DSA RECORD

Programs on Structure, String and Pointers

Lavu Harshitha
AP19110010041
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: firstName, 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 in C(code)

```
#include <stdio.h>
struct student {
  char firstName[50],Department[50];
```

```
  int roll,date_of_birth;
} s[10];

int main() {
  int i;
  printf("Enter information of students:\n");

   // storing information
   for (i = 0; i < 5; ++i) {
       s[i].roll = i + 1;
       printf("\nFor roll number%d,\n", s[i].roll);
       printf("Enter first name: ");
       scanf("%s", s[i].firstName);
       printf("\nEnter your Date of Birth(in DDMMYYYY Format): ");
       scanf("%d", &s[i].date_of_birth);
       printf("\nEnter your Department: ");
       scanf("%s", s[i].Department);
   }
   printf("Displaying Information:\n\n");

   // displaying information
   for (i = 0; i < 5; ++i) {
      printf("\nRoll number: %d", i + 1);
      printf("\nFirst name: ");
      puts(s[i].firstName);
      printf("\nDate of Birth: %d(in DDMMYYYY format)", s[i].date_of_birth);
      printf("\nDepartment: %s", s[i].Department);
   }
   return 0;
}
```

Testcase:

Enter information of students:
For roll number 1,
Enter a first name: Saiharsha

Enter Your Date of Birth(in DDMMYYYY Format): 27112001

Enter your Department: CSE

For roll number 2,
Enter a first name: Arjun

Enter Your Date of Birth(in DDMMYYYY Format): 12022001

Enter your Department: CSE

For roll number 3,
Enter a first name: Aryan

Enter Your Date of Birth(in DDMMYYYY Format): 18062001

Enter your Department: CSE

For roll number 4,
Enter a first name: Deepak

Enter Your Date of Birth(in DDMMYYYY Format): 28012001

Enter your Department: CSE

For roll number 5,
Enter a first Name: Rishi

Enter Your Date of Birth(in DDMMYYYY Format): 11122001

Enter your Department: CSE

Displaying Information:

Roll number: 1
First name: Saiharsha
Date of Birth: 27112001(in DDMMYYYY format)
Department: CSE
Roll number: 2
First name: Arjun
Date of Birth: 12022001(in DDMMYYYY format)
Department: CSE
Roll number: 3
First name: Aryan
Date of Birth: 18062001(in DDMMYYYY format)
Department: CSE

Roll number: 4
First name: Deepak
Date of Birth: 28012001(in DDMMYYYY format)
Department: CSE
Roll number: 5
First name: rishi
Date of Birth: 11122001(in DDMMYYYY format)
Department: CSE

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 in C(code)

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

int string_ln(char*);

void main() {
  char str[20];
  int length;

    printf("\nEnter any string: ");
```

```
    gets(str);

    length = string_ln(str);
    printf("The length of the given string %s is: %d", str, length);
    getch();
}

int string_ln(char*p) /* p=&str[0] */
{
  int count = 0;
  while (*p != '\0') {
     count++;
     p++;
  }
  return count;
}
```

Testcase:

Enter any string: Saiharsha
The length of the given string Saiharsha is: 9

27.
Title: C program to copy one string to another using pointer
Objective:
At the end of this activity, we shall be able to
- 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 in C(code)

```
#include<stdio.h>
int main()
{
   char s1[10],s2[10],*p1,*p2;
   printf("\nEnter a string: ");
   scanf("%s",s1);

    p1 = s1;
    p2 = s2;

    while(*p1 != '\0')
    {
       *p2 = *p1;
       p1++;
       p2++;
    }
    *p2 = '\0';
    printf("Copied string: %s",s2);
}
```

Testcase:

Enter a string: Saiharsha
Copied string: Saiharsha

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 in C(code)

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

  char string1[50],string2[50],*str1,*str2;
  int i,equal = 0;

  printf("Enter The First String: ");
  scanf("%s",string1);

  printf("Enter The Second String: ");
    scanf("%s",string2);

    str1 = string1;
    str2 = string2;

    while(*str1 == *str2)
    {

        if ( *str1 == '\0' || *str2 == '\0' )
            break;

        str1++;
        str2++;

    }

    if( *str1 == '\0' && *str2 == '\0' )
        printf("\n\nBoth Strings Are Equal.");

    else
       printf("\n\nBoth Strings Are Not Equal.");

}
```

Testcase:

Enter The First String: Arjun
Enter The Second String: Aryan

Both Strings Are Not Equal.

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 in C(code)

```
#include<stdio.h>

int string_length(char*);
void reverse(char*);

int main()
{
  char string[100];

    printf("Enter a string\n");
    gets(string);

    reverse(string);

    printf("The reverse of entered string is \"%s\".\n", string);
```

```
    return 0;
}

void reverse(char *string)
{
  int length, i;
  char *begin, *end, temp;

    length = string_length(string);
    begin = string;
    end = string;

   for ( i = 0 ; i < ( length - 1 ) ; i++ )
     end++;
// swap the chars till half of the length of the string
//begin with the end char and so on
   for ( i = 0 ; i < length/2 ; i++ )
   {
     temp = *end;
     *end = *begin;
     *begin = temp;

        begin++;
        end--;
    }
}

int string_length(char *ptr)
{
  int len = 0;

    while( *(ptr+len) != '\0' )
     len++;

    return len;
}
```

Testcase

Enter a string
Saiharsha
The reverse of entered string is "ahsrahiaS"

Programs on Trees and Graphs:

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 in C(code)

```
// Tree traversal in C

#include <stdio.h>
#include <stdlib.h>

struct node {
  int data;
  struct node* left;
  struct node* right;
};
```

```
// Inorder traversal
void inorder(struct node* root) {
  if (root == NULL) return;
  inorder(root->left);
  printf("%d ->", root->data);
  inorder(root->right);
}

// Preorder traversal
void preorder(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->data);
  preorder(root->left);
  preorder(root->right);
}

// Postorder traversal
void postorder(struct node* root) {
  if (root == NULL) return;
  postorder(root->left);
  postorder(root->right);
  printf("%d ->", root->data);
}

// Create a new Node
struct node* createNode(value) {
  struct node* newNode = malloc(sizeof(struct node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
  root->left = createNode(value);
  return root->left;
}

// Insert on the right of the node
struct node* insertRight(struct node* root, int value) {
```

```
  root->right = createNode(value);
  return root->right;
}

int main() {
  struct node* root = createNode(1);
  insertLeft(root, 12);
  insertRight(root, 9);

    insertLeft(root->left, 5);
    insertRight(root->left, 6);

    printf("Inorder traversal \n");
    inorder(root);

    printf("\nPreorder traversal \n");
    preorder(root);

    printf("\nPostorder traversal \n");
    postorder(root);
}
```

