**Practical 1:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem Statement 01 : Store ‘N’ names of items in a 2-D array. Perform following (operations) sorting technique :

a) Insertion sort

b) Selection sort\*/

Write a menu driven cpp program to demonstrate .

#include <iostream>

#include <string>

using namespace std;

void insertionSort(string arr[], int n) {

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

string key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

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

j--;

}

arr[j + 1] = key;

}

}

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

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

int min = i;

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

if (arr[j] < arr[min]) {

min = j;

}

}

if (min != i) {

swap(arr[i], arr[min]);

}

}

}

void displayNames(string arr[], int n) {

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

cout << arr[i] << endl;

}

}

int main() {

int n, choice;

cout << "Enter the number of items: ";

cin >> n;

string items[n];

cout << "Enter the names of " << n << " items:" << endl;

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

cout << "Item " << i + 1 << ": ";

cin >> items[i];

}

while (true) {

cout << "\n1) Sort using Insertion Sort" << endl;

cout << "2) Sort using Selection Sort" << endl;

cout << "3) Display items" << endl;

cout << "4) Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

insertionSort(items, n);

cout << "Items sorted using Insertion Sort." << endl;

break;

case 2:

selectionSort(items, n);

cout << "Items sorted using Selection Sort." << endl;

break;

case 3:

cout << "Items:" << endl;

displayNames(items, n);

break;

case 4:

exit(0);

default:

cout << "Invalid choice! Try again." << endl;

}

}

return 0;

}

OUTPUT:



**Practical 2:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem Statement: Store N integer in an array and perform following sorting technique :

a) Quick sort

b) Merge sort\*/

#include <iostream>

using namespace std;

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int i = 0; i < n2; i++)

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

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

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

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

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

}

cout << endl;

}

int main() {

int n, choice;

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

cin >> n;

int arr[n];

cout << "Enter the elements:" << endl;

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

cin >> arr[i];

}

while (true) {

cout << "\n1) Quick Sort" << endl;

cout << "2) Merge Sort" << endl;

cout << "3) Display array" << endl;

cout << "4) Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

quickSort(arr, 0, n - 1);

cout << "Array sorted using Quick Sort." << endl;

break;

case 2:

mergeSort(arr, 0, n - 1);

cout << "Array sorted using Merge Sort." << endl;

break;

case 3:

cout << "Array: ";

display(arr, n);

break;

case 4:

exit(0);

default:

cout << "Invalid choice! Try again." << endl;

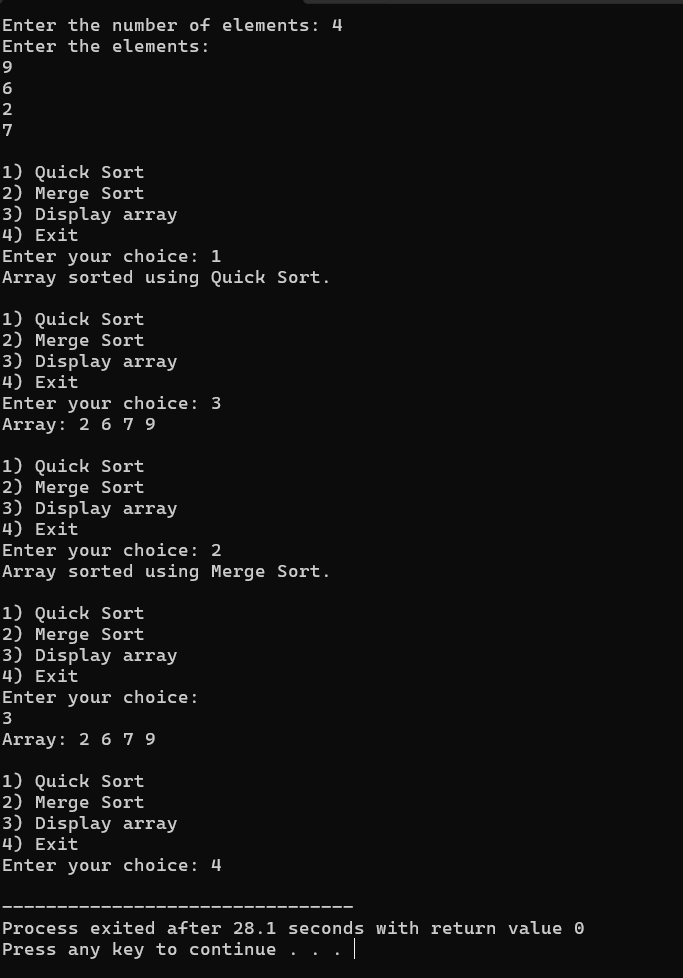
}

}

return 0;

}

OUTPUT:



**Practical 3:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem Statement: Implement Stack of characters using linked list and perform following operations :

a)Create N nodes

b) Insert in between with both cases after a specific Node and before a specific Node

c) display \*/

#include <iostream>

using namespace std;

struct Node {

char data;

