ANJUMAN
COLLEGE OF ENGINEERING & TECHNOLOGY
(MANAGED BY : ANJUMAN HAMI-E-ISLAM, NAGPUR)

Roll No:_____

Name:_____

Sem:_____Section_____

Computer Science & Engineering Department

CERTIFICATE

Certified that this file is submitted by

Shri/Ku._____

Roll No._____a student of_____year of the course _____

_____ as a part of PRACTICAL/ORAL as prescribed by the Rashtrasant Tukadoji Maharaj Nagpur University for the subject_____ in the laboratory of _____during the academic year _____ and that I have instructed him/her for the said work, from time to time and I found him/her to be satisfactory progressive.

And that I have accessed the said work and I am satisfied that the same is up to that standard envisaged for the course.

Date:-

Signature & Name
of Subject Teacher

Signature & Name
of HOD

Anjuman College of Engineering and Technology

Vision

To be a centre of excellence for developing quality technocrats with moral and social ethics, to face the global challenges for the sustainable development of society.

Mission

To create conducive academic culture for learning and identifying career goals.

To provide quality technical education, research opportunities and imbibe entrepreneurship skills contributing to the socio-economic growth of the Nation.

To inculcate values and skills, that will empower our students towards development through technology.

Vision and Mission of the Department

Vision:

To achieve excellent standards of quality education in the field of computer science and engineering, aiming towards development of ethically strong technical experts contributing to the profession in the global society.

Mission:

To create outcome based education environment for learning and identifying career goals.

Provide latest tools in a learning ambience to enhance innovations, problem solving skills, leadership qualities team spirit and ethical responsibilities.

Inculcating awareness through innovative activities in the emerging areas of technology.

The graduates will have a strong foundation in mathematical, scientific and engineering fundamentals necessary to formulate, solve and analyze engineering problem in their career.

Graduates will be able to create and design computer support systems and impart knowledge and skills to analyze, design, test and implement various software applications.

Graduates will work productively as computer science engineers towards betterment of society exhibiting ethical qualities.

Program Specific Outcomes (PSOs)

Foundation of mathematical concepts: To use mathematical methodologies and techniques for computing and solving problem using suitable mathematical analysis, data structures, database and algorithms as per the requirement.

Foundation of Computer System: The capability and ability to interpret and understand the fundamental concepts and methodology of computer systems and programming. Students can understand the functionality of hardware and software aspects of computer systems, networks and security.

Foundations of Software development: The ability to grasp the software development lifecycle and methodologies of software system and project development.

PROGRAM: CSE	DEGREE: B.E
COURSE: Data Structures & Program Design	SEMESTER: IV CREDITS: 2
COURSE CODE: BE4S2P	COURSE TYPE: REGULAR
COURSE AREA/DOMAIN: Data Structures & Program Design	CONTACT HOURS: 2 hours/Week.
CORRESPONDING LAB COURSE CODE : BE4S2P	LAB COURSE NAME : : Data Structures & Program Design Lab

COURSE PRE-REQUISITES:

C.CODE	COURSE NAME	DESCRIPTION	SEM

LAB COURSE OBJECTIVES:

Given knowledge about basic and advanced data structures .

To explain them algorithms for performing various operations on these data structures.

Students will gain practical knowledge by writing and executing programs in C using various data structures such as arrays, linked lists, stacks, queues, trees, graphs and search trees.

COURSE OUTCOMES: Data Structures & Program Design

After completion of this course the students will be able -

SNO	DESCRIPTION	BLOOM'S TAXONOMY LEVEL
CO.1	***Implement*** and ***analyze*** different searching and sorting algorithms.	LEVEL 4 & 6
CO.2	***Develop*** ADT for Stack data structure and its applications.	LEVEL 3 & 6
CO.3	***Develop*** ADT for Queue data structure and its applications.	LEVEL 3& 6
CO.4	***Demonstrate*** ability to apply knowledge of dynamic data structures like linked-lists and ***Extend*** its applications.	LEVEL 2
CO.5	***Apply*** fundamentals of Tree data structures to implement Tree and problems including Tree traversals.	LEVEL 3
CO.6	***Explain*** implementation of Graph data structure and Graph traversals.	LEVEL 2 & 5

Lab Instructions:

Make entry in the Log Book as soon as you enter the Laboratory.

