**ASSIGNMENT#1**

#include <iostream>

using namespace std;

//Menu driven program

//Creation

void createArray()

{

int n;

cout<<"Enter the number of elements in the array"<<endl;

cin>>n;

int b[n];

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

{

cin>>b[i];

}

cout<<endl;

}

//Display

void displayArray(int a[],int l)

{

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

{

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

}

cout<<endl;

}

//Insertion

void insertArray(int a[],int l)

{

int pos,value;

cout<<"Enter the value to be inserted"<<endl;

cin>>value;

cout<<"Enter the index value at which element has to be inserted"<<endl;

cin>>pos;

int b[l+1];

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

{

b[i]=a[i];

}

b[pos]=value;

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

{

b[i]=a[i-1];

}

for(int i=0;i<l+1;i++)

{

cout<<b[i]<<" ";

}

cout<<endl;

}

//Deletion

void deleteArray(int a[],int l)

{

int val,i,j;

cout<<"Enter the value to be deleted"<<endl;

cin>>val;

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

{

if(a[i]==val)

{

for(j=i; j<l; j++)

{

a[j] = a[j+1];

}

l--;

}

}

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

{

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

}

cout<<endl;

}

//Searching

void searchArray(int a[],int l)

{

int val;

cout<<"Enter the value to be searched "<<endl;

cin>>val;

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

{

if(a[i]==val) cout<<"Element found at index "<<i<<endl;

}

}

int main()

{

/\*1.CREATE

2.DISPLAY

3.INSERT

4.DELETE

5.SEARCH

6.EXIT\*/

int a[]={11,22,33,44,55,66,77,88,99};

int l=sizeof(a)/sizeof(a[0]);

int t;

while(true)

{

cout<<"Enter the number coresponding to the required function"<<endl;

cin>>t;

switch(t)

{

case 1:createArray();break;

case 2:displayArray(a,l);break;

case 3:insertArray(a,l);break;

case 4:deleteArray(a,l);break;

case 5:searchArray(a,l);break;

case 6: return 0;

}

}

return 0;

}

#include <iostream>

using namespace std;

*//Reverse an array*

int main()

{

int A[5]={11,22,33,44,55},B[5],i;

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

{

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

}

cout<<endl;

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

{

B[i]=A[5-i-1];

}

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

{

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

}

return 0;

}

#include <iostream>

using namespace std;

*//Matrix Multiplication*

int main()

{

int r1,c1,c2,i,j,k;

cout<<"Enter the number of rows in first matrix : ";

cin>>r1;

cout<<"Enter the nmber of columns in first matrix : ";

cin>>c1;

cout<<"Enter the number of columns in second matrix : ";

cin>>c2;

int m1[r1][c1],m2[c1][c2];

int product[r1][c2];

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

for(int i=0; i<r1; ++i)

{

for(j=0; j<c1; ++j)

cin>>m1[i][j];

cout<<endl;

}

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

for(i=0; i<c1; ++i)

{

for(j=0; j<c2; ++j)

cin>>m2[i][j];

cout<<endl;

}

cout<<"The first matrix is:"<<endl;

for(i=0; i<r1; ++i) {

for(j=0; j<c1; ++j)

cout<<m1[i][j]<<" ";

cout<<endl;

}

cout<<endl;

cout<<"The second matrix is:"<<endl;

for(i=0; i<c1; ++i) {

for(j=0; j<c2; ++j)

cout<<m2[i][j]<<" ";

cout<<endl;

}

for(i=0; i<r1; ++i)

for(j=0; j<c2; ++j)

for(k=0; k<c1; ++k)

{

product[i][j]= product[i][j] + m1[i][k] \* m2[k][j];

}

cout<<"Product of the two matrices is\n"<<endl;

for(i=0; i< r1; ++i)

{

for(j=0; j< c2; ++j)

{

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

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

*//Transpose of a matrix*

int main()

{

int r,c,i,j,i2,j2;

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

cin>>r;

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

cin>>c;

int a[r][c],b[c][r];

cout<<"Enter the value of elements(row major order)"<<endl;

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

{

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

{

cin>>a[i][j];

}

}

cout<<"Original matrix is"<<endl;

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

{

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

{

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

}

cout<<endl;

}

for( i=0,i2=0;i<r,i2<c;i++,i2++)