Node\* next;

};

class Stack {

Node\* top;

public:

Stack() {

top = nullptr;

}

void createNodes(int n) {

char value;

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

cout << "Enter character for node " << i + 1 << ": ";

cin >> value;

push(value);

}

}

void push(char value) {

Node\* newNode = new Node();

newNode->data = value;

newNode->next = top;

top = newNode;

}

void insertAfter(char afterValue, char newValue) {

Node\* current = top;

while (current != nullptr && current->data != afterValue) {

current = current->next;

}

if (current == nullptr) {

cout << "Node with value " << afterValue << " not found." << endl;

return;

}

Node\* newNode = new Node();

newNode->data = newValue;

newNode->next = current->next;

current->next = newNode;

}

void insertBefore(char beforeValue, char newValue) {

if (top == nullptr) {

cout << "Stack is empty." << endl;

return;

}

if (top->data == beforeValue) {

push(newValue);

return;

}

Node\* current = top;

Node\* prev = nullptr;

while (current != nullptr && current->data != beforeValue) {

prev = current;

current = current->next;

}

if (current == nullptr) {

cout << "Node with value " << beforeValue << " not found." << endl;

return;

}

Node\* newNode = new Node();

newNode->data = newValue;

newNode->next = current;

prev->next = newNode;

}

void display() {

Node\* current = top;

if (current == nullptr) {

cout << "Stack is empty." << endl;

return;

}

cout << "Stack elements: ";

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

};

int main() {

Stack stack;

int n;

char afterValue, beforeValue, newValue;

int choice;

cout << "Enter the number of nodes to create: ";

cin >> n;

stack.createNodes(n);

while (true) {

cout << "\nMenu:" << endl;

cout << "1) Insert after a specific node" << endl;

cout << "2) Insert before a specific node" << endl;

cout << "3) Display stack" << endl;

cout << "4) Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the value after which to insert: ";

cin >> afterValue;

cout << "Enter new value to insert: ";

cin >> newValue;

stack.insertAfter(afterValue, newValue);

break;

case 2:

cout << "Enter the value before which to insert: ";

cin >> beforeValue;

cout << "Enter new value to insert: ";

cin >> newValue;

stack.insertBefore(beforeValue, newValue);

break;

case 3:

stack.display();

break;

case 4:

exit(0);

default:

cout << "Invalid choice! Try again." << endl;

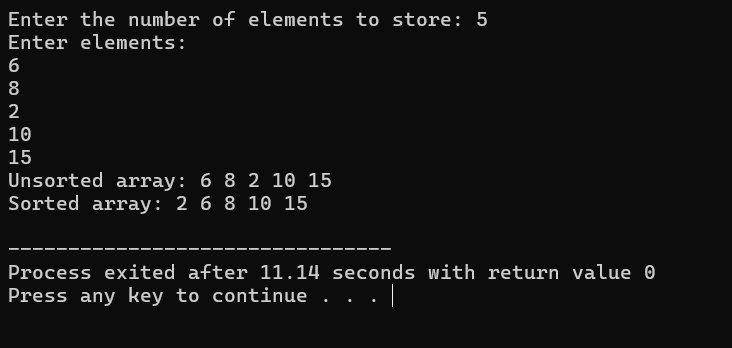
}

}

return 0;

}

OUTPUT:



**Practical 4:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem Statement: Implement Linked List of number and perform following operation :

a) Create N nodes

b) add front and delete front

c) delete specified Node \*/

#include<iostream>

using namespace std;

struct node{

int data;

node\* link;

};

class clinkl

{

node\* last;

public:

clinkl()

{

last=NULL;

}

void create();

void display();

void addf();

void delf();

void delr();

void del\_sp();

};

void clinkl::create()

{

int n;

node\* nn,\*temp;

cout<<"enter number of node:";

cin>>n;

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

{

nn=new node;

if(nn==NULL)

{

cout<<"memory issue";

}

cout<<"enter data:";

cin>>nn->data;

nn->link=NULL;

if(i==0)

{

last=nn;

nn->link=nn;

}

nn->link=last->link;

last->link=nn;

last=nn;

}

}

void clinkl::display()

{

node\* t;

t=last->link;

do

{

cout<<t->data<<"\t";

t=t->link;

}

while(t!=last->link);

}

void clinkl::addf()

{

node\* temp;

temp=new node;

cout<<"\nenter data at front:";

cin>>temp->data;

temp->link=last->link;

last->link=temp;

}

void clinkl::delf()

{

node \*t;

cout<<"\ndeleted node linked list is:";

t=last->link;

last->link=t->link;

delete(t);

}

void clinkl::delr()

{

cout<<"\ndeleting the last node:";

node \*t;

t=last->link;

while(t->link!=last)

{

t=t->link;

}

t->link=last->link;

delete(last);

last=t;

}

void clinkl::del\_sp()

