UNIT 1

INTRODUCTION TO PROGRAMMING, IDEA AND REPRESENTATION OF ALGORITHM

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Define secondary storage devices.

Answer :

Secondary storage device is a permanent storage device, which is also called auxiliary storage. Secondary memory is the slowest and cheapest form of memory. It is not possible to process such memory directly by the CPU. It is necessary to initially copy the data present in secondary memory into the primary storage. Secondary storage devices hold the information until it is deleted or overwritten.

Some of the secondary storage devices are,

(a) Magnetic tapes

(b) Magnetic disks

(c) Optical disks.

Q2. Define magnetic disks.

Answer :

Magnetic disks are the most commonly used secondary storage medium. The advantages of these disks over the magnetic tapes are,

(i) They provide high storage capacity

(ii) They are much reliable

(iii) They have the ability to directly access the stored data.

Magnetic disk comprises of circular plate made of either plastic or metal. This plate is coated with magnetic oxide layer. Data in this disk is stored either on the magnetized or demagnetized layer. The bit-value of 1 is represented on a magnetized spot and bit value of 0 is represented on demagnetized spot. In order to carry out the read operation, the data present on the magnetized surface is converted into electrical impulses which are then sent to the processor for execution. On the other hand, the write operation can be carried out by converting those electrical impulses into magnetic spots. Magnetic disk is engraved inside a protective shield in order to protect it from dust or other interferences.

Q3. What is the role of a compiler?

Answer :

A compiler refers to a software that transforms high-level language codes into machine language code. In other words, a compiler transforms the source code into binary code such that an executable program is created. The operations performed by a compiler includes lexical analysis, preprocessing, parsing, code generation and optimization etc. The compiler sometimes generate program faults that are often difficult to track down. Therefore, the compiler implementors focus more on making a consistent compiler.

Q4. Define algorithm and flowchart.

Answer :

Algorithm

Algorithm is a step-by-step process of solving a particular problem. It is basically, a series of steps in a computer program which provides the solution for a particular problem.

Flowchart

Flowchart is the graphical representation of an algorithm. Earlier, flowcharts were considered as a very crucial part on the process of programming. It is an important tool for planning and designing a new system.

Q5. List the characteristics of an algorithm.

Answer :

The following are the characteristics of an algorithm,

1. Finiteness
2. Definiteness
3. Effectiveness
4. Generality
5. Input/Output.

Q6. Steps for algorithm development.

Answer :

The steps to be followed while developing an algorithm are,

1. Initially understand the problem.
2. Identify the expected output for the problem.
3. Identify the necessary input for the problem.
4. Develop a logic that produces the expected output from the selected input.
5. Finally, test the algorithm with various sets of input.

Q7. Write an algorithm to find the largest among three numbers.

Answer :

Algorithm to find largest among three numbers is as follows,

```
1.  Begin
2.  Declare the variables num1, num2 and num3.
3.  Read the variables num1, num2 and num3.
4.  If num1 > num2
    If num1 > num3
        Display num1 is the largest number
    else
        Display num3 is the largest number
    else
        if num2 > num3
        Display num2 is the largest number
    else
        Display num3 is the largest number
5.  Stop.
```

Q8. Write an algorithm to find the factorial of a number.

Answer :

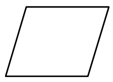
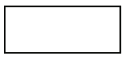
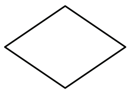
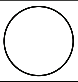
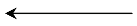
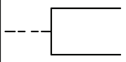
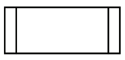
Algorithm to find the factorial of a number is as follows,

```
1.  Begin
2.  Declare the variables num, factorial, i
3.  Initialize factorial to 1
4.  Initialize i to 1
5.  Read num for which factorial has to be found
6.  Repeat the following two steps until i = num
        (a)  Calculate factorial = factorial * i
        (b)  Increment i by 1
7.  Display factorial
8.  Stop.
```

Q9. What symbols are used in flowchart and give their purpose?

Answer :

Following are the standard symbols used in drawing flowcharts.

Oval		Terminal	start/stop/begin/end symbol
Parallelogram		Input/output	Making data available for processing (input) or recording of the processed Information (output)
Rectangle		Process	Any processing to be performed. An assignment operation normally represented by this symbol
Diamond		Decision	Decision or switching type of operations that determines which of the alternative paths is to be followed
Circle		Connector	Used for connecting different parts of flow chart.
Arrow		Flow	Joins two symbols and also represents executions flow.
Bracket with broken line		Annotation	Descriptive comments or explanations
Double sided rectangle		Predefined process	Modules or subroutines given else where

Q10. Define program.

Answer :

Program is a set of instructions developed from an algorithm. It is written in computer understandable language to accomplish certain task. Computers play a major role in solving a problem. This is because of the good communication that exists between computers and their corresponding users. In order to write a program, a process called 'programming' is required. But, before writing this, the problem to be solved should be analyzed and needs to be prepared by generating a step-by-step procedure. The sufficient details about the problem should be obtained. This process is referred to as 'problem analysis'.

Q11. Define variable.

Answer :

Variable

A variable is the name given to the memory location, where the data value (i.e., value assigned to the variable) of the variable is stored. Unlike constant, the value of a variable can be changed during execution. A variable can be classified as integer variable, real variable, character variable depending on the type of value stored in it. A variable name must be chosen in such a way, that it improves the readability of a program and also reflects its function.

Example

cp_int, hr_gs, salary, *x*, *y*, *z*, roll_1231, etc.

Q12. What are syntax errors?

Answer :

Syntax Errors

Syntax errors are generated when the statements are not written according to the syntax or grammatical rules of a language. These type of errors are detected by the complier during compilation process. Hence, it is also called as compile-time error.

If a program contains a syntax error, then the compiler fails to compile it and the program gets terminated after displaying a list of errors with their corresponding line number in which they have took place.

Example

```
main( )
{
    int a, b;
myfun( )
{
    a = 2;
    b = 3;
    sum = a + b;
}
```

In the above code, there is no closing curly braces for the function. Hence, on the execution of the code nothing will be displayed on the screen.

PART-B

ESSAY QUESTIONS WITH SOLUTIONS

1.1 INTRODUCTION TO PROGRAMMING : INTRODUCTION TO COMPONENTS OF A COMPUTER SYSTEM (DISKS, MEMORY, PROCESSOR, WHERE A PROGRAM IS STORED AND EXECUTED, OPERATING SYSTEM, COMPILERS ETC.)

Q13. Explain about primary memory and secondary memory.

Answer :

Primary Memory

Primary memory consist of data which is required to be manipulated immediately. In addition to this it also stores the information about what is being processed by the processor. Primary memory is also known as main memory. They are two types of primary memories.

(a) RAM

Random Access Memory (RAM) forms one of the basic memories of computers today. They cannot hold data permanently. Hence, they are utilized to store several intermediate results and other temporary data. Data remains in these memories as long as there is constant supply of power i.e., once the power is turned off data inside it is lost.

Data inside these memories can be accessed randomly from any location, hence the name of the memory is RAM.

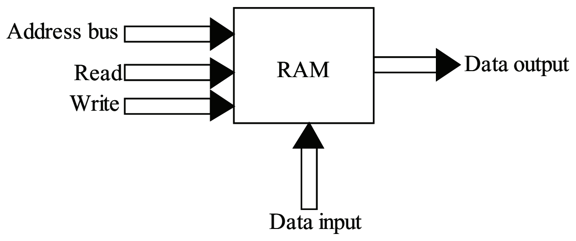


Figure: Block Diagram of Random Access Memory

Data reading and writing mechanisms in this memory are quite simple. In order to write the data, initially the address (which describes the position where the data is to be loaded) is placed on the address bus. Later the required data is placed on the data lines. Finally write line is activated. In the same way in order to read the data from this memory, required address is placed on the address lines and read line is activated. Data can be then collected through data output lines. Hence in this way reading and writing tasks can be accomplished.

(b) ROM

Read only memories are those memories which does not lose its data, even though the power is turned off. Due to this nature these memories are referred as non-volatile memories. Hence by using ROMs, permanent storage of data can be made.

Basically there are many types of Read only memories with differing capabilities, few of them are as follows,

(i) Programmable Read Only Memory (PROM)

(ii) Electrically Erasable and Programmable Read Only Memory (EEPROM)

(iii) Flash Memories etc.

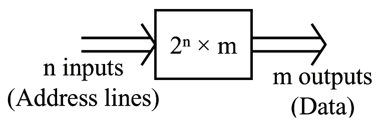


Figure: Block Diagram of ROM

There is no write operation possible with ROMs, since data is added to it only while it is being manufactured.

Secondary Memory

It is a permanent storage device, which is also called auxiliary storage. Secondary memory is the slowest and cheapest form of memory. It is not possible to process such memory directly by the CPU. It is necessary to initially copy the data present in secondary memory into the primary storage. Secondary storage devices hold the information until it is deleted or overwritten.

Some of the secondary storage devices are,

(a) Magnetic tapes

(b) Magnetic disks

(c) Optical disks.

(a) Magnetic Tapes

Magnetic tapes are plastic tapes that have a magnetic coating around it. In such tapes, the data is stored in the form of small portion of magnetized and demagnetized layer. The magnetized portion signifies the bit value 1 whereas the demagnetized portion signifies the bit value 0. There are different types of magnetic tapes each of which differ in their sizes and their speed (with which the tape moves the read/write head). Magnetic tapes also differ with respect to recording density that specifies the amount of data that can be stored on a linear inch of tape.

The advantage of magnetic tapes is that they are very durable. The magnetic tapes can be erased and even reused many number of times. Magnetic tapes are very much reliable and are inexpensive when compared to other secondary storage devices.

Magnetic tapes are sequential in nature and does not allow data to be accessed randomly from them. The data is transmitted at very slow speed in comparison to the magnetic disks.

(b) Magnetic Disks

Magnetic disks are the most commonly used secondary storage medium. The advantages of these disks over the magnetic tapes are,

(i) They provide high storage capacity

(ii) They are much reliable

(iii) They have the ability to directly access the stored data.

Magnetic disk comprises of circular plate made of either plastic or metal. This plate is coated with magnetic oxide layer. Data in this disk is stored either on the magnetized or demagnetized layer. The bit-value of 1 is represented on a magnetized spot and bit value of 0 is represented on demagnetized spot. In order to carry out the read operation, the data present on the magnetized surface is converted into electrical impulses which are then sent to the processor for execution. On the other hand, the write operation can be carried out by converting those electrical impulses into magnetic spots. Magnetic disk is engraved inside a protective shield in order to protect it from dust or other interferences.

(c) Optical Disk

Optical disk is a form of external storage device which is most widely used today for storing large volume of computer data. An optical disk is a round, flat piece of plastic disk. This disk is coated with a material on which data is written in a form of highly reflective areas. The stored data may be read from less reflective areas by using a laser diode.

The advantage of optical disk is that, massive volume of data can be stored in very less space. There are many optical disks available in the market that differ in their sizes and storage capacities. It is cost effective.

Q14. Explain about processor.

Answer :

Processor

The term Central Processing Unit (CPU) has been replaced by the term processor. It is embedded in the small devices in a PC and usually referred to as microprocessor. The role of the processor is to control the functioning of all the hardware and software present in the PC. Therefore, it is regarded as the brain of the computer.

Central processing unit is considered as the brain of computer system.

Functions of CPU

1. Interprets the operations
2. Coordinates the operations
3. Supervises the instructions
4. Controls all internal as well as external peripherals
5. Controls the computer's functions
6. Performs all calculations (Arithmetic and Logic)
7. Processes all user-entered data.

Components of CPU

(a) Arithmetic Logic Unit

(b) Control Unit

(c) Memory Unit

(d) Registers.

(a) Arithmetic Logic Unit

All the arithmetic and logical operations are performed by Arithmetic Logic Unit.

(b) Control Unit

This unit is responsible for controlling the operations sequence.

(c) Memory Unit

This unit stores the intermediate result obtained during the calculations and provides the data to the user depending on their requirements.

(d) Registers

Registers are considered as special purpose, high-speed temporary memory components that are capable of storing different types of information like data, instruction, addresses etc. It generally store that information which is currently being used by CPU.

Types of Registers

The following are different types of registers,

(i) Program Counter (PC) Register

(ii) Instruction Register (IR)

(iii) Memory Address Register (MAR)

(iv) Memory Buffers Register (MBR)

(v) Accumulator (ACC)

(vi) Data Register (DR).

Q15. Write the various steps involved in executing a C program.

Answer :

Execution Steps of a 'C' Program

A 'C' program has to pass through many phases for its successful execution and to achieve desired output.

The phases or steps required for successful execution of a 'C' program can be visualized as follows,

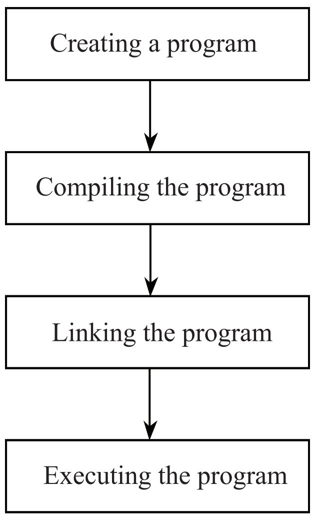


Figure: Steps of 'C' Program

1. Creating a Program

In this, phase, the C program is entered into a file (i.e., source code) through a text editor. A text editor allows the user to enter, modify and store the character data. UNIX operating system provides a text editor which can be accessed through the command *vi*. Modern compiler vendors provide IDEs (Integrated Development Environments) consisting of both editor and compilers (Example: Turbo C, Turbo C++, Borland C++, etc.). The file is saved on the disk with an extension of 'c' (not mandatory, user can have their own extension). Corrections in the program at later stages are done through these editors. Once the program has been written, it requires to be translated into machine language (i.e., compilation phase).

2. Compilation Phase

This phase is carried out by a program called compiler. Compiler translates the source code into the object code (i.e., machine understandable code). The compilation phase cannot be proceeded successfully until and unless the source code is error-free (i.e., no syntactic and semantic errors). The compiler generates messages, if it encounters syntactic errors in the source code. The error-free source code is translated into object code and stored in a separate file with an extension '.obj'.

3. Linking Phase

In this phase, the linker links all the files and functions with the object code under execution. For example, if a user uses a 'printf' function in his/her program, the linker links the user programs object code with the object code of the printf function (available in the 'stdio.h' library). Now, the object code is ready for next phase (i.e., execution phase).

4. Execution Phase

In this phase, the executable object code is loaded into the memory and the program execution begins. We may encounter errors during the execution phase, even though compilation phase is successful. The errors may be runtime errors (example, divided by zero) and logical errors. The process of execution of a 'C' program can be represented by a flowchart as follows,

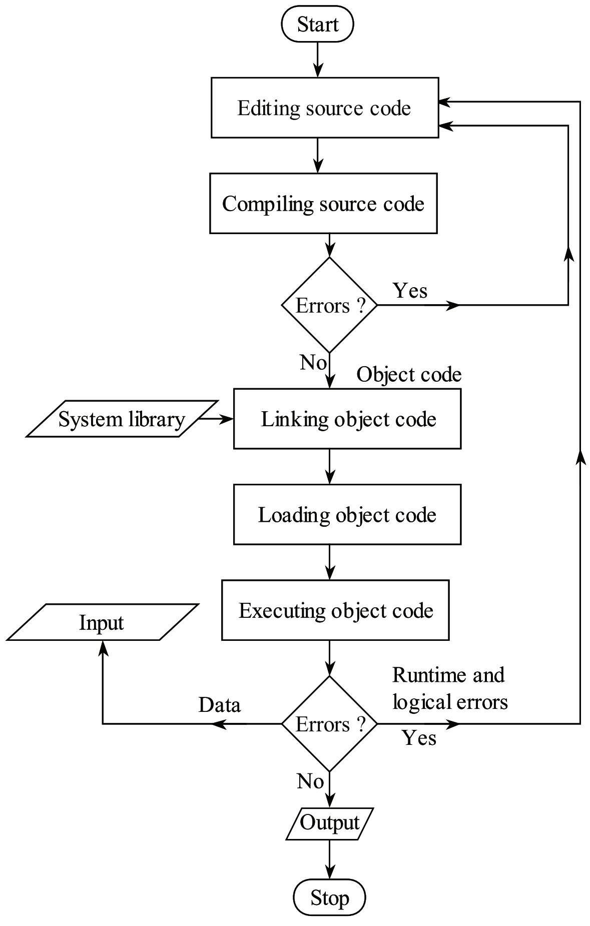


Figure: Flowchart for Process of Compiling and Running a 'C' Program

Q16. Define operating system and discuss the history of operating system.

Answer :

Operating System

An operating system is a program or a collection of programs that controls the computer hardware and acts as an intermediate between the user and hardware. It provides platform for application programs to run on it. It has the following objectives,

(i) Efficiency

An operating system must be capable of managing all the resources present in the system.

(ii) Convenience

An operating system should provide an environment that is simple and easy to use.

(iii) Ability to Evolve

An operating system should be developed in such a way that it provides flexibility and maintainability. Hence, the changes can be done easily.

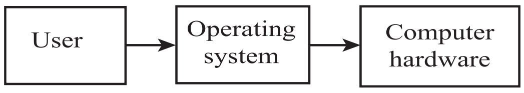


Figure: Operating System as an Interface

History of Operating System

The evolution of operating system is as follows,

❖ **1940's and 1950's**

The digital computers invented in 1940's were capable of inserting the program instructions bitwise through mechanical switches such computers does not carry any operating system. This was then replaced by punched cards and later by assembly languages. And IBM 701 was first successful operating system used in early 1950s. These operating systems were capable of utilizing the computer effectively while executing single process at a time. These systems were typically referred to as single steam batch processing system.

❖ **1960's**

The batch processing systems were invented in 1960's were capable of processing multiple jobs concurrently. These systems allow one process to use a processor and other processes to use other peripherals of the system. This concept leads to the development of multiprogramming environment with which the processor can be switched quickly among multiple processes. Later on in late 1960's this multiprogramming evolved in timesharing and real-time systems which are time-based.

❖ **1970's**

Systems invented in 1970's were capable of providing various computing environments such as batch processing, timesharing and realtime in the form of various applications. They can also provide communication among multiple computers connected over a Local Area Network (LAN). Also personal computing environments began to develop with Apple's Apple-II.

❖ **1980's**

Operating systems invented in 1980's started involving Graphical User Interface(GUI) that uses large set of icons, menus, windows etc. This made the use of computer interactive, interesting and easy to understand. These systems not just act as a personal individual computers but also capable of performing single operation with multiple computers. This concept is known as distributed computing that work on the basis of client/server models.

❖ **1990's**

Computers invented in 1990's were capable of processing thousands of MIPS (Million Instructions per Second) that save the data in gigabytes. Users of these systems were able to store and retrieve data from different remote devices with use of Internet (or) World Wide Web (WWW). Operating systems of these computers provide a wide range of networking features. However, they were prone to network and security threats. Some of the most popular operating systems of 1990's include Microsoft's DOS (Disk Operating System), Windows NT, 95, 98.

❖ **2000 and Above**

Modern computers are now offering extra ordinary features including web services, middleware, parallelism, portability, virtual machines and many more.

Q17. List and explain various types of operating systems.

Answer :

Types of Operating Systems

The different types of Operating Systems (OS) are,

1. Single-user operating system
2. Multi-user operating system
3. Multi-process operating system
4. Batch processing
5. Multiprogramming
6. Time sharing
7. Real time system
8. Distributed system.

1. Single-user Operating System

This operating system allows a single user to perform operation on the computer at a given instance of time. In this type of operating system, a single application rather than multiple application can be executed at one time.

2. Multi-user Operating System

In contrast to single user operating system, this operating system enable multiple users to operate the computer simultaneously. This operating systems perform efficient utilization of CPU by assigning equal amount of time slice to every individual user (connected through different terminals).

3. Multi-process Operating System

In this type of operating system, a single program is processed by multiple CPUs. This operating system is used basically when batch processing has to be supported. The multi-processing operating system is oftenly used because it acts as a backup for the existing CPU.

4. Batch Processing System (1960s)

A batch processing operating system reads a set of separate jobs, each with its own control card. This control card contains information about the task to be performed. Once the job is completed its output is printed. The processing in a batch system does not involve interaction of user and the job during its execution. However, in these systems the CPU was not utilized efficiently due to mismatch in processing speed of mechanical card reader and electronic computer.

Example: MS-DOS

5. Multiprogramming (1970s)

In mono-programming, memory contains only one program at any point of time. Whereas in multiprogramming, memory contains more than one user program.

In case of mono-programming, when CPU is executing the program and I/O operation is encountered then the program goes to I/O devices, during that time CPU sits idle. Thus, in mono-programming CPU is not effectively used i.e., CPU utilization is poor. However, in multiprogramming, when one user program contains I/O operations, CPU switches to the next user program. Thus, CPU is made busy at all the times.

A single user cannot keep CPU busy at all times. Hence, multiprogramming increases CPU utilization by organizing jobs (programs), so that CPU is busy at all times by executing some user program or the other.

The idea in multiprogramming is as follows,

The OS picks one of the jobs from job pool and sends the jobs to CPU. When an I/O operation is encountered in that job, OS allocates I/O devices for that jobs and allocate CPU to next job in the job pool.

However, in mono-programming, CPU sits idle while I/O operation is being performed.

In multiprogramming most of the time CPU is busy. Advantages of multiprogramming are,

(i) CPU utilization is high

(ii) Higher job throughput.

Throughput is the amount of work done in a given time interval.

$$\text{Throughput} = \frac{\text{Amount of time CPU is utilized}}{\text{Total time for executing the program}}$$

Example: Windows

6. Time Sharing System (1970s)

Time sharing is considered as multiprogramming systems logical extension. In time sharing system, the user has a separate program in memory. Each program in time sharing system is given a certain time slot i.e., operating system allocates CPU to any process for a given time period. This time period is known as “time quantum” or time slice. In Unix OS, the time slice is 1 sec i.e., CPU is allocated to every program for one sec. Once the time quantum is completed, the CPU is taken away from the program and it is given to the next waiting program in job post. Suppose, a program executes I/O operation before 1 sec time quantum, then the program on its own releases CPU and performs I/O operation. Thus, when the program starts executing, it executes only for the time quantum period before it finishes or needs to perform I/O operation. Thus, in time sharing any users shares the CPU simultaneously. The CPU switches rapidly from one user to another giving an impression to each user that he has own CPU whereas, actually only one CPU is shared among many users. CPU is distributed among all programs.

In a situation where there are more number of jobs that are ready to be inserted into the memory, when there is not enough memory a decision made to select the jobs among them. This decision making is known as “job scheduling”.

Example: Unix.

7. Real Time System (1980s)

Real time operating systems are time bounded systems, wherein the system must respond to perform a specific task within predefined boundary. There are two types of real time system.

(i) Hard Real Time System

In this real time system, actions must be performed on specified time which could otherwise lead to huge losses. It is widely used in factories and production lines.

Example

In automobile, assembly line welding must be performed on time. This is because a weld before or after the specific instance can damage the product.

(ii) Soft Real Time System

In this real-time a specified deadline can be missed. This is because the level of loss is low compared to hard real time system.

Example

A video game can has voice which is not synchronized to the movie. This is still undesirable but does not causes huge loss.

8. Distributed System

It is a collection of independent, heterogeneous computer systems which are physically separated but are connected together via a network to share resources like files, data, devices, etc. The primary focus of distributed system is to provide transparency while accessing the shared resource i.e., a user should not worry about the location of the data. There are various advantages of distributed systems like they help in increasing computation speed, functionality, data availability and reliability.

Example: Novell network.

Q18. Discuss about the components of operating system.

Answer :

Components/Functions of an Operating System

Operating system is a software program that contains several components. They are,

(i) Process management

(ii) Main memory management

(iii) File management

(iv) Mass storage management

(v) I/O system management.

(i) Process Management

When a program gets executed it is called as process and it requires several resources like CPU time, memory, Input/Output devices, files, etc., for execution. The process management component is responsible for the following activities,

- ❖ Process creation, by allocating memory and other resources to it, and also process deletion by deallocating those resources.
- ❖ Processes can be suspended and can be later resumed.
- ❖ There may be several processes running concurrently in the system. Its responsibility is to provide synchronization and coordination among them.
- ❖ Also provides communication among several processes.
- ❖ Should handle deadlocks.

(ii) Main Memory Management

In order to execute any program, it has to be present in main memory. Hence, main memory is a central resource needed by almost every operation. The main memory management component is responsible for the following activities,

- ❖ Allocation and deallocation of memory space to various processes.
- ❖ Keeping information about the process and memory areas allocated to them.
- ❖ Keeping information about the memory locations that are free.

(iii) File Management

The user stores its data usually in a file which is nothing but a collection of related information. There are several types of files based on the data stored in them, such as text files, binary, executable, image, video and audio files, etc. The operating system's file manager manages all the details regarding their location on secondary memory and provides user with an abstract view and simple interface to access them. The file management component is responsible for the following activities,

- ❖ Creation and deletion of files and directories on user request.
- ❖ Organization of files in various directories.
- ❖ Providing simple GUI or commands to perform operations on files and directories such as read, write, open, close, append files, search files, etc.
- ❖ Storing files on stable storage media like hard disk, tapes, etc.

(iv) Mass Storage Management

Data and programs are usually stored in secondary memory or mass storage devices such as hard disks etc., because they are nonvolatile i.e., when power is lost, data is not lost. All the files of system and users are stored here. Hence, disk management plays a vital role in computer system. The following are the various activities which means storage management component of operating system is responsible for,

- ❖ Allocation and deallocation of memory to various files.
- ❖ Keeping track of free space available.
- ❖ Using disk scheduling algorithms to improve transfer time of data access.

(v) I/O System Management

The input and output devices have several complex hardware specifications, it is the responsibility of operating systems to hide those details and provide user with simple commands like 'read' and 'write' to access these devices. For this purpose operating system uses I/O subsystem which consists of the following,

❖ A component which provides intermediate storage between memory and I/O devices in order to improve access time. These techniques include buffering, caching and spooling.

❖ A generic device driver interface.

❖ A specific device driver interface for each device.

Q19. Write a short note on compilers.

Answer :

A compiler refers to a software that transforms high-level language codes into machine language code. In other words, a compiler transforms the source code into binary code such that an executable program is created. The operations performed by a compiler includes lexical analysis, preprocessing, parsing, code generation and optimization etc. The compiler sometimes generate program faults that are often difficult to track down. Therefore, the compiler implementors focus more on making a consistent compiler.

For remaining answer refer Unit-I, Page No. 1.7, Q.No. 15, Topic: Compilation Phase.

1.2 IDEA OF ALGORITHM : STEPS TO SOLVE LOGICAL AND NUMERICAL PROBLEMS

Q20. Write in brief about solving logical problems.

Answer :

Solving logical problems consists of the below steps,

1. Comprehending the problem
2. Representing the problem in formal terms
3. Planning a solution
4. Executing the plan
5. Interpreting and evaluating the solution.

1. Comprehending the Problem

In this step user need to visualize the situation, identify the problem and problem data. Visualize all the events of the problem and note down statement i.e., problem. Extract all the data related to the problem.

2. Representing the Problem in Formal Terms

Represent the problem in terms of formal concepts and principles such as to which specific area the problem is related. They are useful in simplifying the complexity of the problem.

3. Planning a Solution

Plan a logical solution that are mostly depicted as mathematics. The solution will involve a set of equations. Every equation has a specific goal. One equation might need to solve one a more equations additionally. Finally a logical chain of equations will be created to produce effective logical solution.

4. Executing the Plan

Determine a solution by executing the logical steps. These steps are nothing but the solution planned in the previous step. Execution must be done by applying all the appropriate methods and techniques.

5. Interpreting and Evaluating the Solution

In this step the solution is checked and verified. The solution should directly answer the problem and must not contain unnecessary or unrelated things.

Every step depicted above will use the information extracted from the previous step.

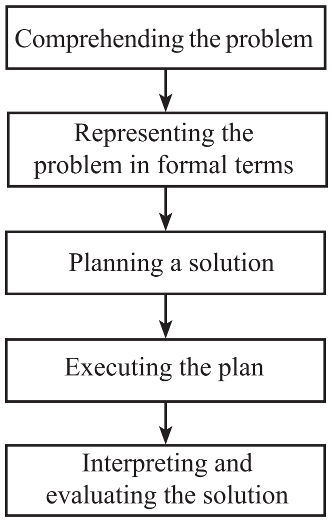

Comprehending the problem
Representing the problem in formal terms
Planning a solution
Executing the plan
Interpreting and evaluating the solution

Figure: Steps to Solve Logical Problems

Q21. Write down the steps to solve numerical problems.

Answer :

Steps for solving numerical problems are as follows,

1. Analyze

Initially determine the starting point and ending point i.e., the inputs, outputs and the units of the output. Develop a plan to solve the problem and get relevant answer. Draw a table or graph to visualize the problem and its data. Even an equation can be used to calculate the result.

2. **Calculate**

   Calculations can be done easily if the planning is done effectively. Some of the problems might need to rearrange the equation or convert the unit measurements.

3. **Evaluate**

   Once the result is calculated, it should also be evaluated. It involves verifying the problem solution whether the data is correctly, whether appropriate equations are used, whether the numbers are correctly rounded off, whether the units and figures are correct. Finally the calculations and checked. Scientific notations must be used in the answer.

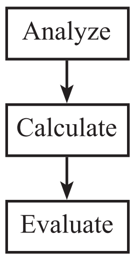


Figure: Steps to Solve Numerical Problems

1.3 REPRESENTATION OF ALGORITHM

Q22. What is an algorithm? Write the various criteria used for judging an algorithm.

Answer :

Algorithm

An algorithm is a method of representing the step-by-step procedure for solving a problem. An algorithm is very useful for finding the right answer to a problem or to a difficult problem by breaking the problem into simple cases.

Properties of Algorithm

An algorithm must possess the following properties.

1. Finiteness
2. Definiteness
3. Effectiveness
4. Generality
5. Input/output.

1. **Finiteness**

   An algorithm should terminate in a finite number of steps.

2. **Definiteness**

   Each step of the algorithm must be precisely stated.

3. **Effectiveness**

   Each step must be effective, in the sense that, it should be easily convertable into program statement and can be performed exactly in a finite amount of time.

4. **Generality**

   The algorithm should be complete, so that it can be used to solve all problems of a given type for any input data.

5. **Input/Output**

   Each algorithm must take zero, one or more quantities as input data and yield one or more output values.

An algorithm can be written in English like sentences or in any standard representations. Sometimes, algorithm written in English like language is called **Pseudo-code.**

Example

Suppose, we want to find the average of three numbers, the algorithm is given as follows.

```
Step-1: Read three numbers a, b and c
Step-2: Compute the sum of a, b and c
Step-3: Divide the sum by 3
Step-4: Store the result in variable d
Step-5: Print the value of d
Step-6: End the program.
```

Advantages of Algorithm

1. It is easily understandable as it is written in english like language.
2. It acts as a blueprints in the developments of program.
3. It easily detects the errors in the program.
4. It efficiently maintains the software.

Disadvantages of Algorithm

1. It consumes more time for solving larger and complex problems.
2. It is difficult to understand when the logic is complex.

1.3.1 Flowchart/Pseudocode With Examples

Q23. What is a flowchart? Explain the different symbols used in a flowchart.

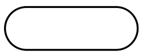
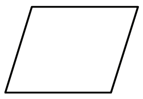
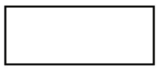
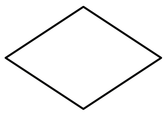
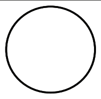
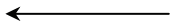
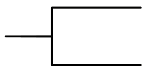
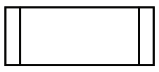
Answer :

Flowchart

Flowchart is a diagrammatic representation of an algorithm. It is built using different types of boxes and symbols. The operation to be performed is written in the box. All symbols are interconnected by arrows to indicate the flow of information and processing.

