**LAB No: 1(a)**

**Largest Element in Array**

**Objective**

To find the Largest element in an Array.

**Theory:**

An Array is collection of cells of the same type. The cells are numbered with consecutive integers. Array cells are contiguous in computer memory. The memory can be thought of as an array.

**Algorithm**

1. [Initialize] Set I: = 1 & INDEX: and LARGEST: =DATA[1]

2. Repeat step 3 and 4 for K < N:

3. If LARGEST<DATA[I], then

Set INDEX: = I and LARGEST: =DATA[I].

[End of If structure]

4.

[End of Step 2 for loop]

5. Write: INDEX, LARGEST.

6. Exit.

**Codes Language C++:**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int arr[] = {4, 9, 1, 3, 8};**

**int largest, i, index;**

**largest = arr[0];**

**for(i=1; i<5; i++) {**

**if(arr[i]>largest) {**

**largest = arr[i];**

**index = i;**

**} }**

**cout<<"The largest element in the array is "<<largest<<" and it is at index "<<index;**

**return 0;**

**}**

**Output : A black screen with white text

Description automatically generated**

**LAB No: 1(b)**

**Largest & Smallest Element in Array**

**Objective**

To find the Largest and Smallest element in an Array.

**Theory:**

An Array is collection of cells of the same type. The cells are numbered with consecutive integers. Array cells are contiguous in computer memory. The memory can be thought of as an array.

**Algorithm** **:**

1. [Initialize] Set I: = 0 & INDEX: and MAX or MIN: =DATA[1]

2. Repeat step 3 and 4 for I < N: N = READ

3. If LARGEST<DATA[I], then

Set INDEX: = I and LARGEST: =DATA[I].

[End of If structure]

4. If SMALLEST<DATA[I], then

Set INDEX: = I and SMALLEST: =DATA[I].

[End of If structure]

[End of Step 2 for loop]

5. Write: INDEX, LARGEST.

6. Write: INDEX, SMALLEST.

7. Exit.

**Codes Language C++:**

#include<iostream>

using namespace std;

int main ()

{

int arr[10], n, i, max, min;

cout << "Enter the size of the array : ";

cin >> n;

cout << "Enter the elements of the array : ";

for (i = 0; i < n; i++)

cin >> arr[i];

max = arr[0];

for (i = 0; i < n; i++)

{

if (max < arr[i])

max = arr[i];

}

min = arr[0];

for (i = 0; i < n; i++)

{

if (min > arr[i])

min = arr[i];

}

cout << "Largest element : " << max;

cout << "\nSmallest element : " << min;

return 0;

}

**Output :**

**A screen shot of a computer

Description automatically generated**

**LAB No: 1(c)**

**Three Largest & Smallest Element in Array**

**Objective**

To find the Three Largest and Smallest element in an Array.

**Theory:**

An Array is collection of cells of the same type. The cells are numbered with consecutive integers. Array cells are contiguous in computer memory. The memory can be thought of as an array.

**Algorithm**

1. [Initialize] Set I: = 0 & INDEX = 0: and MAX or MIN: =DATA[1]

2. Repeat step 3 and 4 for I < N: N = READ

3. If LARGEST<DATA[I], then

Set INDEX: = I and LARGEST: =DATA[I].

[End of If structure]

4. If SMALLEST<DATA[I], then

Set INDEX: = I and SMALLEST: =DATA[I].

[End of If structure]

[End of Step 2 for loop]

5. Write: INDEX, LARGEST.

6. Write: INDEX, SMALLEST.

7. Exit.

**Codes Language C++:**

**#include<iostream>**

**#include <climits>**

**using namespace std;**

**void three\_largest(int arr[], int arr\_size)**

**{**

**int i, first, second, third;**

**if (arr\_size < 3)**

**{ cout << "Invalid Input";**

**}**

**third = first = second = INT\_MIN;**

**for (i = 0; i < arr\_size ; i ++)**

**{**

**if (arr[i] > first)**

**{**

**third = second;**

**second = first;**

**first = arr[i];**

**}**

**else if (arr[i] > second)**

**{**

**third = second;**

**second = arr[i]; }**

**else if (arr[i] > third)**

**third = arr[i]; }**

**cout << "\nThree largest elements are: \n" <<first <<"\n"<<second <<"\n"<<third;}**

**void three\_Smallest(int arr[], int arr\_size)**

**{**

**int i, first, second, third;**

**if (arr\_size < 3)**

**{ cout << "Invalid Input"; }**

**third = first = second = INT\_MAX;**

**for (i = 0; i < arr\_size ; i ++)**

**{**

**if (arr[i] < first)**

**{ third = second;**

**second = first;**

**first = arr[i];**

**} else if (arr[i] < second)**

**{**

**third = second;**

**second = arr[i];**

**} else if (arr[i] < third)**

**third = arr[i]; }**

**cout << "\nThree Smallest elements are: \n" <<first <<"\n"<<second <<"\n"<<third;}**

**int main()**

**{**

**int nums[] = {7, 12, 9, 15, 19, 32, 56, 70};**

**int n = sizeof(nums)/sizeof(nums[0]);**

**cout << "Original array: ";**

**for (int i=0; i < n; i++)**

**cout << nums[i] <<" ";**

**three\_largest(nums, n);**

**three\_Smallest(nums,n);**

**return 0;**

**}**

**Output : A screenshot of a computer

Description automatically generated**

**LAB No: 2(A)**

**Linear and Binary Search of an Array**

**Objective**

To find the element in an Array using Linear Search

**Algorithm (Linear Search)**

A Linear DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in array DATA or sets LOC=0.

1. [Initialize] Set k:=1 and LOC:=0.

2. Repeat Step 3 and 4 while LOC=0 AND k<=N.

3. If ITEM = DATA[k], then:

Set LOC:=k.

4. Else

Set k:=k+1. [Increments counter]

[End of If Structure.]

[End of Step 2 loop.]

5. [Successful?]

If LOC=0 then:

Write: ITEM is not in the array DATA.

Else

Write: LOC is the location of ITEM.

[End of If structure]

6. Exit.

**Programs language (C++ code)**

#include<iostream>

using namespace std;

int main(){

int array[20],n, index = 0,srch;

cout<<"Enter the size of an array: "; cin>>n;

for(int i=0;i<n; i++){

cin>>array[i];

}

cout<<"Enter value you found in array"; cin>>srch;

for(int i=0;i<n;i++){

if (array[i]==srch){

index = i;

} }

if(index != 0){

cout<< "\n Element found at position : " << index+1 ; }

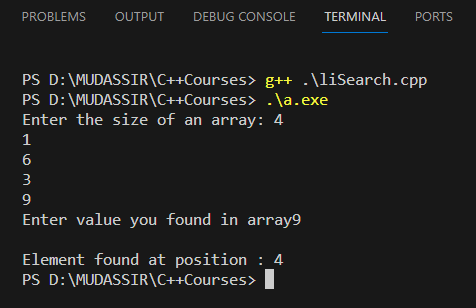
else{

cout<<"\n Value not present\n"; }

return 0;

}

**Output**

****

**LAB No: 2(B)**

**Objective**

To find the element in an Array using Binary Search

**Algorithm (Binary Search):**

BinarySearch (A, SKEY)

**1.** [Initialize segment variables.]

Set START:=0, END:=N-1 and MID=INT((START+END)/2).

**2.** Repeat Steps 3 and 4 while START ≤ END AND A[MID]≠SKEY.

**3.** If SKEY< A[MID], then:

Set END:=MID-1.

Else

Set START:=MID+1.

