1. **Aim: Write a C Program to Sort the Array in an Ascending Order using Bubble sort.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to sort an array using bubble sorting.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES : i, temp, j, num, sort[]
INPUT: Read input from the keyboard
COMPUTATION: Using Bubble sort, it sorts the inputted numbers
DISPLAY: Sorted numbers
STOP

**Program in C :**

```c
#include <stdio.h>
void main()
{
   int i, temp, j, num;
   printf("Enter number of elements : ");
   scanf("%d", &num);
   int sort[num];
   for(i = 0; i < num; i++){
```

```
    printf("Enter an element : ");
    scanf("%d", &sort[i]);
  }
  for(i = 0; i < num; i++){
    for(j = 0; j < num - 1; j++){
      if(sort[j] > sort[j+1]){
        temp = sort[j];
        sort[j] = sort[j+1];
        sort[j+1] = temp;
      }
    }
  }
  for(i = 0; i < num; i++){
    printf("%d  ",sort[i]);
  }
}
```

**Output:**

```
Enter number of elements : 5
Enter an element : 5
Enter an element : 4
Enter an element : 3
Enter an element : 2
Enter an element : 1
1  2  3  4  5
```

2. **Aim: Write a C Program to sort an array in descending order using Selection sort.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to sort an array in descending order using selection sort.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: i, temp, j, num, min_index
INPUT: Read the input from the keyboard
COMPUTATION: Using Selection sort, it sorts the inputted numbers
DISPLAY: Sorted numbers
STOP

**Program in C :**

```c
#include <stdio.h>
void main()
{
   int i, temp, j, num, min_index;
   printf("Enter number of elements : ");
   scanf("%d", &num);
   int sort[num];
   for(i = 0; i < num; i++){
      printf("Enter an element : ");
```

```
    scanf("%d", &sort[i]);
  }
  for(i = 0; i < num - 1; i++){
    min_index = i;
    for(j = i+1; j < num; j++){
      if(sort[j] < sort[min_index]){
        min_index = j;
      }
    }
    temp = sort[min_index];
    sort[min_index] = sort[i];
    sort[i] = temp;
  }
  for(i = num - 1; i <= 0; i--){
    printf("%d  ",sort[i]);
  }
}
```

**Output:**

Enter number of elements : 5
Enter an element : 64
Enter an element : 12
Enter an element : 22
Enter an element : 10
Enter an element : 25
64  25  22  12  10

3. **Aim: Write a C Program to sort an array in ascending order using Insertion sort.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to sort an array in ascending order using Insertion sort.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: i, temp, j, num, key
INPUT: Read the input from the keyboard
COMPUTATION: Using Insertion sort, it sorts the inputted numbers
DISPLAY: Sorted numbers
STOP

**Program in C :**

```c
#include <stdio.h>
void main()
{
   int i, temp, j, num, key;
   printf("Enter number of elements : ");
   scanf("%d", &num);
   int sort[num];
   for(i = 0; i < num; i++){
```

```
    printf("Enter an element : ");
    scanf("%d", &sort[i]);
  }
  for(i = 1; i < num; i++){
    key = sort[i];
    j = i - 1;
    while (j >= 0 && sort[j] > key){
      sort[j + 1] = sort[j];
      j = j - 1;
    }
    sort[j + 1] = key;
  }
  for(i = 0; i < num; i++){
    printf("%d  ",sort[i]);
  }
}
```

**Output:**

Enter number of elements : 6
Enter an element : 7
Enter an element : 8
Enter an element : 6
Enter an element : 4
Enter an element : 5
Enter an element : 2
2  4  5  6  7  8

4. **Aim: Write a C Program for Binary search.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to sort an array using bubble sorting.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START

DEFINING VARIABLES: i, num, search, ans, L, R, mid

INPUT: Read the input from the keyboard

COMPUTATION: Uses Binary Search algorithm to search the given element

DISPLAY: The index of the given element if it exists

STOP

**Program in C :**

```c
#include <stdio.h>
int BinarySearch(int array[], int L, int R, int search);
void main()
{
  int i, num, search;
  printf("Enter number of elements : ");
  scanf("%d", &num);
```

```
  int array[num];
  for(i = 0; i < num; i++){
    printf("Enter an element : ");
    scanf("%d", &array[i]);
  }
  printf("Enter the element to search: ");
  scanf("%d", &search);
  int ans = BinarySearch(array, 0, num - 1, search);
  if (ans == -1){
    printf("The entered element is not found in the given array.");
  }
  else{
    printf("The entered element is found in the array at the index %d.", ans);
  }
}

int BinarySearch(int array[], int L, int R, int search){
  while(L <= R){
    int mid = L + (R - L) / 2;
    if (array[mid] == search){
      return mid;
    }
    if (array[mid] < search){
      L = mid + 1;
    }
    else{
      R = mid - 1;
    }
  }
```

```
  return -1;
}
```

**Output:**

Enter number of elements : 5

Enter an element : 6

Enter an element : 21

Enter an element : 37

Enter an element : 59

Enter an element : 99

Enter the element to search: 21

The entered element is found in the array at the index 1.

5. **Aim: Write a C program to Create a binary tree and output the data with 3 tree traversals.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to create a binary tree and output the data with 3 tree traversals

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: root
INPUT: Read the input from the keyboard
COMPUTATION: Traversals a binary tree in three different ways
DISPLAY: Prints Inorder, Preorder, Postorder traversals
STOP

**Program in C :**

```c
#include <stdio.h>

typedef struct node{
  int data;
  struct node* left;
```

```
  struct node* right;
}node;

node* NewNode(int data);
void Inorder(node* Node);
void Preorder(node* Node);
void Postorder(node* Node);

void main(){
  node* root = NewNode(1);
  root->left = NewNode(2);
  root->right = NewNode(3);
  root->left->left = NewNode(4);
  root->left->right = NewNode(5);

  printf("Inorder traversal: ");
  Inorder(root);
  printf("\n\nPreorder traversal: ");
  Preorder(root);
  printf("\n\nPostorder traversal: ");
  Postorder(root);
}

node* NewNode(int data){
  node* Node = (node*)malloc(sizeof(node));
  Node-> data = data;
  Node->left = NULL;
  Node->right = NULL;
  return Node;
```

```
}

void Inorder(node* Node){
   if (Node == NULL){
      return;
   }
   Inorder(Node->left);
   printf("-> %d ", Node->data);
   Inorder(Node->right);
}

void Preorder(node* Node){
   if (Node == NULL){
      return;
   }
   printf("-> %d ", Node->data);
   Preorder(Node->left);
   Preorder(Node->right);
}

void Postorder(node* Node){
   if (Node == NULL){
      return;
   }
   Postorder(Node->left);
   Postorder(Node->right);
   printf("-> %d ", Node->data);
}
```