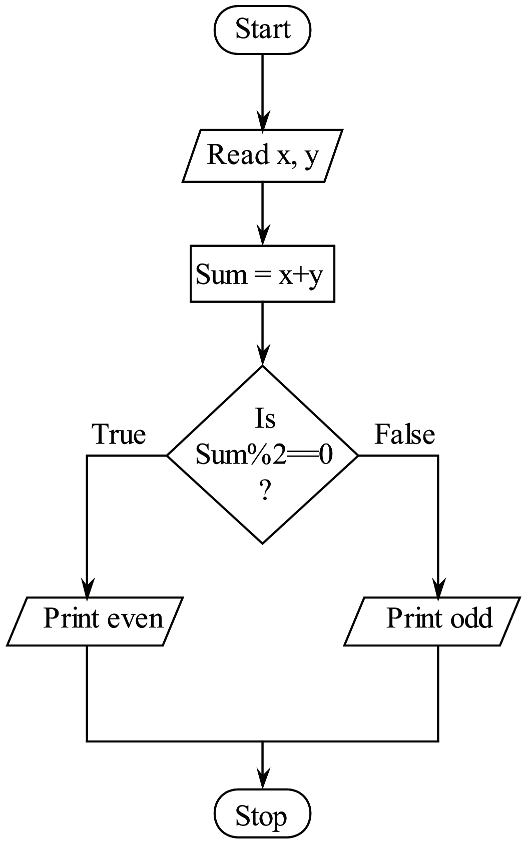
Symbols in Flowchart

Following are the standard symbols used in drawing flowcharts,

Oval		Terminal	start/stop/begin/end symbol
Parallelogram		Input/output	Making data available for processing (input) or recording of the processed Information (output)
Rectangle		Process	Any processing to be performed. An assignment operation normally represented by this symbol
Diamond		Decision	Decision or switching type of operations that determines which of the alternative paths is to be followed
Circle		Connector	Used for connecting different parts of flow chart.
Arrow		Flow	Joins two symbols and also represents executions flow.
Bracket with broken line		Annotation	Descriptive comments or explanations
Double sided rectangle		Predefined process	Modules or subroutines given elsewhere

Example

The flowchart to find whether the sum of two numbers is odd or even is as follows,



Q24. How algorithm is different from flowchart? Write an algorithm and draw flowchart for finding greatest among three given numbers.

Answer :

Algorithm

Algorithm is a step-by-step process of solving a particular problem. It is basically, a series of steps in a computer program which provides the solution for a particular problem.

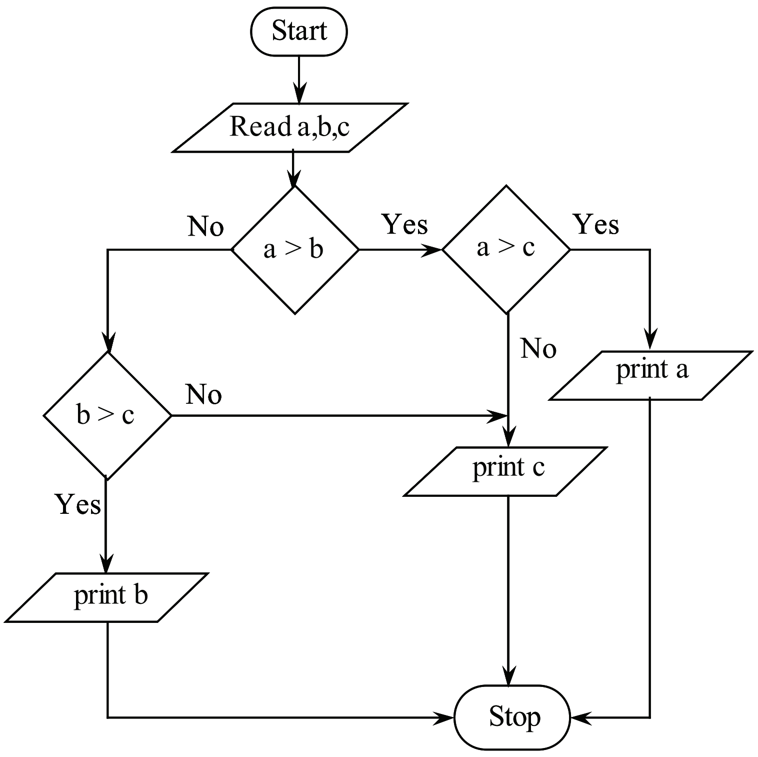
Algorithm to find greatest among three given numbers is given below.

```
Step-1: Read three numbers a, b and c
Step-2: Check if (a > b) and (a > c) then
          print a
          else if (b > c) then
          print b
          else
          print c
Step-3: Stop
```

Flowchart

Flowchart is the graphical representation of an algorithm. Earlier, flowcharts were considered as a very crucial part on the process of programming. It is an important tool for planning and designing a new system.

Flowchart for the above algorithm is given below.



Q25. Write a short note on pseudocode.

Answer :

Pseudocode

Pseudocode is a way of describing general structure of program in simple English language. The ward pseudo means imitation and code means instructions written in some programming language. Pseudocode describes the complete logic of some program so that implementation becomes easy. It does not use any syntax or symbols. It uses only English-like statements.

Pseudocode to Add Two Number

```
1.  BEGIN
2.  NUMBER x, y, sum
3.  OUTPUT("Input number1:")
4.  INPUT x
5.  OUTPUT("Input number2:")
6.  INPUT y
7.  sum = x + y
8.  OUTPUT sum
9.  END
```

Pseudocode to Calculate Area and Perimeter of Rectangle

```
1.  BEGIN
2.  NUMBER len, br, area, perimeter
3.  INPUT len
4.  INPUT br
5.  area = len*br
6.  perimeter = 2 *(len + br)
7.  OUTPUT area
8.  OUTPUT perimeter
9.  END
```

Q26. Differentiate between algorithm and flowchart.

Answer :

Algorithm		Flowchart	
1.	Algorithm is a step-by-step procedure to solve a problem.	1.	Flowchart is the pictorial/diagrammatic representation of the an algorithm.
2.	It defines the instructions in simple English language.	2.	It makes use of various boxes in order to depict the instructions defined by the algorithm.
3.	It is suitable for larger and modular programs.	3.	It is suitable for smaller programs.
4.	It is easier to understand.	4.	It is easier to understand if symbols are known.

1.3.2 From Algorithms to Programs

Q27. What are the different steps followed in the program design?

Answer :

Program Design

Computers play a major role in solving a problem. This is because of the good communication that exists between computers and their corresponding users. In order to write a program, a process called 'programming' is required. But, before writing this, the problem to be solved should be analyzed and need to be prepared by generating a step-by-step procedure. The sufficient details about the problem is obtained. This process is referred to as 'problem analysis'.

While developing a program the following steps are required,

1. **Problem Specification**

   The problem should be studied and understood well. The programmer should have the knowledge about, what should be done in order to begin the first step. A careful examination about the following is required to determine what has to be done next.

(a) Inputs provided

(b) Questions asked

(c) Outputs specifications type

(d) Manual processing.

The information obtained during the problem specification phase is due to the input, output and some other processing specification.

2. Outlining the Solution

In this step the problem to be solved is identified using solution method. The chosen method is a path that defines, what is given and what is required to solve the problem. There may be many solution methods available, but the following conditions need to be considered before selecting a correct/appropriate solution method.

(a) The time needed to solve the problem

(b) Vulnerability to errors.

For some problems, solutions may not exist. Hence, a new solution method need to be developed or the existing solution should include some exceptions while solving the problem. Tools such as structure charts pseudocode and flow charts help in developing a solution.

3. Choosing and Representing Algorithm

Once the appropriate solution method has been defined, it is necessary to convert that method into an algorithm, which is a means of representing the solution for problem solving. These algorithms can be written using the common language or by using the standard notations.

4. Programming or Coding the Algorithm

In this step, the algorithm written is converted into a program. Programs can be written in any programming languages depending on the problems behavior and also on the availability of those languages.

5. Removing Errors

In this step, the errors generated while developing a program are detected and then corrected. Program verification is a process that assures the correctness of a program and debugging is a mechanism that detects and removes the errors.

The following are the three different types of errors generated while developing a program.

(a) Syntax Errors

These errors are generated when the statements are not written according to the syntax or grammatical rules of a language.

(b) Run-time Errors

These errors are generated when the program (which is free from syntax errors) is being executed.

Some of the most common runtime errors are,

(i) Dividing by zero

(ii) Logarithm of a negative number

(iii) Square root of a negative number

(iv) Range out of bounds etc.

(c) Logic Errors

These errors are generated when there are logically incorrect statements in the program. Such errors are very difficult to be detected, because they are found after examining the output.

6. Testing and Validation

After writing a program, it must be tested and validated. It should always produce correct and appropriate results.

Though the conversion of an algorithm into a program is correctly done, a question related to algorithm arises to determine, whether it correctly solves the problem. By a variety of conditions called test cases, the program can be validated. Hence, the selection of test data sets is a major factor which is done through a better understanding of various logical paths present in the program. These logical paths are determined with the help of well-structured programs. The test data sets are classified into four different cases,

(a) Valid Cases

These kind of test cases represent the data sets which are logically related to the program. Hence, correct results must be generated for these type of cases. This information should be selected in such a way that every logical flow path must contain at least one test case.

(b) Boundary Cases

These type of test cases are derived from the valid cases. They signify some of the possible values through which decisions can be made.

(c) Special or Uncommon Cases

These type of test cases represent the data sets that do not complete their normal processing and require some special methods to complete their execution.

(d) Invalid Cases

These type of cases represent some situations which do not provide any meaning and which deny the program statement. Generally, they should not occur during the processing of a program.

7. Documentation and Maintenance

The information about the program is collected, organized and maintained through a process called documentation.

Any change in the program should be done according to the requirements. This process is known as program maintenance.

The programs must be written in a clear and easy way, so that the other users who use the program may not find any difficulty.

In order to achieve this, the program should be maintained well by using the following,

(a) Proper comments.

(b) Proper indentation (which is a coding style used for the better understanding and debugging of the program) for improving the readability and clarity.

(c) Detail description about the different methods used, etc.

The following are the two types of documentation,

(i) Operational Documentation

This kind of documentation provides the details about the following.

(a) Input and output formats

(b) Operating instructions

(c) Various user interactions with the program and their limitations.

(ii) Technical Documentation

This kind of documentation provides the details about the following,

(a) Design aspects

(b) Explanations of the different methods used

(c) Problem occurred

(d) Logic present

(e) Hardware to be operated, etc.

The system can be maintained by providing the documentation of a program in a correct way. The program documentation is referred to as a continuous process that begins from,

(i) Specification of a problem

(ii) Representation of the algorithm and

(iii) Program writing.

Q28. Discuss in brief the steps involved in problem solving. Define algorithm and program.

Answer :

Steps Involved in Problem Solving

Problem solving is a method of solving a computer problem by following a sequence of steps. Following are some steps to be followed while solving a problem.

For remaining answer refer Unit-I, Page No. 1.15, Q.No. 27, Topics: Problem Specification, Outlining the Solution, Choosing and Representing Algorithm, Programming or Coding the Algorithm.

Algorithm

For answer refer Unit-I, Page No. 1.12, Q.No. 22.

Program

Program is a set of instructions developed from an algorithm. It is written in computer understandable language to accomplish certain task. Computers play a major role in solving a problem. This is because of the good communication that exists between computers and their corresponding users. In order to write a program, a process called 'programming' is required. But, before writing this, the problem to be solved should be analyzed and needs to be prepared by generating a step-by-step procedure. The sufficient details about the problem should be obtained. This process is referred to as 'problem analysis'.

1.3.3 Source Code, Variables (With Data Types) and Memory Locations

Q29. Write in short about source code.

Answer :

Source code is the part of program that can be read and understood easily. It can be referred as the text file into which the user types a set of C language statements. The source code will be transformed into the machine language by the compiler. This will be in machine understandable language. Once the source code is compiled, the file containing the output is referred as object code.

The object code will be in zeroes and one's. It will be linked to create an executable file that performs particular program functions.

Source code can be created using text editor, visual programming tool or IDE (Integrated Development Environment). In certain large program development environments, there are management systems to help the programmers to track the states and levels of source code files.

An example of source code in C language is as follows,

```
#include <stdio.h>
main( )
{
printf("Hello world");
}
```

Q30. What is variable? How can variables be characterized? Give the rules for variable declaration.

Answer :

Variable

A variable is the name given to the memory location, where the data value (i.e., value assigned to the variable) of the variable is stored. Unlike constant, the value of a variable can be changed during execution. A variable can be classified as integer variable, real variable, character variable depending on the type of value stored in it. A variable name must be chosen in such a way, that it improves the readability of a program and also reflects its function.

Rules for Variable Declaration

The rules for writing the variable names are given below.

- Variable names consist of letters, digits and underscore.
- Variable names must begin with an alphabet or underscore.
- Variable names could have length upto 31 characters.
- Variable names are case sensitive i.e., 'SUM' and 'sum' are not equivalent.
- Keywords should not be used as variable names.
- Blank spaces, commas and special symbols (except underscore) are not allowed within variable names.

Example

cp_int, hr_gs, salary, *x*, *y*, *z*, roll_1231, etc.

Variable Declaration

A variable stores data value of different types (i.e., integer, real, character) and to avoid the confusion to compiler, variables are declared with a data type. If a variable is declared with a data type, it can store only the data values of the type associated with it.

Syntax

data_type variable_name = data_value;

Example

int *x* = 10;

Thus, '*x*' is a variable that can store only integer values.

Q31. Write short notes on scope of variables.

Answer :

Scope of a variable is defined as a vicinity or space within a program, where value of a given variable can be accessed. Hence, the effect of a variable can be felt only within the block where the variable has been declared.

Similarly, scope of a function can be referred to as part of a program where function can be accessed.

Scope can be classified into two types. They are,

1. Local scope
2. Global scope.

1. Local Scope

(i) The variables which are declared within any function are called local variables.

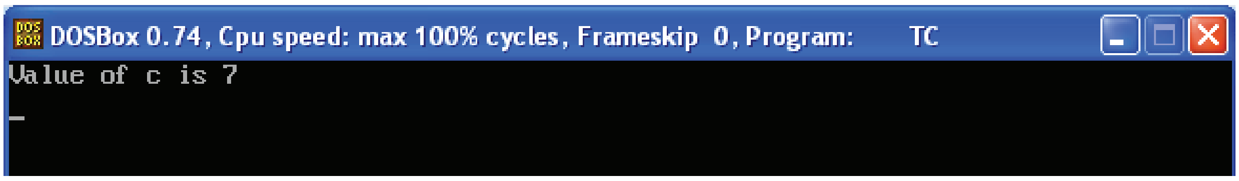
(ii) These variables can be accessed only by the function in which they are declared.

(iii) Default value for local variable is a garbage value.

Example

```
# include< stdio.h >
void main( )
{
void display( ) // Local function
{
int c; // Local variable
c = 7;
printf("Value of c is %d \n", c);
}
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Value of c is 7
_
```

2. **Global Scope**

(i) The variables which are declared outside any function are called global variables.

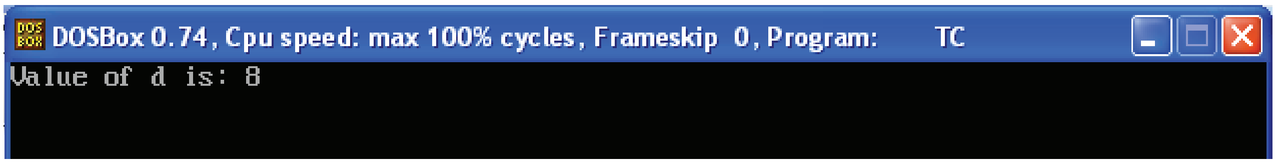
(ii) These variables can be accessed by all the functions in the program.

(iii) Default value for global variable is zero.

Example

```
#include< stdio.h >
int d ; //Global variable declaration
void display( ); //Global function declaration
void main( )
{
d = 8; // Assignment of global variable 'd'
void display( ) //Global function called by main
{
printf("Value of d is: %d \n", d);
}
void display( )
{
printf( " %d", d); // Global variable is accessible here also.
}
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Value of d is: 8
```

Q32. Name the different data types that C supports and explain them in detail.

Answer :

Data Type

A data type is used to indicate the type of data value stored in a variable. The data type of a variable is specified at the time of its declaration and is attached till the end of execution of the program.

In general, data type associated with variable indicates,

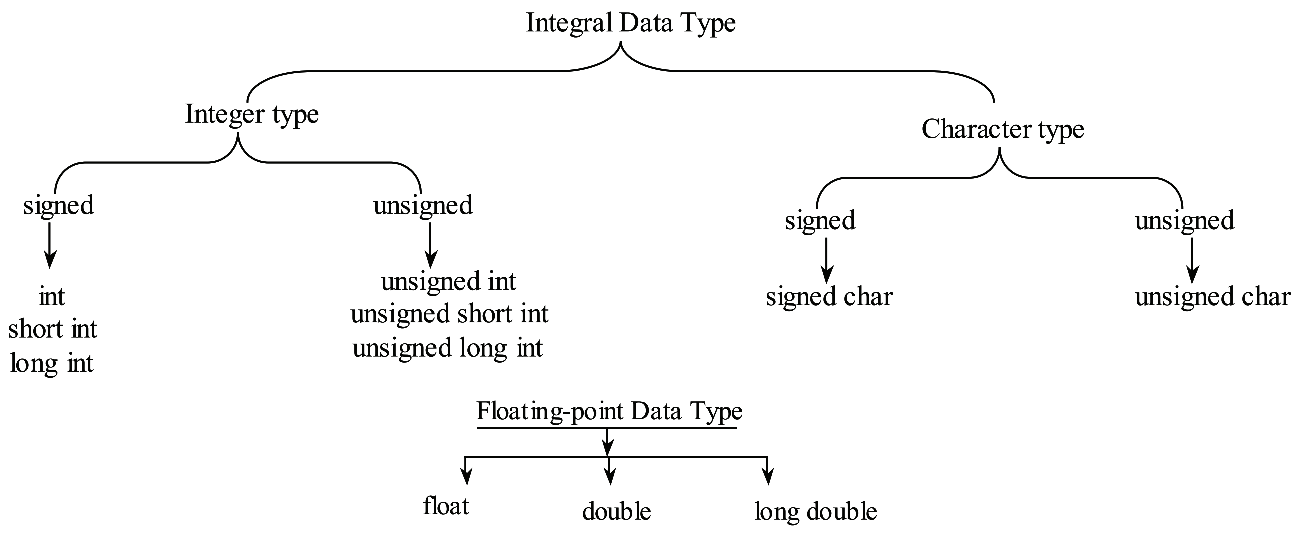
- ❖ Type of value stored in the variable
- ❖ Amount of memory (size) allocated for data value
- ❖ Type of operations that can be performed on the variable.

Data Types in C

'C' supports four basic data types (i.e., int, float, char, double) which can be grouped under two groups of data types i.e.,

1. Integral data type
2. Floating point data type.

The primary data types supported by 'C' compilers can be grouped as shown in figure below,



Figure

1. **Integral Data Types**

Integral data types are used to store whole numbers and characters. It can be further classified as,

(i) Integer data type

(ii) Character data type.

(i) **Integer Data Type**

The integers do not contain the decimal points. They take the binary form for storage. The size of the integer depends on the word length of a machine i.e., it can be either 16 or 32 bits. On a 16-bit machine, the range of integer values is –32768 to +32767.

'C' provides control over the range of integer values and the storage space occupied by these values through integers of types 'short int', 'int', 'long int' in both 'signed' and 'unsigned' form.

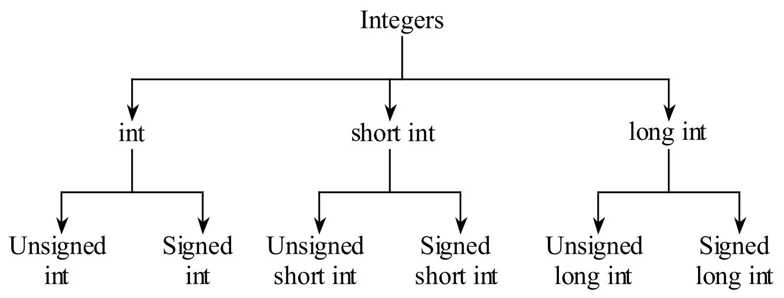
(a) **Signed Integers**

Signed integers are those integers which use 15 bits for storing the magnitude of a number and 1 bit to store its sign. The left-most bit i.e., 1st bit is used for storing the sign, it will be '1' for a negative number and '0' for the positive number.

(b) Unsigned Integers

Unsigned integers are those integers that uses 16 bits for storing positive whole numbers and sign and the sign bit (left-most bit) also contains the value.

Classification of Signed and Unsigned Integers Based on the Memory and Range



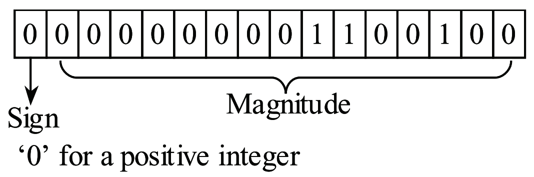
Data Type	Size (Memory)	Range
int	16 bits or 32 bits	–32768 to +32767 or –2147483648 to +2147483647
unsigned int	16 bits	0 to 65535
signed int	16 bits	–32768 to 32767
short int	16 bits	–32768 to 32767
unsigned short int	16 bits	0 to 65535
signed short int	16 bits	–32768 to 32767
long int	32 bits	–2147483648 to 2147483647
unsigned long int	32 bits	0 to 4294967295
signed long int	32 bits	–2147483648 to 2147483647

Table (1): Integer Data Types

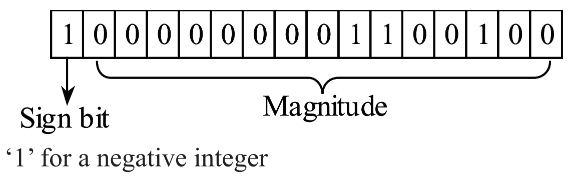
Examples

Consider a 16-bits (word length) machine, the statement signed int x = 100; occupies 16-bits of storage for the binary representation of x, the first bit is reserved for sign and the remaining 15-bits for magnitude.

(a) signed int x = 100;



(b) signed int x = – 100;



(ii) Character Data Type

This data type indicates that the value stored in the associated variable is a character. Each character has an equivalent ASCII value (example '*A*' has an ASCII value of 65)

These characters are internally stored as integers. 'char' data type occupies one byte of memory for the storage of character. The qualifiers signed and unsigned can be applied on 'char' data type.

The size and range of values for character data types are shown in table (2).

Character Data Type			
Type	Size (Byte)	Range	Format specifier
char (or) signed char	1	–128 to 127	%c
unsigned char	1	0 to 255	%c

Table (2): Character Data Type

2. Floating-point Data Types

Floating-point data types are used to store real numbers (i.e., decimal point numbers). The keyword 'float' is used to declare a variable holding a floating-point number. The floating-point numbers with 6 digit of precision occupies 4 bytes for storage. For greater precision, we can use the 'double' data type (i.e., 14 digits of precision) for further precision, we have long double.

The size and range of values for floating-point data types are given in table (3) below.

Floating-point Data Type			
Type	**Size (bytes)**	**Range**	**Format Specifier**
float	4	–3.4E –38 to 3.4E + 38	%f
double	8	–1.7E – 308 to 1.7E + 308	%lf
long double	10	– 1.7E – 4932 to 1.7E+4932	%lf

Table (3): Floating-point Data Type

1.3.4 Syntax and Logical Errors in Compilation

Q33. Write in short about syntax errors and logical errors.

Answer :

Syntax Errors

Syntax errors are generated when the statements are not written according to the syntax or grammatical rules of a language. These type of errors are detected by the complier during compilation process. Hence, it is also called as compile-time error.

If a program contains a syntax error, then the compiler fails to compile it and the program gets terminated after displaying a list of errors with their corresponding line number in which they have took place.

Example

```
main( )
{
int a, b;
myfun( )
{
    a = 2;
    b = 3;
    sum = a + b;
}
```

In the above code, there is no closing curly braces for the function. Hence, on the execution of the code nothing will be displayed on the screen.

Logical Errors

These errors are generated when there are logically incorrect statements in the program. Such errors are very difficult to be detected, since they are formed after examining the output.

Example

A program containing an infinite loop that runs forever is a good example of logic errors. This will lead to hangup of the program. An example code for infinite loop is,

```
for(var i = 2; i > 1; i ++)
{
    //This loop will run forever
}
```

1.3.5 Object and Executable Code

Q34. Explain about object code.

Answer :

Object Code

Machine language is also known as machine code or object code. In this language, programs are written in the form of binary digits or bits with different patterns relating to different commands that are easy to be read and interpret by the computer. The organization of bit-patterns depends mostly on the machine code specification.

It is necessary to have a unique and distinct bit pattern associated with every CPU instruction so as to differentiate those instruction from other instruction set. These instructions are said to be coded because the differentiation between the two unique instruction sets is made possible only if every individual instruction in the instruction sets is encoded as a unique bit pattern.

Machine language is the only language that is easily understood by the machine. This language is basically a system of instructions as well as data that is executed directly by the specific processor unit. It is considered as the low-level or primitive programming language. The time taken for execution is very less since this language doesn't require any interpreter or compiler to translate the program.

The machine code or instruction set is CPU dependent i.e., every CPU has its independent object code. Successor processor comprises of all the instructions associated with the predecessor. In addition, these processors may also include additional instructions. Often, the design of the processor will change the exact meaning of the instruction code associated with the predecessor. Due to this, it becomes very difficult for the machine code to migrate between two different processors. It is not possible to execute machine code or object code on the system having same processor design but different operating system and memory arrangement. This is because, the machine code has no information about the system's configuration.

Advantages

1. It is easily understood by the CPU.
2. The time taken for executing the program is very less.
3. The computational speed is very high.

Disadvantages

1. It is very difficult for a human to understand it as it comprises of sequence of bit patterns.
2. The length of programs written in machine language is very large.
3. It is difficult to correct or modify the programs.
4. It is a machine dependent language.

Q35. Write short notes on executable code.

Answer :

Executable code contains the machine code instructions for the CPU. They represent some tasks in encoded form to be performed by the computer. A file or scripting language file can be considered to be executable. Executable code is obtained through a sequence of steps.

Initially the user need to type the source code into the file and save it in the computer.

The preprocessor will read the source code and modifies it. In the next phase the compiler will compile the code i.e., it translates the source code into the machine language instruction. It shows the syntax errors available in the program. The object code is stored in the object file. In the next phase, the linker will combine the object file with respective library routine to generate executable file. This file will contain machine instructions or executable code. These instructions indicate what tasks a computer must perform. This complete process is depicted in the below figure,

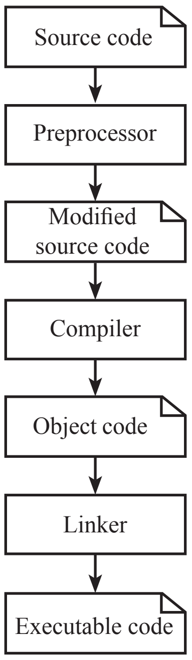


Figure: Conversion of Source Code into Executable Code

UNIT 2

CONTROL STRUCTURES AND ARRAYS

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Explain about if statement in C.

Answer :

if Statement

The 'if' statement is used for decision making. It allows the computer to first evaluate the condition and depending on its resultant value transfer the control to a particular statement in the program.

This statement performs an action if the condition is true, otherwise it skips that action and executes other statement.

Syntax

```
if (condition)
  {
     statement-1;
  }
     statement-next;
```

Here, if the condition is satisfied, then statement-1 is executed followed by statement-next. Otherwise, statement-1 block is skipped and only statement-next is executed.

Q2. Explain in brief about the 'else if'. Give its syntax.

Answer :

'else if' Statement

This statement is an extended form of simple 'if' statement.

Syntax

```
if (condition)
 {
 statement-1;
 }
 else
 {
 statement-2;
 }
 statement-next;
```

In the above syntax, statement-1 will be executed followed by statement-next when the condition is true. Otherwise, statement-2 will be executed followed by statement-next.

Q3. Give the syntax of switch statement with example.

Answer :

'switch' Statement

A 'switch' statement is used in a program containing multiple decisions, hence known as multiway decision making statement.

Syntax

```
switch(expression)
 {
     case constant1: statement-1;
                     break;
     case constant2: statement-2;
                     break;
     ⋮
     case constant n: statement-n;
                     break;
     default: default-statement;
              break;
 }
 statement next;
```

Example

```
switch(Animals)
{
     case('Herbivorous'):
          printf("These are Herbivorous animals");
          break;
     case('Carnivorous'):
          printf("These are carnivorous animals");
          break;
     case('Omnivorous'):
          printf("These are Omnivorous animals");
          break;
     default: printf("These are not animals");
}
```

Q4. Give the syntax of 'for loop' with simple example.

Answer :

for Loop

'for' statement executes for specific number of times. After each iteration it increases by a specified value.

Syntax

```
for (initializing expression; testing expression; updating expression)
 {
 statements;
 }
```

Here the initialization expression specifies the initial value. The testing expression is a condition that is to be tested at each pass. Updating expression is an unary expression that is either incremented or decremented to change the initial value.

Program

```
#include<stdio.h>
#include<conio.h>
void main( )
{
   int i;
   clrscr( );
   for (i = 1; i<=10; i ++)
   printf("SIA PUBLISHERS AND DISTRIBUTORS \n'');
   getch( );
}
```

Q5. What is meant by operator precedence?

Answer :

Each operator in 'C' has a precedence associated with it. This precedence determines, the evaluation of expressions containing more than one operator. There are various levels of precedence. Each operator confines to one of these levels. The operators at the higher level of precedence are evaluated first and the operators of the same precedence are evaluated either from left to right or from right to left, based upon the level.

Q6. Give the rules of precedence and associativity.

Answer :

Precedence Rule

Precedence rule decides the sequence in which the operators must be evaluated. Each operator in C has a precedence associated with it.

Associativity Rule

Associativity rule decides the sequence of using operators multiple times in the same level.

Q7. Explain how to declare an array with example.

Answer :

Array

It is a variable which is capable of holding fixed values in contiguous memory locations. The general format of an array is,

a_0	a_1	a_2	..	..	..	..	a_n

Declare an Array

An array can be declared in the following manner,

```
type array_name[size];
```

Here 'type' represents the data type of array being defined, 'array_name' is the name of the array. The subscript at the end contains a parameter called 'size' which specifies the maximum number of elements that can be stored in an array.

Example

```
int xyz[8];
```

Here, xyz is an array of 8 integers.

Q8. What is a one-dimensional array?

Answer :

One-dimensional Array

One-dimensional array is a simple array in which elements are stored in contiguous memory locations. Array elements are accessed using one subscript.

Syntax

```
datatype array_name [size];
```

Example

int *a*[4];

Here, *a* is an array of 4 integers.

Q9. Define two-dimensional array.

Answer :

Two-dimensional Array

A two-dimensional array is an array of two dimensions. It stores data in the form of rows and columns.

When the data must be stored in the form of a matrix we use two-dimensional arrays.

Syntax

```
type array_name[row_size] [column_size];
```

Example

```
int a[5][3];
```

Above declaration represents a two-dimensional array consisting of 5 rows and 3 columns. So the total number of elements which can be stored in this array are 5 × 3 i.e., 15.

Q10. Define string. Give two examples.

Answer :

String

A string is defined as a series of characters. In C, strings are classified into two types. They are,

1. Fixed-length strings
2. Variable-length strings.

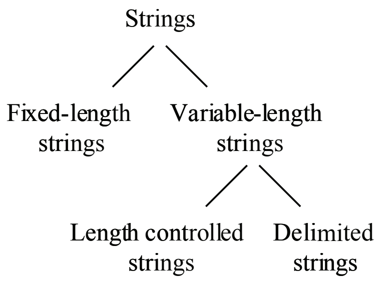


Figure: Classification of Strings

Example 1

char c[5];

Example 2

char str[ ] = "welcome";

Q11. List out some commonly used string handling functions along with syntax.