[End of If Structure.]

**4.** Set MID:=INT((START +END)/2).

[End of Step 2 loop.]

**5.** If A[MID]= SKEY, then:

Set LOC:= MID

Else:

Set LOC := NULL

[End of If structure.]

**6.** Exit

**Programs language (C++ code)**

#include <iostream>

using namespace std;

int main()

{

int count, i, arr[30], num, first, last, middle;

cout<<"how many elements would you like to enter?:";

cin>>count;

for (i=0; i<count; i++)

{

cout<<"Enter number "<<(i+1)<<": ";

cin>>arr[i];

}

cout<<"Enter the number that you want to search:";

cin>>num;

first = 0;

last = count-1;

middle = (first+last)/2;

while (first <= last)

{

if(arr[middle] < num)

{

first = middle + 1;

}

else if(arr[middle] == num)

{

cout<<num<<" found in the array at the location "<<middle+1<<"\n";

break;

}

else {

last = middle - 1;

}

middle = (first + last)/2;

}

if(first > last)

{

cout<<num<<" not found in the array";

}

return 0;

}

**Output: A screenshot of a computer program

Description automatically generated**

**LAB No: 3**

**Objective**

Perform Bubble Sort

**Algorithm (Bubble Sort) BUBBLE (DATA, N)**

Here DATA is an array with N elements. This algorithm sorts the elements in DATA.

1. Repeat step 2 and 3 for K=1 to N-1
2. Set PTR :=1. [Initialize pass pointer PTR.]
3. Repeat While PTR ≤ N-K: [Executes pass.]

If DATA[PTR] > DATA[PTR+1], then

Interchange DATA[PTR] and DATA[PTR+1]

[End of If structure]

[End of Step 3 inner loop.]

Set PTR:= PTR+1.

[End of Step 1 outer loop.]

1. Exit.

**Bubble Sort Program**

|  |
| --- |
| #include<iostream>  using namespace std;  int main ()  {     int i, j,temp,pass=0;     int a[10] = {10,2,0,14,43,25,18,1,5,45};     cout <<"Input list ...\n";     for(i = 0; i<10; i++) {        cout <<a[i]<<"\t";     }  cout<<endl;  for(i = 0; i<10; i++) {     for(j = i+1; j<10; j++)     {        if(a[j] < a[i]) {           temp = a[i];           a[i] = a[j];           a[j] = temp;        }     }  pass++;  }  cout <<"Sorted Element List ...\n";  for(i = 0; i<10; i++) {     cout <<a[i]<<"\t";  }  cout<<"\nNumber of passes taken to sort the list:"<<pass<<endl;  return 0;  } |

**Output:**

A screenshot of a computer program

Description automatically generated

### Optimize solution of Bubble Sort:

Below is the optimized program.

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

cout<<"Enter number of element you want to store: ";

cin>>n;

int arr[n],i,j;

cout<<"Enter array values:\n";

//taking the array value

//from user

for(i=0;i<n;i++)

{

cin>>arr[i];

}

//here this flag will help

//to optimise the solution

//first initialise flag=1

int flag=1;

//Now we will sort the array

//if my flag value is 1 then only

//the loop will execute

for(i=0;i<n-1 && flag==1;i++) {

flag=0;

for(j=0;j<n-i-1;j++)

{ if(arr[j]>arr[j+1]) {

//temp will temporarly store

//the value of arr[j]

//then we will swap the values

int temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

//Here if there is a swap then

// we will make it 1

flag=1; } } }

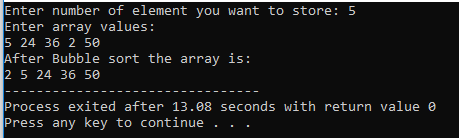
cout<<"After Bubble sort the array is:\n";

for(i=0;i<n;i++)

cout<<arr[i]<<" ";

return 0; }

The output of the above program is following.

[](https://appdividend.com/wp-content/uploads/2019/04/C-Bubble-Sort-Program-Tutorial-With-Example-1.png)

**Lab 4**

**Objective:**

To understand the basic concepts of **Insertion** and **Deletion** in array.

**Algorithm to Insert Element in un-ordered Array is as follows:**

DATA is unsorted linear array with N elements. LOC is the location where ITEM is to be inserted where LOC ≤ N. This Algorithm inserts an element ITEM into the LOCth position in array DATA.

**INSERT (DATA, N, I, ITEM, LOC)**

1. Set I:=N [Initialize counter]

2. Repeat the step 3 and 4 while I ≥ LOC

3. Set DATA[I+1]:=DATA[I] [Move Ith elements downward or forward]

4. Set I=I-1 [Decrease counter by 1]

[End of while loop]

5. Set DATA[LOC]=ITEM [Insert Element]

6. Set N=N+1 [Reset N]

7. Exit

**Algorithm to Delete Element from Array is:**

**DELETE (DATA, N, I, ITEM, LOC)**

1.Set ITEM=DATA[LOC] [Assign en Element to be deleted to ITEM]

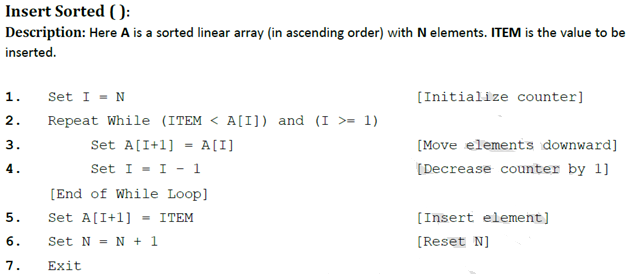
2. Repeat for I=LOC to N-1

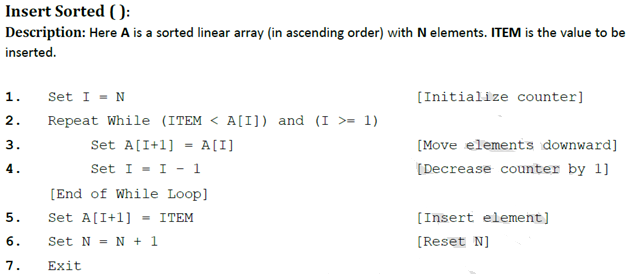
3. Set DATA[I]:=DATA[I+1] [Move the I+1st Element upward or backward]

[End of For loop]

4. Set N:=N-1 [Reset N]

5. Exit.





**/\* C++ program to insert an element in an array \*/**

#include<iostream>

using namespace std;

int main()

{

int n;

cin >> n;

int arr[n];

int i;

for(i = 0; i < n; i++)

{

cin >> arr[i];

}

int pos;

cin >> pos;

int ele;

cin >> ele;

if(pos > n)

cout << “Invalid Input”;

else

{

for (i = n – 1; i >= pos – 1; i–)

arr[i+1] = arr[i];

arr[pos-1] = ele;

cout << “Array after insertion is:\n”;

for (i = 0; i <= n; i++)

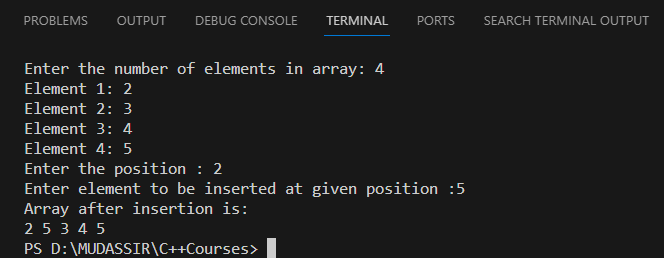
cout << arr[i] << ” “;

}

return 0;

}

**Output:**

****

**// C++ program to delete an element in an array**

#include

using namespace std;

int main()

{

int array[100], position, c, n;

cout << “Enter the number of elements of the array : “;

cin >> n;

cout << “\nInput the array elements : “;

for (c = 0; c < n; c++)

cin >> array[c];

cout << “\nEnter the position : “;

cin >> position;

if (position >= n+1)

cout << “\nDeletion not possible.\n”;

else

{

for (c = position – 1; c < n – 1; c++)

array[c] = array[c+1];

cout << “\nArray after deletion : “;

for (c = 0; c < n – 1; c++)

cout << array[c] << ” “;

}

cout << endl;

return 0;

}

Output

5 (size of the array)  
1 (array elements)  
2  
3  
4  
5  
4 (Position to be deleted)  
Array after deletion is : 1 2 3 5

**LAB No: 5A**

**Objective:**

Sort an array using Selection Sort.

