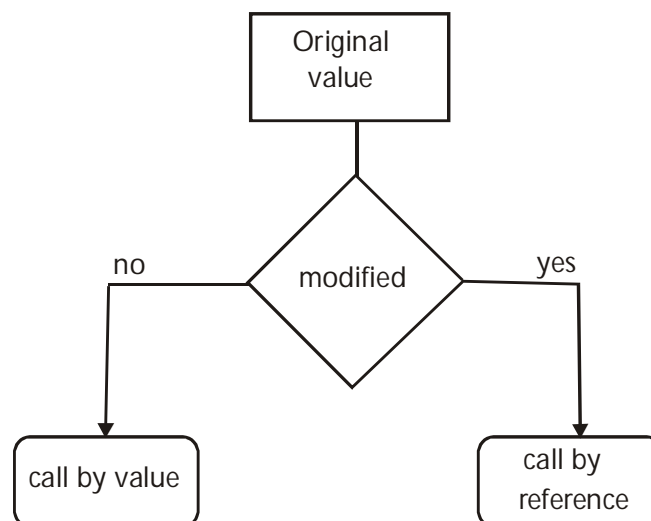
4.1.2 Pointers for Inter-Function Communication

Q5. Explain the pointer for inter function communication in C.

Ans. :

We know that functions can be called by value and called by reference.

- If the actual parameter should not change in called function, pass the parameter-by value.
- If the value of actual parameter should get changed in called function, then use pass-by reference.
- If the function has to return more than one value, return these values indirectly by using call-by-reference.



(i) Call by value in C

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In call by value method, we can not modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Example:

```
#include <stdio.h>

void change(int num) {
    printf("Before adding value inside function num=%d \n", num);
    num=num+100;
    printf("After adding value inside function num=%d \n", num);
}

int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(x); //passing value in function
    printf("After function call x=%d \n", x);
    return 0;
}
```

Output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100
```

(ii) Call by reference in C

- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Example

```
#include<stdio.h>
void change(int *num) {
printf("Before adding value inside function num=%d \n",*num);
(*num) += 100;
printf("After adding value inside function num=%d \n", *num);
}
int main() {
int x=100;
printf("Before function call x=%d \n", x);
change(&x);//passing reference in function
printf("After function call x=%d \n", x);
return 0;
}
```

Output

Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=200

No	Call by value	Call by reference
1	A copy of the value is passed into the function the function	An address of value is passed into the function.
2.	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3.	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

4.1.3 Pointers to pointers

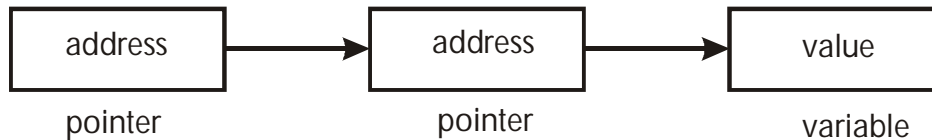
Q6. What is double pointer ? Explain with an example program ?

Ans :

C Double Pointer (Pointer to Pointer)

A pointer is used to store the address of a variable in C. Pointer reduces the access time of a variable. However, In C, we can also define a pointer to store the address of another pointer. Such pointer is known as a double pointer (pointer to pointer). The first pointer is used to store the address of

a variable whereas the second pointer is used to store the address of the first pointer. Let's understand it by the diagram given below.



The syntax of declaring a double pointer is given below.

```
int **p;
```

Example.

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

```
void main ()
```

```
{
```

```
int a = 10;
```

```
int *p;
```

```
int **pp;
```

```
p = &a; // pointer p is pointing to the address of a
```

```
pp = &p; // pointer pp is a double pointer pointing to the address of pointer p
```

```
printf("address of a: %x\n",p); // Address of a will be printed
```

```
printf("address of p: %x\n",pp); // Address of p will be printed
```

```
printf("value stored at p: %d\n",*p); // value stored at the address contained  
by p i.e. 10 will be printed
```

```
printf("value stored at pp: %d\n",**pp); // value stored at the address contained  
by the pointer stored at pp
```

```
}
```

Output

```
address of a: d26a8734
```

```
address of p: d26a8738
```

```
value stored at p: 10
```

```
value stored at pp: 10
```

4.1.4 Compatibility

Q7. What is compatibility of pointers?

Ans:

Two pointer types with the same type qualifiers are compatible if they point to objects of compatible types. The composite type for two compatible pointer types is the similarly qualified pointer to the composite type.

The following example shows compatible declarations for the assignment operation:

```
float subtotal;
float * sub_ptr;
/* ... */
sub_ptr = &subtotal;
printf("The subtotal is %f\n", *sub_ptr);
The next example shows incompatible
declarations for the assignment operation:
double league;
int * minor;
/* ... */
minor = &league;
/* error */
```

4.1.5 L-Value and R-Value

Q8. How are L-Values and R-Values defined in C?

Ans :

L-value: "l-value" refers to memory location which identifies an object. l-value may appear as either left hand or right hand side of an assignment operator(=). l-value often represents as identifier.

Expressions referring to modifiable locations are called "modifiable l-values". A modifiable l-value cannot have an array type, an incomplete type, or a type with the `const` attribute. For structures and unions to be modifiable l-values, they must not have any members with the `const` attribute. The name of the identifier denotes a storage location, while the value of the variable is the value stored at that location.

An identifier is a modifiable lvalue if it refers to a memory location and if its type is arithmetic, structure, union, or pointer. For example, if `ptr` is a pointer to a storage region, then `*ptr` is a modifiable l-value that designates the storage region to which `ptr` points.

In C, the concept was renamed as "locator value", and referred to expressions that locate (designate) objects. The l-value is one of the following:

1. The name of the variable of any type i.e., an identifier of integral, floating, pointer, structure, or union type.

2. A subscript (`[]`) expression that does not evaluate to an array.
3. A unary-indirection (`*`) expression that does not refer to an array
4. An l-value expression in parentheses.
5. A `const` object (a nonmodifiable l-value).
6. The result of indirection through a pointer, provided that it isn't a function pointer.
7. The result of member access through pointer (`->` or `.`)

Example:

```
inti = 10;
```

But this is not:

```
inti;
```

```
10 = i;
```

R-value: r-value" refers to data value that is stored at some address in memory. A r-value is an expression that can't have a value assigned to it which means r-value can appear on right but not on left hand side of an assignment operator(=).

4.1.6 Arrays and Pointers

Q9. How can we use arrays using pointers?

Ans :

(Imp.)

In the c programming language, when we declare an array the compiler allocate the required amount of memory and also creates a constant pointer with array name and stores the base address of that pointer in it. The address of the first element of an array is called as base address of that array.

The array name itself acts as a pointer to the first element of that array. Consider the following example of array declaration...

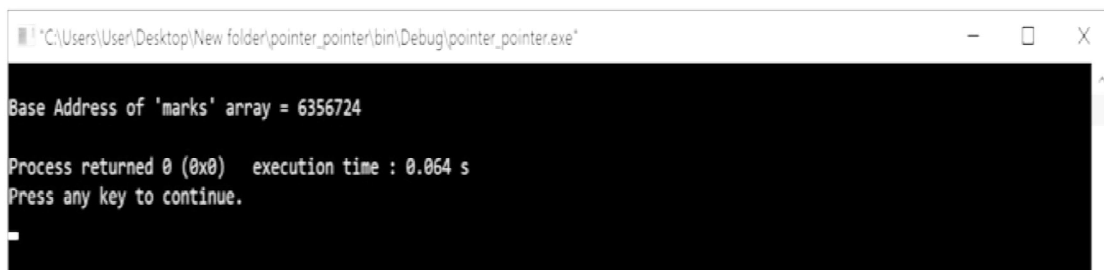
Example

```
int marks[6];
```

For the above declaration, the compiler allocates 12 bytes of memory and the address of first memory location (i.e., `marks[0]`) is stored in a constant pointer called `marks`. That means in the above example, `marks` is a pointer to `marks[0]`.

Example Program

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int marks[6] = {89, 45, 58, 72, 90, 93} ;
    int *ptr ;
    clrscr() ;
    ptr = marks ;
    printf("Base Address of 'marks' array = %u\n", ptr) ;
    return 0;
}
```

Output:**Pointers to Multi Dimensional Array**

In case of multi dimensional array also the array name acts as a constant pointer to the base address of that array. For example, we declare an array as follows...

Example Code

```
int marks[3][3] ;
```

In the above example declaration, the array name `marks` acts as constant pointer to the base address (address of `marks[0][0]`) of that array.

In the above example of two dimensional array, the element `marks[1][2]` is accessed as `*(*(marks + 1) + 2)`.

4.1.7 Pointer Arithmetic and arrays**Q10. How can we perform arithmetic operations using pointers ?**

Ans:

We can perform arithmetic operations on the pointers like addition, subtraction, etc. However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer. In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:

- (i) Increment
- (ii) Decrement
- (iii) Addition
- (iv) Subtraction

(i) Incrementing Pointer in C

If we increment a pointer by 1, the pointer will start pointing to the immediate next location. This is somewhat different from the general arithmetic since the value of the pointer will get increased by the size of the data type to which the pointer is pointing.

We can traverse an array by using the increment operation on a pointer which will keep pointing to every element of the array, perform some operation on that, and update itself in a loop.

The Rule to increment the pointer is given below:

`new_address = current_address + i * size_of(data type)`

Where i is the number by which the pointer get increased.

32-bit

For 32-bit int variable, it will be incremented by 2 bytes.

64-bit

For 64-bit int variable, it will be incremented by 4 bytes.

Let's see the example of incrementing pointer variable on 64-bit architecture.

```
#include<stdio.h>

int main(){
    int number=50;
    int *p;//pointer to int
    p=&number;//stores the address of number variable
    printf("Address of p variable is %u \n",p);
    p=p+1;
    printf("After increment: Address of p variable is %u \n",p); //
    // in our case, p will get incremented by 4 bytes.
    return 0;
}
```

Output

Address of p variable is 3214864300

After increment: Address of p variable is 3214864304

(ii) Decrementing Pointer in C

Like increment, we can decrement a pointer variable. If we decrement a pointer, it will start pointing to the previous location. The formula of decrementing the pointer is given below:

`new_address = current_address - i * size_of(data type)`

32-bit

For 32-bit int variable, it will be decremented by 2 bytes.

64-bit

For 64-bit int variable, it will be decremented by 4 bytes.

Let's see the example of decrementing pointer variable on 64-bit OS.

```
#include <stdio.h>
void main(){
int number=50;
int *p;//pointer to int
p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p-1;
printf("After decrement: Address of p variable is %u \n",p); //P will now point
to the immediate previous location.
}
```

Output

Address of p variable is 3214864300

After decrement: Address of p variable is 3214864296

(iii) C Pointer Addition

We can add a value to the pointer variable. The formula of adding value to pointer is given below:

$\text{new_address} = \text{current_address} + (\text{number} * \text{size_of}(\text{data type}))$

32-bit

For 32-bit int variable, it will add 2 * number.

64-bit

For 64-bit int variable, it will add 4 * number.

Let's see the example of adding value to pointer variable on 64-bit architecture.

```
#include<stdio.h>
int main(){
int number=50;
int *p;//pointer to int
p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p+3; //adding 3 to pointer variable
printf("After adding 3: Address of p variable is %u \n",p);
return 0;
```



```
}
```

Output

Address of p variable is 3214864300

After adding 3: Address of p variable is 3214864312

As you can see, the address of p is 3214864300. But after adding 3 with p variable, it is 3214864312, i.e., $4 \times 3 = 12$ increment. Since we are using 64-bit architecture, it increments 12. But if we were using 32-bit architecture, it was incrementing to 6 only, i.e., $2 \times 3 = 6$. As integer value occupies 2-byte memory in 32-bit OS.

(iv) C Pointer Subtraction

Like pointer addition, we can subtract a value from the pointer variable. Subtracting any number from a pointer will give an address. The formula of subtracting value from the pointer variable is given below:

$$\text{new_address} = \text{current_address} - (\text{number} * \text{size_of}(\text{data type}))$$

32-bit

For 32-bit int variable, it will subtract $2 * \text{number}$.

64-bit

For 64-bit int variable, it will subtract $4 * \text{number}$.

Let's see the example of subtracting value from the pointer variable on 64-bit architecture.

```
#include <stdio.h>

int main(){
    int number=50;
    int *p;//pointer to int
    p=&number;//stores the address of number variable
    printf("Address of p variable is %u \n",p);
    p=p-3; //subtracting 3 from pointer variable
    printf("After subtracting 3: Address of p variable is %u \n",p);
    return 0;
}
```

Output

Address of p variable is 3214864300

After subtracting 3: Address of p variable is 3214864288

You can see after subtracting 3 from the pointer variable, it is 12 (4×3) less than the previous address value.

4.1.8 Passing an Array to a Function

Q11. How can we pass an array to a function

Ans.:

Pointer to Array of functions in C

To understand the concept of an array of functions, we must understand the array of function. Basically, an array of the function is an array which contains the addresses of functions. In other words, the pointer to an array of functions is a pointer pointing to an array which contains the pointers to the functions. Consider the following example.

```
#include<stdio.h>

int show();
int showadd(int);
int (*arr[3])();
int (*ptr)[3]();
int main ()
{
    int result1;
    arr[0] = show;
    arr[1] = showadd;
    ptr = &arr;
    result1 = (**ptr)();
    printf("printing the value returned by
    show : %d",result1);
    ((*ptr+1))(result1);
}

int show()
{
    int a = 65;
    return a++;
}

int showadd(int b)
{
    printf("\nAdding 90 to the value returned
    by show: %d",b+90);
}
```

Output

Printing the value returned by show : 65

Adding 90 to the value returned by show:
155

4.1.9 Memory Allocation Functions

Q12. What is Dynamic Memory allocation? List and explain standard defined functions used for dynamic memory allocation.

Ans.:

(Imp.)

In C programming language, when we declare variables memory is allocated in space called stack. The memory allocated in the stack is fixed at the time of compilation and remains until the end of the program execution. When we create an array, we must specify the size at the time of the declaration itself and it can not be changed during the program execution. This is a major problem when we do not know the number of values to be stored in an array. To solve this we use the concept of Dynamic Memory Allocation. The dynamic memory allocation allocates memory from heap storage. Dynamic memory allocation is defined as follow...

Allocation of memory during the program execution is called dynamic memory allocation. (or) Dynamic memory allocation is the process of allocating the memory manually at the time of program execution.

We use pre-defined or standard library functions to allocate memory dynamically. There are FOUR standard library functions that are defined in the header file known as "stdlib.h". They are as follows...

1. malloc()
2. calloc()
3. realloc()
4. free()

1. malloc()

malloc() is the standard library function used to allocate a memory block of specified number of bytes and returns void pointer. The void pointer can be casted to any datatype. If malloc() function unable to allocate memory due to any reason it returns NULL pointer.