Answer :

The following are the commonly used string handling functions,

1.	strcat( )	: Concatenates two strings.
	Syntax	: strcat(string1,string2);
2.	strcmp( )	: Compares two strings
	Syntax	: strcmp(string1, string2);
3.	strcpy( )	: Copies one string into another
	Syntax	: strcpy(string1, string2);
4.	strlen( )	: Finds the length of a string
	Syntax	: strlen(string1);
5.	strcmpi( )	: Compares two strings without case sensitivity
	Syntax	: strcmpi(string1, string2);
6.	strncmp( )	: Compares two portions of substrings upto specified position
	Syntax	: strncmp(string1, string2, position);
7.	strncmpi( )	: Compares two portions of strings upto specified length without case sensitivity
	Syntax	: strncmpi(string1, string2, position);

PART-B
ESSAY QUESTIONS WITH SOLUTIONS

2.1 CONTROL STRUCTURES

2.1.1 Arithmetic Expressions and Precedence

Q12. What is an arithmetic expression? What kind of information is represented by an expression?

Answer :

Arithmetic Expression

An expression consists of a single entity such as a constant, a variable, an array element or a reference to a function or can be a combination of such entities joined together using one or more operators.

An expression can also represent logical condition, that is either true or false, which can be represented using integer values like '1' and '0' respectively.

Example

Some of the valid expressions of 'C' are,

1. c = a + b* d;
2. x <= y;
3. x = y;
4. x = =y;
5. x += y;

As seen from the above example, each valid expression in 'C' is terminated using a semicolon ';', and each expression reveals its own meaning. Here is the description of above mentioned expressions.

❖ Expression (1) corresponds to homogeneous expression, consisting of four operands (*c*, *a*, *b*, *d*), three operators (=. +, *) and a semicolon. The assignment operator (=) indicates, that the result of the expression (which is present towards the right side of the assignment operator) is assigned to a variable *c*. The execution of homogeneous expression follows BODMAS principle.

❖ The expression (2) is an conditional expression, where a specified condition is matched between the operands *x* and *y* i.e., it checks whether *x* is less than or equal to (<=) *y*.

❖ The expression (3) is an assignment expression where the value of *y* is assigned to a variable *x*.

❖ The expression (4) is also a conditional expression, where a condition is verified (i.e.,) whether the value of *x* is equal to the value of *y* is determined.

❖ Expression (5) corresponds to addition as well as assignment expression. Hence, the expression $x += y$ is equivalent to $x = x + y$, where the values of *x* and *y* are added and the result is assigned to *x*.

Apart from the above mentioned expressions, 'C' supports certain complex expressions like,

x * y – c * d;

5 * z * p / (k + 1) + h/6*(a + b);

(x + y + z) / (p + q);

(p + q) * (m + n); etc.

All these expressions follow the BODMAS principle for their execution.

Q13. What is meant by operator precedence? What are the relative precedence of arithmetic operators?

Answer :

Operator Precedence

Each operator in 'C' has a precedence associated with it. This precedence is used to determine, how each expression involving more than one operator is evaluated. There are distinct levels of precedence and an operator may belong to one of the levels. The operator at the higher level of precedence are evaluated first and the operators of the same precedence are evaluated either from left to right or from right to left, depending upon the level.

Precedence of Arithmetic Operators

An arithmetic expression involving parenthesis will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C.

1. High priority arithmetic operators are *, /, % and
2. Low priority arithmetic operators are +, –.

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

Example

Consider the following example,

x = a – b/3 + c * 2 – 1;

if a = 9, b = 12, c = 3

Then the evaluation of expression is as follows,

First pass

Step 1 : x = 9 – 12/3 + 3 * 2 – 1

Step 2 : x = 9 – 4 + 6 – 1

Second pass

Step 3 : x = 5 + 6 – 1

Step 4 : x = 11 – 1

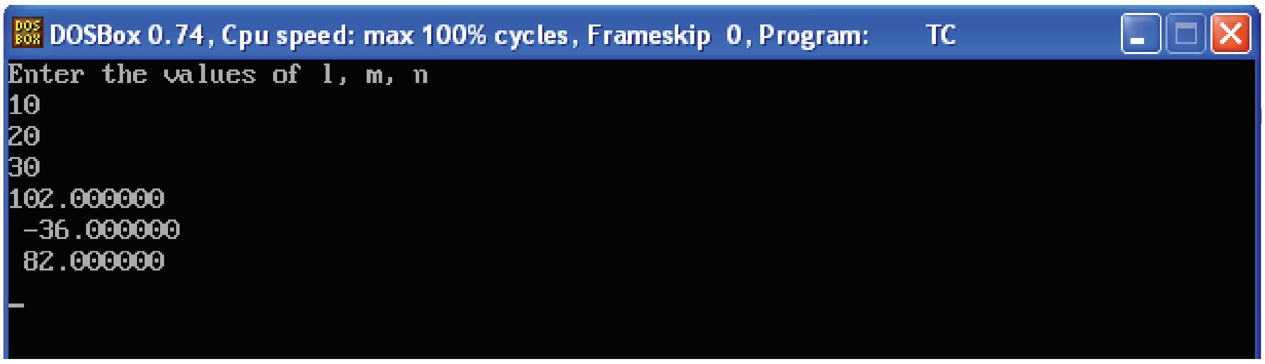
Step 5 : x = 10

The variables a, b, c when used in the program must be defined first before using them in the expressions.

Program

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int l, m, n;
    float p, q, r;
    clrscr( );
    printf("Enter the values of l, m, n\n");
    scanf("%d %d %d", &l, &m, &n);
    p = 2 + m * 5 – n / 3 + l;
    q = 2 + m * (5 – n) / (3 + l);
    r = (2 + m * 5) – (n / 3 + l);
    printf ("%f\n %f\n %f\n", p, q, r);
    getch( );
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the values of l, m, n
10
20
30
102.000000
 -36.000000
 82.000000
_
```

Q14. What is the associativity of arithmetic operators?

Answer :

Associativity of Arithmetic Operators

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 i.e., left to right or right to left. Left to right associativity means that the left operand must be unambiguous. Unambiguous in the 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 not be involved in any other sub-expression.

Example

Consider the expression,

a = 3/2 * 5;

Here, there is a tie between two operators of the same priority i.e., between / and *.

This tie can be settled by using the associativity of * and /. But, both of them enjoy left-to-right priority.

Operator	Left	Right	Remark
/	3	2 or 2 * 5	Left operand is unambiguous, right is not.
*	3/2 or 2	5	Right operand is unambiguous, left is not.

Since, both '/' and '*' have left to right associativity and only '/' has unambiguous left operand, it is performed earlier.

Q15. Explain the process of evaluating expressions in C.

Answer :

In C, there are two types of expressions.

(i) Expression consisting of no side effects

(ii) Expression consisting of side effects.

The process of evaluating both these expressions is different from one another. At the time of evaluating the expression, if a variable is changed more than a single time, then the result generated is an undefined value.

(i) Expression Consisting of No Side Effects

In order to know the process of evaluating an expression that has no side effect, consider an expression without side effects.

3 + 7 * x – y/2

Let the values of the variables x and y be 1 and 4 respectively. As there are no side effects, the above expression can be evaluated as follows,

(i) Initially, the variables in the expression are substituted with their respective values. Hence, the expression becomes,

3 + 7 * 1 – 4/2

(ii) The highest precedence operators i.e., * and / are evaluated first and then the values are replaced with the resulting values below.

3 + (7 * 1) – (4/2)

The resulting expression is,

3 + 7 – 2.

(iii) Finally, step (ii) is repeated till a single value is attained as a result.

Since there is only a single precedence i.e., addition and subtraction, the final value is calculated as 10 – 2 = 8. Also, the values of the variables remain unchanged (i.e., the values of the variables before and after evaluation are equal) because the expression contains no side effects.

(ii) Expression Consisting of Side Effects

Consider the following expression that has side effects in order to know the process of evaluating expressions with side effects.

(4 – *x*) * *y*++ / – – *z* + 3

Let the value of *x*, y, z be 3, 8 and 3 respectively.

As there are side effects the above expression can be evaluated as follows,

1. The parenthesized expression i.e., (4 – *x*) is evaluated first because parentheses has the highest precedence than the other operators in the expression. This evaluation is done by substituting the value of *x* in the expression. Thus, the expression becomes,

   1 * y++/ – –z + 3

2. The postfix expression i.e., y++ that has the next highest precedence is computed. In postfix expression, the value of y++ is the incremented value of *y* obtained after the evaluation. But, before evaluating the expression, the value of *y* ++ does not change i.e., it remains same as the value of *y*. Thus, the expression is,

   1 * 8 / – –z + 3

3. The prefix expression i.e., –– *z* that has the next highest precedence is computed. In prefix expressions, the value of – – *z* is the decremented value of *z* before the evaluation. Thus, the expression is,

   1 * 8 / 2 + 3

4. The multiply and division operators have the next priority and hence they are computed by using the left to right associativity rule :

   8 / 2 + 3

   $\Rightarrow$ 4 + 3

5. Finally, the expression containing the addition operation is computed and the final value is obtained, which is,

   4 + 3 = 7

After the expression has been evaluated using side effects, the values of the variables are changed and thus the new values of *x*, *y* and *z* are 3, 9 and 2 respectively.

There is no specific rule to implement the expression using side effects i.e., it can be implemented in several ways. Thus, different computers used for implementing an expression produces different results.

2.1.2 Conditional Branching and Loops, Writing and Evaluation of Conditionals and Consequent Branching

Q16. Write about below statements,

(i) if statement

(ii) if-else

(iii) switch-case.

Answer :

(i) 'if' Statement

'if' statement is used for decision making. It allows the computer to first evaluate the condition and depending on its resultant value transfer the control to a particular statement in the program.

This statement performs an action if the condition is true, otherwise it skips that action and executes other statement.

Syntax

```
if (condition)
{
statement-1;
}
statement-next;
```

Here, if the condition is satisfied, then statement-1 is executed followed by statement-next. Otherwise, statement-1 block is skipped and only statement-next is executed.

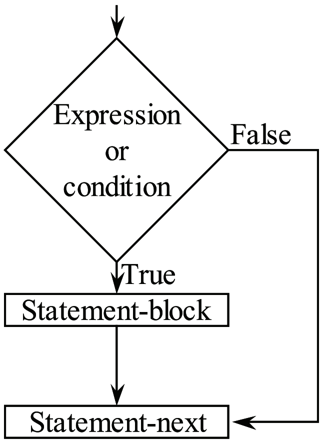
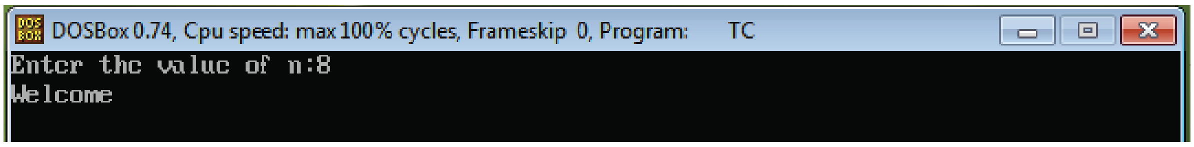


Figure: Flowchart of Simple if Statement

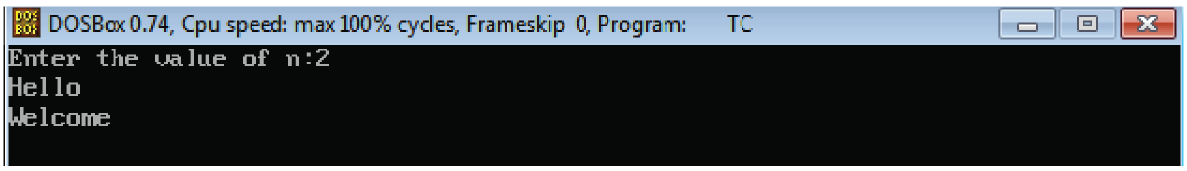
Example

```
#include<stdio.h>
#include<conio.h>
main( )
{
int n;
clrscr( );
printf("Enter the value of n:");
scanf("%d",&n);
if(n<=5)
{
printf("Hello \n");
}
printf("Welcome");
getch( );
return 0;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the value of n:8
Welcome
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the value of n:2
Hello
Welcome
```

(ii) 'if-else' Statement

This statement is an extension of simple 'if' statement.

Syntax

```
if (condition)
{
statement-1;
}
else
{
statement-2;
}
statement-next;
```

In the above syntax, statement-1 will be executed followed by statement-next when the condition is true. Otherwise, statement-2 will be executed followed by statement-3.

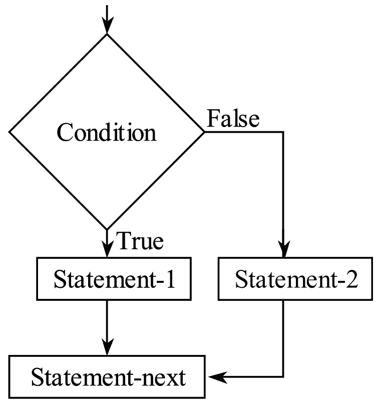
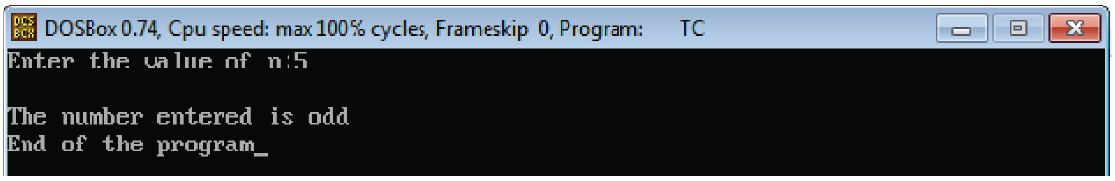


Figure: Flowchart of if-else Statement

Example

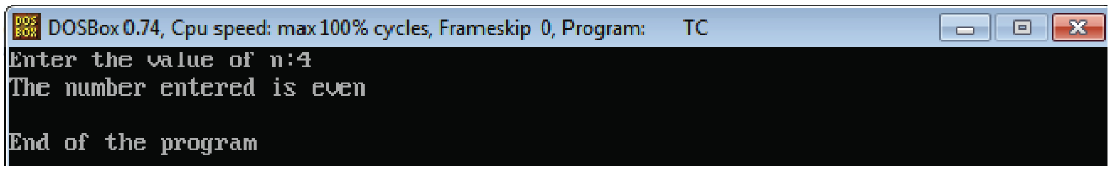
```
#include<stdio.h>
#include<conio.h>
main( )
{
int n;
clrscr( );
printf("Enter the value of n:");
scanf("%d",&n);
if(n%2==0)
{
printf("The number entered is even\n");
}
else
{
printf("\nThe number entered is odd");
}
printf("\nEnd of the program");
getch( );
return 0;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the value of n:5

The number entered is odd
End of the program_
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the value of n:4
The number entered is even

End of the program
```

(iii) 'switch' Statement

A 'switch' statement is a multi-way decision making statement.

Syntax

```
switch (expression)
{
     case constant1: statement-1;
                     break;
     case constant2: statement-2;
                     break;
     :
     case constantn: statement-n;
                     break;
     default: default-statement;
                break;
}
statement next;
```

Here, the expression is a valid 'C' expression and constant is the result of the expression. When the constant value is a character, it has to be enclosed within single quotes.

The value of the expression is evaluated and it is compared with constant case values. When match is found, the corresponding statement block associated with the case is executed.

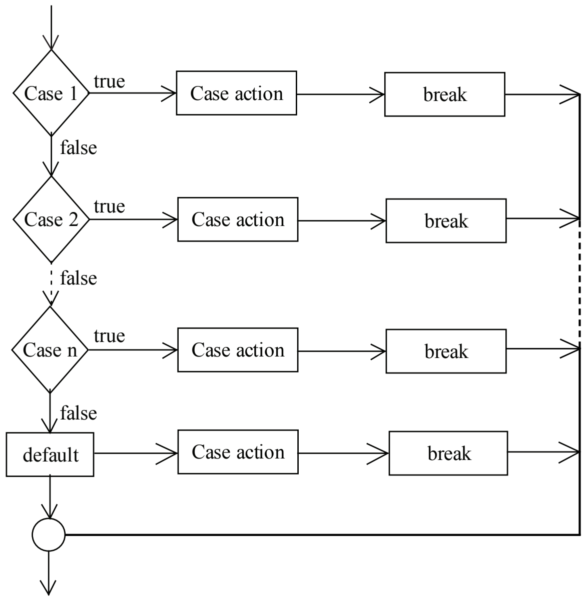
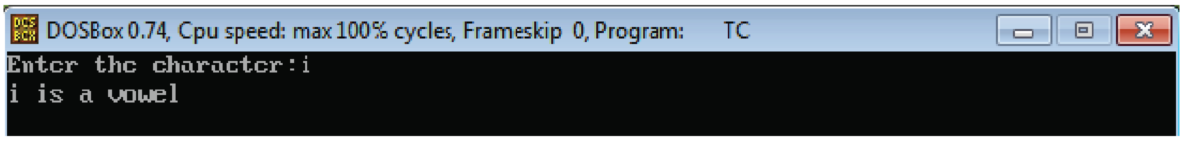


Figure: Flowchart of switch Statement

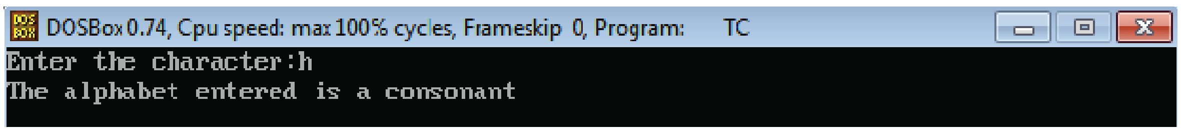
Example

```
#include<stdio.h>
#include<conio.h>
main()
{
char c;
clrscr();
printf("Enter the character:");
scanf("%c",&c);
switch(c)
{
case 'a':
printf("a is a vowel");
break;
case 'e':
printf("e is a vowel");
break;
case 'i':
printf("i is a vowel");
break;
case 'o':
printf("o is a vowel");
break;
case 'u':
printf("u is a vowel");
break;
default:
printf("The alphabet entered is a consonant");
}
getch( );
return 0;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Entcr thc charactcr:i
i is a vowel
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the character:h
The alphabet entered is a consonant
```

Q17. Explain ternary operator in C with an example program.

Answer :

Ternary Operator

Conditional operator is also known as a ternary operator since it takes three arguments.

Syntax

$\boxed{\text{exp}1\,?\,\text{exp}2 : \text{exp}3;}$

Where exp1, exp2 and exp3 are expressions.

The value of exp1 is evaluated first. If it is true, value of exp2 is evaluated otherwise exp3 is evaluated.

Examples

1. int a = 5, b = 10, c = 15;

   y = (a > b) ? b : c;

   In the above statement, the expression a > b is evaluated first and since it is false then value of *c* will be assigned to *y*. So, value of *y* will be 15.

   y = (a < b) ? b : c;

   Here, a < b is true, so value of *b* is assigned to *y*. So, value of *y* will be 10.

2. Greatest of three numbers using conditional operation is as follows.

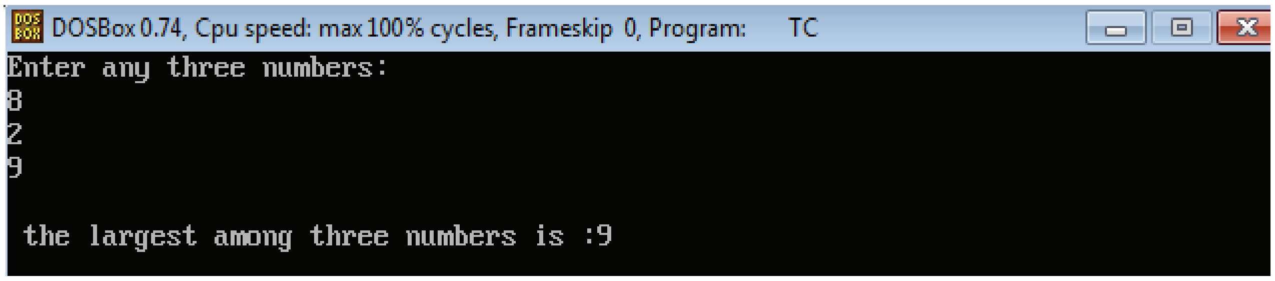
   x = (a > b) ? (a > c) ? (b > c) ? a : b : c;

   Here greatest of three numbers is stored in *x*

Program

```
#include<stdio.h>
#include<conio.h>
main()
{
int num1,num2,num3,largest;
clrscr();
printf("Enter any three numbers:");
scanf("%d%d%d",&num1,&num2,&num3);
largest=(num1>num2 && num1>num3 ? num1:num2>num3?num2:num3);
printf("\n the largest among three numbers is :%d",largest);
getch();
return 0;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter any three numbers:
8
2
9

 the largest among three numbers is :9
```

Q18. What is the purpose of goto statement? How is the associated target statement identified?

Answer :

goto Statement

It is used to alter the normal sequence of program execution by transferring the control (jump) to some other part of program. It is written as,

goto label;

❖ Here, label is an identifier which is used to give the location of target statement to which control must be transferred.

❖ Target statement is given as,
label : statement;

❖ Each labeled statement written within a program must have a unique label.

Syntax

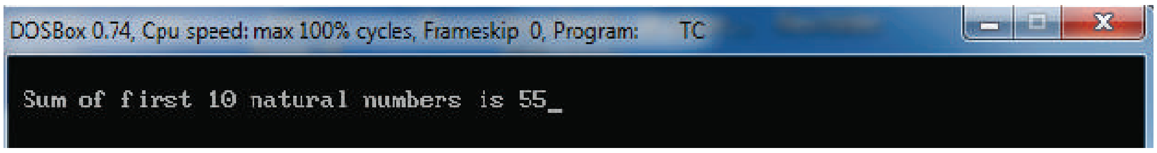
```
label : statement;        goto label;
________                  ________
________                  ________
 goto label;              label : statement;
```

(a) **Backward Jump** (b) **Forward Jump**

Example

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i = 0, s = 0;
    clrscr( );
    sum: s = s + i;
    i = i + 1;
    if (i <= 10)
    goto sum;
    printf("\n Sum of first 10 natural numbers is %d", s);
    getch( );
    return 0;
}
```

Output

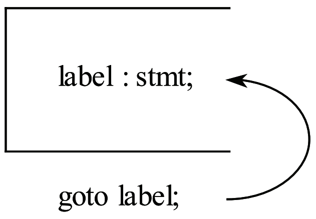

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Sum of first 10 natural numbers is 55_
```

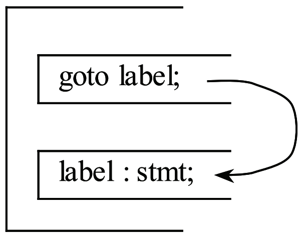
In the above program, goto statement is used to make backward jump. Here, sum is a label. The goto is executed until *i* becomes greater than 10. When *i* is greater than 10, the next statement is executed and sum is printed.

It is always advisable to avoid goto due to the following reasons,

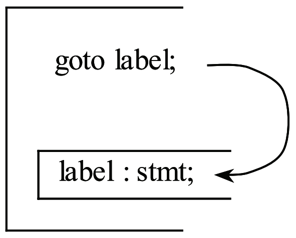
1. Many compilers generate a less efficient code.
2. Many of goto statements make program logic more complicated. Such situations occur when,
   (i) There is a backward jump from a statement to a block.



(ii) When jump occurs from one inner block to another inner block.



(iii) When jump occurs from outer block to inner block.



Q19. Explain various looping statements in 'C' language with example.

Answer :

Repetition (Iteration) Statements or Looping Statements

The iterative statements are statements that are executed repeatedly until a condition is satisfied. The different types of iterative statements are,

(i) while statement

(ii) do-while statement

(iii) for statement.

The above three iterative statements are also known as looping statements. Looping process includes the following four steps as follows,

❖ Set and initialize the counter variable.

❖ Evaluate the test expression.

❖ Execute the body of loop if test expression is true.

❖ Increment/decrement the counter variable.

(i) 'while' Statement

The 'while' statement will be executed repeatedly as long as the expression remains true.

Syntax

```
while (expression)
{
body of the loop;
}
```

When the 'while' is reached, the computer evaluates expression. If it is found to be false, body of loop will not be executed and the loop terminates. Otherwise, the body of loop will be executed, till the expression becomes false.

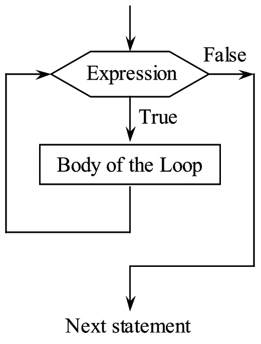
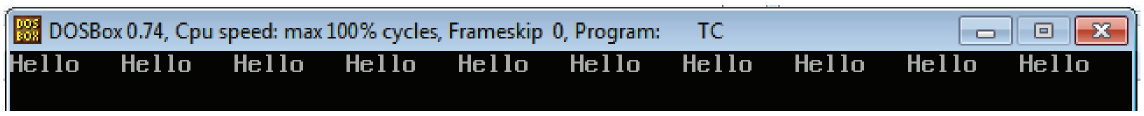


Figure (1): Flowchart of while Statement

Example

```
#include<stdio.h>
#include<conio.h>
main( )
{
    int i=1;
    clrscr( );
    while(i<=10)
    {
        printf("Hello\t");
        i++;
    }
    getch( );
    return 0;
}
```

Output



```
Hello   Hello   Hello   Hello   Hello   Hello   Hello   Hello   Hello   Hello
```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

(ii) 'do-while' Statement

It is similar to that of 'while' loop except that it is executed atleast once. The test of expression for repeating is done after each time body of loop is executed.

Syntax

```
do
{
body of loop;
} while (expression);
```

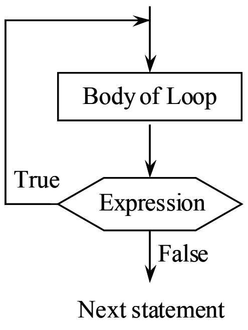


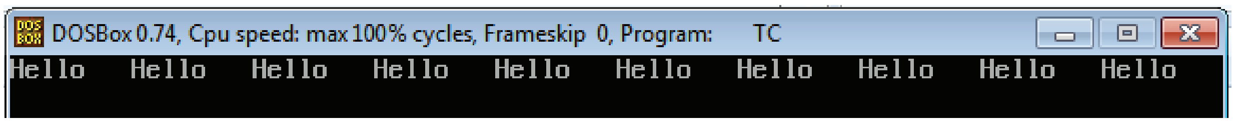
Figure (2): Flowchart of do-while Statement

Example

```
#include<stdio.h>
#include<conio.h>
main( )
{
int i=1;
clrscr( );
do
{
```

```
printf("Hello\t");
i++;
} while(i<=10);
getch( );
return 0;
}
```

Output



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program:    TC
Hello   Hello   Hello   Hello   Hello   Hello   Hello   Hello   Hello   Hello
```

(iii) 'for' Statement

'for' statement executes for specific number of times. After each iteration it inreases by a specified value.

Syntax

```
for (initializing expression; testing expression; updating expression)
{
statements;
}
```

Here the initialization expression specifies the initial value. The testing expression is a condition that is tested at each pass. The program is executed till this condition remains true. Updating expression is an unary expression that is either incremented or decremented to change the initial value. The conditional expression is evaluated and tested at the beginning while unary expression is evaluated at the end of each pass.

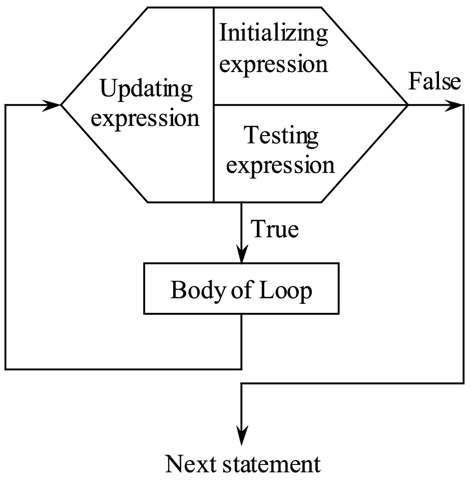
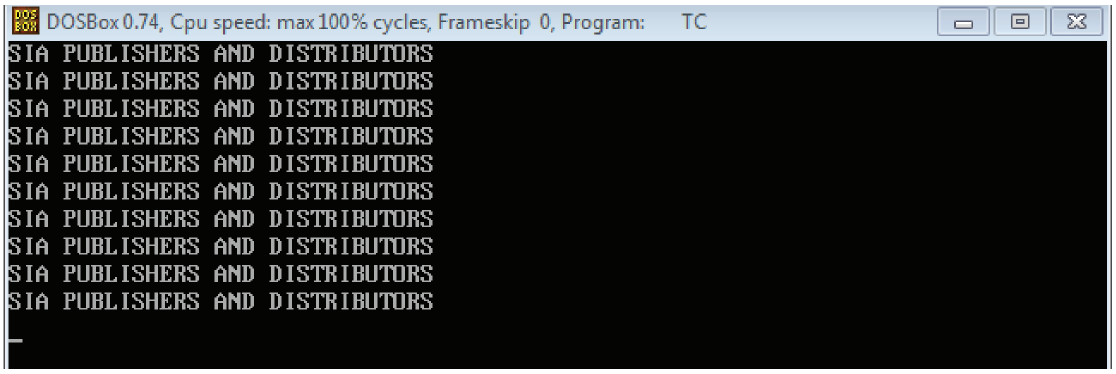


Figure (3): Flowchart of for Statement

Example

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i;
    clrscr( );
    for (i = 1; i<=10; i ++)
    printf("SIA PUBLISHERS AND DISTRIBUTORS \n");
    getch( );
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
SIA PUBLISHERS AND DISTRIBUTORS
_
```

Q20. Write a ‘C’ program to evaluate the power series,

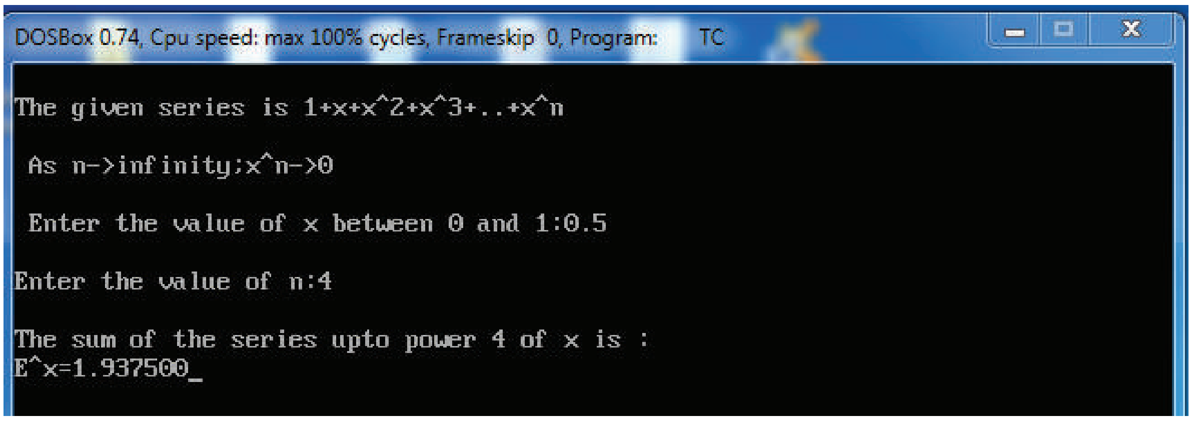
$e^x = 1 + x + x^2 + x^3 + ...+x^n$, $0 < x < 1$.

Answer :

Program

```
/*Program to evaluate the power series*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main( )
{
    long float res = 0.0;
    float x;
    int n, i;
    clrscr( );
    printf("\n The given series is 1 + x + x^ 2 + x^ 3 + ... + x^ n");
    printf("\n");
    printf("\n As n → infinity; x^n→0");
    printf("\n");
    printf("\n Enter the value of x between 0 and 1:");
    scanf("%f", &x);
    if ((0 < x) &&(x < 1))
    {
    printf("\n Enter the value of n:");
    scanf("%d", &n);
        for(i = 0; i < = n; i ++)
        res = res + pow(x, i);
        printf("\nThe sum of the series upto power %d of x is:", n);
        printf("\n E^ x = %lf", res);
    }
    else
    printf("\n x should lie between 0 and 1");
    getch( );
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
The given series is 1+x+x^2+x^3+..+x^n