**Algorithm:** (Selection Sort) SELECTION (A, N)

This algorithm sorts the array A with N elements.

1. Repeat steps 2 and 3 for K=0 to N-2: [Find LOC of the smallest element from A[0] to A[N-2]]

2. Call MIN(A, K, N, LOC). [call function MIN to find minimum element in array A]

3. [Interchange A[K] and A[LOC].]

Set TEMP: = A[K], A[K] = A[LOC] and A[LOC] := TEMP.

[End of step 1 loop.]

4. Exit.

**ANOTHER METHOD**

#include <iostream>

using namespace std;

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void display(int arr[], int n) {

for (int i = 0; i < n; i++) {

cout << arr[i] << " ";

}

cout << endl;

}

//selection sort function

void selectionSort(int arr[], int n) {

for (int j = 0; j < n - 1; j++) {

int min = j;

for (int i = j + 1; i < n; i++) {

if (arr[i] < arr[min])

min = i;

}

swap(&arr[min], &arr[j]);

}

}

int main() {

int arr[] = {20, 12, 10, 15, 2};

int n = sizeof(arr) / sizeof(arr[0]);

cout << "Elements before sorting:\n";

display(arr, n);

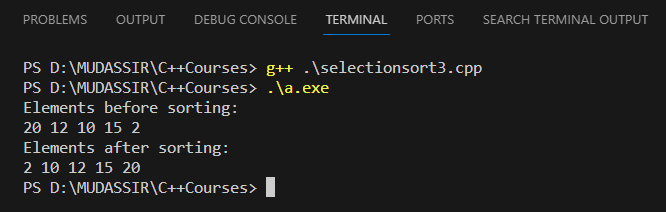
selectionSort(arr, n);

cout << "Elements after sorting:\n";

display(arr, n);

}

**Output:**

****

**LAB No: 5B**

**Objective:**

Sort an array using Insertion Sort.

**Algorithm:** (Insertion Sort) INSERTION (A, N)

This Algorithm sorts an array A with N elements.

1. Repeat steps 2 to 4 for K=1 to N-1: [an outer loop which uses K as an Index from second element A[1] to A[N-1]to last element.]

2. Set TEMP := A[K] and PTR =K-1.

[an inner loop which is essentially controlled by the variable PTR to locate hole position for the element to be inserted]

3. Repeat while PTR>= 0 and TEMP < A[PTR] [check whether the adjacent element in left side is greater or less than the current element]

a) Set A[PTR+1] := A[PTR]. [Move all the left side elements one position forward, which

are greater than key TEMP.]

b) Set PTR := PTR -1.

[End of step 3 loop.]

4. Set A[PTR+1] :=TEMP. [Move (Insert) current element in its proper place (hole position).]

[End of Step 1 loop.]

5 Return.

Insertion Sort

#include<iostream>

using namespace std;

int main ()

{

int myarray[10] = { 12,4,3,1,15,45,33,21,10,2};

cout<<"\nInput list is \n";

for(int i=0;i<10;i++)

{

cout <<myarray[i]<<"\t";

}

for(int k=1; k<10; k++)

{

int temp = myarray[k];

int j= k-1;

while(j>=0 && temp <= myarray[j])

{

myarray[j+1] = myarray[j];

j = j-1;

}

myarray[j+1] = temp;

}

cout<<"\nSorted list is \n";

for(int i=0;i<10;i++)

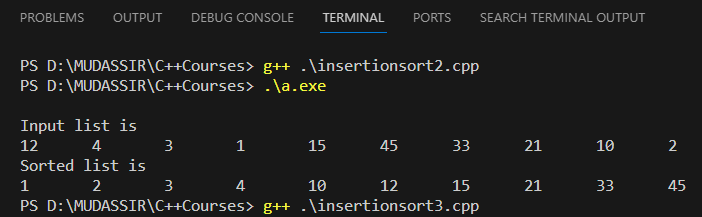
{

cout <<myarray[i]<<"\t";

}

}

Output:



**LAB 6**

**Objective**

Perform Matrix Addition, Subtraction, Multiplication and Transpose operations.

**Matrix Multiplication Algorithm**

**Matmula (A,B,C,M,P,N)**

Let A be an M X P matrix array, and let B be a P X N matrix array. This algorithm stores the product of A and B in an MXN matrix array C.

1. Repeat step 2 to 4 for I=1 to M:
2. Repeat step 3 AND 4 for J=1 to N:
3. Set C[I,J]:=0. [Initializes C[I,J]. ]
4. Repeat for K=1 to P: [loop grants the no. of columns of second matrix]

C[I,J]:=C[I,J]+A[I,K]\*B[K,J] [no. of rows of second matrix =no. of

columns of first matrix]

[End of inner loop.]

[End of step 2 middle loop.]

[End of step 1 outer loop.]

1. Exit.

**Matrix Addition Algorithm**

**MatAdd(A,B,C,M,N)**

1 Repeat for I=1; to M

2 Repeat for J=1 to N

C[I][J]= A[I][J]+ B[I][J]

[End of inner loop.]

[End of outer loop.]

3 Exit.

**Transpose of Matrix Algorithm**

**Transpose (A,M,N)**

1 Repeat for I=1; to M

2 Repeat for J=1 to N

C[J][I]= A[I][J]

[End of inner loop.]

[End of outer loop.]

3 Exit

**Program**

**Addition:**

/\* C++ Program to matrix addition \*/

#include <iostream>

using namespace std;

int main(){

//fill your code

int m, n;

cout << "Enter rows and columns of matrix: ";

cin >> m >> n;

int i, j;

int mat1[m][n], mat2[m][n], mat3[m][n];

cout << "\nEnter elements of matrix 1: " << endl;

for(i = 0; i < m; i++){

for(j = 0; j < n; j++)

cin >> mat1[i][j];

}

cout << "\nEnter elements of matrix 2: " << endl;

for(i = 0; i < n; i++){

for(j = 0; j < n; j++)

cin >> mat2[i][j];

}

for(i = 0; i < m; i++){

for(j = 0; j < n; j++){

mat3[i][j] = mat1[i][j] + mat2[i][j];

}}

cout << "\nAddition of Matrix: " << endl;

for(i = 0; i < m; i++){

for(j = 0; j < n; j++)

cout << mat3[i][j] << " ";

cout << endl;

}

return 0;

}

**Output:**

PS D:\MUDASSIR\C++Courses> g++ .\matrixaddition.cpp

PS D:\MUDASSIR\C++Courses> .\a.exe

Enter rows and columns of matrix: 2

2

Enter elements of matrix 1:

5 4

3 5

Enter elements of matrix 2:

4 2

6 4

Addition of Matrix:

9 6

9 9

**Multiplication Matrix**

**#include <iostream>**

**int main() {**

**int m, n, p, q, c, d, k, sum = 0;**

**int first[10][10], second[10][10], multiply[10][10];**

**std::cout << "Enter the number of rows and columns of the first matrix\n";**

**std::cin >> m >> n;**

**std::cout << "Enter the elements of the first matrix\n";**

**for (c = 0; c < m; c++)**

**for (d = 0; d < n; d++)**

**std::cin >> first[c][d];**

**std::cout << "Enter the number of rows and columns of the second matrix\n";**

**std::cin >> p >> q;**

**if (n != p)**

**std::cout << "Matrices with entered orders can't be multiplied with each other.\n";**

**else {**

**std::cout << "Enter the elements of the second matrix\n";**

**for (c = 0; c < p; c++)**

**for (d = 0; d < q; d++)**

**std::cin >> second[c][d];**

**for (c = 0; c < m; c++) {**

**for (d = 0; d < q; d++) {**

**for (k = 0; k < p; k++) {**

**sum = sum + first[c][k] \* second[k][d];**

**}**

**multiply[c][d] = sum;**

**sum = 0;**

**} }**

**std::cout << "Product of entered matrices:-\n";**

**for (c = 0; c < m; c++) {**

**for (d = 0; d < q; d++)**

**std::cout << multiply[c][d] << "\t";**

**std::cout << "\n";**

**}}**

**return 0; }**

**Output:**

PS D:\MUDASSIR\C++Courses> g++ .\matrixMult.cpp

PS D:\MUDASSIR\C++Courses> .\a.exe

Enter the number of rows and columns of the first matrix

2 2

Enter the elements of the first matrix

3 4 5 4

Enter the number of rows and columns of the second matrix

2 4

Enter the elements of the second matrix 5 4 3

5 6

7 8 6

Product of entered matrices:-

39 40 41 39

49 48 47 49

**Transpose Matrix**

#include <iostream>

using namespace std;

int main() {

int a[10][10], transpose[10][10], row, column, i, j;

cout << "Enter rows and columns of matrix: ";

cin >> row >> column;

cout << "\nEnter elements of matrix: " << endl;

// Storing matrix elements

for (int i = 0; i < row; ++i) {

for (int j = 0; j < column; ++j) {

cout << "Enter element a" << i + 1 << j + 1 << ": ";

cin >> a[i][j];

} }

// Printing the a matrix

cout << "\nEntered Matrix: " << endl;

for (int i = 0; i < row; ++i) {

for (int j = 0; j < column; ++j) {

cout << " " << a[i][j];

if (j == column - 1)

cout << endl << endl;

} }

// Computing transpose of the matrix

for (int i = 0; i < row; ++i)

for (int j = 0; j < column; ++j) {

transpose[j][i] = a[i][j];

}

// Printing the transpose

cout << "\nTranspose of Matrix: " << endl;

for (int i = 0; i < column; ++i)

for (int j = 0; j < row; ++j) {

cout << " " << transpose[i][j];

if (j == row - 1)

cout << endl << endl;

}

return 0; }

**Output:**

PS D:\MUDASSIR\C++Courses> g++ .\matrixTranspose.cpp

PS D:\MUDASSIR\C++Courses> .\a.exe

Enter rows and columns of matrix: 2

2

Enter elements of matrix:

Enter element a11: 3

Enter element a12: 2

Enter element a21: 3

Enter element a22: 4

Entered Matrix:

3 2

3 4

Transpose of Matrix:

3 3

2 4

**\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date Lab Instructor Sign. Course Instructor Sign.**

**LAB No: 7**

**Objective**

Write program to create a Link List and perform different functionality related to it as stated below:

**Operations on Link List**

1. Create a Link List
2. Traversing Link List
3. Searching in Link List
4. Display a Link List

**Algorithm: (Traversing a Link List)** Let LIST be a link list in memory. This Algorithm traverses LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to the node currently being processed.

1. Set PTR:=START.[Initializes pointer PTR.]
2. Repeat step 3 and 4 while PTR≠NULL.
3. Apply PROCESS to INFO[PTR].
4. Set PTR=LINK[PTR]. [PTR now points to the next node.]

[End of Step 2 loop.]

1. Exit

**Algorithm: SEARCH (INFO, LINK,START, ITEM, LOC)**

LIST is a linked list in the memory. This algorithm finds the location LOC of the node where ITEM first appear in LIST, or sets LOC=NULL.

1. Set PTR:=START.

2. Repeat Step 3 while PTR≠NULL:

3. If ITEM = INFO[PTR], then:

Set LOC:=PTR, and Exit. [Search is successful.]

Else:

Set PTR:=LINK[PTR]. [PTR now points to the next node]

[End of If structure.]

[End of Step 2 loop.]

4. Set LOC:=NULL. [Search is unsuccessful.]

5. Exit.

**List is Sorted:**Suppose the data in LIST are sorted. Again we search for ITEM in LIST by traversing the list using a pointer variable PTR and comparing ITEM with the contents INFO[PTR] of each node, one by one, of LIST. Now, however, we can stop once ITEM exceeds INFO[PTR]. (List sorted in descending order)

**Algorithm: SRCHSL (INFO, LINK,START, ITEM, LOC)**LIST is sorted list in memory. This algorithm finds the location LOC of the node where ITEM first appear in LIST, or sets LOC=NULL.

1. Set PTR:=START.  
2. Repeat Step 3 while PTR≠NULL:  
3. If ITEM < INFO[PTR], then:

Set PTR:=LINK[PTR]. [PTR now points to the next node]   
 Else If ITEM=INFO[PTR],then

Set LOC:=PTR, and Exit. [Search is successful.]  
 Else:   
 LOC:=NULL , and Exit. [ITEM now exceeds INFO[PTR].]  
 [End of If structure.]  
 [End of Step 2 loop.]  
4. Set LOC:=NULL.   
5. Exit.

**Program # 1 (A):**

#include <iostream> using namespace std; struct Node {

int data; Node\* next;

};

class LinkedList { private:

Node\* head; public:

LinkedList(){ head = NULL;

}

void push\_back(int newElement) { Node\* newNode = new Node(); newNode->data = newElement; newNode->next = NULL;

if(head == NULL) { head = newNode;

} else {

Node\* temp = head; while(temp->next != NULL) temp = temp->next;

temp->next = newNode;

}

}

void SearchElement(int searchValue) { Node\* temp = head;

int found = 0; int i = 0;

if(temp != NULL) { while(temp != NULL) {

i++;

if(temp->data == searchValue) { found++;

break;

temp = temp->next;

}

if (found == 1) {

cout<<searchValue<<" is found at index = "<<i<<".\n";

} else {

cout<<searchValue<<" is not found in the list.\n";

} else {

cout<<"The list is empty.\n";

}}

void PrintList() { Node\* temp = head; if(temp != NULL) {

cout<<"The list contains: "; while(temp != NULL) { cout<<temp->data<<" "; temp = temp->next;}

cout<<endl;

} else {

cout<<"The list is empty.\n";}

}}

// test the code int main() {

LinkedList MyList;

MyList.push\_back(10);

MyList.push\_back(); MyList.push\_bac0);

MyList.PrintList();

MyList.SearchElement0; MyList.SearchElement5; MyList.SearchElement();return 0;}

**OUTPUT:**Text

Description automatically generated

**LAB No: 8**

**Objective**

Write program to create a Link List and perform different functionality related to it as stated below:

**Operations on Link List**

1. Insertion in Link List
2. Deletion in Link List

**Inserting at the Beginning of a List:**

Suppose our Link List is not necessarily sorted and there is no reason to insert a new node in any special place in the list. Then the easiest place to insert the node is at the beginning of the list. An algorithm the does so follows:

**Algorithm:**

**INSFIRST (INFO, LINK , START, AVAIL, LOC, ITEM)**

This algorithm inserts ITEM as the first node in the list.