Syntax

```
void* malloc(size_in_bytes)
```

Example

```
#include<stdio.h>
#include<conio.h>
int main () {
char *title;
title = (char *) malloc(15);
strcpy(title, "c programming");
printf("String = %s, Address = %u\n", title,
title);
return(0);
}
```

2. calloc()

calloc() is the standard library function used to allocate multiple memory blocks of the specified number of bytes and initializes them to ZERO. calloc() function returns void pointer. If calloc() function unable to allocate memory due to any reason it returns a NULL pointer. Generally, calloc() is used to allocate memory for array and structure. calloc() function takes two arguments and they are 1. The number of blocks to be allocated, 2. Size of each block in bytes

Syntax

```
void*calloc(number_of_blocks,
size_of_each_block_in_bytes)
```

Example

```
#include<stdio.h>
#include<conio.h>
int main () {
inti, n;
int *ptr;
printf("Number of blocks to be created:");
scanf("%d",&n);
ptr = (int*)calloc(n, sizeof(int));
printf("Enter %d numbers:\n",n);
```

```
for(i=0 ; i< n ; i++ ) {
scanf("%d",&ptr[i]);
}
printf("The numbers entered are: ");
for(i=0 ; i< n ; i++ ) {
printf("%d ",ptr[i]);
}
return(0);
}
```

3. realloc()

realloc() is the standard library function used to modify the size of memory blocks that were previously allocated using malloc() or calloc(). realloc() function returns void pointer. If calloc() function unable to allocate memory due to any reason it returns NULL pointer.

Syntax

```
void*realloc(*pointer,
new_size_of_each_block_in_bytes)
```

Example

```
#include<stdio.h>
#include<conio.h>
int main () {
char *title;
title = (char *) malloc(15);
strcpy(title, "c programming");
printf("Before modification : String
= %s, Address = %u\n", title, title);
title = (char*) realloc(title, 30);
strcpy(title,"C Programming Language");
printf("After modification : String = %s,
Address = %u\n", title, title);
return(0);
}
```

4. free()

free() is the standard library function used to deallocate memory block that was previously allocated using malloc() or calloc(). free() function returns void pointer. When free() function is used with memory allocated that was created using calloc(), all the blocks are get deallocated.

Syntax

```
void free(*pointer)
```

Example

```
#include <stdio.h>
#include <conio.h>
int main () {
    char *title;
    title = (char *) malloc(15);
    strcpy(title, "c programming");
    printf("Before modification : String
    = %s, Address = %u\n", title, title);
    title = (char*) realloc(title, 30);
    strcpy(title, "C Programming Language");
    printf("After modification : String = %s,
    Address = %u\n", title, title);
    free(title);
    return(0);
}
```

4.1.10 Array of Pointers

Q13. What is the use of Array of Pointers in C?

Ans:

(Imp.)

In computer programming, an array of pointers is an indexed set of variables, where the variables are pointers (referencing a location in memory).

Pointers are an important tool in computer science for creating, using, and destroying all types of data structures. An array of pointers is useful for the same reason that all arrays are useful: it allows you to numerically index a large set of variables.

Below is an array of pointers in C that points each pointer in one array to an integer in another array. The value of each integer is printed by dereferencing the pointers. In other words, this code prints the value in memory of where the pointers point.

```
#include <stdio.h>
constint ARRAY_SIZE = 5;
int main ()
{
    intarray_of_integers[] = {5, 10, 20, 40, 80};
    inti, *array_of_pointers[ARRAY_SIZE];
    for ( i = 0; i< ARRAY_SIZE; i++)
    {
        array_of_pointers[i] = &array_of_integers[i];
    }
    for ( i = 0; i< ARRAY_SIZE; i++)
    {
        printf("array_of_integers[%d]
        = %d\n", i, *array_of_pointers[i] );
    }
    return 0;
}
```

The output of the above program is:

```
array_of_integers[0] = 5
array_of_integers[1] = 10
array_of_integers[2] = 20
array_of_integers[3] = 40
array_of_integers[4] = 80
```

4.1.11 Programming Application

Q14. What are the applications and uses of pointers.

Ans:

Pointers are considered to be useful tools in programming because of the following reasons:

- (i) Pointers make the programs simple and reduce their length.
- (ii) Pointers are helpful in allocation and deallocation of memory during the execution of the program. Thus, pointers are the instruments of dynamic memory management.
- (iii) Pointers enhance the execution speed of a program.
- (iv) Pointers are helpful in traversing through arrays and character strings. The strings are also arrays of characters terminated by the null character ('\0').
- (v) Pointers also act as references to different types of objects such as variables, arrays, functions, structures, etc. However, C language does not have the concept of references as in C++. Therefore, in C we use pointer as a reference.
- (vi) Storage of strings through pointers saves memory space.
- (vii) Pointers may be used to pass on arrays, strings, functions, and variables as arguments of a function.
- (viii) Passing on arrays by pointers saves lot of memory because we are passing on only the address of array instead of all the elements of an array, which would mean passing on copies of all the elements and thus taking lot of memory space.
- (ix) Pointers are used to construct different data structures such as linked lists, queues, stacks, etc.

4.1.12 Pointers to void

Q15. What is the use of void pointer in C.

Ans :

The void pointer in C is a pointer which is not associated with any data types. It points to some data location in the storage means points to the address of variables. It is also called general purpose pointer. In C, malloc() and calloc() functions return void * or generic pointers.

Till now, we have studied that the address assigned to a pointer should be of the same type as specified in the pointer declaration.

For example, if we declare the int pointer, then this int pointer cannot point to the float variable or some other type of variable, i.e., it can point to only int type variable. To overcome this problem, we use a pointer to void. A pointer to void means a generic pointer that can point to any data type. We can assign the address of any data type to the void pointer, and a void pointer can be assigned to any type of the pointer without performing any explicit typecasting.

Syntax of void pointer

```
void *pointer name;
```

It has some limitations "

- 1) Pointer arithmetic is not possible with void pointer due to its concrete size.
- 2) It can't be used as dereferenced.

Example:

```
#include <stdlib.h>
int main() {
    int a = 7;
    float b = 7.6;
    void *p;
    p = &a;
    printf("Integer variable is = %d", *((int*) p));
    p = &b;
    printf("\nFloat variable is = %f", *((float*) p));
    return 0;
}
```

Output

Integer variable is = 7

Float variable is = 7.600000

Size of the void pointer in C

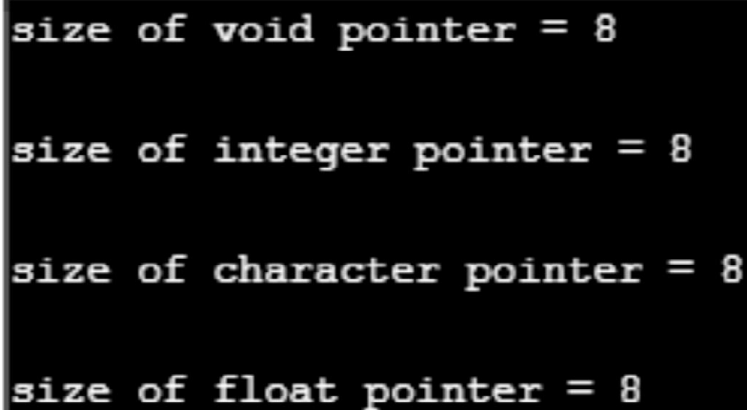
The size of the void pointer in C is the same as the size of the pointer of character type. According to C perception, the representation of a pointer to void is the same as the pointer of character type. The size of the pointer will vary depending on the platform that you are using.

Example:

```
#include <stdio.h>

int main()
{
    void *ptr = NULL; //void pointer
    int *p = NULL; // integer pointer
    char *cp = NULL; //character pointer
    float *fp = NULL; //float pointer

    //size of void pointer
    printf("size of void pointer = %d\n\n", sizeof(ptr));
    //size of integer pointer
    printf("size of integer pointer = %d\n\n", sizeof(p));
    //size of character pointer
    printf("size of character pointer = %d\n\n", sizeof(cp));
    //size of float pointer
    printf("size of float pointer = %d\n\n", sizeof(fp));
    return 0;
}
```

Output

```
size of void pointer = 8

size of integer pointer = 8

size of character pointer = 8

size of float pointer = 8
```

4.1.13 Pointers to Functions

Q16. Explain the concept of Pointers to Functions.*Ans.:* (Imp.)

Pointer as a function parameter is used to hold addresses of arguments passed during function call. This is also known as call by reference. When a function is called by reference any change made to the reference variable will effect the original variable.

Example:

Swapping two numbers using Pointer

```
#include <stdio.h>

void swap(int *a, int *b);

int main()
{
    int m = 10, n = 20;
    printf("m = %d\n", m);
    printf("n = %d\n\n", n);
    swap(&m, &n); //passing address of m and
    n to the swap function
    printf("After Swapping:\n\n");
    printf("m = %d\n", m);
    printf("n = %d", n);
    return 0;
}

/*
pointer 'a' and 'b' holds and
points to the address of 'm' and 'n'
*/

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

}

Output :

m = 10

n = 20

After Swapping:

m = 20

n = 10

Functions returning Pointer variables

A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function. They have scope only inside the function. Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.

```
#include <stdio.h>

int* larger(int*, int*);

void main()
{
    int a = 15;
    int b = 92;
    int *p;
    p = larger(&a, &b);
    printf("%d is larger", *p);
}

int* larger(int *x, int *y)
{
    if(*x > *y)
        return x;
    else
        return y;
}

Copy
92 is larger
```

Pointer to functions

It is possible to declare a pointer pointing to a function which can then be used as an argument in another function. A pointer to a function is declared as follows,

```
type(*pointer-name)(parameter);
```

Copy

Here is an example :

```
int(*sum)(); // legal declaration of pointer to function
```

```
int*sum(); // This is not a declaration of pointer to function
```

Copy

A function pointer can point to a specific function when it is assigned the name of that function.

```
intsum(int,int);
```

```
int(*s)(int,int);
```

```
s = sum;
```

Copy

Here *s* is a pointer to a function *sum*. Now *sum* can be called using function pointer *s* along with providing the required argument values.

```
s(10,20);
```

Copy

Example of Pointer to Function

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

```
intsum(int x,int y)
```

```
{
```

```
return x+y;
```

```
}
```

```
intmain()
```

```
{
```

```
int(*fp)(int,int);
```

```
fp= sum;
```

```
int s =fp(10,15);
```

```
printf("Sum is %d", s);
```

```
return0;
```

```
}
```

Copy

25

4.1.14 Command line arguments**Q17. Why do we use command line arguments in C?**

Ans:

(Imp.)

Command line argument is a parameter supplied to the program when it is invoked. Command line argument is an important concept in C programming. It is mostly used when you need to control your program from outside. Command line arguments are passed to the `main()` method.

Syntax

```
int main(intargc, char *argv[])
```

Here *argc* counts the number of arguments on the command line and *argv[]* is a pointer array which holds pointers of type *char* which points to the arguments passed to the program.

Example:

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

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

```
int main(intargc, char *argv[])
```

```
{
```

```
inti;
```

```
if(argc>= 2 )
```

```
{
```

```
printf("The arguments supplied are:\n");
```

```
for(i = 1; i<argc; i++)
```

```
{
```

```
printf("%s\t", argv[i]);
```

```
}
```

```
}
```

```
else
```

```
{
```

```
printf("argument list is empty.\n");
```

```
}
```

```
return 0;
```

```
}
```


Remember that `argv[0]` holds the name of the program and `argv[1]` points to the first command line argument and `argv[n]` gives the last argument. If no argument is supplied, `argc` will be 1.

4.2 STRINGS

4.2.1 Concepts, C Strings

Q18. Define a String.

Ans :

The string can be defined as the one-dimensional array of characters terminated by a null (`'\0'`). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character (`'\0'`) is important in a string since it is the only way to identify where the string ends. When we define a string as `char s[10]`, the character `s[10]` is implicitly initialized with the null in the memory.

There are two ways to declare a string in c language.

1. By char array
2. By string literal

Example

```
char ch[10]={'r','a','m','u','\0'};
```

We can also define the string by the string literal in C language. For example:

```
char ch[]="ramu";
```

There are two main differences between char array and literal.

- We need to add the null character `'\0'` at the end of the array by ourself whereas, it is appended internally by the compiler in the case of the character array.
- The string literal cannot be reassigned to another set of characters whereas, we can reassign the characters of the array.

Q. Write a sample code for String implementation.

Ans :

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

int main()
{
    char ch[11]={'r','a','m','u','\0'};
    char ch2[11]="ramu";

    printf("Char Array Value is: %s\n", ch);
    printf("String Literal Value is: %s\n", ch2);

    return 0;
}
```

Output

```
Char Array Value is: ramu
String Literal Value is: ramu
```

4.2.2 String Input/Output Functions

Q19. What are string Input Output functions?

Ans : (Imp.)

Input means to provide the program with some data to be used in the program and Output means to display data on the screen or write the data to a printer or a file.

The C programming language provides standard library functions to read any given input and to display data on the console.

The following are the input and output functions of strings in c

Input functions: `scanf()`, `gets()`

Output functions: `printf()`, `puts()`

The `scanf()` and `printf()` are generic i/o functions that they support all built-in data types such as `int`, `float`, `long`, `double`, `strings`,...etc. But `gets()` and `puts()` are specialized to scan and print only string data.

There is a little difference between `scanf()` and `gets()`, while reading string from keyboard, the `scanf()` accepts character by character from keyboard until either a new line (`\n`) or blank space is found, which ever comes earlier.

Whereas `gets()` accepts until a newline is found. That is it accepts white spaces & tab also, these input functions append a null character at end of string, the formatted string `%s` is used in `printf()` and `scanf()`.for example :

Scanning string with “scanf()”	Scanning string with “gets()”
<pre>void main() { char a[20]; printf("enter a string:"); scanf("%s", a); //scanf("%s", &a[0]); printf("\n output=%s", a); }</pre>	<pre>void main() { char a[20]; printf("enter a string:"); gets(a); printf("\n output=%s", a); }</pre>
<p>Input: Jack and Jill // <code>scanf</code> treats as 3 strings Output: Jack</p>	<p>Input: Jack and Jill // <code>gets()</code> treats as single string Output: Jack an Jill</p>

The `scanf()` function consider the jack and Jill as 3 strings, whereas `gets()` considers as single string. In case to scan total string using `scanf()`, then it should be `scanf("%s%s%s", a,b,c);` here a,b,c are three arrays.

The `printf()` and `puts()` is work in similar way. All I/O functions take first byte address (base address of array) as argument and prints the given string using pointer.

Program : The following program is an example of string I/O functions.

```
#include <stdio.h>
#include <string.h>
int main()
{
char name[30];
printf("Enter name: ");
gets(name); //Function to read string from user.
printf("Name: ");
puts(name); //Function to display string.
return 0;
}
```

Output :

Enter name: ramu

Name: ramu

4.2.3 Arrays of Strings

Q20. What is Array of String ?

Ans :

(Imp.)

The string is a collection of characters, an array of a string is an array of arrays of characters. Each string is terminated with a null character. An array of a string is one of the most common applications of two-dimensional arrays.

scanf() is the input function with %s format specifier to read a string as input from the terminal. But the drawback is it terminates as soon as it encounters the space. To avoid this gets() function which can read any number of strings including white spaces.

String is an array of characters terminated with the special character known as the null character ("0").

Syntax

```
datatype name_of_the_array[size_of_elements_in_array]; char str_name[size];
```

Example

```
datatype name_of_the_array [ ] = { Elements of array };
```

```
char str_name[8] = "Strings";
```

Str_name is the string name and the size defines the length of the string (number of characters).

A String can be defined as a one-dimensional array of characters, so an array of strings is two-dimensional array of characters.