{

int pos;

node \*t,\*prev;

prev=NULL;

cout<<"\nenter the position:";

cin>>pos;

t=last->link;

while(--pos)

{

prev=t;

t=t->link;

}

prev->link=t->link;

delete(t);

}

int main()

{

clinkl c1;

c1.create();

c1.display();

c1.addf();

c1.display();

c1.delf();

c1.display();

c1.delr();

c1.display();

c1.del\_sp();

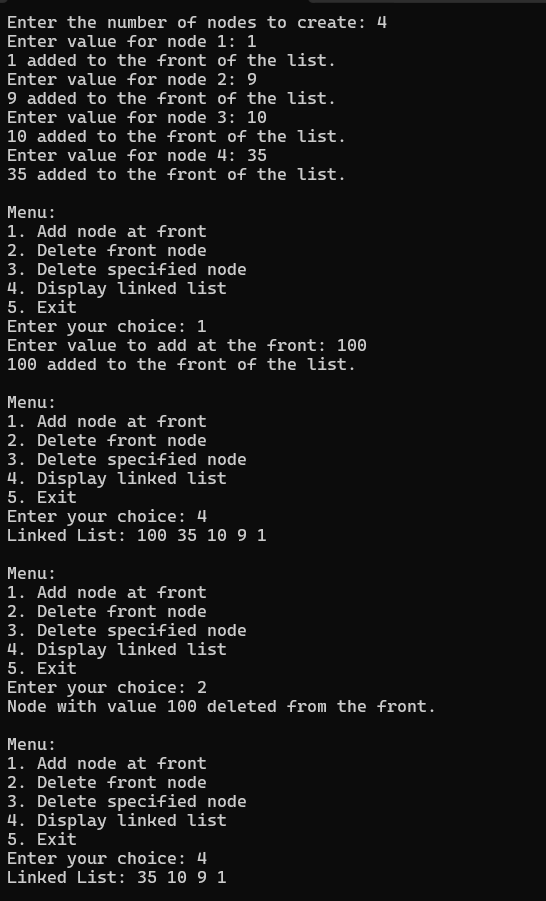
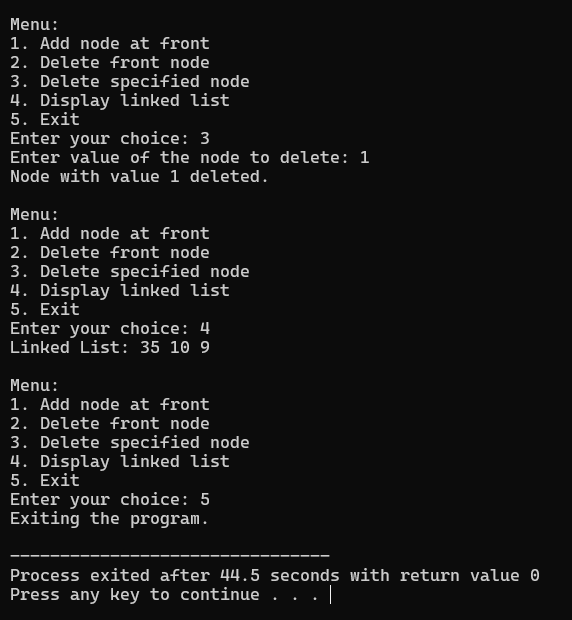
cout<<"\nafter deleting the specified circular link list is:";

c1.display();

return 0;

}

OUTPUT:

**Practical 5:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem Statement: Implement circular Linked list of integer and perform following operation :

a) Create N nodes

b)add front and delete front

c) delete specified Node \*/

#include<iostream>

using namespace std;

struct node{

int data;

node\* link;

};

class clinkl

{

node\* last;

public:

clinkl()

{

last=NULL;

}

void create();

void display();

void addf();

void delf();

void delr();

void del\_sp();

};

void clinkl::create()

{

int n;

node\* nn,\*temp;

cout<<"enter number of node:";

cin>>n;

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

{

nn=new node;

if(nn==NULL)

{

cout<<"memory issue";

}

cout<<"enter data:";

cin>>nn->data;

nn->link=NULL;

if(i==0)

{

last=nn;

nn->link=nn;

}

nn->link=last->link;

last->link=nn;

last=nn;

}

}

void clinkl::display()

{

node\* t;

t=last->link;

do

{

cout<<t->data<<"\t";

t=t->link;

}

while(t!=last->link);

}

void clinkl::addf()

{

node\* temp;

temp=new node;

cout<<"\nenter data at front:";

cin>>temp->data;

temp->link=last->link;

last->link=temp;

}

void clinkl::delf()

{

node \*t;

cout<<"\ndeleted node linked list is:";

t=last->link;

last->link=t->link;

delete(t);

}

void clinkl::delr()

{

cout<<"\ndeleting the last node:";

node \*t;

t=last->link;

while(t->link!=last)

{

t=t->link;