Step 1: [OVERFLOW?] If AVAIL=NULL, then Write: OVERFLOW, and Exit.

Step 2: [Remove first node from AVAIL list.]

Set NEW :=AVAIL and AVAI L:=LINK[AVAIL].

Step 3: Set INFO[NEW] := ITEM.. [Copies new data into new node]

Step 4: Set LINK[NEW] := START. [New node now points to the original node]

Step 5: Set START := NEW. [Changes START so it points to the new node.]

Step 6: Exit.

NEW :=AVAIL and AVAI L:=LINK[AVAIL].

LINK[NEW] := START . START := NEW.

**Insertion at Last:**

**Algorithm:**

**INLAST (INFO, LINK, START, AVAIL, LOC, ITEM,TEMP)**

This algorithm inserts ITEM as the last node.

Step 1: [OVERFLOW?] If AVAIL=NULL, then Write: OVERFLOW, and Exit.

Step 2: [Remove first node from AVAIL list.]

Set NEW :=AVAIL and AVAIL:=LINK[AVAIL].

Step 3: Set INFO[NEW] := ITEM. [Copies new data into new node]

Step 4: If LOC := NULL, then: [Inert as first node.]

Set LINK[NEW] := START and START := NEW.

Else: SET TEMP=START

Step 5: Repeat step 6 while LINK[TEMP]!=NULL

Step 6: SET TEMP=LINK[TEMP]

[End of while loop]

LINK[TEMP]=NEW [Insert at Last]

[End of Else structure]

Step 7: Exit

**Inserting into a Sorted Link List (Finding the location at the last node):**

**Algorithm:**

**FINDA (INFO, LINK, START, ITEM, LOC)**

This procedure finds the location LOC of the last node in as sorted list such that INFO[LOC] < ITEM, or sets LOC=NULL.

Step 1: [List Empty?] If START =NULL,

then: Set LOC := NULL, and Return.

[End of If]

Step 2: [Special case?] If ITEM < INFO[START], [ITEM not in the list]

then: Set LOC := NULL, and Return.

[End of If]

Step 3: Set SAVE : = START and PTR := LINK[START]. [Initializes pointers]

Step 4: Repeat step 5 and 6 while PTR ≠ NULL.

Step 5: If ITEM < INFO[PTR], then:

Set LOC : = SAVE, and Return.

[End of If structure.]

Step 6: Set SAVE := PTR and PTR := LINK[PTR]. [Updates pointers.]

[End of step 4 loop]

Step 7: Set LOC := SAVE, and Return.

**Inserting after a given node:**

**Algorithm:**

**INSLOC (INFO, LINK, START, AVAIL, LOC, ITEM)**

This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC=NULL.

Step 1: [OVERFLOW?] If AVAIL=NULL, then Write: OVERFLOW, and Exit.

Step 2: [Remove first node from AVAIL list.]

Set NEW :=AVAIL and AVAIL:=LINK[AVAIL].

Step 3: Set INFO[NEW] := ITEM. [Copies new data into new node]

Step 4: If LOC := NULL, then:[Inert as first node.]

Set LINK[NEW] := START and START := NEW.

Else: [Insert after node with location LOC.]

Set LINK[NEW] := LINK[LOC] and LINK[LOC] := NEW.

[End of If structure]

**DEL (INFO, LINK, START, AVAIL, LOC, LOCP)**

This algorithm deletes the node N with location LOC. LOCP is the location of the node which precedes N or, when N is the first node, LOCP=NULL.

Step 1: If LOCP = NULL, then:

Set START := LINK[START]. [Deletes first node.]

Else:

Set LINK[LOCP]:= LINK[LOC]. [Deletes node N.]

[End of If structure.]

Step 2: [Return deleted node to the AVAIL list.]

Set LINK[LOC] := AVAIL and AVAIL := LOC.

Step 3: Exit.

**DELLAST (LINK, START, AVAIL, LOC, LOCP)**

This algorithm deletes the last node.

Step 1: If START==NULL [Check if the List is empty or not]

Then Write: UNDERFLOW, and Exit

Step 2: If LINK[START]=NULL, then: [If List have only one node]

Set START=NULL [Deletes first node.]

Set LOC=START [If list has more than one node]

[End of If Statement]

Else: SET LOCP=START and LOC=LINK[START] [Set the start pointer

to temporary pointer]

Step 3: Repeat step 4and 5 while LINK[LOC]!=NULL [Find the last node]

Step 4: Set LOCP=LOC [Move both pointers LOC and LOCP forward]

Step 5: Set LOC=LINK[LOC]

Step 6: [End of while loop]

SET LINK[LOCP]=NULL

[End of Else structure]

Step 7: [Return deleted node to the AVAIL list.]

Set LINK[LOC] := AVAIL and AVAIL := LOC.

Step 8: Exit

**Deleting the node with a given item of information**

* **Procedure which finds the location of the first node with a given item of information**

**FINDB(INFO, LINK, START, AVAIL, LOC, LOCP)**

This procedure finds the location LOC of the first node which contains ITEM and the location LOCP of the node preceding N. If ITEM does not appears in the list, then the procedure set LOC=NULL; and if ITEM appears in the first node, then it sets LOCP=NULL.

Step 1: [List Empty?] If START =NULL, then:

Set LOC := NULL, and LOCP := NULL and Return.

[End of If structure.]

Step 2: [Item in first node?] If INFO[START]=ITEM, then:

Set LOC:= START and LOCP := NULL, and Return,

[End of If structure.]

Step 3: Set SAVE := START and PTR := LINK[START]. [Initializes pointers.]

Step 4: Repeat steps 5 and 6 while PTR ≠ NULL.

Step 5: If INFO[PTR] =ITEM, then:

Set LOC := PTR and LOCP := SAVE, and Return.

[End of If structure.]

Step 6: Set SAVE := PTR and PTR := LINK[PTR]. [Updates pointers.]

[End of step 4 loop.]

Step 7: Set LOC := NULL. [Search Unsuccessful.]

Step 8: Return.

**DELETE (INFO, LINK, START, AVAIL, ITEM)**

This algorithm deletes from a linked list the first node N which contains the given ITEM of information.

Step 1: [Use procedure FINDB to find the location of N and its preceding node.]

Call FINDB(INFO, LINK, START, AVAIL, LOC, LOCP)

Step 2: If LOC=NULL, then: Write: ITEM not in list, and Exit.

Step 3: [Delete node.]

If LOCP=NULL, then:

Set START := LINK[START.] [Deletes first node.]

Else

Set LINK[LOCP] := LINK[LOC]. [Deletes node N.]

[End of If structure.]

Step 4: [Return deleted node to the AVAIL list]

Set LINK[LOC] := AVAIL and AVAIL := LOC

Step 5: Exit

**Program**

**LAB No.8**

**Objective:**

* + Write program to create a Single Link List.