Example:

```
#include <stdio.h>
int main()
{
    char name[10];
    printf("Enter the name: ");
    fgets(name, sizeof(name), stdin);
    printf("Name is : ");
    puts(name);
    return 0;
}
```

Output:

Enter the name:Ramu

Name is : Ramu

Now for two-dimensional arrays, we have the following syntax and memory allocation. For this, we can take it as row and column representation (table format).

```
char str_name[size][max];
```

In this table representation, each row (first subscript) defines as the number of strings to be stored and column (second subscript) defines the maximum length of the strings.

```
charstr_arr[2][6] = { {'g','o','u','r','i','\0'}, {'r','a','m','\0'} };
```

Alternatively, we can even declare it as

Syntax:

```
charstr_arr[2][8] = {"gouri", "ram"};
```

Index	0	1	2	3	4	5	6	7
Rows								
0	g	o	u	r	i	\0	\0	\0
1	r	a	m	\0	\0	\0	\0	\0

From the above example as we know that the name of the array points to the 0th string. Therefore, str_name + 0 points to 0th string "gouri"
str_name + 1 points to 1st string "ram"

As the above example is for two-dimensional arrays so the pointer points to each string of the array.

Example:

```
#include <stdio.h>
int main()
{
    inti;
    char name[2][8] = {
        "gouri",
        "ram"
    };
    for (i = 0; i < 2; i++)
    {
        printf("String = %s \n", name + i, name + i);
    }
    return 0;
}
```

Output

String= gowri

String= ram

4.2.4 String Manipulation Functions

Q21. List out various string manipulation functions in C.

Ans :

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string, second_string)	concatenates or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5)	strrev(string)	returns reverse string.
6)	strlwr(string)	returns string characters in lowercase.
7)	strupr(string)	returns string characters in uppercase.

Q22. Explain the following functions in detail.

(a) strlen()

(b) strcpy()

(c) strcat()

(d) strcmp()

Ans :

(Imp.)

(a) strlen()

The strlen() function returns the length of the given string. It doesn't count null character '\0'.

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

```
#include <string.h>
```

```
int main(){
```

```
char ch[20]={ 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
```

```
printf("Length of string is: %d",strlen(ch));
```

```
return 0;
```

```
}
```

Output

Length of string is: 10

(b) strcpy()

The strcpy(destination, source) function copies the source string in destination.

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

```
#include <string.h>
```

```
int main(){
char ch[20]={ 'j', 'a', 'v', 'a', 't', 'p',
'o', 'i', 'n', 't', '\0' };
char ch2[20];
strcpy(ch2,ch);
printf("Value of second string is: %s",ch2);
return 0;
}
```

Output

Value of second string is: C Language

(c) strcat()

The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.

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

```
#include <string.h>
```

```
int main(){
    char ch[10]={ 'h', 'e', 'l', 'l', 'o', '\0' };
    char ch2[10]={ 'c', '\0' };
    strcat(ch,ch2);
    printf("Value of first string is: %s",ch);
    return 0;
}
```

Output:

Value of first string is: helloc

(d) strcmp()

The strcmp(first_string, second_string) function compares two string and returns 0 if both strings are equal.

Here, we are using gets() function which reads string from the console.

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

```
#include <string.h>
```

```
int main()
{
char str1[20],str2[20];
```

```
printf("Enter 1st string: ");
gets(str1);//reads string from console
printf("Enter 2nd string: ");
gets(str2);
if(strcmp(str1,str2)==0)
printf("Strings are equal");
else
printf("Strings are not equal");
return 0;
}
```

Output

Enter 1st string: hello

Enter 2nd string: hello

Strings are equal

Q23. Explain the following string functions with an example.

(a) strrev() (b) strlwr() (c)strupr()

Ans :

(a) strrev()

The strrev(string) function returns reverse of the given string. Let's see a simple example of strrev() function.

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

```
#include <string.h>
```

```
int main()
{
char str[20];
printf("Enter string: ");
gets(str);//reads string from console
printf("String is: %s",str);

printf("\nReverse String is: %s",strrev(str));
return 0;
}
```

Output

Enter string: RAM
String is: RAM
Reverse String is: MAR

(b) strlwr()

The strlwr(string) function returns string characters in lowercase. Let's see a simple example of strlwr() function.

```
#include<stdio.h>
#include <string.h>
int main(){
char str[20];
printf("Enter string: ");
gets(str);//reads string from console
printf("String is: %s",str);
printf("\nLower String is: %s",strlwr(str));
return 0;
}
```

Output

Enter string: RAKesh
String is: RAKesh
Lower String is: rakesh

(c) strupr()

The strupr(string) function returns string characters in uppercase. Let's see a simple example of strupr() function.

```
#include<stdio.h>
#include <string.h>
int main()
{
char str[20];
printf("Enter string: ");
gets(str);//reads string from console
printf("String is: %s",str);
printf("\nUpper String is: %s",strupr(str));
return 0;
}
```

Output

Enter string: rakesh
String is: rakesh
Upper String is: RAKESH

Short Question & Answers

1. Define pointer

Ans :

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

```
int n = 10;
```

```
int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.
```

Declaring a pointer

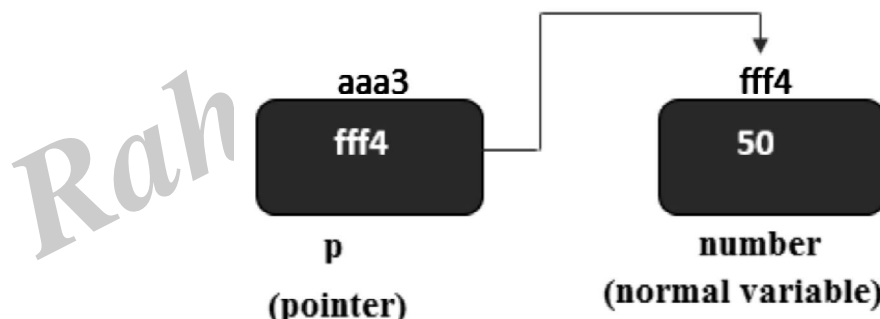
The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

```
int *a;//pointer to int
```

```
char *c;//pointer to char
```

Example

An example of using pointers to print the address and value is given below.



2. What are the advantages and usages of pointers ?

Ans:

Advantages of pointer

- 1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.
- 2) We can return multiple values from a function using the pointer.
- 3) It makes you able to access any memory location in the computer's memory.

Usage of pointer

There are many applications of pointers in c language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where the pointer is used.

2) Arrays, Functions, and Structures

Pointers in c language are widely used in arrays, functions, and structures. It reduces the code and improves the performance.

3. What is Null Pointer.

Ans:

A pointer that is not assigned any value but NULL is known as the NULL pointer. If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value. It will provide a better approach.

```
int *p=NULL;
```

4. What are the differences between call-by-value and call-by- reference.

Ans:

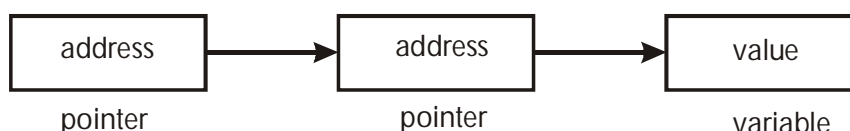
No	Call by value	Call by reference
1	A copy of the value is passed into the function the function	An address of value is passed into the function.
2.	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3.	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

5. What is Double pointer.

Ans:

C Double Pointer (Pointer to Pointer)

A pointer is used to store the address of a variable in C. Pointer reduces the access time of a variable. However, In C, we can also define a pointer to store the address of another pointer. Such pointer is known as a double pointer (pointer to pointer). The first pointer is used to store the address of a variable whereas the second pointer is used to store the address of the first pointer. Let's understand it by the diagram given below.



The syntax of declaring a double pointer is given below.

```
int **p;
```

6. What is Compatibility of pointers.*Ans:*

Two pointer types with the same type qualifiers are compatible if they point to objects of compatible types. The composite type for two compatible pointer types is the similarly qualified pointer to the composite type.

The following example shows compatible declarations for the assignment operation:

```
float subtotal;
float * sub_ptr;
/* ... */
sub_ptr = &subtotal;
printf("The subtotal is %f\n", *sub_ptr);
```

The next example shows incompatible declarations for the assignment operation:

```
double league;
int * minor;
/* ... */
minor = &league;
/* error */
```

7. What is Dynamic Memory allocation?*Ans:*

In C programming language, when we declare variables memory is allocated in space called stack. The memory allocated in the stack is fixed at the time of compilation and remains until the end of the program execution. When we create an array, we must specify the size at the time of the declaration itself and it can not be changed during the program execution. This is a major problem when we do not know the number of values to be stored in an array. To solve this we use the concept of Dynamic Memory Allocation. The dynamic memory allocation allocates memory from heap storage. Dynamic memory allocation is defined as follow...

Allocation of memory during the program execution is called dynamic memory allocation. (or) Dynamic memory allocation is the process of allocating the memory manually at the time of program execution.

We use pre-defined or standard library functions to allocate memory dynamically. There are FOUR standard library functions that are defined in the header file known as "stdlib.h". They are as follows...

1. malloc()
2. calloc()
3. realloc()
4. free()

8. What is the use of realloc() function?*Ans:*

realloc() is the standard library function used to modify the size of memory blocks that were previously allocated using malloc() or calloc(). realloc() function returns void pointer. If calloc() function unable to allocate memory due to any reason it returns NULL pointer.

Syntax

```
void*realloc(*pointer,
new_size_of_each_block_in_bytes)
```

9. What are the applications and uses of pointers.*Ans:*

Pointers are considered to be useful tools in programming because of the following reasons:

- (i) Pointers make the programs simple and reduce their length.
- (ii) Pointers are helpful in allocation and de-allocation of memory during the execution of the program. Thus, pointers are the instruments of dynamic memory management.
- (iii) Pointers enhance the execution speed of a program.
- (iv) Pointers are helpful in traversing through arrays and character strings. The strings are also arrays of characters terminated by the null character ('\0').
- (v) Pointers also act as references to different types of objects such as variables, arrays, functions, structures, etc. However, C language does not

have the concept of references as in C++. Therefore, in C we use pointer as a reference.

- (vi) Storage of strings through pointers saves memory space.
- (vii) Pointers may be used to pass on arrays, strings, functions, and variables as arguments of a function.
- (viii) Passing on arrays by pointers saves lot of memory because we are passing on only the address of array instead of all the elements of an array, which would mean passing on copies of all the elements and thus taking lot of memory space.
- (ix) Pointers are used to construct different data structures such as linked lists, queues, stacks, etc.

10. Why do we use command line arguments in C?

Ans:

Command line argument is a parameter supplied to the program when it is invoked. Command line argument is an important concept in C programming. It is mostly used when you need to control your program from outside. Command line arguments are passed to the `main()` method.

Syntax

```
int main(int argc, char *argv[])
```

Here `argc` counts the number of arguments on the command line and `argv[]` is a pointer array which holds pointers of type `char` which points to the arguments passed to the program.

Example:

```
#include <stdio.h>
#include <conio.h>
int main(int argc, char *argv[])
{
    inti;
    if(argc >= 2 )
    {
        printf("The arguments supplied are:\n");
```

```
for(i = 1; i < argc; i++)
{
    printf("%s\t", argv[i]);
}
}
else
{
    printf("argument list is empty.\n");
}
return 0;
}
```

Remember that `argv[0]` holds the name of the program and `argv[1]` points to the first command line argument and `argv[n]` gives the last argument. If no argument is supplied, `argc` will be 1.

11. Define a String.

Ans :

The string can be defined as the one-dimensional array of characters terminated by a null (`'\0'`). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character (`'\0'`) is important in a string since it is the only way to identify where the string ends. When we define a string as `char s[10]`, the character `s[10]` is implicitly initialized with the null in the memory.

There are two ways to declare a string in c language.

1. By char array
2. By string literal

Example

```
char ch[10]={'r','a','m','u','\0'};
```

We can also define the string by the string literal in C language. For example:

```
char ch[]="ramu";
```

12. List out various string manipulation functions in C.

Ans :

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string, second_string)	concatenates or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5)	strrev(string)	returns reverse string.
6)	strlwr(string)	returns string characters in lowercase.
7)	strupr(string)	returns string characters in uppercase.

Choose the Correct Answers

1. _____ is used to create symbolic constants. [a]
(a) # define (b) Constant
(c) Keyword (d) Variable
2. _____ is used to insert a specific header file into C program. [c]
(a) # undef (b) # if
(c) # include (d) # elif
3. _____ pre processor is used to check whether the macro name is previously defined (or) not? [c]
(a) # undef (b) # ifndef
(c) # include (d) # elif
4. _____ is the collection of similar data type elements. [b]
(a) Loop (b) Array
(c) Pointer (d) Storage
5. Arrays are used for _____. [d]
(a) Matrix operations (b) Searching algorithms
(c) Sorting algorithms (d) All of the above
6. Array of Arrays are known as _____. [c]
(a) Structured array (b) 2D Array
(c) Multi dimensional array (d) None
7. _____ is the process of finding some particular element in the list. [a]
(a) Searching (b) Sorting
(c) Merging (d) None
8. Binary search follows _____ approach. [b]
(a) Merging (b) Divide and conquer
(c) Both (d) None

9. _____ search is less efficient in the case of large size data sets. [a]
- (a) Linear (b) Binary
- (c) Simple (d) None
10. _____ uses the value of specified macro for conditional compilation [c]
- (a) # else (b) # error
- (c) # is (d) # undef

Rahul Publications

Fill in the blanks

1. _____ search is a sequential search.
2. _____ is the process of finding some particular element.
3. _____ search is preferable for the large size data sets.
4. In _____ sorting, the smallest value among the unsorted elements.
5. In _____ sorting, each element of the array is compared with its adjacent element.
6. _____ is used to terminate processor conditional macro.
7. Each element of an array carries _____ size.
8. Single dimensional arrays are used to represent _____ and _____ search algorithms.
9. _____ is the major disadvantage of array.
10. Array within another array is known as _____.

ANSWERS

1. Linear
2. Searching
3. Binary search
4. Selection sort
5. Bubble sort
6. # endif
7. Same
8. Linear, Binary
9. Fixed size
10. Nested array

One Mark Answers

1. **# define.**

Ans :

define is used to create symbolic constants in c.

2. **Array**

Ans :

Array is the collection of similar data type elements. Which are stored in contiguous format.

3. **What is searching?**

Ans :

Searching is the process of finding some particular element in the list.

4. **What is sorting?**

Ans :

Arranging the data in ascending (or) descending order is known as sorting.

5. **Binary search.**

Ans :

It finds the position of the search element by finding the middle element of the array.

UNIT V

Structures: Definition and Initialization of Structures, Accessing Structures, Nested Structures, Arrays of Structures, Structures and Functions, Pointers to Structures, Self Referential Structures, Unions, Type Definition (typedef), Enumerated Types.

Input and Output: Introduction to Files, Modes of Files, Streams, Standard Library Input/Output Functions, Character Input/Output Functions.

5.1 STRUCTURES - DEFINITION, INITIALIZATION OF STRUCTURE, ACCESSING STRUCTURE

Q1. Why we use structures?

Ans :

In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types. For example, an entity Student may have its name (string), roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:

- Construct individual arrays for storing names, roll numbers, and marks.
- Use a special data structure to store the collection of different data types.

