SCHEME : K

Name : _____

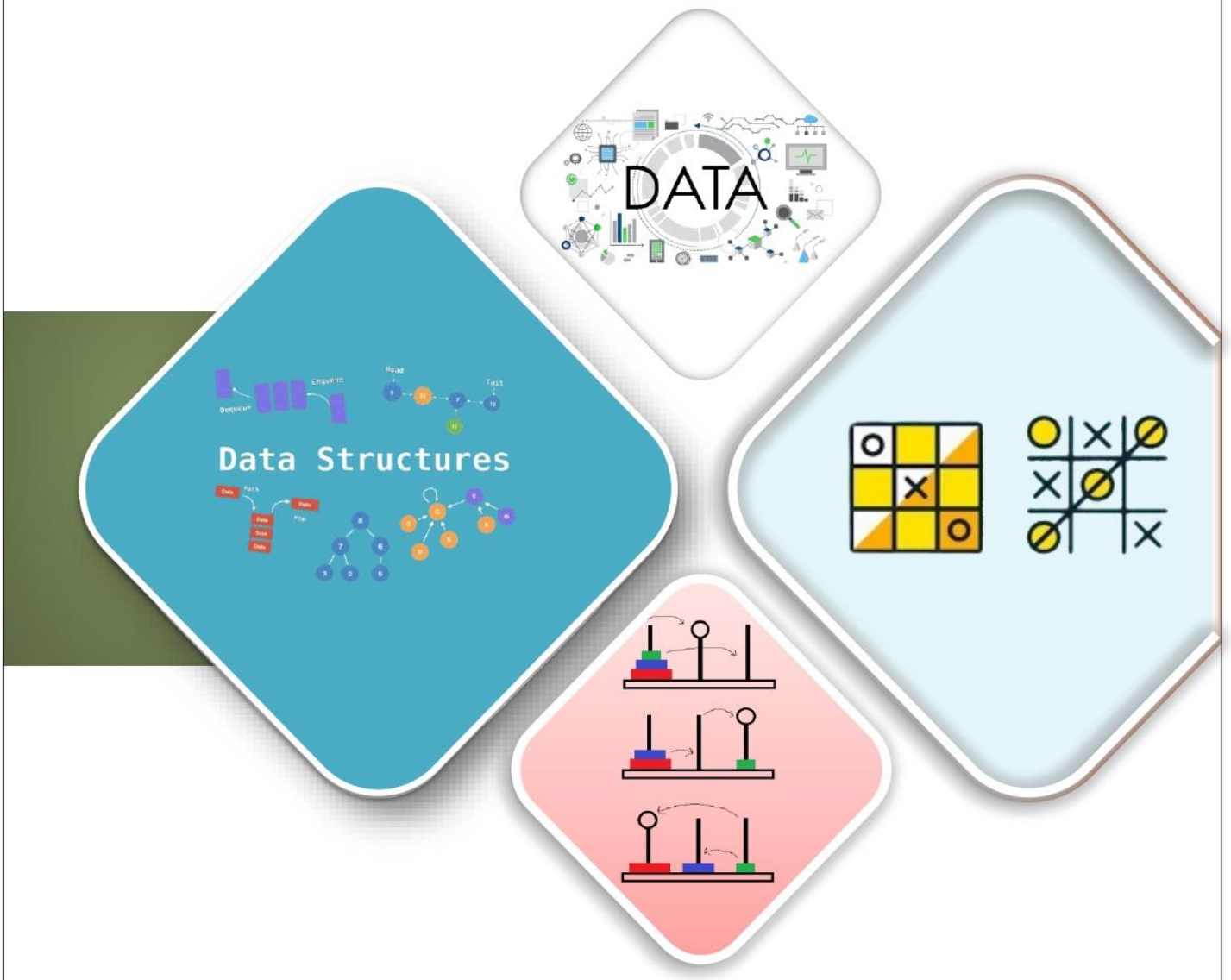
Roll No. : _____ Year : 20___ 20___

Exam Seat No. : _____

LABORATORY MANUAL FOR

DATA STRUCTURE USING C

(313301)



COMPUTER ENGINEERING GROUP

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION, MUMBAI
(Autonomous) (ISO 9001: 2015) (ISO/IEC 27001:2013)

VISION

To ensure that the Diploma level Technical Education constantly matches the latest requirements of technology industry and includes the all-round personal development of students including social concerns and to become globally competitive, technology led organization.

MISSION

To provide high quality technical and managerial manpower, information and consultancy services to the industry and community to enable the industry and community to face the changing technological & environmental challenges.

QUALITY POLICY

We, at MSBTE are committed to offer the best-in-class academic services to the students and institutes to enhance the delight of industry and society. This will be achieved through continual improvement in management practices adopted in the process of curriculum design, development, Implementation, evaluation and monitoring system along with adequate faculty development programmes.

CORE VALUES

MSBTE believes in the followings:

- Education industry produces live products,
- Market requirements do not wait for curriculum changes.
- Question paper is the reflector of academic standards of educational organization.
- Well-designed curriculum needs effective implementation too.
- Competency based curriculum is the backbone of need based program.
- Technical skills do need support of life skills,
- Best teachers are the national assets.
- Effective teaching learning process is impossible without learning resources.

A Laboratory Manual for

Data Structure Using C

(313301)

"K-SCHEME"

Semester-III

(BD/CM/CO/CW/HA/IF/IH)

Maharashtra State

Board of Technical Education, Mumbai

(Autonomous) (ISO 9001:2015)

(ISO/IEC 27001:2013)

Maharashtra State Board of Technical Education,
(Autonomous) (ISO 9001:2015) (ISO/IEC 27001:2013)
4th Floor, Government Polytechnic Building, 49, Kherwadi,
Bandra (East), Mumbai- 400051.
(Printed on July, 2024)

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION

Certificate

This is to certify that Mr. / Ms. …… Roll No……., of Third Semester of Diploma in……of Institute, ……. (Code: ……...) has completed the term work satisfactorily in Subject **Data Structure Using C (313301)** for the academic year 20…….. to 20…….. as prescribed in the curriculum.

Place: …… Enrollment No:……

Date: …… Exam. Seat No: ……

Subject Teacher **Head of the Department** **Principal**

Seal of Institution

Preface

The primary focus of any engineering laboratory/ field work in the technical education system is to develop the much-needed industry relevant competencies and skills. With this in view, MSBTE embarked on this innovative 'K' Scheme curricula for engineering diploma programmes with outcome-base education as the focus and accordingly, relatively large amount of time is allotted for the practical work. This displays the great importance of laboratory work making each teacher; instructor and student to realize that every minute of the laboratory time need to be effectively utilized to develop these outcomes, rather than doing other mundane activities. Therefore, for the successful implementation of this outcome-based curriculum, every practical has been designed to serve as a 'vehicle' to develop this industry identified competency in every student. The practical skills are difficult to develop through 'chalk and duster' activity in the classroom situation. Accordingly, the 'I' scheme laboratory manual development team designed the practical's to focus on the outcomes, rather than the traditional age old practice of conducting practical's to 'verify the theory' (which may become a byproduct along the way).

This laboratory manual is designed to help all stakeholders, especially the students, teachers and instructors to develop in the student the pre-determined outcomes. It is expected from each student that at least a day in advance, they have to thoroughly read through the concerned practical procedure that they will do the next day and understand the minimum theoretical background associated with the practical. Every practical in this manual begins by identifying the competency, industry relevant skills, course outcomes and practical outcomes which serve as a key focal point for doing the practical. The students will then become aware about the skills they will achieve through procedure shown there and necessary precautions to be taken, which will help them to apply in solving real-world problems in their professional life.

This manual also provides guidelines to teachers and instructors to effectively facilitate student-centered lab activities through each practical exercise by arranging and managing necessary resources in order that the students follow the procedures and precautions systematically ensuring the achievement of outcomes in the students.

This course provides essential knowledge to any graduate or undergraduate belonging to any discipline involving computer science and software engineering. Data Structure, specifically demonstrates logical and mathematical model of storing and organizing data in a particular way in a computer, required for designing and implementing efficient algorithms. Through this course student will able to understand different kinds of data structures like arrays, linked lists, stacks, queues etc. are suited to different kinds of applications.

Although all care has been taken to check for mistakes in this laboratory manual, yet it is impossible to claim perfection especially as this is the first edition. Any such errors and suggestions for improvement can be brought to our notice and are highly welcome.

Program Outcomes

Basic and Discipline specific knowledge: Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the engineering problems.

Problem analysis: Identify and analyze well-defined engineering problems using codified standard methods.

Design/ development of solutions: Design solutions for well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

Engineering Tools, Experimentation and Testing: Apply modern engineering tools and appropriate technique to conduct standard tests and measurements.

Engineering practices for society, sustainability and environment: Apply appropriate technology in context of society, sustainability, environment and ethical practices.

Project Management: Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.

Life-long learning: Ability to analyse individual needs and engage in updating in the context of technological changes.

Practical- Course Outcome matrix

Sr. No.	Title of the Practical	CO 1	CO 2	CO 3	CO 4	CO 5	CO 6
1	*Write a 'C' program to perform following Operations on Array: Create, Insert, Delete, Display.	√					
2	Write a 'C' Program to Search a particular data from the given Array of numbers using: Linear Search Method.		√				
3	* Write a 'C' Program to Search a particular data from the given Array of Strings using Linear Search Method.		√				
4	* Write a 'C' program to Search a particular data from the given Array of numbers using Binary Search Method.		√				
5	Write a 'C' Program to Search a particular data from the given Array of Strings using Binary Search Method.		√				
6	* Write a 'C' Program to Sort an Array of numbers using Bubble Sort Method.		√				
7	Write a 'C' Program to Sort an Array of Strings using Bubble Sort Method.		√				
8	* Write a 'C' Program to Sort an Array of numbers using Selection Sort Method.		√				
9	Write a 'C' Program to Sort an Array of Strings using Selection Sort Method.		√				
10	* Write a 'C' Program to Sort an Array of numbers using Insertion Sort Method.		√				
11	Write a 'C' Program to Sort an Array of Strings using Insertion Sort Method.		√				
12	* Write a 'C' Program to Implement Singly Linked List with Operations: (i) Insert at beginning, (ii) Search, (iii) Display			√			
13	* Write a C Program to Implement Singly Linked List with Operations: (i) Insert at end, (ii) Insert After, (iii) Delete (iv) Display			√			
14	Write a C Program to Create Two Polynomials using a Linked List.			√			
15	* Write a 'C' Program to add Two Polynomials using a Linked List.			√			
16	* Write a 'C' Program to perform PUSH and POP Operations on Stack using an Array.				√		
17	* Write a 'C' Program to perform PUSH and POP Operations on a Stack using a Linked List.				√		

Sr. No.	Title of the Practical	CO 1	CO 2	CO 3	CO 4	CO 5	CO 6
18	* Write a 'C' program to perform multiplication of two numbers using recursion.				√		
19	Write a 'C' program to print given string in reverse using recursion.				√		
20	Write a 'C' program to create a Singly Linked List and traverse in reverse order using recursion.			√	√		
21	* Write a 'C' Program to perform INSERT and DELETE Operations on Linear Queue using an Array.					√	
22	* Write a 'C' Program to perform INSERT and DELETE operations on Linear Queue using a Linked List.					√	
23	* Write a 'C' Program to perform INSERT and DELETE operations on Circular Queue using an Array.					√	
24	Write a 'C' Program to perform INSERT and DELETE operations on Circular Queue using a Linked List.					√	
25	Write a 'C' Program to Create a Priority Queue using a Linked List.					√	
26	* Write a 'C' Program to Implement BST (Binary Search Tree) and Traverse in In-Order.						√
27	Write a 'C' Program to Traverse BST in Preorder, and Post-Order.						√

'*' Marked Practical (LLOs) Are mandatory.

Industry / Employer Expected Outcome

The following industry relevant skills of the competency **'Implement algorithm using relevant Data Structures'** are expected to be developed in you by undertaking the practical's of this laboratory manual.

1. Design algorithms to perform different operations on different types of Data Structures
2. Perform basic operations of given Data Structures
3. Implement algorithms to solve real time applications.

Guidelines to Teachers

1. For incidental writing on the day of each practical session every student should maintain a dated logbook for the whole semester, apart from this laboratory manual, which s/he has to submit for assessment to the teacher in the next practical session.
2. Teachers should give opportunity to students for hands-on after the demonstration.
3. Assess the skill achievement of the students and COs of each unit.
4. Explain prior concepts to the students before starting of each experiment.
5. List of few sample questions for reference. Teachers must design more such questions to ensure the achievement of identified CO.
6. Teacher should ensure that the practical skill and competencies are developed in the students after the completion of the practical exercise.
7. Teacher may provide additional knowledge and skills to the students even though it's not covered in the manual but are expected from the students by the industries.
8. Teacher may suggest the students to refer additional related literature of the Technical papers/ Reference books/ Seminar proceedings, etc.
9. Teacher shall assess the performance of students continuously as per norms prescribed by MSBTE.
10. During assessment teacher is expected to ask questions to the students to tap their Achievements grading related knowledge and skills so that student can prepare while submitting record of the practical focus should be given on development of enlisted skills rather than theoretical knowledge.

Instructions for Students

1. Understand the purpose of practical and its implementation.
2. Student shall develop practical skills as expected by the Industries.
3. Listen carefully to the instructions given by the teacher about importance of relevant program outcomes, relevant course outcomes, practical significance, competency and practical skills, practical outcome and the theoretical background during the practical session.
4. Write the answers of the questions allotted by the teacher during practical session.
5. Student should develop the habit of group discussion related to the practical, so that exchange of knowledge/skills could take place.
6. Student shall attempt to develop related hands-on-skills to gain confidence.
7. Student shall refer technical magazines, websites related to the scope of the course.
8. Student should develop habit to submit the practical, exercise continuously and progressively on the scheduled dates and should get the assessment done.
9. Student should be well prepared while submitting the write up of the exercise.
10. Student should not hesitate to ask any difficulty faced during conduct of practical.

Content Page

List of Practical and Progressive Assessment Sheet

Sr. No.	Title of the Practical	Page No	Date of Performance	Date of submission	Assessment marks (25)	Dated sign. of teacher	Remarks (if any)
1	*Write a 'C' program to perform following Operations on Array: Create, Insert, Delete, Display.	1					
2	Write a 'C' Program to Search a particular data from the given Array of numbers using: Linear Search Method.	8					
3	* Write a 'C' Program to Search a particular data from the given Array of Strings using Linear Search Method.	15					
4	* Write a 'C' program to Search a particular data from the given Array of numbers using Binary Search Method.	21					
5	Write a 'C' Program to Search a particular data from the given Array of Strings using Binary Search Method.	28					
6	* Write a 'C' Program to Sort an Array of numbers using Bubble Sort Method.	34					
7	Write a 'C' Program to Sort an Array of Strings using Bubble Sort Method.	43					
8	* Write a 'C' Program to Sort an Array of numbers using Selection Sort Method.	50					
9	Write a 'C' Program to Sort an Array of Strings using Selection Sort Method.	57					
10	* Write a 'C' Program to Sort an Array of numbers using Insertion Sort Method.	64					
11	Write a 'C' Program to Sort an Array of Strings using Insertion Sort Method.	71					

Sr. No.	Title of the Practical	Page No	Date of Performance	Date of submission	Assessment marks (25)	Dated sign. of teacher	Remarks (if any)
12	* Write a 'C' Program to Implement Singly Linked List with Operations: (i) Insert at beginning, (ii) Search, (iii) Display	78					
13	* Write a C Program to Implement Singly Linked List with Operations: (i) Insert at end, (ii) Insert After, (iii) Delete (iv) Display	85					
14	Write a C Program to Create Two Polynomials using a Linked List.	94					
15	* Write a 'C' Program to add Two Polynomials using a Linked List.	100					
16	* Write a 'C' Program to perform PUSH and POP Operations on Stack using an Array.	109					
17	* Write a 'C' Program to perform PUSH and POP Operations on a Stack using a Linked List.	115					
18	* Write a 'C' program to perform multiplication of two numbers using recursion.	121					
19	Write a 'C' program to print given string in reverse using recursion.	127					
20	Write a 'C' program to create a Singly Linked List and traverse in reverse order using recursion.	133					
21	* Write a 'C' Program to perform INSERT and DELETE Operations on Linear Queue using an Array.	139					
22	* Write a 'C' Program to perform INSERT and DELETE operations on Linear Queue using a Linked List.	146					
23	* Write a 'C' Program to perform INSERT and	152					

Sr. No.	Title of the Practical	Page No	Date of Performance	Date of submission	Assessment marks (25)	Dated sign. of teacher	Remarks (if any)
	DELETE operations on Circular Queue using an Array.						
24	Write a 'C' Program to perform INSERT and DELETE operations on Circular Queue using a Linked List.	160					
25	Write a 'C' Program to Create a Priority Queue using a Linked List.	166					
26	* Write a 'C' Program to Implement BST (Binary Search Tree) and Traverse in In-Order.	173					
27	Write a 'C' Program to Traverse BST in Preorder, and Post-Order.	180					

Practical No.1: * Write a 'C' program to perform following Operations on Array: Create, Insert, Delete, Display.

I Practical Significance

Writing a 'C' program to perform operations on an array helps in understanding fundamental data structures, memory management, and algorithm development. It provides a foundation for more complex programming tasks and real-world applications, reinforcing essential problem-solving skills in computer science.

II Industry/ Employer Expected Outcome

Writing a 'C' program to perform following Operations on Array: Create, Insert, Delete, Display will help Industry to assess the candidate's understanding of fundamental programming concepts, data structures, and problem-solving skills.By performing this experiment the student should

1. Properly initialize an array with a specified size.
2. Add elements at specified positions within the array while handling possible edge cases (e.g., inserting at the beginning, end, or beyond the current array size).
3. Remove elements from specified positions, ensuring the array's integrity and shifting elements accordingly.
4. Correctly print all the elements of the array in a readable format.

