**DSA**

**First Program:**

#include <iostream>

using namespace std;

const int Size=10;

class myArray

{

private:

int arr[Size];

int elements;

public:

myArray():elements(0){}

void Input(int val)

{

if(elements>=Size)

{

cout<<"Array Full";

return;

}

arr[elements++]=val;

}

///Selection Sort(AscendingOrder)

void selectionSort()

{

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

{

int Min=i;

for(int j=i+1;j<elements;j++)

{

if(arr[j]<arr[Min])

Min=j;

}

if(Min!=i)

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

}

}

///Bubble Sort(DescendingOrder)

void bubbleSort()

{

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

{

for(int j=i+1;j<elements;j++)

{

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

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

}

}

}

///Binary Search

int binarySearch()

{

int found=-1,k,mid,start=0,endd=elements-1;

cout<<"Enter Searching Number"<<endl;

cin>>k;

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

{

mid=(start+endd)/2;

if(arr[mid]==k)

return mid;

else if(k<arr[mid])

endd=mid-1;

else if(k>arr[mid])

start=mid+1;

}

}

///Linear Search

int linearSearch(int val)

{

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

{

if(val==arr[i])

return i;

}

return -1;

}

void showArray()

{

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

{

cout<<"Value Found At Index "<<i<<" is "<<arr[i]<<endl;

}

}

};

int main()

{

myArray obj;

obj.Input(23);

obj.Input(76);

obj.Input(92);

obj.Input(17);

obj.Input(77);

///obj.selectionSort(); ///AscendingOrder

///obj.bubbleSort(); ///DescendingOrder

obj.showArray();

///binary Search

int ind=obj.binarySearch();

if(ind!=-1)

cout<<"Value Found At Index : "<<ind<<endl;

else

cout<<"Not Found"<<endl;

///Linear Search

int indx=obj.linearSearch(17);

if(indx!=-1)

cout<<"Found at index : "<<indx<<endl;

else

cout<<"not found";

return 0;

}

**TIME COMPLEXITIES**:

bubble & selection sort = O(n²)

Merge & Quick Sort =O(nlogn)

linear search &insertion sort = O(n)

binary search = O(log n)

**///RecursiveSelectionSort**

#include <iostream>

using namespace std;

void RecursiveSelectionSort(int arr[],int n,int i=0)

{

if(i==n-1)

return ;

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]);

RecursiveSelectionSort(arr,n,i+1);

}

int main()

{

const int n=5;

int arr[n]={32,73,62,27,84};

RecursiveSelectionSort(arr,n);

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

{

cout<<"Value at index "<<i<<" : "<<arr[i]<<endl;

}

return 0;

}

**///RecursiveBubbleSort**

#include <iostream>

using namespace std;

void RecursiveBubbleSort(int arr[],int n,int i=0)

{

if(i==n-1)

return ;

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

{

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

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

}

RecursiveBubbleSort(arr,n,i+1);

}

int main()

{

const int n=5;

int arr[n]={32,73,62,27,84};

RecursiveBubbleSort(arr,n);

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

{

cout<<"Value at index "<<i<<" : "<<arr[i]<<endl;

}

return 0;

}

**///Recursive Linear Search**

#include <iostream>

using namespace std;

int RecursiveLinearSearch(int arr[],int n,int val,int ind=0)

{

if(ind==n)

return -1;

if(arr[ind]==val)

return ind;

RecursiveLinearSearch(arr,n,val,ind+1);

}

int main()

{

const int n=5;

int arr[n]={32,73,62,27,84};

int val=73;

int result=RecursiveLinearSearch(arr,n,val);

if (result != -1)

cout << "Element found at index: " << result << endl;

else

cout << "Element not found" << endl;

return 0;

}

**///Recursive Binary Search**

#include <iostream>

using namespace std;

int RecursiveBinarySearch(int arr[],int start,int endd,int val)

{

if(start>endd)

return -1;

int mid=(start+endd)/2;

if(arr[mid]==val)

return mid;

else if(val<arr[mid])

return RecursiveBinarySearch(arr, start,mid-1,val);

else if(val>arr[mid])

return RecursiveBinarySearch(arr, mid+1,endd,val);

}

int main()

{

const int n=5;

int arr[n]={23,45,68,73,79};

int val=73;

int result=RecursiveBinarySearch(arr,0,n-1,val);

if (result != -1)

cout << "Element found at index: " << result << endl;

else

cout << "Element not found" << endl;

return 0;

}

**ARRAY OPERATIONS:**

**~***insertion ~deletion*

*~traverse ~Read*

*~Write*

///**insertion of value at kth index**(value isnt replaced,it is added by moving other values)

#include <iostream>

using namespace std;

int main()

{

int n=5;

int arr[n],ind,val,delind; //if array is full when insertion ,last value is lost

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

{

cout<<"Enter Element of array : "; cin>>arr[i];

}

///Insertion at kth index

cout<<"enter index where vlue is to be inserted : ";

cin>>ind;

cout<<"enter value to be inserted : ";

cin>>val;

for(int i=n-1;i>ind;i--)

{

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

arr[ind]=val;

}

///Deletion of index

cout<<"enter deleting index : ";

cin>>delind;

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

{

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

}

n--;

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

{

cout<<"Value at index "<<i<<" : "<<arr[i]<<endl;

}

return 0;

}

**RECURSION:**

**~Factorial:**

#include <iostream>

using namespace std;

int fact(int n)

{

if(n==0)

return 1;

else

{

int x,y;

x=n-1;

y=fact(x);

return n\*y;

}

}

int main()

{

int no;

cout<<"Enter Number : "; cin>>no;

int f=fact(no);

cout<<"Factorial is : "<<f<<endl;

return 0;

}

~**Fibonacci Series:**

#include <iostream>

using namespace std;

int fibonacci(int n)