{

for( j=0,j2=0;j<c,j2<r;j++,j2++)

{

b[i2][j2]=a[j][i];

}

}

cout<<"Transpose of the matrix is"<<endl;

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

{

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

{

cout<<b[i][j]<<"\t";

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

*//Bubble Sort*

int main()

{

int a[]={64,34,25,12,22,11,90};

for(int i=0;i<6;i++)

{

for (int j=0;j<7-i-1;j++)

{

if(a[j]>a[j+1])

{

int temp;

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

for(int i=0;i<7;i++)

{

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

}

return 0;

}

int size = sizeof(a) / sizeof(a[0]);

int cd=abs(a[6]-a[0])/7;

for(int i=1;i<7;i++)

{

if(a[i]!=a[i-1]+cd)

{

cout<<"The missing element is : "<<a[i-1]+cd<<endl;

break;

}

}

}

**ASSIGNMENT #1A**

#include <iostream>

using namespace std;

int main()

{

int a[100][100],i,j,r,c;

cout<<"Enter the number of rows and columns "<<endl;

cin>>r>>c;

cout<<"Enter the sparse matrix"<<endl;

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

{

for(int j=0;j<c;j++)

{

cin>>a[i][j];

}

}

cout<<"The trplet form of the sparse matrix is"<<endl;

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

{

for(int j=0;j<c;j++)

{

if(a[i][j] !=0)

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

}

}

return 0;

}

#include <iostream>

using namespace std;

int main()

{

int a[100][100],i,j,r,c,count=0;

cout<<"Enter the number of rows and columns "<<endl;

cin>>r>>c;

cout<<"Enter the sparse matrix"<<endl;

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

{

for(int j=0;j<c;j++)

{

cin>>a[i][j];

}

}

cout<<"The triplet form of the sparse matrix is"<<endl;

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

{

for(int j=0;j<c;j++)

{

if(a[i][j] !=0) count++;

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

}

}

int b[count][3];

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

{

for(int j=0;j<c;j++)

{

if(a[i][j] !=0)

{

b[0][j]=i;

b[1][j]=j;

b[2][j]=a[i][j];

}

}

}

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

{

for(int j=0;j<3;j++)

{

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

}

cout <<endl;

}

return 0;

}

#include <iostream>

using namespace std;

//Diagonal matrix

int main()

{

int n,i,j,k=0;

cout<<"Enter the order of matrix : ";

cin>>n;

int diag[n];

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

{

cout<<"Enter the diagonal element number "<<i+1<<" : ";

cin>>diag[i];

cout<<endl;

}

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

{

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

{

if(i==j)

{

cout<<diag[k]<<" ";

k++;

}

else

{

cout<<"0 ";

}

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

//Tri-diagonal matrix

int main()

{

int n,i,j,k=0;

cout<<"Enter the order of matrix : ";

cin>>n;

int t=3\*n-2;

int tri[t];

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

{

cout<<"Enter the tri-diagonal element number "<<i+1<<" : ";

cin>>tri[i];

cout<<endl;

}

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

{

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

{

if(i-j ==1 || j-i ==1 || i==j)

{

cout<<tri[k]<<" ";

k++;

}

else

{

cout<<"0 ";

}

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

//Tri-diagonal matrix

int main()

{

int n,i,j,k=0;

cout<<"Enter the order of matrix : ";

cin>>n;

int t=(n\*(n+1))/2;

int a[t];

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

{

cout<<"Enter the lower diag element number "<<i+1<<" : ";

cin>>a[i];

cout<<endl;

}

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

{

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

{

if(i>=j)

{

cout<<a[k]<<" ";

k++;

}

else

{

cout<<"0 ";

}

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

//Tri-diagonal matrix-Upper

int main()

{

int n,i,j,k=0;

cout<<"Enter the order of matrix : ";

cin>>n;

int t=(n\*(n+1))/2;

int a[t];

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

{

cout<<"Enter the lower diag element number "<<i+1<<" : ";

cin>>a[i];

cout<<endl;

}

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

{

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

{

if(i<=j)

{

cout<<a[k]<<" ";

k++;

}

else

{

cout<<"0 ";

}

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

//Symmetric Matrix

int main()

{

int n,i,j,k;

cout<<"Enter the order of matrix : ";

cin>>n;

int t= (n\*(n+1))/2;

int b[t];