III Course Level Learning Outcomes

Perform basic operations on Arrays.

IV Laboratory Learning Outcome

Implement Array Operations.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

In C programming, arrays are a fundamental data structure used to store a collection of elements of the same data type. Operations on arrays typically involve creating an array, inserting elements into the array, deleting elements from the array, and displaying the contents of the array. Here's a brief theoretical background for each of these operations:

1. **Create (or Declare) Array**:

   To create an array in C, you declare a variable of array type, specifying the data type of its elements and the number of elements it can hold. For example:

   int myArray[10]; // Creates an array of 10 integers

   In C, arrays are zero-indexed, meaning the first element is at index 0 and the last element is at index (size - 1).

2. **Insert into Array**:

   Inserting elements into an array involves assigning values to specific elements within the array.

   To insert an element at a specific index in the array, you can directly assign a value to that index. However, this will overwrite any existing value at that index.

   If you want to insert an element and shift the existing elements to accommodate the new one, you would typically use a loop to move elements to make space for the new element.

3. **Delete from Array**:

   Deleting elements from an array involves removing elements from specific positions within the array.

   To delete an element from an array, you typically shift the elements after the deleted element one position to the left to fill the gap created by the deletion.

   You may also need to adjust the size of the array to reflect the removal of an element, which is not straightforward in C since arrays have fixed sizes once declared.

4. **Display Array**:

   Displaying the contents of an array involves iterating over the elements of the array and printing or displaying each element.

   This can be done using a loop, such as a for loop, to iterate over the array and print each element.

For example:

```
for (int i = 0; i < size; i++) {

printf("%d ", myArray[i]);

}
```

It's important to note that in C, arrays are fixed-size data structures, meaning their size cannot be changed once they are declared. To implement dynamic resizing or other more complex operations, you would typically need to use dynamic memory allocation techniques, such as malloc() and free(), along with pointer arithmetic. Additionally, error handling and boundary checks are crucial to ensure the correctness and robustness of array operations in C.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to find minimum and maximum element in an array
2. Write a C program to search a specific element in an array.

XV Exercise

1. List limitations of array in C.

2. Explain how sizeOf operator works with arrays.

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.javatpoint.com/data-structure-introduction
2. https://www.geeksforgeeks.org/introduction-to-data-structures/

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.2: * Write a 'C' Program to Search a particular data from the given Array of numbers using: Linear Search Method.

I Practical Significance

In C, a program using linear search can efficiently find a specific data within an array of numbers by sequentially checking each element until the target is found or the end of the array is reached. This method is straightforward and suitable for small or unsorted arrays.

II Industry/ Employer Expected Outcome

Demonstrate proficiency in writing C program.This includes understanding how to implement the algorithm correctly, handling edge cases, and writing clean, well-commented code. Additionally, students should be able to analyze the time complexity of their solution and possibly optimize it for better performance. Overall, industry seeks candidates who can effectively apply fundamental programming concepts to solve real-world problems. By performing this experiment student should-

1. Locate a specific data within an array of numbers, demonstrating the candidate's ability to work with arrays, loops, and conditional statements
2. Write clean and well commented code
3. Use optimization technique for performance improvement

III Course Level Learning Outcomes

Perform basic operations on Arrays

IV Laboratory Learning Outcome

Implement Linear Search Method on Numbers

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Linear search is a simple searching algorithm that sequentially checks each element of the array until the desired element is found or until all elements have been checked. It is also known as a sequential search. The basic idea is to start from the beginning of the array and compare each element with the target value until a match is found.

Here's how the linear search algorithm works:

1. Start from the first element of the array.
2. Compare the target value with the current element of the array.
3. If the target value matches the current element, return the index of the element.
4. If the target value does not match the current element, move to the next element in the array.
5. Repeat steps 2-4 until the target value is found or until the end of the array is reached. If the target value is not found after checking all elements, return a special

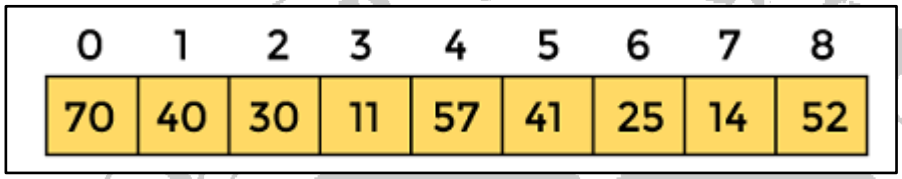
value (such as -1) to indicate that the element is not present in the array.

Linear search has a time complexity of O(n), where n is the number of elements in the array. It is suitable for small arrays or when the elements are not sorted. However, for large arrays or when efficiency is critical, other search algorithms like binary search (which requires a sorted array) may be more appropriate.

Working of Linear search

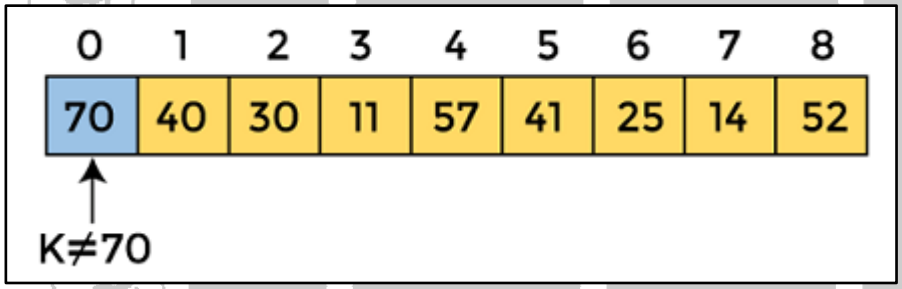
To understand the working of linear search algorithm, let's take an unsorted array. It will be easy to understand the working of linear search with an example.

Let the elements of array are -

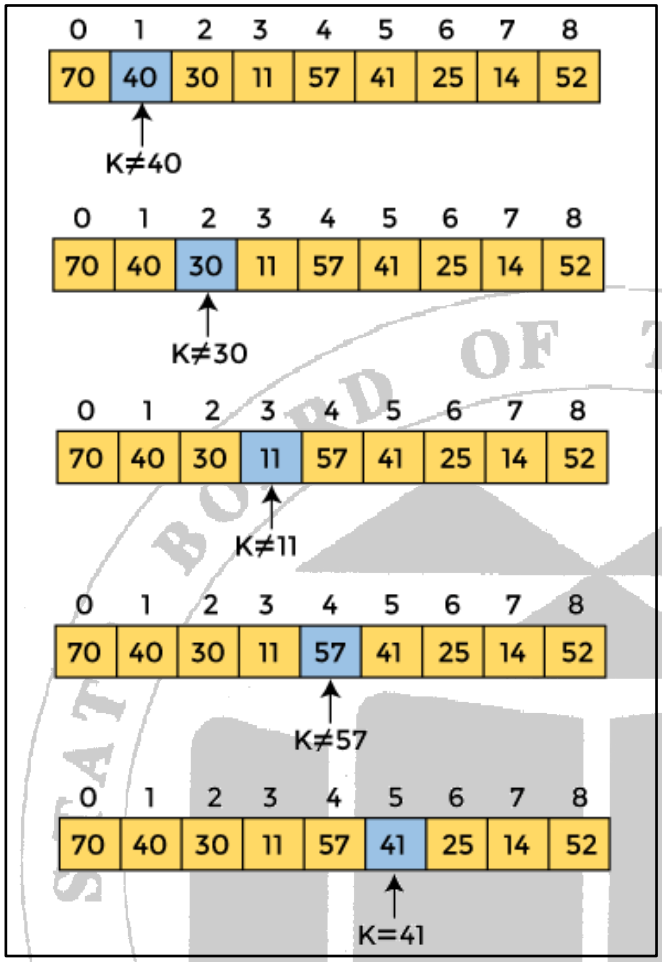


Let the element to be searched is **K = 41**

Now, start from the first element and compare **K** with each element of the array.



The value of **K,** i.e., **41,** is not matched with the first element of the array. So, move to the next element. And follow the same process until the respective element is found.



Now, the element to be searched is found. So algorithm will return the index of the element matched.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Modify the linear search program to find and print all occurrences of the target value in the array.
2. Explore real-world applications where linear search is used, such as searching for elements in a list of contacts or searching for files in a directory.

XV Exercise

1. Calculate time and space complexity of Linear Search.
2. Explain how Linear search can be modified for multiple occurrences.

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.javatpoint.com/data-structure-introduction
2. https://www.geeksforgeeks.org/introduction-to-data-structures/

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.3: Write a 'C' Program to Search a particular data from the given Array of Strings using Linear Search Method.

I Practical Significance

In C, a program using linear search can efficiently find a specific data within an array of Strings by sequentially checking each element until the target is found or the end of the array is reached. This method is straightforward and suitable for small or unsorted arrays.

II Industry/ Employer Expected Outcome

Demonstrate proficiency in writing C program. This includes understanding how to implement the algorithm correctly, handling edge cases, and writing clean, well-commented code. Additionally, students should be able to analyze the time complexity of their solution and possibly optimize it for better performance. Overall, industry seeks candidates who can effectively apply fundamental programming concepts to solve real-world problems. By performing this experiment student should-

1. Locate a specific data within an array of numbers, demonstrating the candidate's ability
2. to work with arrays, loops, and conditional statements
3. Write clean and well commented code
4. Use optimization technique for performance improvement

III Course Level Learning Outcomes

Perform basic operations on Arrays

IV Laboratory Learning Outcome

Implement Linear Search Method on Strings

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

The linear search is one of the most elementary and basic search methods. The specifics of the linear search will be covered in this blog post, along with its implementation in the C programming language, syntax examples, and samples of the expected result.

The fundamental search technique known as linear search that scans every element in a collection until the required element is discovered or the collection has been thoroughly searched. It is often referred to as sequential search. When dealing with short or unordered lists, this algorithm is especially helpful.

Here's how the linear search algorithm works:

- Start from the first element of the array.
- Compare the target value with the current element of the array.
- If the target value matches the current element, return the index of the element.
- If the target value does not match the current element, move to the next element in the array.
- Repeat steps 2-4 until the target value is found or until the end of the array is reached. If the target value is not found after checking all elements, return a special value (such as -1) to indicate that the element is not present in the array.

Linear search has a time complexity of O(n), where n is the number of elements in the array. It is suitable for small arrays or when the elements are not sorted. However, for large arrays or when efficiency is critical, other search algorithms like binary search (which requires a sorted array) may be more appropriate.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a simple C program to perform linear search on an array of Strings to find Case-Insensitive search for a string
2. Write a simple C program to perform linear search on an array of Strings to find Checking for substring in each string

XV Exercise

1. Write applications of Linear Search Algorithm.
2. Modify the linear search function to perform a case-insensitive search.

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.javatpoint.com/data-structure-introduction
2. https://www.geeksforgeeks.org/introduction-to-data-structures/

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved (LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No-4: Write a 'C' program to Search a particular data from the given Array of numbers using Binary Search Method.

I Practical Significance

Binary search is a highly efficient algorithm for finding a specific value in a sorted array. It significantly reduces the search time complexity to O(log n) by repeatedly dividing the search interval in half, making it particularly useful for large datasets.

II Industry/Employer Expected Outcome

The binary search algorithm's efficiency is crucial in various industries, particularly in database management, information retrieval, and software development.

1. Enable rapid data lookup and retrieval
2. Enhance performance in applications like search engines, data analytics platforms, and inventory management systems
3. Lead to faster query responses and improved user experiences.

III Course Level Learning Outcomes

Perform operations on Stack using Array and Linked List Implementations.

IV Laboratory Learning Outcome

Implement Binary search method on numbers.

V Relevant Affective Domain Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Binary Search is an interval searching algorithm used to search for an item in the sorted list. It works by repeatedly dividing the list into two equal parts and then searching for the item that is the part where it can possibly exist.

Unlike linear search, there are a few conditions for applying binary search:

The list must be sorted. Random access to the list member.

Explanation:

Initialize Pointers: left is the starting index, and right is the ending index of the array.

Loop Until the Pointers Meet: Continue the loop while left is less than or equal to right.

Calculate Middle Index: Find the middle index mid of the current search range using (left + right) / 2. To avoid potential overflow, it's better to use left + (right - left) / 2.

Compare Middle Element with Target:

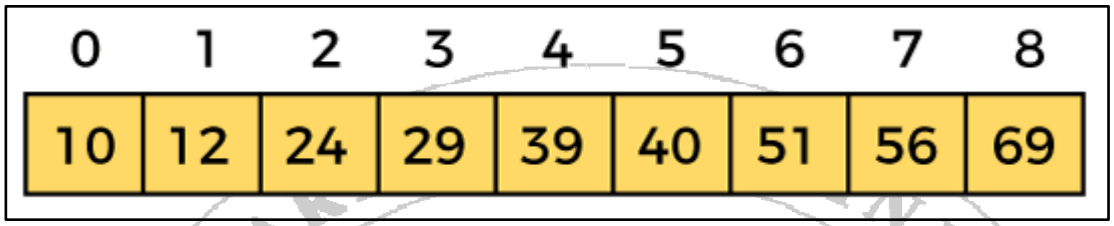
If arr[mid] equals the target, return mid (index of the target).

If arr[mid] is less than the target, narrow the search to the right half (left = mid + 1).

If arr[mid] is greater than the target, narrow the search to the left half (right = mid - 1).

Return Result: If the loop ends without finding the target, return -1 indicating the target is not in the array.

Let the elements of array are –



Let the element to search is, **K = 56**

We have to use the below formula to calculate the **mid** of the array –

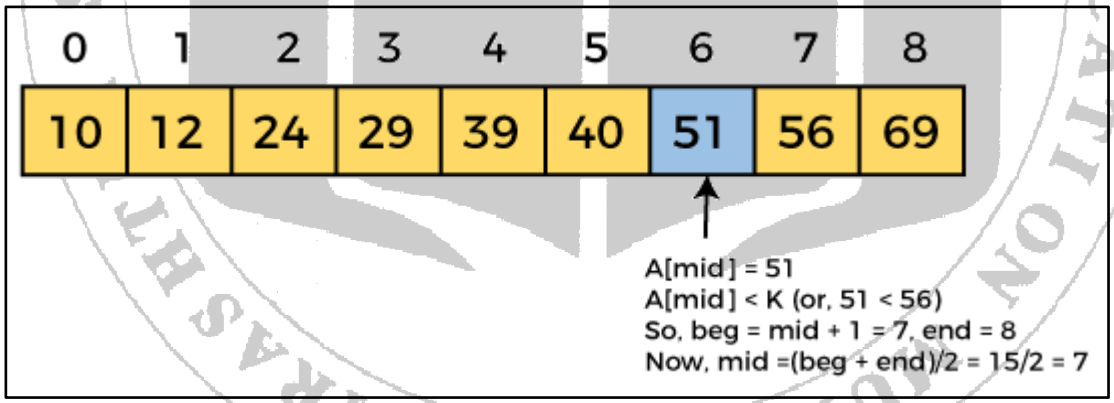
mid = (beg + end)/2

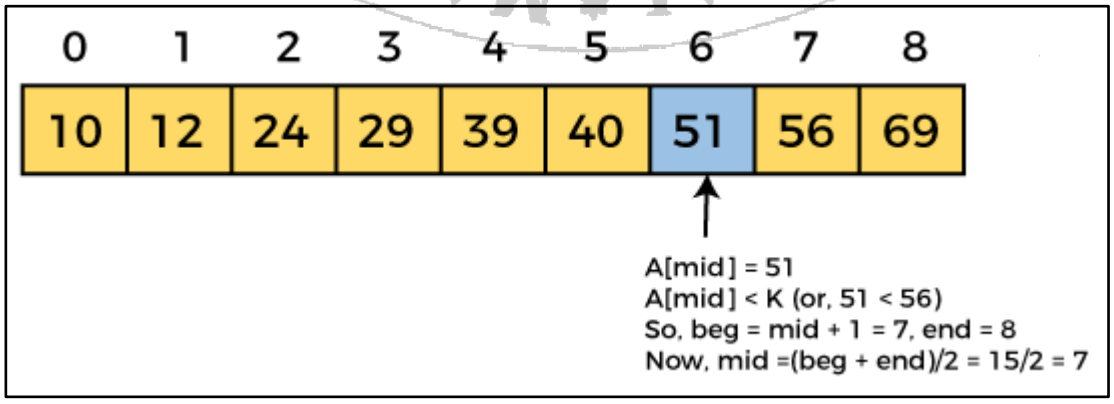
So, in the given array -

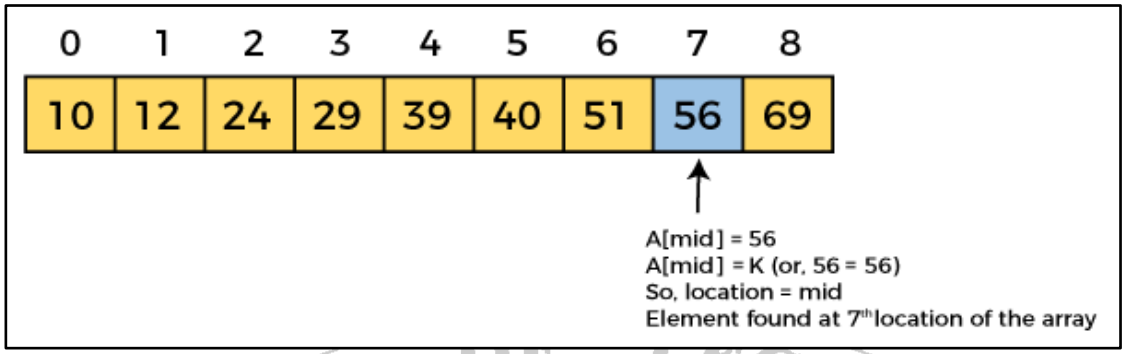
beg = 0

end = 8

mid = (0 + 8)/2 = 4. So, 4 is the mid of the array.