}

t->link=last->link;

delete(last);

last=t;

}

void clinkl::del\_sp()

{

int pos;

node \*t,\*prev;

prev=NULL;

cout<<"\nenter the position:";

cin>>pos;

t=last->link;

while(--pos)

{

prev=t;

t=t->link;

}

prev->link=t->link;

delete(t);

}

int main()

{

clinkl c1;

c1.create();

c1.display();

c1.addf();

c1.display();

c1.delf();

c1.display();

c1.delr();

c1.display();

c1.del\_sp();

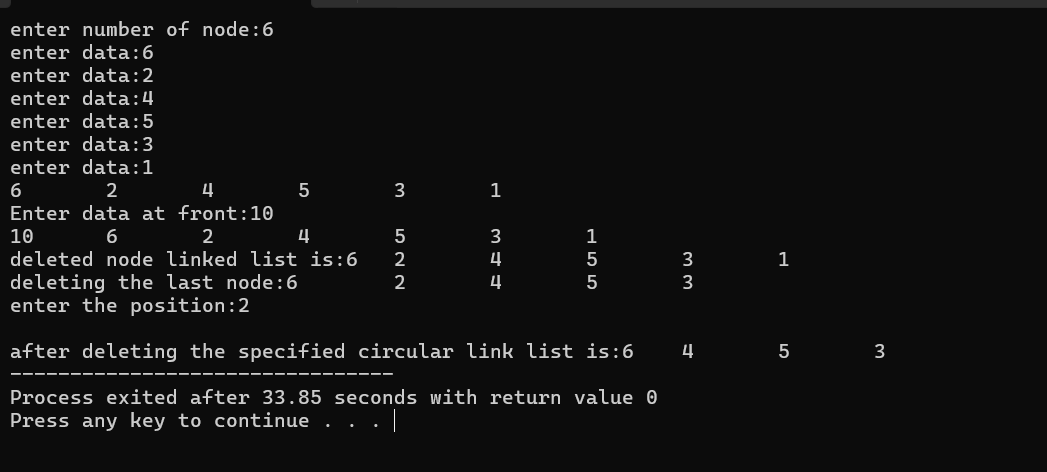
cout<<"\nafter deleting the specified circular link list is:";

c1.display();

return 0;

}

OUTPUT:



**Practical 6:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem Statement: Implement Stack of numbers (double-type) and write a program to evaluate a given postfix expression, if it is given as input \*/

**INPUT:**

#include<iostream>

using namespace std;

const int size = 80;

class Stack {

double a[size];

int top;

public:

Stack() { top = -1; }

void push(double x);

double pop();

void display();

int empty() { return top == -1; }

};

void Stack::push(double x) {

if (top == size - 1) {

cout << "Stack overflow\n";

return;

}

a[++top] = x;

}

double Stack::pop() {

if (top == -1) {

cout << "It's an invalid postfix expression\n";

exit(0);

}

return a[top--];

}

void Stack::display() {

for (int i = top; i >= 0; i--)

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

}

int isoperand(char ch) {

return (ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9');

}

int main() {

Stack s1;

char postexp[80], symb;

double result, op1, op2, x, temp;

cout << "Enter a valid postfix expression\n";

cin >> postexp;

for (int i = 0; postexp[i] != '\0'; i++) {

symb = postexp[i];

if (isoperand(symb)) {

if (symb >= 'A' && symb <= 'Z' || symb >= 'a' && symb <= 'z') {

cout << "Enter value for " << symb << " : ";

cin >> x;

s1.push(x);

} else {

s1.push(symb - '0');

}

} else {

op2 = s1.pop();

op1 = s1.pop();

switch (symb) {

case '+': temp = op1 + op2; break;

case '-': temp = op1 - op2; break;

case '\*': temp = op1 \* op2; break;

case '/': temp = op1 / op2; break;

case '%': temp = static\_cast<int>(op1) % static\_cast<int>(op2); break;

}

s1.push(temp);

}

}

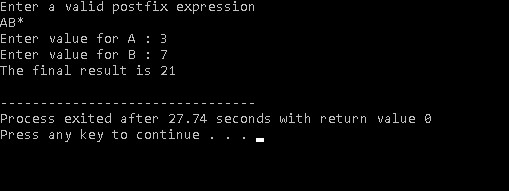
result = s1.pop();

cout << "The final result is " << result << endl;

return 0;

}

OUTPUT:



**Practical 7:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem Statement: Implement Stack of character and read a valid infix expression and perform following expression conversions. a)Infix to postfix . b)Infix to prefix.\*/

**INPUT:**

#include<iostream>

using namespace std;

const int size = 80;

class Stack {

char a[size]; // Array to store stack elements

int top; // Index for the top of the stack

public:

Stack() { top = -1; } // Constructor initializes top to -1 (empty stack)

void push(char x) {

if (top == size - 1) { // Check for stack overflow

cout << "Stack overflow\n";

return;