Let's look at the first approach in detail.

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

```
void main ()
```

```
{
```

```
    char names[2][10], dummy; // 2-dimensionaal character array names is used to store the names of the students
```

```
    int roll_numbers[2], i;
```

```
    float marks[2];
```

```
    for (i=0; i<3; i++)
```

```
    {
```

```
        printf("Enter the name, roll number, and marks of the student %d", i+1);
```

```
        scanf("%s %d %f", &names[i], &roll_numbers[i], &marks[i]);
```

```
        scanf("%c", &dummy); // enter will be stored into dummy character at each iteration
```

```
    }
```

```
    printf("Printing the Student details ...\n");
```

```
    for (i=0; i<3; i++)
```

```
    {
```

```
        printf("%s %d %f\n", names[i], roll_numbers[i], marks[i]);
```

```
    }
```

```
}
```

Output

```
Enter the name, roll number, and marks of the student 1Arun 90 91
Enter the name, roll number, and marks of the student 2Varun 91 56
Enter the name, roll number, and marks of the student 3Sham 89 69
Printing the Student details...
    Arun 90 91.000000
    Varun 91 56.000000
    Sham 89 69.000000
```

The above program may fulfill our requirement of storing the information of an entity student. However, the program is very complex, and the complexity increase with the amount of the input. The elements of each of the array are stored contiguously, but all the arrays may not be stored contiguously in the memory. C provides you with an additional and simpler approach where you can use a special data structure, i.e., structure, in which, you can group all the information of different data type regarding an entity.

Q2. Define Structure ?**(OR)****What is structure ?****(OR)****How can we declare structures?***Ans :***(Imp.)**

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures simulate the use of classes and templates as it can store various information

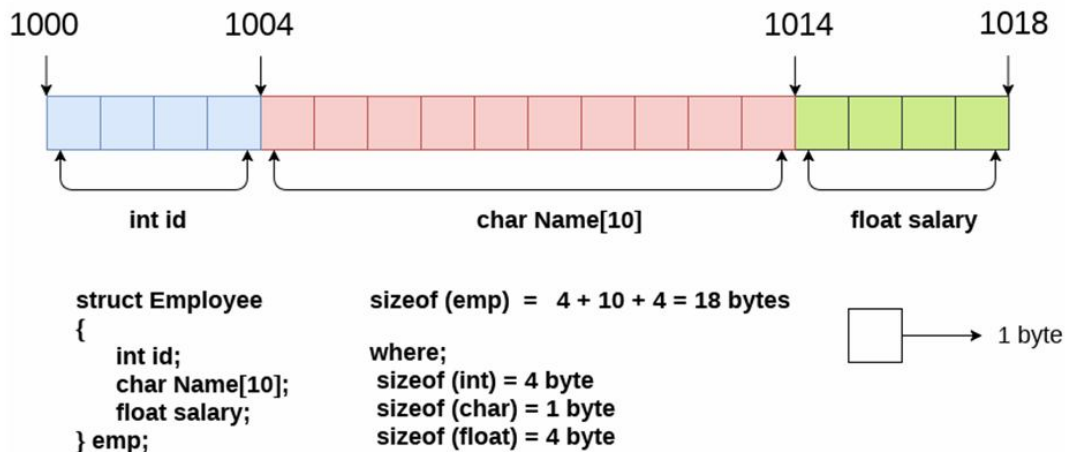
The **struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memberN;
};
```

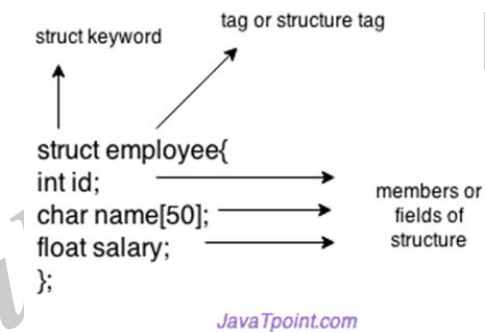
Let's see the example to define a structure for an entity employee in c.

```
struct employee
{
    int id;
    char name[20];
    float salary;
};
```

The following image shows the memory allocation of the structure employee that is defined in the above example.



Here, struct is the keyword; employee is the name of the structure; id, name, and salary are the members or fields of the structure. Let's understand it by the diagram given below:



Declaring structure variable

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

1st way

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

struct employee

{

int id;

char name[50];

float salary;

};

Now write given code inside the main() function.
struct employee e1, e2;

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

2nd way

Let's see another way to declare variable at the time of defining the structure.

struct employee

```
{
int id;
    char name[50];
    float salary;
}e1,e2;
```

Q3. Explain the process of defining a structure and accessing members of structures with a c program ?

Ans :

Defining a Structure

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows:

```
struct[structure tag]
{
    member definition;
    member definition;
    ...
    member definition;
}[one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as `int i;` or `float f;` or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure:

```
structBooks
{
    char title[50];
    char author[50];
    char subject[100];
    intbook_id;
} book;
```

Accessing Structure Members

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type. The following example shows how to use a structure in a program:

```
#include<stdio.h>
#include<string.h>
structBooks
{
    char title[50];
    char author[50];
    char subject[100];
    intbook_id;
};
int main()
{
    structBooksBook1;
        /* Declare Book1 of type Book */
    structBooksBook2;
        /* Declare Book2 of type Book */
        /* book 1 specification */
    strcpy(Book1.title,"C Programming");
    strcpy(Book1.author,"Nuha Ali");
    strcpy(Book1.subject,"C Programming Tutorial");
    Book1.book_id =6495407;
```

```
/* book 2 specification */
strcpy(Book2.title,"Telecom Billing");
strcpy(Book2.author,"Zara Ali");
strcpy(Book2.subject,"Telecom Billing Tutorial");
Book2.book_id =6495700; /* print Book1 info */
printf("Book 1 title : %s\n",Book1.title);
printf("Book 1 author : %s\n",Book1.author);
printf("Book 1 subject : %s\n",Book1.subject);
printf("Book 1 book_id : %d\n",Book1.book_id); /* print Book2 info */
printf("Book 2 title : %s\n",Book2.title);
printf("Book 2 author : %s\n",Book2.author);
printf("Book 2 subject : %s\n",Book2.subject);
printf("Book 2 book_id : %d\n",Book2.book_id);

return0;
}
```

Output

Book 1 title : C Programming

Book 1 author :Nuha Ali

Book 1 subject : C Programming Tutorial

Book 1 book_id : 6495407

Book 2 title : Telecom Billing

Book 2 author : Zara Ali

Book 2 subject : Telecom Billing Tutorial

Book 2 book_id : 6495700

5.1.1 Nested structures

Q4. How can we define a structure with in other structure ?

(OR)

What are nested structures explain in detail ?

Ans :

(Imp.)

When a structure contains another structure, it is called nested structure. For example, we have two structures named Address and Employee. To make Address nested to Employee, we have to define Address structure before and outside Employee structure and create an object of Address structure inside Employee structure.

Syntax for structure within structure or nested structure

```
struct structure1
```

```
{
```

```
-----
```

```
-----
```

```
};
```

```
struct structure2
```

```
{
```

```
-----
```

```
-----
```

```
struct structure1 obj;
```

```
};
```

Example for structure within structure or nested structure

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

```
struct Address
```

```
{
```

```
char HouseNo[25];
```

```
char City[25];
```

```
char PinCode[25];
```

```
};
```

```
struct Employee
```

```
{
```

```
int Id;
```

```
char Name[25];
```

```
float Salary;
```

```
struct Address Add;
```

```
};
```

```
void main()
```

```
{
```

```
inti;
```

```
struct Employee E;
```

```
printf("\nEnter Employee Id : ");
```

```
scanf("%d",&E.Id);
```

```
printf("\nEnter Employee Name : ");
```

```
scanf("%s",&E.Name);
```

```
printf("\nEnter Employee Salary : ");
```

```
scanf("%f",&E.Salary);
```

```
printf("\nEnter Employee House No : ");
```

```
scanf("%s",&E.Add.HouseNo);
```

```
printf("\nEnter Employee City : ");
```

```
scanf("%s",&E.Add.City);
```

```
printf("\nEnter Employee House No : ");
```

```
scanf("%s",&E.Add.PinCode);
```

```
printf("\nDetails of Employees");
```

```
printf("\nEnter Employee Id : %d",E.Id);
```

```
printf("\nEnter Employee Name : %s",E.Name);
```

```
printf("\nEnter Employee Salary : %f",E.Salary);
```

```
printf("\nEnter Employee House No : %s",E.Add.HouseNo);
```

```
printf("\nEnter Employee City : %s",E.Add.City);
```

```
printf("\nEnter Employee House No : %s",E.Add.PinCode);
```

```
}
```

Output

```
Enter Employee Id : 101
```

```
Enter Employee Name : Suresh
```

```
Enter Employee Salary : 45000
```

```
Enter Employee House No : 4598/D
```

```
Enter Employee City : Delhi
```

```
Enter Employee Pin Code : 110056
```

Details of Employees

```
Employee Id : 101
```

```
Employee Name : Suresh
```

```
Employee Salary : 45000
```

```
Employee House No : 4598/D
```

```
Employee City : Delhi
```

```
Employee Pin Code : 110056
```

Q5. Explain structure with In the structure with the context of normal variables and pointer variables.

Ans :

- Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.
- The structure variables can be a normal structure variable or a pointer variable to access the data. You can learn below concepts in this section.

1. Structure within structure in C using normal variable
2. Structure within structure in C using pointer variable

1. STRUCTURE WITHIN STRUCTURE IN C USING NORMAL VARIABLE:

- This program explains how to use structure within structure in C using normal variable. "student_college_detail" structure is declared inside "student_detail" structure in this program. Both structure variables are normal structure variables.
- Please note that members of "student_college_detail" structure are accessed by 2 dot(.) operator and members of "student_detail" structure are accessed by single dot(.) operator.

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

```
#include <string.h>
```

```
struct student_college_detail
```

```
{
```

```
    int college_id;
```

```
    char college_name[50];
```

```
};
```

```
struct student_detail
```

```
{
```

```
    int id;
```

```
    char name[20];
```

```
    float percentage;
```

```
// structure within structure
```

```
struct student_college_detail clg_data;
```

```
} stu_data;
```

```
int main()
```

```
{
```

```
    struct student_detail stu_data = {1, "Raju", 90.5, 71145, "Anna University"};
```

```
    printf(" Id is: %d \n", stu_data.id);
```

```
    printf(" Name is: %s \n", stu_data.name);
```

```
    printf(" Percentage is: %f \n\n", stu_data.percentage);
```

```

printf(" College Id is: %d \n",
      stu_data.clg_data.college_id);
printf(" College Name is: %s \n",
      stu_data.clg_data.college_name);
return 0;
}

```

OUTPUT

```

Id is: 1
Name is: Raju
Percentage is: 90.500000
College Id is: 71145
College Name is: Anna University

```

STRUCTURE WITHIN STRUCTURE (NESTED STRUCTURE IN C) USING POINTER VARIABLE

- This program explains how to use structure within structure in C using pointer variable. "student_college_detail" structure is declared inside "student_detail" structure in this program. one normal structure variable and one pointer structure variable is used in this program.
- Please note that combination of .(dot) and ->(arrow) operators are used to access the structure member which is declared inside the structure.

```

#include <stdio.h>
#include <string.h>

struct student_college_detail
{
    int college_id;
    char college_name[50];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
}

stu_data, *stu_data_ptr;

int main()

```



```

{
    struct student_detail stu_data = {1, "Raju", 90.5, 71145, "Anna University"};
    stu_data_ptr = &stu_data;
    printf(" Id is: %d \n", stu_data_ptr->id);
    printf(" Name is: %s \n", stu_data_ptr->name);
    printf(" Percentage is: %f \n\n", stu_data_ptr->percentage);
    printf(" College Id is: %d \n", stu_data_ptr->clg_data.college_id);
    printf(" College Name is: %s \n", stu_data_ptr->clg_data.college_name);

    return 0;
}

```

OUTPUT

Id is: 1

Name is: Raju

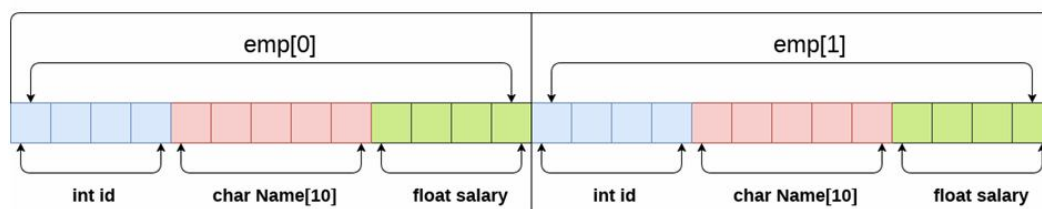
Percentage is: 90.500000

College Id is: 71145

College Name is: Anna University

5.1.2 Array of Structures**Q6. Explain the concept of array of structures.***Ans :***(Imp.)**

An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

Array of structures

```

struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];

```

`sizeof (emp) = 4 + 5 + 4 = 13 bytes``sizeof (emp[2]) = 26 bytes`

Let's see an example of an array of structures that stores information of 5 students and prints it.

```
#include <stdio.h>
#include <string.h>
struct student{
    int rollno;
    char name[10];
};
int main(){
    int i;
    struct student st[5];
    printf("Enter Records of 5 students");
    for(i=0;i<5;i++){
        printf("\nEnter Rollno:");
        scanf("%d",&st[i].rollno);
        printf("\nEnter Name:");
        scanf("%s",&st[i].name);
    }
    printf("\nStudent Information List:");
    for(i=0;i<5;i++){
        printf("\nRollno:%d, Name: %s", st[i].rollno,st[i].name);
    }
    return 0;
}
```

Output:

```
Enter Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
Enter Rollno:2
Enter Name:Ratan
Enter Rollno:3
Enter Name:Vimal
Enter Rollno:4
Enter Name:James
```

```
Enter Rollno:5
```

```
Enter Name:Sarfraz
```

```
Student Information List:
```

```
Rollno:1, Name:Sonoo
```

```
Rollno:2, Name:Ratan
```

```
Rollno:3, Name:Vimal
```

```
Rollno:4, Name:James
```

```
Rollno:5, Name:Sarfraz
```

5.1.3 Structures and Functions

Q7. How can we pass structure members to Functions?

Ans :

C allows programmers to pass a single or entire structure information to or from a function. A structure information can be passed as a function arguments. The structure variable may be passed as a value or reference. The function will return the value by using the return statement.