Now, the element to search is found. So algorithm will return the index of the element matched.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a program to find the first and last occurrence of the element 3 in an array of 20 integers using binary search.
2. Given an array of 15 integers, write a program to find the two middle elements using binary search.

XV Exercise

1. State the applications of Binary Search
2. Compare Binary Search with Linear Search
3. Discuss and write the requirements for using binary search on an array of numbers.

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.javatpoint.com/data-structure-introduction
2. https://www.geeksforgeeks.org/introduction-to-data-structures/

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved (LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.5: Write a 'C' Program to Search a particular data from the given Array of Strings using Binary Search Method

I Practical Significance

Binary search is a highly efficient algorithm for finding a specific value in a sorted array. It significantly reduces the search time complexity to O(log n) by repeatedly dividing the search interval in half, making it particularly useful for large datasets.

II Industry/Employer Expected outcome

The binary search algorithm's efficiency is crucial in various industries, particularly in database management, information retrieval, and software development.

1. Enable rapid data lookup and retrieval
2. Enhance performance in applications like search engines, data analytics platforms, and inventory management systems
3. Lead to faster query responses and improved user experiences.

III Course Level Learning Outcome(s)

Perform operations on Stack using Array and Linked List Implementations.

IV Laboratory Learning Outcome(s)

Implement Binary Search Method on Strings.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Binary search is a popular algorithm for finding an item from a sorted list of items, and it can be applied to strings as well as numbers. When applying binary search to strings in C, there are some important considerations and steps to follow. Below, I will outline the theory and implementation details for performing a binary search on an array of strings in C.

Theory

Array of Strings: The binary search algorithm requires the array to be sorted. For strings, this means they should be sorted lexicographically.

Comparison Function: Since we are dealing with strings, we need a way to compare them. In C, the `strcmp` function from the C standard library can be used for this purpose. `strcmp` compares two strings and returns:

- A negative value if the first string is less than the second string.
- Zero if the two strings are equal.
- A positive value if the first string is greater than the second string.

Algorithm Steps:

- Start with two pointers, `left` and `right`, which initially point to the first and last elements of the array, respectively.
- Calculate the middle index.
- Compare the middle element with the target string using `strcmp`.
- If the middle element matches the target string, return the index.
- If the target string is less than the middle element, search in the left half of the array.
- If the target string is greater than the middle element, search in the right half of the array.
- Repeat the process until the pointers cross or the target string is found.

VII Algorithm

VIII Flow Chart

IX C Program Code

XI Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XII Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Follow safety practices.

XIII Result (Output of Program)

XIV Conclusion

XV Practical Related Questions

1. Write a C program that performs binary search on a sorted array of strings. The program should:

- Read a sorted list of strings from the user.
- Ask the user for a target string to search for.
- Use binary search to find the target string in the list.
- Print the index of the target string if found, or a message indicating that the string is not found.

XVI Exercise

1. Modify the binary search algorithm to be case-insensitive when comparing strings.
2. Write Time Complexity of Binary Search

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVII References / Suggestions for further Reading Software/Learning Websites

1. https://www.javatpoint.com/data-structure-introduction
2. https://www.geeksforgeeks.org/introduction-to-data-structures/

XVIII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related (30)	**Product Related (20)**	**Total (50)**	

Practical No.6: * Write a 'C' Program to Sort an Array of numbers using Bubble Sort Method.

I Practical Significance

Bubble sort is one of the simplest sorting algorithms that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items, and swapping them if they are in the wrong order. This process is repeated until no swaps are needed, which indicates that the list is sorted.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Illustrate basic algorithmic principles, such as iteration, comparison, and swapping.
2. Use Bubble sort for small datasets
3. Highlight importance of algorithmic efficiency
4. Understand Timecomplexity

III Course Level Learning Outcomes

Apply different Searching and Sorting methods.

IV Laboratory Learning Outcome

Apply Bubble Sort method for Sorting Numbers

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Bubble sort is one of the simplest sorting algorithms that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items, and swapping them if they are in the wrong order. This process is repeated until no swaps are needed, which indicates that the list is sorted.

Key Characteristics:

Simple Implementation: Easy to understand and implement.

Comparison-Based: Each element is compared with its adjacent element.

In-Place Sorting: Requires a constant amount of additional memory space.

Stable Sort: Maintains the relative order of records with equal keys (i.e., values).

In summary, the bubble sort algorithm repeatedly swaps adjacent elements until the array is sorted. The algorithm has a time complexity of O(n2), which makes it inefficient for large data sets.

Working of Bubble Sort

Suppose we are trying to sort the elements in **ascending order**.

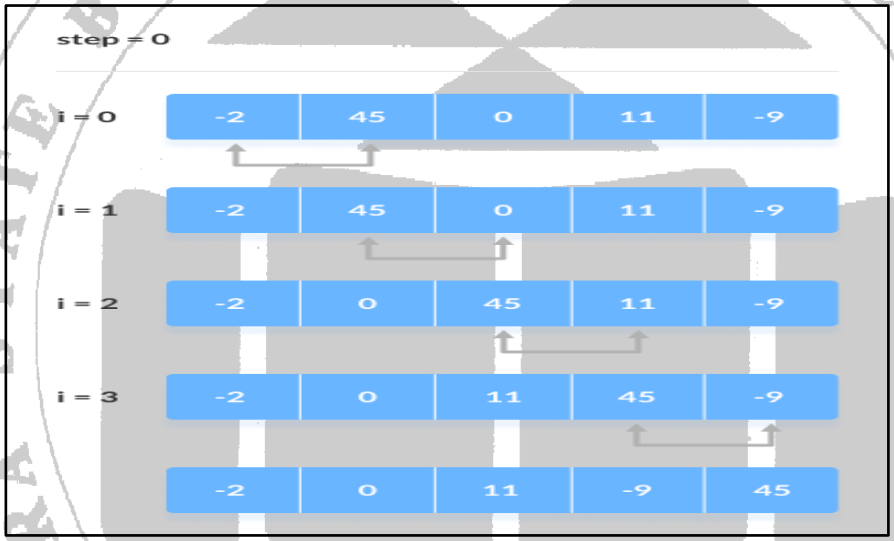
1. First Iteration (Compare and Swap)

Starting from the first index, compare the first and the second elements.

If the first element is greater than the second element, they are swapped.

Now, compare the second and the third elements. Swap them if they are not in order.

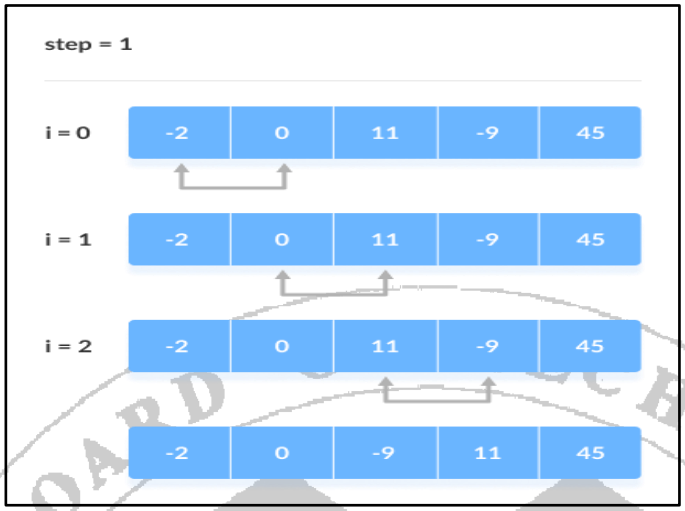
The above process goes on until the last element.



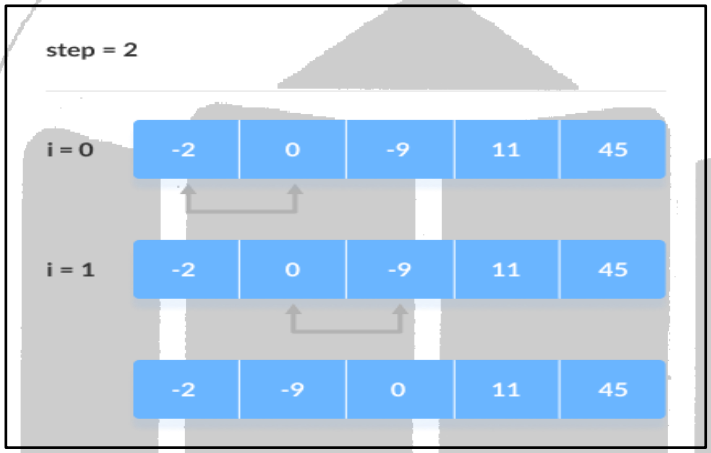
2. Remaining Iteration

The same process goes on for the remaining iterations.

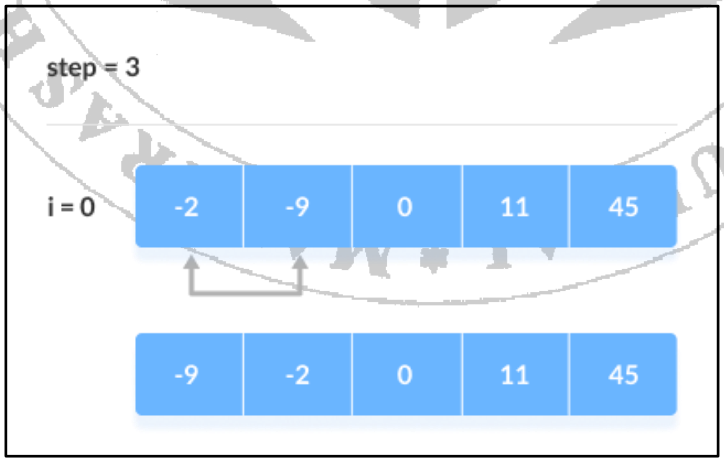
After each iteration, the largest element among the unsorted elements is placed at the end.



In each iteration, the comparison takes place up to the last unsorted element.



The array is sorted when all the unsorted elements are placed at their correct positions.



The array is sorted if all elements are kept in the right order

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Static memory allocation using fixed memory size
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION MUMBAI

XIV Practical Related Questions

1. Optimize the Bubble Sort algorithm to stop early if the array is already sorted.
2. Modify the Bubble Sort algorithm to sort an array in descending order.

XV Exercise

1. Enlist Applications of Bubble Sort.
2. Explain why Bubble sort is called as stable Sort.

[Space for Answer]

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. Bubble sort program in C - javatpoint
2. DSA Bubble Sort (w3schools.com)

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.7: * Write a 'C' Program to Sort an Array of Strings using Bubble Sort Method

I Practical Significance

Bubble sort is one of the simplest sorting algorithms that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items, and swapping them if they are in the wrong order. This process is repeated until no swaps are needed, which indicates that the list is sorted.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Illustrate basic algorithmic principles, such as iteration, comparison, and swapping.
2. Use Bubble sort for small datasets
3. Highlight importance of algorithmic efficiency
4. Understand Timecomplexity

III Course Level Learning Outcomes

Apply different Searching and Sorting methods.

IV Laboratory Learning Outcome

Apply Bubble Sort method for Sorting Strings

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

How Bubble Sort Works:

Bubble sort works by repeatedly stepping through the list, comparing adjacent elements, and swapping them if they are in the wrong order.

This process is repeated until no more swaps are needed, which means the list is sorted.

The name "bubble sort" comes from the way smaller elements "bubble" to the top of the list through the swaps.

Sure! Let's walk through an example of sorting an array of strings using the Bubble Sort algorithm in C.

Example:

Suppose we have the following array of strings:

["banana", "apple", "cherry", "date"]

We will sort this array in lexicographical order using Bubble Sort.

Below is the detailed step-by-step process and

Step-by-Step Process:

1. Initial Array:

["banana", "apple", "cherry", "date"]

2. First Pass:

- Compare "banana" and "apple". Since "banana" > "apple", swap them.

["apple", "banana", "cherry", "date"]

- Compare "banana" and "cherry". No swap needed as "banana" < "cherry".
- Compare "cherry" and "date". No swap needed as "cherry" < "date".

3. Second Pass:

- Compare "apple" and "banana". No swap needed as "apple" < "banana".
- Compare "banana" and "cherry". No swap needed as "banana" < "cherry".

4. Third Pass:

- Compare "apple" and "banana". No swap needed as "apple" < "banana".

Since no swaps were needed in the last pass, the array is now sorted.

Sorted Array:

["apple", "banana", "cherry", "date"]

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Modify the basic Bubble Sort algorithm to include an optimization that stops the algorithm if no swaps were made during a pass.
2. Modify the program to handle invalid input, such as non-string data or strings that exceed the maximum length.

XV Exercise

1. Explain why Bubble sort is not suitable for large datasets.
2. Explain How can you handle sorting array of strings having mix uppercase and lowercase characters?

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

,,

XVI References / Suggestions for further Reading Software/Learning Websites

1. Bubble Sort - Data Structure and Algorithm Tutorials - GeeksforGeeks
2. Bubble sort program in C - javatpoint

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related (30)	**Product Related (20)**	**Total(50)**	

Practical No.8: * Write a 'C' Program to Sort an Array of numbers using Selection Sort Method.

I Practical Significance

Selection Sort is an in-place comparison sorting algorithm. It divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element from the unsorted sublist, exchanging it with the leftmost unsorted element, and moving the sublist boundaries one element to the right.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Introduce sorting algorithms and develop algorithmic thinking
2. Use Selection sort in small utility programs
3. Use Selection sort in Embedded Systems
4. Perform minimal number of write operations

III Course Level Learning Outcomes

Apply different Searching and Sorting methods.

IV Laboratory Learning Outcome

Apply Selection Sort for Sorting numbers

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Selection Sort

Selection Sort is a simple and intuitive comparison-based sorting algorithm. Despite its simplicity, it is rarely used in practice due to its inefficiency on larger datasets. Here's a detailed overview of the theory behind Selection Sort:

Concept

Selection Sort works by repeatedly finding the minimum (or maximum, depending on the sorting order) element from the unsorted portion of the list and moving it to the beginning (or end) of the sorted portion. This process continues until the entire list is sorted.

Let's see the basic working and principles of the selection sort algorithm using your example array: **arr[] = {17, 34, 25, 49, 09}**

Selection Sort Working Principle:

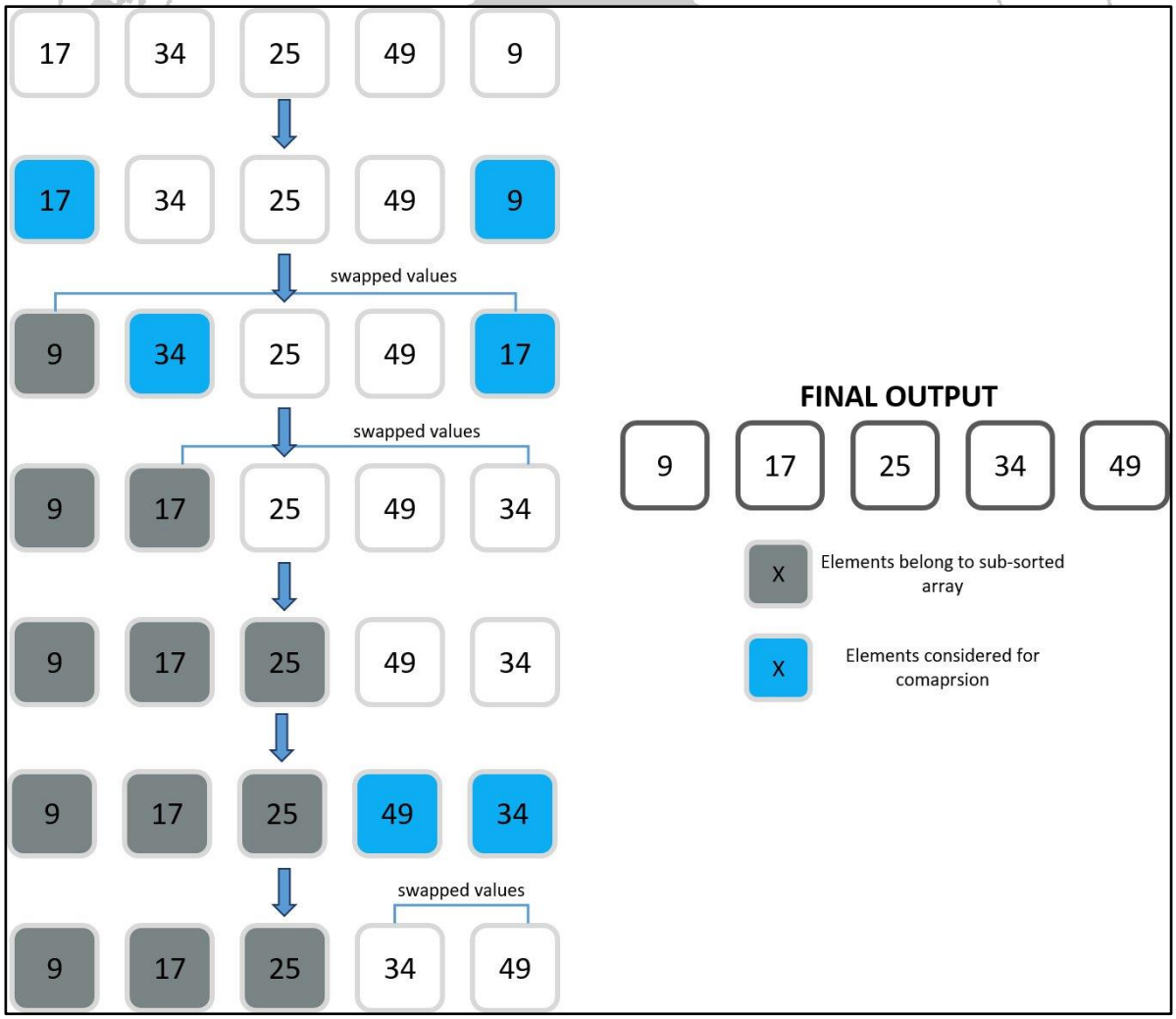
Initialization: Start with the first element as the current "minimum" (initially assuming it's the smallest).