{

if(n==0||n==1)

return n;

else

{

int x,y;

x=fibonacci(n-1);

y=fibonacci(n-2);

return x+y;

}

}

int main()

{

int no,s;

cout << "Enter number : " ; cin>>no;

s=fibonacci(no);

cout<<s;

return 0;

}

**~Sum Of Array:**

#include <iostream>

using namespace std;

const int S=10;

int sumFunc(int s,arr[S])

{

if(s==0)

return 0;

else

{

return arr[s-1]+sumFunc(S-1,arr);

}

}

int main()

{

cout << "Hello world!" << endl;

return 0;

}

~**Power (by recursion):**

#include <iostream>

using namespace std;

int powR(int n,int p)

{

if(p==0)

return 1;

else

{

int pichla;

pichla=powR(n,p-1);

return n\*pichla;

}

}

int main()

{

int n,p;

cout << "Enter Number : " ; cin>>n;

cout << "Enter Power : " ; cin>>p;

int PWR=powR(n,p);

cout<<"Answer is : "<<PWR<<endl;

return 0;

}

/\*///(without Recursion)

int power() ///power ka function

{

int n,i,p,ans=1;

cout << "Enter Number : " ; cin>>n;

cout << "Enter Power : " ; cin>>p;

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

{

ans= ans\*n;

}

return ans;

}\*/

~**Reverse String by recursion:**

#include <iostream>

using namespace std;

void reverse(char \*str)

{

if(\*str!='\0')

{

reverse(str+1);

cout<<\*str;

}

}

int main()

{

reverse("Ali");

    return 0;

}

///**EVEN COUNT BY RECURSION**

#include <iostream>

using namespace std;

int even(int arr[],int S,int ev=0)

{

if(S==0)

{

return ev;

}

if(arr[S-1]%2==0)

{

ev++;

}

return even(arr,S-1,ev);

}

int main()

{

const int S=5;

int arr[S]={2,3,4,1,6};

int e=even(arr,S);

cout<<e<<endl;

return 0;

}

**SORTING:**

**~Insertion Sort:**

/\*#include <iostream>

using namespace std;

int main()

{

int arr[5]={32,12,48,93,29};

for(int i=0;i<5;i++)

{

int j=i-1,current=arr[i];

while(arr[j]>current&&arr[j]>0)

{

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

j--;

}

arr[j+1]=current;

}

for(int i=0;i<5;i++)

{

cout<<"value at index "<<i<<" : "<<arr[i]<<endl;

}

return 0;

}\*/

*///Alphabetically Sorted*

#include <iostream>

using namespace std;

const int S=5;

int main()

{

string arr[S]={"apple","mango","kiwi","banana","grape"};

/\*string arr[S];

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

{

cout<<"Enter value At index "<<i<<" : "<<endl;

cin>>arr[i];

}\*/

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

{

string current=arr[i];

int j=i-1;

while(arr[j]>current&&j>=0)

{

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

j--;

}

arr[j+1]=current;

}

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

{

cout<<"value at "<<i<<" : "<<arr[i]<<endl;

}

return 0;

}

**~Merge Sort:**

#include<iostream>

using namespace std;

const int S=5;

void mergE(int arr[],int p,int q,int r)

{

int n1=q-p+1,n2=r-q,L[n1+1],R[n2+1],i,j;

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

{

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

}

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

{

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

}

i=0,j=0;

L[n1]=2000000000; ///INT\_MAX

R[n2]=2147483647;

for(int k=p;k<=r;k++)

{

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

{

arr[k]=L[i];

i++;

}

else

{

arr[k]=R[j];

j++;

}

}

}

void mergesort(int A[S],int strt,int endd)

{

if(strt<endd)

{

int mid=(strt+endd)/2;

mergesort(A,strt,mid);

mergesort(A,mid+1,endd);

mergE( A, strt, mid, endd);

}

}

int main()

{

int Arr[S]={12,46,22,93,36},s=0,e=S-1;

mergesort(Arr,s,e);

///cout<<mergesort(Arr,s,e)<<endl;

cout<<"MERGE SORT"<<endl;

cout<<endl;

cout<<"Sorted Array Is : " <<endl;

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

{

cout<<"Value at "<<i<<" : "<<Arr[i]<<endl;

}

return 0;

}

**~Quick Sort:**

*///element is supposed to be sorted if the elements at its left are smaller than it and those at its right are larger than it.*

#include <iostream>

using namespace std;

int Partition(int arr[],int low,int high)

{

int pivot=arr[low];

int i=low,j=high;

while(i<j) ///stop when i becomes greater than j and the j index becomes the new pivotal element

{

do

{

i++;

}while(arr[i]<=pivot); *///increment i until array element is greater than pivot(means while array element is less than pivot,when its greater from pivot then stop!)*

do

{

j--;

}while(arr[j]>pivot);

if(i<j)

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

}

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

return j;

}

void QuickSort(int a[],int l,int h)

{

if(l<h)

{

int p=Partition(a,l,h);

QuickSort(a,l,p);

QuickSort(a,p+1,h);

}

}

int main()

{

int Array[5]={35,82,13,60,53},l=0,h=5;

QuickSort(Array,l,h);

for(int i=0;i<5;i++)

{

cout<<"value at "<<i<<" : "<<Array[i]<<endl;

}

return 0;

}

***///Merged Sorting Of EVEN elements Of Two Arrays into Third array***

~Apply mergesorting first.

int even(int arr1[],int arr2[],int arr3[],int s)

{

int ind;

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

{

if(arr1[i]%2==0)

{

arr3[ind++]=arr1[i];

}

}

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

{

if(arr2[i]%2==0)

{

arr3[ind++]=arr2[i];

}

}

}

***///Single Array Even***

int even(int arr1[],int arr3[],int s1)

{

int index;

for(int i=0;i<s1;i++)