**Srilahari Sivanvitha Nori**

**Output:**

Inorder traversal: -> 4 -> 2 -> 5 -> 1 -> 3

Preorder traversal: -> 1 -> 2 -> 4 -> 5 -> 3

Postorder traversal: -> 4 -> 5 -> 2 -> 3 -> 1

6. **Aim: Write a C program to Create a Binary Search Tree(BST) and search for a given value in BST.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to create a Binary Search Tree(BST) and search for a given value in BST.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START

DEFINING VARIABLES: root, search, data, option

INPUT: Read the input from the keyboard

COMPUTATION: Creates a BST and searches for required element

DISPLAY: Inorder traversal and prints whether the element is there in the BST or not

STOP

**Program in C :**

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

```c
#include <stdlib.h>

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

node* NewNode(int data);
node* InsertNode(node* root, int data);
node* SearchTree(node* root, int search);
void Inorder(node* Node);

void main(){
   node* root = NULL;
   int option, data, search;
   while(1){
      printf("\n1. To insert an element into the BST.");
      printf("\n2. To search for an element in the BST.");
      printf("\n3. To print Inorder traversal of BST.");
      printf("\n4. Exit.");
      printf("\nEnter your choice: ");
      scanf("%d", &option);
      switch(option){
         case 1: printf("\nEnter an element to insert into the BST: ");
            scanf("%d", &data);
            if (root == NULL){
               root = InsertNode(root, data);
            }
```

```
            else{
               InsertNode(root, data);
            }
            break;
         case 2: printf("\nEnter an element to search: ");
            scanf("%d", &search);
            SearchTree(root, search);
            break;
         case 3: printf("\nInorder Traversal: ");
            Inorder(root);
            printf("\n");
            break;
         case 4: exit(0);
         default: printf("\nEnter a valid option!!!");
      }
   }
}

node* NewNode(int data){
   node* temp = (node*)malloc(sizeof(node));
   temp->data = data;
   temp->left = temp->right = NULL;
   return temp;
}

node* InsertNode(node* Node, int data){
   if (Node == NULL){
      return NewNode(data);
   }
```

```
  if (data < Node->data){
    Node->left = InsertNode(Node->left, data);
  }
  else if (data > Node->data){
    Node->right = InsertNode(Node->right, data);
  }
  return Node;
}

node* SearchTree(node* Node, int search){
  if (Node == NULL){
    printf("\nThe entered element is not found in the BST!!!\n");
    return;
  }
  if (Node->data == search){
    printf("\nThe element %d is found in the BST!!!\n", search);
    return;
  }
  if (search < Node->data){
    SearchTree(Node->left, search);
  }
  else if (search > Node->data){
    SearchTree(Node->right, search);
  }
}

void Inorder(node* Node){
  if (Node == NULL){
    return;
```

```
  }
  Inorder(Node->left);
  printf("-> %d ", Node->data);
  Inorder(Node->right);
}
```

**Output:**

1. To insert an element into the BST.
2. To search for an element in the BST.
3. To print Inorder traversal of BST.
4. Exit.

Enter your choice: 1

Enter an element to insert into the BST: 11

1. To insert an element into the BST.
2. To search for an element in the BST.
3. To print Inorder traversal of BST.
4. Exit.

Enter your choice: 1

Enter an element to insert into the BST: 39

1. To insert an element into the BST.
2. To search for an element in the BST.
3. To print Inorder traversal of BST.
4. Exit.

Enter your choice: 1

Enter an element to insert into the BST: 54

1. To insert an element into the BST.
2. To search for an element in the BST.
3. To print Inorder traversal of BST.
4. Exit.

Enter your choice: 1

Enter an element to insert into the BST: 20

1. To insert an element into the BST.
2. To search for an element in the BST.
3. To print Inorder traversal of BST.
4. Exit.

Enter your choice: 1

Enter an element to insert into the BST: 93

1. To insert an element into the BST.
2. To search for an element in the BST.
3. To print Inorder traversal of BST.
4. Exit.

Enter your choice: 1

Enter an element to insert into the BST: 8

1. To insert an element into the BST.
2. To search for an element in the BST.

3. To print Inorder traversal of BST.

4. Exit.

Enter your choice: 3

Inorder Traversal: -> 8 -> 11 -> 20 -> 39 -> 54 -> 93

1. To insert an element into the BST.

2. To search for an element in the BST.

3. To print Inorder traversal of BST.

4. Exit.

Enter your choice: 2

Enter an element to search: 8

The element 8 is found in the BTS!!!

1. To insert an element into the BST.

2. To search for an element in the BST.

3. To print Inorder traversal of BST.

4. Exit.

Enter your choice: 4

7. **Aim: Write a C program to find All-to-all Shortest Paths in a Graph.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to find All-to-all Shortest Paths in a Graph.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START

DEFINING VARIABLES: V, i, j, k, graph

INPUT: Read the input from the keyboard

COMPUTATION: Finds the shortest path between two vertices

DISPLAY: A matrix with All-to-all Shortest Paths in a Graph

STOP

**Program in C :**

```c
#include <stdio.h>
#include <stdlib.h>
#define MIN(x, y) (((x) < (y)) ? (x) : (y))
int V;

void PrintGraph(int graph[V][V]);
void ShortestPath(int graph[V][V]);
```

```
int main() {
   printf("Enter number of vertices: ");
   scanf("%d", &V);
   int graph[V][V];
   printf("Enter '99999' for infinity!!\n");
   for (int i = 0; i < V; i++){
      for(int j = 0; j < V; j++){
         printf("Enter the weight of the edge[%d][%d]: ", i, j);
         scanf("%d", &graph[i][j]);
      }
   }
   ShortestPath(graph);
   return;
}

void ShortestPath(int graph[V][V]){
   for(int k = 0; k < V; k++){
      for(int j = 0; j < V; j++){
         for(int i = 0; i < V; i++){
            graph[i][j] = MIN(graph[i][j], graph[i][k] + graph[k][j]);
         }
      }
   }
   PrintGraph(graph);
}

void PrintGraph(int graph[V][V]){
   printf("All to All Shortest paths matrix \n");
   for (int i = 0; i < V; i++){
```

```
    for (int j = 0; j < V; j++){
      if (graph[i][j] == 99999){
        printf(" INF ");
      }
      else{
        printf("  %d  ", graph[i][j]);
      }
    }
    printf("\n");
  }
}
```

**Output:**

Enter number of vertices: 4

Enter '99999' for infinity!!