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

{

cout<<"Enter the element number "<<i+1<<" : ";

cin>>b[i];

cout<<endl;

}

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

{

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

{

if(i<j)

{

k=i+(j\*(j+1))/2;

}

else

{

k=j+(i\*(i+1))/2;

}

cout<<b[k]<<" ";

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

int main()

{

int n,count=0;

cout<<"Enter the order of the matrix"<<endl;

cin>>n;

int a[n][n];

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

{

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

{

cin>>a[i][j];

if(a[i][j] !=0)

{

count++;

}

}

}

int tripletMatrix[count][3];

int k=0;

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

{

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

{

if(a[i][j] !=0)

{

tripletMatrix[k][0]=i;

tripletMatrix[k][1]=j;

tripletMatrix[k][2]=a[i][j];

k++;

}

}

}

cout<<"\n The triplet representation is\n";

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

{

for(int j=0;j<3;j++)

{

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

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

int main()

{

int r,c,count=0;

cout<<"\nRows : ";

cin>>r;

cout<<"\nColumns : ";

cin>>c;

int a[r][c];

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

{

for(int j=0;j<c;j++)

{

cin>>a[i][j];

if(a[i][j] !=0)

{

count++;

}

}

}

int tripletMatrix[count][3];

int k=0;

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

{

for(int j=0;j<c;j++)

{

if(a[i][j] !=0)

{

tripletMatrix[k][0]=i;

tripletMatrix[k][1]=j;

tripletMatrix[k][2]=a[i][j];

k++;

}

}

}

cout<<"\n The triplet representation is\n";

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

{

for(int j=0;j<3;j++)

{

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

}

cout<<endl;

}

return 0;

}

#include <iostream>

using namespace std;

#define MAX 10

//Sparse Matrix Addition

void Print(int b[MAX][3]);

void Read(int b[MAX][3]);

void Add(int b1[MAX][3],int b2[MAX][3],int b3[MAX][3]);

int main()

{

int b1[MAX][3];

int b2[MAX][3];

int b3[MAX][3];

Read(b1);

cout<<"Matrix 1"<<endl;

Print(b1);

cout<<"Matrix 2"<<endl;

Read(b2);

Print(b2);

cout<<"Matrix 1 + Matrix 2"<<endl;

Add(b1,b2,b3);

Print(b3);

return 0;

}

void Read(int b[MAX][3])

{

int i,r,c,count=0;

cout<<"\nRows : ";

cin>>r;

cout<<"\nColumns : ";

cin>>c;

int a[r][c];

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

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

{

for(int j=0;j<c;j++)

{

cin>>a[i][j];

if(a[i][j] !=0)

{

count++;

}

}

}

b[0][0]=r;

b[0][1]=c;

b[0][2]=count;

int k=1;

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

{

for(int j=0;j<c;j++)

{

if(a[i][j] !=0)

{

b[k][0]=i;

b[k][1]=j;

b[k][2]=a[i][j];

k++;

}

}

}

}

void Add(int b1[MAX][3],int b2[MAX][3],int b3[MAX][3])

{

int t1,t2,i,j,k;

if(b1[0][0] != b2[0][0] || b1[0][1] != b2[0][1] )

{

cout<<"Matrices can't be added";

exit(0);

}

t1=b1[0][2];

t2=b1[0][2];

i=j=1;

k=1;

b3[0][0]=b1[0][0];

b3[0][1]=b1[0][1];

while(i<t1 && j<t2)

{

if(b1[i][0]<b2[j][0])

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

k++;

i++;

}

else if(b1[i][0]>b2[j][0])

{

b3[k][0]=b2[j][0];

b3[k][1]=b2[j][1];

b3[k][2]=b2[j][2];

k++;

j++;

}

else if(b1[i][1]>b2[j][1])

{

b3[k][0]=b2[j][0];

b3[k][1]=b2[j][1];

b3[k][2]=b2[j][2];

k++;

j++;

}

else if(b1[i][1]<b2[j][1])

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

k++;

i++;

}

else

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2]+b2[j][2];

k++;

i++;

j++;

}

}

while(i<=t1)

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

k++;

i++;

}

while(j<=t2)

{

b3[k][0]=b2[j][0];

b3[k][1]=b2[j][1];

b3[k][2]=b2[j][2];

k++;

j++;