All the students should sit according to their Roll Numbers.

All the students are supposed to enter the terminal number in the Log Book.

Do not change the terminal on which you are working.

Strictly observe the instructions given by the Faculty / Lab. Instructor.

Take permission before entering in the lab and keep your belongings in the racks.

NO FOOD, DRINK, IN ANY FORM is allowed in the lab.

TURN OFF CELL PHONES! If you need to use it, please keep it in bags.

Avoid all horseplay in the laboratory. Do not misbehave in the computer laboratory. Work quietly.

Save often and keep your files organized.

Don’t change settings and surf safely.

Do not reboot, turn off, or move any workstation or PC.

Do not load any software on any lab computer (without prior permission of Faculty and Technical Support Personnel). Only Lab Operators and Technical Support Personnel are authorized to carry out these tasks.

Do not reconfigure the cabling/equipment without prior permission.

Do not play games on systems.

Turn off the machine once you are done using it.

Violation of the above rules and etiquette guidelines will result in disciplinary action.

Continuous Assessment Practical

Prog. No	NAME OF PROGRAM	Date	Sign	Remark
1	Write a C program for Linear search.			
2	Write a C program for Binary search.			
3	Write a C program for Bubble Sort.			
4	Write a C program for Selection Sort.			
5	Write a menu driven C program to implement the following :- a) Create a linked list. b) Insert an item in a linked list. c) Delete an item from the linked list.			
6	Write a C program to implement the following :- a) Push an element using stack. b) Pop an element using stack			
7	Write a C program to implement the following :- a)Insert an element to the queue. b)Delete an element from queue.			
8	Write a C program for tree traversal techniques.			
09	Write a C program to implement graphs.			
10	Write a C program to implement hashing.			
	ADDITIONAL PROGRAM			
11	Write a C program to implement stack & queue using linked representation.			
12	Write a C program to sort a list of string.			

CONTENTS

EXPERIMENT NO – 1

Aim: Write a C++ program- To search an element from given array using Linear Search.

Theory : Linear Search.

When data items are stored in a collection such as a list, we say that they have a linear or sequential relationship. Each data item is stored in a position relative to the others. In Python lists, these relative positions are the index values of the individual items. Since these index values are ordered, it is possible for us to visit them in sequence. This process gives rise to our first searching technique, the **sequential search or linear search**.

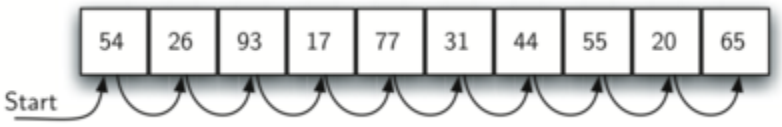


Figure shows how this search works. Starting at the first item in the list, we simply move from item to item, following the underlying sequential ordering until we either find what we are looking for or run out of items. If we run out of items, we have discovered that the item we were searching for was not present.

ALGORITHM:

Linear Search (Array A, Value x)

Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit

PROGRAM:

OUTPUT:

Conclusion: Thus we, have executed a C program To search an element from given array using Linear Search successfully.

Viva Voce Question

1. What is data structure?

2. What are various data structures available?

3. What is searching? Explain different types of searching.

4. Write Complexity of linear search.

Signature of Subject Teacher

EXPERIMENT NO – 2

Aim: Write a C program for Binary search.

Theory : Binary Search

Binary search is a fast search algorithm with run-time complexity of O(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly the data collection should be in sorted form.

Binary search search a particular item by comparing the middle most item of the collection. If match occurs then index of item is returned. If middle item is greater than item then item is searched in sub-array to the right of the middle item otherwise item is search in sub-array to the left of the middle item. This process continues on sub-array as well until the size of subarray reduces to zero.