Example

```
#include <stdio.h>
int add(int, int) ; //function declaration
int main()
{
    //structures declartion
    struct addition{
        int a, b;
        int c;
    }sum;
    printf("Enter the value of a : ");
    scanf("%d",&sum.a);
    printf("\nEnter the value of b : ");
    scanf("%d",&sum.b);
    sum.c = add(sum.a, sum.b); //passing structure members as arguments to function
    printf("\nThe sum of two value are : ");
    printf("%d ", sum.c);
```

```

return 0;
}
//Function definition
int add(int x, int y)
{
int sum1;
sum1 = x + y;
return(sum1);
}

```

Output

Enter the value of a 10
Enter the value of b 20
The sum of two values 30

Q8. Write a C Program to pass the entire Structure to Functions.*Ans :***(Imp.)**

```

#include <stdio.h>
//structures declaration
typedef struct {
int a, b;
int c;
}sum;
void add(sum) ;
//function declaration with struct type sum
int main()
{
sum s1;
printf("Enter the value of a : ");
scanf("%d",&s1.a);
printf("\nEnter the value of b : ");
scanf("%d",&s1.b);
add(s1);
//passing entire structure as an argument to
function
return 0;
}

```

```

//Function Definition
void add(sum s)
{
int sum1;
sum1 = s.a + s.b;
printf("\nThe sum of two values are :%d", sum1);
}

```

Output

Enter the value of a 10
Enter the value of b 20
The sum of two values 30

Q9. Discuss possible ways to passing a structure to a function ?*Ans :*

- A structure can be passed to any function from main function or from any sub function.
- Structure definition will be available within the function only.
- It won't be available to other functions unless it is passed to those functions by value or by address (reference).
- Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.

It can be done in below 3 ways.

1. Passing structure to a function by value
2. Passing structure to a function by address (reference)
3. No need to pass a structure – Declare structure variable as global.

Passing Structure to Function in C by Value

In this program, the whole structure is passed to another function by value. It means the whole structure is passed to another function with all members and their values. So, this structure can be accessed from called function. This concept is very useful while writing very big programs in C.

```
#include <stdio.h>
#include <string.h>
struct student
{
    int id;
    char name[20];
    float percentage;
};
void func(struct student record);
int main()
{
    struct student record;
    record.id = 1;
    strcpy(record.name, "Raju");
    record.percentage = 86.5;
    func(record);
    return 0;
}
void func(struct student record)
{
    printf(" Id is: %d \n", record.id);
    printf(" Name is: %s \n", record.name);
    printf(" Percentage is: %f \n", record.percentage);
}
```

Output

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Passing Structure to Function in C by Address

In this program, the whole structure is passed to another function by address. It means only the address of the structure is passed to another function. The whole structure is not passed to another function with all members and their values. So, this structure can be accessed from called function by its address.

```
#include <stdio.h>
#include <string.h>
struct student
{
    int id;
    char name[20];
    float percentage;
};
void func(struct student* record);
int main()
{
    struct student record;
    record.id = 1;
    strcpy(record.name, "Raju");
    record.percentage = 86.5;
    func(&record);
    return 0;
}
void func(struct student* record)
{
    printf(" Id is: %d \n", record->id);
    printf(" Name is: %s \n", record->name);
    printf(" Percentage is: %f \n", record->percentage);
}
```

Output

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

A Structure Variable as Global in C

Structure variables also can be declared as global variables as we declare other variables in C. So, When a structure variable is declared as global, then it is visible to all the functions in a program. In this scenario, we don't need to pass the structure to any function separately.

```
#include <stdio.h>
#include <string.h>
struct student
{
    int id;
    char name[20];
    float percentage;
};
struct student record; // Global declaration of structure
void structure_demo();
int main()
{
    record.id = 1;
    strcpy(record.name, "Raju");
    record.percentage = 86.5;
    structure_demo();
    return 0;
}
void structure_demo()
{
    printf(" Id is: %d \n", record.id);
    printf(" Name is: %s \n", record.name);
    printf(" Percentage is: %f \n", record.percentage);
}
```

Output

Id is: 1

Name is: Raju

Percentage is: 86.500000

5.1.4 Pointers to Structures**Q10. How can we define pointers to structures ?***Ans :*

We can define pointers to structures in the same way as you define pointer to any other variable:

```
struct Books *struct_pointer;
```

Now, you can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the '&'; operator before the structure's name as follows"

```
struct_pointer = &Book1;
```

To access the members of a structure using a pointer to that structure, you must use the '!' operator as follows:

```
struct_pointer->title;
```

Let us re-write the above example using structure pointer.

```
#include <stdio.h>
#include <string.h>
struct Books{
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};
/* function declaration */
void printBook(struct Books* book );
int main(){
    struct Books Book1;
    /* Declare Book1 of type Book */
    struct Books Book2;
    /* Declare Book2 of type Book */
    /* book 1 specification */
    strcpy(Book1.title, "C Programming");
    strcpy(Book1.author, "Nuha Ali");
    strcpy(Book1.subject, "C Programming Tutorial");
    Book1.book_id = 6495407;
    /* book 2 specification */
    strcpy(Book2.title, "Telecom Billing");
    strcpy(Book2.author, "Zara Ali");
    strcpy(Book2.subject, "Telecom Billing Tutorial");
    Book2.book_id = 6495700;
```

```

    /* print Book1 info by passing address of
    Book1 */
        printBook(&Book1);
/* print Book2 info by passing address of Book2 */
        printBook(&Book2);
return 0;
}

void printBook(struct Books* book ){
    printf("Book title : %s\n", book->title);
    printf("Book author : %s\n", book->author);
    printf("Book subject : %s\n", book->subject);
    printf("Book book_id : %d\n", book-
        >book_id);
}

```

Output

```

Book title : C Programming
Book author :Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700

```

5.1.5 Self Referential Structures

Q11. What are self-referential structures?
Write the syntax of self-referential structures.

Ans :

(Imp.)

A self-referential structure is a structure that can have members which point to a structure variable of the same type. They can have one or more pointers pointing to the same type of structure as their member. The self-referential structure is widely used in dynamic data structures such as trees, linked lists, and so on. The next node of a node will be pointed in linked lists, which consists of the same struct type. It is a unique type of structure containing a member of its type. The member of its type is a

pointer variable of the same structure in which it has been declared. In the context of blockchain, each block is linked to a previous node or a next node, similar to a linked list.

Syntax

```

struct structure_name
{
    datatype datatype_name;
    structure_name * pointer_name;
}

```

Example

```

struct node
{
    int data;
    struct node *next;
};

```

Where 'next' is a pointer to a struct node variable, it should be remembered that the pointer to the structure is similar to the pointer to any other variable. Hence, a self-referential data structure is generally a structure definition that includes at least one member that is a pointer to the structure of its kind.

The self-referential structures are beneficial in many applications that involves linked data members, such as trees and lists. Unlike a static data structure such as an array where the size of the array limits the number of elements that can be inserted into the array, the self-referential structure can dynamically be contracted or expanded. Operations such as insertion or deletion of nodes in a self-referential structure involve simple alteration of the pointers present within them.

Example

```

typedef struct list
{
    void *data;
    struct list *next;
} linked_list;

```

Here, in the above example, the list is self-referential structure as the *next is the type struct list.

```

struct node
{
    int data;
    char value;
    struct node * link;
};

int main()
{
    struct node object;
    return 0;
}

```

In this particular example, 'link' is a pointer to f type 'node' structure. Hence, the structure 'node' is a self-referential structure with 'link' as a referencing pointer. The pointer should be appropriately initialized before accessing, as by default, it contains a garbage value.

Types of self-referential structures

1. Self-referential structure with a single link: these structures are allowed to have only one self-pointer as their member.

Example

```

#include <stdio.h>
#include <conio.h>
struct ref
{
    int data;
    charval;
    struct ref* link;
}

int main()
{
    struct ref object1; //link1
    object1.link = NULL;
    object1.data = 10;
    object1.val = 20;
    struct ref object2; //

```

```

    object2.link = NULL;
    object2.data = 30;
    object2.val = 40;
    object1.link = &object2;
    printf ("%d \n", object1.link -> data);
    printf ("%d \n", object1.link -> val);
    return 0;
}

```

Output:

30 40

2. **Self-referential structure with multiple links:** these types of structures 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.

Example

```

#include <stdio.h>
#include <conio.h>
struct ref
{
    int data;
    struct ref* previous;
    struct ref* next;
};

int main()
{
    struct node object1;
    object1.previous = NULL;
    object1.next = NULL;
    object1.data = 10;
    structprev object2;
    object2.previous = NULL;
    object2.next = NULL;
    object2.data = 20;
    structprev object3;

```

```

object3.previous = NULL;
object3.next = NULL;
object3.data = 30;
object1.next = &object2;//forward links
object2.next = &object3;
object2.next = &object1;
                //backward links
object3.next = &object2;
printf ("%d \t", object1.data);
printf ("%d \t", object1.next -> data);
printf ("%d \n", object1.next -> next ->
data);
printf ("%d \t", object2.prev -> data);
printf ("%d \t", object2.data);
printf ("%d \n", object2.next -> data);
printf ("%d \t", object3.prev -> prev -> data);
printf ("%d \t", object3.prev -> data);
printf ("%d \n", object3.data);
return 0;
}

```

Output

```

10 20 30
10 20 30
10 20 30

```

Q12. List out the advantages and disadvantages of structures*Ans :***Advantages of Structure**

- Structure stores more than one data type of the same object together.
- It is helpful when you want to store data of different or similar data types such as name, address, phone, etc., of the same object.
- It makes it very easy to maintain the entire record as we represent complete records using a single name.

- The structure allows passing a whole set of records to any function with the help of a single parameter.
- An array of structures can also be created to store multiple data of similar data types.

Disadvantages of Structure

- If the complexity of the project goes increases, it becomes hard to manage all your data members.
- Making changes to one data structure in a program makes it necessary to change at several other places. So it becomes difficult to track all changes.
- The structure requires more storage space as it allocates memory to all the data members, and even it is slower.
- The structure takes more storage space as it gives memory to all the different data members, whereas union takes only the memory size required by the largest data size parameters, and the same memory is shared with other data members.

5.1.6 Unions**Q13. Define Union***Ans :***(Imp.)**

Union in c language is a user-defined data type that is used to store the different type of elements.

At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.

Structure	Union
<pre> struct Employee{ char x;// size 1 byte int y; //size 2 byte float z; // size 4 byte }e1;//size of e1 = 7 byte size of e1 = 1 + 2 + 4 = 8 </pre>	<pre> Union Employee{ char x; //size 1 byte int y; //size 2 byte float z; // size 4 byte }e1;//size of e1=4byte Size of e1 = 4 (maximum size of 1 element) </pre>

JavaTpoint.com

Advantage of union over structure

It occupies less memory because it occupies the size of the largest member only.

Disadvantage of union over structure

Only the last entered data can be stored in the union. It overwrites the data previously stored in the union.

The union keyword is used to define the union. Let's see the syntax to define union in c.

```
union union_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memberN;
};
```

Let's see the example to define union for an employee in c.

```
union employee
{
    int id;
    char name[50];
    float salary;
};
```

Example

```
#include <stdio.h>
#include <string.h>
union employee
{
    int id;
    char name[50];
}e1;
//declaring e1 variable for union
int main( )
{
    //store first employee information
    e1.id=101;
    strcpy(e1.name, "Sonoo Jaiswal");
    //copying string into char array
    //printing first employee information
    printf( "employee 1 id: %d\n", e1.id);
    printf( "employee 1 name: %s\n", e1.name);
    return 0;
}
```

Output

```
employee 1 id : 1869508435
employee 1 name : SonooJaiswal
```

Q14. Discuss the differences between Structures and Unions.*Ans :***(Imp.)**

Parameter	Structure	Union
Keyword	A user can deploy the keyword struct to define a Structure.	A user can deploy the keyword union to define a Union.
Internal Implementation	The implementation of Structure in C occurs internally - because it contains separate memory locations allotted to every input member.	In the case of a Union, the memory allocation occurs for only one member with the largest size among all the input variables. It shares the same location among all these members/objects.
Accessing Members Syntax	A user can access individual members at a given time. The Syntax of declaring a Structure in C is: struct [structure name] { type element_1; type element_2; . . } variable_1, variable_2, ...;	A user can access only one member at a given time. The Syntax of declaring a Union in C is: union [union name] { type element_1; type element_2; . . } variable_1, variable_2, ...;
Size	A Structure does not have a shared location for all of its members. It makes the size of a Structure to be greater than or equal to the sum of the size of its data members.	A Union does not have a separate location for every member in it. It makes its size equal to the size of the largest member among all the data members.
Value Altering	Altering the values of a single member does not affect the other members of a Structure.	When you alter the values of a single member, it affects the values of other members.
Storage of Value	In the case of a Structure, there is a specific memory location for every input data member. Thus, it can store multiple values of the various members.	In the case of a Union, there is an allocation of only one shared memory for all the input data members. Thus, it stores one value at a time for all of its members.
Initialization	In the case of a Structure, a user can initialize multiple members at the same time.	In the case of a Union, a user can only initiate the first member at a time.

Q15. List out the advantages and disadvantages of Unions.*Ans :***Advantages of Union**

- Union takes less memory space as compared to the structure.
- Only the largest size data member can be directly accessed while using a union.
- It is used when you want to use less (same) memory for different data members.
- It allocates memory size to all its data members to the size of its largest data member.

Disadvantages of Union

- It allows access to only one data member at a time.
- Union allocates one single common memory space to all its data members, which are shared between all of them.
- Not all the union data members are initialized, and they are used by interchanging values at a time.

5.1.7 Type Definition (typedef)**Q16. What is the use of typedef in C programming?***Ans :***(Imp.)**

The **typedef** is a keyword used in C programming to provide some meaningful names to the already existing variable in the C program. It behaves similarly as we define the alias for the commands. In short, we can say that this keyword is used to redefine the name of an already existing variable.

Syntax

```
typedef<existing_name> <alias_name>
```

In the above syntax, 'existing_name' is the name of an already existing variable while 'alias name' is another name given to the existing variable.

For example, suppose we want to create a variable of type **unsigned int**, then it becomes a tedious task if we want to declare multiple variables of this type. To overcome the problem, we use a **typedef** keyword.

```
typedef unsigned int unit;
```

In the above statements, we have declared the unit variable of type unsigned int by using a typedef keyword.

Example

```
#include <stdio.h>

int main()
{
    typedef unsigned int unit;
    unit i,j;
    i=10;
    j=20;
    printf("Value of i is :%d",i);
    printf("\nValue of j is :%d",j);
    return 0;
}
```

Output:

Value of i is :10

Value of j is :20

Using typedef with pointers

We can also provide another name or alias name to the pointer variables with the help of the typedef.

For example, we normally declare a pointer, as shown below:

```
int* ptr;
```

We can rename the above pointer variable as given below:

```
typedef int* ptr;
```

In the above statement, we have declared the variable of type **int***. Now, we can create the variable of type **int*** by simply using the 'ptr' variable as shown in the below statement:

```
ptr p1, p2 ;
```

In the above statement, **p1** and **p2** are the variables of type '**ptr**'.

Q17. Write a C program for implementing typedef with structures.*Ans :*

```
#include <stdio.h>

typedef struct student
{
    char name[20];
    int age;
}stud;

int main()
{
    stud s1; printf("Enter the details of student
                  s1: ");
    printf("\nEnter the name of the student:");
    scanf("%s",&s1.name);
    printf("\nEnter the age of student:");
    scanf("%d",&s1.age);
    printf("\n Name of the student is : %s",
          s1.name);
    printf("\n Age of the student is : %d",
          s1.age);
    return 0;
}
```

Output

```
Enter the details of student s1:
Enter the name of the student: Peter
Enter the age of student: 28
Name of the student is : Peter
Age of the student is : 28
```

5.1.8 Enumerated types**Q18. What is Enumerated types ? How can we declare enumerated types in C?***Ans :***(Imp.)**

The enum in C is also known as the enumerated type. It is a user-defined data type that consists of integer values, and it provides meaningful

names to these values. The use of enum in C makes the program easy to understand and maintain. The enum is defined by using the enum keyword.