{

if(arr1[i]%2==0)

arr3[index++]=arr1[i];

}

return index;

}

***///Counting Even And Odd Numbers in Array***

#include <iostream>

using namespace std;

const int SIZE = 5;

void mergE(int arr[],int p,int q,int r)

{

int n1=q-p+1,n2=r-q,L[n1+1],R[n2+1],i,j;

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

{

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

}

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

{

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

}

i=0,j=0;

L[n1]=2000000000; ///INT\_MAX

R[n2]=2147483647;

for(int k=p;k<=r;k++)

{

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

{

arr[k]=L[i];

i++;

}

else

{

arr[k]=R[j];

j++;

}

}

}

void mergesort(int A[SIZE],int strt,int endd)

{

if(strt<endd)

{

int mid=(strt+endd)/2;

mergesort(A,strt,mid);

mergesort(A,mid+1,endd);

mergE( A, strt, mid, endd);

}

}

int evenCount(int arr1[], int arr2[], int arr3[], int s1, int s2, int &oddCount)

{

int indexs = 0; // Initialize the index to 0

oddCount = 0; // Initialize oddCount to 0

for (int i = 0; i < s1; i++)

{

if (arr1[i] % 2 == 0)

{

arr3[indexs++] = arr1[i];

}

else

{

oddCount++;

}

}

for (int i = 0; i < s2; i++)

{

if (arr2[i] % 2 == 0)

{

arr3[indexs++] = arr2[i];

}

else

{

oddCount++;

}

}

return indexs; // Return the count of even numbers added to arr3

}

int main()

{

int arr1[SIZE] = {5, 4, 3, 2, 1};

int arr2[SIZE] = {52, 43, 32, 24, 18};

int arr3[SIZE \* 2];

int oddCount = 0;

int evenCountResult = evenCount(arr1, arr2, arr3, SIZE, SIZE, oddCount);

mergesort(arr3, 0, evenCountResult - 1);

cout << "Even numbers sorted in arr3: ";

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

{

cout << arr3[i] << " ";

}

cout << endl;

cout << "Even count = " << evenCountResult << endl;

cout << "Odd count = " << oddCount << endl;

return 0;

}

**TEMPLATE:**

#include <iostream>

using namespace std;

template<class Type>

Type maX(Type a,Type b)

{

return a>b?a:b;

}

int main()

{

float m=maX(6.9,6.8);

cout<<m;

return 0;

}

**STACK:**

*///In-built functions*

#include <iostream>

#include<stack>

using namespace std;

int main()

{

stack<string>stk;

stk.push("apple");

stk.push("mango");

stk.push("banana");

stk.push("kiwi");

cout<<stk.top()<<endl;

stk.pop();

cout<<stk.top()<<endl;

cout<<endl;

while(!stk.empty())

{

cout<<stk.top()<<endl;

stk.pop();

}

return 0;

}

***///5 Functions of Stack(Using Array As Stack)***

#include<iostream>

using namespace std;

const int S=5;

class Stack

{

private:

int stck[S];

int top;

public:

Stack():top(-1){}

bool isEmpty()

{

if(top==-1)

return true;

else

return false;

}

bool isFull()

{

if(top==S-1)

return true;

else

return false;

}

void Push(int val)

{

if(isFull())

cout<<"Its Full"<<endl;

else

stck[++top]=val;

}

int Pop()

{

if(isEmpty())

{

return -1;

}

else

return stck[top--];

}

int Top()

{

if(!isEmpty())

return stck[top];

else

return -1;

}

};

int main()

{

Stack s;

s.Push(23);

s.Push(35);

s.Push(79);

cout<<s.isEmpty();

cout<<s.isFull();

cout<<s.Top();

return 0;

}***///Checking Validity Of Expression***

#include <iostream>

#include<stack>

using namespace std;

class Stack

{

public:

bool isValid(string expr)

{

bool ans=true;

stack<char>stk;

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

{

if(expr[i]=='('||expr[i]=='['||expr[i]=='{')

{

stk.push(expr[i]);

}

else if(expr[i]==')')

{

if(!stk.empty()&&stk.top()=='(')

{

stk.pop();

}

else

{

ans=false;

break;

}

}

else if(expr[i]==']')

{

if(!stk.empty()&&stk.top()=='[')

{

stk.pop();

}

else

{

ans=false;

break;

}

}

else if(expr[i]=='}')

{

if(!stk.empty()&&stk.top()=='{')

{

stk.pop();

}

else

{

ans=false;

break;

}

}

}

if(stk.empty())

return true;

else

return false;

}

};

int main()

{

Stack stck;

string expression="{[(])}";

if(stck.isValid(expression))

cout<<"Valid"<<endl;

else

cout<<"Invalid"<<endl;

return 0;

}

***///Evaluation Of PostFix Expression***

#include <iostream>

#include<stack>

#include<math.h>

using namespace std;

class Stack

{

public:

int post(string s)

{

stack<int> st;

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

{

if(s[i]>='0'&&s[i]<='9')

st.push(s[i]-'0');

else

{

int op2=st.top(); ///jo stack me top pe value dali thi(0 sy 9 k drmyan h wo)

st.pop();

int op1=st.top();

st.pop();

switch(s[i]) /// s[i]-> value found in expression in this itteration

{

case'+':

st.push(op1+op2);

break;

case'-':

st.push(op1-op2);

break;

case'\*':

st.push(op1\*op2);

break;

case'/':

st.push(op1/op2);

break;

}

return st.top();

}

}

}

};

int main()

{

Stack ss;

cout<<ss.post("23+6\*");

return 0;

}

/\* ///using while loop and char

int post(char s[])

{

int i;

stack<int> st;

while(i!='/0')

\*/

***///Conversion of Infix Notation to Postfix***

#include <iostream>

#include<stack>

#include<string>

using namespace std;

class Stack

{

public:

int pres(char c)