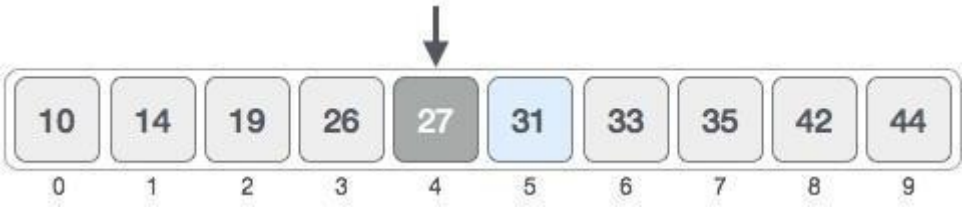
How binary search works?

For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with an pictorial example. The below given is our sorted array and assume that we need to search location of value 31 using binary search.

`First, we shall determine the half of the array by`

`using this formula – mid = low + (high – low) / 2`

Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5). So 4 is the mid of array.



Now we compare the value stored at location 4, with the value being searched i.e. 31. We find that value at location 4 is 27, which is not a match. Because value is greater than 27 and we have a sorted array so we also know that target value must be in upper portion of the array.

We change our low to mid + 1 and find the new mid value again.

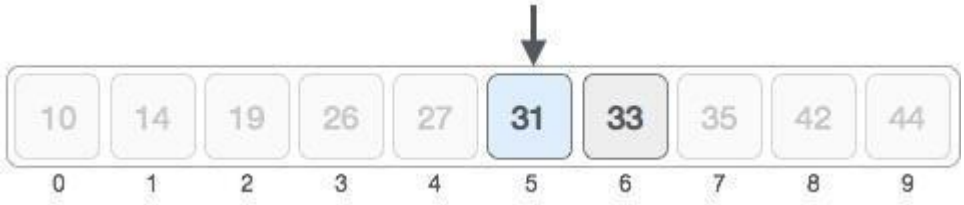
low = mid + 1

mid = low + (high - low) / 2

Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.

The value stored at location 7 is not a match, rather it is less that what we are looking for. So the value must be in lower part from this location.

So we calculate the mid again. This time it is 5.



We compare the value stored ad location 5 with our target value. We find that it is a match.

We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

ALGORITHM:

Algorithm is quite simple. It can be done either recursively or iteratively:

1. get the middle element;
2. if the middle element equals to the searched value, the algorithm stops;
3. otherwise, two cases are possible:
    - searched value is less, than the middle element. In this case, go to the step 1 for the part of the array, before middle element.
    - searched value is greater, than the middle element. In this case, go to the step 1 for the part of the array, after middle element.

Now we should define, when iterations should stop. First case is when searched element is found. Second one is when subarray has no elements. In this case, we can conclude, that searched value doesn't present in the array

PROGRAM:

OUTPUT:

Conclusion: Thus we, have executed a C program To search an element from given array using Binary Search successfully.

Viva Voce Question

1. What is binary search?

2. What are the advantages of binary search?

3. Differentiate between linear search and binary search.

4. Write complexity of binary search.

Signature of Subject Teacher

EXPERIMENT NO – 3

Aim: Write a C program for Bubble Sort.

Theory: Theory : Bubble Sort

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison based algorithm in which each pair of adjacent elements is compared and elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where n are no. of items.

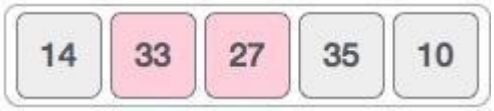
How bubble sort works?

We take an unsorted array for our example. Bubble sort take $O(n^2)$ time so we're keeping short and precise.

Bubble sort starts with very first two elements, comparing them to check which one is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this –

Next we compare 33 and 35. We find that both are in already sorted positions.

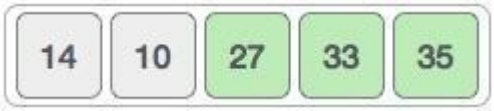
Then we move to next two values, 35 and 10.

We know than 10 is smaller 35. Hence they are not sorted.

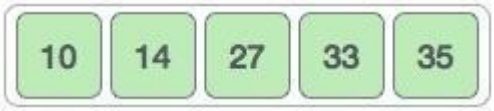
We swap these values. We find that we reach at the end of the array. After one iteration the array should look like this −

To be precise, we are now showing that how array should look like after each iteration. After second iteration, it should look like this −

Notice that after each iteration, at least one value moves at the end.



And when there's no swap required, bubble sorts learns that array is completely sorted.



Now we should look into some practical aspects of bubble sort.

ALGORITHM:

Bubble Sort Algorithm

Following are the steps involved in bubble sort(for sorting a given array in ascending order):

1. Starting with the first element(index = 0), compare the current element with the next element of the array.
2. If the current element is greater than the next element of the array, swap them.
3. If the current element is less than the next element, move to the next element. Repeat Step 1.

PROGRAM:

OUTPUT:

Conclusion: Thus we, have executed a C program to arrange an elements in ascending order by using Bubble sort successfully.

Viva Voce Question

1. What is sorting?

2. What is bubble sort? What is the complexity of bubble sort?

3. What are the basic criteria for selecting appropriate algorithm.

Signature of Subject Teacher

EXPERIMENT NO – 4

Aim: Write a C program for Selection Sort.

Theory : Selection Sort

Selection sort is a simple sorting algorithm. This sorting algorithm is a in-place comparison based algorithm in which the list is divided into two parts, sorted part at left end and unsorted part at right end. Initially sorted part is empty and unsorted part is entire list.

Smallest element is selected from the unsorted array and swapped with the leftmost element and that element becomes part of sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2) where n are no. of items.

How selection sort works?

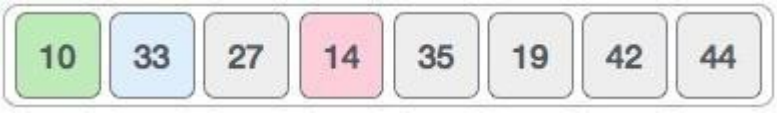
We take the below depicted array for our example.

For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in linear manner.

We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.



ıed at the the beginning in the sorted manner.

The same process is applied on the rest of the items in the array. We shall see an pictorial depiction of entire sorting process

ALGORITHM:

Step 1 – Set MIN to location 0
Step 2 – Search the minimum element in the list Step 3 – Swap with value at location MIN
Step 4 – Increment MIN to point to next element Step 5 – Repeat until list is sorted

PROGRAM:

OUTPUT:

Conclusion: Thus we, have executed a C program to arrange an elements in ascending order by using Selection sort successfully.

Viva Voce Question

1. What is selection sort? Explain complexity of selection sort .

2. How insertion sort and bubble sort are different?

What is the advantage of selection sort over other sorting techniques?.

Signature of Subject Teacher

EXPERIMENT NO – 6

Aim: Write a C program to push and pop an element from stack.

Theory :

What is Stack?

- Stack is an ordered list of the same type of elements.
- It is a linear list where all insertions and deletions are permitted only at one end of the list.
- Stack is a LIFO (Last In First Out) structure.
- In a stack, when an element is added, it goes to the top of the stack.

Definition

"Stack is a collection of similar data items in which both insertion and deletion operations are performed based on LIFO principle".

There are two basic operations performed in a Stack:

1. Push()
2. Pop()

1. **Push()** function is used to add or insert new elements into the stack.
2. **Pop()** function is used to delete or remove an element from the stack.

When a stack is completely full, it is said to be **Overflow state** and if stack is completely empty, it is said to be **Underflow state**.

Stack allows operations at **one end only**. Stack behaves like a real life stack, for example, in a real life, we can remove a plate or dish from the top of the stack only or while playing a deck of cards, we can place or remove a card from top of the stack only. Similarly, here also, we can only access the top element of a stack.

According to its LIFO structure, the element which is inserted last, is accessed first.

