ASSIGNEMNTS AND OUTPUTS

ANKIT SHAH

231001102170

BCA1E

DSA

1. This program is to demonstrate the operation performed on stack & its implementation using array

#include <iostream>

#define MAX 5

using std::cout;

using std::endl;

class Stack {

int top;

int arr[MAX];

public:

Stack() {

top = -1;

}

void push(int data) {

if(top == MAX - 1) {

cout << "Stack is full" << endl;

return;

}

top++;

arr[top] = data;

}

void pop() {

if(top == -1) {

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

return;

}

top--;

}

void traverse() {

if (top == -1) {

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

return;

}

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

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

}

cout << endl;

}

};

int main() {

Stack s;

s.push(1);

s.push(2);

s.push(3);

s.push(4);

s.traverse();

s.pop();

s.pop();

s.traverse();

}

Output-

4 3 2 1

2 1

1. A saddle point of a matrix is an element which is both the largest element in its column and the smallest element in its row. A m x n matrix is said to have a saddle point if some entry a[i][j] is the smallest value in row i and the largest value in column j. Write a program that determines the saddle point if one exists (print location and value in array).

#include <iostream>

using namespace std;

const int MAX\_SIZE = 100;

void findSaddlePoint(int matrix[MAX\_SIZE][MAX\_SIZE], int m, int n) {

bool found = false;

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

int saddleValue = matrix[i][0];

int saddleColumn = 0;

// Find the smallest element in the row

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

if (matrix[i][j] < saddleValue) {

saddleValue = matrix[i][j];

saddleColumn = j;

}

}

// Check if the smallest element is also the largest in its column

bool isSaddlePoint = true;

for (int k = 0; k < m; ++k) {

if (matrix[k][saddleColumn] > saddleValue) {

isSaddlePoint = false;

break;

}

}

// If saddle point is found, print its location and value

if (isSaddlePoint) {

cout << "Saddle point found at position (" << i << ", " << saddleColumn << "): " << saddleValue << endl;

found = true;

}

}

if (!found) {

cout << "No saddle point found in the matrix." << endl;

}

}

int main() {

int m, n;

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

cin >> m >> n;

int matrix[MAX\_SIZE][MAX\_SIZE];

cout << "Enter elements of matrix: " << endl;

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

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

cin >> matrix[i][j];

}

}

findSaddlePoint(matrix, m, n);

return 0;

}

OUTPUT-

Enter number of rows and columns of matrix: 3

3

Enter elements of matrix:

1

2

3

4

5

6

7

8

9

Saddle point found at position (2, 0): 7

1. Write a program to convert the sentence in perfect form where each word will be separated by a single blank

#include <iostream>

#include <string>

using namespace std;

int main() {

string inputSentence, perfectSentence;

// Input the sentence with random number of blanks

cout << "Enter the sentence with words separated by random number of blanks: ";

getline(cin, inputSentence);

bool isPrevSpace = false;

for(char c : inputSentence) {

if (c == ' ') {

if (!isPrevSpace) {

perfectSentence += ' ';

isPrevSpace = true;

}

} else {

perfectSentence += c;

isPrevSpace = false;

}

}

cout << "Perfect form sentence: " << perfectSentence << endl;

return 0;

}

Enter the sentence with words separated by random number of blanks: a n k i t

Perfect form sentence: a n k i t

1. Write a program that accepts an amount in figures and prints that in words –

#include <iostream>

#include <string>

using namespace std;

// Function to convert single-digit number to word

string digitToWord(int digit) {

switch (digit) {

case 0: return "Zero";

case 1: return "One";

case 2: return "Two";

case 3: return "Three";

case 4: return "Four";

case 5: return "Five";

case 6: return "Six";

case 7: return "Seven";

case 8: return "Eight";

case 9: return "Nine";

default: return "";

}

}

// Function to convert two-digit number to word

string twoDigitToWord(int num) {

if (num < 10) {

return digitToWord(num);

} else if (num < 20) {

switch (num) {

case 10: return "Ten";

case 11: return "Eleven";

case 12: return "Twelve";

case 13: return "Thirteen";

case 14: return "Fourteen";

case 15: return "Fifteen";

case 16: return "Sixteen";

case 17: return "Seventeen";

case 18: return "Eighteen";

case 19: return "Nineteen";

default: return "";

