DSA RECORD

SWIKRITI KHADKE

AP19110010555 CSE-G

25. **Title:** C Program using the structure for entering details of the five students like name, admission number, date of the birth, department and display all the details.

Objective: At the end of this activity, we shall be able to use structures for the display the details for a group of people

Problem Statement: In this problem, we aim to understand and use the structures. It is a collection of variables. In this program, the details of 5 students will be displayed which were given by the user. Once the details of the 5 students are collected, we print the details of each student.

Algorithm:

START

DEFINE VARIABLES: first Name, Department, roll, date of birth

INPUT: Reads the input from the user

COMPUTATION: Takes the details of all the 5 students

DISPLAY: Prints the details of the 5 students using structures

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q25.C**

26. Title: C program to find the length of string using pointers.

Objective: At the end of this activity, we shall be able to find the length of string using pointers

Problem Statement:

In this program, we aim to pass this string to the function. Calculate the length of the string using pointer.

Algorithm:

START

DEFINE VARIABLES: str[20], length.

INPUT: Reads the input from the user.

COMPUTATION: Takes the string variable from the user. DISPLAY: It prints the length of the string which was given by the user.

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q26.C**

27. Title: C program to copy one string to another using pointer

Objective: At the end of this activity, we shall be able to copy one string to another string using pointer

Problem Statement: In this problem, we aim to understand how to copy one string to another using pointer. The input string which was given by the user will be copied to another string.

Algorithm:

START

DEFINE VARIABLES: s1,s2,*p1,*p2

INPUT: Reads the input from the user.

COMPUTATION: Takes the string variable from the user.

DISPLAY: It copies the string and displays to another string

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q27.C**

28. Title: C program to compare two strings using pointers.

Objective: At the end of this activity, we shall be able to know whether the two strings were different or same

Problem Statement:

In this problem, we aim to understand the pointers and how to compare the strings variables using pointers in C programming.

Algorithm:

START

DEFINE VARIABLES: string1, string2, *str1, *str2 INPUT: Reads the input from the user.

COMPUTATION: Takes two string variables from the user.

DISPLAY: It compares the two string variables and says whether they are same or not.

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q28.C**

29. Title: C program to find the reverse of a string non-recursively.

Objective: At the end of this activity, we shall be able to Print the reverse of the string which was given by the user as the input.

Problem Statement:

In this problem, we aim to understand how to reverse a string using pointers non-recursively.

Algorithm:

START

DEFINE VARIABLES: string, i, length, *begin, *end INPUT: Reads the input from the user.

COMPUTATION: Takes the string variable from the input. DISPLAY: It will reverse the string variable which was given by the user.

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q29A.C**

30. Title: To create a binary tree and output the data with 3 tree traversals

Objective: At the end of this activity, we shall be able to find the Inorder, Preorder, Postorder transversals in a Binary search tree.

Problem Statement:

In this problem, how the Inorder transversal, Preorder transversal and the Postorder transversal works in a Binary search tree.

Algorithm:

START

DEFINE VARIABLES: data, node* left, node* right

INPUT: Input of a binary search tree was given in the code itself. COMPUTATION: For the Inorder transversal First, It visits all the nodes in the left subtree then the root node lastly all nodes on the right subtree For Preorder transversal, first visit root node then all nodes of left subtree finally visits all the nodes in the right subtree For Post Transversal first it visits all the node in the left subtree then it visits all the nodes in right subtree finally it visits the root node

DISPLAY: It displays all the three transversals in a binary search tree.

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q30.C**

31. Title: To create a Binary Search Tree(BST) and search for a given value in BST.

Objective: At the end of this activity, we shall be able to - Search an element in a given binary search tree

Problem Statement: In this problem, we aim to understand how to search a particular element in a binary search tree.

Algorithm:

START

DEFINE VARIABLES: value, search_val INPUT: Reads the input from the user.

COMPUTATION: Takes the search element from the user and searches in the binary search tree

DISPLAY: It displays whether the element is present in a binary search tree or not.

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q31.c

34.

Title: To implement the STACK operation using array as a data structure. And using push, pop, peek, display elements in the stack

Objective:

At the end of this activity, we shall be able to

- Push an element to the stack

- Pop an element to the stack

- Peek element in the stack

Problem Statement:

Stack is basically a data object. A stack is a data structure in which items can be inserted only from one end and get items back from the same end. There, the last item inserted into stack, is the first item to be taken out from the stack. In short it's also called Last in First out.

Algorithm:

START

DEFINE VARIABLES: value, choice

INPUT: Takes the input from the user

COMPUTATION: Push, Add an element to the top of the stack.

Pop, Remove the element at the top of the stack.

Peek, prints the value of the top most element of the stack without deleting that element from the stack.

DISPLAY: It displays the elements in the stack after the operations.

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q34.C

35. Title: To write a C program to reverse a string using STACK.

Objective: At the end of this activity, we shall be able to reverse a string using stack

Problem Statement: In a data structure stack allows you to access the last data element that you inserted to stack, if you remove the last element of the stack, you will be able to access the next to last element. We can use this method or operation to reverse a string value.