Search for Minimum: Iterate through the remaining unsorted elements, comparing each one with the current "minimum." If you find a smaller element, update the current "minimum."

Swap: After completing the iteration and finding the actual minimum among the unsorted elements, swap it with the first unsorted element (the one you started with).

Repeat: Repeat steps 1-3 for the next unsorted element, then the next, until the entire array is sorted.

Here's how selection sort works step-by-step for your array



VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Modify the Selection Sort algorithm to handle arrays containing negative numbers.
2. Adapt the Selection Sort algorithm to sort an array of floating-point numbers.

XV Exercise

1. Write Applications of Selection Sort.
2. Compare Selection Sort with Bubble Sort.

[Space for Answer]

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. Selection Sort – Data Structure and Algorithm Tutorials - GeeksforGeeks
2. Selection Sort - javatpoint
3. C Program for Selection Sort? (tutorialspoint.com)

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved (LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.9: * Write a 'C' Program to Sort an Array of Strings using Selection Sort Method.

I Practical Significance

Selection Sort is an in-place comparison sorting algorithm. It divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element from the unsorted sublist, exchanging it with the leftmost unsorted element, and moving the sublist boundaries one element to the right.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Introduce sorting algorithms and develop algorithmic thinking
2. Use Selection sort in small utility programs
3. Use Selection sort in Embedded Systems
4. Perform minimal number of write operations

III Course Level Learning Outcomes

Apply different Searching and Sorting methods.

IV Laboratory Learning Outcome

Apply Selection Sort for Sorting Strings

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Selection Sort

Selection Sort is a simple and intuitive comparison-based sorting algorithm. Despite its simplicity, it is rarely used in practice due to its inefficiency on larger datasets. Here's a detailed overview of the theory behind Selection Sort:

Concept

Selection Sort works by repeatedly finding the minimum (or maximum, depending

on the sorting order) element from the unsorted portion of the list and moving it to the beginning (or end) of the sorted portion. This process continues until the entire list is sorted.

Steps of Selection Sort

Initialize the sorted sublist: The sorted sublist is initially empty, and the unsorted sublist contains all the elements.

Find the minimum element: Find the smallest element in the unsorted sublist.

Swap the minimum element: Swap the smallest element found with the first element of the unsorted sublist, effectively expanding the sorted sublist by one element and shrinking the unsorted sublist by one element.

Repeat: Repeat the process for the remaining unsorted sublist until the entire list is sorted.

Example-

Consider the array of strings: ["banana", "apple", "cherry", "date", "elderberry"]

We want to sort this array lexicographically using Selection Sort.

1. Initial Array:

["banana", "apple", "cherry", "date", "elderberry"]

2. First Pass:

- Find the minimum string in the array, which is "apple".
- Swap "apple" with the string at the first position.

["apple", "banana", "cherry", "date", "elderberry"]

3. Second Pass:

- Find the minimum string in the remaining array, which is "banana".
- Swap "banana" with the string at the second position.

["apple", "banana", "cherry", "date", "elderberry"]

4. Third Pass:

- The minimum string in the remaining array is "cherry", which is already in its correct position.

["apple", "banana", "cherry", "date", "elderberry"]

5. Fourth Pass:

- The minimum string in the remaining array is "date", which is already in its correct position.

["apple", "banana", "cherry", "date", "elderberry"]

6. Fifth Pass:

- The remaining string "elderberry" is already in its correct position as it is the largest.

["apple", "banana", "cherry", "date", "elderberry"]

7. Array is Sorted:

- The array is now sorted lexicographically.

["apple", "banana", "cherry", "date", "elderberry"]

This demonstrates how Selection Sort operates on an array of strings, progressively finding the smallest string and moving it to its correct position in each pass.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to sort an array of Strings using Selection Sort.
2. Extend your Selection Sort implementation to support case-insensitive sorting of strings. Ensure that uppercase and lowercase versions of the same letter are treated as equal during sorting.

XV Exercise

1. Discuss whether Selection Sort maintains the stability of the original array when sorting an array of strings. Provide reasoning for your answer and, if necessary, demonstrate with examples.
2. Explain why Selection Sort is considered inefficient for large datasets.

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. Selection Sort – Data Structure and Algorithm Tutorials - GeeksforGeeks

2. Selection Sort - javatpoint

3. C Program for Selection Sort? (tutorialspoint.com)

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related (20)**	**Total(50)**	

Practical No.10: * Write a 'C' Program to Sort an Array of numbers using Insertion Sort Method

I Practical Significance

Insertion sort is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list. It is a **stable sorting** algorithm, meaning that elements with equal values maintain their relative order in the sorted output.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Sort Relatively small data
2. Work on incremental data changes
3. Maintain sorted Lists
4. Use Insertion sort for small partitions of data

III Course Level Learning Outcomes

Apply different searching and sorting methods

IV Laboratory Learning Outcome

Apply Insertion Sort for Sorting Numbers

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

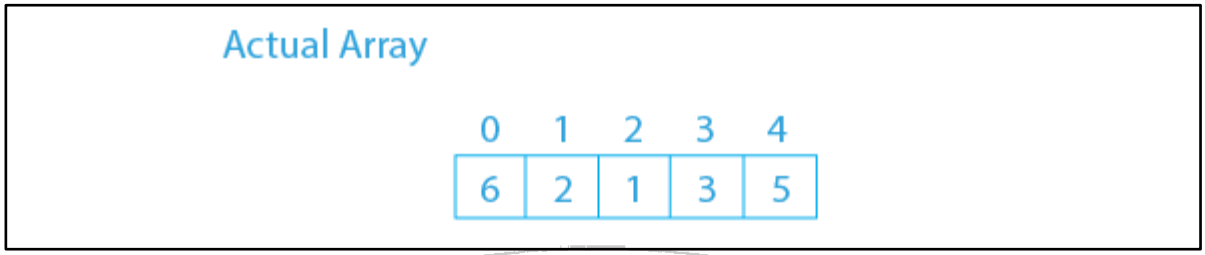
VI Relevant Theoretical Background

Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms. However, it has the advantage of being easy to implement and efficient for small data sets.

Insertion sort works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

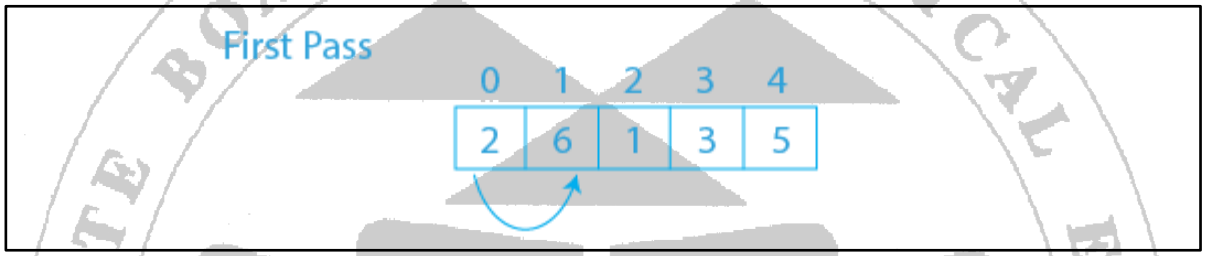
The same approach is applied in insertion sort. The idea behind the insertion sort is that first take one element, iterate it through the sorted array. Although it is simple to use, it is not appropriate for large data sets as the time complexity of insertion sort in the average case and worst case is O(n2), where n is the number of items. Insertion sort is less efficient than the other sorting algorithms like heap sort, quick sort, merge sort, etc.

Now that we have clarity of the concept and working of the Insertion Sort program in C, let us take an example array and dry run with illustrations to understand the working in a step by step manner:-



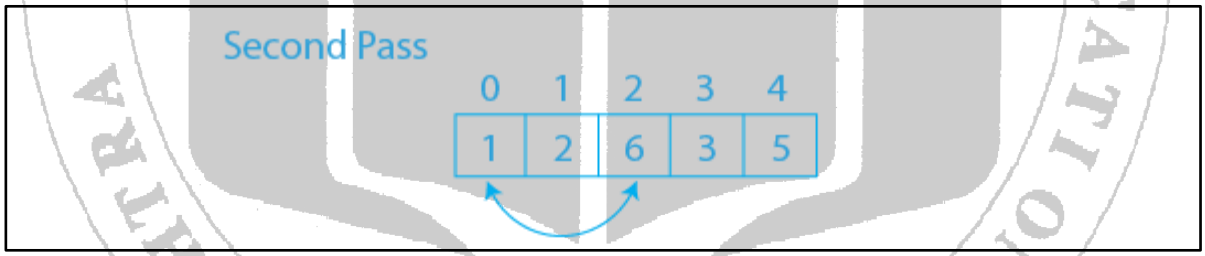
Suppose we have an array [6,2,1,3,5], the insertion sort algorithm considers the first element is already sorted. There are going to be n-1 passes to sort the array in ascending order.

First Pass:



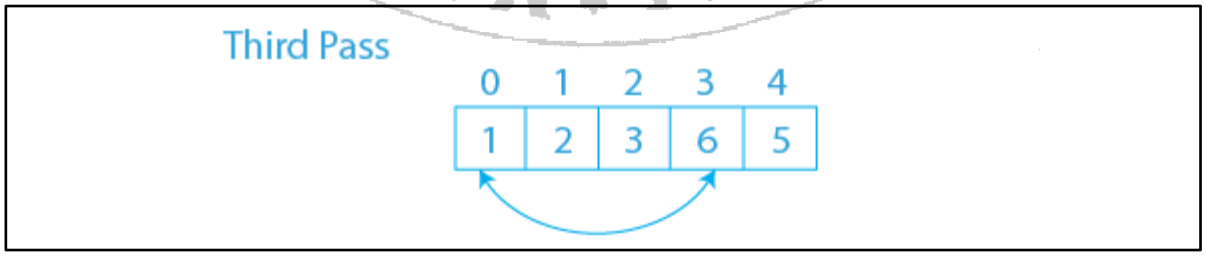
The second element i.e. 2 is compared to the first element 6, which is lesser, hence swapping is performed with the first pass making it [6,6,1,3,5], ending with the processed array [2,6,1,3,5]

Second Pass:



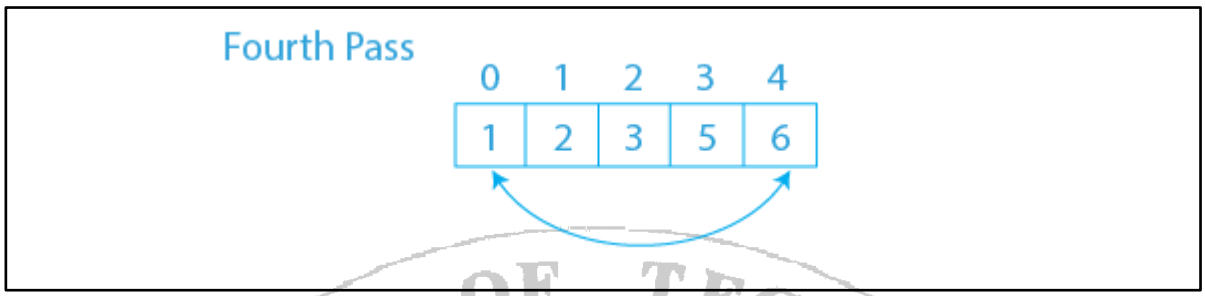
For this pass, the index will be at the 2nd position or 3rd element, as the swapping is valid now that 1>6, the array turns [2,6,6,3,5] and on the following iteration, the result of this pass is obtained as [2,2,6,3,5] and the final processed array obtained becomes [1,2,6,3,5]

Third Pass:



As the array up and until now is [1,2,6,3,5], 3 will be the key, with its transition being, [1,2,6,6,5] and result as [1,2,3,6,5] on the assignment of the key.

Fourth Pass:



The n-1th and also the last pass on this array of 5 elements, only one iteration will be possible with 5 replacing the second last element 6 i.e. [1,2,3,6,6] and assignment of the key to give us the obtained array in the form of [1,2,3,5,6] as the output.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Modify the Insertion Sort algorithm to use binary search for finding the correct position to insert the current element. Implement this modified algorithm and compare its performance with the standard Insertion Sort.
2. Use the Insertion Sort algorithm to count the number of inversions in an array. An inversion is a pair of elements where the earlier element is greater than the later element.

XV Exercise

1. Write Time and Space Complexities of Insertion Sort.
2. List Applications of Insertion Sort
3. How does insertion sort handle duplicate values in the array?

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. Insertion Sort - Data Structure and Algorithm Tutorials - GeeksforGeeks
2. Insertion Sort - javatpoint

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.11: * Write a 'C' Program to Sort an Array of Strings using Insertion Sort Method

I Practical Significance

Insertion sort is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list. It is a **stable sorting** algorithm, meaning that elements with equal values maintain their relative order in the sorted output.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Sort Relatively small data
2. Work on incremental data changes
3. Maintain sorted Lists
4. Use Insertion sort for small partitions of data

III Course Level Learning Outcomes

Apply different searching and sorting methods

IV Laboratory Learning Outcome

Apply Insertion Sort for sorting numbers

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Insertion sort is a simple sorting algorithm that works by building a sorted array one element at a time. It repeatedly takes an element from the unsorted part and inserts it into its correct position in the sorted part of the array.

Example:

Let's consider an array of strings: ["banana", "apple", "cherry", "date"]

Start with the second element "apple".

Compare "apple" with "banana". Since "apple" < "banana", no swap is needed.

"apple" is in the correct position relative to "banana".

Next, compare "apple" with "cherry". "apple" < "cherry", so it stays in its position.

Now compare "apple" with "date". "apple" < "date", so it stays in its position.

Move to the next unsorted element "cherry".

Compare "cherry" with "banana". Since "banana" < "cherry", "cherry" move one position to the right.

Now, compare "cherry" with "apple". Since "apple" < "cherry", "cherry" moves one position to the right again.

"cherry" is now in the correct position.

Repeat this process for "date".

Time Complexity:

In the worst-case scenario, when the array is in reverse order, insertion sort has a time complexity of O(n^2).

In the best-case scenario, when the array is already sorted, insertion sort has a time complexity of O(n).

The average-case time complexity is also O(n^2).

Space Complexity:

Insertion sort typically has a space complexity of O(1) since it sorts the array in-place without requiring any additional space proportional to the size of the input.

Stability:

Insertion sort is a stable sorting algorithm, meaning it preserves the relative order of equal elements.

Adaptiveness:

Insertion sort is adaptive. It performs better when the input array is partially sorted or nearly sorted, as it requires fewer comparisons and swaps in such case s.

Practical Use Cases:

Insertion sort is often used for small arrays or partially sorted arrays where simplicity and adaptiveness outweigh the efficiency concerns of other sorting algorithms.

It's also useful in scenarios where stability is important, such as sorting structures with multiple fields where one field serves as the primary key and the other as a secondary key.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Create an interactive game where the user enters a list of strings, and the program sorts them using insertion sort. Allow the user to add or remove strings dynamically and see the sorting process step by step.
2. Implement insertion sort with a custom comparison function that sorts strings based on a specific criterion (e.g., number of vowels, number of consonants, etc.).

XV Exercise

1. Write when it is suitable to use Insertion Sort.
2. List Applications of Insertion Sort
3. Explain how does insertion sort handle strings with different lengths?

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. Insertion Sort - Data Structure and Algorithm Tutorials - GeeksforGeeks
2. Insertion Sort - javatpoint

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.12: *Write a 'C' Program to Implement Singly Linked List with Operations: (i) Insert at beginning, (ii) Search, (iii) Display

I Practical Significance

A **Singly Linked List** is a linear data structure in which the elements are not stored in contiguous memory locations and each element is connected only to its next element using a pointer.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Provide a reliable and efficient data structure that can be used as a building block for more complex algorithms and applications.
2. Perform basic operations such as insertion, deletion, and traversal without errors or unexpected behavior.
3. Minimize memory usage.
4. Optimize traversal and search operations

III Course Level Learning Outcomes

Implement basic operations on Linked List.

IV Laboratory Learning Outcome

Create Singly Linked List

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.

A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.

The last node of the list contains pointer to the null.

Uses of Linked List

The list is not required to be contiguously present in the memory. The node can reside anywhere in the memory and linked together to make a list. This achieves optimized utilization of space.

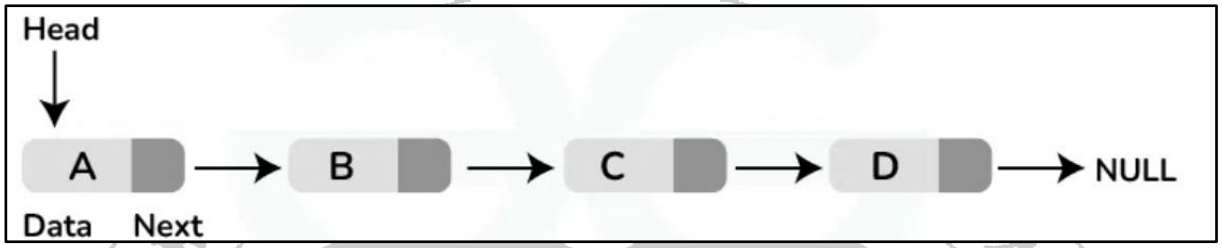
list size is limited to the memory size and doesn't need to be declared in advance.

Empty node cannot be present in the linked list.

We can store values of primitive types or objects in the singly linked list.