}

} else {

int tens = num / 10;

int ones = num % 10;

string result = "";

switch (tens) {

case 2: result += "Twenty"; break;

case 3: result += "Thirty"; break;

case 4: result += "Forty"; break;

case 5: result += "Fifty"; break;

case 6: result += "Sixty"; break;

case 7: result += "Seventy"; break;

case 8: result += "Eighty"; break;

case 9: result += "Ninety"; break;

}

if (ones != 0) {

result += " " + digitToWord(ones);

}

return result;

}

}

// Function to convert three-digit number to word

string threeDigitToWord(int num) {

int hundreds = num / 100;

int remainder = num % 100;

string result = "";

if (hundreds != 0) {

result += digitToWord(hundreds) + " Hundred";

}

if (remainder != 0) {

if (hundreds != 0) {

result += " ";

}

result += twoDigitToWord(remainder);

}

return result;

}

// Function to convert amount to word

string amountToWords(int amount) {

if (amount == 0) {

return "Zero";

}

string words = "";

int crore = amount / 10000000;

int lakh = (amount % 10000000) / 100000;

int thousand = (amount % 100000) / 1000;

int hundred = amount % 1000;

if (crore != 0) {

words += threeDigitToWord(crore) + " Crore ";

}

if (lakh != 0) {

words += threeDigitToWord(lakh) + " Lakh ";

}

if (thousand != 0) {

words += threeDigitToWord(thousand) + " Thousand ";

}

if (hundred != 0) {

words += threeDigitToWord(hundred);

}

return words;

}

int main() {

int amount;

cout << "Enter the amount in figures: ";

cin >> amount;

cout << "Amount in words: " << amountToWords(amount) << endl;

return 0;

}

Enter the amount in figures: 436

Amount in words: Four Hundred Thirty Six

1. Quicksort is a Divide and Conquer algorithm. Write a program to sort an array using quick sort algorithm.

#include<iostream.h>

using namespace std;

int partition(int arr[],int s, int e)

{

int pivot =arr[e];

int pindex=s;

for(int i=s;i<e;i++)

{

if(arr[i]<pivot)

{

int temp=arr[i];

arr[i]=arr[pindex];

arr[pindex]=temp;

pindex++;

}

}

int temp=arr[e];

arr[e]=arr[pindex];

arr[pindex]=temp;

return pindex;

}

void quicksort(int arr, int s,int e)

{

if(s<e)

{

int p=partition(arr,s,e);

quicksort(arr,s,(p-1));

quicksort(arr,(p+1),e);

}

}

int main()

{

int size=0;

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

cin>>size;

int myarray[size];

cout<<"enter "<<size<<"in any order"<<endl;

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

{

cin>>myarray[i];

}

cout<<"Before sorting"<<endl;

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

{

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

}

cout<<endl;

quicksort(myarray,a,(size-1));

cout<<"After sorting"<<endl;

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

{

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

}

return 0;

}

1. Merge sort divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. Merge sort is also a sorting technique based on divide and conquer technique. Write a program to sort an array using merge sort technique

#include <iostream>

using namespace std;

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

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

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

for (j = 0; j < n2; j++)

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

i = 0;

j = 0;

k = l;

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 l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main() {

int n;

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

cin >> n;

int arr[n];

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

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

cin >> arr[i];

}

mergeSort(arr, 0, n - 1);

cout << "The sorted array is: ";

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

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

}

return 0;

}

Enter the size of the array: 6

Enter the elements of the array: 11

22

55

66

88

99

The sorted array is: 11 22 55 66 88 99

1. Write a program to search an element in an array using binary search technique.

#include<iostream>

using namespace std;

int main()

{

int i, arr[10], num, first, last, middle;

cout<<"Enter 10 Elements (in ascending order): ";

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

cin>>arr[i];

cout<<"\nEnter Element to be Search: ";

cin>>num;

first = 0;

last = 9;

middle = (first+last)/2;

while(first <= last)

{

if(arr[middle]<num)

first = middle+1;

else if(arr[middle]==num)

{

cout<<"\nThe number, "<<num<<" found at Position "<<middle+1;

break;

}

else

last = middle-1;

middle = (first+last)/2;

}

if(first>last)

cout<<"\nThe number, "<<num<<" is not found in given Array";

cout<<endl;

return 0;