Enter the weight of the edge[0][0]: 0

Enter the weight of the edge[0][1]: 5

Enter the weight of the edge[0][2]: 99999

Enter the weight of the edge[0][3]: 10

Enter the weight of the edge[1][0]: 99999

Enter the weight of the edge[1][1]: 0

Enter the weight of the edge[1][2]: 3

Enter the weight of the edge[1][3]: 99999

Enter the weight of the edge[2][0]: 99999

Enter the weight of the edge[2][1]: 99999

Enter the weight of the edge[2][2]: 0

Enter the weight of the edge[2][3]: 1

Enter the weight of the edge[3][0]: 99999

Enter the weight of the edge[3][1]: 99999

```
Enter the weight of the edge[3][2]: 99999
Enter the weight of the edge[3][3]: 0
All to All Shortest paths matrix
 0   5   8   9
INF  0   3   4
INF INF  0   1
INF INF INF  0
```

8. **Aim: Write a C program to implement the STACK operation using an array as a data structure. Users must be given the following choices to perform the relevant tasks.**
    1. **Push an element on to the STACK.**
    2. **Pop and element from the STACK.**
    3. **Peek the STACK.**
    4. **Display the STACK.**
    5. **Exit the program.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to implement the STACK operation using an array as a data structure.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: SIZE, choice, value, STACK
INPUT: Read the input from the keyboard
COMPUTATION: Creates a stack and performs the required functions
DISPLAY: STACK
STOP

**Program in C :**

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

```c
#include <stdlib.h>
#define SIZE 10
void push(int);
void pop();
void peek();
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 \n4. Display \n5. 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 peek(){
    if (top == -1){
        printf("\nStack is Empty!!!");
```

```
  }
  else{
    printf("PEEK VALUE = %d", stack[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\t", stack[i]);
  }
}
```

**Output:**

*** 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: 3

Stack is Empty!!!

*** MENU ***

1. Push

2. Pop

3. Peek

4. Display

5. Exit

Enter your choice: 2

Stack is Empty!!! Deletion is not possible!!!

*** MENU ***

1. Push

2. Pop

3. Peek

4. Display

5. Exit

Enter your choice: 1

Enter the value to be insert: 753

Insertion success!!!

*** MENU ***

1. Push
2. Pop
3. Peek
4. Display
5. Exit

Enter your choice: 1

Enter the value to be insert: 9541

Insertion success!!!

*** MENU ***

1. Push
2. Pop
3. Peek
4. Display
5. Exit

Enter your choice: 1

Enter the value to be insert: 4

Insertion success!!!

*** MENU ***

1. Push
2. Pop
3. Peek
4. Display

5. Exit

Enter your choice: 1

Enter the value to be insert: 8

Insertion success!!!

*** MENU ***

1. Push

2. Pop

3. Peek

4. Display

5. Exit

Enter your choice: 3

PEEK VALUE = 8

*** MENU ***

1. Push

2. Pop

3. Peek

4. Display

5. Exit

Enter your choice: 4

Stack elements are:

8 4 9541 753

*** MENU ***

1. Push

2. Pop

3. Peek
4. Display
5. Exit

Enter your choice: 2

Deleted : 8

*** MENU ***
1. Push
2. Pop
3. Peek
4. Display
5. Exit

Enter your choice: 2

Deleted : 4

*** MENU ***
1. Push
2. Pop
3. Peek
4. Display
5. Exit

Enter your choice: 2

Deleted : 9541

*** MENU ***
1. Push

2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 2

Deleted : 753

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

Stack is Empty!!! Deletion is not possible!!!

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

9. **Aim: Write a C program to reverse a string using STACK.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to reverse a string using STACK.

**Algorithm:**

START

DEFINING VARIABLES: choice, value, SIZE, TOP

INPUT: Reads the input from the keyboard

COMPUTATION: Uses stack operations to reverse a string

DISPLAY: Displays the reversed string

STOP

**Program in C :**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 10
void push(char);
void pop();
void display();
int top = -1;
char stack[SIZE];
void main()
{
   int choice;
```

```
  char value;
  while(1)
  {
    printf("\n\n*** MENU ***\n");
    printf("1. Push \n2. Pop \n3. Display \n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d", &choice);
    getchar();
    switch(choice)
    {
      case 1: printf("Enter the value to be insert: ");
        value = getchar();
        push(value);
        break;
      case 2: pop();
        break;
      case 3: display();
        break;
      case 4: exit(0);
        default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}
void push(char 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 : %c", 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("%c ", stack[i]);
  }
}
```

**Output:**

*** MENU ***

1. Push

2. Pop

3. Display

4. Exit

Enter your choice: 1

Enter the value to be insert: s

Insertion success!!!

*** MENU ***

1. Push

2. Pop

3. Display

4. Exit

Enter your choice: 1

Enter the value to be insert: t

Insertion success!!!

*** MENU ***

1. Push

2. Pop

3. Display

4. Exit

Enter your choice: 1

Enter the value to be insert: r

Insertion success!!!

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

Insertion success!!!

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

Insertion success!!!

*** MENU ***
1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1

Enter the value to be insert: g

Insertion success!!!

*** MENU ***

1. Push

2. Pop

3. Display

4. Exit

Enter your choice: 3

Stack elements are:

g n i r t s

*** MENU ***

1. Push

2. Pop

3. Display

4. Exit

Enter your choice: 4

10. **Aim: Write a C program to convert the given infix expression to postfix expression using STACK.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to convert the given infix expression to postfix expression using STACK.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: top, STACK, exp[], *ele, x
INPUT: Reads input from the keyboard
COMPUTATION: Converts Infix expression to Postfix expression
DISPLAY: Postfix expression
STOP

**Program in C :**

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

void Push(char x);
char Pop();
int Priority(char x);

int top = -1;
char STACK[100];
```

```
void main() {
   char exp[100], *ele, x;
   printf("Enter the Infix expression: ");
   scanf("%s", exp);
   ele = exp;
   printf("The Postfix expression: ");
   while (*ele != '\0') {
      if (isalnum(*ele)) {
         printf("%c", *ele);
      }
      else if (*ele == '(') {
         Push(*ele);
      }
      else if (*ele == ')') {
         while ((x = Pop()) != '(') {
            printf("%c", x);
         }
      }
      else {
         while(Priority(STACK[top]) >= Priority(*ele)) {
            printf("%c", Pop());
         }
         Push(*ele);
      }
      ele++;
   }
   while(top != -1) {
      printf("%c", Pop());
   }
}

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;
  }
}
```