Algorithm:

START

DEFINE VARIABLES: top, stack

INPUT: Takes the input from the user

COMPUTATION: Creates an empty stack. One by one push all characters of string to stack. One by one pop all characters from stack and put them back to string.

DISPLAY: It displays the reverse of the given string

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q35.C

36. Title: C program to convert the given infix expression to postfix expression using STACK.

Objective: At the end of this activity, we shall be able to Convert infix expression to the postfix expression

Problem Statement:

The compiler first scans the expression to evaluate the expression b * c, then again scan the expression to add a to it. The result is then added to d after another scan. The repeated scanning makes it very in-efficient. It is better to convert the expression to postfix(or prefix) form before evaluation.

Algorithm:

START

INPUT: Takes the input from the user

COMPUTATION: Scan the infix expression from left to right, If the scanned character is an operand, output it or else it will precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.

DISPLAY: Displays the postfix of the given Infix

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q36.C

37. Title: C program to convert the given in-fix expression to prefix expression using STACK.

Objective: At the end of this activity, we shall be able to convert an Infix expression to an Prefix expression using stack

Problem Statement: While we use infix expressions in our day to day lives. Computers have trouble understanding this format because they need to keep in mind rules of operator precedence and also brackets. Prefix and Postfix expressions are easier for a computer to understand and evaluate.

Algorithm:

START

Step 2. Scan A from right to left and repeat step 3 to 6 for each element of A until the

STACK is empty

Step 3. If an operand is encountered add it to B

Step 4. If a right parenthesis is encountered push it onto STACK Step 5. If an operator is encountered then:

a. Repeatedly pop from STACK and add to B each operator (on the top of STACK) which has the same or higher precedence than the operator. b. Add operator to STACK

Step 6. If left parenthesis is encountered then

a. Repeatedly pop from the STACK and add to B (each operator on top of stack until a

left parenthesis is encountered) b. Remove the left parenthesis

Step 7. STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q37.C

38. Title: C program to evaluate the given prefix expression.

Objective: At the end of this activity, we shall be able to Prefix and Postfix expressions can be evaluated faster than an infix expression. This is because we don’t need to process any brackets or follow operator precedence rules. In postfix and prefix expressions whichever operator comes before will be evaluated first, irrespective of its priority. Also, there are no brackets in these expressions.

Algorithm:

START

DEFINE VARIABLES: n1, n2, n3, num

1) Create a stack to store operands (or values).

2) Scan the given expression and do the following for every scanned element.

- If the element is a number, push it into the stack
- If the element is an operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack

3) When the expression is ended, the number in the stack is the final answer and prints the answer

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q38.C

39. Title: C program to implement a Linear-Queue, Adding an element; Removing an element; displaying elements.

Objective: At the end of this activity, we shall be able to To know more about Queue Like a stack, a queue is also a list. However, with a queue, insertion is done at one end, while deletion is performed at the other end.

Problem Statement:

Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other end called FRONT. Front points to the beginning of the queue and Rear points to the end of the queue.

Algorithm:

START For Enqueue:

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 the rear pointer to point to the next empty space.

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

Step 5 − return success.

For Dequeue:

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 the front is pointing.

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

Step 5 − Return success.

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q39.C

40. Title: C program to implement a Circular-Queue, Adding an element; Removing an element, displaying elements.

Objective: At the end of this activity, we shall be able to implement the Circular-Queue and add, remove elements in the queue.

Problem Statement: Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called ‘Ring Buffer’.

Algorithm:

START

Initialize the queue, with size of the queue defined (maxSize), and head and tail pointers. enqueue: Check if the number of elements is equal to maxSize - 1:

If Yes, then return Queue is full.

If No, then add the new data element to the location of the tail pointer and increment the tail pointer.

dequeue: Check if the number of elements in the queue is zero:

If Yes, then return Queue is empty.

If No, then increment the head pointer.

Finding the size:

If, tail >= head, size = (tail - head) + 1

But if, head > tail, then size = maxSize - (head - tail) + 1

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q40.C

41. Title: C program to create a singly linked list with 5 nodes. And display the linked-list elements.

Objective: At the end of this activity, we shall be able to create a singly linked list, Linked lists are often used because of their efficient insertion and deletion. They can be used to implement stacks, queues, and other abstract data types.

Problem Statement: A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers.

Algorithm:

START

A linked list is a series of connected nodes. Each node contains at least. A piece of data

(any type). Pointer to the next node in the list. Head: pointer to the first node. The last node points to NULL

Empty Linked list is a single pointer having the value of NULL.

head = NULL; head

Let’s assume that the node is given by the following type declaration: struct Node{ int data;

struct Node *next;

};

To start with, we have to create a node (the first node), and make a head point to it.

head = (struct Node*)malloc(sizeof(struct Node));

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q41.C**

42. Title: C program to search an element in a singly-linked list.

Objective: At the end of this activity, we shall be able to searching in singly linked list. Searching is performed in order to find the location of a particular element in the list. Searching any element in the list needs traversing through the list and making the comparison of every element of the list with the specified element.

Problem Statement: Search is one of the most common operations on performing any data structure. In this post I will explain how to search an element in a linked list (iterative and recursive) using the C program. I will explain both ways to search, how to search an element in linked list using loop and recursion.

Algorithm:

START

Input element to search from user. Store it in some variable say keyToSearch.

Declare two variables one to store the index of the found element and other to iterate

through the list. Say index = 0; and struct node *curNode = head; If curNode is not NULL and its data is not equal to keyToSearch. Then, increment the index and move curNode to its next node.

Repeat step 3 till curNode != NULL and element is not found, otherwise move to 5th

step. If curNode is not NULL, then element is found hence return index otherwise -1.

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q42.C**

43. Title: C program to perform, Insertion at the beginning; Insertion at the end; Insertion at the middle; Deletion from the beginning; Deletion from the end of a singly linked list.

Objective: At the end of this activity, we shall be able to Insert a node, Delete a node at the beginning of a singly linked list. Insert a node, Delete a node at the middle of a singly linked list. Insert a node, Delete a node at the end of a singly linked list.

Problem Statement: There are three different possibilities for inserting a node into a linked list. These three possibilities are: Insertion at the beginning of the list. Insertion at the end of the list Inserting a new node except the above-mentioned positions.

Algorithm:

START

A) Insert node at beginning of linked list

Step1: Create a Node

Step2: Set the node data Value in the node just created

Step3: Connect the pointers

B) Insert node at end of linked list

Step1: Create a Node

Step2: Set the node data Values

Step3: Connect the pointers

C) Insert node at middle of linked list

Step1: Create a Node

Step2: Set the node data Values

Step3: Break pointer connection

Step 4: Re-connect the pointers

D) Delete node at beginning of the linked list

Step1: Break the pointer connection

Step2: Re-connect the nodes

Step3: Delete the node

E) Delete node at end of linked list

Step1: Break the pointer connection

Step2: Set previous node pointer to NULL

Step3: Delete the node

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q43A.C

44. Title: C program to create a doubly linked list with 5 nodes.

Objective: At the end of this activity, we shall be able to Travers in both forward and backward direction. The delete operation in DLL is more efficient if a pointer to the node to be deleted is given. We can quickly insert a new node before a given node.

Problem Statement: A doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains three fields: two link fields (references to the previous and to the next node in the sequence of nodes) and one data field.

Algorithm:

START

DEFINE VARIABLES: num, n, *fnNode, *temp

INPUT: Takes the input from the user

COMPUTATION: Navigation is possible in both ways either forward and backward.

DISPLAY: It displays the data entered in the doubly linked list.

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q44.C

45. Title: C program to create a circular linked list with 5 nodes.

Objective: At the end of this activity, we shall be able to Accessing any node of the linked list, we start traversing from the first node. If we are at any node in the middle of the list, then it is not possible to access nodes that precede the given node. This problem can be solved by slightly altering the structure of singly linked lists.

Problem Statement:

Implement a circular singly linked list, we take an external pointer that points to the last node of the list. If we have a pointer last pointing to the last node, then last -> next will point to the first node.

Algorithm:

START

Step 1- To implement a circular singly linked list, we take an external pointer that points

Step 2- To the last node of the list. If we have a pointer last pointing to the last node

Step 3- Then last -> next will point to the first node.

Step 4- The pointer last points to node Z and last -> next points to node P.

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q45.C

46. Title: C program to implement the stack using linked lists.

Objective: At the end of this activity, we shall be able to Create a linked list and implement the stack using a linked list.

Problem Statement: This C Program implements a stack using linked lists. Stack is a type of queue that in practice is implemented as an area of memory that holds all local variables and parameters used by any function, and remembers the order in which functions are called so that function returns occur correctly.

Algorithm:

START

push

The steps for push operation are:

1.Make a new node.

2.Give the ‘data’ of the new node its value.

3.Point the ‘next’ of the new node to the top of the stack. 4.Make the ‘top’ pointer point to this new node pop

1.Make a temporary node.

2.Point this temporary node to the top of the stack

3.Store the value of ‘data’ of this temporary node in a variable.

4.Point the ‘top’ pointer to the node next to the current top node.

5.Delete the temporary node using the ‘free’ function.

6.Return the value stored in step 3.

STOP

Program: https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q46.C

47. Title: C program to implement the queue using a linked list.

Objective: At the end of this activity, we shall be able to Making a queue using a linked list is obviously a linked list.

Problem Statement: The major problem with the queue implemented using an array is, It 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.

Algorithm:

START enQueue

Step 1 - Create a newNode with given value and set 'newNode → next' to NULL.

Step 2 - Check whether queue is Empty (rear == NULL) Step 3 - If it is Empty then, set front = newNode and rear = newNode.

Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode. deQueue

Step 1 - Check whether the queue is Empty (front == NULL).

Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.

Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

Display

Step 1 - Check whether the queue is Empty (front == NULL).

Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.

Step 4 - Display 'temp → data --->' and move it to the next node.

Repeat the same until

'temp' reaches to 'rear' (temp → next != NULL).

Step 5 - Finally! Display 'temp → data ---> NULL'.

STOP

Program: **https://github.com/swikriti04/DSA_Lab_Record/blob/master/Q47.C**