The following is the way to define the enum in C:

```
enum flag{integer_const1, integer_const2,
..... integer_constN};
```

In the above declaration, we define the enum named as flag containing 'N' integer constants. The default value of integer_const1 is 0, integer_const2 is 1, and so on. We can also change the default value of the integer constants at the time of the declaration.

For example

```
enum fruits{mango, apple, strawberry,
papaya};
```

The default value of mango is 0, apple is 1, strawberry is 2, and papaya is 3. If we want to change these default values, then we can do as given below:

```
enum fruits
{
    mango=2,
    apple=1,
    strawberry=5,
    papaya=7,
};
```

Enumerated type declaration

As we know that in C language, we need to declare the variable of a pre-defined type such as int, float, char, etc. Similarly, we can declare the variable of a user-defined data type, such as enum. Let's see how we can declare the variable of an enum type.

Suppose we create the enum of type status as shown below:

```
enum status{false,true};
```

Now, we create the variable of status type:

```
enum status s; // creating a variable of the
status type.
```

In the above statement, we have declared the 's' variable of type status.

To create a variable, the above two statements can be written as:

```
enum status{false,true} s;
```

In this case, the default value of false will be equal to 0, and the value of true will be equal to 1.

Example:

```
#include <stdio.h>

enum weekdays{Sunday=1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

int main()
{
    enum weekdays w; // variable declaration of weekdays type
    w=Monday; // assigning value of Monday to w.
    printf("The value of w is %d",w);
    return 0;
}
```

In the above code, we create an enum type named as weekdays, and it contains the name of all the seven days. We have assigned 1 value to the Sunday, and all other names will be given a value as the previous value plus one.

Output

The value of w is 2

The enum is used when we want our variable to have only a set of values. For example, we create a direction variable. As we know that four directions exist (North, South, East, West), so this direction variable will have four possible values. But the variable can hold only one value at a time. If we try to provide some different value to this variable, then it will throw the compilation error.

The enum is also used in a switch case statement in which we pass the enum variable in a switch parenthesis. It ensures that the value of the case block should be defined in an enum.

```
#include <stdio.h>

enum days{sunday=1, monday, tuesday, wednesday, thursday, friday, saturday};

int main()
{
    enum days d;
    d=monday;
    switch(d)
    {
        case sunday:
            printf("Today is sunday");
            break;
        case monday:
            printf("Today is monday");
            break;
    }
```

```

    case tuesday:
        printf("Today is tuesday");
        break;
    case wednesday:
        printf("Today is wednesday");
        break;
    case thursday:
        printf("Today is thursday");
        break;
    case friday:
        printf("Today is friday");
        break;
    case saturday:
        printf("Today is saturday");
        break;
}

return 0;
}

```

Output

Today is Monday

5.2 INPUT AND OUTPUT

5.2.1 Introduction to files**Q19. List out the basic operations of files ?**

Ans :

In programming, we may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling in C.

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- Creation of the new file
- Opening an existing file
- Reading from the file
- Writing to the file
- Deleting the file

Q20. List out the built in functions to perform basic file operations in C.

Ans :

There are many functions in the C library to open, read, write, search and close the file. A list of file functions are given below:

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

5.2.2 Modes of Files**Q21. Explain fopen() function in detail.**

(OR)

List out various modes of files associated with fopen() function.

Ans :

(Imp.)

We must open a file before it can be read, write, or update. The fopen() function is used to open a file. The syntax of the fopen() is given below.

1. **FILE** *fopen(**const char** * filename, **const char** * mode);

The fopen() function accepts two parameters:

- The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like "**c://some_folder/some_file.ext**".
- The mode in which the file is to be opened. It is a string.

We can use one of the following modes in the fopen() function.

Mode	Description
R	opens a text file in read mode
W	opens a text file in write mode
A	opens a text file in append mode
r +	opens a text file in read and write mode
w +	opens a text file in read and write mode
a +	opens a text file in read and write mode
Rb	opens a binary file in read mode
Wb	opens a binary file in write mode
Ab	opens a binary file in append mode
rb +	opens a binary file in read and write mode
wb +	opens a binary file in read and write mode
ab +	opens a binary file in read and write mode.

The f open function works in the following way.

- Firstly, It searches the file to be opened.
- Then, it loads the file from the disk and place it into the buffer. The buffer is used to provide efficiency for the read operations.
- It sets up a character pointer which points to the first character of the file.

Consider the following example which opens a file in write mode.

```
#include <stdio.h>

void main( )
{
    FILE *fp ;
    char ch ;
    fp = fopen("file_handle.c","r") ;
    while ( 1 )
    {
        ch = fgetc ( fp ) ;
        if ( ch == EOF )
```

```

    break ;
    printf("%c",ch) ;
}
fclose (fp ) ;
}

```

Q22. How can we close a file ?

Ans :

The `fclose()` function is used to close a file. The file must be closed after performing all the operations on it. The syntax of `fclose()` function is given below:

```
int fclose( FILE *fp );
```

The `fclose(-)` function returns zero on success, or `EOF` if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The `EOF` is a constant defined in the header file `stdio.h`.

Q23. Discuss the following file operations.

- Writing to the file
- Reading from the file

Ans :

Writing a File

Following is the simplest function to write individual characters to a stream:

```
int fputc( int c, FILE *fp );
```

The function `fputc()` writes the character value of the argument `c` to the output stream referenced by `fp`. It returns the written character written on success otherwise `EOF` if there is an error. We can also use the following functions to write a null-terminated string to a stream "

```
int fputs( const char *s, FILE *fp );
```

The function `fputs()` writes the string `s` to the output stream referenced by `fp`. It returns a non-negative value on success, otherwise `EOF` is returned in case of any error. You can use `fprintf(FILE *fp, const char *format, ...)` function as well to write a string into a file.

Example

```

#include <stdio.h>

main(){
    FILE *fp;

    fp=fopen("/tmp/test.txt","w+");
    fprintf(fp,"This is testing for fprintf...\n");
    fputs("This is testing for fputs...\n",fp);
    fclose(fp);
}

```

Reading a File

Given below is the simplest function to read a single character from a file

```
int fgetc( FILE * fp );
```

The `fgetc()` function reads a character from the input file referenced by `fp`. The return value is the character read, or in case of any error, it returns `EOF`. The following function allows to read a string from a stream

```
char *fgets( char *buf, int n, FILE *fp );
```

The functions `fgets()` reads up to `n-1` characters from the input stream referenced by `fp`. It copies the read string into the buffer `buf`, appending a `null` character to terminate the string.

If this function encounters a newline character `'\n'` or the end of the file `EOF` before they have read the maximum number of characters, then it returns only the characters read up to that point including the new line character. You can also use `int fscanf(FILE *fp, const char *format, ...)` function to read strings from a file, but it stops reading after encountering the first space character.

Example

```

#include <stdio.h>

main()
{
    FILE *fp;
    char buff[255];

    fp=fopen("/tmp/test.txt","r");
    fscanf(fp,"%s", buff);
}

```



```
printf("1 : %s\n", buff );
fgets(buff,255,(FILE*)fp);
printf("2: %s\n", buff );
fgets(buff,255,(FILE*)fp);
printf("3: %s\n", buff );
fclose(fp);
}
```

When the above code is compiled and executed, it reads the file created in the previous section and produces the following result \$

1 : This

2: is testing for fprintf...

3: This is testing for fputs...

5.2.3 Streams, Standard Library Input/Output Functions, Character Input/Output Functions

Q24. What is a Stream ?

Ans :

(Imp.)

A stream is a popular concept for how to do input/output. Basically, a stream is a sequence of characters with functions to take characters out of one end, and put characters into the other end. In the case of input/output streams, one end of the stream is connected to a physical I/O device such as a keyboard or display. If it is a console output stream, your program puts characters into one end of the stream, and the display system takes characters out of the other and puts them on the screen. If it is a console input stream, the keyboard puts characters into one end of the stream, and your program takes characters out of the other and stores the results in variables in the program. If no characters are waiting in the input stream, your program must wait until you supply some by typing on the keyboard. File streams follow the same principle, except that the file system is attached to the other end of the stream.

Two streams exist to allow you to communicate with your "console" - the screen and keyboard setup that makes up the traditional computer user interface. These are the input stream **stdin**(console input), which is connected to the keyboard, and the output stream **stdout** (console

output), which is connected to your display. These two streams are created when the system starts your program. To use stdin and stdout and the input and output functions for them, you have to tell the compiler about them by including the relevant declarations in the library header file.

Q25. Discuss in detail Standard library input and output functions in C.

Ans :

(Imp.)

C programming language libraries that allow input and output in a program. the Stdio.h or Standard input-output library in C that has a method for input and output.

Input

When we talking about the input, so it means to feed some data into a program. an input can be given in the form the command line. C programming has a set of built-in functions to read the given input and feed it to the program as per requirement. **scanf()** is a function which is used to take input.

Output

When we talking about the output, so it means to display something in the screen. The data we want to print in the screen that is print as it is. C programming has a set of built-in function to output the data on the computer screen. **printf()** is a function which is used for output.

C Output

As we see the printf() is an output function in C. this function sends formatted output to the screen.

To use the printf() in our program, we need to include stdio.h header fil using **#include <stdio.h>** statement.

The return 0; statement inside main() function is the Exit status of the program.

Example:

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

```
int main()
```

```
{
```

```
    printf("Hello");
```

```
    return 0;
```

```
}
```

Output

Hello

C Input:

As we see the scanf() is an input function in C this function used to take input from the user. thescanf() function reads formatted input from the standard input such as keyboards.

Example

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

```
int main()
```

```
{
```

```
    int testInteger;
```

```
        float testFloat;
```

```
        double testDouble;
```

```
        char testChar='9';
```

```
        printf("enter a character value ");
```

```
        scanf("%c",&testChar);
```

```
        printf("\nyour entered char value = %c", testChar);
```

```
        printf("\nenter an Integer value ");
```

```
        scanf("%d",&testInteger);
```

```
        printf("\nyour entered integer value = %d ", testInteger);
```

```
        printf("\nenter a float value ");
```

```
        scanf("%f",&testFloat);
```

```
        printf("\nyour entered float value = %f", testFloat);
```

```
        printf("\nenter a double value ");
```

```
        scanf("%lf",&testDouble);
```

```
        printf("\nyour entered ndouble value = %lf", testDouble);
```

```
        return 0;
```

```
}
```

Output

enter a character value g

your entered char value = g

enter an Integer value 45

your entered integer value = 45

enter a float value 4.4

your entered float value = 4.400000

enter a double value 45.89

your entered ndouble value = 45.890000

here we have used %d,%f,%lf and %c formate inside the scanf() function to take int,float, double and character input respectively from user. when user input integer value, it stored in the testInteger variable.

Here's the list of commonly used data type and their formatespecifier. which is used in printf() and scanf() function for take input and print the value of any variable of data type.

Data type	Formate Specifier
int	%d
char	%c
double	%lf
float	%f
short int	%hd
unsigned int	%u
long int	%li
long longint	%lli
unsigned long int	%lu
unsigned long longint	%llu
signed char	%c
unsigned charc	%c
long double	%Lf

Q26. Discuss in detail various character Input and Output functions in C.

Ans :

Character input functions read one character at a time from a text stream, while character output functions write one character at a time to a text stream.

The getchar() function reads the next character from stdin and returns its value. The putchar() function writes one character to standard output.

The getchar() and putchar() Functions

The **intgetchar(void)** function reads the next available character from the screen and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one character from the screen.

The **intputchar(int c)** function puts the passed character on the screen and returns the same character. This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character on the screen.

Example

```
#include<stdio.h>
int main(){
int c;
printf("Enter a value :");
c =getchar();
```

```
printf("\nYou entered: ");
putchar( c );
return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads only a single character and displays it as follows

Enter a value : this is test

You entered: t

The `gets()` and `puts()` Functions

The **`char *gets(char *s)`** function reads a line from **`stdin`** into the buffer pointed to by **`s`** until either a terminating newline or EOF (End of File).

The **`int puts(const char *s)`** function writes the string 's' and 'a' trailing newline to **`stdout`**.

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

```
int main(){
```

```
char str[100];
```

```
printf("Enter a value :");
```

```
gets(str);
```

```
printf("\nYou entered: ");
```

```
puts(str);
```

```
return 0;
```

```
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads the complete line till end, and displays it as follows:

Enter a value : this is test

You entered: this is test

Short Question and Answers

1. Define Structure ?

Ans :

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures simulate the use of classes and templates as it can store various information

The **struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memberN;
};
```

Let's see the example to define a structure for an entity employee in c.

```
struct employee
{
    int id;
    char name[20];
    float salary;
};
```

2. How can we declare structures?

Ans :

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

1st way

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

```
struct employee
{
    int id;
    char name[50];
    float salary;
};
```

Now write given code inside the main() function.

```
struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

2nd way

Let's see another way to declare variable at the time of defining the structure.

```
struct employee
{
    int id;
    char name[50];
    float salary;
}e1,e2;
```

3. What are nested structures explain in detail ?

Ans :

When a structure contains another structure, it is called nested structure. For example, we have two structures named Address and Employee. To make Address nested to Employee, we have to define Address structure before and outside Employee structure and create an object of Address structure inside Employee structure.