**Output:**

Enter the Infix expression: (a+b)*c+(d-a)

The Postfix expression: ab+c*da-+

11. **Aim: Write a C program to convert the given infix expression to prefix expression using STACK.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to convert the given infix expression to prefix expression using STACK.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: top, TOP, STACK, PREFIX, exp, rev_exp, *ele, x, i, j
INPUT: Reads input from the keyboard
COMPUTATION: Converts Infix expression to Prefix expression
DISPLAY: Prefix expression
STOP

**Program in C :**

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

void Push(char x);
char Pop();
void Push_Prefix(char x);
char Pop_Prefix();
```

```
int Priority(char x);

int top = -1, TOP = -1;
char STACK[100], PREFIX[100];
void main() {
   int i, j;
   char exp[100], rev_exp[100], *ele, x;
   printf("Enter the Infix expression: ");
   scanf("%s", exp);
   for (i = strlen(exp) - 1, j = 0; i + 1 != 0; i--, j++) {
      if (exp[i] == '(') {
         rev_exp[j] = ')';
      }
      else if (exp[i] == ')') {
         rev_exp[j] = '(';
      }
      else {
         rev_exp[j] = exp[i];
      }
   }
   ele = rev_exp;
   printf("The Prefix expression: ");
   while (*ele != '\0') {
      if (isalnum(*ele)) {
         Push_Prefix(*ele);
      }
      else if (*ele == '(') {
         Push(*ele);
      }
      else if (*ele == ')') {
         while ((x = Pop()) != '(') {
            Push_Prefix(x);
         }
      }
      else {
         while(Priority(STACK[top]) >= Priority(*ele)) {
            Push_Prefix(Pop());
```

```
        }
        Push(*ele);
      }
      ele++;
    }
    while(top != -1) {
      Push_Prefix(Pop());
    }
    while (TOP != -1) {
      printf("%c", Pop_Prefix());
    }

}

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;
  }
```

```
  if(x == '^') {
    return 3;
  }
}

void Push_Prefix(char x) {
  PREFIX[++TOP] = x;
}
char Pop_Prefix() {
  return PREFIX[TOP--];
}
```

**Output:**

Enter the Infix expression: (A+B^C)*D+E^5

The Prefix expression: +*+A^BCD^E5

12. **Aim: Write a C program to evaluate the given prefix expression, post-fix expression.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to evaluate the given prefix expression, post-fix expression.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: top, STACK, choice, exp
INPUT: Reads the input from the keyboard
COMPUTATION: Evaluates the prefix and postfix expressions
DISPLAY: The result of the given expression
STOP

**Program in C :**

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

void Push(char x);
char Pop();
```

```
void Prefix(char prefix[]);
void Postfix(char postfix[]);

int top = -1;
int STACK[100];
void main() {
   int choice;
   char exp[100];
   printf("\n1. Prefix \n2. Postfix \nEnter you choice: ");
   scanf("%d", &choice);
   if (choice == 1){
      printf("Enter the Prefix expression: ");
      scanf("%s", exp);
      Prefix(exp);
   }
   else if (choice == 2) {
      printf("Enter the Postfix expression: ");
      scanf("%s", exp);
      Postfix(exp);
   }
}

void Push(char x) {
   STACK[++top] = x;
}

char Pop() {
   if (top == -1) {
      return -1;
   }
   else{
      return STACK[top--];
   }
}

void Prefix(char prefix[]) {
   int i, a, b, temp, result;
```

```
  printf("The evaluation Prefix expression: ");
  for (i = strlen(prefix) -1; i >= 0; i--) {
    if (prefix[i] <= '9' && prefix[i] >= '0') {
      Push(prefix[i] -48);
    }
    else {
      b = Pop();
      a = Pop();
      switch (prefix[i]){
        case '+': temp = b +a;
          break;
        case '-': temp = b -a;
          break;
        case '*': temp = b *a;
          break;
        case '/': temp = b /a;
          break;
        case '^': temp = pow(b, a);
      }
      Push(temp);
    }
  }
  printf("%d", Pop());
}

void Postfix(char postfix[]) {
  int i, a, b, temp, result;
  printf("The evaluation Postfix expression: ");
  for (i = 0; i < strlen(postfix); i--) {
    if (postfix[i] <= '9' && postfix[i] >= '0') {
      Push(postfix[i] -'0');
    }
    else {
      a = Pop();
      b = Pop();
      switch (postfix[i]){
        case '+': temp = b +a;
```

```
            break;
          case '-': temp = b -a;
            break;
          case '*': temp = b *a;
            break;
          case '/': temp = b /a;
            break;
          case '^': temp = pow(b, a);
        }
        Push(temp);
      }
    }
    printf("%d", Pop());
}
```

**Output:**

1. Prefix
2. Postfix

Enter you choice: 1

Enter the Prefix expression: -+8/632

The evaluation Prefix expression: 8

13. **Aim: Write a C program to implement a Linear-Queue, a user must choose the following options:**
    1. **Add an element to the Queue – EnQueue.**
    2. **Remove an element from the Queue – DeQueue.**
    3. **Display the elements of the Queue.**
    4. **Terminate the program.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to implement a Linear-Queue.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: SIZE, front, rear, queue[], value, choice
INPUT: Reads input from the keyboard
COMPUTATION: Performs Queue operations with given numbers
DISPLAY: Displays the output value after operations
STOP

**Program in C :**

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100
void enQueue(int);
```

```
void deQueue();
void display();
void peek();
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(rear == SIZE-1)
    printf("\nQueue is Full!!! Insertion is not possible!!!");
  else {
```

```
    if(front == -1)
        front = 0;
    rear++;
    queue[rear] = value;
    printf("\nInserted -> %d", value);
  }
}

void deQueue(){
  if(front == -1)
    printf("\nQueue is Empty!!! Deletion is not possible!!!");
  else{
    printf("\nDeleted : %d", queue[front]);
    front++;
    if(front == rear)
      front = rear = -1;
  }
}

void display(){
  if(rear == -1)
    printf("\nQueue is Empty!!!");
  else{
    int i;
    printf("\nQueue elements are:\n");
    for(i = front; i <= rear; i++)
      printf("%d\t", queue[i]);
  }
}
```

**Output:**

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 2

Queue is Empty!!! Deletion is not possible!!!

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 3

Queue is Empty!!!

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 1

Enter the value to be insert: 88

```
Inserted -> 88

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 45

Inserted -> 45

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 51

Inserted -> 51

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

Queue elements are:

88 45 51

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 2

Deleted : 88

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 2

Deleted : 45

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 2

Queue is Empty!!! Deletion is not possible!!!