{

if(c=='^')

return 3;

else if(c=='/'||c=='\*')

return 2;

else if(c=='+'||c=='-')

return 1;

else

return -1;

}

string infixtopost(string expr)

{

stack<char>stk;

int i=0;

string resultstr;

while(expr[i]!='\0')

{

if(expr[i]>='0'&&expr[i]<='9')

resultstr+=expr[i];

else if(expr[i]=='(')

stk.push(expr[i]);

else if(expr[i]==')')

{

while(!stk.empty()&&stk.top()!='(')

{

resultstr+=stk.top();

stk.pop();

}

if(!stk.empty())

stk.pop();

}

else

{

while(!stk.empty()&&pres(stk.top())>=pres(expr[i]))

{

resultstr+=stk.top();

stk.pop();

}

stk.push(expr[i]);

}

i++;

}

while(!stk.empty())

{

resultstr+=stk.top();

stk.pop();

}

return resultstr;

}

};

int main()

{

Stack ss;

string ans=ss.infixtopost("(23+6)/12\*(15-3)/2");

cout<<"answer : "<<ans<<endl;

return 0;

}

**///STRING REVERSE IN STACK**

#include<iostream>

#include<stack>

#include<string>

using namespace std;

class Stack

{

public:

string Reverse(string origstr)

{

stack<char>stk1;

for(int i=0;i<origstr.length();i++)

{

stk1.push(origstr[i]);

}

string newstr="";

while(!stk1.empty())

{

newstr+=stk1.top();

stk1.pop();

}

return newstr;

}

};

int main()

{

Stack stk;

string s1="Aslam";

string s2=stk.Reverse(s1);

cout<<s2;

return 0;

}

**QUEUES:**

**///BASIC QUEUE OPERATIONS( While using Array as Queue)**

///insertion back pe....remove fronts sy

#include <iostream>

using namespace std;

const int S=3;

class Queue

{

private:

int Front,rear,q[S];

public:

Queue()

{

Front=-1;

rear=-1;

}

bool isEmpty()

{

if(Front==-1||Front>rear)

return true;

else

return false;

}

bool isFull()

{

if(rear==S-1)

return true;

else

return false;

}

void enqueue(int val)

{

if(!isFull())

q[++rear]=val;

else

cout<<"Can't insert , Queue is full !"<<endl;

if(Front==-1)

Front++;

}

int dequeue()

{

if(!isEmpty())

{

return q[Front--];

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

{

q [i] = q [i+1];

rear--;

}

}

else

{

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

return -1;

}

}

int peak()

{

if(!isEmpty())

return q[Front];

else

{

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

return -1;

}

}

};

int main()

{

Queue q;

q.enqueue(23);

q.enqueue(35);

//q.enqueue(78);

cout<<q.isFull()<<endl; ///1 for true 0 for false

cout<<q.peak()<<endl;

q.dequeue();

q.dequeue();

//q.dequeue();

cout<<q.isEmpty()<<endl;

    return 0;

}

**///Arranging even numbers from array 1 to queue 2 and odd elements in q3**

#include <iostream>

#include <queue>

using namespace std;

void Arrange(int Q1[],int S,queue<int>&q2,queue<int>&q3)

{

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

{

if(Q1[i]%2==0)

q2.push(Q1[i]);

else

q3.push(Q1[i]);

}

}

int main()

{

const int S=5;

int Q1[S]={32,61,16,83,86};

queue<int>q2;

queue<int>q3;

Arrange(Q1,S,q2,q3);

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

{

cout<<"Queue-1 is : "<<Q1[i]<<endl;

}

cout<<"Queue-2 is : "<<endl;

while(!q2.empty())

{

cout<<q2.front()<<" ";

q2.pop();

}

cout<<"\nQueue-3 is : "<<endl;

while(!q3.empty())

{

cout<<q3.front()<<" ";

q3.pop();

}

return 0;

}

***LINKED LIST***

***Singly linked list***

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

};

class singlelist

{

private:

node\* head;

public:

singlelist():head(NULL){}

bool isempty()

{

if(head == NULL)

{

return true;

}

else

{

return false;

}

}

void insertathead(int val)

{

node\* newer = new node();

newer->data = val;

newer->next = head;

head = newer;

}

void showlist()

{

node\* temp = head;

while(temp != NULL)

{

cout<<"data is : "<<temp->data<<endl;

temp = temp->next;

}

}

void insertatend(int val)

{

if(isempty())

{

insertathead(val);

}

else

{

node\* temp = head;

while(temp->next != NULL)

{

temp = temp->next;

}

node\* newer = new node();

newer->data = val;

newer->next = NULL;

temp->next = newer;

}

}

void deleteathead()

{

if(isempty())

{

cout<<"list is empty : "<<endl;

return;

}

{

node\* temp = head;

head = temp->next;

cout<<"deleting the value : "<<temp->data<<endl;

delete temp;

}

}

void deleteatend()

{

if(isempty())

{

cout<<"list is empty : "<<endl;

return;

}

if(head->next == NULL)

{

cout<<"deleting the value : "<<head->data<<endl;

delete head;

head = NULL;

cout<<"Now the list is empty : "<<endl;

return;

}

node\* temp = head;

node\* temp2 = temp->next;

while(temp2->next != NULL)

{

temp = temp->next;

temp2 = temp->next;

}

temp->next = NULL;

cout<<"deleting the value : "<<temp2->data<<endl;

delete temp2;

}

};

int main()

{

singlelist s;

s.insertathead(12);

s.insertatend(14);

s.insertathead(45);

s.showlist();

cout<<endl;

s.deleteathead();

s.showlist();

cout<<endl;

s.deleteatend();

s.showlist();

return 0;