Understanding Node Structure:

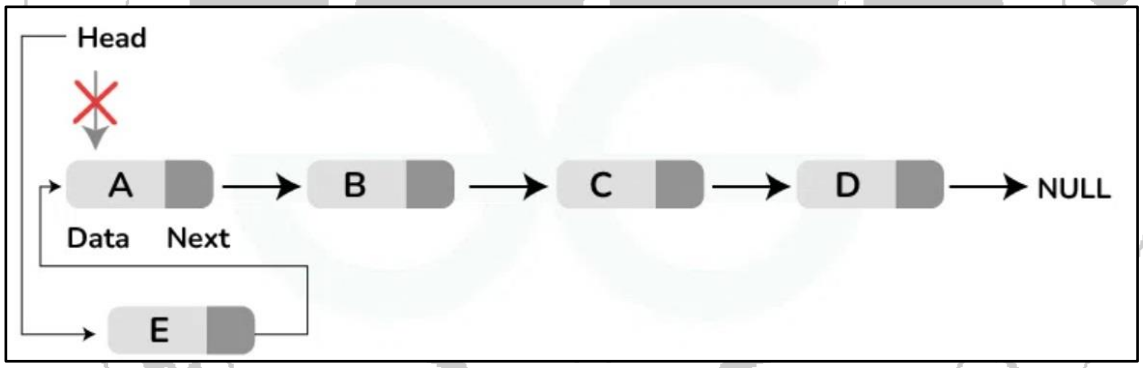
In a singly linked list, each node consists of two parts: data and a pointer to the next node. The data part stores the actual information, while the pointer (or reference) part stores the address of the next node in the sequence. This structure allows nodes to be dynamically linked together, forming a chain-like sequence.



Insertion in Singly Linked List

Insertion is a fundamental operation in linked lists that involves adding a new node to the list. There are several scenarios for insertion:

Insertion at the beginning



To insert a node at the start/beginning/front of a Linked List, we need to:

1. Make the first node of Linked List linked to the new node
2. Remove the head from the original first node of Linked List
3. Make the new node as the Head of the Linked List.

Traversal in Singly Linked List:

Traversal involves visiting each node in the linked list and performing some operation on the data. A simple traversal function would print or process the data of each node.

Steps for Traversal in Singly Linked List:

1. Initialize a pointer current to the head of the list.
2. Use a while loop to iterate through the list until the current pointer reaches nullptr.
3. Inside the loop, print the data of the current node and move the current pointer to the next node.

Searching in Singly Linked List:

Searching in a Singly Linked List refers to the process of looking for a specific element or value within the elements of the linked list.

Steps for Searching in Singly Linked List:

1. Initialize a pointer head to the starting node of the Linked List.
2. Use a while loop to traverse the Linked List until the end is reached (i.e., head becomes nullptr).
3. Inside the loop, check if the current node's data is equal to the target value.
4. If true, return true indicating that the value is found in the Linked List.
5. If false, move to the next node by updating head to point to the next node in the list.
6. If the loop completes without finding the value, return false indicating that the value is not present in the Linked List.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a function to insert a node at the beginning of a Singly Linked List.
2. Write a function to detect if a singly linked list has a cycle. If a cycle is detected, return the starting node of the cycle.

XV Exercise

1. Explain the use of Singly Linked List
2. Write advantages and disadvantages of Singly Linked List
3. Calculate length of Singly Linked List

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. Introduction to Singly Linked List - GeeksforGeeks
2. Linked List (Data Structures) - javatpoint

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.13: *Write a 'C' Program to Implement Singly Linked List with Operations: (i) Insert at end (ii) Insert after (iii)Delete(iv) Display

I Practical Significance

A **Singly Linked List** is a linear data structure in which the elements are not stored in contiguous memory locations and each element is connected only to its next element using a pointer.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Provide a reliable and efficient data structure that can be used as a building block for more complex algorithms and applications.
2. Perform basic operations such as insertion, deletion, and traversal without errors or unexpected behavior.
3. Minimize memory usage
4. Optimize traversal and search operations

III Course Level Learning Outcomes

Implement basic operations on Linked List.

IV Laboratory Learning Outcome

Create Singly Linked List

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.

A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.

The last node of the list contains pointer to the null.

Uses of Linked List

The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.

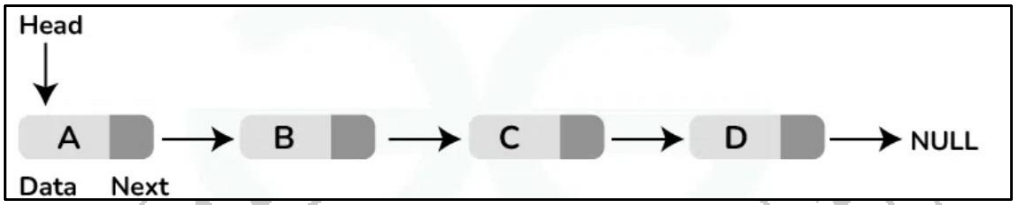
list size is limited to the memory size and doesn't need to be declared in advance.

Empty node cannot be present in the linked list.

We can store values of primitive types or objects in the singly linked list.

Understanding Node Structure:

In a singly linked list, each node consists of two parts: data and a pointer to the next node. The data part stores the actual information, while the pointer (or reference) part stores the address of the next node in the sequence. This structure allows nodes to be dynamically linked together, forming a chain-like sequence.



Insertion in Singly Linked List

Insertion is a fundamental operation in linked lists that involves adding a new node to the list. There are several scenarios for insertion:

Insertion at the end

In order to insert a node at the last, there are two following scenarios which need to be mentioned.

The node is being added to an empty list.

The node is being added to the end of the linked list

In the First Case

The condition (head == NULL) gets satisfied. Hence, we just need to allocate the space for the node by using malloc statement in C. Data and the link part of the node are set up by using the following statements.

ptr->data = item;

ptr -> next = NULL;

Since, **ptr** is the only node that will be inserted in the list hence, we need to make this node pointed by the head pointer of the list. This will be done by using the following Statements.

Head = ptr

In the second case

The condition **Head = NULL** would fail, since Head is not null. Now, we need to declare a temporary pointer temp in order to traverse through the list. **temp** is made to point the first node of the list.

Temp = head

Then, traverse through the entire linked list using the statements:

while (temp→ next != NULL)

temp = temp → next;

At the end of the loop, the temp will be pointing to the last node of the list. Now, allocate the space for the new node, and assign the item to its data part. Since, the new node is going to be the last node of the list hence, the next part of this node needs to be pointing to the **null**. We need to make the next part of the temp node (which is currently the last node of the list) point to the new node (ptr) .

```
Temp=head;
    while (temp -> next != NULL)
    {
       temp = temp -> next;
         }
    temp->next = ptr;
    ptr->next = NULL;
```

Deletion in singly linked list at beginning

Deleting a node from the beginning of the list is the simplest operation of all. It just need a few adjustments in the node pointers. Since the first node of the list is to be deleted, therefore, we just need to make the head, point to the next of the head. This will be done by using the following statements.

ptr = head;

head = ptr->next;

Now, free the pointer ptr which was pointing to the head node of the list. This will be done by using the following statement.

free(ptr)

Deletion in singly linked list at the end

There are two scenarios in which, a node is deleted from the end of the linked list.

There is only one node in the list and that needs to be deleted.

There are more than one node in the list and the last node of the list will be deleted.

In the first scenario, the condition head → next = NULL will survive and therefore, the only node head of the list will be assigned to null. This will be done by using the following statements.

ptr = head

head = NULL

free(ptr)

In the second scenario, the condition head → next = NULL would fail and therefore, we have to traverse the node in order to reach the last node of the list.

For this purpose, just declare a temporary pointer temp and assign it to head of the list. We also need to keep track of the second last node of the list. For this purpose, two pointers ptr and ptr1 will be used where ptr will point to the last node and ptr1 will point to the second last node of the list.

```
ptr = head;
while(ptr->next != NULL)
{  ptr1 = ptr;
ptr = ptr ->next;
}
```

Now, we just need to make the pointer ptr1 point to the NULL and the last node of the list that is pointed by ptr will become free. It will be done by using the following statements.

```
ptr1->next = NULL;
free(ptr);
```

Traversal in Singly Linked List:

Traversal involves visiting each node in the linked list and performing some operation on the data. A simple traversal function would print or process the data of each node.

Steps for Traversal in Singly Linked List:

Initialize a pointer current to the head of the list.

Use a while loop to iterate through the list until the current pointer reaches nullptr.

Inside the loop, print the data of the current node and move the current pointer to the next node.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a function to insert a node at the end in a Singly Linked List.
2. Write a function to check whether a singly linked list is a palindrome or not.

XV Exercise

1. Write applications of Singly Linked List.
2. Explain the procedure to find middle of singly linked list.

[Space for Answer]

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. Introduction to Singly Linked List - GeeksforGeeks
2. Linked List (Data Structures) - javatpoint

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.14: Write a C Program to Create Two Polynomials using a Linked List.

I Practical Significance

Polynomial is a fundamental operation in mathematics and computer science, widely used in various applications such as signal processing, computer graphics, and scientific simulations. When dealing with polynomials, it is essential to have an efficient and flexible data structure to represent and perform operations on them. Linked lists provide an elegant solution for efficiently handling polynomials due to their dynamic memory allocation and straightforward implementation.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Represent node structure of linked list in order to create polynomial.
2. Write an algorithm to create Two Polynomials using a Linked List.
3. Develop simple C program to create Two Polynomials using a Linked List.
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Understand how to create, insert a node to form a polynomial using linked list structure.

IV Laboratory Learning Outcome

Create Polynomials using Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Represent a Polynomial with a Linked List:

We can use a linked list to represent a polynomial. In the linked list, each node has two data fields: coefficient and power and one address field: address of next node.

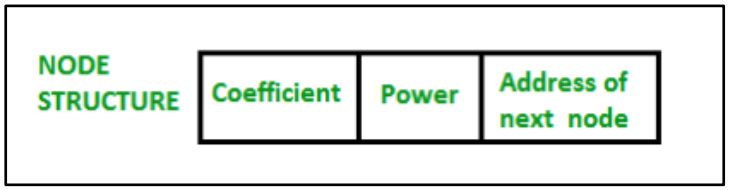


Figure 14.1: Node structure of linked list

Let us consider an example an example to show how the addition of two polynomials is performed,

P(x) = 3x4 + 2x3 - 4 x2 + 7

Q (x) = 5x3 + 4 x2 - 5

These polynomials are represented using a linked list in order of decreasing exponents as follows:

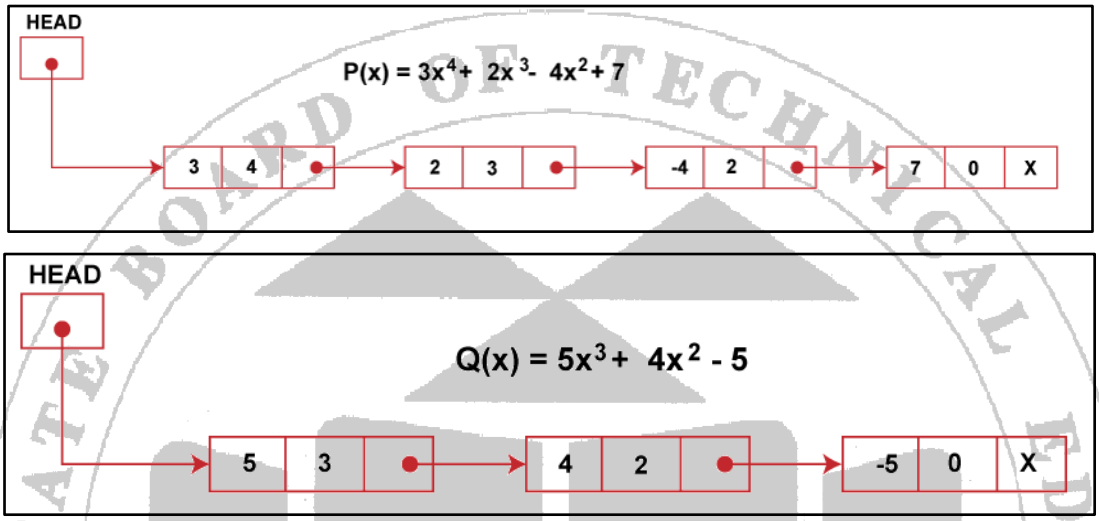


Figure 14.2: Linked list representation of polynomial

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of linked list to create polynomials
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a node structure to represent a polynomial using linked list
2. Write a C program to Create a polynomial 5x4 + 3x2 + 1.
3. Write a C program to display created polynomial.

XV Exercise

1. Give reason for using linked lists for polynomial creation?
2. Compare creating polynomial using Array and Linked List. (any two points)

[Space for Answer]

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.prepbytes.com/blog/linked-list/adding-two-polynomials-using-linked-list/
2. https://www.javatpoint.com/application-of-linked-list
3. https://www.geeksforgeeks.org/adding-two-polynomials-using-linked-list/
4. https://gist.github.com/jamesgeorge007/336622f348815f0bef589bae5b622cff

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved (LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.15: *Write a C Program to add Two Polynomials using a Linked List.

I Practical Significance

Polynomial is a fundamental operation in mathematics and computer science, widely used in various applications such as signal processing, computer graphics, and scientific simulations. When dealing with polynomials, it is essential to have an efficient and flexible data structure to represent and perform operations on them. Linked lists provide an elegant solution for efficiently handling polynomials due to their dynamic memory allocation and straightforward implementation.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Represent node structure of linked list in order to create polynomial.
2. Write an algorithm to create and add two Polynomials using a Linked List.
3. Develop simple C program to create Two Polynomials using a Linked List.
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Understand how to create, insert a node to form a polynomial using linked list structure.

IV Laboratory Learning Outcome

Create Polynomials using Linked List

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Addition of Polynomials:

To add two polynomials, we traverse the list P and Q. We take corresponding terms of the list P and Q and compare their exponents. If the two exponents are equal, the coefficients are added to create a new coefficient. If the new coefficient is equal to 0, then the term is dropped, and if it is not zero, it is inserted at the end of the new linked list containing the resulting polynomial. If one of the exponents is larger than the other, the corresponding term is immediately placed into the new linked list, and the term with the smaller exponent is held to be compared with the next term from the other list. If one list ends before the other, the rest of the terms of the longer list is inserted at the end of the new linked list containing the resulting polynomial.

Let us consider an example an example to show how the addition of two polynomials is performed,

P(x) = 3x4 + 2x3 - 4 x2 + 7

Q (x) = 5x3 + 4 x2 - 5

These polynomials are represented using a linked list in order of decreasing exponents as follows:

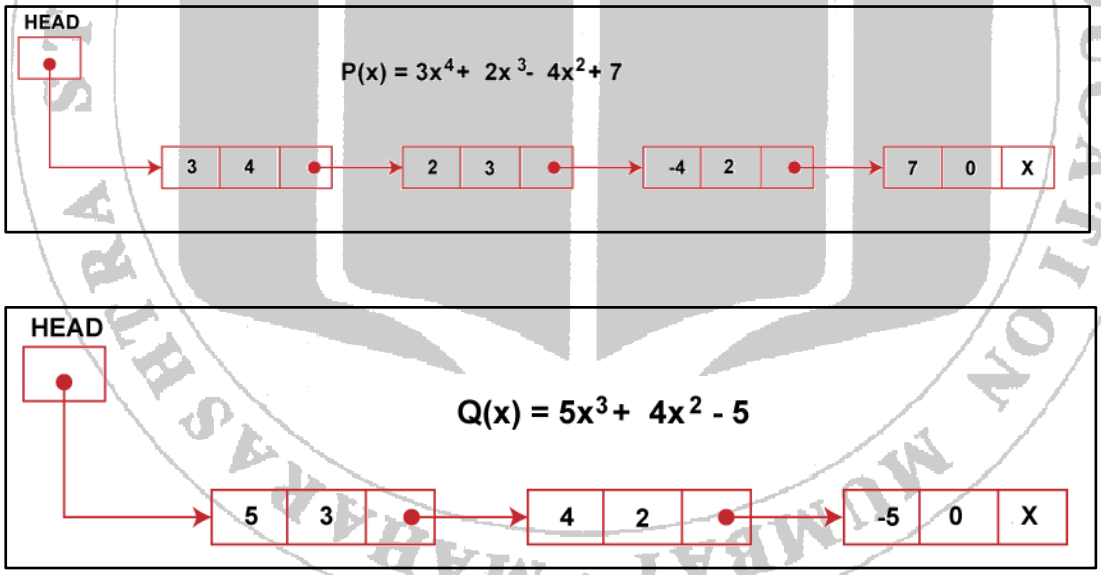
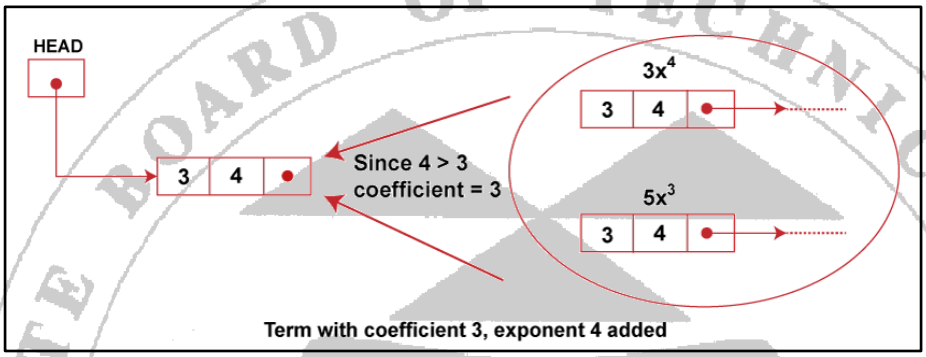


Figure 14.2: Linked list representation of polynomial

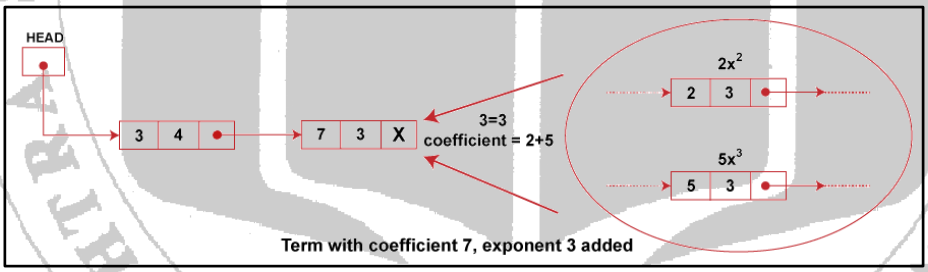
To generate a new linked list for the resulting polynomials that is formed on the addition of given polynomials P(x) and Q(x), we perform the following steps,

1. Traverse the two lists P and Q and examine all the nodes.
2. We compare the exponents of the corresponding terms of two polynomials. The first term of polynomials P and Q contain exponents 4 and 3, respectively. Since the exponent of the first term of the polynomial P is greater than the other polynomial Q, the term having a larger exponent is inserted into the new list.