}

b3[0][2]=k-1;

}

void Print(int b[MAX][3])

{

int i,t;

t=b[0][2];

cout<<"The matrix in triplet form is "<<endl;

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

{

cout<<b[i][0]<<" "<<b[i][1]<<" "<<b[i][2]<<endl;

}

}

#include <iostream>

using namespace std;

#define MAX 10

//To find the transpose of the matrix

void Print(int b[MAX][3]);

void Read(int b[MAX][3]);

void Transpose(int b1[MAX][3],int b2[MAX][3]);

int main()

{

int b1[MAX][3];

int b2[MAX][3];

Read(b1);

cout<<"Matrix 1"<<endl;

Print(b1);

cout<<"Transpose of matrix is"<<endl;

Transpose(b1,b2);

Print(b2);

return 0;

}

// To read the matrix

void Read(int b[MAX][3])

{

int i,r,c,count=0;

cout<<"\nRows : ";

cin>>r;

cout<<"\nColumns : ";

cin>>c;

int a[r][c];

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

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

{

for(int j=0;j<c;j++)

{

cin>>a[i][j];

if(a[i][j] !=0)

{

count++;

}

}

}

b[0][0]=r;

b[0][1]=c;

b[0][2]=count;

int k=1;

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

{

for(int j=0;j<c;j++)

{

if(a[i][j] !=0)

{

b[k][0]=i;

b[k][1]=j;

b[k][2]=a[i][j];

k++;

}

}

}

}

//To find the transpose of the matrix

void Transpose(int b1[MAX][3],int b2[MAX][3])

{

int i,j,k=1,n;

b2[0][0]=b1[0][1];

b2[0][1]=b1[0][0];

b2[0][2]=b1[0][2];

n=b1[0][2];

for(i=0;i<=b1[0][1];i++)

{

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

{

if(i==b1[j][1])

{

b2[k][0]=b1[j][1];

b2[k][1]=b1[j][0];

b2[k][2]=b1[j][2];

k++;

}

}

}

}

//To print the matrix

void Print(int b[MAX][3])

{

int i,t;

t=b[0][2];

cout<<"The matrix in triplet form is "<<endl;

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

{

cout<<b[i][0]<<" "<<b[i][1]<<" "<<b[i][2]<<endl;

}

}

#include <iostream>

using namespace std;

#define MAX 10

//SPARSE MATRIX MULTIPLICATION

void Print(int b[MAX][3]);

void Read(int b[MAX][3]);

void Multiply(int b1[MAX][3],int b2[MAX][3],int b3[MAX][3]);

void Transpose(int b1[MAX][3],int b3[MAX][3]);

int main()

{

int b1[MAX][3];

int b2[MAX][3];

int b3[MAX][3];

int b4[MAX][3];

Read(b1);

cout<<"Matrix 1"<<endl;

Print(b1);

cout<<"Matrix 2"<<endl;

Read(b2);

Print(b2);

Transpose(b1,b4);

cout<<"Matrix 1 \* Matrix 2"<<endl;

Multiply(b4,b2,b3);

Print(b3);

return 0;

}

void Read(int b[MAX][3])

{

int i,r,c,count=0;

cout<<"\nRows : ";

cin>>r;

cout<<"\nColumns : ";

cin>>c;

int a[r][c];

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

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

{

for(int j=0;j<c;j++)

{

cin>>a[i][j];

if(a[i][j] !=0)

{

count++;

}

}

}

b[0][0]=r;

b[0][1]=c;

b[0][2]=count;

int k=1;

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

{

for(int j=0;j<c;j++)

{

if(a[i][j] !=0)

{

b[k][0]=i;

b[k][1]=j;

b[k][2]=a[i][j];

k++;

}

}

}

}

void Transpose(int b1[MAX][3],int b2[MAX][3])

{

int i,j,k=1,n;

b2[0][0]=b1[0][1];

b2[0][1]=b1[0][0];

b2[0][2]=b1[0][2];

n=b1[0][2];

for(i=0;i<=b1[0][1];i++)

{

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

{

if(i==b1[j][1])

{

b2[k][0]=b1[j][1];

b2[k][1]=b1[j][0];

b2[k][2]=b1[j][2];

k++;

}

}

}

}

void Multiply(int b1[MAX][3],int b2[MAX][3],int b3[MAX][3])