}

Enter 10 Elements (in ascending order): 2

4

6

8

10

14

19

25

30

31

Enter Element to be Search: 55

The number, 55 is not found in given Array

1. Write a program/algorithm to evaluate postfix expression.

#include<stdio.h>

#include<ctype.h>

# define size 100

int stack[size],top=-1;

void push(int e);

int pop();

void main()

{

char pe[30];

int i,v,op1,op2;

printf("Enter postfix operation");

gets(pe);

for(i=0;pe[i]!='\0';i++)

{

if(isalpha(pe[i]))

{

printf("\n enter the value for %c",pe[i]);

scanf("%d",&v);

push(v);

}

else if(isdigit(pe[i]))

push(p[i]-'0');

else

{

op2=pop();

op1=pop();

switch(pe[i])

{

case '+':push(op1+op2);

break;

case '-':push(op1-op2);

break;

case '\*':push(op1\*op2);

break;

case '/':push(op1/op2);

break;

case '%':push(op1%op2);

break;

case '^':push(op1^op2);

break;

default:printf("\n Invalid operation");

}

}

}

printf("The result is %d",stack[top]);

}

void push(int e)

{

if(top==size-1)

printf("\n Stck full ");

else

stack[++top]=e;

}

int pop()

{

if(top==-1)

printf("\n stack is empty ");

else

return stack[top--];

}

1. Write a C program to find sum of the two diagonals of a square matrix (m x n).

#include <stdio.h>

#define MAX\_SIZE 100

int main() {

int matrix[MAX\_SIZE][MAX\_SIZE];

int m, n;

// Input the size of the square matrix

printf("Enter the size of the square matrix (m x n): ");

scanf("%d %d", &m, &n);

if (m != n) {

printf("Error: Input matrix is not square.\n");

return 1;

}

// Input the elements of the matrix

printf("Enter the elements of the matrix:\n");

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

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

scanf("%d", &matrix[i][j]);

}

}

// Calculate the sum of the two diagonals

int sumMainDiagonal = 0, sumSecondaryDiagonal = 0;

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

sumMainDiagonal += matrix[i][i]; // Elements of the main diagonal have equal row and column indices

sumSecondaryDiagonal += matrix[i][n - i - 1]; // Elements of the secondary diagonal have row index i and column index n - i - 1

}

// Output the sums of the diagonals

printf("Sum of the main diagonal: %d\n", sumMainDiagonal);

printf("Sum of the secondary diagonal: %d\n", sumSecondaryDiagonal);

return 0;

}

Enter the size of the square matrix (m x n): 3

3

Enter the elements of the matrix:

1

2

3

4

5

6

7

8

9

Sum of the main diagonal: 15

Sum of the secondary diagonal: 15

10. Write a program to delete an element from an array of integers where position of the element is the input of the program.

#include <iostream>

using namespace std;

const int MAX\_SIZE = 100;

int main() {

int arr[MAX\_SIZE];

int n, position;

// Input the size of the array

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

cin >> n;

// Input the elements of the array

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

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

cin >> arr[i];

}

// Input the position of the element to be deleted

cout << "Enter the position of the element to delete (0-indexed): ";

cin >> position;

if (position < 0 || position >= n) {

cout << "Invalid position. Deletion failed." << endl;

} else {

// Shift elements to the left starting from the position to be deleted

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

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

}

// Decrease the size of the array

n--;

cout << "Element at position " << position << " deleted successfully." << endl;

// Print the updated array

cout << "Updated array:" << endl;

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

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

}

cout << endl;

}

return 0;

}

Enter the size of the array: 5

Enter the elements of the array:

10

20

50

60

90

Enter the position of the element to delete (0-indexed): 3

Element at position 3 deleted successfully.

Updated array:

1. 20 50 90

11 . Write a program/algorithm to sort an array of integers using insertion sort technique.