# Program # 1 (A):

#include<iostream> using namespace std;

struct node

{

int data; node \*next;

};

class list {

private:

node \*head, \*tail; public:

list() {

head=NULL; tail=NULL;

}

void createnode(int value) {

node \*temp=new node; temp->data=value; temp->next=NULL; if(head==NULL) {

head=temp; tail=temp; temp=NULL;

}

else {

tail>nexmp;tail=temp;

}

}

void display()

{

node \*temp=new node; temp=head; while(temp!=NULL) {

cout<<temp->data<<"\t"; temp=temp->next;

}

}

void insert\_start(int value) {

node \*temp=new node; temp->data=value; temp->next=head; head=temp;

}

void insert\_position(int pos, int value) {

node \*pre=new node; node \*cur=new node; node \*temp=new node; cur=head;

for(int i=1;i<pos;i++) {

pre=cur;

cur=cur->next;

}

temp->data=value; pre->next=temp; temp->next=cur;

}

void delete\_first() {

node \*temp=new node; temp=head;

head=head->next; delete temp;

}

void delete\_last() {

node \*current=new node; node \*previous=new node; current=head; while(current->next!=NULL) {

previous=current; current=current->next;

}

tail=previous; previous->next=NULL; delete current;

}

void delete\_position(int pos) {

node \*current=new node;

node \*previous=new node;

current=head;

for(int i=1;i<pos;i++) {

previous=current; current=current->next;

}

previous->next=current->next;

}

};

int main() {

list obj; obj.createnode(25); obj.createnode(50); obj.createnode(90); obj.createnode(40);

cout<<"\n \n"; cout<<"---------------Displaying All nodes ";

cout<<"\n \n"; obj.display();

cout<<"\n \n"; cout<<" Inserting At End- "; cout<<"\n \n"; obj.createnode(55);

obj.display();

cout<<"\n \n"; cout<<"----------------Inserting At Start ";

cout<<"\n \n"; obj.insert\_start(50);

obj.display();

cout<<"\n \n"; cout<<"-------------Inserting At Particular ";

cout<<"\n \n"; obj.insert\_position(5,60);

obj.display();

cout<<"\n \n"; cout<<"----------------Deleting At Start ";

cout<<"\n \n"; obj.delete\_first();

obj.display();

cout<<"\n \n"; cout<<" Deleing At End "; cout<<"\n \n"; obj.delete\_last();

obj.display();

cout<<"\n \n"; cout<<"--------------Deleting At Particular ";

cout<<"\n \n"; obj.delete\_position(4);

obj.display();

cout<<"\n \n"; system("pause");

return 0;

}

# OUTPUT:

Text

Description automatically generated

**Program#2:**

#include<iostream> #include<conio.h> #include<stdlib.h> using namespace std; class Node {

public:

int info;

Node\* next;

};

class List:public Node

{

Node \*first,\*last; public:

List() {

first=NULL; last=NULL;}

void create(); void insert(); void delet(); void display(); void search();

};

void List::create() {

Node \*temp; temp=new Node; int n;

cout<<"\nEnter an Element:"; cin>>n;

temp->info=n; temp->next=NULL; if(first==NULL) {

first=temp; last=first;

else {

last->next=temp; last=temp;

}

}

void List::insert() {

Node \*prev,\*cur; prev=NULL; cur=first;

int count=1,pos,ch,n; Node \*temp=new Node;

cout<<"\nEnter an Element:"; cin>>n;

temp->info=n;

temp->next=NULL;

cout<<"\nINSERT AS\n1:FIRSTNODE\n2:LASTNODE\n3:IN BETWEEN FIRST&LAST NODES";

cout<<"\nEnter Your Choice:"; cin>>ch;

switch(ch)

{

case 1:

temp->next=first; first=temp; break;

case 2:

last->next=temp;

last=temp; break;

case 3:

cout<<"\nEnter the Position to Insert:"; cin>>pos;

while(count!=pos) {

prev=cur; cur=cur->next; count++;

}

if(count==pos) {

prev->next=temp; temp->next=cur;

}

else

cout<<"\nNot Able to Insert"; break;

}

}

void List::delet() {

Node \*prev=NULL,\*cur=first; int count=1,pos,ch;

cout<<"\nDELETE\n1:FIRSTNODE\n2:LASTNODE\n3:IN BETWEEN FIRST&LAST NODES";

cout<<"\nEnter Your Choice:"; cin>>ch;

switch(ch)

{

case 1:

if(first!=NULL)

{

cout<<"\nDeleted Element is "<<first->info; first=first->next;

}

else

cout<<"\nNot Able to Delete"; break;

case 2:

while(cur!=last)

{

prev=cur; cur=cur->next;

}

if(cur==last) {

cout<<"\nDeleted Element is: "<<cur->info; prev->next=NULL;

last=prev;}

else

cout<<"\nNot Able to Delete";

break;

case 3:

cout<<"\nEnter the Position of Deletion:"; cin>>pos;

while(count!=pos) {

prev=cur; cur=cur->next; count++;

}

if(count==pos) {

cout<<"\nDeleted Element is: "<<cur->info; prev->next=cur->next;

}

else

cout<<"\nNot Able to Delete"; break;

}

}

void List::display() {

Node \*temp=first; if(temp==NULL) {

cout<<"\nList is Empty";

}

while(temp!=NULL) {

cout<<temp->info; cout<<"-->"; temp=temp->next;

}

cout<<"NULL";

}

void List::search() {

int value,pos=0; bool flag=false; if(first==NULL)

{

cout<<"List is Empty"; return;

}

cout<<"Enter the Value to be Searched:"; cin>>value;

Node \*temp; temp=first; while(temp!=NULL{pos++;

if(temp->info==value) {

flag=true;

cout<<"Element"<<value<<"is Found at "<<pos<<" Position"; return;

temp=temp->next;

}

if(!flag) {

cout<<"Element "<<value<<" not Found in the List";}}

int main() {

List l; int ch; while(1

cout<<"\n\*\*\*\* MENU \*\*\*\*"; cout<<"\n1:CREATE\n2:INSERT\n3:DELETE\n4:SEARCH\n5:DISPLAY\n6:EXIT\n";

cout<<"\nEnter Your Choice:"; cin>>ch;

switch(ch) {

case 1:

* 1. reate(); break;

case 2:

l.insert(); break;

case 3:

* 1. elet(); break;

case 4:

l.search(); break;

case 5:

l.display(); break;

case 6:

return 0;

}}

return 0;}

# Text Description automatically generatedOUTPUT:

**LAB No.9**

**Objective:**

* + Evaluate Postfix Expression

**Algorithms:**

Algorithm: StackCalculator

Private:

mystack: stack of strings

Function isOperator(opr):

If length(opr) = 1:

op = "+-\*/"

If opr is in op:

Return true

Else:

Return false

Else:

Return false

Function isNumber(number):

If length(number) > 0:

For each character c in number:

If c is not a digit:

Return false

Return true

Else:

Return false

Function evaluate(stck):

op = stck.top()

stck.pop()

value1 = stck.top()

stck.pop()

value2 = stck.top()

stck.pop()

val1 = convert\_to\_double(value1)

val2 = convert\_to\_double(value2)

result = 0

If op is "+":

result = val1 + val2

Else If op is "-":

result = val2 - val1

Else If op is "\*":

result = val1 \* val2

Else If op is "/":

result = val2 / val1

stck.push(convert\_to\_string(result))

Public:

Function push(str):

temp = ""

If size(mystack) > 0:

top(temp)

If isOperator(str) and isNumber(temp) and size(mystack) = 1:

Print "Enter another value before performing operation"

Return

Else:

mystack.push(str)

If isOperator(str):

evaluate(mystack)

Function pop():

mystack.pop()

Function top(str):

str = mystack.top()

Function main():

myStack = new StackCalculator()

input = ""

Print "Enter a number or Q to quit: "