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 3

Queue is Empty!!!

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 4

14. **Aim: Write a C program to implement a Circular-Queue, a user must choose the following options:**
    1. **Add an element to the Queue – EnQueue.**
    2. **Remove an element from the Queue – DeQueue.**
    3. **Display the elements of the Queue.**
    4. **Terminate the program.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to implement a Circular-Queue.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: SIZE, front, rear, queue[], value, choice
INPUT: Reads input from the keyboard
COMPUTATION: Performs Circular Queue operations with given numbers
DISPLAY: Displays the output value after operations
STOP

**Program in C :**

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 3
void enQueue(int);
void deQueue();
```

```
void display();
int queue[SIZE], 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 == rear + 1) || (front == 0 && rear == SIZE-1))
    printf("\nQueue is Full!!! Insertion is not possible!!!");
  else {
    if(front == -1)
      front = 0;
```

```
    rear = (rear + 1) % SIZE;
    queue[rear] = value;
    printf("\nInserted -> %d", value);
  }
}

void deQueue(){
  if(front == -1)
    printf("\nQueue is Empty!!! Deletion is not possible!!!");
  else{
    printf("\nDeleted : %d", queue[front]);
    if(front == rear)
      front = rear = -1;
    else
    {
      front = (front + 1) % SIZE;
    }

  }
}

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]);
```

```
    printf("%d", queue[i]);
  }
}
```

**Output:**

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 2

Queue is Empty!!! Deletion is not possible!!!

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 3

Queue is Empty!!!

***** MENU *****

1. Insertion

2. Deletion

3. Display

4. Exit

Enter your choice: 1

Enter the value to be insert: 4

Inserted -> 4

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 8

Inserted -> 8

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 9

Inserted -> 9

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit

```
Enter your choice: 3

Queue elements are:
4      8      9

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

Deleted : 4

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

Queue elements are:
8      9

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

Enter your choice: 1
Enter the value to be insert: 4

Inserted -> 4

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

Queue elements are:
8 9 4

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

Deleted : 8

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit

Enter your choice: 2

Deleted : 9

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

Deleted : 4

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

Queue is Empty!!! Deletion is not possible!!!

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

15. **Aim: Write a C program to create a single linked list with 5 nodes.**
**(5 integers are taken from user input) and display the linked-list elements**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to create a singly linked list with 5 nodes.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: n, i
INPUT: Reads input from the keyboard
COMPUTATION: Creates linked lists with n nodes
DISPLAY: Prints the final linked list
STOP

**Program in C :**

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

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

node * CreateNode(int n);
```

```
void DisplayNodes(node * head);

int main(void) {
   int n;
   printf("Enter how many nodes: ");
   scanf("%d", &n);
   node * HEAD = NULL;
   HEAD = CreateNode(n);
   DisplayNodes(HEAD);
   return 0;
}

node * CreateNode(int n) {
   node * head = NULL;
   node * temp = NULL;
   node * nodes = NULL;
   for (int i = 1; i <= n; i++) {
      temp = (node *)malloc(sizeof(node));
      printf("Enter the data for the node %d: ", i);
      scanf("%d", &temp->data);
      temp->next = NULL;
      if (head == NULL) {
         head = temp;
      }
      else {
         nodes = head;
         while (nodes->next != NULL) {
            nodes = nodes->next;
         }
```

```
        nodes->next = temp;
      }
    }
    return head;
}

void DisplayNodes(node * head) {
    node * nodes = head;
    while (nodes != NULL) {
        printf("-> %d ", nodes->data);
        nodes = nodes->next;
    }
}
```

**Output:**

Enter how many nodes: 5

Enter the data for the node 1: 8

Enter the data for the node 2: 1

Enter the data for the node 3: 4

Enter the data for the node 4: 2

Enter the data for the node 5: 6

-> 8 -> 1 -> 4 -> 2 -> 6

16. **Aim: Write a C program to search an element in a singly-linked list.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to search an element in a singly-linked list.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: n, choice, search, i
INPUT: Reads input from the keyboard
COMPUTATION: Searches for an element in a linked if it exists
DISPLAY: Prints the element if it exists in list
STOP

**Program in C :**

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

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

```c
node * CreateNode(int n);
void SearchNode(node * head, int search);

int main(void) {
   int n, choice, search;
   printf("Enter how many nodes: ");
   scanf("%d", &n);
   node * HEAD = NULL;
   HEAD = CreateNode(n);

   while(1)
   {
      printf("\n\n*** MENU ***\n");
      printf("1. Search an element \n2. Exit");
      printf("\nEnter your choice: ");
      scanf("%d", &choice);
      getchar();
      switch(choice)
      {
         case 1: printf("Enter an element to search in the Linked List: ");
            scanf("%d", &search);
            SearchNode(HEAD, search);
            break;
         case 2: exit(0);
         default: printf("\nWrong selection!!! Try again!!!");
      }
   }
   return 0;
}
```

```
node * CreateNode(int n) {
  node * head = NULL;
  node * temp = NULL;
  node * nodes = NULL;
  for (int i = 1; i <= n; i++) {
    temp = (node *)malloc(sizeof(node));
    printf("Enter the data for the node %d: ", i);
    scanf("%d", &temp->data);
    temp->next = NULL;
    if (head == NULL) {
      head = temp;
    }
    else {
      nodes = head;
      while (nodes->next != NULL) {
        nodes = nodes->next;
      }
      nodes->next = temp;
    }
  }
  return head;
}

void SearchNode(node * head, int search) {
  node * nodes = head;
  int counter = 0, check = 0;
  while (nodes != NULL) {
    counter++;
```