#include<iostream>

using namespace std;

void display(int \*array, int size) {

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

cout << array[i] << " ";

cout << endl;

}

void insertionSort(int \*array, int size) {

int key, j;

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

key = array[i];//take value

j = i;

while(j > 0 && array[j-1]>key) {

array[j] = array[j-1];

j--;

}

array[j] = key; //insert in right place

}

}

int main() {

int n;

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

cin >> n;

int arr[n]; //create an array with given number of elements

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

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

display(arr, n);

insertionSort(arr, n);

cout << "Array after Sorting: ";

display(arr, n);

}

Enter the number of elements: 4

Enter elements:

1

36

25

55

Array before Sorting: 1 36 25 55

Array after Sorting: 1 25 36 55

12. Write a program/algorithm to represent a Stack using an array.

#include <iostream>

using namespace std;

int stack[100], n=100, top=-1;

void push(int val) {

if(top>=n-1)

cout<<"Stack Overflow"<<endl;

else {

top++;

stack[top]=val;

}

}

void pop() {

if(top<=-1)

cout<<"Stack Underflow"<<endl;

else {

cout<<"The popped element is "<< stack[top] <<endl;

top--;

}

}

void display() {

if(top>=0) {

cout<<"Stack elements are:";

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

cout<<stack[i]<<" ";

cout<<endl;

} else

cout<<"Stack is empty";

}

int main() {

int ch, val;

cout<<"1) Push in stack"<<endl;

cout<<"2) Pop from stack"<<endl;

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

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

do {

cout<<"Enter choice: "<<endl;

cin>>ch;

switch(ch) {

case 1: {

cout<<"Enter value to be pushed:"<<endl;

cin>>val;

push(val);

break;

}

case 2: {

pop();

break;

}

case 3: {

display();

break;

}

case 4: {

cout<<"Exit"<<endl;

break;

}

default: {

cout<<"Invalid Choice"<<endl;

}

}

}while(ch!=4);

return 0;

}

1) Push in stack

2) Pop from stack

3) Display stack

4) Exit

Enter choice:

1

Enter value to be pushed:

5

Enter choice:

2

The popped element is 5

Enter choice:

3

Stack is empty

Enter choice:

4

Exit

13.

Write down the following functions/algorithm for a linear linked list and double linked list.

a) Insert a node at the beginning of the list

b) Delete a node at the end of the list

c) Append a node at the end of the list

d) Insert a node at any middle position of the list

e) Delete a node from any middle position of the list

f) Count number of nodes present in the list and find the sum of all nodes

#include <iostream>

using namespace std;

// Node structure for linear linked list

struct Node {

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

// Node structure for double linked list

struct DNode {

int data;

DNode\* prev;

DNode\* next;

DNode(int value) : data(value), prev(nullptr), next(nullptr) {}

};

// Class for linear linked list operations

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

// Function to insert a node at the beginning of the list

void insertAtBeginning(int value) {

Node\* newNode = new Node(value);

newNode->next = head;

head = newNode;

}

// Function to delete a node at the end of the list

void deleteAtEnd() {

if (head == nullptr) {

cout << "List is empty. Deletion failed." << endl;

return;

}

if (head->next == nullptr) {

delete head;

head = nullptr;

return;

}

Node\* current = head;

while (current->next->next != nullptr) {

current = current->next;

}

delete current->next;

current->next = nullptr;

}

// Function to append a node at the end of the list

void append(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

return;

}

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

// Function to insert a node at any middle position of the list

void insertAtMiddle(int position, int value) {

Node\* newNode = new Node(value);

if (position == 0) {

newNode->next = head;

head = newNode;

return;

}

Node\* current = head;

for (int i = 0; i < position - 1 && current != nullptr; ++i) {

current = current->next;

}

if (current == nullptr) {

cout << "Invalid position. Insertion failed." << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

// Function to delete a node from any middle position of the list

void deleteAtMiddle(int position) {

if (position == 0) {

Node\* temp = head;

head = head->next;

delete temp;

return;

}

Node\* current = head;

Node\* prev = nullptr;

for (int i = 0; i < position && current != nullptr; ++i) {

prev = current;

current = current->next;

}

if (current == nullptr) {

cout << "Invalid position. Deletion failed." << endl;

return;

}

prev->next = current->next;

delete current;

}

// Function to count number of nodes present in the list

int countNodes() {

int count = 0;

Node\* current = head;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

// Function to find the sum of all nodes in the list

int sumOfNodes() {

int sum = 0;

Node\* current = head;

while (current != nullptr) {

sum += current->data;

current = current->next;

}

return sum;

}

};

// Class for double linked list operations

class DoubleLinkedList {

private:

DNode\* head;