{

int t1,t2,i,j,k;

t1=b1[0][2];

t2=b1[0][2];

i=j=1;

k=1;

b3[0][0]=b1[0][0];

b3[0][1]=b1[0][1];

while(i<t1 && j<t2)

{

if(b1[i][0]<b2[j][0])

{

i++;

}

else if(b1[i][0]>b2[j][0])

{

j++;

}

else if(b1[i][1]>b2[j][1])

{

j++;

}

else if(b1[i][1]<b2[j][1])

{

i++;

}

else

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2]\*b2[j][2];

k++;

i++;

j++;

}

}

while(i<=t1)

{

i++;

}

while(j<=t2)

{

j++;

}

b3[0][2]=k-1;

}

void Print(int b[MAX][3])

{

int i,t;

t=b[0][2];

cout<<"The matrix in triplet form is "<<endl;

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

{

cout<<b[i][0]<<" "<<b[i][1]<<" "<<b[i][2]<<endl;

}

}

**STACKS**

#include <iostream>

using namespace std;

#define n 100

class Stack

{

int \*a;

int top;

public:

Stack()

{

a=new int[n];

top=-1;

}

void push(int x)

{

if(top==n-1)

{

cout<<"Stack Overflow"<<endl;

return ;

}

top++;

a[top]=x;

}

void pop()

{

if(top==-1)

{

cout<<"The stack is empty so no elements to pop"<<endl;

return ;

}

top--;

}

int Top()

{

if(top==-1)

{

cout<<"The stack is empty "<<endl;

return 0;

}

return a[top];

}

bool empty()

{

return top==-1;

}

};

int main()

{

Stack st;

st.push(1);

st.push(2);

st.push(3);

st.push(4);

st.push(5);

cout<<st.Top()<<endl;

st.pop();

st.pop();

st.pop();

st.pop();

st.pop();

st.pop();

cout<<st.empty()<<endl;

return 0;

}

#include <iostream>

using namespace std;

#define MAX 100

class Stack

{

int top;

public:

int a[MAX];

Stack()

{

top=-1;

}

bool push(int x);

int pop();

int peek();

bool isEmpty();

};

bool Stack::push(int x)

{

if(top>= MAX-1)

{

cout<<"Stack overflow"<<endl;

return false;

}

else

{

a[++top]=x;

cout<<x<<" is pushed into the stack"<<endl;

return true;

}

}

int Stack::pop(void)

{

if(top<= -1)

{

cout<<"There are no elements in the stack"<<endl;

return 0;

}

else

{

int x=a[top--];

return x;

}

}

int Stack::peek(void)

{

if(top<= -1)

{

cout<<"There are no elements in the stack"<<endl;

return 0;

}

else

{

int x=a[top];

return x;

}

}

bool Stack::isEmpty(void)

{

return (top<= -1);

}

int main()

{

Stack s;

s.push(11);

s.push(22);

s.push(33);

s.push(44);

s.push(55);

s.push(66);

cout<<s.pop()<<" is the popped out"<<endl;

cout<<"The elements of the stack are "<<endl;

while(!s.isEmpty())

{

cout<<s.peek()<<" ";

s.pop();

//int a = s.peek();

// cout<<a<<" ";

}

return 0;

}

#include <iostream>

#include <stack>

#include <string>

using namespace std;

void reverseString(string s)

{

stack<string> st;

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

{

string word="";

while(s[i] !=' ' && i<s.length())

{

word +=s[i];

i++;

}

st.push(word);

}

while(!st.empty())

{

cout<<st.top()<<" "<<endl;

st.pop();

}

cout<<endl;

}

int main()

{

string s="Hey, my name is Mishti";

reverseString(s);

return 0;

}

#include <iostream>

#include <stack>

#include <string>

#include <math.h>

using namespace std;

// Program for prefix evaluation

int prefixEvaluvation(string s)

{

stack<int> st;

for(int i=s.length()-1;i>=0;i--)

{

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

{

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

}

else

{

int op1=st.top();

st.pop();

int op2=st.top();

st.pop();

switch(s[i])

{

case '+':st.push(op1+op2);break;

case '-':st.push(op1-op2);break;

case '\*':st.push(op1\*op2);break;

case '/':st.push(op1/op2);break;

case '^':st.push(pow(op1,op2));break;

default :break;