Implementation of Stack

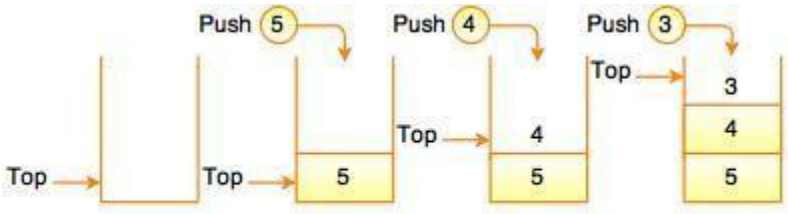


Fig. Insertion of Elements in a Stack

The above diagram represents a stack insertion operation. In a stack, inserting and deleting of elements are performed at a single position which is known as, **Top**. Insertion operation can be performed using Push() function. New element is added at top of the stack and removed from top of the stack, as shown in the diagram below:

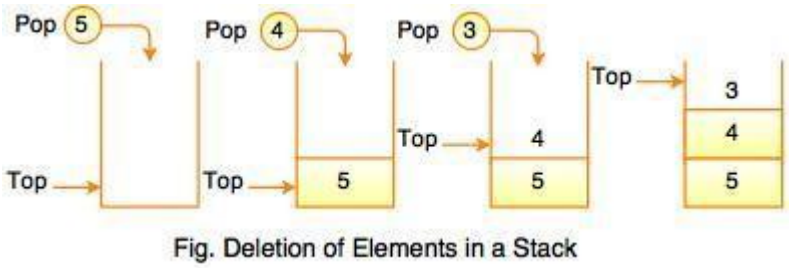


Fig. Deletion of Elements in a Stack

An element is removed from top of the stack. Delete operation is based on LIFO principle. This operation is performed using a Pop() function. It means that the insertion and deletion operations are performed at one end i.e at Top.

Following table shows the Position of Top which indicates the status of stack:

Position of Top	Status of Stack
-1	Stack is empty.
0	Only one element in a stack.
N - 1	Stack is full.
N	Stack is overflow. (Overflow state)

ALGORITHM:

a) Push Operation

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps –

Step 1 – Checks if the stack is full.

Step 2 – If the stack is full, produces an error and exit.

Step 3 – If the stack is not full, increments top to point next empty space.

Step 4 – Adds data element to the stack location, where top is pointing.

Step 5 – Returns success.

b) Pop Operation

Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead **top** is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.

A Pop operation may involve the following steps –

Step 1 − Checks if the stack is empty.

Step 2 – If the stack is empty, produces an error and exit.

Step 3 – If the stack is not empty, accesses the data element at which **top** is pointing.

Step 4 – Decreases the value of top by 1.

Step 5 – Returns success.

PROGRAM:

OUTPUT:

Conclusion: Thus we, have executed a C program to push and pop an element from stack successfully.

Viva Voce Question

1. What is stack?

2. What do we use stack?

3. What operation performed on stack.

Signature of Subject Teacher

EXPERIMENT NO – 7

Aim: Write a C program to insert and delete an element from queue.

Theory :

What is Queue?

Queue is a linear data structure where the first element is inserted from one end called **REAR** and deleted from the other end called as **FRONT**.

Front points to the **beginning** of the queue and **Rear** points to the **end** of the queue.

Queue follows the **FIFO (First - In - First Out)** structure.

According to its FIFO structure, element inserted first will also be removed first.

In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue), because queue is open at both its ends.

The enqueue() and dequeue() are two important functions used in a queue.

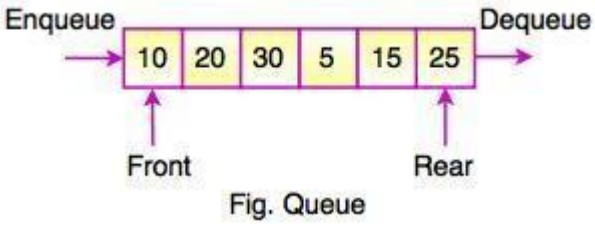


Fig. Queue

Operations on Queue

Following are the basic operations performed on a Queue.

Operations	Description
enqueue()	This function defines the operation for adding an element into queue.
dequeue()	This function defines the operation for removing an element from queue.
init()	This function is used for initializing the queue.
Front	Front is used to get the front data item from a queue.
Rear	Rear is used to get the last item from a queue.