}

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

};

class singlelist

{

private:

node\* head;

public:

singlelist():head(NULL){}

bool isempty()

{

if(head == NULL)

{

return true;

}

else

{

return false;

}

}

void insertathead(int val)

{

node\* newer = new node();

newer->data = val;

newer->next = head;

head = newer;

}

void showlist()

{

node\* temp = head;

while(temp != NULL)

{

cout<<"data is : "<<temp->data<<endl;

temp = temp->next;

}

}

void insertatend(int val)

{

if(isempty())

{

insertathead(val);

}

else

{

node\* temp = head;

while(temp->next != NULL)

{

temp = temp->next;

}

node\* newer = new node();

newer->data = val;

newer->next = NULL;

temp->next = newer;

}

}

void deleteathead()

{

if(isempty())

{

cout<<"list is empty : "<<endl;

return;

}

{

node\* temp = head;

head = temp->next;

cout<<"deleting the value : "<<temp->data<<endl;

delete temp;

}

}

void deleteatend()

{

if(isempty())

{

cout<<"list is empty : "<<endl;

return;

}

if(head->next == NULL)

{

cout<<"deleting the value : "<<head->data<<endl;

delete head;

head = NULL;

cout<<"Now the list is empty : "<<endl;

return;

}

node\* temp = head;

node\* temp2 = temp->next;

while(temp2->next != NULL)

{

temp = temp->next;

temp2 = temp->next;

}

temp->next = NULL;

cout<<"deleting the value : "<<temp2->data<<endl;

delete temp2;

}

node\* searchnode(int val)

{

node\* temp = head;

while(temp != NULL)

{

if(temp->data == val)

{

break;

}

temp = temp->next;

}

return temp;

}

void insertafter(int val)

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

node\* newer = new node();

cout<<"enter a value to insert after : "<<endl;

cin>>newer->data;

newer->next = temp->next;

temp->next = newer;

}

}

void insertbefore(int val)

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp == head)

{

int value;

cout<<"enter a value to insert before : "<<endl;

cin>>value;

insertathead(value);

}

else

{

node\* temp2 = head;

while(temp2 != NULL)

{

if(temp2->next == temp)

{

break;

}

temp2 = temp2->next;

}

node\* newer = new node();

cout<<"enter a value to insert before : "<<endl;

cin>>newer->data;

newer->next = temp;

temp2->next = newer;

}

}

}

/\*

void insertbefore(int val)

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp == head)

{

int value;

cout<<"enter a value to insert before : "<<endl;

cin>>value;

insertathead(value);

}

else

{

node\* temp2 = head;

while(temp2->next != temp)

{

temp2 = temp2->next;

}

node\* newer = new node();

cout<<"enter a value to insert before : "<<endl;

cin>>newer->data;

newer->next = temp;

temp2->next = newer;

}

}

}

\*/

void deleteparticular(int val)

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp == head)

{

deleteathead();

}

else

{

node\* temp2 = head;

while(temp2 != NULL)

{

if(temp2->next == temp)

{

break;

}

temp2 = temp2->next;

}

if(temp->next == NULL)

{

temp2->next = NULL;

cout<<"deleting the value : "<<temp->data<<endl;

delete temp;

}

else

{

temp2->next = temp->next;

cout<<"deleting the value : "<<temp->data<<endl;

delete temp;

}

}

}

}

/\*void deleteparticular(int val)

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp == head)

{

deleteathead();

}

else if(temp->next == NULL)

{

deleteatend();

}

else

{

node\* temp2 = head;

node\* temp3 = head;

while(temp2->next != temp)

{

temp2 = temp2->next;

}

while(temp3 != temp)

{

temp3 = temp3->next;

}

temp2->next = temp3->next;

cout<<"deleting the value : "<<temp->data<<endl;

delete temp;

}

}

}\*/

void deletebefore(int val)

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp == head)

{

cout<<"deletion is not possible : "<<endl;

}

else if (head->next == temp)

{

deleteathead();

}

else

{

node\* temp1 = head;

node\* temp2 = head;

while(temp2->next != temp)

{

temp1 = temp2;

temp2 = temp2->next;

}

temp1->next = temp;

cout<<"deleting the value : "<<temp2->data<<endl;

delete temp2;

}

}

}

}

void deleteafter(int val)

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp->next == NULL)

{

cout<<"deletion not possible : "<<endl;

}

else if((temp->next)->next == NULL)

{

deleteatend();

}

else

{

node\* temp3 = temp->next;

node\* temp2 = temp3->next;

cout<<"deleting the value : "<<temp3->data<<endl;

temp->next = temp2;

delete temp3;

}

}

}

}

};

int main()

{

singlelist s;

s.insertathead(12);

s.insertatend(14);

s.insertatend(45);

s.insertatend(67);

s.insertatend(57);

s.showlist();

s.deleteatend();

s.deleteatend();

s.deleteatend();

s.showlist();

return 0;

}

***Linked Queue***

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

};

class linkedqueues

{

private:

node\* front;

node\* rear;

public:

linkedqueues():front(NULL),rear(NULL){}

bool isempty()

{

if(front == NULL)

{

return true;

}

else

{

return false;

}

}

void enqueue(int val)

{

node\* newer = new node();

newer->data = val;

newer->next = NULL;

if(rear == NULL)

{

front = newer;

}

else

{

rear->next = newer;

}

rear = newer;

}

void dequeue()

{

if(isempty())

{

cout<<"not possible list is empty : "<<endl;

}

else

{

node\* temp = front;

if(front == rear)

{

cout<<"dequeuing the value : "<<front->data<<endl;

front = rear = NULL;

}

else

{

cout<<"dequeuing the value : "<<front->data<<endl;

front = front->next;

}

delete temp;

}

}

void frontelement()

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

node\* temp = front;

cout<<"value at front is : "<<temp->data<<endl;