DNode\* tail;

public:

DoubleLinkedList() : head(nullptr), tail(nullptr) {}

// Function to insert a node at the beginning of the list

void insertAtBeginning(int value) {

DNode\* newNode = new DNode(value);

if (head == nullptr) {

head = tail = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

// Function to delete a node at the end of the list

void deleteAtEnd() {

if (head == nullptr) {

cout << "List is empty. Deletion failed." << endl;

return;

}

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

DNode\* temp = tail;

tail = tail->prev;

tail->next = nullptr;

delete temp;

}

// Function to append a node at the end of the list

void append(int value) {

DNode\* newNode = new DNode(value);

if (head == nullptr) {

head = tail = newNode;

} else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

// Function to insert a node at any middle position of the list

void insertAtMiddle(int position, int value) {

DNode\* newNode = new DNode(value);

if (position == 0) {

insertAtBeginning(value);

return;

}

DNode\* current = head;

for (int i = 0; i < position - 1 && current != nullptr; ++i) {

current = current->next;

}

if (current == nullptr) {

cout << "Invalid position. Insertion failed." << endl;

return;

}

newNode->next = current->next;

newNode->prev = current;

if (current->next != nullptr) {

current->next->prev = newNode;

}

current->next = newNode;

}

// Function to delete a node from any middle position of the list

void deleteAtMiddle(int position) {

if (position == 0) {

if (head == nullptr) {

cout << "List is empty. Deletion failed." << endl;

return;

}

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

DNode\* temp = head;

head = head->next;

head->prev = nullptr;

delete temp;

return;

}

DNode\* current = head;

for (int i = 0; i < position && current != nullptr; ++i) {

current = current->next;

}

if (current == nullptr) {

cout << "Invalid position. Deletion failed." << endl;

return;

}

current->prev->next = current->next;

if (current->next != nullptr) {

current->next->prev = current->prev;

}

delete current;

}

// Function to count number of nodes present in the list

int countNodes() {

int count = 0;

DNode\* current = head;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

// Function to find the sum of all nodes in the list

int sumOfNodes() {

int sum = 0;

DNode\* current = head;

while (current != nullptr) {

sum += current->data;

current = current->next;

}

return sum;

}

};

int main() {

LinkedList lList;

DoubleLinkedList dList;

// Operations on linear linked list

lList.insertAtBeginning(10);

lList.insertAtBeginning(20);

lList.append(30);

lList.insertAtMiddle(2, 25);

lList.deleteAtMiddle(1);

cout << "Number of nodes in linear linked list: " << lList.countNodes() << endl;

cout << "Sum of nodes in linear linked list: " << lList.sumOfNodes() << endl;

// Operations on double linked list

dList.insertAtBeginning(10);

dList.insertAtBeginning(20);

dList.append(30);

dList.insertAtMiddle(2, 25);

dList.deleteAtMiddle(1);

cout << "Number of nodes in double linked list: " << dList.countNodes() << endl;

cout << "Sum of nodes in double linked list: " << dList.sumOfNodes() << endl;

return 0;

}

Number of nodes in linear linked list: 3

Sum of nodes in linear linked list: 75

Number of nodes in double linked list: 3

Sum of nodes in double linked list: 75

14.

Write a program to create a circular linked list.

#include <iostream>

using namespace std;

// Node structure for circular linked list

struct Node {

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

class CircularLinkedList {

private:

Node\* head;

public:

CircularLinkedList() : head(nullptr) {}

// Function to insert a node at the beginning of the list

void insertAtBeginning(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

newNode->next = newNode; // Point to itself in circular list

head = newNode;

} else {

Node\* last = head;

while (last->next != head) {

last = last->next;

}

last->next = newNode;

newNode->next = head;

head = newNode;

}

}

// Function to append a node at the end of the list

void append(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

newNode->next = newNode; // Point to itself in circular list

head = newNode;

} else {

Node\* last = head;

while (last->next != head) {

last = last->next;

}

last->next = newNode;

newNode->next = head;

}

}

// Function to display the circular linked list

void display() {

if (head == nullptr) {

cout << "Circular linked list is empty" << endl;

return;

}

Node\* current = head;

do {

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

current = current->next;

} while (current != head);

cout << endl;

}

};

int main() {

CircularLinkedList cList;

// Inserting elements into circular linked list

cList.insertAtBeginning(10);

cList.append(20);

cList.append(30);

cList.insertAtBeginning(5);

// Displaying the circular linked list

cout << "Circular linked list: ";

cList.display();

return 0;

}

Circular linked list: 5 10 20 30

15.