}

}

}

return st.top();

}

int main()

{

//string s="-+7\*45+20";

cout<<prefixEvaluvation("-+7\*45+20");

return 0;

}

#include <iostream>

#include <stack>

#include <string>

#include <math.h>

using namespace std;

// Program for post evaluation

int postfixEvaluvation(string s)

{

stack<int> st;

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

{

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

{

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

}

else

{

int op2=st.top();

st.pop();

int op1=st.top();

st.pop();

switch(s[i])

{

case '+':st.push(op1+op2);break;

case '-':st.push(op1-op2);break;

case '\*':st.push(op1\*op2);break;

case '/':st.push(op1/op2);break;

case '^':st.push(pow(op1,op2));break;

default:break;

}

}

}

return st.top();

}

int main()

{

cout<<postfixEvaluvation("46+2/5\*7+");

return 0;

}

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int val)

{

data=val;

next=NULL;

}

};

void insertAtHead(Node\* &head,int val)

{

Node \*n=new Node(val);

if(head==NULL)

{

head=n;

return;

}

n->next=head;

head=n;

}

void insertAtTail(Node\* &head,int val)

{

Node \*n=new Node(val);

if(head==NULL)

{

head=n;

return;

}

Node \*temp=head;

while(temp->next !=NULL)

{

temp=temp->next;

}

temp->next=n;

}

bool Search(Node\* head,int key)

{

Node\* temp=head;

while(temp->next !=NULL)

{

if(temp->data==key)

{

return true;

}

temp=temp->next;

}

return false;

}

void PrintList(Node\* head)

{

Node \*temp=head;

while(temp!=NULL)

{

cout<<temp->data<<" -> ";

temp=temp->next;

}

cout<<"NULL"<<endl;

}

void deletionAtHead(Node\* &head)

{

Node \*todelete=head;

head=head->next;

delete todelete;

}

void deletion(Node\* &head,int key)

{

if(head==NULL)

{

return ;

}

if(head->next==NULL)

{

deletionAtHead(head);

return;

}

Node\* temp=head;

while(temp->next->data !=key)

{

temp=temp->next;

}

Node \*todelete=temp->next;

temp->next=temp->next->next;

delete todelete;

}

Node\* reverse(Node \*&head)

{

Node \*prevptr=NULL;

Node \*currptr=head;

Node \*nextptr;

while(currptr!=NULL)

{

nextptr=currptr->next;

currptr->next=prevptr;

prevptr=currptr;

currptr=nextptr;

}

return prevptr;

}

Node\* reverseRecursive(Node \*&head)

{

if(head==NULL || head->next==NULL)

{

return head;

}

Node \*newHead=reverseRecursive(head->next);

head->next->next=head;

head->next=NULL;

return newHead;

}

int main()

{

Node \*head=NULL;

insertAtTail(head,11);

insertAtTail(head,22);

insertAtTail(head,33);

insertAtTail(head,44);

insertAtTail(head,55);

insertAtTail(head,66);

insertAtHead(head,77);

PrintList(head);

cout<<Search(head,5)<<endl;

cout<<Search(head,55)<<endl;

deletion(head,44);

deletionAtHead(head);

Node\* newHead =reverseRecursive(head);

PrintList(newHead);

return 0;

}

#include <iostream>

using namespace std;

//Reversing the link list: Iteratively , Recursively , K nodes at a time

class Node

{

public:

int data;

Node \*next;

Node(int val)

{

data=val;

next=NULL;

}

};

void insertAtHead(Node\* &head,int val)

{

Node \*n=new Node(val);

if(head==NULL)

{

head=n;

return;

}

n->next=head;

head=n;

}

void insertAtTail(Node\* &head,int val)

{

Node \*n=new Node(val);

if(head==NULL)

{

head=n;

return;

}

Node \*temp=head;

while(temp->next !=NULL)

{

temp=temp->next;

}

temp->next=n;

}

bool Search(Node\* head,int key)

{

Node\* temp=head;

while(temp->next !=NULL)

{

if(temp->data==key)

{

return true;

}

temp=temp->next;

}

return false;

}

void PrintList(Node\* head)

{

Node \*temp=head;

while(temp!=NULL)

{

cout<<temp->data<<" -> ";

temp=temp->next;