}

}

node\* searchelement(int val)

{

node\* temp = front;

while(temp != NULL)

{

if(temp->data == val)

{

break;

}

temp = temp->next;

}

return temp;

}

void showlist()

{

node\* temp = front;

while(temp != NULL)

{

cout<<"value is : "<<temp->data<<endl;

temp = temp->next;

}

}

};

int main()

{

linkedqueues q;

q.enqueue(12);

q.enqueue(13);

q.enqueue(14);

q.enqueue(15);

q.enqueue(4);

q.enqueue(55);

q.showlist();

node\* lll = q.searchelement(4);

if(lll == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

cout<<"value is founded : "<<lll->data<<endl;

}

return 0;

}

***Linked Stack***

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

};

class linkedstack

{

private:

node\* top;

public:

linkedstack():top(NULL){}

bool isempty()

{

if(top == NULL)

{

return true;

}

else

{

return false;

}

}

void push(int val)

{

node\* newer = new node();

newer->data = val;

newer->next = top;

top = newer;

}

void pop()

{

if(isempty())

{

cout<<"lits is empty : "<<endl;

}

else

{

node\* temp = top;

if(top->next == NULL)

{

cout<<"deleting the value : "<<top->data<<endl;

top = NULL;

}

else

{

cout<<"deleting the value : "<<top->data<<endl;

top = top->next;

}

delete temp;

}

}

void topelement()

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

cout<<"top element is : "<<top->data<<endl;

}

}

node\* searchelement(int val)

{

node\* temp = top;

while(temp != NULL)

{

if(temp->data == val)

{

break;

}

temp = temp->next;

}

return temp;

}

void showlist()

{

node\* temp = top;

while(temp != NULL)

{

cout<<"value is : "<<temp->data<<endl;

temp = temp->next;

}

}

};

int main()

{

linkedstack l;

l.push(12);

l.push(13);

l.push(14);

l.push(15);

l.push(16);

l.topelement();

l.push(17);

l.topelement();

l.push(18);

l.topelement();

cout<<endl;

l.showlist();

cout<<endl;

l.pop();

l.pop();

l.pop();

l.pop();

l.pop();

cout<<endl;

l.showlist();

node\* lll = l.searchelement(12);

if(lll == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

cout<<"value is founded : "<<lll->data<<endl;

}

return 0;

}

***Simple circular list***

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

};

class Circularlist

{

private:

node\* last;

public:

Circularlist():last(NULL){}

bool isempty()

{

if(last == NULL)

{

return true;

}

else

{

return false;

}

}

void add(int val)

{

node\* newer = new node();

newer->data = val;

if(last == NULL)

{

newer->next = newer;

last = newer;

}

else

{

newer->next = last->next;

last->next = newer;

last = newer;

}

}

void pop()

{

if(isempty())

{

cout<<"pop operation is not supported : "<<endl;

}

else

{

node\* temp = last->next;

if(last == temp)

{

last = NULL;

}

else

{

last->next = temp->next;

}

cout<<"deleting the value : "<<temp->data<<endl;

delete temp;

}

}

void show()

{

node\* temp = last->next;

while(true)

{

cout<<"value is : "<<temp->data<<endl;

temp = temp->next;

if(temp == last->next)

{

break;

}

}

}

node\* searchnode(int val)

{

node\* temp = last->next;

while(true)

{

if(temp->data == val)

{

break;

}

temp = temp->next;

if(temp == last->next)

{

break;

}

}

if(isempty())

{

return NULL;

}

else

{

return temp;

}

}

};

int main()

{

Circularlist c;

c.add(111);

c.add(21);

c.add(43);

c.add(58);

c.add(11);

c.pop();

c.pop();

c.show();

node\* lll = c.searchnode(11);

if(lll == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

cout<<"value is founded : "<<lll->data<<endl;

}

return 0;

}

***Circular linked stack***

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

};

class StackCircularList

{

private:

node\* last;

public:

StackCircularList(): last(NULL) {}

bool isempty()

{

if(last == NULL)

{

return true;

}

else

{

return false;

}

}

void push(int val)

{

node\* newer = new node();

newer->data = val;

if (last == NULL)

{

newer->next = newer;

last = newer;

}

else

{

newer->next = last->next;

last->next = newer;

}

}

void pop()

{

if (isempty())

{

cout << "Stack is empty. Pop operation not supported." << endl;

return;

}

node\* temp = last->next;

if (last == temp)

{

last = NULL;

}

else

{

last->next = temp->next;

}

cout << "Popping value: " << temp->data << endl;

delete temp;

}

void topelement()

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

node\* temp = last->next;

cout<<"top element is : "<<temp->data<<endl;

}

}

void display()

{

if (isempty())

{

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

return;

}

node\* temp = last->next;

while(true)

{

cout<<"value is : "<<temp->data<<endl;

temp = temp->next;

if(temp == last->next)

{

break;

}

}

cout << endl;

}

};

int main()

{

StackCircularList scl;

scl.push(10);

scl.push(20);

scl.push(30);

scl.topelement();

scl.display();

scl.pop();

scl.pop();

scl.display();

return 0;

}

***Code for simple circular list and circular linked queue is same and can also be implemented in two ways second is by using front and rear***

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\* next;

};

class CircularQueue

{

private:

Node\* front;

Node\* rear;

public:

CircularQueue() : front(NULL), rear(NULL) {}

bool isEmpty()

{

return (front == NULL);

}

void enqueue(int value)

{

Node\* newNode = new Node();

newNode->data = value;

if (isEmpty())

{

front = rear = newNode;

newNode->next = newNode;

}

else

{

newNode->next = rear->next;

rear->next = newNode;

rear = newNode;

}

}

void dequeue()

{

if (isEmpty())

{

cout << "Queue is empty, cannot dequeue!" << endl;