 As n->infinity;x^n->0

 Enter the value of x between 0 and 1:0.5

Enter the value of n:4

The sum of the series upto power 4 of x is :
E^x=1.937500_
```

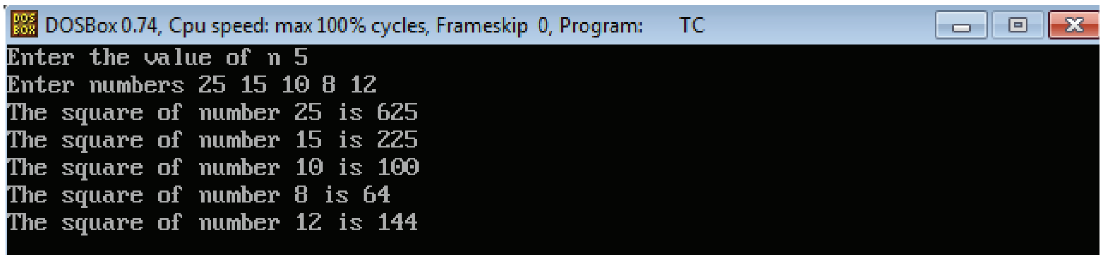
Q21. Write a 'C' program to find the squares of N numbers using do-while.

Answer :

Program

```
/*Program to find the squares of N number using do-while*/
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i, n, res;
    int a [10];
    clrscr( );
    printf("Enter the value of n");
    scanf("%d", &n);
    printf("Enter numbers");
    for(i = 0; i <n; i++)
    scanf("%d", &a[i]);
    i = 0;
    do
    {
        res = a[i] * a[i];
        printf("The square of number %d is %d \n", a[i], res);
        i ++;
    }
    while(i<n);
    getch( );
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the value of n 5
Enter numbers 25 15 10 8 12
The square of number 25 is 625
The square of number 15 is 225
The square of number 10 is 100
The square of number 8 is 64
The square of number 12 is 144
```

Q22. Write a program to print the following outputs using for loops.

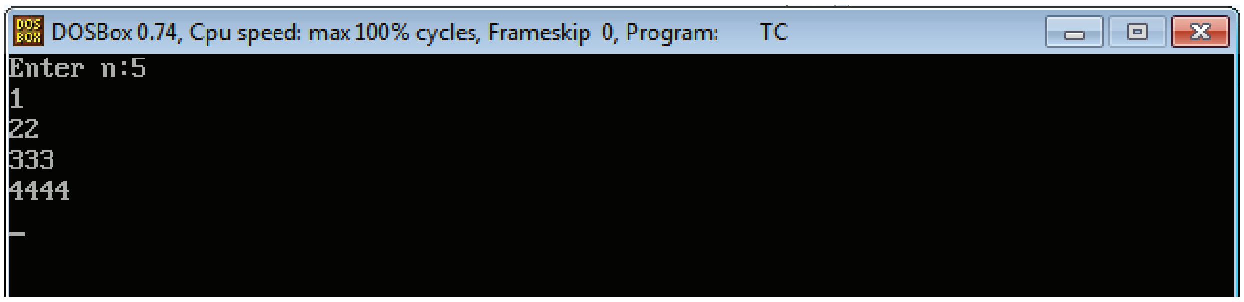
```
(i)                          (ii)
         1                                   1
         2  2                           2        2
         3  3  3                   3        3        3
         4  4  4  4           4        4        4        4
```

Answer :

(i)

```
#include<stdio.h>
#include<conio.h>
 main( )
{
int n,i,j;
clrscr( );
printf("Enter n:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
for(j=1;j<=i;j++)
printf("%d",i);
printf("\n");
}
getch( );
return 0;
}
```

Output



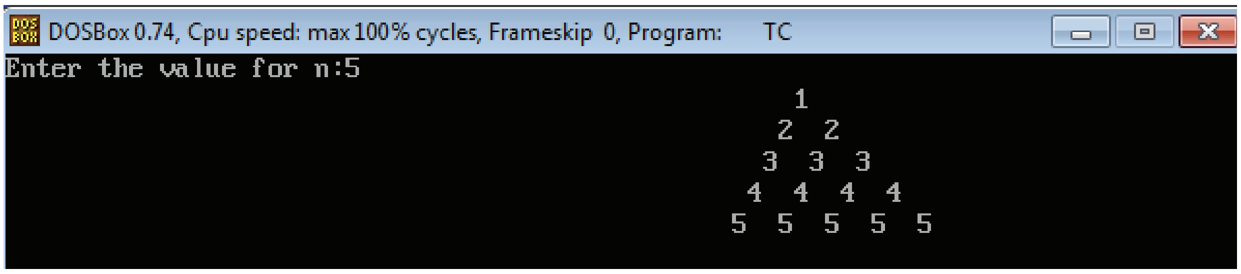
```
Enter n:5
1
22
333
4444
_
```

(ii)

```
#include<stdio.h>
#include<conio.h>
void main( )
{
        int n, i = 1, j = 1;
        clrscr( );
        printf("Enter the value for n:");
        scanf("%d", &n);
        for(i = 1; i < = n; i++)
        {
               for(j = 1; j < = 50 – i; j ++)
               printf(" ")
               for(j = 1; j<=i; j++)
```

```
                printf("%3d", i);
                printf("\n");
            }
            getch( );
        }
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the value for n:5
                                                  1
                                                2   2
                                              3   3   3
                                            4   4   4   4
                                          5   5   5   5   5
```

2.2 ARRAYS

2.2.1 Arrays (1-D, 2-D)

Q23. What is an array? What are different types of arrays? Explain with examples.

Answer :

Array

An array is a collective name given to a group of similar elements whereas a variable is any entity that may change during program execution.

An array is a variable that is capable of holding many values. Whereas, an ordinary variable can hold a single value at a time. For example, an array initialization would be,

int number[8];

For which there are eight storage locations reserved where eight different values can be stored belonging to the same type.

Types of Arrays

There are two types of arrays. They are as follows,

(a) One-dimensional arrays

(b) Two-dimensional arrays.

(a) One-dimensional Arrays

For answer refer Unit-II, Page No. 2.24, Q.No. 24.

(b) Two-dimensional Arrays

For answer refer Unit-II, Page No. 2.26, Q.No. 27.

Q24. Define one-dimensional array. Give syntax for declaration, accessing, initialization and printing one-dimensional array.

Answer :

One-dimensional Array

It is the simplest form of array in which list of elements are stored in contiguous memory locations and are accessed using only one subscript.

Declaration of One-dimensional Array

The general format for declaring one-dimensional array is,

Syntax

datatype array-name[size];

Here, 'datatype' specifies the data type of array elements, 'array-name' indicates the name of the array variable, and 'size' specifies the number of elements in the array. The size of the array is fixed at compile time, it cannot be changed during execution time.

Example

int a[10];

Here array '*a*' is of type int, that can store 10 elements.

Accessing Elements of an Array

Once the arrays are declared, every individual element can be accessed using the subscript which is an integer expression or an integer constant. The subscript specifies the position of element in the array this is also called index of array element. The array subscripts start at 0 and ends at "size of array – 1".

Syntax

array-name[i];

Here variable '*i*' refers to *i*th element of an array.

For example, a[0] refers to first element, a[1] refers to second element and so on.

Initialization of One-dimensional Array

User can initialize array elements at the place of their declaration itself. The general format of this initialization is,

type array-name[size] = {list of elements separated by commas}.

Examples

1. int a[4] = {5,10,1,55};

   Here, the size of this array is 4, the value 5 is assigned to first element i.e., a[0] = 5, the value 10 is assigned to second element i.e., a[1] = 10, the value 1 is assigned to third element i.e., a[2] = 1 and the value 55 is assigned to 4th element a[3] = 55.

   Suppose the list of elements specified are less than the size of array then only that many elements are initialized. The remaining elements are assigned garbage values.

2. int xyz[5] = {10, 5, 20};

   Here, the first element is assigned value 10, second element is assigned value 5 and third element is assigned value 20. Since, the list of elements is less than size of the array, the remaining two elements are assigned garbage values. Index in general runs from 0 to n–1 i.e., for an array a[n] we have values from a[0], a[1], ..., a[n–1].

   xyz[0] = 10, xyz[1] = 5, xyz[2] = 20,

   xyz[3] is assigned garbage value,

   xyz[4] is assigned garbage value.

   Consider the following statement.

   int xyz[ ] = {5, 6, 7, 8, 9};

In the above statement, size of an array is not specified. In such cases compiler fixes the size of the array based on the number of values in the list. In the above statement size of the this array is 5.

Printing One-dimensional Array

Array elements can be printed by using "for" loop and a 'printf' statement.

Example

```
int a[5] = {10, 20, 30, 40, 50};
for(int i = 0; i <=5; i++)
   printf("\n The elements of array are: %d", a[i]);
```

Q25. Write C function int minpos(float x[ ], int n) that returns position of the first minimum value among the first n elements of the given array x.

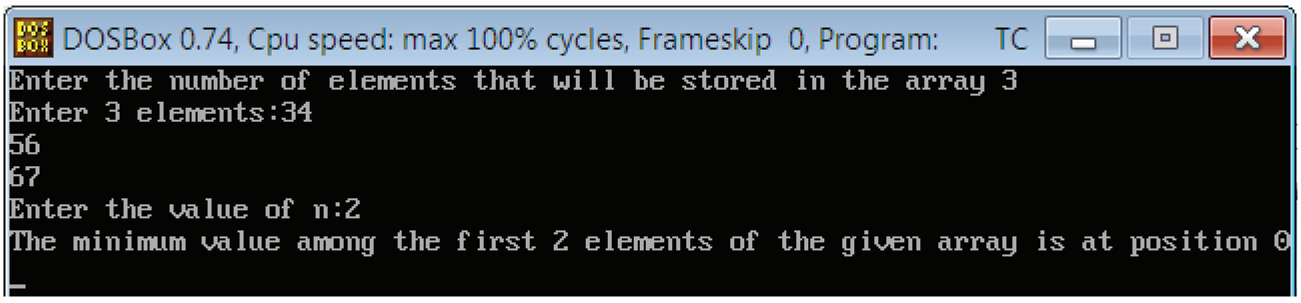
Answer :

Program

```
#include<stdio.h>
#include<conio.h>
int minpos(float x[ ], int n);
void main( )
{
```

```
int n,i, s, p;
float x[10];
clrscr();
printf("Enter the number of elements that will be stored in the array");
scanf("%d", &s);
printf("Enter %d elements:", s);
for(i = 0; i < s; i++)
scanf("%f", &x[i]);
printf("Enter the value of n:");
scanf("%d", &n);
if((n<s) && (n!=0))
{
p = minpos(x, n);
printf("The minimum value among the first %d elements of the given array is at position %d", n, p);
}
else
printf("Inappropriate value of n");
getch( );
}
int minpos(float x[ ], int n)
{
int i, pos = 0;
float min;
min = x[0];
for(i = 0; i < n; i ++)
{
if(x[i] < min)
{
min = x[i];
pos = i;
}
}
return pos;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the number of elements that will be stored in the array 3
Enter 3 elements:34
56
67
Enter the value of n:2
The minimum value among the first 2 elements of the given array is at position 0
_
```

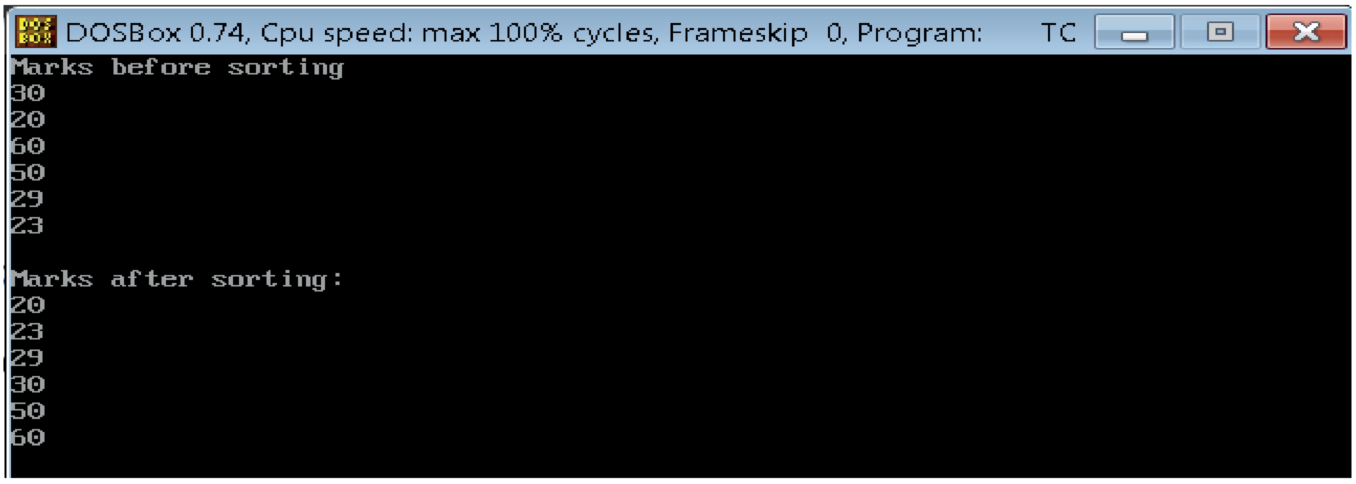
Q26. Write a C program that uses a function to sort an array of integers.

Answer :

Program

```
#include <stdio.h>
#include<conio.h>
int arraysort(int*,int);
void main( )
{
int i;
static int score[6]={30,20,60,50,29,23};
clrscr();
printf("Marks before sorting\n");
for(i=0;i<=5;i++)
printf("%d\n",score[i]);
printf("\n");
arraysort(&score[0],6); // User-defined function
getch();
}
arraysort(int *j,int n)
{
int i,t,k;
int a[6];
for(i=0;i<=n-1;i++)
{
a[i]=*j;
j++;
}
for(k=0;k<=n-2;k++)
{
for(i=k+1;i<=n-1;i++)
{
if(a[k]>a[i])
{
t=a[k];
a[k]=a[i];
a[i]=t;
}
}
}
printf("Marks after sorting:");
for(i=0;i<=n-1;i++)
{
printf("\n%d",a[i]);
j++;
}
return 0;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Marks before sorting
30
20
60
50
29
23

Marks after sorting:
20
23
29
30
50
60
```

Q27. Explain two-dimensional arrays.

Answer :

Two-dimensional Arrays

When the data must be stored in the form of a matrix two-dimensional arrays are used. For example,

int a[5][3];

Above declaration represents a two-dimensional array consisting of 5 rows and 3 columns. So the total number of elements which can be stored in this array are 5 × 3 i.e., 15.

Declaration of Two-dimensional Arrays

As shown in the above example, the general format of declaring a two-dimensional array is,

type array-name[row-size][column-size];

Where 'type' represents the data type of array elements, 'array-name' is the name of the array, 'row-size' represents the number of rows in the array and 'column-size' represents number of columns of the array. Both row-size and column-size must be integer constants.

Examples

1. int xyz[2][5];

   Here, xyz is an array consisting of two rows with each row consisting of 5 integer elements.

2. float a[5][5];

   Here, a is an array consisting of 5 rows with each row consisting of 5 floating point numbers.

Initialization of Two-dimensional Arrays

Two-dimensional arrays can also be initialized at the place of declaration itself. The general format of initializing two-dimensional array is,

type array-name[row-size][column-size] = {{list of elements in row 1},
{list of elements in row 2},
⋮
{list of elements in row n}};

Examples

1. int xyz[2][3] = {{2, 3, 4}, {5, 6, 7}};

   Here 2, 3 and 4 are assigned to three columns of first row and 5, 6, 7 are assigned to three columns of second row in order.

   If the values are missing in initialization, they are set to garbage values.

2. int xyz[2][3] = {{0}, {0}};

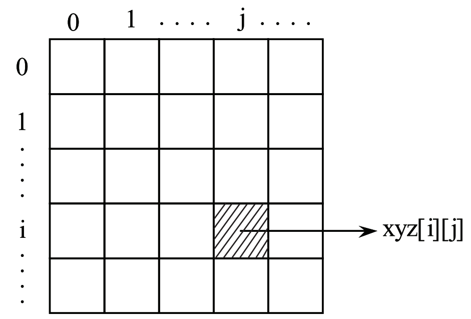
   Here, first element of each row is initialized to zero while other elements have some garbage values.

Referencing Elements of Two-dimensional Array

The elements of an array can be referenced using two indices, one for obtaining row number and another for obtaining column number.

Example

xyz[i][j] is used to access *i*th row and *j*th column element.



The number of elements in a two-dimensional array is given as,

(Upper_bound – Lower_bound + 1) × Number of columns.

or

Number of rows × Number of columns.

Storage Representation of Two-dimensional Arrays

Two-dimensional arrays consists of rows and columns but when it is stored in memory, no facility for two-dimensional storage is available. The memory is linear. Hence, the actual storage differs from our matrix representation. Two major types of representations can be used for two-dimensional arrays.

(a) Row major representation

(b) Column major representation.

(a) Row Major Representation

In row major representation, the elements of first row are stored in contiguous memory locations followed by elements of second row and so on.

Example

Consider an array,

int a[3][4];

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Figure: Logical View of a[3][4]

Its actual row major representation is,

row 0	0	1	500
	1	2	502
	2	3	504
	3	4	506
row 1	0	5	508
	1	6	510
	2	7	512
	3	8	514
row 2	0	9	516
	1	10	518
	2	11	520
	3	12	522

Figure: Physical View of Array a[3][4]

Here 500 is the base address of the array and each element is stored at a distance of 2 bytes because each integer element requires two bytes of storage.

(b) Column Major Representation

In column major representation, the elements of first column are stored in contiguous memory locations followed by elements of second row and so on.

The column major representation for the array given in row major representation, is given below.

Column 1	0	1	500
	1	5	502
	2	9	504
Column 2	0	2	506
	1	6	508
	2	10	510
Column 3	0	3	512
	1	7	514
	2	11	516
Column 4	0	4	518
	1	8	520
	2	12	522

Accessing Address of an Array Element

In an array a[m][n] stored using column major order, the address of a[i][j] is given as,

Base address + (i + j × m) × Size of array element.

Example

For the array a[3][4], address of a[2][3] is,

500 + (2 + 3 × 3) × 2 = 522.

Q28. Explain different methods of passing a two-dimensional array to a function.

Answer :

Two-dimensional arrays can be passed to a function in three different ways,

1. Passing Individual Elements

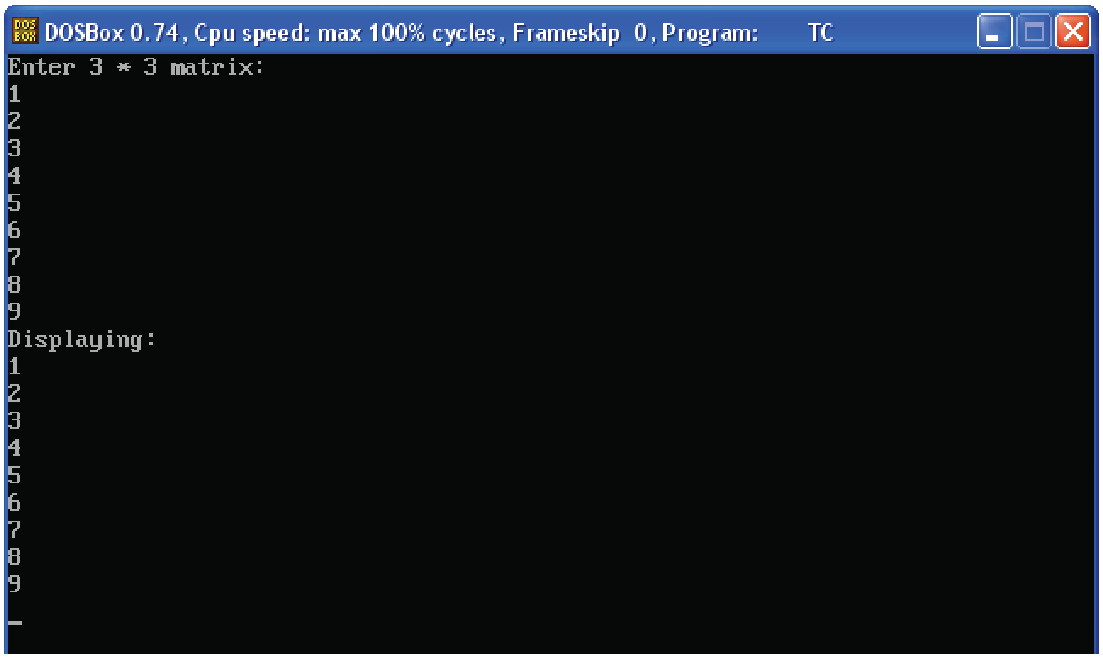
Passing individual elements in 2-D array is similar to that in 1-D array.

Program

```
#include<stdio.h>
#include<conio.h>
void PassArrEle(int c[3][3]);
void main( )
{
    int arr[3][3],i,j;
    clrscr( );
    printf("Enter 3 * 3 matrix:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        scanf("%d",&arr[i][j]);
    }
```

```
        PassArrEle(arr);
        return 0;
        getch( );
    }
    void PassArrEle(int arr[3][3])
    {
        int i,j;
        printf("Displaying:\n");
        for(i=0;i<3;++i)
        for(j=0;j<3;++j)
        printf("%d\n",arr[i][j]);
    }
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter 3 * 3 matrix:
1
2
3
4
5
6
7
8
9
Displaying:
1
2
3
4
5
6
7
8
9
_
```

2. Passing a Row

To pass a row of an array, array name must be indexed with the row number. Passing a row of 2D array is similar to "passing the entire array" of 1D array.

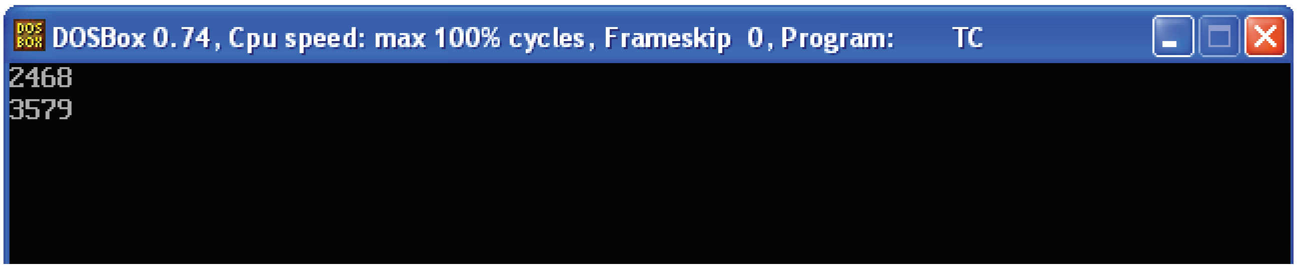
Program

Example

```
#include<stdio.h>
#include<conio.h>
void display (int[ ]);
int rows = 2;
int cols = 4;
void main( )
{
    int c[2] [4] = {
                    {2,4,6,8},
                    {3,5,7,9}
                };
```

```
        for(int i = 0; i< rows; i++)
        display (c[i]);
        getch( );
        }
        void display(int y[ ])
        {
                for (int j = 0; j< cols; j++)
                printf ("%d", y[j]);
                printf ("\n");
        }
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
2468
3579
```

3. Passing the Whole Array

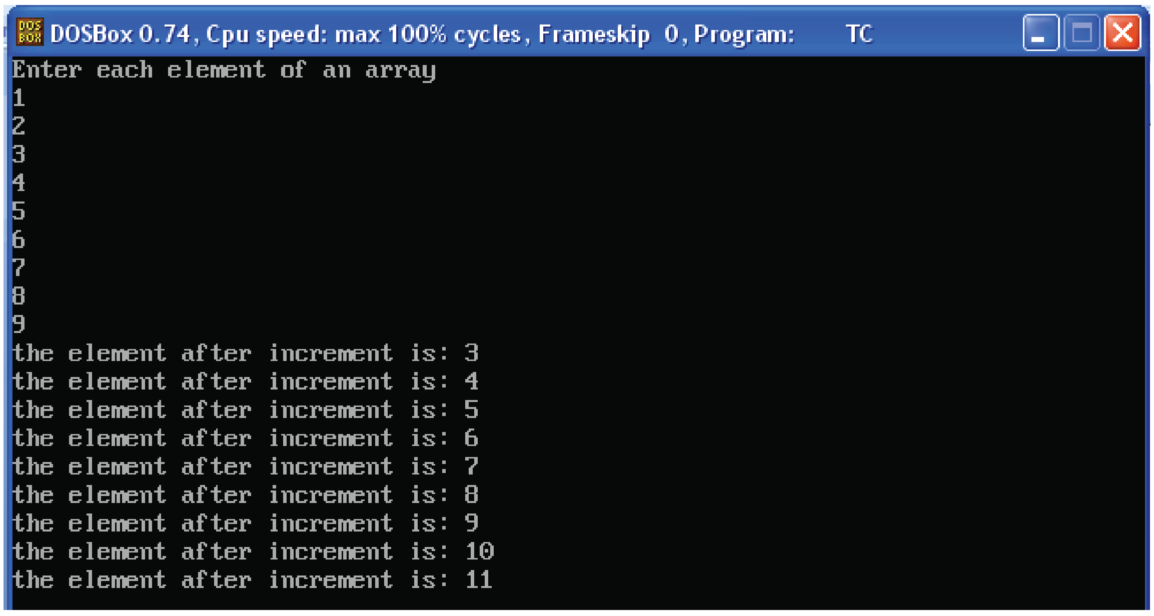
A two-dimensional array can be passed to a function by just specifying the array name as the actual parameter. In the called function, the formal parameter must receive the array as a two-dimensional array. It must indicate the array with its two dimensions. For a fixed length array, the size of the second dimension is compulsory whereas, for a variable length array, both dimensions are necessary.

Program

```
        #include<stdio.h>
        #include<conio.h>
        void incr(int);
        void main( )
        {
        int j, k, b[3] [3], rows = 3, cols = 3;
        clrscr( );
        printf("Enter each element of an array\n");
        for(j = 0; j< rows; j++)
        {
        for(k = 0; k< cols; k++)
        scanf("%d", &b[j][k]);
        }
        for(j = 0; j<rows; j++)
        for(k = 0; k<cols; k++)
        incr(b[j][k]);
        getch( );
        }
```

```
void incr(int y)
{
y = y+2;
printf("the element after increment is :");
printf("%d", y);
printf("\n");
}
```

Output



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program:     TC
Enter each element of an array
1
2
3
4
5
6
7
8
9
the element after increment is: 3
the element after increment is: 4
the element after increment is: 5
the element after increment is: 6
the element after increment is: 7
the element after increment is: 8
the element after increment is: 9
the element after increment is: 10
the element after increment is: 11
```

Q29. Explain how two-dimensional arrays can be used to represent matrices. Write C code to perform matrix addition and matrix multiplication.

Answer :

Matrix Representation of Two-dimensional Arrays

A two-dimensional array is an array of arrays. In contrast to one-dimensional representation of arrays in which the data (or the elements) is organized linearly in one-direction. The two dimensional representation of array's store's the data in two dimensions or directions. This representation of elements in two directions is like a table with row's and column's which is nothing but a matrix.

The diagrammatic representation of two-dimensional arrays is shown in the figure below.

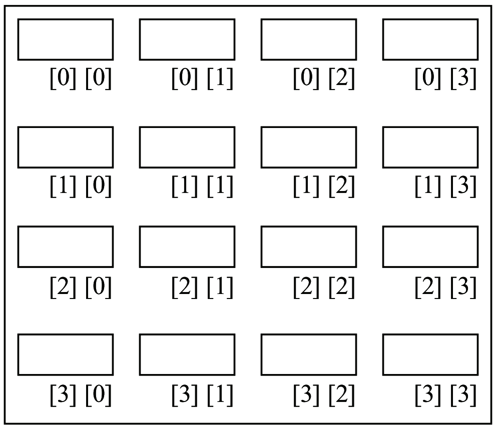


Figure: Representation of Elements in Two-dimensional Arrays

From this figure it is clear that a two-dimensional array is an array of one-dimensional array.

Now, consider the initialization of a two-dimensional array which has 4 elements in first dimension and 4 elements in second dimension. i.e., int sample [4] [4]

This array will store the elements in the following format which forms a 4 × 4 matrix.