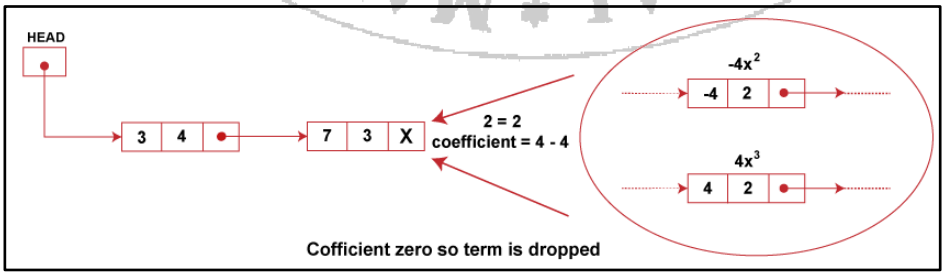
The new list initially looks as shown below:



1. We then compare the exponent of the next term of the list P with the exponents of the present term of list Q. Since the two exponents are equal, so their coefficients are added and appended to the new list as follows:



2. Then we move to the next term of P and Q lists and compare their exponents. Since exponents of both these terms are equal and after addition of their coefficients, we get 0, so the term is dropped, and no node is appended to the new list after this



3. Moving to the next term of the two lists, P and Q, we find that the corresponding terms have the same exponents equal to 0. We add their coefficients and append them to the new list for the resulting polynomial as shown below:

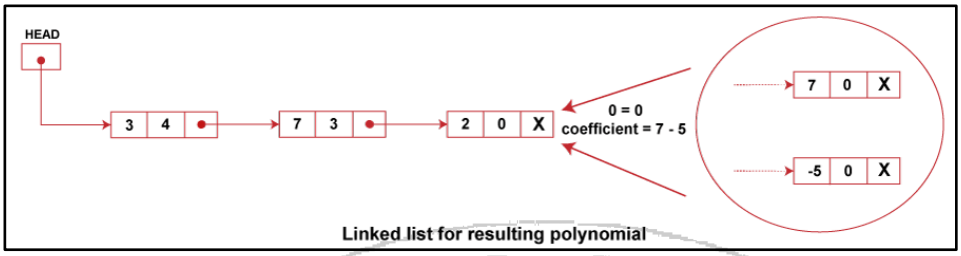


Figure 15.2 Resultant Polynomial- addition of two polynomial

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of linked list to create and two polynomials
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to Create two polynomial P(x) = 3x4 + 2x3 - 4 x2 + 7 and Q (x) = 5x3 + 4 x2 - 5
2. Write a C program to display addition of two created polynomial.

XV Exercise

1. Give applications of Polynomial Addition using Linked List in C.
2. List advantages of Polynomial Addition using Linked List in C

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.prepbytes.com/blog/linked-list/adding-two-polynomials-using-linked-list/
2. https://www.javatpoint.com/application-of-linked-list
3. https://www.geeksforgeeks.org/adding-two-polynomials-using-linked-list/
4. https://gist.github.com/jamesgeorge007/336622f348815f0bef589bae5b622cff

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.16: * Write a 'C' Program to perform PUSH and POP Operations on Stack using an Array.

I Practical Significance

Stacks are a fundamental data structure in computer science and have numerous practical applications based on their LIFO (Last In, First Out) property. Hence Stack is Linear Data Structure where insertion and deletion take place only from one end called as Stack Top.

II Industry/ Employer Expected Outcome

Through this practical student should

1. Understand Stack as ADT
2. Represent Stack in memory using Array.
3. Write an algorithm to perform PUSH and POP operations on Stack.
4. Develop simple C program to implement stack operation using Array.
5. Save/Compile/ Debug/ the C program.
6. Check for the desired output.

III Course Level Learning Outcomes

Demonstrate Stack operation using Array.

IV Laboratory Learning Outcome

Perform Operations on the Stack using the Array.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

A stack is an ordered list in which insertion and deletion are done at one end, called top. The last element inserted is the first one to be deleted. Hence, it is called the Last in First out (LIFO) or First in Last out (FILO) list. Two major operations through which changes can be made are PUSH and POP. When an element is inserted in a stack, the concept is called push, and when an element is removed from the stack, the concept is called pop. Trying to pop out an empty stack is called underflow and trying to push an element in a full stack is called overflow.

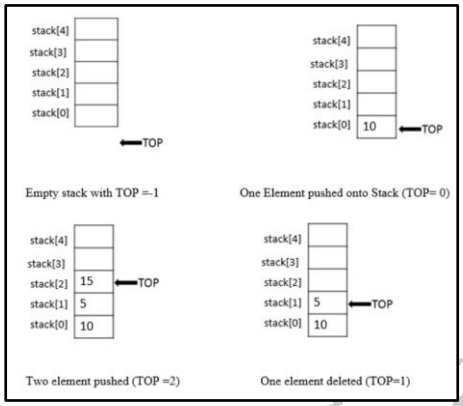


Figure 16.1 Stack PUSH and POP operations

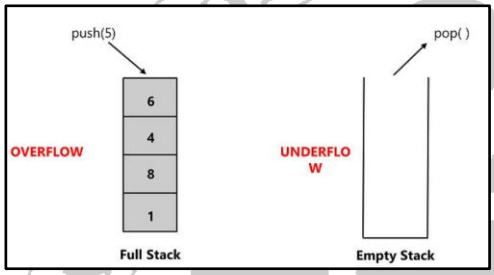


Figure 16.2 Stack Overflow and Underflow

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Static memory allocation using fixed memory size
4. Follow safety practices.

XII Result (Output of Program)

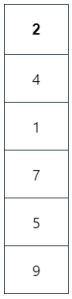
XIII Conclusion

XIV Practical Related Questions

1. Write a C program to perform following operations on Empty stack:
    a. PUSH (10), PUSH (20), POP, PUSH (10), PUSH (20), POP, POP, POP, PUSH (20), POP.
2. Write a C program to reverse a String using Stack.
3. Write a C program to check whether the given string is palindrome or not using stack.

XV Exercise

1. Enlist stack applications.
2. Consider the following stack and observe how many operations are required to delete smallest element from given stack.



[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

https://www.scholarhat.com/tutorial/datastructures/binary-search-tree-in-data-structures

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.17: * Write a 'C' Program to perform PUSH and POP Operations on Stack using Linked List.

I Practical Significance

Stacks are a fundamental data structure in computer science and have numerous practical applications based on their LIFO (Last In, First Out) property. Hence Stack is Linear Data Structure where insertion and deletion take place only from one end called as Stack Top.

II Industry/ Employer Expected Outcome

Through this practical student should,

1. Understand linked list structure
2. Represent stack in memory using Linked List.
3. Write an algorithm to perform PUSH and POP operations on Stack using Linked List.
4. Write simple C program to implement stack operation using Linked List.
5. Save/Compile/ Debug/ the C program.
6. Check for the desired output.

III Course Level Learning Outcomes

Demonstrate Stack operations using Linked List.

IV Laboratory Learning Outcome

Perform Operations on the Stack using a Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

A stack is an ordered list in which insertion and deletion are done at one end, called top. The last element inserted is the first one to be deleted. Hence, it is called the Last in First out (LIFO) or First in Last out (FILO) list.

Stack using Linked List:

Push operation: The first thing we need before pushing an element is to create a new node. Check if the stack is not already full. Now, we follow the same concept we learnt while inserting an element at the head or at the index 0 in a linked list. Just set the address of the current top in the next member of the new node, and update the top element with this new node.

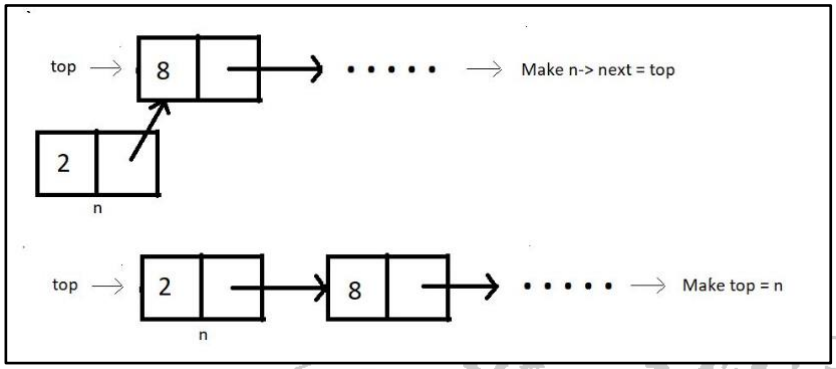


Figure 17.1 Stack Using Linked List- (PUSH operation)

POP operation:

First thing is to check if the stack is not already empty Now, we follow the same concept we learnt while deleting an element at the head or at the index 0 in a linked list. Just update the top pointer with the next node, skipping the current top.

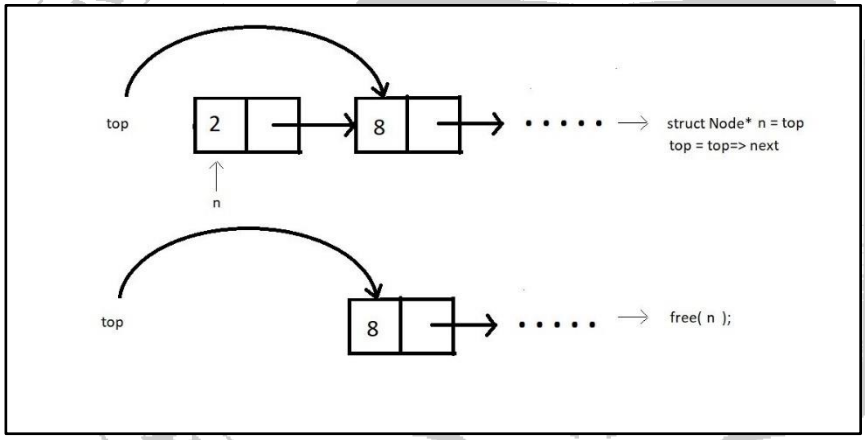


Figure 17.2 Stack Using Linked List- POP operation

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of Dynamic memory allocation to store Stack data
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to perform following operations on Stack as Linked List.

   PUSH (10), PUSH (20), POP, PUSH (10), PUSH (20), POP, PUSH (20), POP.

2. What is the time complexity of the push, pop, and peek operations in your implementation?

XV Exercise

1. What are the advantages of using a linked list to implement a stack?
2. Give limitations of using Linked List based Stack.

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

https://www.scholarhat.com/tutorial/datastructures/binary-search-tree-in-data-structures

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.18: * Write a 'C' program to perform multiplication of two numbers using recursion.

I Practical Significance

In essence, recursion is a fundamental concept in computer science and mathematics that provides a method for solving problems by breaking them down into smaller, more manageable sub-problems. Particularly in Data Structure, it is a powerful tool to work with recursive data structures like trees and graphs. For example, tree traversals (in-order, pre-order, post-order) are most easily implemented using recursion. Similarly, many graph algorithms, such as depth-first search (DFS), are naturally expressed recursively. Some classical algorithms like Merge Sort, Quick Sort use a recursive divide-and-conquer strategy. Hence by storing the results of subproblems (memorization), recursive algorithms can be optimized to avoid redundant computations.

II Industry/ Employer Expected Outcome

Through this practical student will,

1. Understand the concept of recursion.
2. Write an algorithm to perform perform given task using Recursion.
3. Implement C program for the designed algorithm using concept of Recursion.
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Demonstrate multiplication of two numbers using recursion.

IV Laboratory Learning Outcome

Perform Operations on the Stack using a Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Recursion:

In programming, a recursive function is a function that calls itself in order to break down the problem into more manageable sub-problems.

Example:

Multiplication using Recursion

In high-level languages, there is the multiplication operators available to directly perform multiplication. However, the multiplication is actually a repetitive addition. So the result of A * B is repetitive addition of A, B number of times, or we can say repetitive addition of B, A number of times. Whenever there is a repetition, we can do this using recursion.

Logic:

1. Get the inputs from the user and store it in the variables x and y,
2. The function product is used to calculate the product of given numbers,
3. A nested if-else statement is used to check x is less than y,
4. if the condition is satisfied, add the value of x with the value, else execute the else-if condition,
5. check the value of y is not equal to 0,
6. If the condition statement is true, execute the statement, or if the condition is not satisfied execute else block,
7. Print the product of two numbers.

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Implement a C program to print Fibonacci series using recursion
2. Write a program in C to calculate the sum of numbers from 1 to n using recursion.

XV Exercise

1. Define a term recursion and give its advantages.
2. Describe how stack is used in recursion with example.

[Space for Answer]

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://pwskills.com/blog/recursion-in-c/
2. https://www.shiksha.com/online-courses/articles/types-of-recursion-in-c/
3. https://www.w3resource.com/c-programming-exercises/recursion/index.php

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.19: Write a 'C' program to print given string in reverse using recursion.

I Practical Significance

Recursion is a powerful programming technique where a function calls itself in order to solve smaller instances of the same problem. This approach can be particularly useful for certain types of string operations.

II Industry/ Employer Expected Outcome

Through this practical student should,

1. Understand the concept of recursion.
2. Write an algorithm to perform given task using Recursion.
3. Implement C program for the designed algorithm using concept of Recursion.
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Demonstrate multiplication of two numbers using recursion.

IV Laboratory Learning Outcome

Perform Operations on the Stack using a Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Reverse a string using recursive function:

The recursive function takes a string as input. The function starts by declaring two static or global variables, i and revstr. The variable i is used to keep track of the index of the reversed string and is initialized to 0. The variable revstr is used to store the reversed string.

The function checks if the first character of the input string is not null. If it is not null, the function calls itself recursively with the input string pointer incremented by 1 (i.e., the address of the second character). This recursively moves through the string until it reaches the end.

For each recursive call, the function stores the current character at the i-th index of the 'revstr' array and increments the i variable. Finally, when the first character is null, the function returns the revstr array which contains the reversed string.

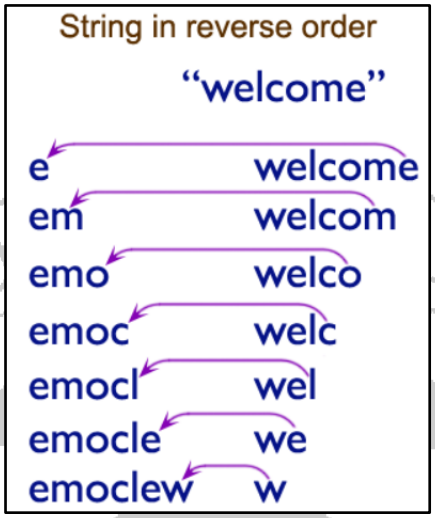


Figure 19.1: Reverse of a String using recursive function

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of recursive function call
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

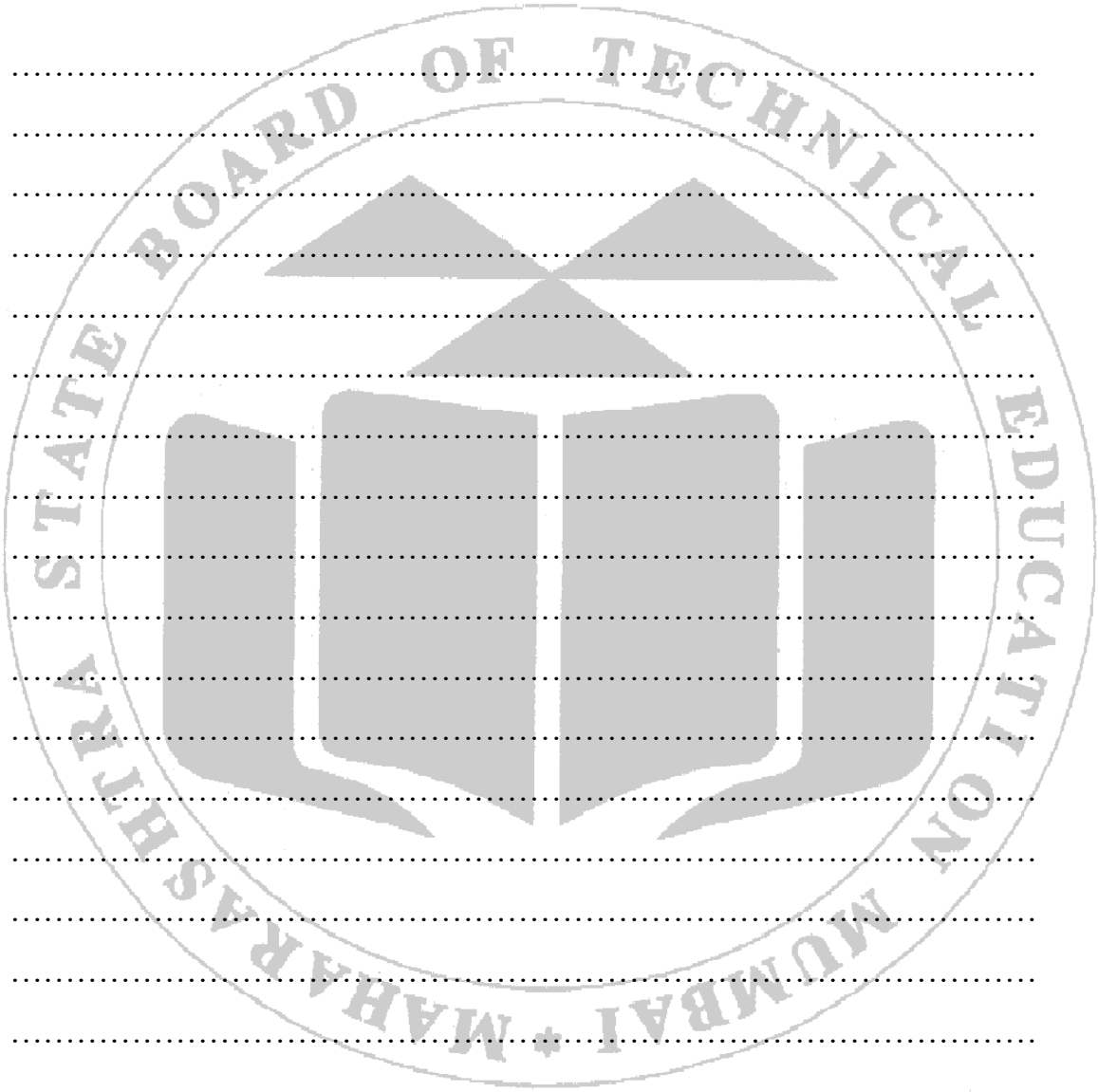
XIV Practical Related Questions

1. Write a C program to check whether a given string is a palindrome or not using recursion.
2. Write a program in C to copy one string to another using recursion.

XV Exercise

1. Differentiate between recursion and iteration
2. Give recursive function to calculate length of string.

[Space for Answer]



……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

https://www.w3resource.com/c-programming-exercises/recursion/index.php

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.20: Write a 'C' program to create a Singly Linked List and traverse in reverse order using recursion.

I Practical Significance

Linear data structure, particularly Singly Linked List is also known as One-way header list. It can be traversed only in one direction from first node to last node of linked list. In order to traverse it in reverse order we have two approaches- one is using stack and other is using recursion. Although the first approach works well but it requires extra space, hence to avoid it we can use recursive function call to traverse linked list in reverse order.

II Industry/ Employer Expected Outcome

Through this practical student should,

1. Recall and understand concepts of linked list and stack.
2. Analyze and apply appropriate method to solve above given statement.
3. Write an algorithm to traverse Linked List using Recursion.
4. Implement C program to traverse Linked List using Recursion.
5. Save/Compile/ Debug/ the C program.
6. Check for the desired output.

III Course Level Learning Outcomes

Demonstrate multiplication of two numbers using recursion.

IV Laboratory Learning Outcome

Perform Operations on the Stack using a Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

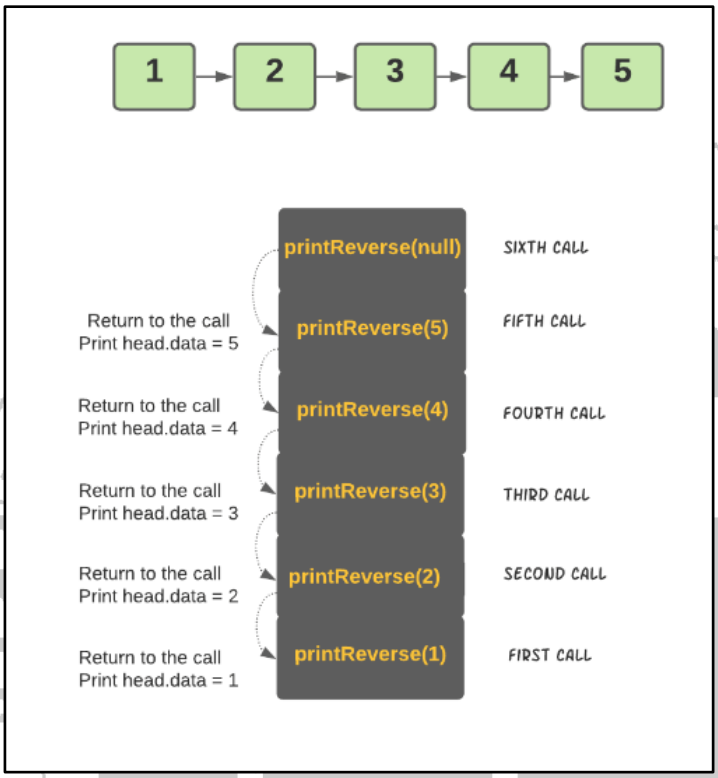
VI Relevant Theoretical Background

Traverse Linked list in reverse order using recursion

The idea is to traverse the linked list recursively. Since Recursion preserves a stack of calls during the execution.

In this recursive function call, If the linked list is empty, i.e., the head is null, simply return from the function. Otherwise, make a recursive call by passing the next node of a list as an argument. As we reach to terminating condition backtrack to previous function calls in order to Print the current node data and hence achieving reverse of a liked list.

Consider the below representation to understand the algorithm:



OUTPUT: 5 4 3 2 1

Figure 20.1: Traversing a Linked List using recursion

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of recursive function call
4. Follow safety practices

XII Result (Output of Program)

XIII Conclusion

XIV Related Questions

1. Write a C program to traverse linked list using recursion
2. Write a program in C to calculate length of linked list using recursion.

XV Exercise

1. Give recursive function to create a copy of linked list using recursion.

[Space for Answer]

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.sanfoundry.com/c-program-linked-list-reverse-using-recursion/#google_vignette
2. https://www.geeksforgeeks.org/c-program-to-create-copy-of-a-singly-linked-list-using-recursion/
3. https://www.techcrashcourse.com/2016/06/program-to-reverse-linked-list-iteration-recursion.html

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.21: * Write a 'C' Program to perform INSERT and DELETE Operations on Linear Queue using an Array.

I Practical Significance

The ability to manage ordered sequences of elements and handling of tasks and resources efficiently makes the use of queue indispensable in various applications across different fields, from computing and networking to business operations and service management. It is a linear data structure works in First in First (FIFO) Order with two operational ends Front End to add new element and Rear End to delete or process new element.

II Industry/ Employer Expected Outcome

Through this practical student should,

1. Understand basic operations to be performed on Linear Queue.
2. Write an algorithm to perform basic operations on Linear Queue.
3. Implement C program to Queue as ADT using Array.
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Implement basic operations on linear queue using Array.

IV Laboratory Learning Outcome

Perform Operations on Linear Queue using Array.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Linear Queue using Array

Queue data structure is a linear data structure in which the operations are performed based on FIFO principle.

Queue is an object (an abstract data structure - ADT) that allows the following operations:

Enqueue: Add an element to the rear end of the queue.

Dequeue: Remove an element from the front of the queue.

IsEmpty: Check if the queue is empty.

IsFull: Check if the queue is full.

Peek: Get the value of the front of the queue without removing it.

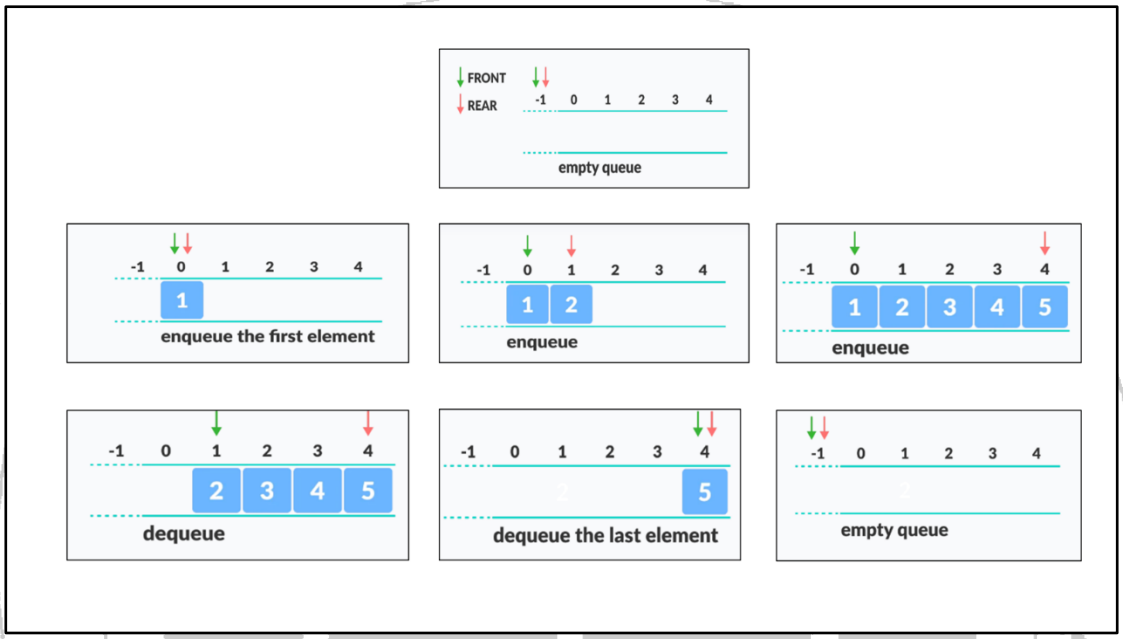


Figure 21.1: Insertion (enqueue) and Deletion (dequeue) operations on Linear Queue

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of recursive function call
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to implement a linear queue with operations for enqueue (15), enqueue (48), enqueue (69), dequeue, enqueue (12), enqueue (23), dequeue, dequeue checking if the queue is empty or full, and displaying the current queue.
2. Write a C program to demonstrate Queue Underflow and Queue Overflow.

XV Exercise

1. Describe Queue as Abstract Data Type (ADT).
2. Give applications of Queue in different domains.

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION MUMBAI

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.programiz.com/dsa/queue
2. https://ds1-iiith.vlabs.ac.in/exp/stacks-queues/queues/index.html
3. https://ds1-iiith.vlabs.ac.in/exp/stacks-queues/queues/queuesarrays.html

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.22: * Write a 'C' Program to perform INSERT and DELETE operations on Linear Queue using a Linked List.

I Practical Significance

The major problem with the queue implemented using an array is, t will work for an only fixed number of data values. That means, the amount of data must be specified at the beginning itself. Queue using an array is not suitable when we don't know the size of data which we are going to use. A queue data structure can be implemented using a linked list data structure. The queue which is implemented using a linked list can work with the variable size of data. Implementing a queue using a linked list provides dynamic size management, efficient insertions and deletions, avoidance of overflow issues, better memory utilization, and flexibility. These characteristics make linked list-based queues suitable for various real-world applications, especially where dynamic and efficient data handling is critical.

II Industry/ Employer Expected Outcome

Through this practical student should,

1. Understand memory representation of queue using Linked List.
2. Differentiate between static and dynamic memory allocation.
3. Implement C program to Queue as ADT using Linked List.
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Implement basic operations on linear queue using Linked List.

IV Laboratory Learning Outcome

Perform Operations on Linear Queue using Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Linear Queue using Linked List

A linked queue consists of two pointers, i.e., the front pointer and the rear pointer. The front pointer stores the address of the first element of the queue, and the rear pointer stores the address of the last element of the queue.

Insertion is performed at the rear end, whereas deletion is performed at the front end of the queue. If front and rear both point to NULL, it signifies that the queue is empty.

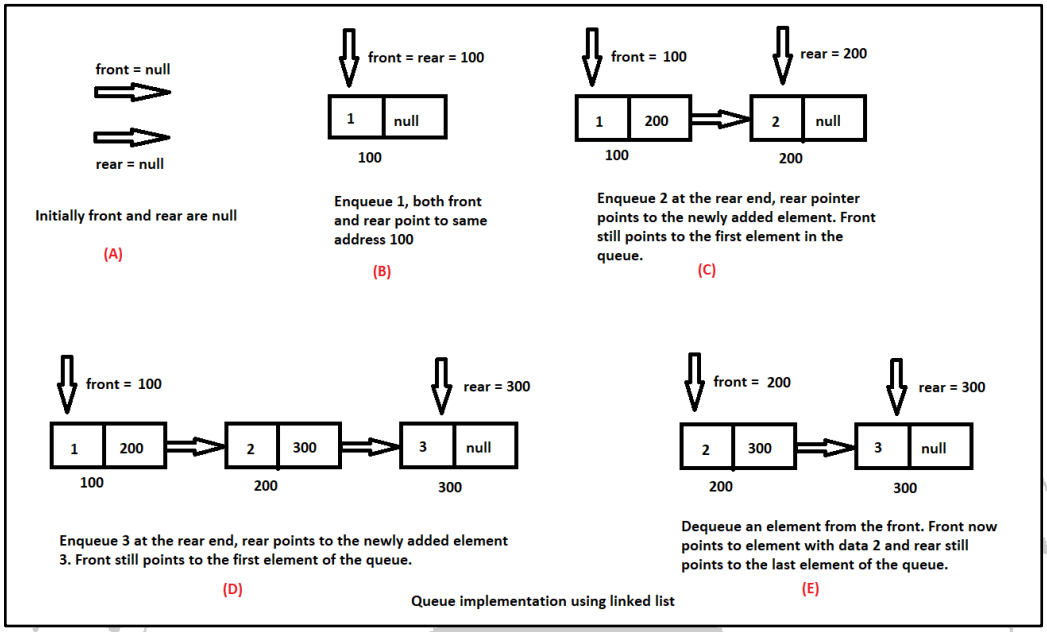


Figure 22.1: Insertion (enqueue) and Deletion (dequeue) operations on Linear Queue using Linked List

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of recursive function call
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to implement a linear queue using linked list with operations for enqueue (15), enqueue (48), enqueue (69), dequeue, enqueue (12), enqueue (23), dequeue, dequeue.
2. Define queue overflow and underflow in case of linked list based implementation.

XV Exercise

1. Differentiate between stack and queue.
2. Give advantages of using linked list in implementation of Queue.

[Space for Answer]

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.log2base2.com/data-structures/queue/queue-using-linked-list.html
2. https://ds1-iiith.vlabs.ac.in/exp/stacks-queues/queues/queueslinkedlist.html
3. https://www.scaler.com/topics/c/implementation-of-queue-using-linked-list/
4. https://www.log2base2.com/data-structures/queue/queue-using-linked-list.html

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.23: * Write a 'C' Program to perform INSERT and DELETE operations on Circular Queue using an Array

I Practical Significance

Circular queues offer significant advantages over linear queues, particularly in terms of space efficiency, fixed size implementation, efficient operations, and suitability for real-time systems. They are widely used in various applications where predictable and efficient data handling is crucial.

II Industry/ Employer Expected Outcome

Through this practical student should,

1. Understand working of circular queue using Array
2. Differentiate between Linear Queue and Circular Queue.
3. Implement basic operations to be performed on Circular Queue using Array in C.
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Implement basic operations on Circular queue using Linked List.

IV Laboratory Learning Outcome

Perform Operations on Circular Queue using Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Circular Queue using Array:

In normal queue Implementation, after a few enQueue and deQueue operations, there are empty spaces(positions) left out in the queue that are not being used till both the front and rear pointers are reset to -1.

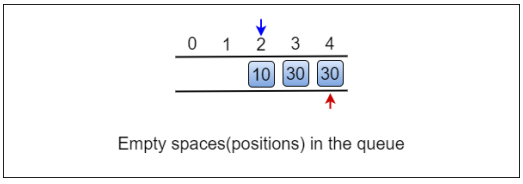


Figure 23.1 Wastage of memory in normal queue implementation.

Hence to utilize the empty spaces for adding new items in the queue we can use a circular queue implementation in which both pointers front and rear are circular increments that mean after reaching the end of the queue the pointers come back to the start of the queue.

Circular Queue empty condition:

If FRONT == -1 then the queue is empty.

Circular Queue full condition:

FRONT == 0 and REAR == (SIZE – 1): - Queue full and

FRONT == (REAR + 1): - Queue full

OR

FRONT == (REAR+1) % SIZE: - Queue full

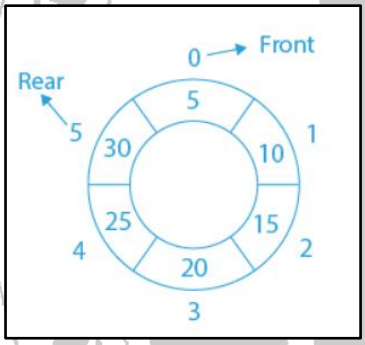


Figure 23.1: Enqueue operation (Circular Queue is Full)

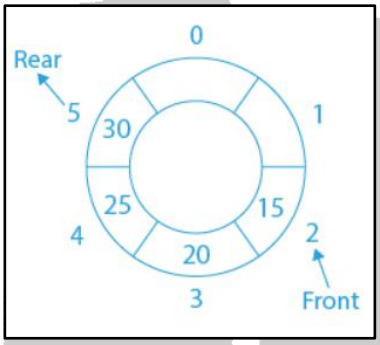


Figure 23.2: Dequeue operation (2 values deleted)

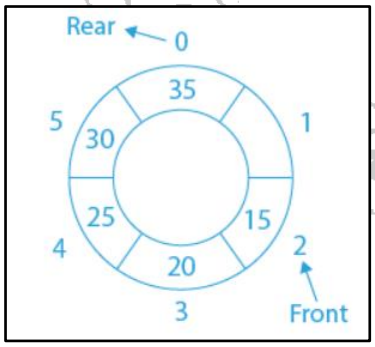


Figure 23.3: Enqueue operation on Circular queue

VII Algorithm

VIII Flow Chart

IX C Program Code

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION MUMBAI

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of recursive function call
4. Follow safety practice

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to implement a Circular queue using Array with operations for enqueue (15), enqueue (48), enqueue (69), dequeue, enqueue (12), enqueue (23), dequeue, dequeue.
2. Write a C program to implement a Queue overflow and underflow operations on Circular Queue using Array.