Syntax for structure within structure or nested structure

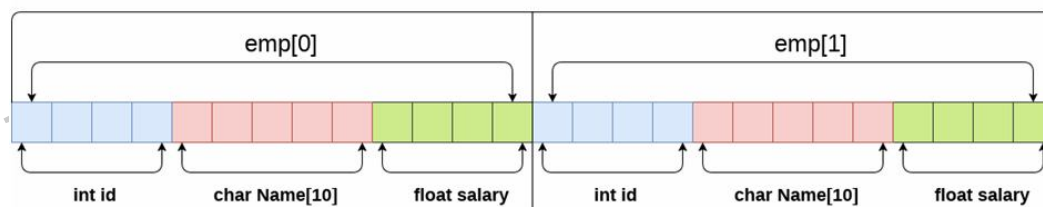
```
struct structure1
{
    -----
    -----
};
struct structure2
{
    -----
    -----
};
struct structure1 obj;
```

4. Explain the concept of array of structures

Ans :

An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

Array of structures



```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

`sizeof (emp) = 4 + 5 + 4 = 13 bytes`

`sizeof (emp[2]) = 26 bytes`

5. Discuss possible ways to passing a structure to a function ?

Ans :

- A structure can be passed to any function from main function or from any sub function.
- Structure definition will be available within the function only.
- It won't be available to other functions unless it is passed to those functions by value or by address(reference).

- Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.

It can be done in below 3 ways.

1. Passing structure to a function by value
2. Passing structure to a function by address (reference)
3. No need to pass a structure – Declare structure variable as global.

6. What are self-referential structures?

Ans :

(Imp.)

A self-referential structure is a structure that can have members which point to a structure variable of the same type. They can have one or more pointers pointing to the same type of structure as their member. The self-referential structure is widely used in dynamic data structures such as trees, linked lists, and so on. The next node of a node will be pointed in linked lists, which consists of the same struct type. It is a unique type of structure containing a member of its type. The member of its type is a pointer variable of the same structure in which it has been declared. In the context of blockchain, each block is linked to a previous node or a next node, similar to a linked list.

Syntax

```
struct structure_name
{
    datatype datatype_name;
    structure_name * pointer_name;
}
```

Example

```
struct node
{
    int data;
    struct node *next;
};
```

7. List out the advantages and disadvantages of structures

Ans :

Advantages of Structure

- Structure stores more than one data type of the same object together.
- It is helpful when you want to store data of different or similar data types such as name, address, phone, etc., of the same object.
- It makes it very easy to maintain the entire record as we represent complete records using a single name.
- The structure allows passing a whole set of records to any function with the help of a single parameter.
- An array of structures can also be created to store multiple data of similar data types.

Disadvantages of Structure

- If the complexity of the project goes increases, it becomes hard to manage all your data members.
- Making changes to one data structure in a program makes it necessary to change at several other places. So it becomes difficult to track all changes.
- The structure requires more storage space as it allocates memory to all the data members, and even it is slower.
- The structure takes more storage space as it gives memory to all the different data members, whereas union takes only the memory size required by the largest data size parameters, and the same memory is shared with other data members.

8. Define Union*Ans :*

Union in c language is a user-defined data type that is used to store the different type of elements.

At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.

Structure	Union
<pre>struct Employee{ char x;// size 1 byte int y; //size 2 byte float z; // size 4 byte }e1;//size of e1 = 7 byte size of e1 = 1 + 2 + 4 = 8</pre>	<pre>Union Employee{ char x; //size 1 byte int y; //size 2 byte float z; // size 4 byte }e1;//size of e1=4byte Size of e1 = 4 (maximum size of 1 element)</pre>

JavaTpoint.com

Advantage of union over structure

It occupies less memory because it occupies the size of the largest member only.

Disadvantage of union over structure

Only the last entered data can be stored in the union. It overwrites the data previously stored in the union.

9. List out the advantages and disadvantages of Unions.*Ans :***Advantages of Union**

- Union takes less memory space as compared to the structure.
- Only the largest size data member can be directly accessed while using a union.
- It is used when you want to use less (same) memory for different data members.
- It allocates memory size to all its data members to the size of its largest data member.

Disadvantages of Union

- It allows access to only one data member at a time.
- Union allocates one single common memory space to all its data members, which are shared between all of them.
- Not all the union data members are initialized, and they are used by interchanging values at a time.

10. What is Enumerated types ? How can we declare enumerated types in C?*Ans :*

The enum in C is also known as the enumerated type. It is a user-defined data type that consists of integer values, and it provides meaningful names to these values. The use of enum in C makes the program easy to understand and maintain. The enum is defined by using the enum keyword.

The following is the way to define the enum in C:

```
enum flag{integer_const1, integer_const2,
..... integer_constN};
```

In the above declaration, we define the enum named as flag containing 'N' integer constants. The default value of integer_const1 is 0, integer_const2 is 1, and so on. We can also change the default value of the integer constants at the time of the declaration.

For example

```
enum fruits{mango, apple, strawberry,
papaya};
```

The default value of mango is 0, apple is 1, strawberry is 2, and papaya is 3. If we want to change these default values, then we can do as given below:

```
enum fruits
{
    mango=2,
    apple=1,
    strawberry=5,
    papaya=7,
};
```

11. What is Enumerated types?*Ans :*

The enum in C is also known as the enumerated type. It is a user-defined data type that consists of integer values, and it provides meaningful names to these values. The use of enum in C makes the program easy to understand and maintain. The enum is defined by using the enum keyword.

The following is the way to define the enum in C:

```
enum flag{integer_const1, integer_const2,
..... integer_constN};
```

In the above declaration, we define the enum named as flag containing 'N' integer constants. The default value of integer_const1 is 0, integer_const2 is 1, and so on. We can also change the default value of the integer constants at the time of the declaration.

For example

```
enum fruits{mango, apple, strawberry,
papaya};
```

The default value of mango is 0, apple is 1, strawberry is 2, and papaya is 3. If we want to

change these default values, then we can do as given below:

```
enum fruits
{
    mango=2,
    apple=1,
    strawberry=5,
    papaya=7,
};
```

12. List out the basic operations of files ?*Ans :*

In programming, we may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling in C.

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- Creation of the new file
- Opening an existing file
- Reading from the file
- Writing to the file
- Deleting the file

13. List out the built in functions to perform basic file operations in C.*Ans :*

There are many functions in the C library to open, read, write, search and close the file. A list of file functions are given below:

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

14. How can we close a file ?

Ans :

The fclose() function is used to close a file. The file must be closed after performing all the operations on it. The syntax of fclose() function is given below:

```
int fclose( FILE *fp );
```

The fclose(-) function returns zero on success, or EOF if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file stdio.h.

15. What is a Stream ?

Ans :

A stream is a popular concept for how to do input/output. Basically, a stream is a sequence of characters with functions to take characters out of one end, and put characters into the other end. In the case of input/output streams, one end of the stream is connected to a physical I/O device such as a keyboard or display. If it is a console output stream, your program puts characters into one end of the stream, and the display system takes characters out of the other and puts them on the screen. If it is a console input stream, the keyboard puts characters into one end of the stream, and your program takes characters out of the other and stores the results in variables in the program. If no characters are waiting in the input stream, your program must wait until you supply some by typing on the keyboard. File streams follow the same principle, except that the file system is attached to the other end of the stream.

Two streams exist to allow you to communicate with your "console" - the screen and keyboard setup that makes up the traditional computer user interface. These are the input stream **stdin**(console input), which is connected to the keyboard, and the output stream **stdout** (console output), which is connected to your display. These two streams are created when the system starts your program. To use stdin and stdout and the input and output functions for them, you have to tell the compiler about them by including the relevant declarations in the library header file.

16. Discuss the differences between Structures and Unions.*Ans :***(Imp.)**

Parameter	Structure	Union
Keyword	A user can deploy the keyword struct to define a Structure.	A user can deploy the keyword union to define a Union.
Internal Implementation	The implementation of Structure in C occurs internally - because it contains separate memory locations allotted to every input member.	In the case of a Union, the memory allocation occurs for only one member with the largest size among all the input variables. It shares the same location among all these members/objects.
Accessing Members	A user can access individual members at a given time.	A user can access only one member at a given time.
Syntax	The Syntax of declaring a Structure in C is: <pre>struct [structure name] { type element_1; type element_2; . . } variable_1, variable_2, ...;</pre>	The Syntax of declaring a Union in C is: <pre>union [union name] { type element_1; type element_2; . . } variable_1,variable_2, ...;</pre>
Size	A Structure does not have a shared location for all of its members. It makes the size of a Structure to be greater than or equal to the sum of the size of its data members.	A Union does not have a separate location for every member in it. It makes its size equal to the size of the largest member among all the data members.
Value Altering	Altering the values of a single member does not affect the other members of a Structure.	When you alter the values of a single member, it affects the values of other members.

Choose the Correct Answers

1. Structure is a _____ data type [b]
(a) System defined (b) User defined
(c) Pre defined (d) None
2. _____ keyword useful to define structures. [c]
(a) int (b) char
(c) struct (d) Enum
3. To access any member of a structure, we use _____ operator. [a]
(a) . (b) ;
(c) < (d) >
4. Structure with in the structure is called as _____ [c]
(a) System defined structure (b) User defined structure
(c) Nested structure (d) All of the above
5. _____ are used to store information about multiple entities of different data types. [b]
(a) Nested structures (b) Array of structures
(c) Structure members (d) Structure variables
6. Structure variable may be passed as _____ [c]
(a) a value (b) a reference
(c) a and b (d) None
7. In the case of _____ a user can only initiate the first member at a time [b]
(a) Structure (b) Union
(c) typed ef (d) Enum
8. _____ mode open a text file in read mode. [a]
(a) R (b) W
(c) Wt (d) Ab
9. _____ function is useful to writes a character into the file. [b]
(a) fopen() (b) fget w()
(c) ftell() (d) fopen()
10. _____ returns current position. [c]
(a) fput w() (b) f get w()
(c) ftell () (d) fopen()

Fill in the blanks

1. Enum in C is also known as _____.
2. Enum is a _____ defined data type.
3. _____ is a sequence of characters with functions to take characters out of one end, and put characters at other end.
4. _____ is a output statement.
5. _____ function is useful to close the file.
6. _____ function reads data from the file.
7. The enum is defined by _____ keyword.
8. _____ is used to store the different type of elements.
9. _____ opens a binary file in read mode.
10. _____ opens a binary file in read and write mode.

ANSWERS

1. Enumerated data type
2. User
3. Streams
4. Printf ()
5. fclose()
6. fscan()
7. Enum
8. Union
9. Rb
10. abt

One Mark Answers

1. Define structure

Ans :

Structure is a user defined data type that enables us to store the collection of different data types.

2. Nested Structure?

Ans :

A structure within another structure is called nested structures.

3. Union?

Ans :

Union is a user defined data type that is used to store the different type of elements.

4. Enumaration?

Ans :

It is a user defined data type that consists of integer values, and it provides meaningful names to these values.

5. Typedef ?

Ans :

The typedef is a keyword. Used to provide some meaningful names to the already existing variable in the C-program.

FACULTY OF INFORMATICS
BCA I-Year, I-Semester (CBCS) (Main) (New) Examination
Model Paper - I
PROGRAMMING IN C

Time : 3 Hours]

[Max. Marks : 70

Note : Answer all questions from Part - A, & any five questions from Part - B
Choosing one questions from each unit.

PART - A (10 × 2 = 20 Marks)

ANSWERS

- | | | |
|----|--|-------------------|
| 1. | (a) Define computer? | (Unit-I, SQA-1) |
| | (b) What is number system? | (Unit-I, SQA-4) |
| | (c) What is Function ? What are the advantages of functions ? | (Unit-II, SQA-10) |
| | (d) What are the differences between If-else and switch statements. | (Unit-II, SQA-5) |
| | (e) Discuss #ifdef preprocessor directive. | (Unit-III, SQA-1) |
| | (f) Write a Binary search algorithm. | (Unit-III, SQA-8) |
| | (g) What are the differences between call-by-value and call-by-reference. | (Unit-IV, SQA-4) |
| | (h) What are the advantages and usages of pointers ? | (Unit-IV, SQA-2) |
| | (i) How can we close a file ? | (Unit-V, SQA-14) |
| | (j) List out the built in functions to perform basic file operations in C. | (Unit-V, SQA-13) |

PART - B (5 × 10 = 50 Marks)

UNIT - I

- | | | |
|----|--|--------------------|
| 2. | (a) Briefly describe about various Computer Languages with examples. | (Unit-I, Q.No. 3) |
| | (b) How to evaluate C Expression ? Explain with the help of an example | (Unit-I, Q.No. 42) |

OR

- | | | |
|----|--|--------------------|
| 3. | (a) What is SDLC? | (Unit-I, Q.No. 6) |
| | (b) Explain, how can we convert an octal number system to other number systems | (Unit-I, Q.No. 24) |

UNIT - II

- | | | |
|----|--|---------------------|
| 4. | (a) List out various C Library functions. | (Unit-II, Q.No. 28) |
| | (b) List out various standard function of C. | (Unit-II, Q.No. 32) |

OR

- | | | |
|----|---|---------------------|
| 5. | (a) List out various types of storage classes in C. | (Unit-II, Q.No. 37) |
| | (b) List and explain all the relational operators in C. | (Unit-II, Q.No. 3) |

UNIT - III

6. (a) What are the differences between linear search and binary search. (Unit-III, Q.No. 21)
(b) Write an algorithm to implement bubble sorting. (Unit-III, Q.No. 25)

OR

7. (a) Write a C Program for multiplication of two matrices. (Unit-III, Q.No. 14)
(b) Discuss in detail about Multi dimensional arrays in C. (Unit-III, Q.No. 11)

UNIT - IV

8. (a) Explain the following functions in detail. (Unit-IV, Q.No. 22)
(i) strlen()
(ii) strcpy()
(iii) strcat()
(iv) strcmp()
(b) Explain the following string functions with an example. (Unit-IV, Q.No. 23)
(a) strrev()
(b) strlwr()
(c)strupr()

OR

9. (a) How can we pass an array to a function (Unit-IV, Q.No. 11)
(b) What is Dynamic Memory allocation? List and explain standard defined functions used for dynamic memory allocation. (Unit-IV, Q.No. 12)

UNIT - V

10. (a) How can we define a structure with in other structure ? (Unit-V, Q.No. 4)
(b) Explain the concept of array of structures. (Unit-V, Q.No. 6)

OR

11. (a) How can we define pointers to structures ? (Unit-V, Q.No. 10)
(b) List out the built in functions to perform basic file operations in C. (Unit-V, Q.No. 20)

FACULTY OF INFORMATICS
BCA I-Year, I-Semester (CBCS) (Main) (New) Examination
Model Paper - II
PROGRAMMING IN C

Time : 3 Hours]

[Max. Marks : 70

Note : Answer all questions from Part - A, & any five questions from Part - B
Choosing one questions from each unit.

PART - A (10 × 2 = 20 Marks)

ANSWERS

- | | | |
|----|--|-------------------|
| 1. | (a) What is SDLC? | (Unit-I, SQA-2) |
| | (b) What is decimal number system? | (Unit-I, SQA-6) |
| | (c) How can we define a function in C ? Explain with an example ? | (Unit-II, SQA-11) |
| | (d) Explain Ternary Operator. | (Unit-II, SQA-1) |
| | (e) What are the differences between linear search and binary search. | (Unit-III, SQA-8) |
| | (f) How memory allocation is done in arrays ? | (Unit-III, SQA-4) |
| | (g) What is Null Pointer. | (Unit-IV, SQA-3) |
| | (h) What is Dynamic Memory allocation? | (Unit-IV, SQA-7) |
| | (i) What is Enumerated types ? How can we declare enumerated types in C? | (Unit-V, SQA-10) |
| | (j) List out the advantages and disadvantages of structures | (Unit-V, SQA-7) |

PART - B (5 × 10 = 50 Marks)

UNIT - I

- | | | |
|----|---|--------------------|
| 2. | (a) How many types of computing Environments are there? Explain them with their advantages and disadvantages. | (Unit-I, Q.No. 2) |
| | (b) Draw a flowchart for calculating the area of a rectangle. | (Unit-I, Q.No. 14) |

OR

- | | | |
|----|---|--------------------|
| 3. | (a) What is Flowchart ? | (Unit-I, Q.No. 8) |
| | (b) Explain standard I/O functions in C | (Unit-I, Q.No. 40) |

UNIT - II

- | | | |
|----|---|---------------------|
| 4. | (a) How can we use while loop in c programming? | (Unit-II, Q.No. 16) |
| | (b) What is mean by nested loops? | (Unit-II, Q.No. 19) |

OR

- | | | |
|----|---|---------------------|
| 5. | (a) What is Function? What are the advantages of functions ? | (Unit-II, Q.No. 24) |
| | (b) Write a C program to find the largest of given three numbers. | (Unit-II, Q.No. 7) |

UNIT - III

6. (a) How memory allocation is done in arrays ? (Unit-III, Q.No. 5)
(b) What is array ? Explain its advantages. (Unit-III, Q.No. 3)

OR

7. (a) Discuss in detail about Multi dimensional arrays in C. (Unit-III, Q.No. 11)
(b) What is linear search? Write a linear search algorithm. (Unit-III, Q.No. 16)

UNIT - IV

8. (a) Explain the concept of Pointers to Functions. (Unit-IV, Q.No. 16)
(b) Why do we use command line arguments in C? (Unit-IV, Q.No. 17)

OR

9. (a) What is Array of String ? (Unit-IV, Q.No. 20)
(b) What are string Input Output functions? (Unit-IV, Q.No. 19)

UNIT - V

10. (a) Discuss the differences between Structures and Unions. (Unit-V, Q.No. 14)
(b) What is Enumerated types ? How can we declare enumerated types in C? (Unit-V, Q.No. 18)

OR

11. (a) Explain fopen() function in detail. (Unit-V, Q.No. 21)
(b) What is a Stream ? (Unit-V, Q.No. 24)

FACULTY OF INFORMATICS
BCA I-Year, I-Semester (CBCS) (Main) (New) Examination
Model Paper - III
PROGRAMMING IN C

Time : 3 Hours]

[Max. Marks : 70

Note : Answer all questions from Part - A, & any five questions from Part - B
Choosing one questions from each unit.

PART - A (10 × 2 = 20 Marks)

ANSWERS

- | | | |
|----|---|-------------------|
| 1. | (a) What is Flowchart ? | (Unit-I, SQA-3) |
| | (b) Explain the types of type conversions in C with example? | (Unit-I, SQA-10) |
| | (c) What is Function ? What are the advantages of functions ? | (Unit-II, SQA-10) |
| | (d) What are the differences between call by value and call by reference. | (Unit-II, SQA-13) |
| | (e) What is linear search. | (Unit-III, SQA-7) |
| | (f) Write a C program for sorting of an elements using array . | (Unit-III, SQA-5) |
| | (g) List out various string manipulation functions in C. | (Unit-IV, SQA-12) |
| | (h) What is the use of realloc() function? | (Unit-IV, SQA-8) |
| | (i) Define Union. | (Unit-V, SQA-8) |
| | (j) Define Structure ? | (Unit-V, SQA-1) |

PART - B (5 × 10 = 50 Marks)

UNIT - I

- | | | |
|----|--|--------------------|
| 2. | (a) Explain the various stages of SDLC life cycle. | (Unit-I, Q.No. 7) |
| | (b) Explain the types of type conversions in C with example? | (Unit-I, Q.No. 48) |

OR

- | | | |
|----|---|--------------------|
| 3. | (a) Design a flowchart for finding the largest among three numbers entered by the user. | (Unit-I, Q.No. 10) |
| | (b) How to declare and initialize a variable? | (Unit-I, Q.No. 38) |

UNIT - II

- | | | |
|----|---|--------------------|
| 4. | (a) What is Operator? List out various types of Operators supported by C? | (Unit-II, Q.No. 1) |
| | (b) Explain in detail If statement with an example. | (Unit-II, Q.No. 6) |

OR

- | | | |
|----|---|---------------------|
| 5. | (a) Explain in detail switch statement in C. | (Unit-II, Q.No. 12) |
| | (b) What are the differences between If-else and switch statements. | (Unit-II, Q.No. 13) |

UNIT - III

6. (a) List out various preprocessor commands supported by C. (Unit-III, Q.No. 1)
(b) Explain the process of declaring and initializing an array ? (Unit-III, Q.No. 6)

OR

7. (a) Write a Binary search algorithm. (Unit-III, Q.No. 18)
(b) What are the differences between linear search and binary search. (Unit-III, Q.No. 21)

UNIT - IV

8. (a) Define pointer? Explain the process of declaring a pointer in c program. (Unit-IV, Q.No. 1)
(b) How can we use arrays using pointers? (Unit-IV, Q.No. 9)

OR

9. (a) What is Dynamic Memory allocation? List and explain standard defined functions used for dynamic memory allocation. (Unit-IV, Q.No. 12)
(b) What is the use of Array of Pointers in C? (Unit-IV, Q.No. 13)

UNIT - V

10. (a) How can we declare structures? (Unit-V, Q.No. 2)
(b) Explain the concept of array of structures. (Unit-V, Q.No. 6)

OR

11. (a) How can we define a structure with in other structure ? (Unit-V, Q.No. 4)
(b) What are self-referential structures? Write the syntax of self-referential structures. (Unit-V, Q.No. 11)

FACULTY OF INFORMATICS

BCA I - Semester (CBCS) (Main & Backlog) (New) Examination

July - 2021

PROGRAMMING IN C

Time : 3 Hours]

[Max. Marks : 70

Note : Answer all questions from Part - A, & any five questions from Part - B

Choosing one questions from each unit.

PART - A (10 × 2 = 20 Marks)

ANSWERS

- | | |
|---|------------------------------------|
| 1. What are generations of computer languages ? | (Unit-I, Q.No.3) |
| 2. What are the rules to construct a variable name ? | (Unit-I, Q.No.37) |
| 3. List out bitwise operators of C. | (Unit-II, Q.No.2) |
| 4. Define the array and pointers, write their syntax. | (Unit-III, Q.No.3, Unit-IV, SQA-1) |
| 5. What is recursion ? Write its syntax. | (Unit-II, SQA-14) |
| 6. Define the structure and write its syntax. | (Unit-V, SQA-1) |
| 7. List out memory management functions of 'C' with their syntax. | (Unit-IV, SAQ.7) |
| 8. What is L-value and R-value ? | (Unit-IV, Q.No.8) |
| 9. Write the file operating modes. | (Unit-V, Q.No.21) |
| 10. List out the file predefined pointers. | |

Ans :

File pointer is a pointer which is used to handle and keep track on the files being accessed. A new data type called "FILE" is used to declare file pointer. This data type is defined in stdio.h file. File pointer is declared as FILE *fp. Where, 'fp' is a file pointer.

fopen() function is used to open a file that returns a FILE pointer. Once file is opened, file pointer can be used to perform I/O operations on the file. fclose() function is used to close the file.

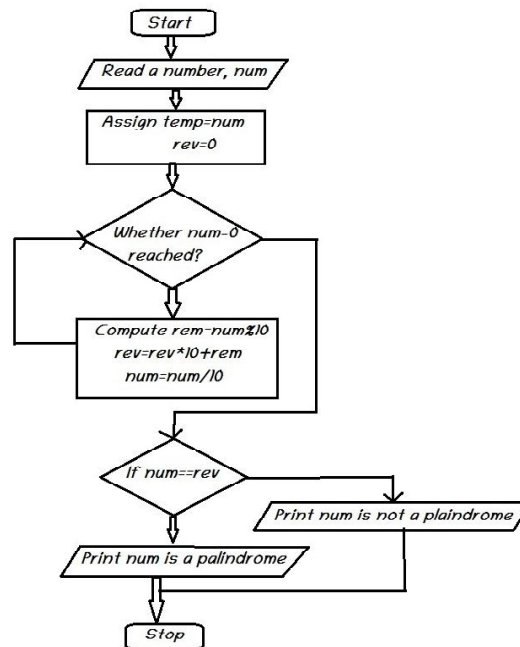
PART - B (5 × 10 = 50 Marks)

UNIT - I

11. (a) Draw a flow chart for given number is palindrome or not ?

Ans :

Program to check whether a number is palindrome or not.



(b) Convert the $(144.128)_{10}$ to Octal & Hexadecimal?

Ans :

To avoid the decimal separator, multiply the decimal number with the base raised to the power of decimals in result:

$$144.128 \times 16^8 = 619025046438$$

Divide by the base 16 to get the digits from the remainders:

Division by 16	Quotient	Remainder (Digit)	Digit #
(619025046438)/16	38689065402	6	0
(38689065402)/16	2418066587	10	1
(2418066587)/16	151129161	11	2
(151129161)/16	9445572	9	3
(9445572)/16	590348	4	4
(590348)/16	36896	12	5
(36896)/16	2306	0	6
(2306)/16	144	2	7
(144)/16	9	0	8
(9)/16	0	9	9

$$= (9020C49BA5)_{16} / 16^8$$
$$= (90.20C49BA5)_{16}$$

OR

12. (a) Describe C-programming language data types with their syntax. (Unit-I, Q.No.36)
(b) Explain the operators of C-programming language. (Unit-I, Q.No.41)
13. Explain storage classes of C with example program. (Unit-II, Q.No.37)

OR

14. Discuss the all conditional control statements with example program. (Unit-II, Q.No.5,6,7,8,9,12)
15. Describe the demonstrate inter function communication with arrays. (Unit-III, Q.No.8)

OR

16. Write a C-program for Binary search ? (Unit-III, Q.No.20)
17. (a) Write a C-program to demonstrate command line arguments ? (Unit-IV, Q.No.17)
(b) Write a C-program to demonstrate pointer for inter function communication ?
(Unit-IV, Q.No.5)

OR

18. What is string and write a program to represent at least 10-pre defined string functions ?
(Unit-IV, Q.No.18, 22, 23)
19. Write a C-program to demonstrate array of structure variable with suitable example.
(Unit-V, Q.No.6)

OR

20. Write a program to copy contents from one file to another.

Ans :

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main(){
    char ch; // source_file[20], target_file[20];
    FILE *source, *target;
    char source_file[] = "x1.txt";
    char target_file[] = "x2.txt";
    source = fopen(source_file, "r");
```

```
        if(source == NULL){
            printf("Press any key to exit...\n");
            exit(EXIT_FAILURE);
        }
        target=fopen(target_file,"w");
        if(target == NULL){
            fclose(source);
            printf("Press any key to exit...\n");
            exit(EXIT_FAILURE);
        }
        while((ch=fgetc(source))!= EOF)
            fputc(ch, target);
        printf(" File copied successfully.\n");
        fclose(source);
        fclose(target);
        return 0;
    }
```


