#include<iostream>

using namespace std;

template <typename T>

class Node

{

public:

Node();// constructor

Node(T element);

//sets the KeyType data in the Node

void setData(T pVal);

// returns the KeyType data in the Node

T getData();

// returns the link to the next node

Node\* GetNext();

// sets the link to the next node

void SetNext(Node \*x);

private:

T data;

Node \*link;

};

//////////////////////////////////////////

template <typename T>

Node<T>::Node()

{

data = 0;

link = 0;

}

template <typename T>

Node<T>::Node(T element)

{

data = element;

link = 0;

};

template <class T>

void Node<T>::setData(T pVal)

{

data = pVal;

};

template <class T>

T Node<T>::getData()

{

return data;

};

template <class T>

Node<T>\* Node<T>::GetNext()

{

return link;

};

template <class T>

void Node<T>::SetNext(Node \*x)

{

link = x;

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SLIST\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <typename T>

class List

{

public:

// constructor of the Singly Linked List

List();

//Returns the pointer to the first node of the list

Node<T>\* GetFirst();

//Inserts the node pNew after the node pBefore

// if the list is empty, it makes pNew the first node of the list

void Insert(Node<T>\* pBefore, Node<T>\* pNew);

//Deletes the node pToBeDeleted

void Delete(Node<T>\* pToBeDeleted);

//prints the contents of the list

void printList();

//Recursive function that prints list contents in reverse order

void printReverse();

private:

Node<T> \*first;

void printReverseR(Node<T>\* n);

};

///////////////////////////

template<class T>

List<T>::List()

{

first = 0;

};

template<class T>

Node<T>\* List<T>::GetFirst()

{

return first;

};

template <class T>

void List<T>::Insert(Node<T>\* pBefore, Node<T>\* pNew)

{

if (!first)

{

first = pNew;

}

else if (first && !pBefore)

{

pNew->SetNext(first);

first = pNew;

}

else

{

pNew->SetNext(pBefore->GetNext());

pBefore->SetNext(pNew);

}

};

template <class T>

void List<T>::printList()

{

cout << " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

Node<T> \*temp;

temp = first;

int i = 0;

while (temp)

{

T test2 = temp->getData();

cout << "Value in Node # " << i + 1 << " is " << test2 << endl;

temp = temp->GetNext();

i++;

}

};

template <class T>

void List<T>::printReverseR(Node<T>\* n)

{

if (n)

{

printReverseR(n->GetNext());

cout << n->getData() << " ";

}

};

template <class T>

void List<T>::printReverse()

{

printReverseR(first);

};

template <class T>

void List<T>::Delete(Node<T>\* pToBeDeleted)

{

Node<T> \*tmp;

tmp = first;

if (pToBeDeleted == first)

{ //cout<< " in the if "<<endl;

first = first->GetNext();

}

else{

while (tmp->GetNext() != pToBeDeleted)

{

tmp = tmp->GetNext();

}

tmp->SetNext(pToBeDeleted->GetNext());

}

delete pToBeDeleted;

};

//queue

template<class DT>

class Queue

{

private:

List<DT>\*list;

Node<DT>\*rear;

Node<DT>\*front;//include private variables according to the underlying data structure

public:

//contructor

Queue();

//puts element at the rear end of the Queue if it is not full. Must be O(1)

void Put(DT element);

//if queue not empty then delete the element at front of the Queue. Must be O(1)

DT Get();

//return true if the Queue is empty and false if it is not

bool IsEmpty();

//return true if the Queue is full and false if it is not

bool IsFull();

};

///////////////////////////

template<class DT>

Queue<DT>::Queue()

{

list = new List<DT>();

rear = 0;

front = 0;

}

template<class DT>

void Queue<DT>::Put(DT a)

{

Node<DT>\*n = new Node<DT>();

n->setData(a);

list->Insert(rear, n);

rear = n;

if (!front)

{

front = rear;

}

}

template<class DT>

bool Queue<DT>::IsEmpty()

{

return(front == 0 && rear == 0);

}

template<class DT>

bool Queue<DT>::IsFull()

{

return false;

}

template<class DT>

DT Queue<DT>::Get()

{

DT a;

if (!IsEmpty())

{

Node<DT>\*temp;

a = front->getData();

temp = front;

front = front->GetNext();

if (front == 0)

{

rear = 0;

}

list->Delete(temp);

}

return a;

}

#include <iostream>

#include<cstring>

#include<string>

#include"Header.h"

using namespace std;

bool isPalindrome(char\* word)

{

Queue<char> \*q = new Queue<char>();

for (int i = 0; i < strlen(word); i++)

{

q->Put(word[i]);

}

int count = 0;

char\*wrd = new char[strlen(word)];

int x = strlen(word) - 1;

while (!q->IsEmpty())

{

if (word[x--] == q->Get())

{

count++;

}

}

if (count == strlen(word))

{

return true;

}

else

{

return false;

}

delete[]wrd;

}

int main()

{

char \*palindrome=new char[100];

cout << "Enter your word : ";

cin >> palindrome;

bool b = isPalindrome(palindrome);

if (b == true)

{

cout << "It's a palindrome" << endl;

}

else

{

cout << "Not a palindrome" << endl;

}

system("pause");

return 0;

}