XV Exercise

1. Differentiate between Linear Queue and Circular Queue.
2. Give advantages of Circular Queue.

[Space for Answer]

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.dremendo.com/c-programming-tutorial/c-circular-queue
2. https://www.prepbytes.com/blog/queues/advantages-of-circular-queue-over-linear-queue/#:~:text=The%20circular%20queue%20has%20several,queue%20only%20supports%20FIFO%20structure.
3. https://levelup.gitconnected.com/visualize-design-and-analyse-the-circular-queue-data-structure-d3fe78b12b4b

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved (LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related (30)	**Product Related (20)**	**Total (50)**	

Practical No.24: Write a 'C' Program to perform INSERT and DELETE operations on Circular Queue using Linked List

I Practical Significance

Unlike arrays, linked lists do not have a fixed size. This allows the circular queue to grow and shrink as needed without wasting memory or encountering overflow issues. Using Linked List Memory is allocated dynamically, ensuring that only the necessary amount of memory is used, reducing waste. These advantages make circular queues implemented with linked lists a good choice, especially in scenarios where the maximum number of elements is not known in advance or can vary significantly over time.

II Industry/ Employer Expected Outcome

Through this practical student should,

1. Understand working of circular queue using Linked List.
2. Differentiate between Circular Queue array implementation and Linked list implementation.
3. Implement basic operations to be performed on Circular Queue using Linked List in C.
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Implement basic operations on Circular queue using Linked List.

IV Laboratory Learning Outcome

Perform Operations on Circular Queue using Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Circular Queue using Linked List:

Circular queues implemented using linked lists are flexible as the size can dynamically adjust. The front and rear pointers are used to manage the linked list, ensuring a circular structure. Enqueueing involves adding a new node at the end of the list, while memory allocation is checked to avoid overflow.

enqueue(data)

Create a struct node type node.

Insert the given data in the new node data section and NULL in address section.

If Queue is empty then initialize front and rear from new node.

Queue is not empty then initialize rear next and rear from new node.

New node next initialize from front

dequeue()

Check if queue is empty or not.

If queue is empty then dequeue is not possible.

Else Initialize temp from front.

If front is equal to the rear then initialize front and rear from null.

Print data of temp and free temp memory.

If there is more than one node in Queue then make front next to front hen initialize rear next from front.

Print temp and free temp.

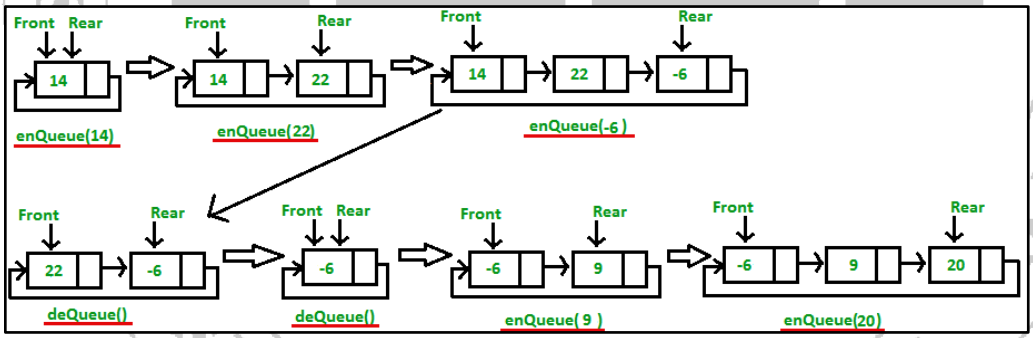


Figure 24.1: Circular Queue Operation using Linked List.

VI Algorithm

VII Flow Chart

VIII C Program Code

IX Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

X Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of linked list
4. Follow safety practices.

XI Result (Output of Program)

XII Conclusion

XIII Practical Related Questions

1. Write a C program to implement enqueue operation on Circular queue using Linked List.
2. Write a C program to implement dequeue operation on Circular queue using Linked List.

XIV Exercise

1. How do you initialize a circular queue using a linked list?.
2. What are some real-world applications of circular queues?

[Space for Answer]

……

……

……

……

……

XV References / Suggestions for further Reading Software/Learning Websites

1. https://www.simplilearn.com/tutorials/data-structure-tutorial/circular-queue-in-data-structure
2. https://www.prepbytes.com/blog/queues/circular-queue-using-linked-list/
3. https://prepinsta.com/c-program/circular-queue-using-linked-list/#:~:text=Implementation%20of%20Circular%20Queue%20using,it%20is%20easy%20to%20create.

XVI Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved (LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related (30)	**Product Related (20)**	**Total (50)**	

Practical No.25: Write a 'C' Program to Create a Priority Queue using a Linked List.

I Practical Significance

A priority queue is a specialized data structure that operates similarly to a regular queue, but with an added feature: each element is associated with a priority. Elements are dequeued not based on their order of arrival, but according to their priority. It can be used in different applications like process scheduling, networking, simulations etc.

II Industry/ Employer Expected Outcome

Through this practical student should,

1. Understand working of Priority queue.
2. Implement basic operations to be performed on Circular Queue using Linked List in C.
3. Save/Compile/ Debug/ the C program.
4. Check for the desired output.

III Course Level Learning Outcomes

Develop basic operations like create and display Priority Queue using Linked List.

IV Laboratory Learning Outcome

Implement Priority Queue using Linked List.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Priority Queue using Linked List:

Priority queue is an abstract data type, It is a type of queue in which each element has a priority assigned to it. The priority of the element determines the order in which elements are removed from the priority queue. In the priority queue, all the elements are arranged either in ascending order or descending order.

Priority Queue Properties:

In the priority queue, every element has priority assigned to it.

The element with highest priority processed first.

If two elements are having the same priority then they are served according to their order in the queue.

Operations:

Push: Pushing/inserting a new item in the priority queue at a position in accordance with its priority

Pop: Popping/removing a new item in the priority queue in accordance with its priority

Peek: Displaying the topmost priority item of the queue

Note:

A lower priority number for an item means higher priority.

Example: (item1, 2) and (item2, 5). Here is item one is higher priority as it has lower priority number,

Syntax for Priority Queue as Linked List

```
struct Node
{
int data;
int priority; // lower value means higher priority
struct Node* next;
};
```

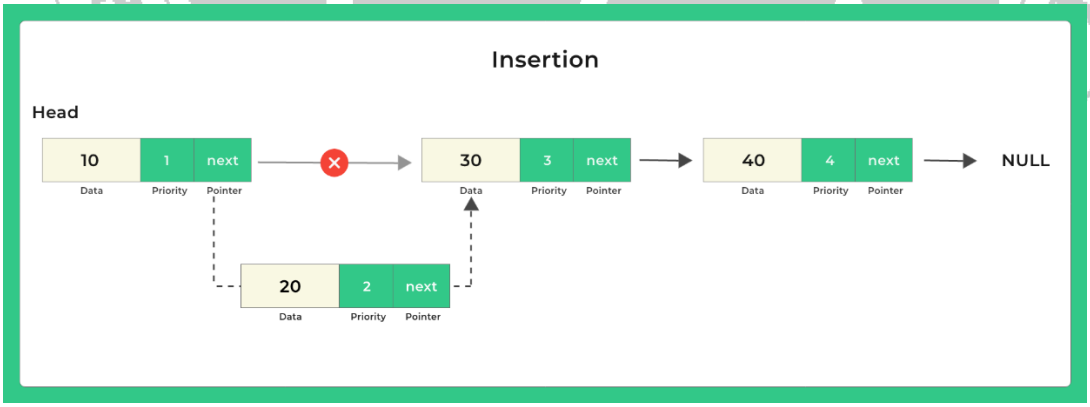


Figure 25.1: Insertion in Priority Queue

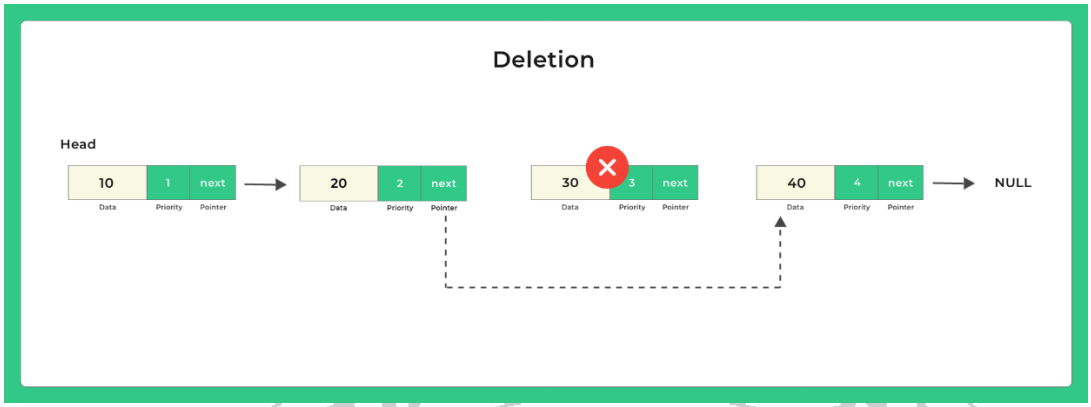


Figure 25.1: Deletion From Priority Queue

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure correct use of linked list in implementing priority queue.
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to implement enqueue operation on Priority queue using Linked List.
2. Write a C program to implement dequeue operation on Priority queue using Linked List.

XV Exercise

1. Enlist application of priority queue?
2. How linked list is beneficial to implement linked list over an array?

[Space for Answer]

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.prepbytes.com/blog/linked-list/priority-queue-using-linked-list/
2. https://prepinsta.com/c-program/implementation-of-priority-queue-using-linked-list/
3. https://www.javatpoint.com/priority-queue-using-linked-list
4. https://www.scholarhat.com/tutorial/datastructures/priority-queue-in-data-structures

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved (LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related (30)	**Product Related (20)**	**Total (50)**	

Practical No.26: * Write a 'C' Program to Implement BST (Binary Search Tree) and Traverse in In-Order.

I Practical Significance

Binary trees are a fundamental and versatile data structure with numerous applications in computer science and software engineering. Their efficient representation of hierarchical data, support for various traversal methods, and foundational role in more complex structures make them an essential concept to understand and utilize effectively. Binary trees support various traversal methods (Pre-order, In-order, Post-order), which are useful for different applications, such as expression evaluation, syntax parsing, and more.

II Industry/ Employer Expected Outcome

This practical is expected to develop the following skills in you

1. Write an algorithm to create and traverse (In-order) a BST.
2. Write simple C program to traverse a BST in In-order.
3. Save/Compile/ Debug/ the C program.
4. Check for the desired output.

III Course Level Learning Outcomes

Create Binary Search Tree and Demonstrate In-order Traversal on constructed BST.

IV Laboratory Learning Outcome

Implement Binary Search Tree and perform In-Order Traversal.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

1. Binary Search Tree (BST):

A Binary Search Tree (BST) is a binary tree in which each node has the following properties:

Each node in a BST contains three fields- A key (or value), a reference (or pointer) to the left child and a reference (or pointer) to the right child.

For each node-All keys in the left subtree are less than the key of the node. All keys in the right subtree are greater than the key of the node.

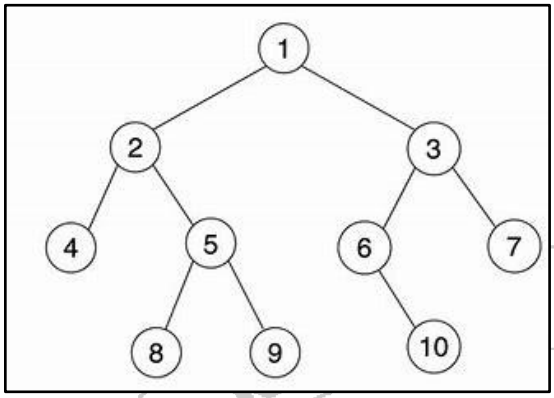


Figure 26.1 Binary Search Tree

2. **Binary Search Tree Traversal Methods:**
Binary Search Trees (BSTs) support several traversal methods that allow visiting all the nodes in a specific order. The primary traversal methods are:

In-order Traversal

In an In-order traversal, nodes are visited in ascending order. For a BST, this means visiting the left subtree, then the current node, and finally the right subtree.

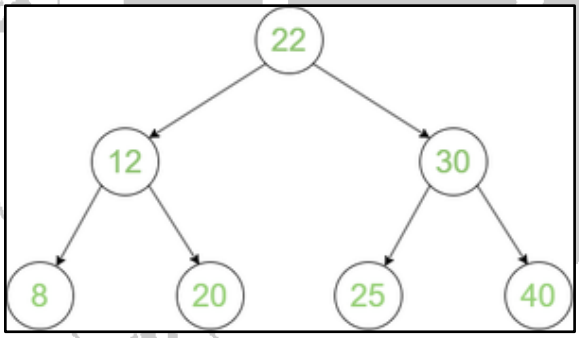


Figure 26.2: BST- In-order Traversal

For the above BST In-order Tree Traversal - 8 12 20 22 25 30 40

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (;).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of dynamic memory allocation
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to create Binary Search Tree and display it.
2. Write a C program to search a key element in Binary Search Tree.

XV Exercise

1. Create Binary Search Tree of alphabet A to G.
2. Write In-order Traversal for above Tree.

[Space for Answer]

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.scholarhat.com/tutorial/datastructures/binary-search-tree-in-data-structures

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	

Practical No.27: Write a 'C' Program to Traverse BST in Preorder, and Post-Order.

I Practical Significance

In the fields of computer science and software engineering Binary Search Trees (BSTs) enable efficient searching of elements. Understanding BSTs provides foundation for advanced Data Structures like AVL trees, Red-Black Trees, B-trees etc. Binary Search Trees is not only fundamental for mastering data structures and algorithms but also has numerous practical applications in real-world software development and problem-solving like file directory structures, Networking etc.

II Industry/ Employer Expected Outcome

Through execution of this practical student should

1. Understand methods to traverse Binary Trees.
2. Write an algorithm to traverse (pre-order and post-order traversal) for BST
3. Write simple C program to demonstrate pre-order and post-order traversal
4. Save/Compile/ Debug/ the C program.
5. Check for the desired output.

III Course Level Learning Outcomes

Create Binary Search Tree and Demonstrate pre-order and post-order Traversal on constructed BST.

IV Laboratory Learning Outcome

Implement Tree Traversal Operations.

V Relevant Affective domain related Outcome(s)

1. Follow safety practices.
2. Follow ethical practices.

VI Relevant Theoretical Background

Binary Search Tree Traversal Methods:

Binary Search Trees (BSTs) support several traversal methods that allow visiting all the nodes in a specific order. The primary traversal methods are:

Pre-order Traversal

In a preorder traversal, nodes are visited in the order: current node, left subtree, right subtree. This is useful for creating a copy of the tree.

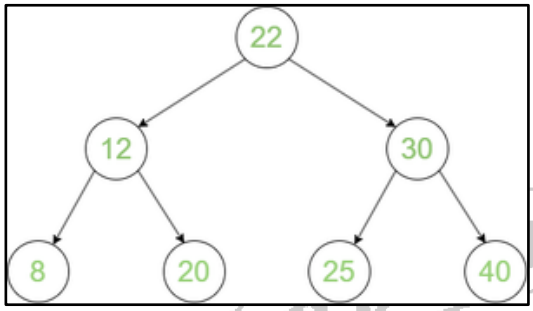


Figure 27.1: Pre-order Tree Traversal – 22 12 8 20 30 25 40

Post-order Traversal

In a post-order traversal, nodes are visited in the order: left subtree, right subtree, current node. This is useful for deleting the tree.

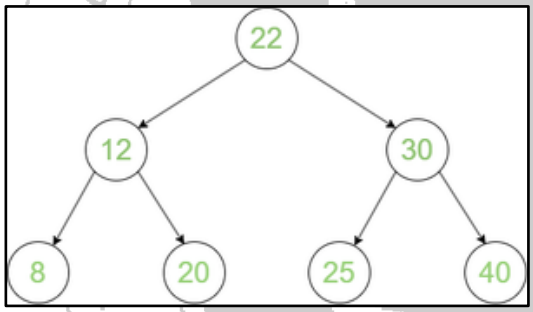


Figure 27.2: Post-order Tree Traversal – 8 20 12 25 40 30 22

VII Algorithm

VIII Flow Chart

IX C Program Code

X Resources required

Sr. No.	Equipment Name with broad specification	Relevant LLO Number
1	Computer System with all necessary Peripherals and Internet Connectivity 'C' Compiler / GCC Compiler/ Online 'C' Compiler All	ALL

XI Precautions to be Followed

1. Ensure that all C statements must end with a semicolon (**;**).
2. Use white spaces in c to describe blanks and tabs.
3. Ensure use of dynamic memory allocation
4. Follow safety practices.

XII Result (Output of Program)

XIII Conclusion

XIV Practical Related Questions

1. Write a C program to construct the binary search tree from following data. Elements: [1, 5, 7, 8, 10, 12] and traverse in pre-order.
2. Write a C program to construct the binary search tree from following data. Elements: [1, 5, 7, 8, 10, 12] and traverse in post-order.

XV Exercise

1. Create Expression tree for expression- (3 + ((4 * 5) - 2))
2. Write infix and postfix expression for the above constructed tree.

[Space for Answer]

……

……

……

……

XVI References / Suggestions for further Reading Software/Learning Websites

1. https://www.scholarhat.com/tutorial/datastructures/binary-search-tree-in-data-structures

XVII Assessment Scheme

Performance indicators		**Weightage**
Process related: 30 Marks		**60%**
1.	Debugging ability	20%
2.	Correctness of Program codes	30%
3.	Quality of output achieved(LLO mapped)	10%
Product related: 20 Marks		**40%**
1.	Completion and submission of practical in time	20%
2.	Answer to sample questions	20%
	Total 50 Marks	**100%**

Marks obtained			**Dated Sign of Teacher**
Process Related(30)	**Product Related(20)**	**Total(50)**	