Queue Implementation

Array is the easiest way to implement a queue. Queue can be also implemented using Linked List or Stack.

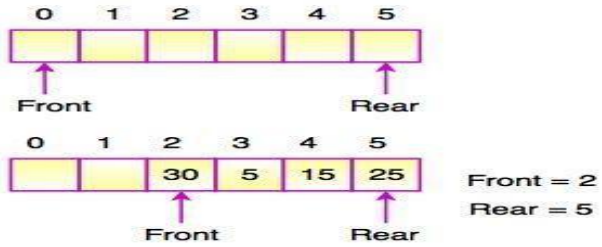


Fig. Implementation of Queue using Array

In the above diagram, Front and Rear of the queue point at the first index of the array. (Array index starts from 0).

While adding an element into the queue, the Rear keeps on moving ahead and always points to the position where the next element will be inserted. Front remains at the first index.

ALGORITHM:

a) Enqueue Operation

Queues maintain two data pointers, **front** and **rear**. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue −

Step 1 − Check if the queue is full.

Step 2 − If the queue is full, produce overflow error and exit.

Step 3 − If the queue is not full, increment **rear** pointer to point the next empty space.

Step 4 − Add data element to the queue location, where the rear is pointing.

Step 5 − return success.

b) Dequeue Operation

Accessing data from the queue is a process of two tasks − access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation −

Step 1 − Check if the queue is empty.

Step 2 − If the queue is empty, produce underflow error and exit.

Step 3 − If the queue is not empty, access the data where front is pointing.

Step 4 − Increment front pointer to point to the next available data element.

Step 5 − Return success.

PROGRAM:

OUTPUT:

Conclusion: Thus we, have executed a C program to insert and delete an element from queue successfully.

Viva Voce Question

1. What is a the queue in data structure?

2. What are the limitation of simple queue?

3. Application of queue?

Signature of Subject Teacher

EXPERIMENT NO – 5

Aim: Write a C program to create, insert and delete item in a singly link list.

Theory : What is Linked List?

Linked list is a linear data structure. It is a collection of data elements, called nodes pointing to the next node by means of a pointer.

Linked list is used to create trees and graphs.

In linked list, each node consists of its own data and the address of the next node and forms a chain.

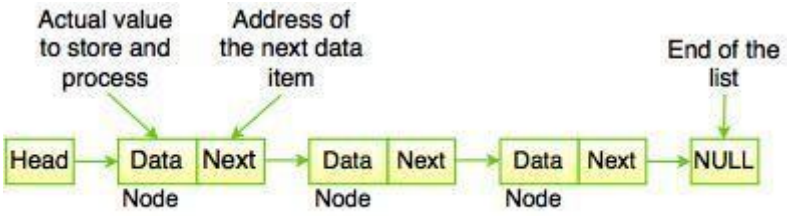


Fig. Linked List

The above figure shows the sequence of linked list which contains data items connected together via links. It can be visualized as a chain of nodes, where every node points to the next node.

Linked list contains a link element called first and each link carries a data item. Entry point into the linked list is called the head of the list.

Link field is called next and each link is linked with its next link. Last link carries a link to null to mark the end of the list.

Note: Head is not a separate node but it is a reference to the first node. If the list is empty, the head is a null reference.

Linked list is a dynamic data structure. While accessing a particular item, start at the head and follow the references until you get that data item.

Linked list is used while dealing with an unknown number of objects:

In the above diagram, Linked list contains two fields - First field contains value and second field contains a link to the next node. The last node signifies the end of the list that means NULL.

ALGORITHM:

How to create a linked list?

Step by step descriptive logic to create a linked list.

1. The first step of creating linked list of n nodes starts from defining node structure. We need a custom type to store our data and location of next linked node. Let us define our custom node structure

   ```
   struct node {
       int data;
       struct node *next;
   };
   ```

Where data is the data you want to store in list. *next is pointer to the same structure type.
The *next will store location of next node if exists otherwise NULL.

Note: The node structure may vary based on your requirement. You can also have user defined types as node data section.