}

a[++top] = x; // Increment top and add the element

}

// Pop function to remove elements from the stack

char pop() {

if (top == -1) { // Check for stack underflow

cout << "Stack underflow\n";

exit(0);

}

return a[top--];

}

int empty() {

return (top == -1);

}

};

int precedence(char ch) {

if (ch == '$') return 3;

else if (ch == '\*' || ch == '/') return 2;

else if (ch == '+' || ch == '-') return 1; // Addition/Subtraction precedence

return 0; // For non-operators like parentheses

}

int isOperand(char ch) {

return (ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9');

}

void convertToPostfix(char\* infix, char\* postfix) {

Stack s1; // Stack to hold operators and parentheses

char symbol, stackSymbol;

int i = 0, j = 0, flag;

while (infix[i] != '\0') {

symbol = infix[i];

if (isOperand(symbol)) {

postfix[j++] = symbol;

}

// If the symbol is '(', push it onto the stack

else if (symbol == '(') {

s1.push(symbol);

}

// If the symbol is an operator, handle precedence

else if (symbol == '+' || symbol == '-' || symbol == '\*' || symbol == '/' || symbol == '$')

{

if (s1.empty()) { // If stack is empty, push the operator onto the stack

s1.push(symbol);

} else {

flag = 0;

stackSymbol = s1.pop(); // Pop the top of the stack

// Compare precedence and add to postfix as needed

while (precedence(stackSymbol) >= precedence(symbol)) {

postfix[j++] = stackSymbol;

if (s1.empty()) {

flag = 1; // Stack is empty, break the loop

break;

}

stackSymbol = s1.pop(); // Pop next element for comparison

}

// Push the previously popped operator back onto the stack

if (flag == 0) {

s1.push(stackSymbol);

}

// Finally, push the current operator onto the stack

s1.push(symbol);

}

}

// If the symbol is ')', pop and append everything until '('

else if (symbol == ')') {

stackSymbol = s1.pop();

while (stackSymbol != '(') {

postfix[j++] = stackSymbol;

stackSymbol = s1.pop(); // Continue popping until '(' is found

}

}

i++;

}

// Pop remaining operators from the stack and append to postfix expression

while (!s1.empty()) {

postfix[j++] = s1.pop();

}

postfix[j] = '\0'; // Null-terminate the postfix expression

}

// Function to convert infix expression to prefix expression

void convertToPrefix(char\* infix, char\* prefix) {

Stack s1; // Stack to hold operators and parentheses

char symbol, stackSymbol;

int i = 0, j = 0, flag;

// Reverse the infix expression

int len = 0;

while (infix[len] != '\0') len++;

for (int k = 0; k < len / 2; k++) {

swap(infix[k], infix[len - 1 - k]);

}

// Loop through each character of the reversed infix expression

while (infix[i] != '\0') {

symbol = infix[i];

// If the symbol is an operand, add it to the prefix expression

if (isOperand(symbol)) {

prefix[j++] = symbol;

}

// If the symbol is ')', push it onto the stack (reverse logic for '(')

else if (symbol == ')') {

s1.push(symbol);

}

// If the symbol is an operator, handle precedence

else if (symbol == '+' || symbol == '-' || symbol == '\*' || symbol == '/' || symbol == '$')

{

if (s1.empty()) { // If stack is empty, push the operator onto the stack

s1.push(symbol);

} else {

flag = 0;

stackSymbol = s1.pop(); // Pop the top of the stack

// Compare precedence and add to prefix as needed

while (precedence(stackSymbol) > precedence(symbol)) {

prefix[j++] = stackSymbol;

if (s1.empty()) {

flag = 1; // Stack is empty, break the loop

break;

}

stackSymbol = s1.pop(); // Pop next element for comparison

}

// Push the previously popped operator back onto the stack

if (flag == 0) {

s1.push(stackSymbol);

}

// Finally, push the current operator onto the stack

s1.push(symbol);

}

}

// If the symbol is '(', pop and append everything until ')' (reverse logic for ')')

else if (symbol == '(') {

stackSymbol = s1.pop();

while (stackSymbol != ')') {

prefix[j++] = stackSymbol;

stackSymbol = s1.pop(); // Continue popping until ')' is found

}

}

i++;

}

// Pop remaining operators from the stack and append to prefix expression

while (!s1.empty()) {

prefix[j++] = s1.pop();

}

prefix[j] = '\0'; // Null-terminate the prefix expression

// Reverse the prefix expression to get the final result

for (int k = 0; k < j / 2; k++) {

swap(prefix[k], prefix[j - 1 - k]);

}

}

int main() {

char infix[80], postfix[80], prefix[80]; // Arrays to hold the infix, postfix, and prefix

expressions

cout << "Enter a valid infix expression: ";

cin >> infix; // Input the infix expression from the user

// Convert the infix expression to postfix

convertToPostfix(infix, postfix);

cout << "The converted postfix expression is: " << postfix << endl;

// Convert the infix expression to prefix

convertToPrefix(infix, prefix);

cout << "The converted prefix expression is: " << prefix << endl;

return 0;

}

**OUTPUT:**

****

**Practical 8:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem Statement Implement Circular Queue without using count and perform following operation : a) Insert b)Delete c)Display \*/

**INPUT:**

#include<iostream>

using namespace std;

const int size=10;

class cqueue{

//int size=5;

int arr[size];

int f,r;

public:

cqueue()

{

r=-1;

f=-1;

}

void push(int x);

void pop();

void display();

};

void cqueue::push(int x)

{

if ((r+ 1) % size == f)

{

cout << "Queue overflow " << x<< endl;

return;

}

if (f == -1)

{

f = 0;

}

r= (r + 1) % size;

arr[r] = x;

cout <<x<< " inserted into the queue." << endl;

}

void cqueue::pop()

{

if (f == -1)

{

cout << "Queue is empty. Cannot delete." << endl;

return;

}

int deletedValue = arr[f];

cout << deletedValue << " deleted from the queue." << endl;

if (f == r)

{

f=-1;

r=-1;

}

else

{

f= (f + 1) % size;

}

}

void cqueue::display()

{

if (f == -1)

{

cout << "Queue is empty." << endl;

return;

}

cout << "Queue elements: ";

int i = f;

while (true)

{

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

if (i == r) break;

i = (i + 1) % size;

}

}

int main()

{

int t,num,num1,choice;

cqueue q1;

do {

cout << "\nMenu:\n";

cout << "1. Insert\n";

cout << "2. Delete\n";

cout << "3. Display\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout<<"how many element need's to be inserted:";

cin>>num;

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

{

cout<<"enter data:";

cin>>t;

q1.push(t);

}

break;

case 2:

cout<<"how many elements in queue to be popped:";

cin>>num1;

for(int i=0;i<num1;i++)

{

q1.pop();

}

break;

case 3:

q1.display();

break;

case 4:

cout << "Exiting the program.\n";

break;

default:

cout << "Invalid choice. Please try again.\n";

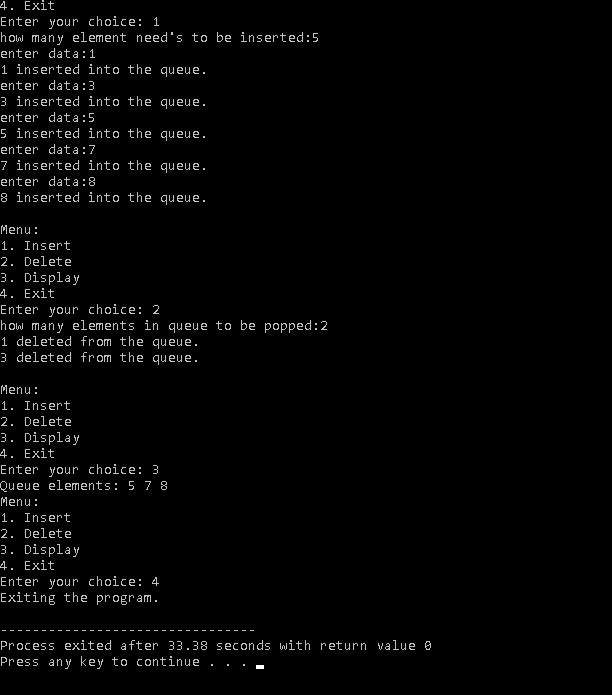
}

} while (choice != 4);

return 0;

}

**OUTPUT:**

****

**Practical 9:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem statement: Implement Circular Queue using with count and perform following operations: a)Insert b)delete c)Display

CODE:

#include<iostream>

using namespace std;

const int siz=100;

class CQueue{

    int a[siz];

    int r,f,count;

    public:

    CQueue(){r=-1;f=0;count=0;}

    void insert(int x);

    int del();

    void Disp();

};

void CQueue::insert(int x){

    if(count==siz){

        cout<<"Queue is Full!";

        return;

    }

        r = (r + 1) % siz;

        a[r]=x;

        count++;

}

int CQueue::del(){

    if(count==0){

        cout<<"Queue is Empty";

        return -1;

    }

        int y=a[f];

        f = (f + 1) % siz;

        count--;

        return(y);

}

void CQueue::Disp(){

    for(int i=0;i<count;i++){

        cout<<a[i]<<" ";

    }

    cout<<endl;

}

int main(){

    CQueue c1;

    int choice,x;

     while(true){

    cout<<"1.Insert\n 2.Delete\n 3.Display\n 4.Exit\n";

    cout<<"Enter your choice:";

    cin>>choice;

    switch(choice){

        case 1:

            cout<<"Enter element to insert:";

            cin>>x;

            c1.insert(x);

            break;

        case 2:

            c1.del();

            break;

        case 3:

            c1.Disp();

            break;

        case 4:

                cout << "Exiting..." << endl;

                return 0;

        default:

            cout<<"Invalid Input!!";