}

else

{

Node\* temp = front;

if (front == rear)

{

front = rear = NULL;

}

else

{

front = front->next;

rear->next = front;

}

cout << "Dequeued value: " << temp->data << endl;

delete temp;

}

}

void display()

{

if (isEmpty())

{

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

return;

}

Node\* temp = front;

while(true)

{

cout<<"value is : "<<temp->data<<endl;

temp = temp->next;

if(temp == front)

{

break;

}

}

}

};

int main() {

CircularQueue cq;

cq.enqueue(10);

cq.enqueue(20);

cq.enqueue(30);

cq.enqueue(40);

cq.display();

cq.dequeue();

cq.display();

cout<<endl;

cq.enqueue(50);

cq.display();

return 0;

}

***Doubly link list***

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* left;

node\* right;

};

class doubly

{

private:

node\* head;

public:

doubly():head(NULL){}

bool isempty()

{

if(head == NULL)

{

return true;

}

else

{

return false;

}

}

void insertathead(int val)

{

node\* newer = new node();

newer->data = val;

newer->left = NULL;

if(head == NULL)

{

newer->right = NULL;

}

else

{

node\* temp = head;

temp->left = newer;

newer->right = temp;

}

head = newer;

}

void deleteathead()

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

node\* temp = head;

if(temp->right == NULL)

{

head = NULL;

}

else

{

head = temp->right;

temp->right->left = NULL;

}

cout<<"deleting the value : "<<temp->data<<endl;

delete temp;

}

}

void insertatend(int val)

{

if(isempty())

{

insertathead(val);

}

else

{

node\* newer = new node();

newer->data = val;

newer->right = NULL;

node\* temp = head;

while(temp->right != NULL)

{

temp = temp->right;

}

newer->left = temp;

temp->right = newer;

}

}

void deleteatend()

{

if(isempty())

{

cout<<"deletion is not possible : "<<endl;

}

else

{

node\* temp = head;

node\* temp2 = temp->right;

if(temp->right == NULL)

{

deleteathead();

}

else

{

while(temp2->right != NULL)

{

temp = temp->right;

temp2 = temp->right;

}

cout<<"deleting the value : "<<temp2->data<<endl;

temp->right = NULL;

delete temp2;

}

}

}

void deleteparticular(int val)

{

if(isempty())

{

cout<<"deletion not possible : "<<endl;

}

else

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp->left == NULL)

{

deleteathead();

}

else if(temp->right == NULL)

{

deleteatend();

}

else

{

node\* temp2 = temp->left;

node\* temp3 = temp->right;

temp2->right = temp3;

temp3->left = temp2;

cout<<"deleting the value : "<<temp->data<<endl;

delete temp;

}

}

}

}

void insertbefore(int val)

{

if(isempty())

{

cout<<"list is empty : " <<endl;

}

else

{

node \* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp->left == NULL)

{

int v;

cout<<"enter the value : "<<endl;

cin>>v;

insertathead(v);

}

else

{

node\* newer = new node();

cout<<"enter the value : "<<endl;

cin>>newer->data;

node\* temp2 = head;

while(temp2->right != temp)

{

temp2 = temp2->right;

}

newer->right = temp;

newer->left = temp2;

temp->left = newer;

temp2->right = newer;

}

}

}

}

void insertafter(int val)

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp->right == NULL)

{

int v;

cout<<"enter the value : "<<endl;

cin>>v;

insertatend(v);

}

else

{

node\* newer = new node();

cout<<"enter the value : "<<endl;

cin>>newer->data;

node\* temp2 = temp->right;

newer->right = temp2;

newer->left = temp;

temp->right = newer;

temp2->left = newer;

}

}

}

}

void deletebefore(int val)

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp->left == NULL)

{

cout<<"deletion not possible : "<<endl;

}

else if(temp->left->left == NULL)

{

deleteathead();

}

else

{

node\* temp2 = temp->left->left;

node\* temp3 = temp->left;

temp2->right = temp;

temp->left = temp2;

cout<<"deleting the value : "<<temp3->data<<endl;

delete temp3;

}

}

}

}

void deleteafter(int val)

{

if(isempty())

{

cout<<"list is empty : "<<endl;

}

else

{

node\* temp = searchnode(val);

if(temp == NULL)

{

cout<<"value not found : "<<endl;

}

else

{

if(temp->right == NULL)

{

cout<<"deletion not possible : "<<endl;

}

else if(temp->right->right == NULL)

{

deleteatend();

}

else

{

node\* temp2 = temp->right;

node\* temp3 = temp->right->right;

temp->right = temp3;

temp3->left = temp;

cout<<"deleting the value : "<<temp2->data<<endl;

delete temp2;

}

}

}

}

node\* searchnode(int val)

{

node\* temp = head;

while(temp != NULL)

{

if(temp->data == val)

{

break;

}

temp = temp->right;

}

return temp;

}

void show()

{

node\* temp = head;

while(temp != NULL)

{

cout<<"value is : "<<temp->data<<endl;

temp = temp->right;

}

}

};

int main()

{

doubly d;

d.insertathead(1);

d.insertatend(2);

d.insertatend(3);

d.insertatend(4);

d.insertatend(5);

d.deleteafter(1);

d.show();

return 0;

}

**TREES:**

#include <iostream>

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\*left,\*right;

Node(int val)

{

data=val;

left=nullptr;

right=nullptr;

}

};

class Tree

{

private:

Node\*root;

public:

Tree():root(NULL){}

***///--->> Basic Functions <<---/////***

Node\*addnode(int val)

{

return new Node(val);

}

int info(Node\* node)

{

return node->data;

}

Node\* left(Node\* node)

{

return node->left;

}

Node\* right(Node\* node)

{

return node->right;

}

void setleft(Node\*p,int val)

{

p->left=addnode(val);