```
    if (nodes->data == search){
        check++;
        break;
    }
    nodes = nodes->next;
  }
  if (check > 0){
    printf("The element is found in the node %d", counter);
  }
  else{
    printf("The entered is not found in the Linked List");
  }
}
```

**Output:**

Enter how many nodes: 5

Enter the data for the node 1: 1

Enter the data for the node 2: 2

Enter the data for the node 3: 3

Enter the data for the node 4: 4

Enter the data for the node 5: 5

*** MENU ***

1. Search an element

2. Exit

Enter your choice: 8

Wrong selection!!! Try again!!!

*** MENU ***

1. Search an element

2. Exit

Enter your choice: 1

Enter an element to search in the Linked List: 8

The entered is not found in the Linked List

*** MENU ***

1. Search an element

2. Exit

Enter your choice: 1

Enter an element to search in the Linked List: 4

The element is found in the node 4

*** MENU ***

1. Search an element

2. Exit

Enter your choice: 1

Enter an element to search in the Linked List: 5

The element is found in the node 5

*** MENU ***

1. Search an element

2. Exit

Enter your choice: 2

17. **Aim: Write a C program to perform the following tasks:**
    1. **Insert a node at the beginning of a singly-linked list.**
    2. **Insert a node at end of a singly-linked list.**
    3. **Insert a node in the middle of a singly-linked list.**
    4. **Delete a node from the beginning of the singly-linked list.**
    5. **Delete a node from the end of a singly-linked list.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to perform all the tasks

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: choice, start
INPUT: Reads input from the keyboard
COMPUTATION: Performs the given tasks
DISPLAY: Linked List and the result of the tasks
STOP

**Program in C :**

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

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

```
node *start = NULL;

void create();
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
void delete_end();

void main() {
   int choice;
   while(1){
      printf("\n1. Create \n2. Display \n3. Insert at the beginning \n4. Insert at the end \n5. Insert at specified position \n6. Delete from beginning \n7. Delete from the end \n8. Exit\n");
      printf("\nEnter your choice:");
      scanf("%d", &choice);
      switch(choice) {
         case 1: create();
               break;
         case 2: display();
               break;
         case 3: insert_begin();
               break;
         case 4: insert_end();
               break;
         case 5: insert_pos();
               break;
         case 6: delete_begin();
               break;
         case 7: delete_end();
               break;
         case 8: exit(0);
               break;
         default: printf("\n Wrong Choice");
                  break;
```

```
    }
  }
}

void create() {
  node *temp, *ptr;
  temp = (node *)malloc(sizeof(node));
  if (temp == NULL) {
    printf("\nOut of Memory Space:");
    exit(0);
  }
  printf("\nEnter the data value for the node:");
  scanf("%d", &temp->data);
  temp->next = NULL;
  if (start == NULL) {
    start = temp;
  }
  else {
    ptr = start;
    while(ptr->next != NULL) {
      ptr = ptr->next;
    }
    ptr->next = temp;
  }
}

void display() {
  node *ptr;
  if(start == NULL) {
    printf("\nList is empty");
    return;
  }
  else {
    ptr = start;
    printf("\nThe List elements are: \n");
    while (ptr != NULL) {
      printf("%d\t", ptr->data );
```

```
        ptr = ptr->next;
      }
    }
}

void insert_begin() {
    node *temp;
    temp = (node *)malloc(sizeof(node));
    if (temp == NULL) {
        printf("\nOut of Memory Space:");
        return;
    }
    printf("\nEnter the data value for the node:" );
    scanf("%d", &temp->data);
    temp->next = NULL;
    if (start == NULL) {
        start = temp;
    }
    else {
        temp->next = start;
        start = temp;
    }
}
void insert_end() {
    node *temp, *ptr;
    temp = (node *)malloc(sizeof(node));
    if (temp == NULL) {
        printf("\nOut of Memory Space:");
        return;
    }
    printf("\nEnter the data value for the node:\t" );
    scanf("%d", &temp->data);
    temp->next = NULL;
    if (start == NULL) {
        start = temp;
    }
    else {
```

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

void insert_pos() {
  node *ptr, *temp;
  int i, pos;
  temp = (node *)malloc(sizeof(node));
  if(temp == NULL) {
    printf("\nOut of Memory Space:");
    return;
  }
  printf("\nEnter the position for the new node to be inserted:\t");
  scanf("%d", &pos);
  printf("\nEnter the data value of the node:\t");
  scanf("%d", &temp->data) ;
  temp->next = NULL;
  if(pos == 0) {
    temp->next = start;
    start = temp;
  }
  else {
    for(i = 0, ptr = start; i < pos - 1; i++) {
         ptr = ptr->next;
       if (ptr == NULL) {
         printf("\nPosition not found:[Handle with care]");
         return;
       }
    }
    temp->next = ptr->next ;
    ptr->next = temp;
  }
}
```

```
void delete_begin() {
  node *ptr;
  if (ptr == NULL) {
    printf("\nList is Empty:");
    return;
  }
  else {
    ptr = start;
    start = start->next ;
    printf("\nThe deleted element is: %d\t", ptr->data);
    free(ptr);
  }
}

void delete_end() {
  node *temp, *ptr;
  if (start == NULL) {
    printf("\nList is Empty:");
    exit(0);
  }
  else if (start->next == NULL) {
    ptr = start;
    start = NULL;
    printf("\nThe deleted element is: %d\t", ptr->data);
    free(ptr);
  }
  else {
    ptr = start;
    while (ptr->next != NULL) {
      temp = ptr;
      ptr = ptr->next;
    }
    temp->next = NULL;
    printf("\nThe deleted element is: %d\t", ptr->data);
    free(ptr);
  }
```

```
}
```

**Output:**

1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

Enter your choice:1

Enter the data value for the node:11

1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

Enter your choice:1

Enter the data value for the node:22

1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

Enter your choice:1

Enter the data value for the node:33

1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

Enter your choice:2

The List elements are:
11 22 33
1.Create
2.Display
3.Insert at the beginning

4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

Enter your choice:5

Enter the position for the new node to be inserted: 2

Enter the data value of the node: 65

1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

Enter your choice:2

The List elements are:
11 22 65 33
1.Create
2.Display
3.Insert at the beginning
4.Insert at the end

5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

Enter your choice:6

The deleted element is: 11
1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