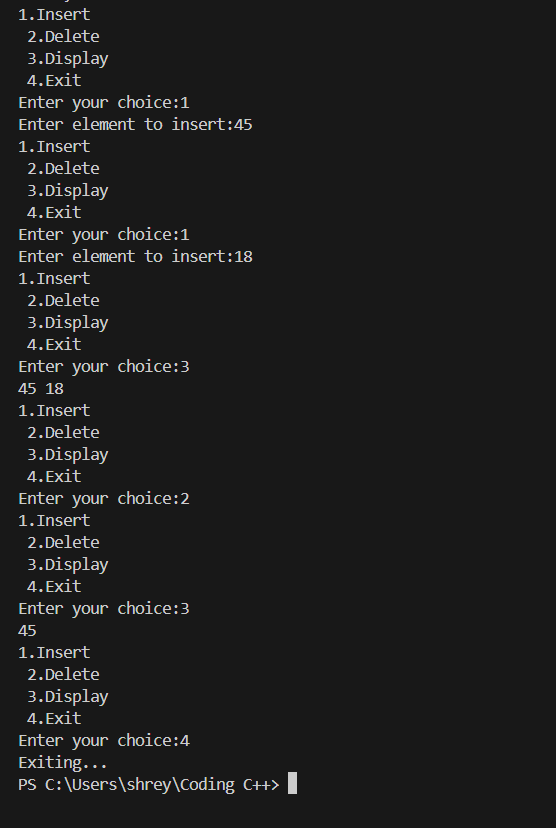
            break;

    }

    }

    return 0;

}

OUTPUT: 

**Practical 10:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem statement Create a BST and perform following operations : a)Insert a Node b)Delete a specified Node c)All traversals

Code:

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

};

Node\* createNode(int value) {

Node\* newNode = new Node();

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

Node\* insert(Node\* root, int value) {

if (root == NULL) {

return createNode(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

} else if (value > root->data) {

root->right = insert(root->right, value);

}

return root;

}

Node\* findMin(Node\* root) {

while (root && root->left != NULL) {

root = root->left;

}

return root;

}

Node\* deleteNode(Node\* root, int value) {

if (root == NULL) {

return root;

}

if (value < root->data) {

root->left = deleteNode(root->left, value);

} else if (value > root->data) {

root->right = deleteNode(root->right, value);

} else {

if (root->left == NULL) {

Node\* temp = root->right;

delete root;

return temp;

} else if (root->right == NULL) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

void preorderTraversal(Node\* root) {

if (root == NULL) {

return;

}

cout << root->data << " ";

preorderTraversal(root->left);

preorderTraversal(root->right);

}

void inorderTraversal(Node\* root) {

if (root == NULL) {

return;

}

inorderTraversal(root->left);

cout << root->data << " ";

inorderTraversal(root->right);

}

void postorderTraversal(Node\* root) {

if (root == NULL) {

return;

}

postorderTraversal(root->left);

postorderTraversal(root->right);

cout << root->data << " ";

}

int main() {

Node\* root = NULL;

int choice, value;

do {

cout << "\nMenu:\n";

cout << "1. Insert a Node\n";

cout << "2. Delete a Specified Node\n";

cout << "3. Preorder Traversal\n";

cout << "4. Inorder Traversal\n";

cout << "5. Postorder Traversal\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert: ";

cin >> value;

root = insert(root, value);

break;

case 2:

cout << "Enter value to delete: ";

cin >> value;

root = deleteNode(root, value);

break;

case 3:

cout << "Preorder Traversal: ";

preorderTraversal(root);

cout << endl;

break;

case 4:

cout << "Inorder Traversal: ";

inorderTraversal(root);

cout << endl;

break;

case 5:

cout << "Postorder Traversal: ";

postorderTraversal(root);

cout << endl;

break;

case 6:

cout << "Exiting the program.\n";

break;

default:

cout << "Invalid choice. Please try again.\n";