2. Declare a pointer to node type variable to store link of first node of linked list.
   Say struct node *head;.

Note: You can also declare variable of node type along with node structure definition.

3. Input number of nodes to create from user, store it in some variable say n.
4. Declare two more helper variable of node type, say struct node *newNode, *temp;.
5. If n > 0 then, create our first node i.e. head node. Use dynamic memory allocation to allocate memory for a node. Say head = (struct node*)malloc(sizeof(struct node));.
6. If there is no memory to allocate for head node i.e. head == NULL. Then print some error message and terminate program, otherwise move to below step.
7. Input data from user and assign to head using head->data = data;.
8. At first head node points to NULL. Hence, assign head->next = NULL;.
9. Now, we are done with head node we should move to creation of other nodes. Copy reference of head to some other temporary variable, say temp = head;. We will use temp to store reference of previous node.
10. Allocate memory and assign memory reference to newNode,
    say newNode = (struct node*)malloc(sizeof(node));.
11. If memory got allocated successfully then read data from user and assign to data section of new node. Say newNode->data = data;.
12. Make sure new node points to NULL.
13. Now link previous node with newly created node i.e. temp->next = newNode;.
14. Make current node as previous node using temp = temp->next;.
15. Repeat step 10-14 for remaining n - 2 other nodes.

How to traverse a linked list?

Step by step descriptive logic to traverse a linked list.

1. Create a temporary variable for traversing. Assign reference of head node to it, say temp = head.
2. Repeat below step till temp != NULL.
3. temp->data contains the current node data. You can print it or can perform some calculation on it.
4. Once done, move to next node using temp = temp->next;.
5. Go back to 2nd step.

All possible cases OF INSERTION:

1. Insertion at beginning
2. Insertion at middle
3. Insertion at the ending

Algorithms:

Inserting at the beginning

In this case, a new node is to be inserted before the current head node, i.e. , every time the node that got inserted is being updated to the head node. Such insertion can be done using following steps.

1. Update *next* pointer of the new node (node to be inserted) to point to the current node.
2. Update *new* node as head node.

Inserting at the ending

In such case the new node is going to be the last node, i.e. , the next pointer of the new node is going to be NULL. The steps are:

1. Set the *next* pointer of the new node to be NULL.
2. Last node of the existing node is linked with the *new* node, i.e. , the last node's(existing) *next*pointer points to the new node.

Inserting at the middle

Such case can be handles using following steps:

1. Traverse to the desired node, after which we want to add the new node, let the pointed node be current node.
2. Set a temp node having the address of immediately next node of the current node.

3. Link the new node to the current node and the temp node to the new node. Current node's next node is the new node & new node's next one is the temp node (new node inserted in the middle).

Deletion can be at various positions like:

1. Deleting the first node
2. Deleting the last node
3. Deleting the intermediate node

Deleting the first node in single linked list

In such case, the head node is to be removed and the next node needs to be assigned as updated head node.

1. Create a temporary node, say temp which points to the head node (first node) of the list.
2. Move head node pointer to the immediate next node and delete (dispose) the temp node.

Deleting the last node in the single linked list

In such case the last node is to be removed from list. The steps are following:

1. We need to keep track of the previous node of last node. That's why while traversing to the last node we need to set a prev node also which will point to the previous node of the tail node(last node) after traversal.
2. So, we have two pointer, one tail node pointing to the last node and another is prev node pointing to the previous node of the last node.
3. Set the next pointer of the prev node to NULL and delete the tail node. (last node)

Deleting an intermediate node in the single linked list

In such case an intermediate node is to be deleted. The approach is quite similar to the previous.

1. Similar to the previous case, a curr node and a prev node is to be maintained. Curr node will point to the node to be deleted and prev node will point to the previous node.
2. Set the next pointer of the prev node to the next pointer of curr node.
3. Delete the curr node.

PROGRAM:

OUTPUT:

Conclusion: Thus we, have executed a C program to create, insert and delete item in a singly link list successfully.

Viva Voce Question

1. What type of memory allocation is referred for Linked lists?

2. What is the difference between array and linked list?

3. What are the application of linked list?

Signature of Subject Teacher