FACULTY OF SCIENCE
B.C.A. I Year I-Semester(CBCS) Examination
December - 2019
PROGRAMMING IN C

Time : 3 Hours]

[Max. Marks : 80

PART - A (5 × 4 = 20 Marks)

Note : Answer all questions from Part-A & any Five questions from Part-B choosing one question from each unit.

ANSWERS

1. (a) What are assemblers'?

Ans :

The Assembler is a Software that converts an assembly language code to machine code. It takes basic Computer commands and converts them into Binary Code that Computer's Processor can use to perform its Basic Operations. These instructions are assembler language or assembly language.

- | | |
|---|--------------------|
| (b) What are types of computing environments? | (Unit-I, Q.No.2) |
| (c) What is call by value vnte the syntax? | (Unit-II, Q.No.33) |
| (d) What is Recursion? Write the syntax. | (Unit-II, SQA-14) |
| (e) List the pre processors | (Unit-III, Q.No.1) |
| (f) Write the Array Applications. | (Unit-III, Q.No.9) |
| (g) What is L-value and R-value. | (Unit-IV, Q.No.8) |
| (h) List string function. | (Unit-IV, SQA-12) |
| (i) Wnte the standard library Input output functions. | (Unit-V, Q.No.25) |
| (j) What is array of structures? | (Unit-V, SQA-4) |

PART - B (4 × 15 = 50 Marks)

UNIT - I

2. (a) Convert the numbers Decimal to Hexa Decimal.

- (i) 512 (ii) 1536 (iii) 1024

Ans :

- (i) **512**

$$(512)_{10} = (200)_{16}$$

Step by step solution

Step 1: Divide $(512)_{10}$ successively by 16 until the quotient is 0:

$$512/16 = 32, \text{ remainder is } 0$$

$32/16 = 2$, remainder is 0

$2/16 = 0$, remainder is 2

Step 2: Read from the bottom (MSB) to top (LSB) as 200. This is the hexadecimal equivalent of decimal number 512

(ii) 1536

$(1536)_{10} = (600)_{16}$

Step by step solution

Step 1: Divide $(1536)_{10}$ successively by 16 until the quotient is 0:

$1536/16 = 96$, remainder is 0

$96/16 = 6$, remainder is 0

$6/16 = 0$, remainder is 6

Step 2: Read from the bottom (MSB) to top (LSB) as 600.

(iii) 1024

$(1024)_{10} = (400)_{16}$

Step by step solution

Step 1: Divide $(1024)_{10}$ successively by 16 until the quotient is 0:

$1024/16 = 64$, remainder is 0

$64/16 = 4$, remainder is 0

$4/16 = 0$, remainder is 4

Step 2: Read from the bottom (MSB) to top (LSB) as 400.

(b) Explain about type conversion.

(Unit-I, Q.No.47, 48)

OR

3. (a) Explain about precedence and Associativity of operators.

(Unit-I, Q.No.43)

(b) Explain about identifiers and variables

(Unit-I, Q.No.34, 37)

UNIT - II

4. (a) Explain about storage classes.

(Unit-II, Q.No.37)

(b) Write a program for Armstrong number.

Ans :

```
#include <stdio.h>
int main()
{
    int n,r,sum=0,temp;
    printf("enter the number=");
    scanf("%d",&n);
    temp=n;
    while(n>0)
    {
```

```

r=n%10;
sum=sum+(r*r*r);
n=n/10;
}
if(temp==sum)
printf("armstrong  number ");
else
printf("not armstrong number");
return 0;
}

```

Output:

```

enter the number=153
armstrong number
enter the number=5
notarmstrong number

```

OR

5. (a) What is call by reference? Write a program for call by value. (Unit-II, Q.No.33)
 (b) Explain about the methods of parameter passing. (Unit-II, Q.No.33)

UNIT - III

6. (a) Explain about types of Arrays with examples. (Unit-III, Q.No.9,10,11)
 (b) Write a program for Matrix Multiplication. (Unit-III, Q.No.14)

OR

7. (a) What is the difference between Selection sort and Bubble sort. (Unit-III, Q.No.23, 25)
 (b) Write a program for linear search. (Unit-III, Q.No.17)

UNIT - IV

8. Explain about pointers to pointers. (Unit-IV, Q.No.6)

OR

9. (a) Write a program to print the reverse of the given array. (Unit-IV, Q.No.23(a))
 (b) Write a program for string concat and string copy functions. (Unit-IV, Q.No.23 b,c)

UNIT - V

10. (a) What are nested structures explain. (Unit-V, Q.No.4)
 (b) Difference between structures and unions. (Unit-V, Q.No.14)

OR

11. (a) Explain about type definition. Explain about modes of a file. (Unit-V, Q.No.16,21)

(b) Write a program to find no. of line, characters, no.of spaces present in a file.

Ans :

```
#include <stdio.h>
int main()
{
    char in_name[80];
    FILE *in_file;
    int ch, character = 0, line = 0, space = 0, tab = 0; printf("Enter file name:\n");
    scanf("%s", in_name);
    in_file = fopen(in_name, "r");
    if (in_file == NULL)
        printf("Can't open %s for reading.\n", in_name);
    else
    {
        while ((ch = fgetc(in_file)) != EOF)
        {
            character++; if (ch == ' ') space++; if (ch == '\n') line++; if (ch == '\t') tab++;
        }
        fclose(in_file);
        printf("\nNumber of characters = %d", character); printf("\nNumber of spaces = %d", space);
        printf("\nNumber of tabs = %d", tab);
        printf("\nNumber of lines = %d", line);
    }
    return 0;
}
```

Output:

Enter file name:

count.txt

Number of characters = 82

Number of spaces = 12

Number of tabs = 1

Number of lines = 8