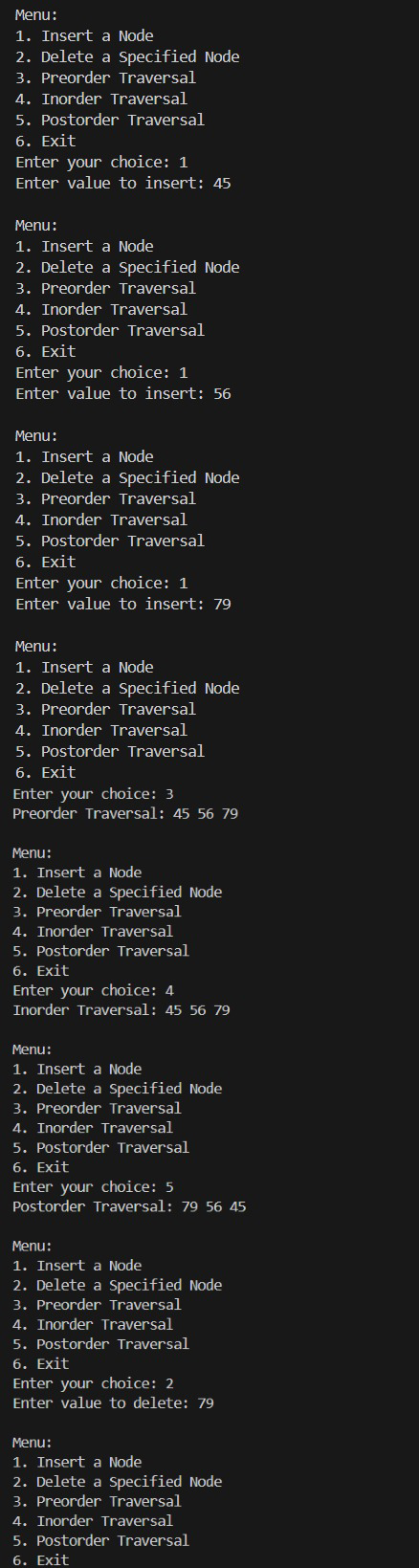
}

} while (choice != 6);

return 0;

}

OUTPUT:



**Practical 11:**

**CODE:**

/\*Class - SY B

Roll No.: B-55

Name: Vinay Shinde

Problem statement Create a BST and Implement non-recursive pre-order , post-order & In- order traversals.

INPUT:

#include <iostream>

using namespace std;

const int siz = 100;

struct node {

    int data;

    node\* left;

    node\* right;

};

class BST {

    node\* root;

public:

    BST(){root=NULL;}

    void create\_BST();

    void nr\_pre() { nr\_pre(root); }

    void nr\_inorder(){nr\_inorder(root);}

    void nr\_post(){nr\_post(root);}

private:

    void nr\_pre(node\* t);

    void nr\_inorder(node\* t);

    void nr\_post(node\* t);

};

class stack {

    node\* a[siz];

    int top;

public:

    stack(){top=-1;}

    bool empty() {

        return top == -1;

    }

    void push(node\* t){

        if(top!=siz-1)

            a[++top]=t;

    }

    node\* pop() {

    if (top != -1)

        return a[top--];

     else

        return NULL;

}

};

void BST::nr\_pre(node\* t) {

    stack s1;

    while (t != NULL) {

        cout << t->data << " ";

        s1.push(t);

        t = t->left;

    }

    while (!s1.empty()) {

        t = s1.pop();

        t = t->right;

        while (t != NULL) {

            cout << t->data << " ";

            s1.push(t);

            t = t->left;

        }

    }

}

void BST::nr\_inorder(node\* t) {

    stack s1;

    while (t != NULL) {

        s1.push(t);

        t = t->left;

    }

    while (!s1.empty()) {

        t = s1.pop();

        cout<<t->data<<" ";

        t = t->right;

        while (t != NULL) {

            s1.push(t);

            t = t->left;

        }

    }

}

void BST::nr\_post(node\* t) {

    stack s1, s2;

    if (t == NULL) return;

    s1.push(t);

    while (!s1.empty()) {

        t = s1.pop();

        s2.push(t);

        if (t->left != NULL) s1.push(t->left);

        if (t->right != NULL) s1.push(t->right);

    }

    while (!s2.empty()) {

        t = s2.pop();

        cout << t->data << " ";

    }

}

void BST::create\_BST() {

    int n, i;

    cout << "Enter No. of nodes: ";

    cin >> n;

    for (i = 0; i < n; i++) {

        node\* nn = new node;

        if (!nn) {

            cout << "Memory full" << endl;

            exit(1);

        }

        cout << "Enter data: ";

        cin >> nn->data;

        nn->left = nn->right = NULL;

        if (root == NULL) {

            root = nn;

            continue;

        }

        node\* t = root;

        node\* parent = NULL;

        while (t != NULL) {

            parent = t;

            if (nn->data < t->data) {

                t = t->left;

            } else {

                t = t->right;

            }

        }

        if (nn->data < parent->data) {

            parent->left = nn;

        } else {

            parent->right = nn;

        }

    }

}

int main() {

    BST b1;

    b1.create\_BST();

    cout << "Pre-order Traversal: ";

    b1.nr\_pre();

    cout << endl;

    cout << "In-order Traversal: ";

    b1.nr\_inorder();

    cout << endl;

    cout << "Post-order Traversal: ";

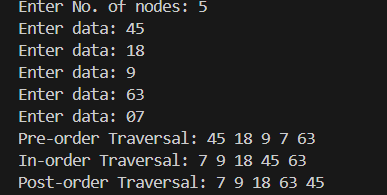
    b1.nr\_post();

    cout<<endl;

    return 0;

}

OUTPUT:



Practical 12