}

cout<<"NULL"<<endl;

}

void deletionAtHead(Node\* &head)

{

Node \*todelete=head;

head=head->next;

delete todelete;

}

void deletion(Node\* &head,int key)

{

if(head==NULL)

{

return ;

}

if(head->next==NULL)

{

deletionAtHead(head);

return;

}

Node\* temp=head;

while(temp->next->data !=key)

{

temp=temp->next;

}

Node \*todelete=temp->next;

temp->next=temp->next->next;

delete todelete;

}

Node\* reverse(Node \*&head)

{

Node \*prevptr=NULL;

Node \*currptr=head;

Node \*nextptr;

while(currptr!=NULL)

{

nextptr=currptr->next;

currptr->next=prevptr;

prevptr=currptr;

currptr=nextptr;

}

return prevptr;

}

Node\* reverseRecursive(Node \*&head)

{

if(head==NULL || head->next==NULL)

{

return head;

}

Node \*newHead=reverseRecursive(head->next);

head->next->next=head;

head->next=NULL;

return newHead;

}

//Reversing k nodesc at a time

Node \* reversek(Node \*&head,int k)

{

int count=0;

Node \*prevptr=NULL;

Node \*currptr=head;

Node \*nextptr;

while(count<k && currptr!=NULL)

{

nextptr=currptr->next;

currptr->next=prevptr;

prevptr=currptr;

currptr=nextptr;

count++;

}

if(nextptr !=NULL)

head->next=reversek(nextptr,k);

return prevptr;

}

int main()

{

Node \*head=NULL;

insertAtTail(head,11);

insertAtTail(head,22);

insertAtTail(head,33);

insertAtTail(head,44);

insertAtTail(head,55);

insertAtTail(head,66);

insertAtHead(head,77);

PrintList(head);

Node\* newHead =reversek(head,2);

PrintList(newHead);

return 0;

}

#include <iostream>

using namespace std;

//Floyd's algorithm Or Hare and tortoise algorithm

class Node

{

public:

int data;

Node \*next;

Node(int val)

{

data=val;

next=NULL;

}

};

void insertAtHead(Node\* &head,int val)

{

Node \*n=new Node(val);

if(head==NULL)

{

head=n;

return;

}

n->next=head;

head=n;

}

void insertAtTail(Node\* &head,int val)

{

Node \*n=new Node(val);

if(head==NULL)

{

head=n;

return;

}

Node \*temp=head;

while(temp->next !=NULL)

{

temp=temp->next;

}

temp->next=n;

}

bool Search(Node\* head,int key)

{

Node\* temp=head;

while(temp->next !=NULL)

{

if(temp->data==key)

{

return true;

}

temp=temp->next;

}

return false;

}

void PrintList(Node\* head)

{

Node \*temp=head;

while(temp!=NULL)

{

cout<<temp->data<<" -> ";

temp=temp->next;

}

cout<<"NULL"<<endl;

}

void deletionAtHead(Node\* &head)

{

Node \*todelete=head;

head=head->next;

delete todelete;

}

void deletion(Node\* &head,int key)

{

if(head==NULL)

{

return ;

}

if(head->next==NULL)

{

deletionAtHead(head);

return;

}

Node\* temp=head;

while(temp->next->data !=key)

{

temp=temp->next;

}

Node \*todelete=temp->next;

temp->next=temp->next->next;

delete todelete;

}

bool detectCycle(Node\* &head)

{

Node \*slow=head;

Node \*fast=head;

while(fast!=NULL && fast->next !=NULL)

{

slow=slow->next;

fast=fast->next->next;

if(fast==slow)

{

return true;

}

}

return false;

}

int main()

{

Node \*head=NULL;

insertAtTail(head,11);

insertAtTail(head,22);

insertAtTail(head,33);

insertAtTail(head,44);

insertAtTail(head,55);

insertAtTail(head,66);

insertAtHead(head,77);

PrintList(head);

cout<<detectCycle(head);

return 0;

}

#include <iostream>

using namespace std;

class Node

{

public:

char data;

Node \*next;

Node(char val)

{

data=val;

next=NULL;

}

};

void insertAtHead(Node\* &head,char val)

{

Node \*n=new Node(val);

if(head==NULL)

{

head=n;

return;

}

n->next=head;

head=n;