}

void setright(Node\*p,int val)

{

p->right=addnode(val);

}

***///--->>Function to Build a tree <<--//***

void BuidTree()

{

int no;

cout << "Enter root number: ";

cin >> no;

root = addnode(no);

cout << "Enter numbers to be inserted in tree (end with -1):" << endl;

while(true)

{

cin>>no;

if(no==-1)

break;

Node\*p,\*q;

p=q=root;

while(no!=info(p)&&q!=NULL)

{

p=q;

if(no<info(p))

q=left(p);

else

q=right(p);

}

if(no==info(p))

cout<<"!!!Duplication!!!"<<endl;

else if(no<info(p))

setleft(p,no);

else

setright(p,no);

}

}

***///---->> PRE ORDER TRAVERSAL <<--/////***

void Preorder(Node\*root)

{

if(root==NULL)

{

return;

}

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

Preorder(root->left);

Preorder(root->right);

}

void DisplayPreorder()

{

Preorder(root);

}

***///---->> POST ORDER TRAVERSAL<<--/////***

void Postorder(Node\*root)

{

if(root==NULL)

{

return;

}

Postorder(root->left);

Postorder(root->right);

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

}

void DisplayPostorder()

{

Postorder(root);

}

***///---->> IN ORDER TRAVERSAL <<-----//////////////***

void inorder(Node\*root)

{

if(root==NULL)

{

return;

}

inorder(root->left);

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

inorder(root->right);

}

void DisplayInorder()

{

inorder(root);

}

***///---->> FIND MAXIMUM <<-----***//////////////

int maX(Node\*Node)

{

if(Node==nullptr)

cout<<"Empty"<<endl;

else if(Node->right==nullptr)

return Node->data;

int mX=maX(Node->right);

return mX;

}

void maXdisplay()

{

int mX=maX(root);

cout<<"Maximum Value is : "<<mX<<endl;

}

***///---->> FIND MINIMUM <<-----//////////////***

int miN(Node\*Node)

{

if(Node==nullptr)

cout<<"Empty"<<endl;

else if(Node->left==nullptr)

return Node->data;

int mN=miN(Node->left);

return mN;

}

void miNdisplay()

{

int mN=miN(root);

cout<<"Minimum Value is : "<<mN<<endl;

}

***///---->> FIND HEIGHT <<-----//////////////***

int height(Node\*Node)

{

if(Node==nullptr)

return -1;

else

{

int leftheight=height(Node->left);

int rightheight=height(Node->right);

return max(leftheight,rightheight)+1;

}

}

void heightdisplay()

{

int hT=height(root);

cout<<"Maximum Height is : "<<hT<<endl;

}

***///---->> FIND NO. OF NODES <<-----//////////////***

int no\_ofnodes(Node\*Node)

{

if(Node==nullptr)

return 0;

else

{

return 1+(no\_ofnodes(Node->left)+no\_ofnodes(Node->right));

}

}

void nodecountdisplay()

{

int nC=no\_ofnodes(root);

cout<<"Number of Nodes is : "<<nC<<endl;

}

***///---->> FIND SUM OF CONTENTS <<-----//////////////***

int sum(Node\*Node)

{

if(Node==nullptr)

return 0;

else

{

return Node->data+(sum(Node->left)+sum(Node->right));

}

}

void sumdisplay()

{

int S=sum(root);

cout<<"Sum of Contents of Nodes is : "<<S<<endl;

}

***///---->> FIND A NUMBER <<-----//////////////***

bool searcH(Node\*Node,int key)

{

if(Node==nullptr)

return false;

if(Node->data==key)

return true;

return searcH(Node->left,key)||searcH(Node->right,key);

}

void searchdisplay()

{

int key;

cout<<"Enter Key : "; cin>>key;

if(searcH(root,key))

cout<<"Target "<<key<<" is found "<<endl;

else

cout<<"Target Not Found "<<endl;

}

};

int main()

{

Tree t;

t.BuidTree();

t.DisplayPreorder(); ///or other orders

t.maXdisplay();

t.miNdisplay();

t.heightdisplay();

t.nodecountdisplay();

t.sumdisplay();

t.searchdisplay();

return 0;

}

/\*

///--->>NON RECURSIVE IN-ORDER TRAVERSAL BY USING STACK<<---/////

void inorder()

{

if(root==NULL)

cout<<"Empty Tree"<<endl;

stack<Node\*>st;

Node\*p;

p=root;

do

{

while(p!=NULL)

{

st.push(p);

p=p->left;

}

if(!st.empty())

{

p=st.top();

st.pop();

cout<<p->data<<" ";

p=p->right;

}

}while(p!=NULL||!st.empty());

}\*/

**THREADED-TREES**

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\*left,\*right;

bool thread;

/\*Node(int val)

{

data=val;

left=NULL;

right=NULL;

thread=true;

}\*/

};

Node\*maketree(int valu)

{

Node\*newer=new Node();

newer->data=valu;

newer->left=NULL;

newer->right=NULL;

newer->thread=true;

}

void setleft(Node\*p,int val)

{

if(p==NULL)

cout<<"void insertion"<<endl;

else if(p->left!=NULL)

cout<<"Already occupied(invalid insertion)"<<endl;

else

{

Node\*newer=new Node();

newer->data=val;

newer->left=NULL;

newer->right=p;

newer->thread=true;

}

}

void setright(Node\*p,int val)

{

if(p==NULL)

cout<<"void insertion"<<endl;

else if(p->right!=NULL)

cout<<"Already occupied(invalid insertion)"<<endl;

else

{

Node\*newer=new Node();

Node\*r=p->right;

p->right=newer;

p->thread=false;

newer->data=val;

newer->left=NULL;

newer->right=r;

newer->thread=true;

}

}

int main()

{

cout << "Hello world!" << endl;

return 0;

}