Enter your choice:7

The deleted element is: 33
1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Exit

```
Enter your choice:2

The List elements are:
22     65
 1.Create
 2.Display
 3.Insert at the beginning
 4.Insert at the end
 5.Insert at specified position
 6.Delete from beginning
 7.Delete from the end
 8.Exit

Enter your choice:4

Enter the data value for the node:      99

 1.Create
 2.Display
 3.Insert at the beginning
 4.Insert at the end
 5.Insert at specified position
 6.Delete from beginning
 7.Delete from the end
 8.Exit

Enter your choice:2

The List elements are:
```

```
22    65    99
 1.Create
 2.Display
 3.Insert at the beginning
 4.Insert at the end
 5.Insert at specified position
 6.Delete from beginning
 7.Delete from the end
 8.Exit

Enter your choice:7

The deleted element is: 99
 1.Create
 2.Display
 3.Insert at the beginning
 4.Insert at the end
 5.Insert at specified position
 6.Delete from beginning
 7.Delete from the end
 8.Exit

Enter your choice:2

The List elements are:
22    65
 1.Create
 2.Display
 3.Insert at the beginning
```

4.Insert at the end

5.Insert at specified position

6.Delete from beginning

7.Delete from the end

8.Exit

Enter your choice:8

18. **Aim: Write a C program to create a doubly linked list with 5 nodes.**

Objective:

At the end of this activity we will be able to understand different data types, operators and expressions to create a doubly linked list with 5 nodes.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START

DEFINING VARIABLES: HEAD, n

INPUT: Reads input from the keyboard

COMPUTATION: Creates a Doubly Linked List

DISPLAY: Doubly Linked List

STOP

**Program in C :**

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

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

node * CreateNode(int n);
void DisplayNodes(node * head);
```

```
int main(void) {
   int n;
   printf("Enter how many nodes: ");
   scanf("%d", &n);
   node * HEAD = NULL;
   HEAD = CreateNode(n);
   DisplayNodes(HEAD);
   return 0;
}

node * CreateNode(int n) {
   node * head = NULL;
   node * temp = NULL;
   node * nodes = NULL;
   for (int i = 1; i <= n; i++) {
      temp = (node *)malloc(sizeof(node));
      printf("Enter the data for the node %d: ", i);
      scanf("%d", &temp->data);
      temp->prev = NULL;
      temp->next = NULL;
      if (head == NULL) {
         head = temp;
      }
      else {
         nodes = head;
         while (nodes->next != NULL) {
            nodes = nodes->next;
         }
         nodes->next = temp;
         temp = nodes;
         nodes = nodes->next;
         nodes->prev = temp;
      }
   }
   return head;
}
```

```
void DisplayNodes(node * head) {
  node * nodes = head;
  while (nodes != NULL) {
    printf("-> %d ", nodes->data);
    nodes = nodes->next;
  }
}
```

**Output:**

Enter how many nodes: 5

Enter the data for the node 1: 3

Enter the data for the node 2: 4

Enter the data for the node 3: 5

Enter the data for the node 4: 6

Enter the data for the node 5: 9

-> 3 -> 4 -> 5 -> 6 -> 9

19. **Aim: Write a C program to create a circular linked list with 5 nodes.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to create a circular linked list with 5 nodes.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START

DEFINING VARIABLES: HEAD, n

INPUT: Reads input from the keyboard

COMPUTATION: Creates a Circular Linked List

DISPLAY: Circular Linked List

STOP

**Program in C :**

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

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

node * CreateNode(int n);
void DisplayNodes(node * head);
```

```
int main(void) {
  int n;
  printf("Enter how many nodes: ");
  scanf("%d", &n);
  node * HEAD = NULL;
  HEAD = CreateNode(n);
  DisplayNodes(HEAD);
  return 0;
}

node * CreateNode(int n) {
  node * head = NULL;
  node * temp = NULL;
  node * nodes = NULL;
  for (int i = 1; i <= n; i++) {
    temp = (node *)malloc(sizeof(node));
    printf("Enter the data for the node %d: ", i);
    scanf("%d", &temp->data);
    temp->prev = NULL;
    temp->next = NULL;
    if (head == NULL) {
      head = temp;
    }
    else {
      nodes = head;
      while (nodes->next != NULL) {
        nodes = nodes->next;
      }
      head->prev = temp;
      nodes->next = temp;
      temp = nodes;
      nodes = nodes->next;
      nodes->prev = temp;
    }
  }
  return head;
```

```
}

void DisplayNodes(node * head) {
   node * nodes = head;
   while (nodes != NULL) {
      printf("-> %d ", nodes->data);
      nodes = nodes->next;
   }
}
```

**Output:**

Enter how many nodes: 7

Enter the data for the node 1: 1

Enter the data for the node 2: 5

Enter the data for the node 3: 8

Enter the data for the node 4: 8

Enter the data for the node 5: 4

Enter the data for the node 6: 4

Enter the data for the node 7: 4

-> 1 -> 5 -> 8 -> 8 -> 4 -> 4 -> 4

20.**Aim: Write a C program to implement the stack using a linked list.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to implement the stack using a linked list.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START

DEFINING VARIABLES: size, value, choice

INPUT: Reads input from the keyboard

COMPUTATION: Uses linked lists to perform stack operations

DISPLAY: Displays the output after the operations

STOP

**Program in C :**

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

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

```
node * PUSH(node * head, int SIZE, int value);
void DISPLAY(node * head);
node * POP(node * head);
void PEEK(node * head);

int main(void) {
 int SIZE, value, choice;
 printf("Enter the size of STACK: ");
 scanf("%d", &SIZE);
 node * STACK = NULL;
 while(1) {
  printf("\nChoose the following: \n");
  printf("1. Push \n2. Pop \n3. Display \n4. Peek \n5. Exit \n");
  printf("Enter your choice: ");
  scanf("%d", &choice);
  switch(choice) {
   case 1: printf("Enter an element to push: ");
           scanf("%d", &value);
           STACK = PUSH(STACK, SIZE, value);
           break;
   case 2: STACK = POP(STACK);
           break;
   case 3: DISPLAY(STACK);
           break;
   case 4: PEEK(STACK);
           break;
   case 5: exit(0);
           break;
   default: printf("\nIncorrect option choosen!!!\n");
```

```
        break;
   }
  }
}

node * PUSH(node * head, int SIZE, int value) {
 node * temp = NULL;
 node * tail = NULL;
 temp = (node *)malloc(sizeof(node));
 temp->data = value;
 temp->next = NULL;
 if (head == NULL) {
  head = temp;
  printf("\nINSERTION SUCCESS!!!\n");
 }
 else {
  tail = head;
  int count = 1;
  while(tail->next != NULL) {
   tail = tail->next;
   count++;
   }
  if (count == SIZE) {
     printf("\nSTACK OVERFLOW!!!\n");
  }
  else{
   tail->next = temp;
   printf("\nINSERTION SUCCESS!!!\n");
  }
```

```
 }
 return head;
}

void DISPLAY(node * head) {
 node * tail = head;
 printf("\n");
 if (head == NULL) printf("STACK EMPTY!!!");
 while(tail != NULL) {
  printf("-> %d ", tail->data);
  tail = tail->next;
 }
 printf("\n");
}

node * POP(node * head) {
 node * tail = head;
 node * pop = head;
 if(tail == NULL) {
  printf("\nSTACK IS EMPTY!!!\n");
 }
 else{
  int count = 0;
  while(tail->next != NULL) {
   count++;
   pop = tail;
   tail = tail->next;
  }
  printf("\nPOPPED ELEMENT = %d\n", tail->data);
```

```
  pop->next = NULL;
  if (count == 0) head = NULL;
 }
 return head;
}

void PEEK(node * head) {
 node * tail = head;
 if(tail == NULL) {
  printf("\nSTACK IS EMPTY!!!\n");
 }
 else{
  while(tail->next != NULL) {
   tail = tail->next;
  }
  printf("\nPEEK ELEMENT = %d\n", tail->data);
 }
}
```