}

void insertAtTail(Node\* &head,char val)

{

Node \*n=new Node(val);

if(head==NULL)

{

head=n;

return;

}

Node \*temp=head;

while(temp->next !=NULL)

{

temp=temp->next;

}

temp->next=n;

}

bool Search(Node\* head,char key)

{

Node\* temp=head;

while(temp->next !=NULL)

{

if(temp->data==key)

{

return true;

}

temp=temp->next;

}

return false;

}

void PrintList(Node\* head)

{

Node \*temp=head;

while(temp!=NULL)

{

cout<<temp->data<<" --> ";

temp=temp->next;

}

cout<<"NULL"<<endl;

}

void deletionAtHead(Node\* &head)

{

Node \*todelete=head;

head=head->next;

delete todelete;

}

void deletion(Node\* &head,int key)

{

if(head==NULL)

{

return ;

}

if(head->next==NULL)

{

deletionAtHead(head);

return;

}

Node\* temp=head;

while(temp->next->data !=key)

{

temp=temp->next;

}

Node \*todelete=temp->next;

temp->next=temp->next->next;

delete todelete;

}

Node\* reverse(Node \*&head)

{

Node \*prevptr=NULL;

Node \*currptr=head;

Node \*nextptr;

while(currptr!=NULL)

{

nextptr=currptr->next;

currptr->next=prevptr;

prevptr=currptr;

currptr=nextptr;

}

return prevptr;

}

void insertionAtMid(Node\* &head, char key,char c)

{

Node\* newNode = new Node(key);

if (head == NULL)

{

head = newNode;

}

else

{

Node\* ptr = head;

int len = 0;

while (ptr != NULL)

{

len++;

ptr = ptr->next;

}

int count;

if(len%2==0)

{

if(c >= key)

count=(len/2) +1;

else

count=(len/2);

}

else

{

count=(len+1)/2;

}

ptr = head;

while (count-- > 1)

{

ptr = ptr->next;

}

newNode->next = ptr->next;

ptr->next = newNode;

}

}

char printMiddle(Node \*head)

{

Node \*slowPtr = head;

Node \*fastPtr = head;

if (head!=NULL)

{

while (fastPtr != NULL && fastPtr->next != NULL)

{

fastPtr = fastPtr->next->next;

slowPtr = slowPtr->next;

}

return slowPtr->data;

}

return slowPtr->data;

}

int main()

{

Node\* head=NULL;

char s[100],com;

cin.get(s, 1000);

int i=1;

insertAtTail(head,s[0]);

while(s[i])

{

if(!Search(head,s[i]) )

{

insertAtTail(head,s[i]);

}

else if(Search(head,s[i]))

{

char key=s[i];

deletion(head,key);

//com=;

insertionAtMid(head,s[i],printMiddle(head));

}

i++;

}

PrintList(head);

return 0;

}

QUEUE

#include <iostream>

using namespace std;

class Queue

{

public:

int front ,rear,size;

int capacity;

int \*array;

};

Queue \*createQueue(int capacity)

{

Queue \*queue =new Queue();

queue->capacity=capacity;

queue->front=queue->size=0;

queue->rear=capacity-1;

queue->array=new int[queue->capacity];

return queue;

}

int isEmpty(Queue \*queue)

{

return (queue->size==0);

}

int isFull(Queue \*queue)

{

return (queue->capacity==queue->size);

}

int main()

{

cout<<"Hello World";

return 0;

}

#include <iostream>

#include <bits/stdc++.h>

#define debug(x) cout<<x<<endl;

using namespace std;

template <typename T>

class Queue

{

int f,r,ms,cs;

T \*a;

public:

Queue(int ds=4)

{

f=0;

ms=ds;

r=ms-1;

a=new T[ms];

cs=0;

}

bool isFull()

{

return cs==ms;

}

bool isEmpty()

{

return cs==0;

}

void push(T value)

{

if(! isFull())

{

r=(r+1)%ms;

a[r]=value;

cs++;

}

}

void pop()

{

if(!isEmpty())

{

f=(f+1)%ms;

cs--;

}

}

T getFront()

{

return a[f];

}

};

int main()

{

Queue <int> q;

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

{

q.push(i);

}

cout<<endl;

while(!q.isEmpty())

{

cout<<q.getFront()<<" ";

q.pop();

}

return 0;

}