Testcase:

Inorder traversal
5 ->12 ->6 ->1 ->9 ->
Preorder traversal
1 ->12 ->5 ->6 ->9 ->
Postorder traversal
5 ->6 ->12 ->9 ->1 ->

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 in C(code)

```
#include <stdio.h>
#include <malloc.h>
/* Structure to create the binary tree */

struct btnode
{
   int value;
   struct btnode *l;
   struct btnode *r;
};
struct btnode *root = NULL;
int flag;

/* Function Prototypes */
void in_order_traversal(struct btnode *);
void in_order_search(struct btnode *,int);
struct btnode *newnode(int);

void main()
{
  /* Inserting elements in the binary tree */
  int search_val;
  root = newnode(50);
  root->l = newnode(20);
  root->r = newnode(30);
  root->l->l = newnode(70);
  root->l->r = newnode(80);
  root->l->l->l = newnode(10);
  root->l->l->r = newnode(40);
  root->l->r->r = newnode(60);

    printf("The elements of Binary tree are:");
    in_order_traversal(root);
    printf("\nEnter the value to be searched:");
```

```
    scanf("%d", &search_val);
    in_order_search(root, search_val);
    if (flag == 0) // flag to check if the element is present in the tree or not
    {
        printf("Element not present in the binary tree\n");
    }
}

/* Code to dynamically create new nodes */
struct btnode* newnode(int value)
{
   struct btnode *temp = (struct btnode *)malloc(sizeof(struct btnode));
   temp->value = value;
   temp->l = NULL;
   temp->r = NULL;
    return temp;
}

/* Code to display the elements of the binary tree */

void in_order_traversal(struct btnode *p)
{
  if (!p)
  {
      return;
  }
  in_order_traversal(p->l);
  printf("%d->", p->value);
  in_order_traversal(p->r);
}

/* Code to search for a particular element in the tree */
void in_order_search(struct btnode *p, int val)
{
   if (!p)
   {
       return;
   }
   in_order_search(p->l, val);
   if(p->value == val)
   {
       printf("\nElement present in the binary tree.\n");
       flag = 1;
```

```
  }
  in_order_search(p->r, val);
}
```

Testcase:

The elements of Binary tree are:10->70->40->20->80->60->50->30->
Enter the value to be searched:80

Element present in the binary tree.

32.

Title: To implement a single source shortest path algorithm by Bellman-Ford

Objective:
At the end of this activity, we shall be able to
- Find the shortest distance using a Bellman-Ford Algorithm

Problem Statement:
In this problem, we aim to understand how the Bellman-Ford Algorithm works to find the shortest path from the source vertex to all the vertex

Algorithm:
START
DEFINE VARIABLES: V, edge, G, i, j, k=0
INPUT: Takes the input from the user in the matrix form
COMPUTATION: Bellman Ford's algorithm is used to find the shortest paths from the source vertex to all other vertices in a weighted graph.
DISPLAY: It displays the shortest path from the source vertex
STOP

Program in C(code)

```
#include <stdio.h>
#include <stdlib.h>
int Bellman_Ford(int G[20][20] , int V, int E, int edge[20][2])
{
int i,u,v,k,distance[20],parent[20],S,flag=1;
```

```
for(i=0;i<V;i++)
distance[i] = 1000 , parent[i] = -1 ;
printf("Enter source: ");
scanf("%d",&S);
distance[S-1]=0 ;
for(i=0;i<V-1;i++)
{
for(k=0;k<E;k++)
{
u = edge[k][0] , v = edge[k][1] ;
if(distance[u]+G[u][v] < distance[v])
distance[v] = distance[u] + G[u][v] , parent[v]=u ;
}
}
for(k=0;k<E;k++)
{
u = edge[k][0] , v = edge[k][1] ;
if(distance[u]+G[u][v] < distance[v])
flag = 0 ;
}
if(flag)
for(i=0;i<V;i++)
printf("Vertex %d -> cost = %d parent = %d\n",i+1,distance[i],parent[i]+1);
return flag;
}
int main()
{
int V,edge[20][2],G[20][20],i,j,k=0;
printf("BELLMAN FORD\n");
printf("Enter no. of vertices: ");
scanf("%d",&V);
printf("Enter graph in matrix form:\n");
for(i=0;i<V;i++)
for(j=0;j<V;j++)
{
scanf("%d",&G[i][j]);
if(G[i][j]!=0)
edge[k][0]=i,edge[k++][1]=j;
}
if(Bellman_Ford(G,V,k,edge))
printf("\nNo negative weight cycle\n");
else printf("\nNegative weight cycle exists\n");
return 0;
```

}

Testcase:

BELLMAN FORD
Enter no. of vertices: 5
Enter graph in matrix form:
0 2 1000 1 1000
1000 0 3 1000 1000
1000 1000 0 1000 1
1000 -2 1000 0 1000
1000 1000 1000 1 0
Enter source: 1
Vertex 1 -> cost = 0 parent = 0
Vertex 2 -> cost = -1 parent = 4
Vertex 3 -> cost = 2 parent = 2
Vertex 4 -> cost = 1 parent = 1
Vertex 5 -> cost = 3 parent = 3

No negative weight cycle

33.
Title: To find All-to-all Shortest paths in a Graph using C program

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

- To find all pairs shortest path problems from a given weighted graph. As a result of this algorithm, it will generate a matrix, which will represent the minimum distance from any node to all other nodes in the graph.

Problem Statement:
In this problem, we aim to understand all pair shortest path problems from a given weighted graph.

Algorithm:
START
DEFINE VARIABLES: costMat
INPUT: Takes the input from the user in the matrix form
COMPUTATION: It is used to find all pairs shortest path problem from a given weighted graph. As a result of this algorithm.
DISPLAY: It displays that the output matrix will be updated with all vertices k as the intermediate vertex.
STOP

Program in C(code)

```
#include<iostream>
#include<iomanip>
#define NODE 7
#define INF 999
using namespace std;
//Cost matrix of the graph
int costMat[NODE][NODE] = {
  {0, 3, 6, INF, INF, INF, INF},
  {3, 0, 2, 1, INF, INF, INF},
  {6, 2, 0, 1, 4, 2, INF},
  {INF, 1, 1, 0, 2, INF, 4},
  {INF, INF, 4, 2, 0, 2, 1},
  {INF, INF, 2, INF, 2, 0, 1},
  {INF, INF, INF, 4, 1, 1, 0}
};
void floydWarshal(){
  int cost[NODE][NODE]; //defind to store shortest distance from any node to any node
 for(int i = 0; i<NODE; i++)
  for(int j = 0; j<NODE; j++)
    cost[i][j] = costMat[i][j]; //copy costMatrix to new matrix
    for(int k = 0; k<NODE; k++){
      for(int i = 0; i<NODE; i++)
        for(int j = 0; j<NODE; j++)
          if(cost[i][k]+cost[k][j] < cost[i][j])
            cost[i][j] = cost[i][k]+cost[k][j];
 }
 cout << "The matrix:" << endl;
 for(int i = 0; i<NODE; i++){
  for(int j = 0; j<NODE; j++)
    cout << setw(3) << cost[i][j];
```

```
   cout << endl;
 }
}
int main(){
  floydWarshal();
}
```

Testcase:

Input
036∞∞∞∞
3021∞∞∞
620142∞
∞1102∞4
∞∞42021
∞∞2∞201
∞∞∞4110

Output
0345677
3021344
4201323
5110233
6332021
7423201
7433110

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 in C(code)

```
#include<stdio.h>
#define SIZE 10
void push(int);
void pop();
void display();
int stack[SIZE], top = -1;
void main()
{
  int value, choice;
  while(1){
     printf("\n\n***** MENU *****\n");
     printf("1. Push\n2. Pop\n3. Peek \n 4. Display \n 5. Exit");
     printf("\nEnter your choice: ");
     scanf("%d",&choice);
     switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d",&value);
                 push(value);
                 break;
```

```
         case 2: pop();
                 break;
        case 3: peek();
                 break;
         case 4: display();
                 break;
         case 5: exit(0);
         default: printf("\nWrong selection!!! Try again!!!");
         }
         }
}
void push(int value){
  if(top == SIZE-1)
     printf("\nStack is Full!!! Insertion is not possible!!!");
  else{
     top++;
     stack[top] = value;
     printf("\nInsertion success!!!");
  }
}
void pop(){
  if(top == -1)
     printf("\nStack is Empty!!! Deletion is not possible!!!");
 else{
   printf("\nDeleted : %d", stack[top]);
   top--;
 }
}
void display(){
  if(top == -1)
     printf("\nStack is Empty!!!");
  else{
     int i;
     printf("\nStack elements are:\n");
     for(i=top; i>=0; i--)
          printf("%d\n",stack[i]);
  }
}
void peek(){
  if(top == -1)
     printf("\nStack is Empty!!!");
  else{
     int i;
```

```
    printf("\nStack top most element is: %d\n",stack[top]);
}
}
```

Testcase:

```
***** MENU *****
1. Push
2. Pop
3. Peek
  4. Display
  5. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion success!!!

***** MENU *****
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 2

Deleted : 10

***** MENU *****
1. Push
2. Pop
3. Peek
 4. Display
 5. Exit
Enter your choice: 3

Stack is Empty!!!

***** MENU *****
1. Push
2. Pop
3. Peek
```

 4. Display
 5. Exit
Enter your choice: 4

Stack is Empty!!!

***** MENU *****
1. Push
2. Pop
3. Peek
 4. Display
 5. Exit
Enter your choice: 5

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 in C(code)

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

#define max 100
int top,stack[max];

void push(char x){

    // Push(Inserting Element in stack) operation
    if(top == max-1){
        printf("stack overflow");
    } else {
        stack[++top]=x;
      }

}

void pop(){
  // Pop (Removing element from stack)
    printf("%c",stack[top--]);
}



main()
{
  char str[50];
  printf("Enter the string\n");
  scanf("%s",str);
  int len = strlen(str);
  int i;

    for(i=0;i<len;i++)
        push(str[i]);

    for(i=0;i<len;i++)
      pop();
}
```

Testcase:

Enter the string: saiharsha
Reversed string: ahsrahias

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 in C(code)

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

```
char stack[20];
int top = -1;
void push(char x)
{
   stack[++top] = x;
}

char pop()
{
  if(top == -1)
      return -1;
  else
      return stack[top--];
}

int priority(char x)
{
   if(x == '(')
       return 0;
   if(x == '+' || x == '-')
       return 1;
   if(x == '*' || x == '/')
       return 2;
}

main()
{
  char exp[20];
  char *e, x;
  printf("Enter the expression :: ");
  scanf("%s",exp);
  e = exp;
  while(*e != '\0')
  {
     if(isalnum(*e))
         printf("%c",*e);
     else if(*e == '(')
         push(*e);
     else if(*e == ')')
     {
         while((x = pop()) != '(')
            printf("%c", x);
     }
```

```
    else
    {
        while(priority(stack[top]) >= priority(*e))
            printf("%c",pop());
        push(*e);
    }
    e++;
  }
  while(top != -1)
   {
       printf("%c",pop());
   }
}
```

Testcase

Enter the expression :: (a+b)*c+(d-a)
ab+c*da-+
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 in C(code)

```
#define SIZE 50 /* Size of Stack */
#include<string.h>
#include <ctype.h>
#include<stdio.h>
char s[SIZE]; int top=-1; /* Global declarations */
push(char elem)
{ /* Function for PUSH operation */
s[++top]=elem;
}
char pop()
{ /* Function for POP operation */
return(s[top--]);
}
int pr(char elem)
{ /* Function for precedence */
switch(elem)
{
case '#': return 0;
case ')': return 1;
case '+':
case '-': return 2;
case '*':
case '/':return 3;
}
}
```

```
main()
{ /* Main Program */
char infx[50],prfx[50],ch,elem;
int i=0,k=0;
printf("\n\nInfix Expression: ");
scanf("%s",infx);
push('#');
strrev(infx);
while( (ch=infx[i++]) != '\0')
{
if( ch == ')')
push(ch);
else if(isalnum(ch))
prfx[k++]=ch;
else if( ch == '(')
{
while( s[top] != ')')
prfx[k++]=pop();
elem=pop(); /* Remove ) */
}
else
{ /* Operator */
while( pr(s[top]) >= pr(ch) )
prfx[k++]=pop(); push(ch);
}
}
while( s[top] != '#') /* Pop from stack till empty */
prfx[k++]=pop();
prfx[k]='\0'; /* Make prfx as valid string */
strrev(prfx);
strrev(infx);
printf("\n\nGiven Infix Expn: %s \nPrefix Expn: %s\n",infx,prfx);
}
```

Testcase

```
Infix Expression: (A+B)*(B-C)



Given Infix Expn: (A+B)*(B-C)
Prefix Expn: *+AB-BC
```

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 in C(code)

```
#include<stdio.h>
int stack[20];
int top = -1;

void push(int x)
{
    stack[++top] = x;
}
int pop()
{
     return stack[top--];
}
```

```
int main()
{
    char exp[20];
    char *e;
    int n1,n2,n3,num;
    printf("Enter the expression :: ");
    scanf("%s",exp);
    e = exp;
    while(*e != '\0')
    {
        if(isdigit(*e))
        {
            num = *e - 48;
            push(num);
        }
        else
        {
            n1 = pop();
            n2 = pop();
            switch(*e)
            {
                case '+':
                {
                    n3 = n1 + n2;
            break;
                }
                case '-':
                {
                    n3 = n2 - n1;
                    break;
                }
                case '*':
                {
                    n3 = n1 * n2;
                    break;
                }
                case '/':
                {
                    n3 = n2 / n1;
                    break;
                }
            }
```

```
            push(n3);
        }
        e++;
    }
    printf("\nThe result of expression %s = %d\n\n",exp,pop());
    return 0;

}
```

Testcase

Enter the expression :: 245+*

The result of expression 245+* = 18

Programs on Queues:

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 in C(code)

```
#include<stdio.h>
#define SIZE 10
void enQueue(int);
void deQueue();
void display();
int queue[10], front = -1, rear = -1;
void main() {
  int value, choice;
  while(1){
  printf("\n\n***** MENU *****\n");
  printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
  printf("\nEnter your choice: ");
  scanf("%d",&choice);
  switch(choice){
 case 1: printf("Enter the value to be insert: ");
 scanf("%d",&value);
 enQueue(value);
 break;
 case 2: deQueue();
 break;
 case 3: display();
 break;
 case 4: exit(0);
 default: printf("\nWrong selection!!! Try again!!!");
 }
}}
```

```
void enQueue(int value){
 if((front==0 &&rear == SIZE-1) || front==rear+1)
 printf("\nQueue is Full!!! Insertion is not possible!!!");
 else{
 if(front == -1)
 front = 0;
 rear=(rear+1)%SIZE;
 queue[rear] = value;
 printf("\nInsertion success!!!");
 }}
void deQueue(){
 if(front == -1)
 printf("\nQueue is Empty!!! Deletion is not possible!!!");
 else{
 printf("\nDeleted : %d", queue[front]);
 front=(front+1)%SIZE;
 if(front == rear)
 front = rear = -1;
 }}
void display(){
 if(front == -1)
 printf("\nQueue is Empty!!!");
 else{
 int i;
 printf("\nQueue elements are:\n");
 for(i=front; i!=rear; i=(i+1)%SIZE)
printf("%d\t",queue[i]);
}}
```

Testcase:

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion success!!!

***** MENU *****
1. Insertion
```

2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 30

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 10

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
20

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 4

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 in C(code)

```
#include<stdio.h>
#define SIZE 10
void enQueue(int);
void deQueue();
void display();
```

```
int queue[10], front = -1, rear = -1;
void main() {
  int value, choice;
  while(1){
  printf("\n\n***** MENU *****\n");
  printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
  printf("\nEnter your choice: ");
  scanf("%d",&choice);
  switch(choice){
  case 1: printf("Enter the value to be insert: ");
  scanf("%d",&value);
  enQueue(value);
  break;
  case 2: deQueue();
  break;
 case 3: display();
 break;
 case 4: exit(0);
 default: printf("\nWrong selection!!! Try again!!!");
 }
}}
void enQueue(int value){
 if((front==0 &&rear == SIZE-1) || front==rear+1)
 printf("\nQueue is Full!!! Insertion is not possible!!!");
 else{
 if(front == -1)
 front = 0;
 rear=(rear+1)%SIZE;
 queue[rear] = value;
 printf("\nInsertion success!!!");
 }}
void deQueue(){
 if(front == -1)
 printf("\nQueue is Empty!!! Deletion is not possible!!!");
 else{
 printf("\nDeleted : %d", queue[front]);
 front=(front+1)%SIZE;
 if(front == rear)
 front = rear = -1;
 }}
void display(){
 if(front == -1)
 printf("\nQueue is Empty!!!");
```

```
else{
int i;
printf("\nQueue elements are:\n");
for(i=front; i!=rear; i=(i+1)%SIZE)
printf("%d\t",queue[i]);
}}
```

Testcode:

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 30

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 40

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 10

***** MENU *****
1. Insertion
2. Deletion
3. Display
```

4. Exit
Enter your choice: 3

Queue elements are:
20   30

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 4

Programs on Linked-list

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 in C(code):

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

struct node
{
   int num;                 //Data of the node
   struct node *nextptr;        //Address of the next node
}*stnode;

void createNodeList(int n); // function to create the list
void displayList();    // function to display the list

int main()
{
   int n;
            printf("\n\n Linked List : To create and display Singly Linked List :\n");
            printf("-------------------------------------------------------------\n");

  printf(" Input the number of nodes : ");
  scanf("%d", &n);
  createNodeList(n);
  printf("\n Data entered in the list : \n");
  displayList();
  return 0;
}
void createNodeList(int n)
{
```

```
  struct node *fnNode, *tmp;
  int num, i;
    stnode = (struct node *)malloc(sizeof(struct node));

    if(stnode == NULL) //check whether the fnnode is NULL and if so no memory
allocation
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
// reads data for the node through keyboard

      printf(" Input data for node 1 : ");
      scanf("%d", &num);
      stnode->num = num;
      stnode->nextptr = NULL; // links the address field to NULL
      tmp = stnode;
// Creating n nodes and adding to linked list
      for(i=2; i<=n; i++)
      {
        fnNode = (struct node *)malloc(sizeof(struct node));
        if(fnNode == NULL)
        {
            printf(" Memory can not be allocated.");
            break;
        }
        else
        {
            printf(" Input data for node %d : ", i);
            scanf(" %d", &num);

              fnNode->num = num;     // links the num field of fnNode with num
              fnNode->nextptr = NULL; // links the address field of fnNode with NULL

              tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode
              tmp = tmp->nextptr;
          }
      }
    }
}
void displayList()
{
```

```
  struct node *tmp;
  if(stnode == NULL)
  {
      printf(" List is empty.");
  }
  else
  {
      tmp = stnode;
      while(tmp != NULL)
      {
         printf(" Data = %d\n", tmp->num);    // prints the data of current node
         tmp = tmp->nextptr;              // advances the position of current node
      }
  }
}
```

Testcase:

```
 Linked List : To create and display Singly Linked List :
-------------------------------------------------------------
 Input the number of nodes : 5
 Input data for node 1 : 9
 Input data for node 2 : 18
 Input data for node 3 : 27
 Input data for node 4 : 36
 Input data for node 5 : 45

Data entered in the list :
Data = 9
Data = 18
Data = 27
Data = 36
Data = 45
```

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 in C(code):

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

struct node
{
 int num;
   struct node *nextptr;
}

stnode, *ennode;

int FindElement(int);
void main()
{
```

```
    int n,i,FindElem,FindPlc;
    stnode.nextptr=NULL;
    ennode=&stnode;
        printf("\n\n Linked List : Search an element in a Singly Linked List :\n");
        printf("-------------------------------------------------------------\n");

   printf(" Input the number of nodes : ");
   scanf("%d", &n);
     printf("\n");
     for(i=0;i< n;i++)
     {
         ennode->nextptr=(struct node *)malloc(sizeof(struct node));
         printf(" Input data for node %d : ",i+1);
         scanf("%d",&ennode->num);
         ennode=ennode->nextptr;
     }
     ennode->nextptr=NULL;
     printf("\n Data entered in the list are :\n");

   ennode=&stnode;
    while(ennode->nextptr!=NULL)
    {
        printf(" Data = %d\n",ennode->num);
        ennode=ennode->nextptr;
    }

     printf("\n");
     printf(" Input the element to be searched : ");
     scanf("%d",&FindElem);
     FindPlc=FindElement(FindElem);
     if(FindPlc<=n)
        printf(" Element found at node %d \n\n",FindPlc);
    else
        printf(" This element does not exists in linked list.\n\n");
}
int FindElement(int FindElem)
{
    int ctr=1;
    ennode=&stnode;
    while(ennode->nextptr!=NULL)
    {
        if(ennode->num==FindElem)
            break;
```

```
        else
            ctr++;
            ennode=ennode->nextptr;
    }
    return ctr;
}
```

Testcase:

Linked List : Search an element in a Singly Linked List :

Input the number of nodes : 3

Input data for node 1 : 30
Input data for node 2 : 40
Input data for node 3 : 50

Data entered in the list are :
Data = 30
Data = 40
Data = 50

Input the element to be searched: 20
This element does not exist in the linked list.
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

Program in C(code):
A)
```
#include <stdio.h>
#include <stdlib.h>

struct node
{
  int num;              //Data of the node
  struct node *nextptr;     //Address of the node
}*stnode;

void createNodeList(int n); //function to create the list
```

```
void NodeInsertatBegin(int num);          //function to insert node at the beginning
void displayList();       //function to display the list

int main()
{
   int n,num;
               printf("\n\n Linked List : Insert a new node at the beginning of a Singly
Linked List:\n");
               printf("-------------------------------------------------------------------------\n");
   printf(" Input the number of nodes : ");
   scanf("%d", &n);
   createNodeList(n);
   printf("\n Data entered in the list are : \n");
   displayList();
   printf("\n Input data to insert at the beginning of the list : ");
   scanf("%d", &num);
   NodeInsertatBegin(num);
   printf("\n Data after inserted in the list are : \n");
   displayList();
  return 0;
}
void createNodeList(int n)
{
  struct node *fnNode, *tmp;
  int num, i;

    stnode = (struct node *)malloc(sizeof(struct node));
    if(stnode == NULL) //check whether the stnode is NULL and if so no memory
allocation
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
// reads data for the node through keyboard
        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        stnode-> num = num;
        stnode-> nextptr = NULL; //Links the address field to NULL
        tmp = stnode;

//Creates n nodes and adds to linked list
     for(i=2; i<=n; i++)
```

```
  {
    fnNode = (struct node *)malloc(sizeof(struct node));



    if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory
allocation
    {
      printf(" Memory can not be allocated.");
      break;
    }
    else
    {
      printf(" Input data for node %d : ", i);
      scanf(" %d", &num);
      fnNode->num = num;         // links the num field of fnNode with num
        fnNode->nextptr = NULL; // links the address field of fnNode with NULL
        tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode
        tmp = tmp->nextptr;
      }
    }
  }
}

void NodeInsertatBegin(int num)
{
 struct node *fnNode;
 fnNode = (struct node*)malloc(sizeof(struct node));
 if(fnNode == NULL)
 {
    printf(" Memory can not be allocated.");
 }
 else
 {
    fnNode->num = num; //Links the data part
    fnNode->nextptr = stnode; //Links the address part
    stnode = fnNode; //Makes stnode as first node
 }
}

void displayList()
{
 struct node *tmp;
```

```
  if(stnode == NULL)
  {
      printf(" No data found in the list.");
  }
  else
  {
      tmp = stnode;
      while(tmp != NULL)
      {
         printf(" Data = %d\n", tmp->num); // prints the data of current node
         tmp = tmp->nextptr;               // advances the position of current node
      }
    }
}
```

Testcase:

```
  Linked List : Insert a new node at the beginning of a Singly Linked List:
-------------------------------------------------------------------------------
  Input the number of nodes : 3
  Input data for node 1 : 10
  Input data for node 2 : 20
  Input data for node 3 : 30

Data entered in the list are :
Data = 10
Data = 20
Data = 30

Input data to insert at the beginning of the list : 5

Data after inserted in the list are :
Data = 5
Data = 10
Data = 20
Data = 30
```

B.

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

struct node
{
```

```
   int num;              //Data of the node
   struct node *nextptr;     //Address of the node
}*stnode;

void createNodeList(int n); //function to create the list
void NodeInsertatEnd(int num); //function to insert node at the end
void displayList();       //function to display the list
int main()
{
   int n,num;
            printf("\n\n Linked List : Insert a new node at the end of a Singly Linked
List :\n");
            printf("-------------------------------------------------------------------------\n");

   printf(" Input the number of nodes : ");
   scanf("%d", &n);
   createNodeList(n);
   printf("\n Data entered in the list are : \n");
   displayList();
   printf("\n Input data to insert at the end of the list : ");
   scanf("%d", &num);
   NodeInsertatEnd(num);
   printf("\n Data, after inserted in the list are : \n");
   displayList();
   return 0;
}
void createNodeList(int n)
{
    struct node *fnNode, *tmp;
    int num, i;
    stnode = (struct node *)malloc(sizeof(struct node));
    if(stnode == NULL) //check whether the stnode is NULL and if so no memory
allocation
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
// reads data for the node through keyboard
        printf(" Input data for node 1 : ");
        scanf("%d", &num);

     stnode-> num = num;
```

```
    stnode-> nextptr = NULL; //Links the address field to NULL
    tmp = stnode;
//Creates n nodes and adds to linked list
    for(i=2; i<=n; i++)
    {
      fnNode = (struct node *)malloc(sizeof(struct node));
      if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory
allocation
      {
         printf(" Memory can not be allocated.");
         break;
      }
      else
      {
         printf(" Input data for node %d : ", i);
         scanf(" %d", &num);
         fnNode->num = num;         // links the num field of fnNode with num
         fnNode->nextptr = NULL; // links the address field of fnNode with NULL
         tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode
         tmp = tmp->nextptr;
      }
    }
  }
}

void NodeInsertatEnd(int num)
{
  struct node *fnNode, *tmp;
  fnNode = (struct node*)malloc(sizeof(struct node));
  if(fnNode == NULL)
  {
     printf(" Memory can not be allocated.");
  }
  else
  {
     fnNode->num = num; //Links the data part
     fnNode->nextptr = NULL;
     tmp = stnode;
     while(tmp->nextptr != NULL)
       tmp = tmp->nextptr;
     tmp->nextptr = fnNode; //Links the address part
  }
}
```

```
void displayList()
{
  struct node *tmp;
  if(stnode == NULL)
  {
      printf(" No data found in the empty list.");
  }
  else
  {
      tmp = stnode;
      while(tmp != NULL)
      {
         printf(" Data = %d\n", tmp->num); // prints the data of current node
         tmp = tmp->nextptr;            // advances the position of current node
      }
  }
}
```

Testcase:

```
  Linked List : Insert a new node at the end of a Singly Linked List :
---------------------------------------------------------------------
  Input the number of nodes : 3
  Input data for node 1 : 10
  Input data for node 2 : 20
  Input data for node 3 : 30

Data entered in the list are :
Data = 10
Data = 20
Data = 30

Input data to insert at the end of the list : 5

Data, after inserted in the list are :
Data = 10
Data = 20
Data = 30
Data = 5
```

C.

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

struct node
{
   int num;              //Data of the node
   struct node *nextptr;     //Address of the node
}*stnode;

void createNodeList(int n);                              //function to create the list
void insertNodeAtMiddle(int num, int pos);              //function to insert node at the middle
void displayList();                                 //function to display the list

int main()
{
   int n,num,pos;
            printf("\n\n Linked List : Insert a new node at the middle of the Linked List
:\n");
            printf("-------------------------------------------------------------------\n");

  printf(" Input the number of nodes (3 or more) : ");
  scanf("%d", &n);
  createNodeList(n);
  printf("\n Data entered in the list are : \n");
  displayList();
  printf("\n Input data to insert in the middle of the list : ");
  scanf("%d", &num);
  printf(" Input the position to insert new node : " );
  scanf("%d", &pos);
    if(pos<=1 || pos>=n)
  {
   printf("\n Insertion can not be possible in that position.\n ");
  }
    if(pos>1 && pos<n)
   {
        insertNodeAtMiddle(num, pos);
    printf("\n Insertion completed successfully.\n ");
   }
  printf("\n The new list are : \n");
  displayList();
  return 0;
}
```

```
void createNodeList(int n)
{
   struct node *fnNode, *tmp;
   int num, i;
   stnode = (struct node *)malloc(sizeof(struct node));
   if(stnode == NULL) //check whether the stnode is NULL and if so no memory
allocation
   {
      printf(" Memory can not be allocated.");
   }
   else
   {
// reads data for the node through keyboard
      printf(" Input data for node 1 : ");
      scanf("%d", &num);
      stnode-> num = num;
      stnode-> nextptr = NULL; //Links the address field to NULL
      tmp = stnode;
//Creates n nodes and adds to linked list
      for(i=2; i<=n; i++)
      {
        fnNode = (struct node *)malloc(sizeof(struct node));
        if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory
allocation
        {
           printf(" Memory can not be allocated.");
           break;
        }
        else
        {
           printf(" Input data for node %d : ", i);
            scanf(" %d", &num);

            fnNode->num = num;     // links the num field of fnNode with num
            fnNode->nextptr = NULL; // links the address field of fnNode with NULL

            tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode
            tmp = tmp->nextptr;
         }
      }
   }
}
```

```
void insertNodeAtMiddle(int num, int pos)
{
  int i;
  struct node *fnNode, *tmp;
  fnNode = (struct node*)malloc(sizeof(struct node));
  if(fnNode == NULL)
  {
      printf(" Memory can not be allocated.");
  }
  else
  {
      fnNode->num = num; //Links the data part
      fnNode->nextptr = NULL;
      tmp = stnode;
      for(i=2; i<=pos-1; i++)
      {
         tmp = tmp->nextptr;

            if(tmp == NULL)
                break;
        }
        if(tmp != NULL)
        {
            fnNode->nextptr = tmp->nextptr; //Links the address part of new node
            tmp->nextptr = fnNode;
        }
        else
        {
            printf(" Insert is not possible to the given position.\n");
        }
    }
}

void displayList()
{
  struct node *tmp;
  if(stnode == NULL)
  {
      printf(" No data found in the empty list.");
  }
  else
  {
      tmp = stnode;
```

```
    while(tmp != NULL)
    {
      printf(" Data = %d\n", tmp->num); // prints the data of current node
      tmp = tmp->nextptr;            // advances the position of current node
    }
  }
}
```

Testcase:

```
  Linked List : Insert a new node at the middle of the Linked List :
-------------------------------------------------------------------------
  Input the number of nodes (3 or more) : 3
  Input data for node 1 : 10
  Input data for node 2 : 20
  Input data for node 3 : 30

Data entered in the list are :
Data = 10
Data = 20
Data = 30
Input data to insert in the middle of the list : 5
Input the position to insert new node : 2

Insertion completed successfully.

The new list are :
Data = 10
Data = 5
Data = 20
Data = 30
```

D.

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

struct node
{
   int num;              //Data of the node
   struct node *nextptr;     //Address of the node
}*stnode;
```

```
void createNodeList(int n); //function to create the list
void FirstNodeDeletion();      //function to delete the first node
void displayList();       //function to display the list

int main()
{
   int n,num,pos;
              printf("\n\n Linked List : Delete first node of Singly Linked List :\n");
              printf("-----------------------------------------------------------\n");
   printf(" Input the number of nodes : ");
   scanf("%d", &n);
   createNodeList(n);
   printf("\n Data entered in the list are : \n");
   displayList();
   FirstNodeDeletion();
   printf("\n Data, after deletion of first node : \n");
   displayList();
   return 0;
}
void createNodeList(int n)
{
    struct node *fnNode, *tmp;
    int num, i;
    stnode = (struct node *)malloc(sizeof(struct node));
    if(stnode == NULL)                          //check whether the stnode is NULL and if so
no memory allocation
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
// reads data for the node through keyboard
        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        stnode-> num = num;
        stnode-> nextptr = NULL; //Links the address field to NULL
        tmp = stnode;
//Creates n nodes and adds to linked list
        for(i=2; i<=n; i++)
        {
           fnNode = (struct node *)malloc(sizeof(struct node));
           if(fnNode == NULL)                          //check whether the fnnode is NULL and
```

```
if so no memory allocation
         {
            printf(" Memory can not be allocated.");
            break;
         }
         else
         {
            printf(" Input data for node %d : ", i);
            scanf(" %d", &num);
            fnNode->num = num;         // links the num field of fnNode with num
            fnNode->nextptr = NULL; // links the address field of fnNode with NULL
             tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode
            tmp = tmp->nextptr;
         }
       }
    }
}

void FirstNodeDeletion()
{
  struct node *toDelptr;
  if(stnode == NULL)
  {
     printf(" There are no node in the list.");
  }
  else
  {
     toDelptr = stnode;
     stnode = stnode->nextptr;
     printf("\n Data of node 1 which is being deleted is : %d\n", toDelptr->num);
     free(toDelptr); // Clears the memory occupied by first node
  }
}



void displayList()
{
  struct node *tmp;
  if(stnode == NULL)
  {
     printf(" No data found in the list.");
  }
```

```
  else
  {
      tmp = stnode;
      while(tmp != NULL)
      {
         printf(" Data = %d\n", tmp->num); // prints the data of current node
         tmp = tmp->nextptr;               // advances the position of current node
      }
  }
}
```
Testcase:

```
  Linked List : Delete first node of Singly Linked List :
-------------------------------------------------------------
  Input the number of nodes : 3
  Input data for node 1 : 10
  Input data for node 2 : 20
  Input data for node 3 : 30

Data entered in the list are :
Data = 10
Data = 20
Data = 30

Data of node 1 which is being deleted is : 10

Data, after deletion of first node :
Data = 20
Data = 30
```

E.

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

struct node
{
   int num;              //Data of the node
   struct node *nextptr;     //Address of the node
}*stnode;

void createNodeList(int n); //function to create the list
void LastNodeDeletion();      //function to delete the last nodes
void displayList();       //function to display the list
```

```
int main()
{
   int n,num,pos;
              printf("\n\n Linked List : Delete the last node of Singly Linked List :\n");
              printf("------------------------------------------------------------\n");
  printf(" Input the number of nodes : ");
  scanf("%d", &n);
  createNodeList(n);

  printf("\n Data entered in the list are : \n");
  displayList();
  LastNodeDeletion();
          printf("\n The new list after deletion the last node are : \n");
  displayList();
  return 0;
}
void createNodeList(int n)
{
  struct node *fnNode, *tmp;
  int num, i;

    stnode = (struct node *)malloc(sizeof(struct node));
    if(stnode == NULL) //check whether the stnode is NULL and if so no memory
allocation
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
// reads data for the node through keyboard
        printf(" Input data for node 1 : ");
        scanf("%d", &num);

    stnode-> num = num;
    stnode-> nextptr = NULL; //Links the address field to NULL
    tmp = stnode;

//Creates n nodes and adds to linked list
     for(i=2; i<=n; i++)
     {
        fnNode = (struct node *)malloc(sizeof(struct node));
        if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory
```

```
allocation
        {
            printf(" Memory can not be allocated.");
            break;
        }
        else
        {
           printf(" Input data for node %d : ", i);
           scanf(" %d", &num);
           fnNode->num = num;         // links the num field of fnNode with num
           fnNode->nextptr = NULL; // links the address field of fnNode with NULL
           tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode
           tmp = tmp->nextptr;
        }
     }
  }
}
// Deletes the last node of the linked list
void LastNodeDeletion()
{
  struct node *toDelLast, *preNode;
  if(stnode == NULL)
  {
     printf(" There is no element in the list.");
  }
  else
  {
     toDelLast = stnode;
     preNode = stnode;
     /* Traverse to the last node of the list*/
     while(toDelLast->nextptr != NULL)
     {
        preNode = toDelLast;
        toDelLast = toDelLast->nextptr;
     }
     if(toDelLast == stnode)
     {
        stnode = NULL;
     }
     else
    {

        /* Disconnects the link of second last node with last node */
```

```
        preNode->nextptr = NULL;
      }

      /* Delete the last node */
      free(toDelLast);
  }
}
// function to display the entire list
void displayList()
{
    struct node *tmp;
    if(stnode == NULL)
    {
        printf(" No data found in the empty list.");
    }
    else
    {
        tmp = stnode;
        while(tmp != NULL)
        {
            printf(" Data = %d\n", tmp->num); // prints the data of current node
            tmp = tmp->nextptr;             // advances the position of current node
        }
    }
}
```

Testcase:

```
  Linked List : Delete the last node of Singly Linked List :
-------------------------------------------------------------
  Input the number of nodes : 3
  Input data for node 1 : 10
  Input data for node 2 : 20
Input data for node 3 : 30

Data entered in the list are :
Data = 10
Data = 20
Data = 30
```

The new list after deletion the last node are :
Data = 10
Data = 20

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 in C(code):

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

struct node {
  int num;
  struct node * preptr;
  struct node * nextptr;
```

```
}*stnode, *ennode;



void DlListcreation(int n);
void displayDlList();

int main()
{
   int n;
   stnode = NULL;
   ennode = NULL;
        printf("\n\n Doubly Linked List : Create and display a doubly linked list :\n");
        printf("-------------------------------------------------------------------\n");

    printf(" Input the number of nodes : ");
    scanf("%d", &n);

    DlListcreation(n);
    displayDlList();
    return 0;
}

void DlListcreation(int n)
{
  int i, num;
  struct node *fnNode;

    if(n >= 1)
    {
        stnode = (struct node *)malloc(sizeof(struct node));
      if(stnode != NULL)
      {
          printf(" Input data for node 1 : "); // assigning data in the first node
          scanf("%d", &num);

         stnode->num = num;
         stnode->preptr = NULL;
         stnode->nextptr = NULL;
         ennode = stnode;
// putting data for rest of the nodes
         for(i=2; i<=n; i++)
         {
```

```
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode != NULL)
            {
                printf(" Input data for node %d : ", i);
                scanf("%d", &num);
                fnNode->num = num;
                fnNode->preptr = ennode; // new node is linking with the previous node
                fnNode->nextptr = NULL;

              ennode->nextptr = fnNode; // previous node is linking with the new node
              ennode = fnNode;       // assign new node as last node
            }
            else
            {
               printf(" Memory can not be allocated.");
               break;
            }
        }
      }
      else
      {
         printf(" Memory can not be allocated.");
      }
  }
}
void displayDlList()
{
    struct node * tmp;
    int n = 1;
    if(stnode == NULL)
    {
        printf(" No data found in the List yet.");
    }
    else
    {
        tmp = stnode;
        printf("\n\n Data entered on the list are :\n");

        while(tmp != NULL)
        {
          printf(" node %d : %d\n", n, tmp->num);
          n++;
          tmp = tmp->nextptr; // current pointer moves to the next node
```

```
        }
    }
}
```

Testcase:

```
  Doubly Linked List : Create and display a doubly linked list :
-------------------------------------------------------------------
  Input the number of nodes : 5
  Input data for node 1 : 10
  Input data for node 2 : 20
  Input data for node 3 : 30
  Input data for node 4 : 40
  Input data for node 5 : 50



Data entered on the list are :
node 1 : 10
node 2 : 20
node 3 : 30
node 4 : 40
node 5 : 50
```

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 in C(code):

```
#include <stdio.h>
#include <stdlib.h>
/*
 * Basic structure of Node
 */
struct node {
    int data;
    struct node * next;
}*head;



/*
 * Functions used in this program
 */
void createList(int n);
void displayList();



int main()
{
   int n, data, choice=1;
```

```
  head = NULL;

  /*
   * Run forever until user chooses 0
   */
  while(choice != 0)
  {
      printf("==============================================\n");
      printf("CIRCULAR LINKED LIST PROGRAM\n");
      printf("==============================================\n");
      printf("1. Create List\n");
      printf("2. Display list\n");
      printf("0. Exit\n");
      printf("----------------------------------------------\n");
      printf("Enter your choice : ");

     scanf("%d", &choice);

     switch(choice)
       {
          case 1:
            printf("Enter the total number of nodes in list: ");
            scanf("%d", &n);
            createList(n);
            break;
          case 2:
            displayList();
            break;
          case 0:
            break;
          default:
            printf("Error! Invalid choice. Please choose between 0-2");
       }

       printf("\n\n\n\n\n");
    }

    return 0;
}

void createList(int n)
{
```

```
  int i, data;
  struct node *prevNode, *newNode;

    if(n >= 1)
    {
            for(i=2; i<=n; i++)
        {
          newNode = (struct node *)malloc(sizeof(struct node));

            printf("Enter data of %d node: ", i);
            scanf("%d", &data);

            newNode->data = data;
            newNode->next = NULL;

            // Link the previous node with newly created node
            prevNode->next = newNode;

            // Move the previous node ahead
            prevNode = newNode;
        }

        // Link the last node with first node
        prevNode->next = head;

        printf("\nCIRCULAR LINKED LIST CREATED SUCCESSFULLY\n");
    }
}



/**
 * Display the content of the list
 */
void displayList()
{
    struct node *current;
    int n = 1;

    if(head == NULL)
    {
        printf("List is empty.\n");
    }
```

```
    else
    {
        current = head;
        printf("DATA IN THE LIST:\n");

        do {
          printf("Data %d = %d\n", n, current->data);

          current = current->next;
          n++;
        }while(current != head);
    }
}
```

Testcase:

```
==============================================
CIRCULAR LINKED LIST PROGRAM
==============================================
1. Create List
2. Display list
0. Exit
--------------------------------------------
Enter your choice : 1
Enter the total number of nodes in list: 5
Enter data of 1 node: 10
Enter data of 2 node: 20
Enter data of 3 node: 30
Enter data of 4 node: 40
Enter data of 5 node: 50

CIRCULAR LINKED LIST CREATED SUCCESSFULLY




==============================================
CIRCULAR LINKED LIST PROGRAM
==============================================
1. Create List
2. Display list
0. Exit
--------------------------------------------
Enter your choice : 2
```

DATA IN THE LIST:
Data 1 = 10
Data 2 = 20
Data 3 = 30
Data 4 = 40
Data 5 = 50
==
CIRCULAR LINKED LIST PROGRAM
==
1. Create List
2. Display list
0. Exit

Enter your choice : 0

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 in C(code):

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0

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

node *top;

void initialize()
{
  top = NULL;
}

void push(int value)
{
  node *tmp;
  tmp = malloc(sizeof(node));
  tmp -> data = value;
  tmp -> next = top;
  top = tmp;
}

int pop()
{
   node *tmp;
   int n;
```

```
   tmp = top;
   n = tmp->data;
   top = top->next;
   free(tmp);
   return n;
}

int Top()
{
   return top->data;
}

int isempty()
{
   return top==NULL;
}

void display(node *head)
{
  if(head == NULL)
  {
     printf("NULL\n");
  }
  else
  {
     printf("%d\n", head -> data);
     display(head->next);
  }
}

int main()
{
   initialize();
    push(10);
    push(20);
    push(30);
    printf("The top is %d\n",Top());
    pop();
    printf("The top after pop is %d\n",Top());
    display(top);
    return 0;
}
```

Testcase:

The top is 30
The top after pop is 20
20
10
NULL

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'.

Program in C(code):

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

struct Node
{
  int data;
  struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();
void main()
{
  int choice, value;

  printf("\n:: Queue Implementation using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d", &value);
                 insert(value);
                 break;
         case 2: delete(); break;
```

```
        case 3: display(); break;
        case 4: exit(0);
        default: printf("\nWrong selection!!! Please try again!!!\n");
    }
  }
}
void insert(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode -> next = NULL;
  if(front == NULL)
     front = rear = newNode;
  else{
     rear -> next = newNode;
     rear = newNode;
  }
  printf("\nInsertion is Success!!!\n");
}
void delete()
{
  if(front == NULL)
   printf("\nQueue is Empty!!!\n");
 else{
   struct Node *temp = front;
   front = front -> next;
   printf("\nDeleted element: %d\n", temp->data);
   free(temp);
 }
}
void display()
{
  if(front == NULL)
     printf("\nQueue is Empty!!!\n");
  else{
     struct Node *temp = front;
     while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
     }
     printf("%d--->NULL\n",temp->data);
  }
```

}

Testcase:

Insertion is Success!!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2

Deleted element: 10

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
20--->30--->NULL

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4