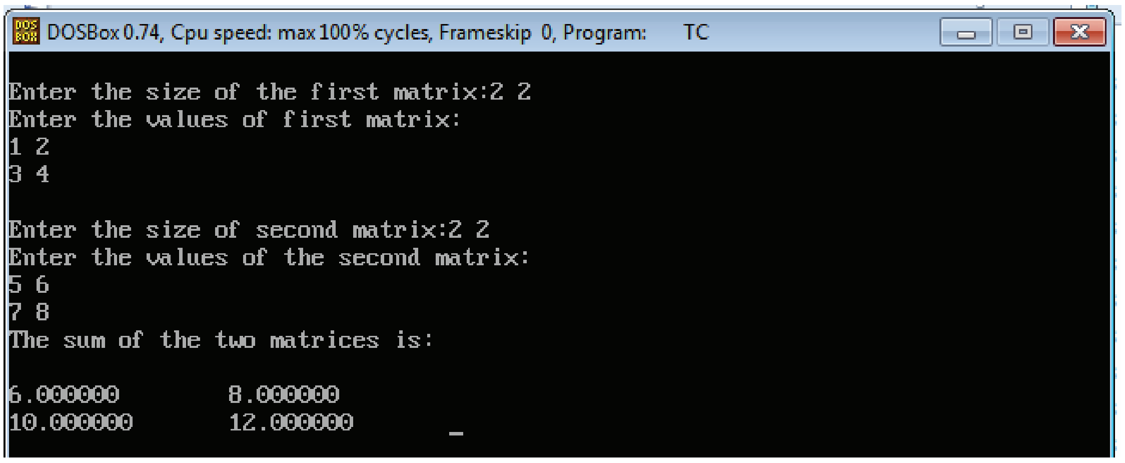
$$\begin{bmatrix} 2 & 4 & 8 & 9 \\ 3 & 6 & 10 & 12 \\ 7 & 5 & 2 & 4 \\ 21 & 28 & 1 & 5 \end{bmatrix}$$

Program for Addition of Two Matrices

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main( )
{
    int i,j,x,y,a,b;
    float p[6][6], q[6][6],r[6][6];
    clrscr( );
    printf("\nEnter the size of the first matrix:");
    scanf("%d %d",&x,&y);
    printf("Enter the values of first matrix:\n");
    for(i=0;i<x;i++)
    for(j=0;j<y;j++)
    scanf("%f",&p[i][j]);
    printf("\nEnter the size of second matrix:");
    scanf("%d %d",&a,&b);
    printf("Enter the values of the second matrix:");
    for(i=0;i<a;i++)
    for(j=0;j<b;j++)
    scanf("%f",&q[i][j]);
    if((x= =a)&&(y= =b))
    {
        printf("The sum of the two matrices is:\n");
        for(i=0;i<x;i++)
        {
            printf("\n");
```

```
                for(j=0;j<y;j++)
                {
                    r[i][j]=p[i][j]+q[i][j];
                    printf("%f\t",r[i][j]);
                }
            }
        }
        else
        printf("\n Addition of these two matrices is not possible");
        getch( );
        return 0;
    }
```

Output



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the size of the first matrix:2 2
Enter the values of first matrix:
1 2
3 4

Enter the size of second matrix:2 2
Enter the values of the second matrix:
5 6
7 8
The sum of the two matrices is:

6.000000        8.000000
10.000000       12.000000      _
```

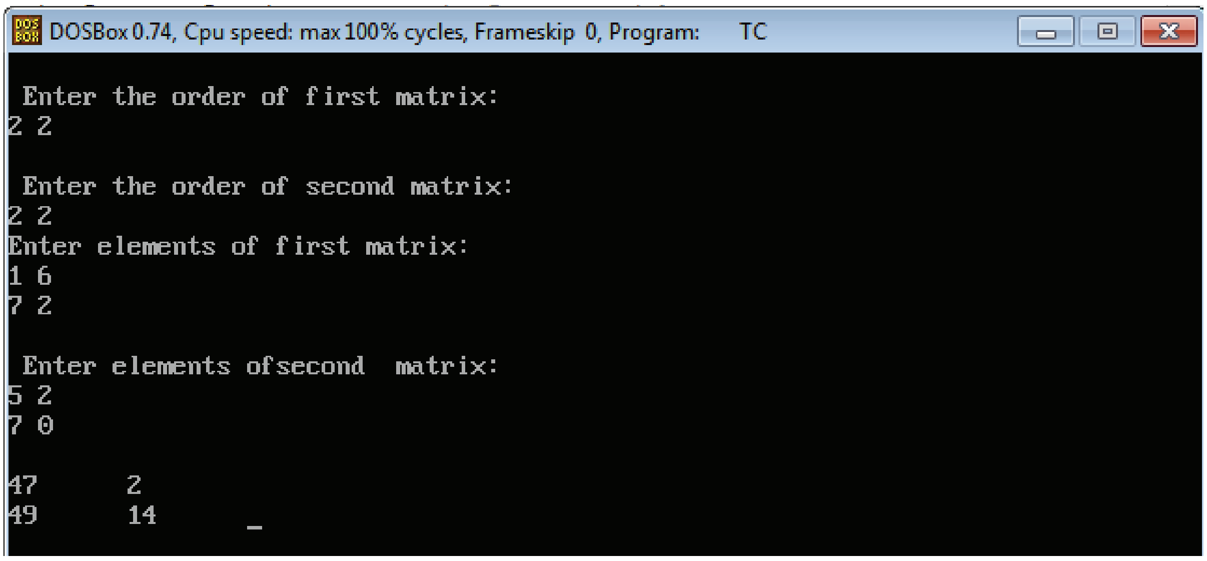
Program for Multiplication of Two Matrices

```
/* Program to multiply two matrices */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int r1, c1, r2, c2, i, j, k;
    int a[10][10], b[10][10], c[10][10];
    clrscr();
    printf("\n Enter the order of first matrix:");
    scanf("%d%d", &r1, &c1);
    printf("\n Enter the order of second matrix:");
    scanf("%d%d", &r2, &c2);
```

```
printf("Enter elements of first matrix:");
for(i = 0; i<r1; i++)
{
      for(j = 0; j<c1; j++)
      {
           scanf("%d", &a[i][j]);
      }
}
printf("\n Enter elements of second  matrix:");
for(i = 0; i< r2; i++)
{
      for(j = 0; j< c2; j++)
      {
           scanf("%d", &b[i][j]);
      }
}
/*Matrix multiplication */
if(c1 == r2)
{
      for(i = 0; i <r2; i++)
      {
            for(j = 0; j <c2; j++)
           {
                c[i][j] = 0;
                for(k = 0; k<r2; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
           }
      }
      for(i = 0; i <r1; i++)
      {
           printf("\n");
           for(j = 0; j <c1; j++)
           {
                printf("%d\t", c[i][j]);
           }
      }
}
```

```
else
printf("\n Multiplication not possible");
getch( );
}
```

Output



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
 Enter the order of first matrix:
2 2

 Enter the order of second matrix:
2 2
Enter elements of first matrix:
1 6
7 2

 Enter elements ofsecond  matrix:
5 2
7 0

47      2
49      14      _
```

Q30. Explain how matrices can be represented using two-dimensional arrays. Explain with code how transpose of matrix can be done.

Answer :

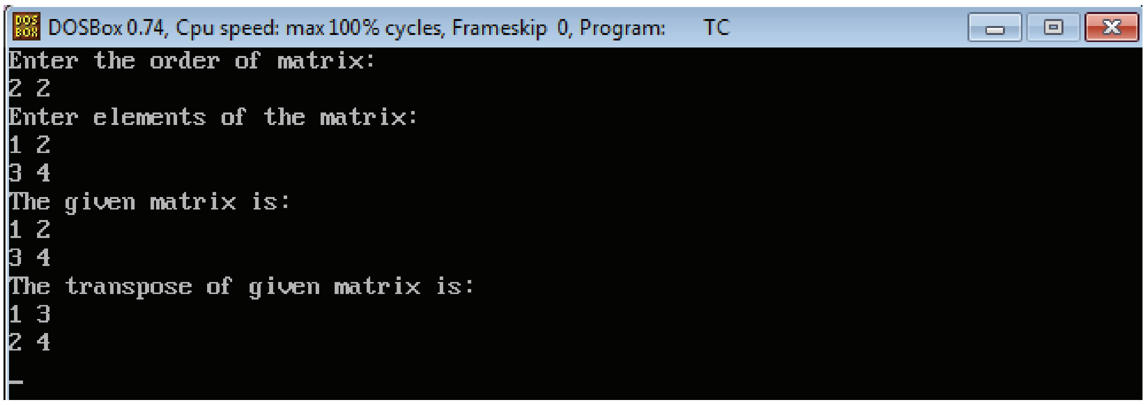
Representation of a Matrix using Two-dimensional Arrays

For answer refer Unit-II, Page No. 2.31, Q.No. 29, Topic: Matrix Representation of Two-dimensional Arrays.

Transpose of a Matrix

```
# include <stdio.h>
# include <conio.h>
void main( )
{
int a[10] [10], m, n, i, j;
clrscr();
printf("Enter the order of matrix:\n");
scanf("%d%d",&m,&n);
printf("Enter elements of the matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
 }
printf("The given matrix is: \n");
for (i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d", a[i][j]);
}
printf("\n");
}
printf("The transpose of given matrix is: \n");
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
printf("%d", a[j][i]);
}
printf("\n");
}
getch();
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the order of matrix:
2 2
Enter elements of the matrix:
1 2
3 4
The given matrix is:
1 2
3 4
The transpose of given matrix is:
1 3
2 4
_
```

2.2.2 Character Arrays and Strings

Q31. What is character array? How character arrays are declared?

Answer :

Character Arrays

The term character array refers to the array with character values. Since, C programming does not support string data type, therefore to create a string an array of type character is used. Thus, a character array can be called as 'String'. It is used by programming language for manipulating text such as words and sentences. In C, a string is always terminated using null character ('\0'). Character arrays are always declared before using in a program.

Declaration

The declaration of character array is done as follows,

Syntax

```
char arrayname[arraysize];
```

Example

```
char dept[10];
```

Here, dept is the array of character or string of size 10.

Initialization

The initialization of character array is done as follows,

Syntax

```
char arrayname[arraysize] = {list of characters};
```

Example

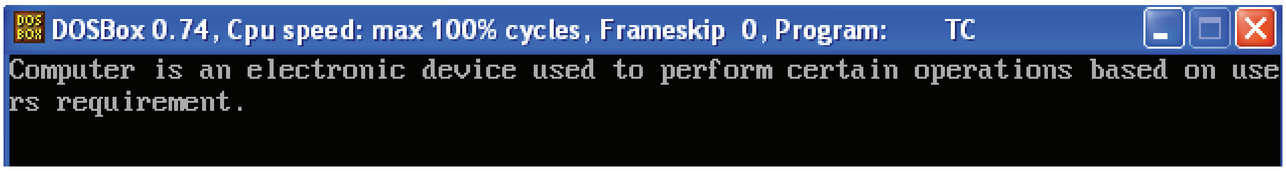
```
char [10] = {'A', 'B', 'C', 'D', 'E', 'F'};
```

Character arrays are generally used in the similar way to that of numeric arrays. If the characters are handled on the basis of single character, then the I/O functions such as scanf( ) and printf( ) must be provided with only %c conversion specifier and such character arrays can not be called as string. But if these I/O functions are provided with %s conversion specifier then that character array can be called as strings.

Program

```
#include<stdio.h>
#include<string.h>
void main( )
{
    char msg[10];
    clrscr( );
    strcpy(msg, "Computer is an electronic device used to perform certain operations based on users requirement.");
    printf("%s\n", msg);
    getch( );
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Computer is an electronic device used to perform certain operations based on use
rs requirement.
```

Q32. Explain briefly about string classification.

Answer :

String

A string is considered as an array of characters. In C, strings are classified into two types. They are,

1. Fixed-length strings
2. Variable-length strings.

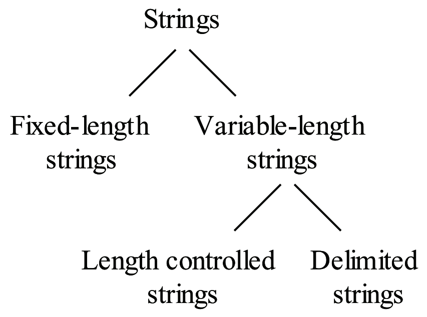


Figure (1): Classification of Strings

1. Fixed-length Strings

Fixed-length strings have a fixed length which is set at the time of their creation.

The first decision in implementing fixed-length strings is size of the variable. If the size of the variable is small then, it can not store all the data. On the other hand, if the size of the variable is larger then memory is wasted.

Moreover, it is difficult to identify the data from a set of nondata. But, this problem can be overcomed by adding nondata characters like spaces at the end of the data.

2. Variable-length Strings

Variable-length strings vary in size. They have no maximum limit. Thus to identify the end of a variable-length string two techniques are used. They are,

(i) Length controlled strings

(ii) Delimited strings.

(i) Length Controlled Strings

Length-controlled strings are the strings that adds a count at the beginning of the string. This count represents the number of characters present in the string. String manipulation functions use this count to find the length of the string. Figure (2) illustrates the format of length-controlled string.

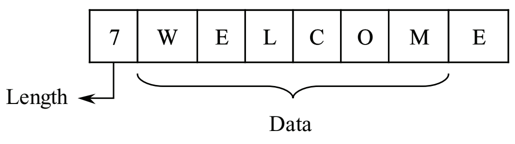


Figure (2): Length Controlled String Format

(ii) Delimited Strings

Delimited strings use a delimiter at the end of the string. The null character (\0), which is the very first character in the ASCII character set is used as a delimiter. Delimiter (\0) should not be used as data. Figure (3) below illustrates the format of delimited string.

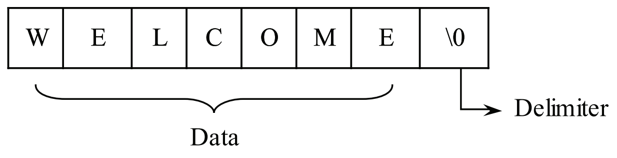


Figure (3): Delimited String Format

Q33. Explain the process of declaration and initialization of strings.

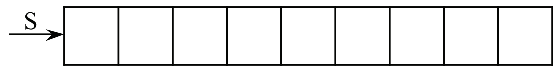
Answer :

Declaration of Strings

In C, there is no specific data type for the declaration of strings. There are two approaches for declaration of strings. In the first approach, strings are implemented as an array of characters. The general format of declaring a string is as follows,

char str_var[size];

E.g: char S[9];

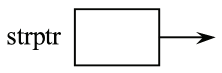


The string '*S*' will hold upto 9 characters including delimiter (\0).

In this case, memory for the string '*S*' is allocated when it is declared as an array in local memory.

In the second approach, strings are implemented using pointers. The general format of declaring strings as pointer is as follows,

char *strptr;



The declaration of string as a pointer allocates storage space only for the pointer but not for the strings. Therefore, to use the string, memory must be allocated for it otherwise logical error occurs.

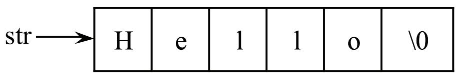
Initialization of Strings

In C, there are three approaches for the initialization of strings. In the first approach, strings are initialized as string literals. For instance, char S[8] = "Welcome";

If the string is initialized at the time of its definition then the size of the string is not necessary to be mentioned. For instance,

char str[ ] = "Hello";

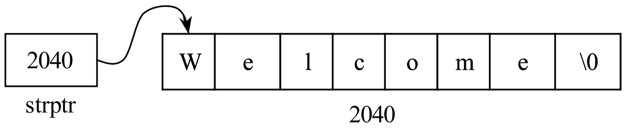
Here, compiler allocates storage space of 7 bytes for the string "str" and initializes it with "Hello" and ends it with a delimiter (\0).



In the second approach, strings are initialized using character pointers. For instance,

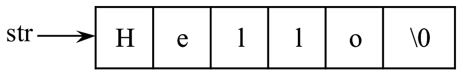
char *strptr = “Welcome”;

In this case, address of string is stored in the string pointer (strptr).



In the third approach, strings are initialized as an array of characters. For instance,

char str[10] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};



In this case, null character is not assigned by the compiler. It must be specified at the end of the string.

Q34. What are the string manipulation functions? Explain their usage.

Answer :

String Handling Functions

The ‘C’ library provides a large number of string handling functions that can be used for performing string manipulations. These functions are available in the header file string.h. This header file is used or included whenever these functions are used.

The following are the commonly used string handling functions

strcat( )	:	Concatenates two strings.
strcmp( )	:	Compares two strings.
strcpy( )	:	Copies one string into another.
strlen( )	:	Finds the length of a string.
strcmpi( )	:	Compares two strings without case sensitivity.
strncmp( )	:	Compares two portions of substrings upto specified position.
strncmpi( )	:	Compares two portions of strings upto specified length without case sensitivity.

1. strcat( )

This function is used to join two strings. The general format of this function is given below,

Syntax

strcat(string1,string2);

Here, string1 and string2 are character arrays. When the function strcat( ) is executed, string2 is appended to string1. The string2 remains unchanged and string1 should be large enough to accommodate the final string. The string2 can be string variable or constant, but string1 should be a string variable.

strcat(string1, string2); → Valid statement

strcat(string1, “Hello”); → Valid statement

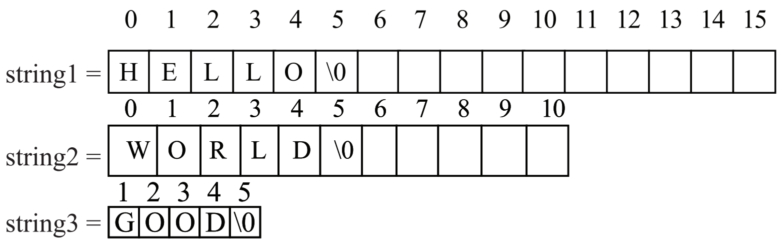
strcat(“Hello”, string2); → Invalid statement.

Example

char string1[16] = “HELLO”;

char string2[11] = “WORLD”;

char string3[5] = “GOOD”;



strcat(string1, string2); will result in,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
H	E	L	L	O	W	O	R	L	D	\0					

strcat(string2, string3); will result in,

0	1	2	3	4	5	6	7	8	9	10
W	O	R	L	D	G	O	O	D	\0	

'C' language permits nesting of strcat function.

Example

strcat(strcat(string1, string2), string3);

Concatenates the three strings string1, string2 and string3. The resultant string is stored in string1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
H	E	L	L	O	W	O	R	L	D	G	O	O	D	\0	

2. strcmp( )

This function is used to compare two strings. The general format of this function is given below,

Syntax

strcmp(string1, string2);

Here, both string1 and string2 are character arrays. One of the strings, string1 or string2 can be a string constant. String1 and string2 are compared as per ASCII collating sequence. The strcmp() returns an integer value according to the following rules,

(i) If string1 is equal to string2, value 0 is returned.

(ii) If they are not, it has a numeric difference between the first non matching characters in strings.

(iii) If string1 is greater than string2, a positive value is returned.

(iv) If string1 is less than string2, a negative value is returned.

Example

char string1[6] = "HELLO", string2[6]= "HELP";

(i) int i = strcmp(string1, string2);

The value returned when the above statement is executed is – 4, since string1 and string2 are not equal and the difference of O(ASCII value 76) and P(ASCII value 80) is – 4.

(ii) int i = strcmp(string2, string1);

The value returned when the above statement is executed is + 4 since string1 and string2 are not equal therefore their difference (80 – 76) i.e., 4 is returned.

(iii) int i = strcmp(string1, "HELLO");

The value returned when the above statement is executed is 0 because the strings are equal.

(iv) int i = strcmp("world", string1);

The value returned when the above statement executed is 15.

3. strcmpi( )

It is same as strcmp( ) except the case of characters is not considered. Its format is,

strcmpi(string1, string2);

Example

char str1[ ] = "HELLO";

char str2[ ] = "hello";

int i = strcmpi(str1, str2);

The value returned when this statement is executed is 0 because this function does not consider the case of characters.

4. strncmp( )

This function compares two strings upto specified position. The comparison is performed similar to that of strcmp( ) function. Its syntax is given below.

Syntax

strncmp(string1, string2, position);

Here, both string1 and string2 are character arrays and position is an integer value. The function returns an integer value.

Example

char string1[6] = “HELLO”, string2[6] = “HELP”;

int i = strncmp(string1, string2, 3);

When above statement is executed, the value of i will be 0 because it compares only first three characters of both the strings.

5. strncmpi( )

It is similar to that of strncmp( ) except that it is not case sensitive.

Syntax

strncmpi(string1, string2, position);

Example

char string1[6] = “HELLO”, string2[6] = “hello”;

int j = strncmpi(string1, string2, 4);

The value of j will be 0 because here first 4 characters are compared and case of characters is not considered.

6. strcpy( )

This function is used to copy one string to a string variable. The general format of this function is given below.

Syntax

strcpy(string1, string2);

Here, both string1 and string2 are character arrays. string2 may be a string variable or a string constant, the contents of string2 are copied into string1. Almost it works like string assignment operator. String1 should be large enough to accommodate string2.

Example

strcpy(name, “HELLO WORLD”);

When the above statement is executed the string “HELLO WORLD” will be assigned to string variable “name”.

strcpy(name1, name2);

When the above statement is executed the contents of string variable name2 are assigned to string variable name1.

7. strlen( )

This function is used to determine the length of the string. The general format for this function is given below.

Syntax

strlen(string1);

Here string1 is a character array. String1 may be a string variable or a string constant. The function returns the length of the string.

Example

string1 = “HELLOWORLD”;

int n = strlen(string1);

It gives n = 10.

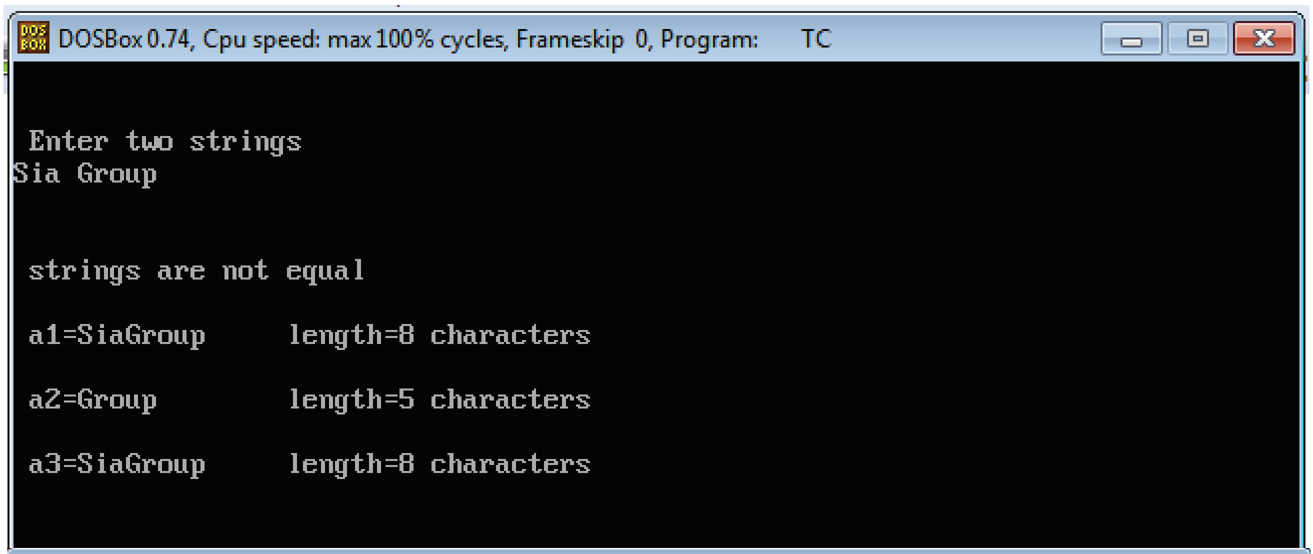
Q35. Write a sample program for various string manipulation functions.

Answer :

Program

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
        char a1[15], a2[15], a3[15];
        int y, l0, l1, l2;
        clrscr();
        printf(“\n\n Enter two strings\n”);
        scanf(“%s%s”, a1, a2);
        y=strcmp(a1, a2);
        if(y!=0)
        {
        printf(“\n\n strings are not equal\n”);
        strcat(a1, a2);
        }
        else
        printf(“\n\n strings are equal\n”);
        strcpy(a3, a1);
        l0 = strlen(a1);
        l1 = strlen(a2);
        l2 = strlen(a3);
        printf(“\n a1=%s\t length=%d characters\n”,a1, l0);
        printf(“\n a2=%s\t length=%d characters\n”,a2, l1);
        printf(“\n a3=%s\t length=%d characters\n”,a3, l2);
        getch();
}
```

Output



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

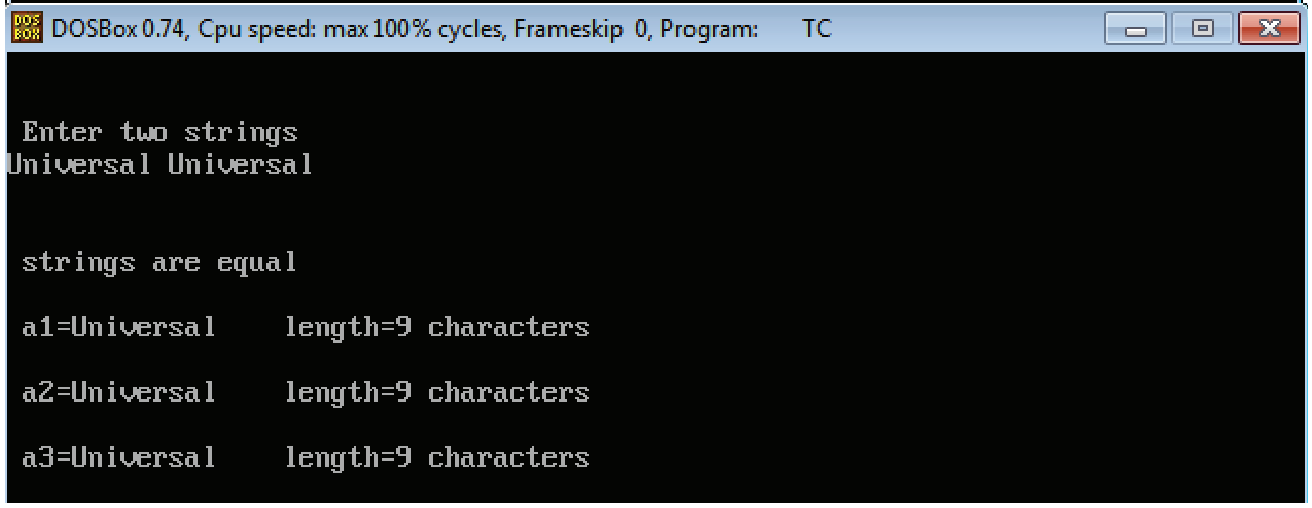
```
 Enter two strings
Sia Group


 strings are not equal

 a1=SiaGroup     length=8 characters

 a2=Group        length=5 characters

 a3=SiaGroup     length=8 characters
```



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
 Enter two strings
Universal Universal


 strings are equal

 a1=Universal    length=9 characters

 a2=Universal    length=9 characters

 a3=Universal    length=9 characters
```

UNIT 3

BASIC ALGORITHMS, FUNCTIONS AND PASSING ARRAYS TO FUNCTIONS

PART-A

SHORT QUESTIONS WITH SOLUTIONS

Q1. Define algorithm. Give the steps for algorithm development.

Answer :

Algorithm

An algorithm is a method of representing step-by-step procedure for solving a problem. An algorithm is very useful for finding the right answer to a problem or to a difficult problem by breaking the problem into simple cases.

Steps for Algorithm Development

The steps to be followed while developing an algorithm are,

1. Initially understand the problem
2. Then, identify the expected output for the problem.
3. Identify the necessary input for the problem.
4. Develop a logic that produces the expected output from the selected input.
5. Finally, test the algorithm with various sets of input.

Q2. Write about functions.

Answer :

Function

A function is a self contained program segment that performs specific and well defined tasks.

Use of functions ease the complex tasks by dividing the program into modules.

There are two functions in C language.

1. Programmer-defined Functions

These functions allow the programmer to define their own functions according to the requirement of the program.

2. Library Functions

These are predefined functions that are provided to perform computations such as sin(x), cos(x), pow(x, i), sqrt(x) etc. These functions cannot be modified by the programmer.

Q3. How a function is declared?

Answer :

Declaration of Function

A function declaration declares a function that consists of a function type (or return type), function name parameter (or argument) list and a terminating semicolon. Function declaration is also called function prototype. A function must be declared before it is invoked.

Syntax

function-type function-name (parameter-list);

Where, parameter-list is a list of arguments with data types separated by a comma. However, it is not necessary for the parameter names to be same in the prototype declaration and the function definition, but their types must match.

Q4. Define actual and formal parameters.

Answer :

Actual Parameters

The parameters that are passed in function call are called actual parameters. These parameters are defined in the calling function. Actual parameters can be variables, constants or expressions.

Formal Parameters

The parameters that are used in function definition are called formal parameters. These parameters belong to the called function. These can be only variables but not expressions or constants.

Q5. Write about return statement.

Answer :

Return Statement

This statement returns control to the calling function. There can be number of return statements in the program but each statement can return only one value.

Example

For example, return($p + q + r$);

Here the expression in the parenthesis is solved and then the result is returned.

Moreover, multiple return statements can be present in a function depending on certain conditions.

Q6. List any five mathematical functions.

Answer :

Many important library functions for mathematical calculations are available in math.h header file.

Function	Description	Function Prototype	Example
abs( )	Returns the absolute value of an integer number.	int abs(int num);	abs(–3) = 3
fabs( )	Returns the absolute value of a floating point number.	double fabs(double num);	fabs(3.2) = 3.2
labs( )	Returns the absolute value of a long integer number.	long labs(long num);	labs(3.2L) = 3.2
ceil( )	Returns the smallest integer value greater than or equal to a number.	double ceil(double num);	ceil(3.005) = 4
floor( )	Returns the largest integer value less than or equal to a number.	double floor(double num);	floor(3.9999) = 3.0

Q7. How arrays are passed to a function?

Answer :

Arrays are passed to a function by just sending the array name without the address operator. That is, passing an array name to a function is nothing but passing an address of the first element in the array. The function call with array name as a argument is as follows.

funcsia(array name);

There are two approaches for the declaration of an array in the called program. In the first approach, simple array notation is used which looks as follows.

int funcsia(int array[ ])

In the second approach, array is declared as a pointer in the function header, which looks as shown below.

int funcsia(int *array);

PART-B

ESSAY QUESTIONS WITH SOLUTIONS

3.1 BASIC ALGORITHMS

Q8. What is an algorithm? Write the various criteria used for judging an algorithm.

Answer :

For answer refer Unit-I, Page No. 1.12, Q.No. 22.

3.1.1 Searching

Q9. Write an algorithm to perform linear search on an array.

Answer :

Algorithm for linear search of an array is as follows,

```
Step 1
    Initialize index (i.e., position) $i$ = 0 and number of elements in list = $n$
Step 2
    Check, if($i < n$) then
    Goto step 3
    else
    Goto step 5
Step 3
    Compare, if(key matches with list element at $i$) then
    printf("Element found ! successful search")
    Return the index ($i$) of the found element
    Goto step 6
    else
Step 4
    Increment $i$ by 1
    Goto step 2
Step 5
    printf("End of list; Element not found ! unsuccessful search")
Step 6
    Stop
```

Q10. Write an algorithm to perform Binary search on an array.

Answer :

Algorithm of Binary Search

A non-recursive binary search routine has three arguments i.e., L[ ] (i.e., address of the list of elements), *n* (i.e., the number of elements in the list), *key* (i.e., the element to be searched). In this search method, the list of elements is divided into two groups and the *key* element is searched in these groups. This process is carried out until the element is found in the list. If the *key* element is not found in the list then a message "Unsuccessful search, element not found" is displayed.

Step 1

```
Initialize low_val = 0 and high_val = n
while(high_val > = low_val) do
mid_val = (low_val + high_val)/2
Goto step 2
Otherwise when condition fails
printf("Unsuccessful search, element not found")
Goto step 5
```

Step 2

```
Check, if(L[mid_val] = key) then
printf("Element found at mid_val")
Goto step 5
else
```

Step 3

```
Check, if(L[mid_val] > key) then
high_val = mid_val –1
Goto step1
else
```

Step 4

```
Check, if (L[mid_val] < key) then
low _val = mid_val + 1
Goto step 1
```

Step 5

```
Stop.
```

In the above algorithm low_val, high_val, mid_val are the index values of the list (i.e., L[ ]) to be searched.

The algorithm computes mid_val at step 1 to divide the list into two equal values and the search is carried out depending on the three conditions i.e., step 2, step 3, step 4.

3.1.2 Basic Sorting Algorithms (Bubble and Selection)

Q11. Write an algorithm for bubble sort.

Answer :

Algorithm for bubble sort is as follows,

Step 1

```
Initialize index element i=1
```

Step 2

```
Repeat step 3 to step 11 until all the elements are sorted.
```

Step 3

```
Set j to n–1
```

Step 4

```
If sortList[j]≤ sortList[j+1] then go to step 8
```

Step 5

Set temp to sortList[j+1]

Step 6

Set sortList[j+1] to sortList[j]

Step 7

Set sortList[j] to temp.

Step 8

Increment i and store the value in itself.

Step 9

If $i \leq j$ then goto step 4.

Step 10

Set j to j–1

Step 11

If $j \geq 1$ then go to step 1

Step 12

Print the elements after sorting.

Step 13

End

Q12. Write an algorithm to sort the list of elements using selection sort.

Answer :

Algorithm for selection sort is as follows,

Step 1

Read all the elements from the list.

Step 2

Store each element in an array

Step 3

Set the index value i=1

Step 4

Repeat step 5 to step 10 until all the elements in the list are sorted.

Step 5

Set first to i

Step 6

Set smallest to i+1

Step 7

Repeat step 8 and step 9 while smallest < size_of_the_list.

Step 8

If smallest < min, swap min and smallest.

Step 9

Increment smallest

Step 10

Increment i

Step 11

End

3.1.3 Finding Roots of Equations

Q13. Write an algorithm to find the roots of a quadratic equations.

Answer :

Algorithm to find all roots of quadratic equation is as follows,

```
1.  Begin
2.  Declare the variables x, y, z, D, p, q, rp and ip
3.  Calculate the discriminant
        $D \leftarrow y^2 - 4xz$
4.  If $D \geq 0$ then calculate
        root 1 $\leftarrow (-y + \sqrt{D})/2a$
        root2 $\leftarrow (-y - \sqrt{D})/2a$
    Display the roots i.e., root1 and root2
    else
        Calculate real part and imaginary part
        real part $\leftarrow y/2a$
        imaginary-part $\leftarrow \pm\sqrt{(D)}/2a$
    Display real-part + j(imaginary-part) and
        real-part – j(imaginary-part) as roots
5.  End
```

3.2 FUNCTIONS

3.2.1 Functions (Including Using Built-in Libraries)

Q14. What is the need for functions? Explain the elements of functions.

Answer :

Need for Functions

Every program in a C language needs a main function from where it has to begin its execution. In some cases, using a single 'main' function to code a complete program may result in the development of very large program with many complications. Such programs are difficult to debug, test and maintain. For this reason, a program in 'C' is divided into a number of functions (for each independent task) which are combined in a later stage into a single unit. Such functions are known as 'sub-programs' or 'functions'.

Functions are also very helpful in programs where certain operations or calculations are repeated at various parts of that program. In these types of programs, a function can be designed for required operation and can be called at the points where it is required. The process results in saving of both time and space.

Elements of Functions

A function consists of three elements,

(i) Function declaration

(ii) Function definition

(iii) Function call.

(i) Function Declaration

A function declaration declares a function that consists of a function type (or return type), function name parameter (or argument) list and a terminating semicolon. Function declaration is also called function prototype. A function must be declared before it is invoked.

Syntax

Function-type function-name (parameter-list);

Where, parameter-list is a list of arguments with data types separated by a comma. However, it is not necessary for the parameter names to be same in the prototype declaration and the function definition, but their types must match.

Example