**Output:**

Enter the size of STACK: 5

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 1

Enter an element to push: 1

INSERTION SUCCESS!!!

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 1

Enter an element to push: 2

INSERTION SUCCESS!!!

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 1

Enter an element to push: 3

INSERTION SUCCESS!!!

Choose the following:

1. Push
2. Pop

3. Display
4. Peek
5. Exit
Enter your choice: 1
Enter an element to push: 4

INSERTION SUCCESS!!!

Choose the following:
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1
Enter an element to push: 5

INSERTION SUCCESS!!!

Choose the following:
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1
Enter an element to push: 6

STACK OVERFLOW!!!

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 3

-> 1 -> 2 -> 3 -> 4 -> 5

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 4

PEEK ELEMENT = 5

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 2

POPPED ELEMENT = 5

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 2

POPPED ELEMENT = 4

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 2

POPPED ELEMENT = 3

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 2

POPPED ELEMENT = 2

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 2

POPPED ELEMENT = 1

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 2

STACK IS EMPTY!!!

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 3

STACK EMPTY!!!

Choose the following:

1. Push
2. Pop
3. Display
4. Peek
5. Exit

Enter your choice: 5

21. **Aim: Write a C program to implement the queue using a linked list.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to implement the queue using a linked list.

**Problem Statement:** In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START
DEFINING VARIABLES: *front, *rear, choice, value
INPUT: Reads input from the keyboard
COMPUTATION: Uses linked lists to perform operations
DISPLAY: Displays the output after the operations
STOP

**Program in C :**

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

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

element *front = NULL, *rear = NULL;

void enQueue(int value);
void Display();
```

```
void deQueue();
void Peek();

void main() {
 int choice, value;
 element * Queue = NULL;
 while(1) {
  printf("\n1. enQueue \n2. deQueue \n3. Display \n4. Peek \n5. Exit \n");
  printf("Select an option: ");
  scanf("%d", &choice);
  switch(choice) {
   case 1: printf("Enter a value: ");
       scanf("%d", &value);
       enQueue(value);
       break;
   case 2: deQueue();
       break;
   case 3: Display(Queue);
       break;
   case 4: Peek();
       break;
   case 5: exit(0);
   default: printf("\nInvalid choice of option!!!\n");
       break;
  }
 }
}

void enQueue(int value) {
  element * NewElement = (element *)malloc(sizeof(element));
  NewElement->data =  value;
  NewElement->next = NULL;
  if (front == NULL && rear == NULL) {
    front = rear = NewElement;
  }
  else {
    rear->next = NewElement;
```

```
    rear = NewElement;
  }
  printf("\nInsertion Successfull\n");
}

void Display() {
  if (front == rear) {
    printf("\nQueue is empty\n");
  }
  else {
    element * temp = front;
    while(temp) {
      printf("-> %d  ", temp->data);
      temp = temp -> next;
    }
  }
}

void deQueue() {
  if (front == rear) {
    printf("\nQueue is empty\n");
  }
  else {
    front = front->next;
    if (front == NULL) {
      rear = NULL;
    }
    printf("\ndeQueue successfull\n");
  }
}

void Peek() {
  if (front == rear) {
    printf("\nQueue is empty\n");
  }
  else {
    while(rear->next != NULL) {
```

```
    rear = rear->next;
    }
    printf("\nPeek element = %d\n", rear->data);
  }
}
```

Output:

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 1

Enter a value: 4

Insertion Successfull

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 1

Enter a value: 8

Insertion Successfull

1. enQueue
2. deQueue
3. Display

4. Peek
5. Exit
Select an option: 1
Enter a value: 9

Insertion Successfull

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit
Select an option: 1
Enter a value: 2

Insertion Successfull

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit
Select an option: 3
-> 4 -> 8 -> 9 -> 2
1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 4

Peek element = 2

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 2

deQueue successfull

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 2

deQueue successfull

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 2

deQueue successfull

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 2

Queue is empty

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 3

Queue is empty

1. enQueue
2. deQueue
3. Display
4. Peek
5. Exit

Select an option: 5

22. **Aim: Write a C program to implement a single source shortest path algorithm. Either Bellman-Ford or Dijkstra's algorithm.**

**Objective:**

At the end of this activity we will be able to understand different data types, operators and expressions to implement a single source shortest path algorithm.

**Problem Statement:**

In this program, we aim to understand the usage of different data types and take the following input from the users.

**Algorithm:**

START

DEFINING VARIABLES: V, edge, G, i, j, k

INPUT: Reads input from the keyboard

COMPUTATION: Finds the shortest path for all vertices

DISPLAY: A matrix of single source shortest path

STOP

**Program in C :**

```c
#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;
}
void main() {
   int V, edge[20][2], G[20][20], i, j, k=0;
   printf("BELLMAN FORD\n");
   printf("Enter number 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");
   }
```

```
}
```

**Output:**

BELLMAN FORD

Enter number of vertices: 5

Enter graph in matrix form:

0 6 7 1000 1000

1000 0 8 5 -4

1000 1000 0 -3 9

1000 -2 1000 0 1000

2 1000 1000 7 0

Enter source: 1

Vertex 1 -> cost = 0 parent = 0

Vertex 2 -> cost = 2 parent = 4

Vertex 3 -> cost = 7 parent = 1

Vertex 4 -> cost = 4 parent = 3

Vertex 5 -> cost = -2 parent = 2

No negative weight cycle