Read input

While input is not equal to "Q" and input is not equal to "q":

myStack.push(input)

Print "Enter a number or Q to quit: "

Read input

temp = ""

myStack.top(temp)

If length(temp) > 0:

Print "The value on the top of the stack is: " + formatted(temp)

# Program # 1 (A):

#include <iostream>

#include <stack> #include<string> #include<stdlib.h> #include<iomanip>

using namespace std;

class classStack {

private:

stack<string> mystack;

bool isOperator(string &opr){ if(opr.length() == 1) {

string op ="+-/\*";

if(op.find(opr) != std::string::npos){ return true;

}

else {

return false;}}

else {

return false;

}

}

bool isNumber(string &number){ if(number.length()>0){

for(int i=0; i<number.length(); i++){ if( !isdigit(number.at(i))) {

return false;

}}

return true;}

else {

return false;}}

void evaluate(stack<string> & stck){ string op = stck.top();

stck.pop();

string value1 = stck.top(); stck.pop(); string value2 = stck.top(); stck.pop();

double val1 = atoi(value1.c\_str()); double val2 = atoi(value2.c\_str()); double result = 0;

if(op.compare("+") == 0) result = val1 + val2;

else if(op.compare("-") ==0) result = val2-val1;

else if(op.compare("\*") ==0) result = val1\*val2;

else if(op.compare("/") ==0) result = val2/val1;

char buffer[10] ;

sprintf(buffer,"%lf", result);

stck.push(string(buffer));

}

public:

void push(string &str){

string temp; if(mystack.size()) top(temp);

if(isOperator(str) && isNumber(temp)

&& mystack.size()==1) {

cout<<"\nEnter another value before"; cout<<" performing operation"; return;}

else {

mystack.push(str);

if(isOperator(str)){ evaluate(mystack);

}}

void pop(){ mystack.pop();}

void top(string &str){ str = mystack.top();

}};

int main(int argc, char\*\* argv) {

classStack myStack;

string input;

cout<<"\n Enter a number of Q to quit :";

cin>>input;

while(input.compare("Q") !=0 &&

input.compare("q") !=0 ){ myStack.push(input);

cout<<"\nEnter a number or Q to quit :"; cin>>input;

}

string temp;

myStack.top(temp); if(temp.length())

cout<<"\nThe value on the top of the stack is: "

<< std::fixed<<std::setprecision(2)<<temp<<endl;

return 0;

}

# OUTPUT:

Text

Description automatically generated

**LAB No.10**

**Objective:**

* + Convert Infix to Postfix Expression

**Algorithms:**

Algorithm: InfixToPostfix

Function isOperator(character):

If character is '+' or '-' or '\*' or '/':

Return true

Else:

Return false

Function isOperand(character):

If not isOperator(character) and character is not '(' and character is not ')':

Return true

Else:

Return false

Function compareOperators(op1, op2):

If (op1 is '\*' or '/') and (op2 is '+' or '-'):

Return -1

Else If (op1 is '+' or '-') and (op2 is '\*' or '/'):

Return 1

Else:

Return 0

Function InfixToPostfix(infixExpression):

opStack = Stack of characters

postFixString = Empty string

For each character in infixExpression:

If isOperand(character):

Append character to postFixString

Else If isOperator(character):

While opStack is not empty and opStack.top() is not '(' and compareOperators(opStack.top(), character) <= 0:

Append opStack.top() to postFixString

opStack.pop()

Push character onto opStack

Else If character is '(':

Push character onto opStack

Else If character is ')':

While opStack is not empty:

If opStack.top() is '(':

opStack.pop()

Break

Append opStack.top() to postFixString

opStack.pop()

While opStack is not empty:

Append opStack.top() to postFixString

opStack.pop()

Return postFixString

# Program # 1 (A):

#include <iostream> #include <stack> #include <string> using namespace std;

bool isOperator(char character) {

if (character == '+' || character == '-' || character == '\*' || character == '/') { return true;

}

return false;

}

bool isOperand(char character) {

if (!isOperator(character) && character != '(' && character != ')') { return true;

}

return false;

}

int compareOperators(char op1, char op2) {

if ((op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-')) { return -1; }

else if ((op1 == '+' || op1 == '-') && (op2 == '\*' || op2 == '/')) { return 1; }

return 0;

}

int main() {

stack<char> opStack; string postFixString = ""; char input[100];

Collect input

cout << "Enter an expression: ";

cin >> input;

char \*cPtr = input;

while (\*cPtr != '\0') {

if (isOperand(\*cPtr)) { postFixString += \*cPtr;

else if (isOperator(\*cPtr)) {

while (!opStack.empty() && opStack.top() != '(' && compareOperators(opStack.top(),\*cPtr) <=

0) {

postFixString += opStack.top();

opStack.pop();

}

opStack.push(\*cPtr);

}

else if (\*cPtr == '(') { opStack.push(\*cPtr); }

else if (\*cPtr == ')') {

while (!opStack.empty()) {

if (opStack.top() == '(') { opStack.pop(); break; } postFixString += opStack.top();

opStack.pop();

}

}

// Advance our pointer to next character in string. cPtr++;

}

while (!opStack.empty()) { postFixString += opStack.top(); opStack.pop();

}

cout << "Postfix is: " << postFixString << endl;

return 0;

}

# Text Description automatically generatedOUTPUT:

**Algorithms:**

Algorithm: InfixToPostfix

Function preced(ch):

If ch is '+' or '-':

Return 1

Else If ch is '\*' or '/':

Return 2

Else If ch is '^':

Return 3

Else:

Return 0

Function inToPost(infix):

stk = Stack of characters

postfix = Empty string

For each character in infix:

If character is alphanumeric:

Append character to postfix

Else If character is '(':

Push character onto stk

Else If character is '^':

Push character onto stk

Else If character is ')':

While stk.top() is not '#' and stk.top() is not '(':

Append stk.top() to postfix

stk.pop()

stk.pop()

Else:

If preced(character) > preced(stk.top()):

Push character onto stk

Else:

While stk.top() is not '#' and preced(character) <= preced(stk.top()):

Append stk.top() to postfix

stk.pop()

Push character onto stk

While stk.top() is not '#':

Append stk.top() to postfix

stk.pop()

Return postfix

**Program #2:**

#include<iostream> #include<stack>

#include<locale> //for function isalnum() using namespace std;

int preced(char ch) { if(ch == '+' || ch == '-') {

return 1; //Precedence of + or - is 1

}else if(ch == '\*' || ch == '/') {

return 2; //Precedence of \* or / is 2

}else if(ch == '^') {

return 3; //Precedence of ^ is 3

}else { return 0;

}

}

string inToPost(string infix ) { stack<char> stk;

stk.push('#'); //add some extra character to avoid underflow

string postfix = ""; //initially the postfix string is empty string::iterator it;

for(it = infix.begin(); it!=infix.end(); it++) { if(isalnum(char(\*it)))

postfix += \*it; //add to postfix when character is letter or number else if(\*it == '(')

stk.push('('); else if(\*it == '^') stk.push('^');

else if(\*it == ')') {

while(stk.top() != '#' && stk.top() != '(') {

postfix += stk.top(); //store and pop until ( has found stk.pop();

}

stk.pop(); //remove the '(' from stack

}else {

if(preced(\*it) > preced(stk.top())) stk.push(\*it); //push if precedence is high

else {

while(stk.top() != '#' && preced(\*it) <= preced(stk.top())) {

postfix += stk.top(); //store and pop until higher precedence is found stk.pop();

}

stk.push(\*it);