```
float sum(float x float y);
```

This example is a function prototype for the function sum( ) that takes two float arguments and returns a float value as result.

(ii) Function Definition

A function definition is the complete description of a function. Actually a function definition tells what a function does and how it performs. A function definition contains a function body (a block of statements) in addition to function name, arguments list and return type. A function definition has the following general syntax.

Syntax

```
return_type function_name(parameter_list)
{
    local variable declarations;
    - - - -
    statements;
    - - - -
    return (expression);
}
```

Example

```
float sum(float x, float y)
{
    float z;
    z = x + y;
    return (z);
}
```

(iii) Function Call

A function can be called with function name and a list of actual parameters enclosed in parameters. When a function call is encountered, the control is transferred to the respective function, which will be then executed and the resulting value is returned to the main function.

Syntax

The general syntax of a function call is,

F(. . .)

Here, 'F' represents the operand or the function name, and (. . . .) indicates the operator or the parenthesis set which consists of actual parameters. Multiple actual parameters are separated by commas.

Example

A function can be called in various ways. For example, consider a function 'mul( )' it can be called in six different ways,

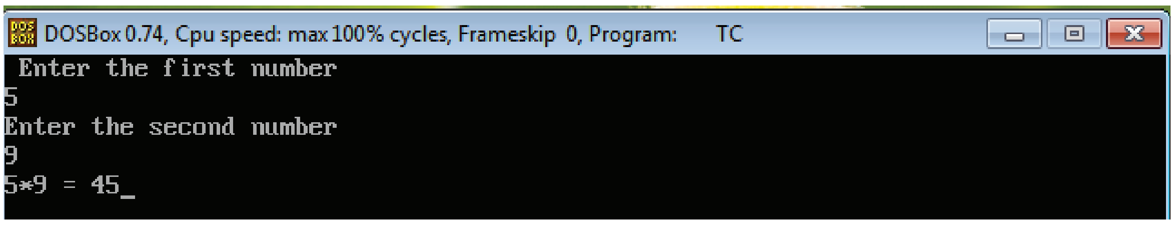
- mul(8, 6)
- mul(z, 6)
- mul(8, z)
- mul(z + 8, 6)
- mul(6, mul(z, i))
- mul(Expression1, Expression2)

Here, the first three examples indicate that mul( ) is called with primary expressions. In the fourth example, mul( ) is called with binary expression z + 8 as the first argument value. The fifth example shows that mul( ) is called with mul(z, i) as its own first argument. The sixth example is the summation of the above five examples.

Program

```
#include<stdio.h>
#include<conio.h>
int mul(int x, int y);
void  main( )
{
        int x,y,z;
        clrscr( );
        printf("Enter the first number \n");
        scanf("%d",&x);
        printf("Enter the second number \n");
        scanf("%d",&y);
        z=mul(x, y);
        printf("%d*%d = %d", x, y, z);
        getch( );
}
int mul(int x, int y)
{
        int p;
        p = x*y;
        return(p);
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
 Enter the first number
5
Enter the second number
9
5*9 = 45_
```

Q15. What do you mean by functions? Give the structure of the functions and explain about the arguments and their return values.

Answer :

Function

A function is a self contained program segment that performs specific and well defined tasks.

Use of functions ease the complex tasks by dividing the program into modules.

There are two functions in C language.

1. **Programmer-defined Functions**

   These functions allow the programmer to define their own functions according to the requirement of the program.

2. **Library Functions**

   These are predefined functions that are provided to perform computations such as sin(x), cos(x), pow(x, i), sqrt(x) etc. These functions cannot be modified by the programmer.

Structure of a Function

The structure of a function is,

```
return_value_type function_name(parameter_list)
{
declarations
 - - - - -
statements
 - - - - -
return(expression);
}
```

The various sections of the function format are discussed as follows,

(i) **return_value_type**

It is the data type of a value that the function returns.

(ii) **function_name**

Every function is known by a particular name.

(iii) **parameter_list**

This list that is enclosed within the parenthesis in the above syntax can have variables that are separated by a comma (,). Parameters are also known as arguments. These arguments have communication between the calling function and the called function.

(iv) **declarations**

In this section, the local variables are declared. These variables are local because their life is confined to the scope of the function in which they are declared.

(v) **statements**

These refer to the different statements such as the printf( ), scanf( ) statements or other assignment statements.

(vi) **return(expression)**

This statement returns the value of expression to the calling function.

Arguments in Functions

Depending on the location of parameters in functions, they are categorized as actual parameters and formal parameters.

(i) Actual Parameters

The parameters that are passed in function call are called actual parameters. These parameters are defined in the calling function. Actual parameters can be variables, constants or expressions.

(ii) Formal Parameters

The parameters that are used in function definition are called formal parameters. These parameters belong to the called function. These can be only variables but not expressions or constants.

Consider a small program,

```
          void sum(int i, int j, int k)   //Function definition
          {
Called        int s;
function      s = i + j + k;
              printf("Sum is %d", s);
          }
          void main( )
          {
Calling       int m = 4;
function      sum(2*m, m, 5);      // Function call; Actual parameters
          }
```

In this program, 2*m, m, 5 are actual parameters and *i*, *j* and *k* are formal parameters.

Return Statement

This statement returns control to the calling function. There can be number of return statements in the program but each statement can return only one value.

Example

For example, return($p + q + r$);

Here the expression in the parenthesis is solved and then the result is returned.

Moreover, multiple return statements can be present in a function depending on certain conditions.

Return Type in Functions

This specifies the data type of value returned.

Example

For example, the return type of a function, say sum( ) can be integer (int) or float depending on the value returned to that function.

❖ If no value is returned to the calling function then return type is declared as void.

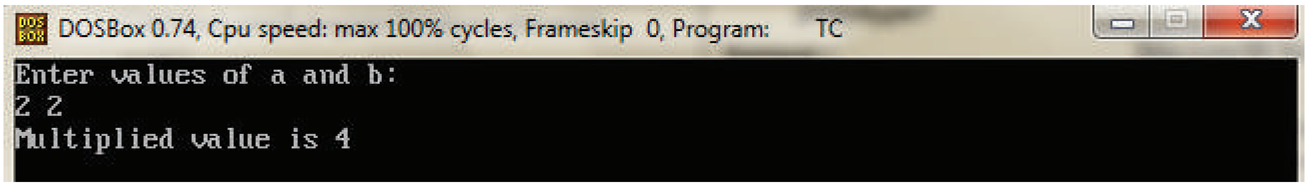
❖ By default, return type of any function will be of type int

Program

Consider a small program for return statement,

```
#include<stdio.h>
#include<conio.h>
int mul(int , int );
int main( )
{
int a, b, c;
clrscr();
printf("Enter values of a and b:\n");
scanf("%d %d", &a, &b);
c = mul(a, b);
printf("Multiplied value is %d", c);
getch();
return;
}
int mul(int p, int q)
{
int cross;
cross = p*q;
return cross;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter values of a and b:
2 2
Multiplied value is 4
```

In this program, the return type of mul( ) is int and the "return statement" returns an integer value in variable 'cross' to the main function.

Q16. What are standard library functions? Illustrate them with suitable examples.

Answer :

Standard C Library Functions

C has a large set of useful built-in library functions which are ready to be used in our programs. In order to use these functions their prototype declarations must be included in our programs. The function prototypes for these functions are available in general header files. Therefore, the appropriate header file for a standard function must be included at the beginning of our programs.

The standard library functions are as follows,

(a) Mathematical Functions

Many important library functions for mathematical calculations are available in math.h header file.

Function	Description	Function Prototype	Example
abs( )	Returns the absolute value of an integer number.	int abs(int num);	abs(–3) = 3
fabs( )	Returns the absolute value of a floating point number.	double fabs(double num);	fabs(3.2) = 3.2
labs( )	Returns the absolute value of a long integer number.	long labs(long num);	labs(3.2L) = 3.2
ceil( )	Returns the smallest integer value greater than or equal to a number.	double ceil(double num);	ceil(3.005) = 4
floor( )	Returns the largest integer	double floor(double num); value less than or equal to a number.	floor(3.9999) = 3.0
pow( )	Returns the value of a raised to the power b. An error occurs if a = 0 and b ≤ 0 or if a < 0 and b is not an integer.	double pow(double a, double b);	pow(2.0, 5.0) = 32.0
sqrt( )	Returns the square root of a number.	double sqrt(double num);	sqrt(144.0) = 12

(b) String Handling Functions

C provides many string handling functions for string manipulations. These functions are available in string.h header file.

Function	Format	Description	Example

strcpy( )	strcpy(target, source)	Copies source string to target string and returns target.	char S1[ ] = "ASH";
			char S[10], S2[10];
			S = strcpy(S2, S1);
			Output: S = "ASH"

strcat( )	strcat(target, source)	Concatenates source string to the end of the target and returns target.	char S1[ ] = "ASH";
			char S2[ ] = "RAF";
			char S[10];
			S = strcat(S1, S2);
			Output: S = "ASHRAF"

strlen( )	strlen(source)	Returns the length of the source string.	int n;
			n = strlen(S);
			Output: n = 6

strcmp( )	strcmp(str1, str2)	Compares two strings. Its return values are,
		0 – if str1 == str2
		<0 – if str1 < str2
		>0 – if str1 > str2
			char S1[ ] = "ASHRAF";
			char S2[ ] = "SHAZIA";
			int r;
			r = strcmp(S1, S2);
			Output: r = –18

(c) General Library Functions

The general library functions of 'C' library are available in stdlib.h header file.

Function	Description	Function Prototype	Example
srand( )	Creates the first seed for a pseudo random number series. Seed is a variable that is used by the random number generator (rand) to generate next number in the series.	void srand(unsigned int seed);	srand(920);
rand( )	Returns a pseudo random integer in the range 0 to RAND_MAX. In ANSI/ISO standard the RAND_MAX value is 32,767.	int rand(void);	rand( );

Q17. Explain the following library functions,

(a) sqrt(x)

(b) fmod(x,y)

(c) toupper(x).

Answer :

(a) sqrt(x)

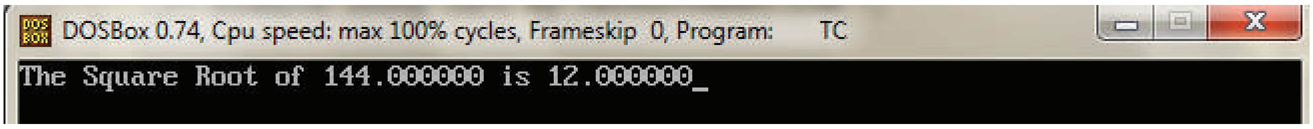
This library function returns the non-negative square root of the given *x* value. However, the *x* value should be greater than zero.

Example

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

```
int main()
{
float n=144, res;
clrscr();
res = sqrt(n);
printf("The Square Root of %f is %f", n, res);
getch();
return 0;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
The Square Root of 144.000000 is 12.000000_
```

This function has a single argument and is of type double.

(b) fmod(x, y)

The fmod(*x*, *y*) library function computes the *x* modulo *y* (i.e.,) floating point remainder of *x* by *y*.

This function is declared by the following statement,

double(double *x*, double *y*);

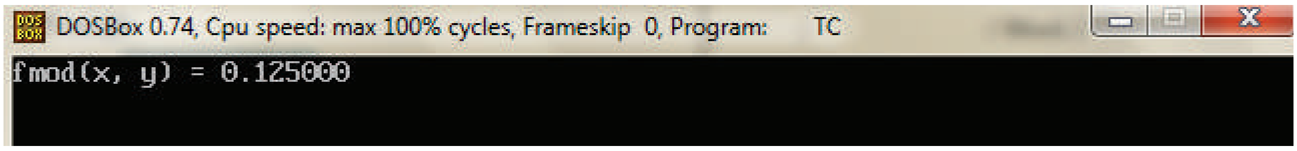
The return type of this function is *x-qy*

Where, '*q*' is the quotient of *x* by *y*.

Example

```
#include <stdio.h>
#include <math.h>
void main( )
{
double x = 3*3.14159 + 0.125;
double y = 3.14159;
clrscr();
printf("fmod(x, y) = %f\n", fmod(x,y));
getch();
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
fmod(x, y) = 0.125000
```

(c) toupper(x)

This library function transforms the lowercase letters into uppercase letters.

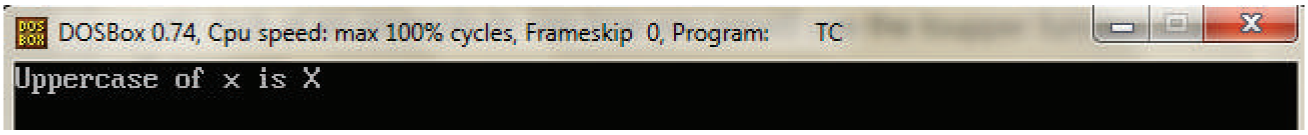
The declaration of toupper(x) function is given below

int toupper(int ch);

Example

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
int main()
{
int ch = ‘x’;
clrscr();
printf(“Uppercase of %c is %c \n”, ch, toupper(ch));
getch();
return 0;
}
```

Output



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Uppercase of x is X
```

3.2.2 Parameter Passing in Functions, Call by Value

Q18. Explain in detail about passing parameters to functions.

Answer :

Parameters Passing Techniques

There are two ways of passing parameters to a function,

1. Call-by-value
2. Call-by-reference.

1. Call-by-Value Mechanism

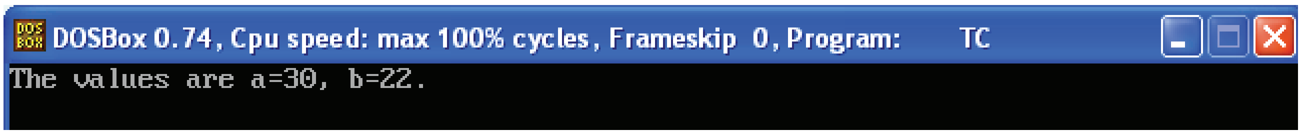
In this method, values of the actual arguments in the ‘calling function’ are copied into the corresponding parameters in the ‘called function’. Hence, the modifications done to the arguments in the ‘called’ function will not be reflected back to the actual arguments.

Example

```
#include<stdio.h>
#include<conio.h>
void add(int, int);
main( )
{
    int x = 10, y = 20;
    add(x, y);
    printf(“The values of x and y in the calling function are”);
    printf(“%d %d”,x, y);
}
```

```
void add(int a, int b)
{
      a = a+b;
      b = b+2;
      printf("a=%d b=%d",a,b);
}
```

Output



```
The values are a=30, b=22.
```

In the above program, the values of the actual parameters i.e., *x* and *y* are passed to the function "add(int a, int b) and a stack is maintained in the memory for storing the parameters i.e., when a function call is executed, the actual values of the parameters are placed on the stack. These values can be used and modified by the 'called' function without making any changes to the original variables.

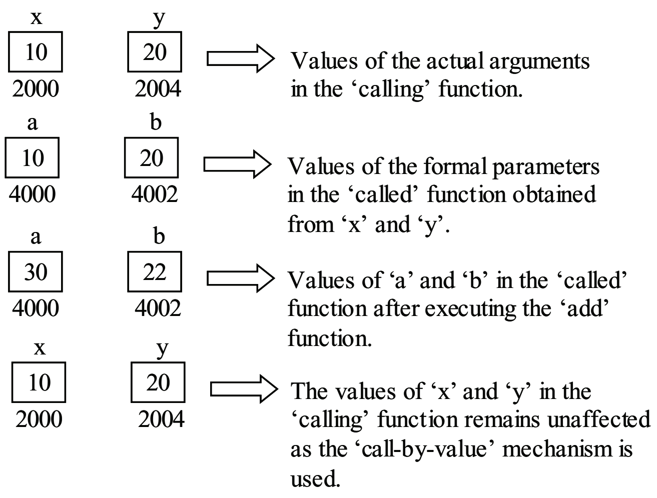

x
y
10
20
2000
2004
Values of the actual arguments in the 'calling' function.
a
b
10
20
4000
4002
Values of the formal parameters in the 'called' function obtained from 'x' and 'y'.
a
b
30
22
4000
4002
Values of 'a' and 'b' in the 'called' function after executing the 'add' function.
x
y
10
20
2000
2004
The values of 'x' and 'y' in the 'calling' function remains unaffected as the 'call-by-value' mechanism is used.

Figure (1): Call-by-Value Mechanism

2. Call-by-Reference Mechanism

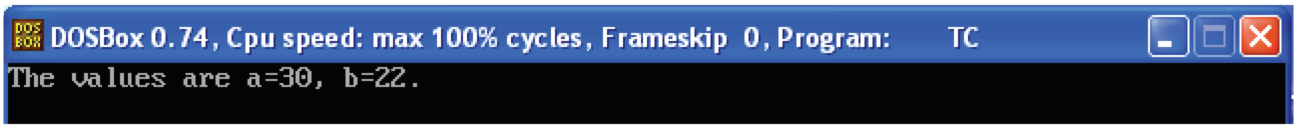
In this method, the addresses of the actual arguments are copied to the corresponding parameters in the 'called' function. Hence, any modifications done to the formal parameters in the 'called function' causes the actual parameters to change.

Example

```
#include<stdio.h>
#include<conio.h>
void add(int*, int*);
main( )
{
      int x = 10, y = 20;
      add(&x, &y);
      printf("The values of x and y in the calling function \n");
      printf("x=%d y=%d", x, y);
}
```

```
void add(int *a, int *b)
{
    *a=*a + *b;
    *b=*b + 2;
    printf("a=%d b=%d",*a,*b);
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
The values are a=30, b=22.
```

In the above program, the addresses of '*x*' and '*y*' are passed to '*a*' and '*b*'. Hence, they refer to the same address location as referenced by '*x*' and '*y*'. The addresses of the arguments are placed in the stack hence any modifications done in the 'called' function are made to the 'calling' function.

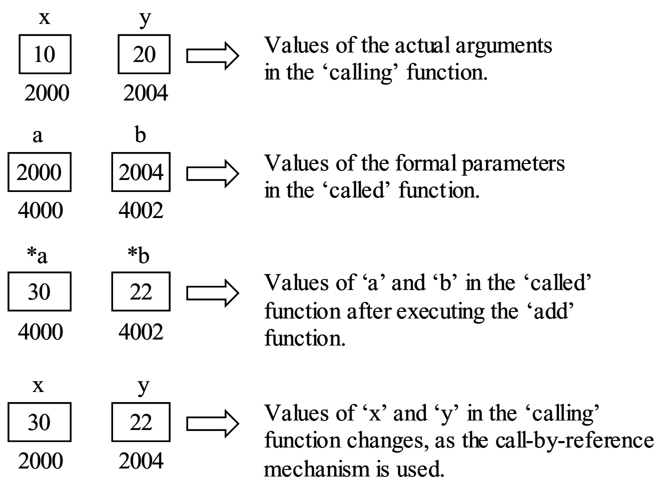

x
y
10
20
2000
2004
Values of the actual arguments in the 'calling' function.
a
b
2000
2004
4000
4002
Values of the formal parameters in the 'called' function.
*a
*b
30
22
4000
4002
Values of 'a' and 'b' in the 'called' function after executing the 'add' function.
x
y
30
22
2000
2004
Values of 'x' and 'y' in the 'calling' function changes, as the call-by-reference mechanism is used.

Figure (2): Call-by-Reference Mechanism

3.3 PASSING ARRAYS TO FUNCTIONS: IDEA OF CALL BY REFERENCE

Q19. Explain how arrays are passed to a function.

Answer :

Passing Arrays to a Function

Arrays are passed to a function by just sending the array name without the address operator. That is, passing an array name to a function is nothing but passing an address of the first element in the array. The function call with array name as a argument is as follows.

funcsia(array name);

There are two approaches for the declaration of an array in the called program. In the first approach, simple array notation is used which looks as follows.

int funcsia(int array[ ])

In the second approach, array is declared as a pointer in the function header, which looks as shown below.

int funcsia(int *array);

When passing a multidimensional array to a function, it is necessary to use the array syntax and specify the size of the dimensions. Whereas in case of passing a one dimensional array to a function, it is not necessary to specify the size of the dimension.

The declaration in the function header to receive three dimensional array is shown below.

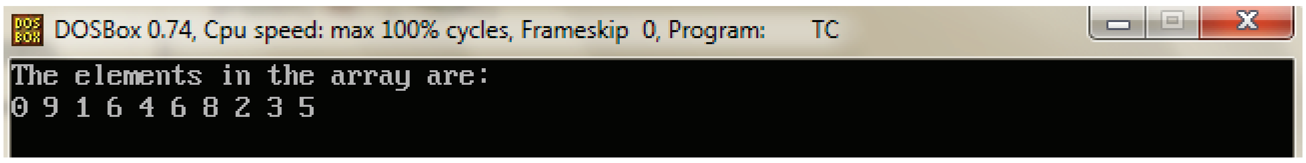
int funcsia(int array[ ] [2] [5])

Example

The following program illustrates passing an array to a function.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int array[ ] = {0, 9, 1, 6, 4, 6, 8, 2, 3, 5};
int i;
clrscr();
printf("The elements in the array are:\n");
for (i=0; i<10; i++)
{
 show (&array[i]);
}
getch();
return 0;
}
int show(int *n)
{
printf("%d", *n);
return ;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
The elements in the array are:
0 9 1 6 4 6 8 2 3 5
```

UNIT 4

RECURSION AND STRUCTURE

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Define recursion.

Answer :

Recursion

Recursion is the process or technique by which a function calls itself. A recursive function contains a statement within its body, which calls the same function. Thus, it is also called as circular definition. A recursion can be classified as direct recursion and indirect recursion. In direct recursion, the function calls itself and in the indirect recursion, a function (f1) calls another function (f2) and the called function (f2) calls the calling function (f1).

Example

Implementation of factorial using direct recursion.

Q2. What are the advantages of Recursion?

Answer :

The advantages of recursion are as follows,

- ❖ Recursion solves the problem in the most general way as possible.
- ❖ Recursive function is small, simple and more reliable than other coded versions of the program.
- ❖ Recursion is used to solve the more complex problems that have repetitive structure.
- ❖ Recursion is more useful because sometimes a problem is naturally recursive.
- ❖ For some problems it is easy to understand them using recursion.

Q3. Define structure.

Answer :

Structure Definition

A structure is a collection of data items of different data types under a single name.

(or)

A structure is a collection of heterogeneous data items.

Example

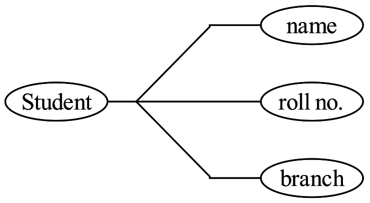


Figure: Structure of a Student

Q4. List the advantages of structure.

Answer :

The advantages of structure are given as follows,

1. They can create complex data types.
2. They can easily represent complex numbers in mathematics that consist of real and imaginary parts.
3. They can hold the locations of points in multi-dimensional space.
4. They cannot only hold data but can also be used as members of other structures. For instance, arrays of structures can store lists of data with regular fields such as databases.
5. Pointers to structures can be used to create linked lists, trees, etc.

Q5. What is array of structures?

Answer :

Array is a collection of similar data types. Grouping structure variables into one array is referred to as an array of structures. C permits us to declare arrays of structures as shown below.

```
struct student
    {
        int marks1;
        int marks2;
        int marks3;
    }st[3];
```

Where st[3] is an array of 3 elements containing three objects of student structure i.e., each element st[3] has structure of student with three members that are marks1, marks2 and marks3.

Q6. How to initialize a structure?

Answer :

Structures can be initialized in two ways, they are as follows,

1. By assigning values while declaring structure variables.

   Example

   ```
   struct employee e1 = {"JOHN", "32", "MALE", "DESIGNER"};
   ```

2. By assigning values separately to each member.

   Example

   ```
   strcpy(e1.name, "JOHN");
   strcpy(e1.age = 32);
   strcpy(e1.gender, "MALE");
   strcpy(e1.department, "DESIGNER");
   ```

Q7. Explain how to find the size of the structure.

Answer :

Size of Structure

To evaluate the size of structure a special operator called "sizeof" operator is used,

Syntax

```
sizeof(struct s);
```

The above expression will return the number of bytes to store the members of structure s.

Example

```
typedef struct s
{
char sname[20];
int sage;
}st;
st s = {“abcdef”, 25};
int main( )
{
printf(“\nSize of structure : %d”, sizeof(s));
return 0;
}
```

In the above example, the output is,

```
Size of structure : 22
```

PART-B
ESSAY QUESTIONS WITH SOLUTIONS

4.1 RECURSION

4.1.1 Recursion as a Different Way of Solving Problems

Q8. What is recursion? What are its advantages?

Answer :

Recursion

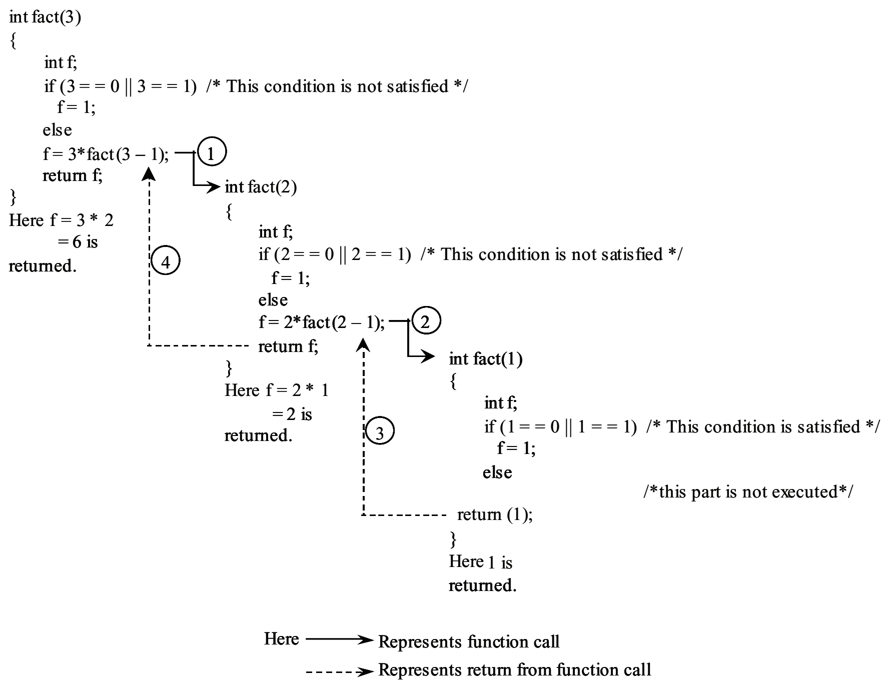
Recursion is the process or technique by which a function calls itself. A recursive function contains a statement within its body, which calls the same function. Thus, it is also called as circular definition. It is completely a different way of solving problems. It reduces the problem into another problem of same type instead of dividing the solution into steps. A recursion can be classified as direct recursion and indirect recursion. In direct recursion, the function calls itself and in the indirect recursion, a function (f1) calls another function (f2) and the called function (f2) calls the calling function (f1).

Example: Implementation of Factorial using Recursion in C

```
int fact(int n)
{
        int f;
        if(n == 0 || n == 1)
                f =1;
        else
        f = n * fact(n – 1); /* Here fact function is calling itself*/
        return f;
}
```

Now let us see how factorial of a number is obtained using this function.

Let us find factorial of 3.



```
int fact(3)
{
      int f;
      if (3 = = 0 || 3 = = 1)  /* This condition is not satisfied */
        f = 1;
      else
      f = 3*fact(3 – 1);  ─(1)
      return f;
}
Here f = 3 * 2
        = 6 is
returned.

                            int fact(2)
                            {
                                  int f;
                                  if (2 = = 0 || 2 = = 1)  /* This condition is not satisfied */
                                    f = 1;
                                  else
                                  f = 2*fact(2 – 1);  ─(2)
                                  return f;
                            }
                            Here f = 2 * 1
                                    = 2 is
                            returned.

                                              int fact(1)
                                              {
                                                   int f;
                                                   if (1 = = 0 || 1 = = 1)  /* This condition is satisfied */
                                                     f = 1;
                                                   else
                                                                      /*this part is not executed*/
                                                return (1);
                                              }
                                              Here 1 is
                                              returned.
```

1
2
3
4
Here ⟶ Represents function call
- - - -> Represents return from function call

Hence the value of fact(3) is 6.

When a recursive call is made, the parameters and return address gets saved on a stack. The stack gets wrapped when the control is returned back.

Advantages of Recursion

❖ Recursion solves the problem in the most general way as possible.

❖ Recursive function is small, simple and more reliable than other coded versions of the program.

❖ Recursion is used to solve the more complex problems that have repetitive structure.

❖ Recursion is more useful because sometimes a problem is naturally recursive.

❖ For some problems it is easy to understand them using recursion.

4.1.2 Example Programs Such as Finding Factorial, Fibonacci Series

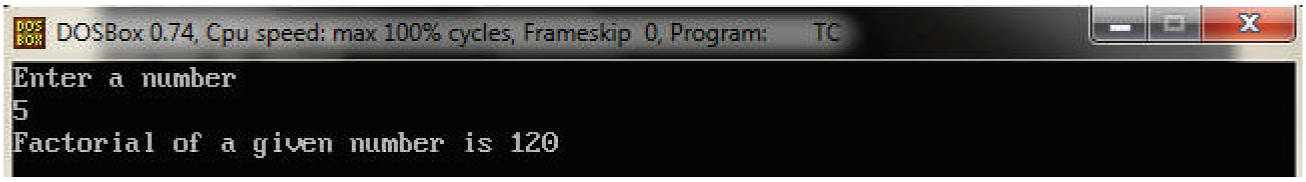
Q9. Write a C program to find the factorial of a number using recursion.

Answer :

Program

```
#include<stdio.h>
#include<conio.h>
int fact(int x)
{
if(x <= 1)
return(1);
else
return(x *fact(x – 1));
}
void main( )
{
int n, res;
clrscr( );
printf("Enter a number \n");
scanf("%d", &n);
res = fact(n);
printf("Factorial of a given number is %d", res);
getch( );
}
```

Output



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter a number
5
Factorial of a given number is 120
```

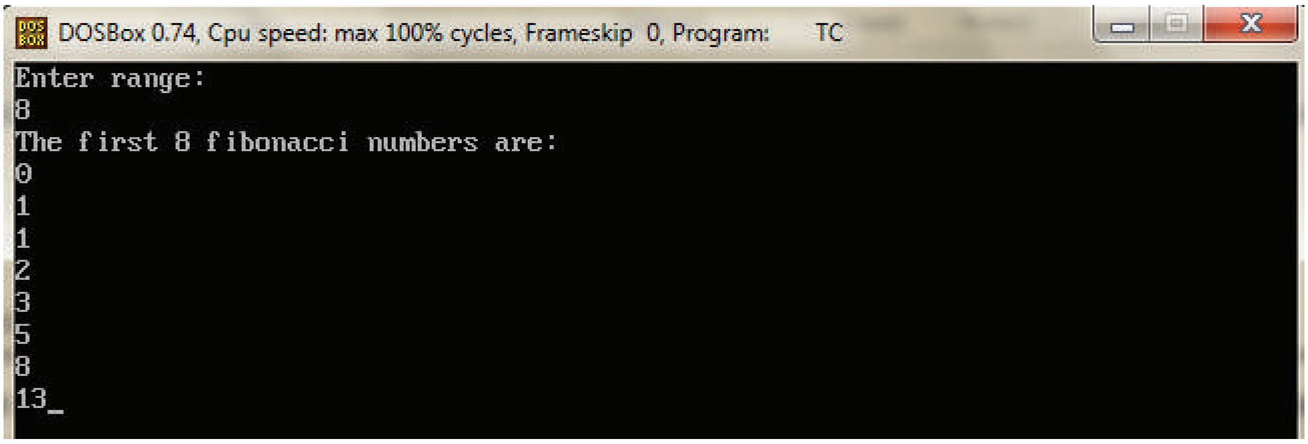
Q10. Write a complete C program that reads a positive integer N, compute the first N fibonacci numbers using recursion and print the results.

Answer :

Program

```
#include<stdio.h>
#include<conio.h>
int fb(int n)
{
if(n == 0 || n == 1)
return (n);
else
return (fb(n – 1) + fb(n – 2));
}
void main( )
{
int n, i, res;
clrscr();
printf("Enter range:\n");
scanf("%d", &n);
printf("The first %d fibonacci numbers are:",n);
for (i = 0; i <= n – 1; i++)
{
res = fb(i);
printf("\n%d", res);
}
getch( );
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter range:
8
The first 8 fibonacci numbers are:
0
1
1
2
3
5
8
13_
```

4.2 STRUCTURE : STRUCTURES, DEFINING STRUCTURES AND ARRAY OF STRUCTURES

Q11. Define structure. Explain how a structure is declared.

Answer :

Structure Definition

A structure is a collection of data items of different data types under a single name.

(or)

A structure is a collection of heterogeneous data items.

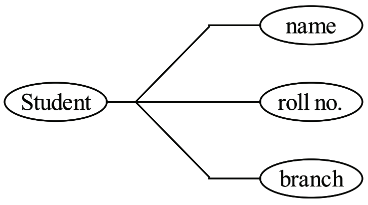


Figure: Structure of a Student

Declaring a Structure

A structure is declared as follows,

```
struct <tag>
{
data_type member1;
data_type  member2;
⋮
data_type member n;
};
```

❖ In this declaration struct is a keyword.

❖ <tag> is the name given to the structure declared.

❖ data_type can be int, float, char or any other data types.

❖ member1, member2,..., membern are members of a structure. Individual members can be variables, pointers, arrays or other structures.

❖ Individual members cannot be initialized within structure definition.

❖ Members of structure themselves do not occupy any space. Memory is allocated only after declaring structure variables.

❖ After structure has been declared, structure type variables are declared as follows.

struct <tag> variable1, variable2,..., variable n; Here, variable1, variable2,..., variable n are **structure variables** of type tag.

❖ After closing bracket semicolon ';' must be specified.

❖ Structure can also be declared as,

```
struct tag
{
type1    data1;
type2    data2;
⋮         ⋮
typen    datan;
}variable1, variable2, ...., variable n;
```

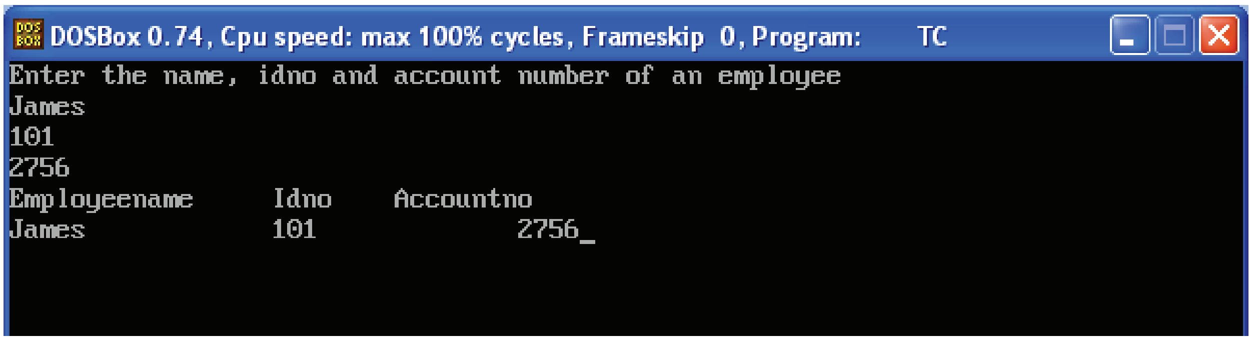
In this declaration, tag is optional.

Program

```
#include <stdio.h>
#include <conio.h>
void main( )
{
    struct employee
    {
        char name[20];
        int idno;
        int acno;
    };
```

```
        int i, n;
        struct employee emp;
        clrscr( );
        printf("Enter the name, idno and account number of an employee\n");
        scanf("%s", &emp.name);
        scanf("%d", &emp.idno);
        scanf("%d", &emp.acno);
        printf("Employeename\t Idno\t Accountno\n");
        printf("%s\t\t %d\t\t %d", emp.name, emp.idno, emp.acno);
        getch( );
    }
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the name, idno and account number of an employee
James
101
2756
Employeename      Idno     Accountno
James             101              2756_
```

Q12. Explain how to access members of structure. What are the advantages of structure?

Answer :

Accessing Members of Structure

Structure members can be accessed by writing variable_name.member;

Dot(**.**) operator links structure variable to the data member. It is the highest precedence operator and associativity is left to right. For example, the members of structure P1 are accessed as P1.name, P1.age, P1.gender, P1.occupation.

Initializing a Structure

❖ One way is to assign values at the time of declaring structure variables.

  struct person P1 = {"Ravi", 21,"Male","Student"};

❖ Another way is to assign values to each member separately as shown below.

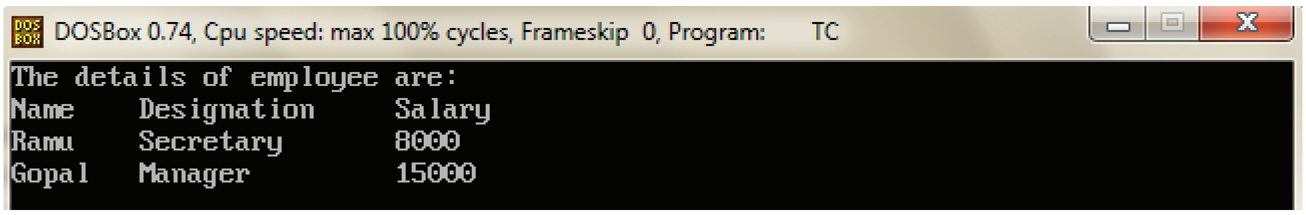
```
strcpy(P1.name,"Ravi");
P1.age = 21;
strcpy(P1.gender,"Male");
strcpy(P1.occupation,"Student");
```

Example

Here is an example demonstrating the structure with initialized values.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
struct organisation
{
char name[20];
char designation[10];
int sal;
};
struct organisation emp1 = {"Ramu", "Secretary", 8000};
struct organisation emp2 = {"Gopal", "Manager  ", 15000};
clrscr();
printf("The details of employee are:\n");
printf("Name\tDesignation\tSalary\n");
printf("%s\t%s\t%d\n",emp1.name, emp1.designation, emp1.sal);
printf("%s\t%s\t%d\n",emp2.name, emp2.designation, emp2.sal);
getch();
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
The details of employee are:
Name    Designation     Salary
Ramu    Secretary       8000
Gopal   Manager         15000
```

Advantages of Structure

1. They can create complex data types.
2. They can easily represent complex numbers in mathematics that consist of real and imaginary parts.
3. They can hold the locations of points in multi-dimensional space.
4. They cannot only hold data but can also be used as members of other structures. For instance, arrays of structures can store lists of data with regular fields such as databases.
5. Pointers to structures can be used to create linked lists, trees, etc.

Q13. Explain the concept of Array of structures.

Answer :

Declaration of Array of Structures

Array is a collection of similar data types. Grouping structure variables into one array is referred to as an array of structures. C permits us to declare arrays of structures as shown below,

```
struct student
{
    int marks1;
    int marks2;
    int marks3;
}st[3];
```

Where st[3] is an array of 3 elements containing three objects of student structure i.e., each element st[3] has structure of student with three members that are marks1, marks2 and marks3.

Usage of Array of Structures

Structure is a collection of dissimilar data types, for example in a structure for representing student information, the ideal members include studentname, rollno, marks and it can be programmatically represented as shown below.

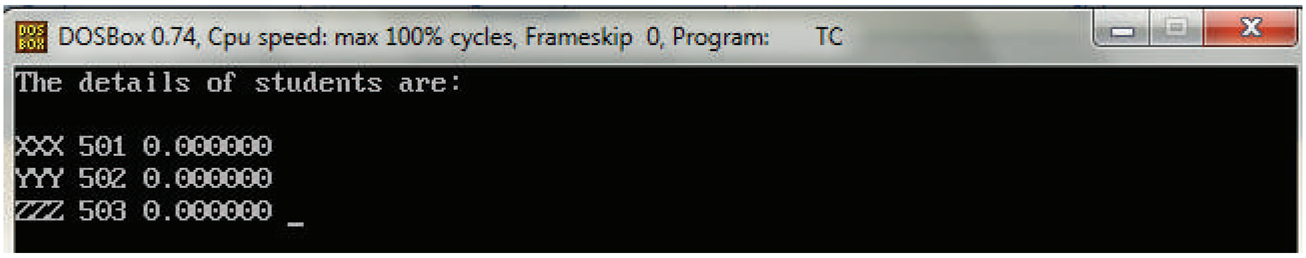
```
struct student
{
        char studentname[30];
        int rollno;
        int marks;
};
```

In order to store data of 100 students, 100 different structure variables say st1, st2,.., st100 i.e., struct student st1, st2,..., st100; are required. Which is definitely not very convenient. A better approach would be to use an array of structures i.e., store data of 100 students into an array, wherein each element of array would represent individual student information.

Example

```
#include <stdio.h>
#include<conio.h>
struct student
{
char sname[20];
int rollno;
float average;
};
void main( )
{
int i;
clrscr();
struct student st[3] = {{“XXX”, 501, 79.6}, {“YYY”, 502, 80.3}, {“ZZZ”, 503, 96.25}};
printf(“The details of the student are:\n”);
for(i = 0; i <= 2; i++)
{
printf(“\n%s %d %f “, st[i].sname, st[i].rollno, st[i].average);
}
getch();
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
The details of students are:

XXX 501 0.000000
YYY 502 0.000000
ZZZ 503 0.000000 _
```

UNIT 5

POINTERS AND INTRODUCTION TO FILE HANDLING

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Define a pointer. Give an example.

Answer :

Pointer

Pointer is a variable that holds the memory address of another variable.

Example

int x = 2;

This statement will allow the system to locate integer variable ‘*x*’ and store ‘2’ in it. If ‘*x*’ is stored at location ‘1210’ by the system, then address of the variable ‘*x*’ will be ‘1210’.

Let ‘*y*’ be another variable stored at location ‘1315’ and ‘1210’ be stored in ‘*y*’. Since, *y* stores the address of *x*, it is a pointer. It is declared as follows,

```
int *y;
y = 1210;
   or
y = &x;
```

As address of ‘*x*’ is stored in ‘*y*’, the value of ‘*x*’ can be accessed using ‘*y*’ and it is thus said to be a ‘pointer’ to ‘*x*’.

Q2. What is the use of & (ampersand) operator in C?

Answer :

Ampersand & operator in ‘C’ is used to obtain (or return) the address of a variable with which it is associated. The ‘&’ is known as ‘address of’ operator. The address of a variable is stored in another variable called pointer variable. For example, consider the statement

p = &n;

&n → It returns the address of variable ‘n’

p → It is a pointer variable.

Q3. What is indirection operator?

Answer :

Asterisk * operator is used to access the value of a variable. This can be done, by declaring a pointer variable which is then assigned to another variable whose value is needed to be accessed (for example, x = *p). A variable is said to be declared as a pointer variable if it is associated with an asterisk (for example, *p). The ‘*’ is known as the indirection operator or ‘dereferencing’ operator.

Consider the below statements,

```
int a, *p, x;
a = 20;
p = &a;
x = *p;
```

Q4. Define file.

Answer :

A file is a permanent named unit that holds large volumes of related data. It stores the data in secondary or auxiliary memory such as CDs, tapes, DVDs and hard disks. Hence, it is a permanently stored memory. It is used to support the volatile nature of main memory. That is, when the system is shut down, main memory looses the data and relies on secondary storage. When the system is switched on, main memory gets loaded with the data from the files. Moreover, main memory cannot hold the entire data at any point. Rather, it accesses the data from files as and when needed.

In addition to reading data from files, main memory will also write data into the files if it has made any changes to the data. The operating speeds of main memory and secondary memory differ largely, hence, a temporary storage is needed to fill the speed difference between them. A buffer is such a temporary storage used for data transfers between main memory and secondary memory.

During the read operation, the buffer holds data till the program in main memory is ready to accept data. During the write operation, the buffer holds data till the data is sufficient enough to be written into the file. The entire process of buffering is managed by device drivers or access methods of the operating system.

Q5. Define text files.

Answer :

Text files stores data only in the form of characters. The non-character datatypes cannot be directly stored in the text file without converting them into a sequence of characters. Input/Output functions can be used to convert non-character datatypes to characters while writing into the text files. If input has to be read from a text file, it is read in the form of character sequence, converted to their appropriate internal formats and then stored in memory.

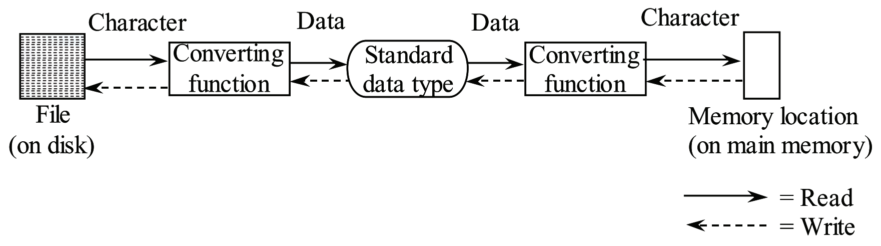


Figure: Process of Reading and Writing Text Files

Q6. What are binary files?

Answer :

In contrast to text files in a binary file, the data is stored in the internal format of computer, the format similar to that of memory. Binary streams also called as block I/O functions are used for reading and writing binary files.

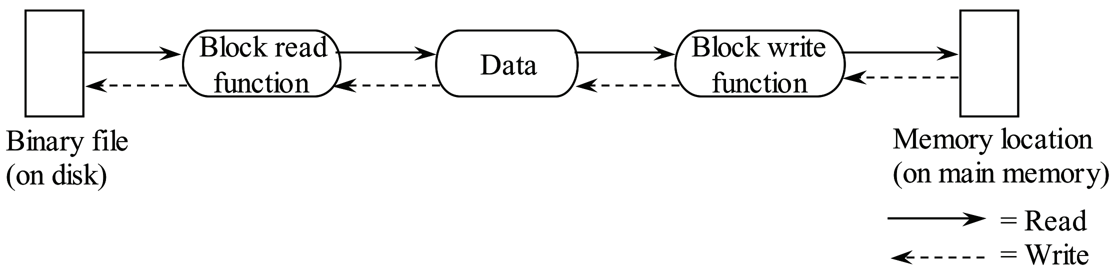


Figure: Process of Reading and Writing Binary Files

Q7. Write about fputc( ) and fgetc( ).

Answer :

fputc( )

The fputc functions are commonly used to write a character to a predefined or a user-defined file stream. The first parameter in this function represents the character to be written while the second parameter is the file itself. The function returns the character if it is successfully written. Otherwise, it returns EoF.

The prototype function is given below,

    int fputc(int oneChar, File *spOut);

fgetc( )

The fgetc functions are used to read the next character from the user defined or predefined file stream and converts it into an integer. During the read process, if an error is encountered, then this function returns EoF. The prototype function for fgetc( ) is given below,

    int fgetc(File *spMyFile);

Q8. Write about fread( ) and fwrite( ).

Answer :

fread( )

This block I/O function reads the data from the binary files as per the number of bytes specified. The bytes that are read are then placed into memory at the location specified.

Syntax

    int fread(void* S, int size, int counter, FILE* fp);

Where, *S* is the pointer to the input area in the memory and *S* uses a void pointer which indicates that any pointer type can be passed to the function. The amount of data to be transferred is given by the product of parameters 'size' and 'counter'. The associated stream from where data has to be read is given by FILE* fp.

fwrite( )

The fwrite( ) block I/O function is used to write to a binary file the number of items that are specified.

Syntax

int fwrite(void* S1, int size, int counter, FILE* fp1); Where, S1 is the pointer to the input area in the memory and it uses a void pointer which indicates that any type of pointer can be passed to the function. The amount of data to be transferred is given by "size*counter". The associated stream is obtained from FILE* fp1.

Q9. Discuss in brief about ftell( ) and fseek( ).

Answer :

ftell( )

This function gives the current position of file pointer in terms of bytes from the beginning.

Syntax

    n = ftell(fp);

Here 'n' gives the current position of the file pointer from the beginning. It gives the information regarding number of bytes read or written already.

fseek( )

fseek( ) is a file function that is used to move the file pointer to the desired location in the file. It is generally used when a particular part of the file is to be accessed. The main purpose of file fseek( ) is to position the file pointer to any location on the stream.

Syntax

    fseek(fileptr, offset, position);

'fileptr' is a file pointer.

Offset specifies the number or variable of type long to reposition the file pointer. It tells the number of bytes to be moved. Offset can either be positive or negative number where positive integer is used to reposition the pointer forward and negative integer is used to move the pointer backward.

PART-B
ESSAY QUESTIONS WITH SOLUTIONS

5.1 POINTERS

5.1.1 Idea of Pointers, Defining Pointers

Q10. What is a pointer? List out the reasons for using pointers.

Answer :

Pointer

A variable that stores address of another variable is called a 'pointer'.

Example

int x = 2;

This statement will allow the system to locate integer variable '*x*' and store '2' in that location. If '*x*' is stored at location '1210' by the system, then address of the variable '*x*' will be '1210'.

Let '*y*' be another variable stored at location '1315' and '1210' be stored in '*y*'. Since, *y* stores the address of *x*, it is a pointer. It is declared as follows,

```
int * y;
y = 1210;
   or
y = &x;
```

As address of '*x*' is stored in '*y*', the value of '*x*' can be accessed using '*y*' and is thus said to be a 'pointer' to '*x*'.

Reasons for using Pointers

1. Pointers are used in indirect addressing which is most oftenly used in assembly languages.
2. Pointers are used in dynamic memory management (i.e., dynamic memory allocation and deallocation).
3. A pointer is used to access a location in the memory whose storage is allocated dynamically. This dynamically allocated storage is called a heap.
4. Variables that are dynamically allocated from the heap are called heap dynamic variables. The heap dynamic variables do not have the associated identifiers. Hence, they can be referenced only by pointers and reference type variables.
5. Use of pointers makes the coding process simple and easy.
6. Pointers are used to create recursive types that are important in data structures and algorithms.
7. It enhances the language's writability.

Q11. Write short notes on,

(a) Pointer constants

(b) Pointer values

(c) Pointer variables.

Answer :

(a) Pointer Constants

The pointer constants are the set of addresses or locations within the memory, which are assigned to the variables. The pointer constants themselves cannot be changed, but the pointer constants that are being assigned to the variables during runtime, change in each run. This is because, the modern operating system assigns addresses to the variables based on its convenience. Thus, the pointer constants must be referred symbolically.

Example

Consider a variable 'var'. If it is stored at memory address '528960' in the first execution, it is not necessary that in the next execution it will be stored at the same address. It may be stored at some other address such as '256392'.

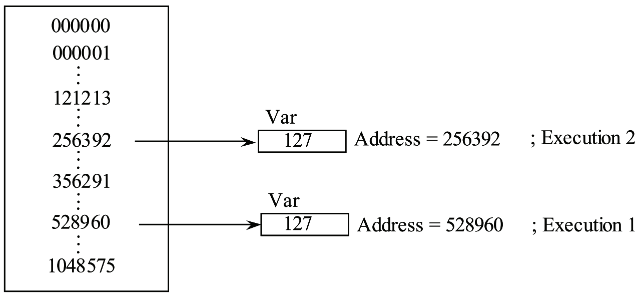


Figure (1): Pointer Constants

(b) Pointer Values

The pointer values are nothing but the values of pointer constants (addresses) that are assigned to variables. The address of a variable is obtained using the address operator(&).

The expression that obtains the address of a variable is called **'address expression'**. The format of an address expression is,

$\boxed{\text{\& Variable Name}}$

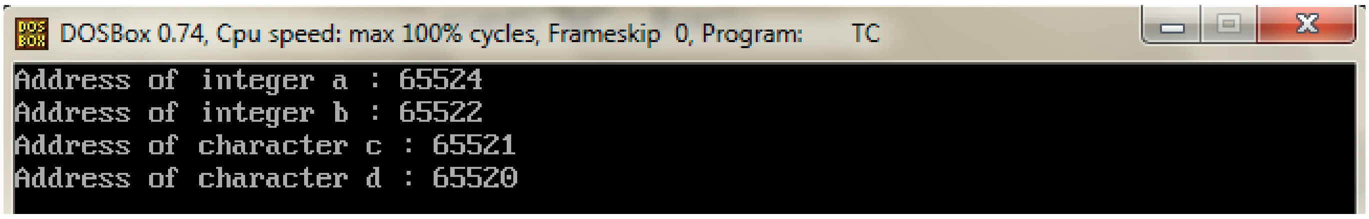
This expression is a type of unary expression as it requires only a single operand i.e., the variable name.

The addresses are assigned to variables based on their datatype. The integer takes 4 bytes and the characters take only 1 byte of memory space. Thus, the first byte that stores the variable will be the address of that variable.

Example

```
#include<stdio.h>
#include<conio.h>
int main( )
{
int a=10, b=20;
char c, d;
clrscr();
printf("Address of integer a : %u \n", &a);
printf("Address of integer b : %u \n", &b);
printf("Address of character c : %u \n", &c);
printf("Address of character d : %u \n", &d);
getch();
return 0;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Address of integer a : 65524
Address of integer b : 65522
Address of character c : 65521
Address of character d : 65520
```

(c) Pointer Variables

A pointer variable is a variable that stores the address of another variable.

Example

Consider variables **'var'** and **'ptr-var'**. The variable **'var'** stores a value '127' at address '562962', and, variable 'ptr-var' stores the address of **'var'**, i.e., 562962. Then **'ptr-var'** is said to be the **pointer variable**.

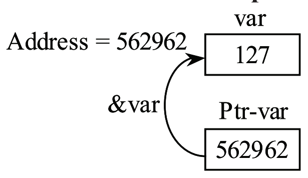


Figure (1): Pointer Variable

There can be more than one pointer variable pointing to a single variable. That is, the address of a variable can be stored in multiple pointer variables.

Example

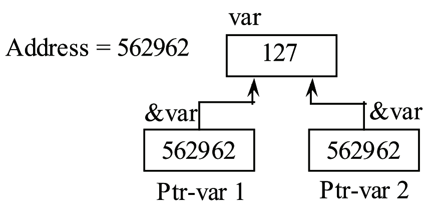


Figure (2): Multiple Pointer Variable

A pointer variable that does not store address of any variable, will store a null pointer constant, i.e., NULL. NULL is a macro that has an integer value '0' and is defined in 'stdio.h' header file.

Q12. Explain the process of accessing a variable through its pointer. Give an example.

Answer :

Accessing a Variable Through Pointer

When a variable (E.g.: int n = 5) is declared then the 'C' compiler performs the following functions,

(i) Allocates space in the memory for the variable.

(ii) Associates the name with this memory location.

(iii) Stores the value at this memory location.

The memory location in the memory is a whole number (E.g.: 6490, 3000, ... etc.).

'C' provides '&' ('address of') operator, which returns the address of associated variable.

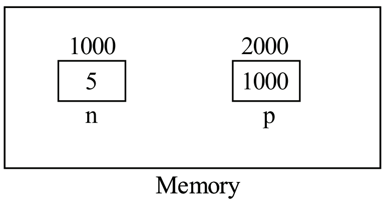
Example

&n → Returns the address of variable 'n'.

The address of the variable can be collected in an another variable (known as pointer variable). The pointer variable holds the address of a variable.

Example

```
int *pi
p = &n;
p → It is a pointer variable
```



Memory

Since, 'p' is a variable, memory space is also allocated for 'p'.

Note

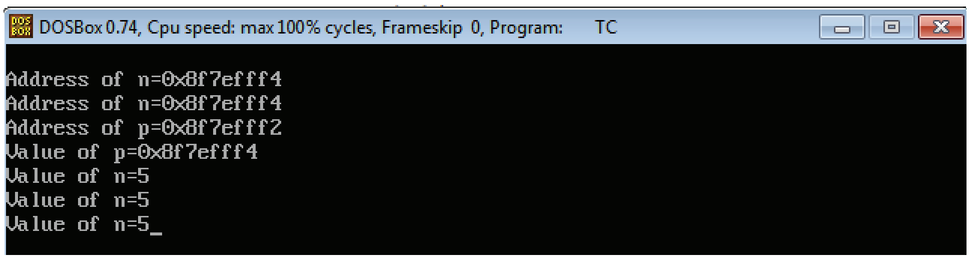
'*' is known as 'value of address' operator.

The variable 'n' can be accessed through its pointer 'p' in different ways which is illustrated in the following program.

Program

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int n = 5;
    int *p;
    p = &n;
    printf("\n Address of n = %u",&n);
    printf("\n Address of n = %u",p);
    printf("\n Address of p = %u",&p);
    printf("\n Value of p = %d",p);
    printf("\n Value of n = %d",n);
    printf("\n Value of n = %d",*(&n));
    printf("\n Value of n = %d",*p);
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Address of n=0x8f7efff4
Address of n=0x8f7efff4
Address of p=0x8f7efff2
Value of p=0x8f7efff4
Value of n=5
Value of n=5
Value of n=5_
```

In the above program, we can observe the statement,

cout<<"\n Value of n = %d", *(&n);

In this statement, we are accessing the value of variable 'n' through 'value at address' (*) operator. The '*' operator is associated with the pointer variable.

Similarly, value of 'n' can be accessed by statement, cout<<"\n Value of n = %d" *p;

Here, 'p' contains the address of 'n' and *p gives the value at address stored in 'p'. Thus, the value of 'n' can be accessed by, n, *(&n), *p.

Q13. Explain the process of declaring and initializing pointers. Give an example.

Answer :

Declaration of Pointers

Pointers are declared as follows,

datatype *ptr_variable;

Here, '*' indicates that ptr_variable is a pointer variable representing value stored at a particular address.

Example

```
int *p;
char *q;
float *f;
```

Here,

'p'= Pointing to address location where an integer type data is stored.

'q' = Pointing to address location where character type data is stored.

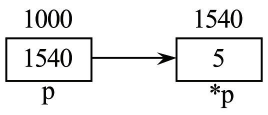
'f' = Pointing to address location where float type data is stored.

Initialization of Pointer Variable

Pointer variable can be initialized after declaration as given below,

```
int *p;
*p = 5;
```

On compilation of the above statement, the value 5 is stored in an address pointed by p. Suppose, p points to an address 1540. So, 5 is stored in location 1540.



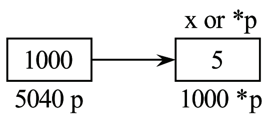
Here p is 1540 and *p gives value 5.

Assigning of Address of Variable to Pointer Variable

A pointer variable is used to point to addresses of other variables.

```
int *p;
int x = 5;
p = &x;
```

Here, address of variable x is assigned to pointer p.



*p gives values at location 1000 i.e., 5 which is same as value of x.

Here, the '&' used is same as done in scanf function. '&' is called "address of" operator. This is used to assign the address of a variable next to it.

Example

```
int *p = &x;
```

This statement will transfer the address of x to p.

Note

In the above initialization, the address of x is assigned to p but not *p,

p will point to address of x.

*p will point to value of x.

&p will point to address of p.

Example

```
printf("%d", p);
printf("%d", *p);
printf("%d", &p);
```

will give the output as,

```
1000    [address of x]
5       [value of x]
5040    [address of p]
```

5.1.2 Use of Pointers in Self-referential Structures, Notion of Linked List (No Implementation)

Q14. Illustrate the usage of pointers in self referential structures. Also discuss the usage of self referential structures in linked list.

Answer :

Pointers in Self Referential Structures

A structure definition which includes at least one member as a pointer to the same structure is known as self-referential structure. Pointer can be used to point the members of self referential structures.

Example

```
struct student
{
    char name[50];
    int rollno;
    struct student *ptr;
};
```

In the above example, the pointer ptr will point to the structure element student.

Self Referential Structures in Linked Lists

Self-referential structures are mainly used in the implementation of linked data structures like lists, trees etc.

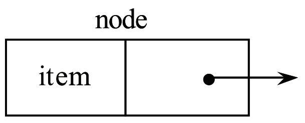
Consider an implementation of linked-lists. A linked list is a collection of nodes, where each node points to the next node in the list. The advantage of using linked lists is that the data can be added or deleted easily.

In a linked list, each node consists of two items, one containing the data and the other containing the address of the next element.

Example

```
struct node
{
        char item [15];
        struct node *ptrnode;
};
```

The structure declared is of type node. It consists of two members. The first member is an char type i.e., item and the second member is a pointer to the next node in the list, which is shown below.



In a linked list, nodes represent self-referential structures and the arrows represent pointers that makes a linked list a self-referential structure.

5.2 INTRODUCTION TO FILE HANDLING

Q15. Define the following,

(i) File

(ii) File name

(iii) File information table.

Answer :

(i) File

A file is a permanent named unit that holds large volumes of related data. It stores the data in secondary or auxiliary memory such as CDs, tapes, DVDs and hard disks. Hence, it is a permanently stored memory. It is used to support the volatile nature of main memory. That is, when the system is shut down, main memory looses the data and relies on secondary storage. When the system is switched on, main memory gets loaded with the data from the files. Moreover, main memory cannot hold the entire data at any point. Rather, it accesses the data from files as and when needed.

In addition to reading data from files, main memory will also write data into the files if it has made any changes to the data. The operating speeds of main memory and secondary memory differ largely, hence, a temporary storage is needed to fill the speed difference between them. A buffer is such a temporary storage used for data transfers between main memory and secondary memory.

During the read operation, the buffer holds data till the program in main memory is ready to accept data. During the write operation, the buffer holds data till the data is sufficient enough to be written into the file. The entire process of buffering is managed by device drivers or access methods of the operating system.

Each file has a beginning and ending. The programs in main memory must be allowed to read only till the end of file. Otherwise, the programs will get garbage data. To suffice this, special "end of file" characters are used and passed automatically by the device to the programs. The program must test the end of file on every read operation, which is usually done in loop control statements.

While a program, reading from or writing to a file the program needs information about the file. This includes the position of the current character in the file, the name of the operating system and much more. 'C' offers a file structure that is defined in stdio.h to store the file information. The identifier FILE of stdio.h is used as the data type in declaring a file.

Each operating system has its own set of rules for naming files. Programmers must follow these rules for naming any file.

(ii) File Name

The file name must be specified for a particular file. The file name is typically a string of characters. The file names can be "Data.txt", "Add.C" or "PR1.h" and so on.

The extensions can be given depending upon the usage of the file such as,

❖ If the file is used for storing text use '.txt' extension.

❖ If the file stores 'C' program give '.c' extension.

❖ If it is a header file, then give '.h' extension and so on.

(iii) File Information Table

To read or write a file in a program, certain information is required like file name, the location of the current character of the file etc. Such information is stored in a predefined structure called file information table. This table is defined by " stdio.h" header file with 'FILE' as its identifier. Hence, whenever a file is to be read or written, it is declared using the 'FILE' type.

Q16. Describe various types of files with an example for each.

OR

Explain what is a text file and what is a binary file.

Answer :

A programming language typically supports two types of files. They are text files and binary files.

1. Text Files

A text file stores data only in the form of characters. The non-character datatypes cannot be directly stored in the text file without converting them into a sequence of characters. Input/ Output functions can be used to convert non-character datatypes to characters while writing into the text files. If input has to be read from a text file, it is read in the form of character sequence, converted to their appropriate internal formats and then stored in memory.

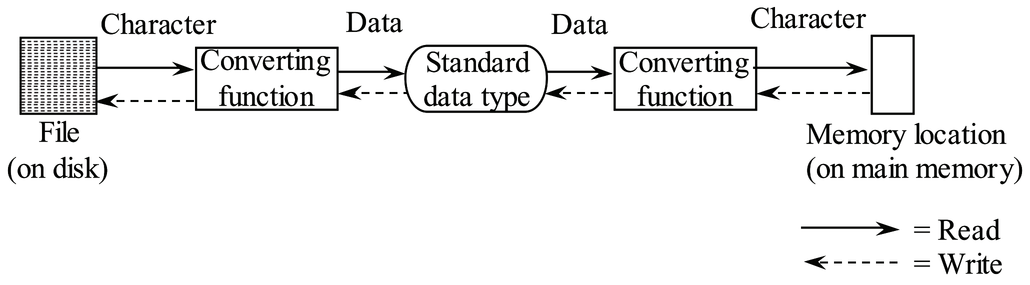


Figure: Process of Reading and Writing Text Files

The converting function can be one of the following three types,

(i) Formatting I/O functions

(ii) Character I/O functions

(iii) String I/O functions.

(i) Formatting I/O Functions

In case of reading an input, formatting I/O functions convert the series of input characters to standard types. On the other hand, while displaying the output, standard datatypes are converted to sequence of characters.

Example

scanf( ), printf( ).

(ii) Character I/O Functions

These functions can read or write only one character at a time.

Example

getchar( ), putchar( ).

(iii) String I/O Functions

These functions can read or write strings of characters.

Example

fgets( ), fputs( ).

Note that these functions are used only with text files.

2. Binary Files

In contrast to text files in a binary file, the data is stored in the internal format of computer, the format similar to that of memory. Binary streams also called as block I/O functions are used for reading and writing binary files.

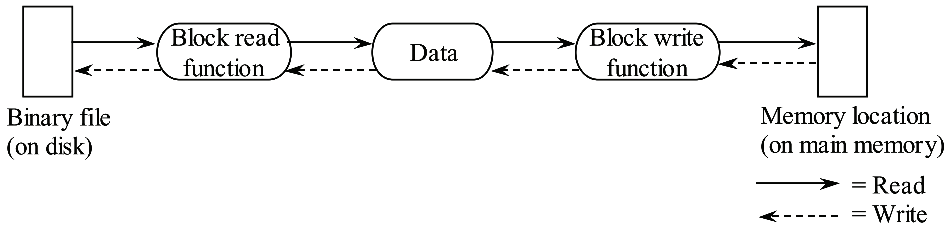


Figure: Process of Reading and Writing Binary Files

Q17. List and explain the stream functions for text files along with their prototypes.

Answer :

Stream Functions for Text Files

The following are the list of stream functions available. They can be applied on text files,

1. fputc( )
2. fputs( )
3. fgetc( )

4. fgets( )
5. fprintf( )
6. fscanf( )
7. fopen( )
8. fclose( ).

1. fputc( )

The fputc functions are commonly used to write a character to a predefined or a user-defined file stream. The first parameter in this function represents the character to be written while the second parameter is the file itself. The function returns the character if it is successfully written. Otherwise, it returns EoF.

The prototype function is given below,

int fputc(int oneChar, File *spOut);

2. fputs( )

fputs( ) function writes to a file. It removes null character from string. If the file is specified as 'stdout' then fputs writes to the standard output. This function returns non-negative integer if no errors occur. Otherwise, it returns EoF.

The prototype function is given below,

int fputs(const char *Pstring, File *P);

3. fgetc( )

The fgetc functions are used to read the next character from the user defined or predefined file stream and converts it into an integer. During the read process, if an error is encountered, then this function returns EoF. The prototype function for fgetc( ) is given below.

int fgetc(File *spMyFile);

4. fgets( )

The fgets( ) function accepts data from standard input or a file. The parameters for this function are a string pointer, array size and stream. If the stream is specified as "stdin" then, it reads data from the standard input (i.e., keyboard). The read operation continues until a new line character is found or until end-of-file is reached. The prototype function is given below,

char* fgets(char* strptr, int size, File *sp);

5. fprintf( )

fprintf( ) function is used for writing records into a file. Its function is analogous to the printf( ) function. The function outputs the matter given in quotations in fprintf( ) to the display screen. This function handles a group of mixed data simultaneously.

fprintf( ) function consists of different parameters.

fprintf(file_descriptor, "format string", list);

6. fscanf( )

This function is used for reading records from the file.

fscanf( ) function is analogous to the scanf( ) function except that it works on files.

fscanf(file_descriptor, "format string", address of list);

7. fopen( )

The fopen( )function is used to open a file. It has two parameters,

(i) File name

(ii) File open mode.

The fopen( ) is common for both the text files and the binary files, the only difference is in the mode being used.

The prototype structure for this function is given below,

fopen(file_name, file_open_mode);

The modes of fopen( ) function for text files include,

w, r, a, w+, a+, r+

8. fclose( )

After the required operations such as reading, writing or appending a file are done, the file needs to be closed. This can be done using fclose( ) function.

The prototype for this function is given below,

fclose(file_pointer);

The fclose function has only one parameter (i.e., file_pointer). This parameter represents a pointer to the file that should be closed. If multiple files are to be closed, then multiple fclose( ) functions must be written. If all the opened files are to be closed, then fcloseall( ) function can be used.

Q18. Write the features of text file and a binary file.

Answer :

Text File

A text file contains only textual information like alphabets, digits and special symbols. The ASCII code of these characters are stored in text files.

Features of Text Files

Some of the features of text files are as follows,

(i) In text file, data is stored in the human readable form.

(ii) A new line character (\n) is used to terminate every line of data in the text file.

(iii) A special character marker called "EOF" (End-Of-File) is used to indicate the end of a file.

Binary Files

A binary file is a collection of bytes. This collection is a compiled version of a C program (E.g: program1.exe), or music data stored in a wave file or a picture stored in a graphic file.

Features of Binary Files

Some of the features of binary files are as follows,

(i) No new line character is used to indicate the end of a line.

(ii) In binary files, data is stored in the manner similar to that in memory.

(iii) Like text files, binary files also uses “EOF” marker to indicate the end of a file.

Q19. Write in detail about various states of a file.

Answer :

States of a File

A file can be in one of the following three states,

(i) Read state

(ii) Write state

(iii) Error state.

(i) Read State

The file must be in the read state when some data has to be read from it. Opening a file in read state requires read mode (r) to be specified in the open statement. If the file does not exist then the open process would fail and file goes into error state.

(ii) Write State

A file must be in the write state when some data has to be written into it. Opening a file in write state requires either write mode or append mode to be specified in the open statement. If write mode is specified, writing operation is performed from the beginning of the file. If data is written into an already existing file, then all the previous contents are lost. In case of append mode, the data is written from the end of the file (i.e., appended). If the file opened in append mode already exists then new data is written after the existing data. A new file is created if no such file exists.

(iii) Error State

When an error occur during file operations like open, read or write, then the file goes into error state. If we try to write into a file that is in read state or if we try to read from a file that is in write state, then, in both cases, the state of file will be changed to error-state. No action can be performed if the file is in error state.

In addition to read, write and append modes, there is an update mode in which the files can be read as well as write. When a file is opened in update mode, both read and write operations can be performed but not simultaneously. Opening a file in update mode is done by adding ‘+’ to the basic mode.

Specifying the mode as r+ opens the file in read state. Similarly w+, a+ open the file in write state and append state.

Q20. What are the different functions of file?

OR

Write the syntax for opening a file with various modes and closing a file.

Answer :

Opening a File

A file needs to be opened when it is to be used. A file is opened by the open( ) function with two parameters in that function. These two parameters are,

1. file name and

2. file open mode.

Syntax

fopen(file_name, file_open_mode);

Example

fopen(“myFile.txt”, “r”);

The fopen( ) function is defined in the “stdio.h” header file.

❖ The “file name” parameter refers to any name of the file with an extension such as “Data.txt” or “program1.c” and so on.

❖ File open mode refers to the mode of opening the file. The file can be opened in ‘read mode’, ‘write mode’ or ‘append mode’. It can also be opened with a combination of any two modes out of these three modes.

Basically, for opening the file, the secondary storage device is searched and if the file to be opened exists then that file is loaded into memory and it returns a file pointer of type FILE to identify a file. In case, the file does not exists, a macro in the header file “stdio.h” called NULL is returned.

Reading/Writing a File

Once the file is opened (i.e., it is loaded into the main memory) the associated pointer points to the starting character of the file contents. This file can be read completely by using various functions provided by the ‘C’ standard library (E.g.: fgetc( )).

Example: ch = fgetc(fp);

fgetc( ) reads the character pointed by fp and the pointer position gets incremented, ‘ch’ holds the character read from the file.

For writing data into an opened file we can use the fputc( ) function.

Example: fputc(ch, fp);

‘ch’ is the character to be written into the file.

Closing a File

After the required operations such as reading, writing or appending operations on a file are done, the file needs to be closed.

Syntax

fclose(file_pointer);

Example: fclose(fp);

The fclose( ) function has one parameter i.e., file_pointer for a particular file to be closed. If more than one files are to be closed then more than one fclose( ) function can be written with the respective file-pointers. Also, if more than one files are to be closed then a function called "fcloseall( )" can also be used for closing all the files that are opened. This function does not require any parameter.

Different Modes of Opening File

1. **"r" (Read) Mode**

   This mode opens the file that already exists for reading only. If the file to be read does not exist then an error occurs.

   Syntax

   fp = fopen("READ.txt", "r");

2. **"w" (Write) Mode**

   A text file for writing is created when the "w" mode is given as the second parameter in the fopen( ) function. The previous contents of the existing file are discarded without any confirmation.

   Syntax

   fp = fopen("WRITE.txt", "w");

3. **"a" (Append) Mode**

   To append or add data to an existing file, the "a" mode is declared in the fopen( ) function. The file pointer points to the end of file, of the existing file so that the data can be appended.

   If a file does not exist then a file with a specified name is opened for writing.

   Syntax

   fp = fopen("Append.txt", "a");

4. **"r+" (Read and Write) Mode**

   This mode is declared for both reading and writing the data in the file. If a file does not exist then NULL is returned to the file pointer.

   Syntax

   fp = fopen("EVEN.txt", "r+");

5. **"w+" (Write and Read) Mode**

   In this mode, writing and reading operations can be done on the file. If a file already exists then the contents of that file are eliminated. If the file does not exist then a new file is created.

   Syntax

   fp = fopen("GOOD.txt", "w+");

6. **"a+" (Append and Read) Mode**

   The file can be read as well as data can be added by using "a+" mode.

   Syntax

   fp = fopen("Data.txt", "a+");

7. **"wb" (Write Binary) Mode**

   When this mode is given in the fopen( ) function a binary file is opened in write mode.

   Syntax

   fp = fopen("INPUT.txt", "wb");

8. **"rb" (Read Binary) Mode**

   To read a binary file, "rb" mode is given.

   Syntax

   fp = fopen("Output.txt", "rb");

9. **"ab" (Append Binary) Mode**

   In order to add data at the end of a binary file "ab" mode is given.

   Syntax

   fp = fopen("ADD.txt", "ab");

Q21. List and explain different categories of standard I/O functions.

Answer :

Standard Library Input/Output Functions

The standard library I/O functions are categorized as follows,

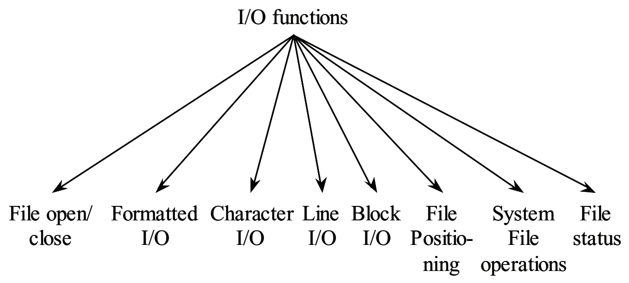


Figure: Standard I/O functions

All these functions are present in "stdio.h" header file.

1. **File Open/Close Functions**

For answer refer Unit-V, Page No. 5.12, Q.No. 20, Topics: Opening a File, Closing a File.

2. **Formatted I/O Functions**

The following are the various formatting input/output functions.

(a) **scanf( )**

   This function is used to read values for variables from keyboard.

   Syntax

   scanf("control string", address_list);

(b) printf( )

printf( ) function is used to print result on monitor.

Syntax

printf("control string", arg1, arg2,...., argn);

(c) fscanf( )

For answer refer Unit-V, Page No. 5.11, Q.No. 17, Topic: fscanf( ).

(d) fprintf( )

For answer refer Unit-V, Page No. 5.11, Q.No. 17, Topic: fprintf( ).

3. Block Input/Output Functions

Block I/O functions are used to read data from binary files and write data in to binary files. The functions fread( ) and fwrite( ) are the block I/O functions.

(a) fread( )

This block I/O function reads the data from the binary files as per the number of bytes specified. The bytes that are read are then placed into memory at the location specified.

Syntax

int fread(void* S, int size, int counter, FILE* fp);

Where, *S* is the pointer to the input area in the memory and *S* uses a void pointer which indicates that any pointer type can be passed to the function. The amount of data to be transferred is given by the product of parameters 'size' and 'counter'. The associated stream from where data has to be read is given by FILE* fp.

Example

fread(S, sizeof (int), 4, fp);

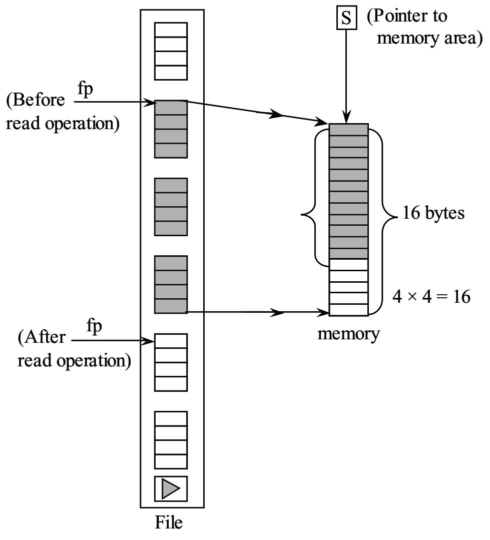

S (Pointer to memory area)
(Before read operation) fp
16 bytes
4 × 4 = 16
memory
(After read operation) fp
File

Figure: File Read

In the example above, the contents of a file are read and stored into a memory area, which is an array of integers. Size of integer is specified as 4 bytes. Therefore, for storing 4 integers 16 bytes of storage space is required.

(b) fwrite( )

The fwrite( ) block I/O function is used to write to a binary file the number of items that are specified.

Syntax

int fwrite(void* S1, int size, int counter, FILE* fp1); Where, S1 is the pointer to the input area in the memory and it uses a void pointer which indicates that any type of pointer can be passed to the function. The amount of data to be transferred is given by "size*counter". The associated stream is obtained from FILE* fp1.

Example

fwrite (S1, sizeof(int), 4, fp1);

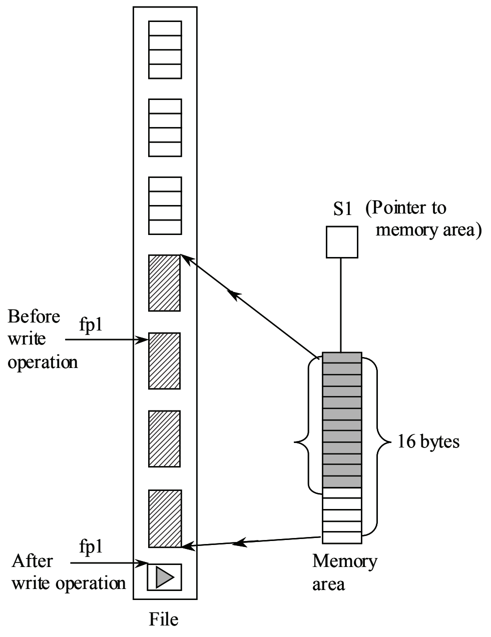


Figure: File Write

The fwrite( ) function in this example copies 3*4 (size * counter) = 12 bytes from the address pointed by S1 to the file pointed by fp1.

4. Character I/O Functions

Character I/O functions are used to read or write a single character at a time. The two categories under these functions are,

(a) I/O functions used with terminal device.

(b) I/O functions used with terminal devices as well as text files.

The following figure describes the general classification of character I/O functions.

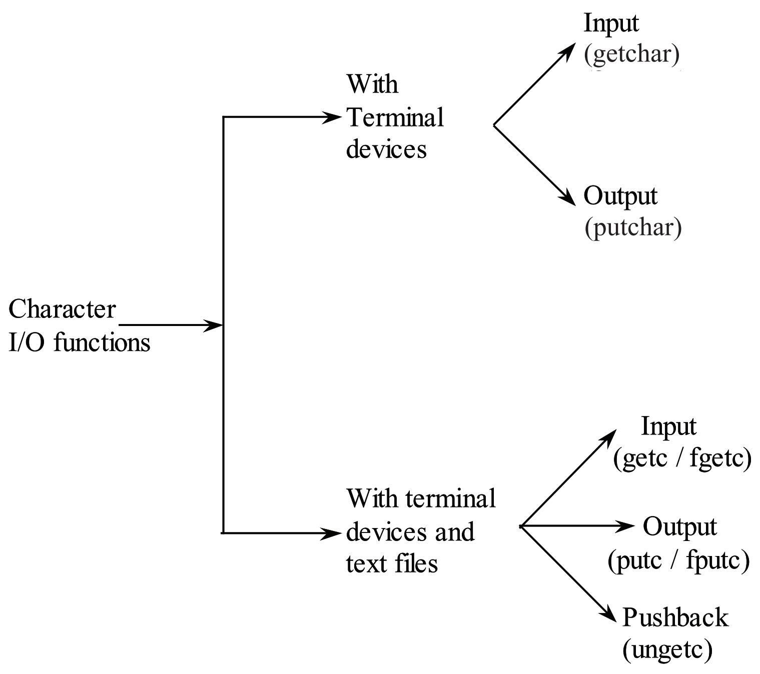


Figure: Classification of Character I/O Functions

(a) I/O Functions Used With Terminal Devices

These functions are used with standard streams like standard input (stdin) and standard output (stdout) only. getchar( ) and putchar( ) are the two character I/O functions used with terminal devices only.

(i) getchar( )

getchar( ) is a console I/O function that is used for reading a character from standard input device (keyboard). It reads one character at a time. This function does not perform any operation until a key is pressed after which it returns the value.

Syntax

int getchar(void);

The getchar( ) reads an integer value. It is also possible to use char data type because it is low-order byte. The getchar( ) macro returns the next byte from standard input stream or EOF if any error occurs.

The **getc( )** is similar to the getchar( ) function. However, it is a file I/O function used to read a character from the file.

Syntax

getc(file_pointer);

That is, getchar( ) is equivalent to getc(stdin). The getc( ) and getchar( ) macros are implemented as subroutines for ANSI compatibility.

(ii) putchar( )

putchar( ) I/O function is used for writing a character to standard output device (monitor). Write operation is started from the current cursor position.

Syntax

int putchar(int c);

putchar( ) takes integer as a parameter, but most commonly 'char' can also be taken as its argument. The output of putchar( ) are the characters that are written or EOF in case of an error. putchar( ) prints one character at a time.

putc( ) is similar to putchar( ) and performs the same function of putchar( ).

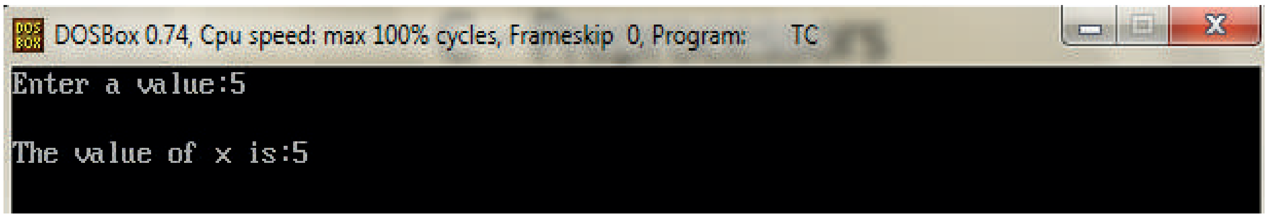
Syntax

putc(char c, file_pointer);

Program

```
/*Program to Demonstrate getchar( ) and putchar( )*/
#include<stdio.h>
#include<conio.h>
int main( )
{
int x;
clrscr();
printf("Enter a value:");
x = getchar();
printf("\nThe value of x is:");
putchar(x);
getch();
return 0;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter a value:5

The value of x is:5
```

(b) I/O Functions Used With terminal Devices as well as Text Files

These functions are not confined to a particular stream, but can be used with any stream. They need an argument that specifies the stream of terminal device or a file.

When these functions are used with a terminal device, there is no need to declare the stream explicitly. The system implicitly declares and opens the stream. The standard input stream stdin is declared for the keyboard where as the standard output stream stdout is declared for the monitor.

When I/O functions are used with files, system does not declare the stream. It is the responsibility of user to explicitly declare the stream, open it and associated it with the file.

The following are the functions used with both terminal devices and files.

(i) getc( ) and fgetc( ) to read a character.

(ii) putc( ) and fputc( ) to write a character.

(iii) ungetc( ) to push a character back.

(i) getc( ) and fgetc( )

The getc( ) and fgetc( ) functions are used to read the next character from a user defined stream or stdin and converts the character to an integer.

If the entire file has been read and pointer points to end of the file, then the functions return EOF. Even if any error is encountered, they return EOF. Both the functions getc( ) and fgetc( ) are essentially same however, fgetc( ) is most widely used.

Prototype

int getc(FILE* a)

x = getc(a file);

int fgetc(FILE* b)

y = fgetc(b file);

(ii) putc( ) and fputc( )

The functions putc( ) and fputc( ) are used to write a character to a user defined file stream, i.e., stdout or stderr. These functions require two parameters, first is the character to be written and the second is the file pointer that points to a file where the character has to be written.

If the character is successfully written to a file, then the same character is returned by the fputc. In case of an error, fputc( ) returns EOF.

Prototype

int putc(int a, FILE* fname);

int fputc(int a, FILE* fname);

Example

ret = fputc('a', abc);

Here, character 'a' is to be written to the file pointed by file.pointer 'abc', 'ret' is an integer variable that takes the return value.

Note that 'C' automatically type casts the character value 'a' to integer value.

(iii) ungetc( )

ungetc( ) is a push back function used to write one or more characters to an input stream. When more than one character is pushed back, the reading operation is done in reverse order, i.e., the first character that is read is the last character that was pushed.

If push back operation is successful, the function returns the pushed character, else returns EOF.

The two parameters of ungetc( ) function are a character and an input stream.

Prototype

int ungetc(int a, FILE* fp)

Example

```
op = fgetc(stdin);

if(isdigit(op)) /* checks whether character is a digit */

ungetc(op, stdin);  /* If so, put it back on the stream */

else

{

        ≡

        /* The character read is not a number */

            some function( );

}
```

The above example reads standard input stream and determines whether the read data is a number or character. If it is a number, we put the data back on the stream using the ungetc( ) function. If it is not a number (i.e., the else part), we perform the function some function( ).

5. Line I/O Functions

In C, there are two ways to read and write strings. One is using formatted I/O functions and the other is using string-only functions.

(a) Formatted String Input/Output

scanf( ) and fscanf( ) are the two formatted input functions that can be used to read the string as input.

Formatted String Input

The conversion code for a string is s(%s). While reading the input, the conversion code ignores the leading white space and the remaining characters are copied one after the other into the array in an order by the scanf( ) function. Once it finds the trailing white space, it terminates the string with '\D' i.e., a null character.

Flag, maximum field size and size are the three options that exist for conversion specification of strings.

The flag asterisk (*) is used to discard the data read. Maximum field size is responsible for holding maximum number of characters that can be read. Size option is useful if wide characters are to be read. If this option is not used, then by default normal characters are read.

Example

scanf("%s", &name);

This statement reads the input from the keyboard and stores the read value into the variable "name".

scanf("%5s", name);

This statement limits the user to enter only five characters.

scanf("%5[0 1 2 3 4:, –$]",s);

This statement not only limits the user to enter five characters but also restricts him/her to use only 0 to 4 digits, fullstop, a comma, a hyphen and a dollar sign.

scanf("%5s [^\n])", l);

This statement specifies what is not accepted while reading. It can read all the characters except the new line character.

scanf("%15[ ] [0 1 2 3 4 5 6 7 8 9]", s);

This statement reads the string which contains only the brackets and digits.

Formatted String Output

printf( ) and fprintf( ) are the two formatted output functions that are used to produce output. String conversion codes of these functions are similar to those used in string input functions.

Left-justify flag, width, precision and size are the four options that can be used for formatted string output functions.

(i) Left-Justify Flag(–)

This flag is responsible for left-justifying the output. This option is applicable only when "width" option is used in printf( ).

The effect of justification can be seen only when the length of string is less than the format width. If the justification flag (–) is not used, then, by default the output is right justified.

Example

printf("| % – 20s |\n" "Hello world");

Output

```
|Hello World         |
```

This statement restricts the minimum size of the string in the output to be 20 characters and since the justification flag is used, the justification is left.

(ii) Minimum Width

This option is used to set the minimum size of the string in the output. If this field is not accomplished with a flag, the justification is by default right.

Example

printf("| % – 20s |\n", "Hello world");

Output

```
|        Hello World|
```

(iii) Precision

Precision is responsible for setting the maximum number of characters that can be written. This is required to avoid printing of long strings that exceed the specification of width.

Example

printf(" | % – 10.9s |", Hello world");

Output

```
|Hello World|
```

(iv) Size

It is similar to that of string input functions. If size is not specified, normal characters are written in the output. If size option is used wide characters are written in the output.

(b) String Input/Output Functions

These functions are used to read and write strings but the data is not reformatted. The functions that convert text-file lines to strings and strings to text-file lines are two variations of string I/O functions.

A string is terminated by a null character ('\0'), whereas a line is terminated by a newline character ('\n').

(i) Line to String Conversion

gets( ) and fgets( ) are the input function that perform line to string conversion. They take a line as input from the input stream and construct a string that is terminated by '\0'.

Syntax

char* gets(char* str);

char* fgets(char* str, int size, file * f_p);

The functions of gets( ) and fgets( ) are not similar. The gets( ) function is responsible for converting the newline character to '\0', whereas fgets( ) function adds '\0' to the line and converts it to a string.

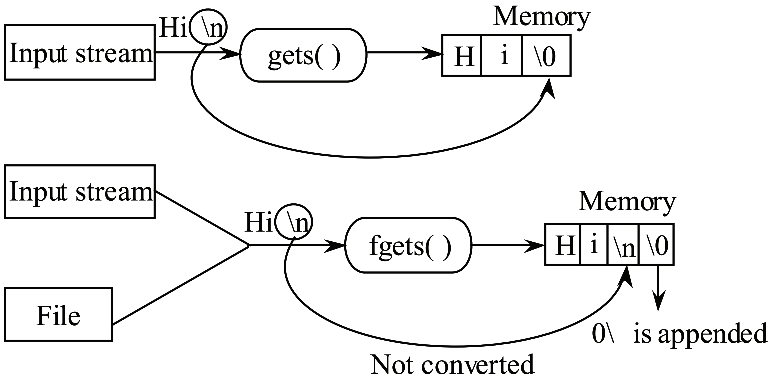


Figure: Illustration of gets( )/fgets( ) functions

Example

char s[100];

gets(s);

fgets(s, sizeof(s), stdin);

The gets( ) function reads a line from standard input where as fgets( ) function reads 100 bytes (size of s) from standard input.

(ii) String to Line Conversion

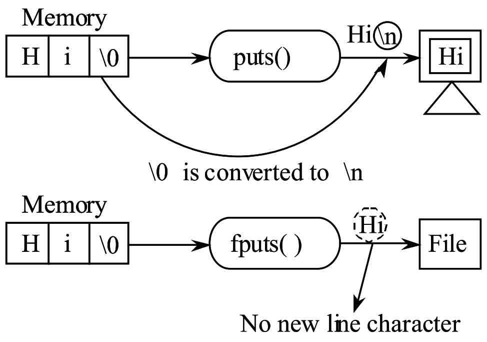
puts/fputs are the two string line output functions that take a string which is terminated by '\0' from memory and write the output as a line either to a file or to a monitor (display device).

Syntax

int puts(const char* str);

int fputs(const char* str, FILE * fp);

The string that is pointed by 'str' is written to the file pointed by 'str' is written to the file pointed by fp. If the writing operation is successful, it returns a non negative integer. On encountering an error, EOF is returned.



Example

char string[ ] = "Hello world";

puts(string);

fputs(string, f_1);

The puts( ) function displays the output as a string on the monitor, whereas fputs writes it (string) to the file pointed by f1.

6. File Positioning Functions

For answer refer Unit-V, Page No. 5.21, Q.No. 22.

7. System File Operations

System file operations are carried out by some functions like remove( ), rename( ), tmpfile( ). These functions make use of operating system calls to carry out the operations on the entire file. The operations include removing a file, renaming a file, creating a temporary binary file.

(a) remove( ) Function

This function is used to delete (or remove) a file using its external name. It has a single parameter, which is of pointer type that points to the name of the file.

Syntax

int remove(char* name);

Example

remove("abc.dat");

If the deletion process is successful, the function returns zero. If any errors exist like file not found, it returns a non-zero value.

(b) rename( ) Function

The rename( ) function is used to rename a file. Suppose a new version of some file is created and user would like the name of the file to be same as that of older version, then the older version file has to be removed.

Syntax

int rename(const char* old name const char* newname);

Example

rename("abc.dat", "def.dat");

Here, the parameters to the function are pointers to old and the new file names. If renaming is done successfully, the function returns zero. If any error is found, it returns a non-zero value.

(c) tmpfile( )

This function is used to create a new temporary output file. This file will not be available always. It can be used only while the program is under execution. Once the execution of the program is finished, the file will be closed and then erased completely.

Syntax

FILE* tmpfile(void);

Example

FILE* P;

P = tmpfile( );

Here 'P' is a file pointer. The second statement opens the temporary file that is created. This file is accessible in w+b mode.

8. File Status Functions

These functions are used to deal with the questions regarding the status of a file. The functions feof( ), ferror( ), clearerr( ) are the examples of file status functions.

(a) feof( )

feof( ) is a library function to determine if the file pointer is at the end of file or not i.e., it takes file pointer (fp) as its parameter to test end-of-file condition. It returns a non-zero value if EOF is reached, otherwise it returns zero.

Example

```
if(feof(fp))
        printf("End of data\n");
```

This statements prints the message at the end of data when end of file is reached.

(b) ferror( )

ferror( ) is a library function which finds the error when read/write operations are performed on a file. It takes file pointer (fp) as its argument that returns zero if read/write operation is performed successfully and non-zero value if read/write operation is terminated unsuccessfully.

Example

```
if(ferror(fp)!= 0)
        printf("There is an error in a file");
```

This statement displays the message if read/write operation is unsuccessful.

(c) clearerr( )

The function clearerr( ), is responsible for resetting the error status of a file. It is required because once an error has been detected, the ferror( ) function returns true (non-zero) for subsequent calls to it, until the status of the file have been reset. Therefore, we use clearerr( ) to reset the status of the file.

The syntax of clearerr( ) is given below. It takes file pointer as an argument.

Syntax

```
void clearerr(FILE* stream);
```

However, use of clearerr( ) once does not solve the problems completely. It just clears the error once. So, there is a possibility that next read or write operation may results in an error state.

Q22. Write about the following functions,

(a) rewind( )

(b) ftell( )

(c) fseek( ).

OR

Write short notes on random access file functions.

Answer :

(a) rewind( )

This function takes file pointer to the starting of the file and resets it.

Syntax

```
rewind(fp);
```

This statement rewinds the file pointer fp to the beginning of the file.

(b) ftell( )

This function gives the current position of file pointer in terms of bytes from the beginning.

Syntax

n = ftell(fp);

Here 'n' gives the current position of the file pointer from the beginning. It gives the information regarding number of bytes read or written already.

(c) fseek( )

fseek( ) is a file function that is used to move the file pointer to the desired location in the file. It is generally used when a particular part of the file is to be accessed. The main purpose of file fseek( ) is to position the file pointer to any location on the stream.

Syntax

fseek(fileptr, offset, position);

'fileptr' is a file pointer.

Offset specifies the number or variable of type long to reposition the file pointer. It tells the number of bytes to be moved. Offset can either be positive or negative number where positive integer is used to reposition the pointer forward and negative integer is used to move the pointer backward.

Position

It specifies the current position.

The values that can be assigned to position are,

Value	Position
0	Beginning of file
1	Current position
2	End Of File (EOF)

The fseek( ) returns zero, if all the operations are performed successfully or –1 if an error occurs such as EOF is reached.