Write a program to check whether a given number is triad or not.

#include <iostream>

#include <cmath>

using namespace std;

// Function to check if a number is a perfect square

bool isPerfectSquare(int n) {

int root = sqrt(n);

return root \* root == n;

}

// Function to check if a given number is a triad

bool isTriad(int n) {

// Check all possible combinations of three integers (a, b, c)

for (int a = 1; a \* a <= n; ++a) {

for (int b = a; a \* a + b \* b <= n; ++b) {

int cSquared = n - a \* a - b \* b;

if (isPerfectSquare(cSquared)) {

return true;

}

}

}

return false;

}

int main() {

int num;

cout << "Enter a number to check if it is a triad: ";

cin >> num;

if (isTriad(num)) {

cout << num << " is a triad." << endl;

} else {

cout << num << " is not a triad." << endl;

}

return 0;

}.

Enter a number to check if it is a triad: 25

25 is a triad.

16.

Write a program to find the sum of the upper/lower triangle elements of a giver matrix.

#include <iostream>

using namespace std;

const int MAX\_SIZE = 10;

int main() {

int matrix[MAX\_SIZE][MAX\_SIZE];

int rows, cols;

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

cin >> rows >> cols;

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

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

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

cin >> matrix[i][j];

}

}

char choice;

cout << "Enter 'U' to find the sum of upper triangle elements or 'L' for lower triangle: ";

cin >> choice;

int sum = 0;

if (choice == 'U' || choice == 'u') {

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

for (int j = i; j < cols; ++j) {

sum += matrix[i][j];

}

}

} else if (choice == 'L' || choice == 'l') {

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

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

sum += matrix[i][j];

}

}

} else {

cout << "Invalid choice!";

return 1;

}

cout << "The sum of ";

if (choice == 'U' || choice == 'u') {

cout << "upper triangle elements is: ";

} else {

cout << "lower triangle elements is: ";

}

cout << sum << endl;

return 0;

}

Enter the number of rows and columns of the matrix: 3 3

Enter the elements of the matrix:

1 2 3

4 5 6

7 8 9

Enter 'U' to find the sum of upper triangle elements or 'L' for lower triangle: U

The sum of upper triangle elements is: 30

17. Write a program to implement the Queue data structure.

#include <iostream>

using namespace std;

int queue[100], n = 100, front = - 1, rear = - 1;

void Insert() {

int val;

if (rear == n - 1)

cout<<"Queue Overflow"<<endl;

else {

if (front == - 1)

front = 0;

cout<<"Insert the element in queue : "<<endl;

cin>>val;

rear++;

queue[rear] = val;

}

}

void Delete() {

if (front == - 1 || front > rear) {

cout<<"Queue Underflow ";

return ;

} else {

cout<<"Element deleted from queue is : "<< queue[front] <<endl;

front++;;

}

}

void Display() {

if (front == - 1)

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

else {

cout<<"Queue elements are : ";

for (int i = front; i <= rear; i++)

cout<<queue[i]<<" ";

cout<<endl;

}

}

int main() {

int ch;

cout<<"1) Insert element to queue"<<endl;

cout<<"2) Delete element from queue"<<endl;

cout<<"3) Display all the elements of queue"<<endl;

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

do {

cout<<"Enter your choice : "<<endl;

cin>>ch;

switch (ch) {

case 1: Insert();

break;

case 2: Delete();

break;

case 3: Display();

break;

case 4: cout<<"Exit"<<endl;

break;

default: cout<<"Invalid choice"<<endl;

}

} while(ch!=4);

return 0;

}

1) Insert element to queue

2) Delete element from queue

3) Display all the elements of queue

4) Exit

Enter your choice : 1

Insert the element in queue : 4

Enter your choice : 1

Insert the element in queue : 3

Enter your choice : 1

Insert the element in queue : 5

Enter your choice : 2

Element deleted from queue is : 4

Enter your choice : 3

Queue elements are : 3 5

Enter your choice : 7

Invalid choice

Enter your choice : 4

Exit