}

}

}

while(stk.top() != '#') {

postfix += stk.top(); //store and pop until stack is not empty. stk.pop();

}

return postfix;

}

int main() {

string infix = "x^y/(5\*z)+2";

cout << "Postfix Form Is: " << inToPost(infix) << endl;

}

# OUTPUT:Text Description automatically generated

**Program#2:**

#include <iostream>

#include <stack>

using namespace std;

// Function to evaluate given postfix expression int postfixEval(string exp) {

// create an empty stack stack<int> stack;

// traverse the given expression for (int i = 0; i < exp.length(); i++) {

// if current char is an operand, push it to the stack if (exp[i] >= '0' && exp[i] <= '9') {

stack.push(exp[i] - '0');}

// if current char is an operator else {

// pop top two elements from the stack int x = stack.top();

stack.pop();

int y = stack.top(); stack.pop();

// evaluate the expression x op y, and push the

// result back to the stack if (exp[i] == '+')

stack.push(y + x);

else if (exp[i] == '-')

stack.push(y - x);

else if (exp[i] == '\*') stack.push(y \* x);

else if (exp[i] == '/') stack.push(y / x);

}

}

// At this point, the stack is left with only one element

// i.e. expression result return stack.top();

}

int main() {

string exp = "138\*+";

cout << postfixEval(exp);

return 0;

}

# OUTPUT:Text Description automatically generated

**LAB No.11**

**Objective:**

**Algorithms:**

Algorithm: QueueImplementation

Function enqueue(num):

If rear is equal to max:

Print "OVERFLOW!"

Else If front is -1 and rear is -1:

Set front to 0, rear to 0

Set queue[rear] to num

Else:

Increment rear

Set queue[rear] to num

Function dequeue():

If front is -1 or front is greater than rear:

Print "UNDERFLOW!"

Return -1

Else:

Print "The deleted data is: " + queue[front]

Increment front

Return queue[front-1]

Function show():

Set i to front

If front is -1 or front is greater than rear:

Print "UNDERFLOW!"

Else:

While i is less than or equal to rear:

Print queue[i]

Increment i

# Program # 1 (A):

#include<iostream> using namespace std; #define max 10

int queue[max],front=-1,rear=-1; //queue declaration

void enqueue(int num) //enqueue() inserts an element into the Queue {

if(rear==max) //check if Queue is full {

cout<<"OVERFLOW!";

}

else if(front==-1 && rear==-1) //For 1st insertion in Queue {

front++; rear++;

queue[rear]=num;

}

else {

rear++; queue[rear]=num;}

}

int dequeue() //dequeue() deletes out the 1st element from the Queue {

if(front==-1 || front>rear) //check if Queue is empty {

cout<<"UNDERFLOW!";

return -1;

}

else {

cout<<"The deleted data is : "<<queue[front++]; //printing the deleted element return queue[front-1];

}}

void show() {

int i=front;

if(front==-1 || front>rear) //if Queue is empty {

cout<<"UNDERFLOW!";

}

else {

while(i<=rear) //printing the current Queue elements {

cout<<"\t"<<queue[i]; i++;

}

cout<<endl;}

}

int main() {

int ch,val;

cout<<" :::MENU:::"; //Menu for Queue operations cout<<"\n1.enqueue\n2.dequeue\n3.Show\n4.Exit"; while(1) {

printf("\nEnter the choice:"); scanf("%d",&ch);

switch(ch) {

case 1: cout<<"Enter the value to be pushed: "; cin>>val;

enqueue(val);

break;

case 2: dequeue(); break;

case 3: cout<<"Stack : "; show();

break; case 4: exit(0);

default: printf("\nError! Invalid choice!...");

}}

return 0;

# Text Description automatically generatedOUTPUT:

**Algorithms:**

Algorithm: CircularQueue

Class Queue:

Data Members:

arr: array to store queue elements

capacity: maximum capacity of the queue

front: front points to front element in the queue (if any)

rear: rear points to last element in the queue

count: current size of the queue

Constructor Queue(size):

Initialize arr with new int[size]

Set capacity to size

Set front to 0

Set rear to -1

Set count to 0

Destructor ~Queue():

Delete[] arr

Function dequeue():

If isEmpty():

Print "UnderFlow\nProgram Terminated"

Exit(EXIT\_FAILURE)

Print "Removing " + arr[front]

Set front to (front + 1) % capacity

Decrement count

Function enqueue(item):

If isFull():

Print "OverFlow\nProgram Terminated"

Exit(EXIT\_FAILURE)

Print "Inserting " + item

Set rear to (rear + 1) % capacity

Set arr[rear] to item

Increment count

Function peek():

If isEmpty():

Print "UnderFlow\nProgram Terminated"

Exit(EXIT\_FAILURE)

Return arr[front]

Function size():

Return count

Function isEmpty():

Return size() is 0

Function isFull():

Return size() is capacity

Function main():

Create a Queue q with capacity 5

Call q.enqueue(1)

Call q.enqueue(2)

Call q.enqueue(3)

Print "Front element is: " + q.peek()

Call q.dequeue()

Call q.enqueue(4)

Print "Queue size is " + q.size()

Call q.dequeue()

Call q.dequeue()

Call q.dequeue()

If q.isEmpty():

Print "Queue Is Empty"

Else:

Print "Queue Is Not Empty"

**Program#2:**

#include <iostream> #include <cstdlib> using namespace std; #define SIZE 10

class queue {

int \*arr; // array to store queue elements

int capacity; // maximum capacity of the queue

int front; // front points to front element in the queue (if any) int rear; // rear points to last element in the queue

int count; // current size of the queue

public:

queue(int size = SIZE); // constructor

~queue(); // destructor

void dequeue();

void enqueue(int x); int peek();

int size();

}

bool isEmpty(); bool isFull();

};

// Constructor to initialize queue queue::queue(int size) {

arr = new int[size]; capacity = size; front = 0;

rear = -1;

count = 0;

}

// Destructor to free memory allocated to the queue queue::~queue() {

delete[] arr;

}

// Utility function to remove front element from the queue void queue::dequeue() {

// check for queue underflow if (isEmpty()) {

cout << "UnderFlow\nProgram Terminated\n"; exit(EXIT\_FAILURE);

}

cout << "Removing " << arr[front] << '\n';

front = (front + 1) % capacity; count--;

}

// Utility function to add an item to the queue void queue::enqueue(int item) {

// check for queue overflow if (isFull()) {

cout << "OverFlow\nProgram Terminated\n"; exit(EXIT\_FAILURE);

}

cout << "Inserting " << item << '\n';

rear = (rear + 1) % capacity; arr[rear] = item;

count++;

}

// Utility function to return front element in the queue int queue::peek() {

if (isEmpty()) {

cout << "UnderFlow\nProgram Terminated\n"; exit(EXIT\_FAILURE);

}

return arr[front];

}

// Utility function to return the size of the queue int queue::size() {

return count;

}

// Utility function to check if the queue is empty or not bool queue::isEmpty() {

return (size() == 0);

}

// Utility function to check if the queue is full or not bool queue::isFull() {

return (size() == capacity);

}

int main() {

// create a queue of capacity 5 queue q(5);

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

cout << "Front element is: " << q.peek() << endl; q.dequeue();

q.enqueue(4);

cout << "Queue size is " << q.size() << endl;

q.dequeue();

q.dequeue();

q.dequeue();

if (q.isEmpty())

cout << "Queue Is Empty\n"; else

cout << "Queue Is Not Empty\n";

return 0;

}

# Text Description automatically generatedOUTPUT: