一、问题分析

- 处理的数据对象 字符串。
- 实现的功能 判断一个字符串是否是回文的，即判断一个字符串是否是首尾对称的。
- 处理后结果的显示 分两种情况，如果一个字符串是回文的就输出 yes ， 不是回文的就输出 no
- 样例举例求解结果 例如输入 sdsfegrhglp 这个字符串，现在对它进行判断。 由题目定义可知，回文要首尾字符相等，拿出这个字符串的首尾 s 和 p 进行判断，发现它们不相等就输出 no 再例如输入 helloolleh，拿出首 ‘h’ 和尾巴‘h’发现它们相等，在拿出头的下一位和尾巴的前一位，重复这个判断过程，直至首尾碰到一起或者首尾字符不等，不等就输出 no，重复到结束就输出 yes 根据判断输入 helloolleh，输出 yes

二、数据结构和算法设计

抽象数据类型的设计

```
/*
  *本数据结构是链表的 ADT，只有声明，没有实现，之后再来实现
 *类有构造函数和各种功能函数，来实现列表的插入，删除，遍历，
 *移动元素的操作
 *@author liuqi
*/
template <typename E> class List //List ADT
{
private:
     void operator =(const List& ){}        //protect assignment
     List(const List& ){}            //protect copy destructor
public:
     List(){;}   //default constructor
     virtual ~List() {}//Base destructor

     //Clear constens from the list , to make it empty
     virtual void clear ()= 0;

     //insert an element at the current location .
     //item : The element to be inserted
     virtual void insert (const E& it) = 0 ;
```

```
    //append an element at the end of the list.
    //item : The element to be appended
    virtual void append (const E& it) = 0 ;

    //Remove and return the curent element
    //Retrurn : the element that was removed
    virtual E remove() = 0;

    //Set the current position to the start of the list .
    virtual void moveToStart () = 0;

    //Set the curent posion to the end of the list .
    virtual void moveToEnd () = 0;

   //Move the current position one step left . No change
    //if already at the beginning
    virtual void prev ()   = 0;

    //Move the current posion one step right . No change
    //if already at the end
    virtual void next () = 0;

    //Return : the number of elements in the list
    virtual int length () const = 0;

   //Return : The position of the current element
    virtual int currPos () const = 0 ;

    //Set current posion
    //pos : The position to make current
    virtual void moveToPos (int Pos) = 0 ;

    //Return : The current element
    virtual const E& getValue() const = 0;
};
```

物理数据对象设计

先得定义一个结点类，来存储数据，进而来实现 ADT 规定的功能

```
/*
  *此头文件用于 Link 结点的实现，附有两个构造函数
 *
  *
template<typename E>
class Link
{
     public:
          E element;     //存放数据
         Link<E>* next; //指针域
         Link(const E& elemval , Link* nextval = NULL)
          {
               element = elemval;
               next = nextval;
          }
          Link(Link* nextval = NULL)
          {
            next = nextval;
          }

};
```

现在是 LList 的实现

```
/*
  *私有变量由三个结点
 *curr  指向当前结点的前面一个结点
 *head  标示链表的头
 *tail  标示链表的尾巴
 *其他函数的功能都在 List ADT  里面有说明
 */
#include<iostream>
#include "Link.h"
#include"assert.h"
#include "List.h"
using namespace std;
template<typename E>class LList : public List<E>
{
     private:
         Link<E>* head;
         Link<E>* tail;
          Link<E>* curr;
```

```
    int cnt;

    void init()
    {
        curr = tail = head = new Link<E>;
        cnt = 0;
    }

    void removeall()
    {
        while(head != NULL)
        {
            curr = head;
            head = head -> next;
            delete curr;
        }
    }

public:
    LList(int size = 100)
    {
        init();
    }
    ~LList(){removeall();}
    void clear(){removeall();init();}

    void insert(const E& it)
    {
        curr -> next = new Link<E>(it ,curr->next);
        if(tail == curr)tail = curr->next;
        cnt++;
    }

    void append(const E& it)
    {
        tail = tail -> next = new Link<E>(it , NULL);
        cnt++;
    }

    E remove()
    {
        Assert(curr->next != NULL , "No element");
        E it= curr -> next ->element;
        Link<E>* tmp = curr -> next;
```

```
        curr -> next = tmp -> next;
        if(tail == curr -> next) tail = curr;
        cnt --;
        delete tmp;
        return it;
    }

    void moveToStart()
    {
        curr = head;
    }

    void moveToEnd()
    {
        curr = tail;
    }

    void prev()
    {
        if( curr == head ) return ;
        Link<E>* tmp = head;
        while(tmp->next != curr)tmp = tmp -> next;
        curr = tmp;
    }

    void next()
    {
        Assert(curr != tail,"No next element");
        curr = curr -> next;
    }

    int length() const{
        return cnt;
    }

    int currPos()const{
        Link<E>* tmp = head;
        int i;
        for(i =   0; curr != tmp ; i++)
        {
            tmp = tmp -> next;
        }
        return i;
    }
```

```
        void moveToPos(int Pos)
        {
            Assert(Pos>=0 && Pos <=cnt , "Position out of range");
            curr = head;
            for(int i = 0; i < Pos ; i++)
            curr = curr -> next;
        }
         const E & getValue()const{
            Assert(curr->next != NULL,"No value");
            return curr->next->element;
        }
};
```

算法思想的设计

- 此题目的数据用 char 存储，最开始可以用 string 读入数据然后依次调用函数 **insert** 插入数据到 list 中。
- 关键之处在于判断一个字符串是否是回文串

```
我们可以设置哨兵 i 和 j 来标定将要进行比较的字符的位置，然后依次把链表里对应 i 和 j 位置的元素取出来，进行比较，如果相等则 i 向后移，j 向前移，不相等就说明不是回文串。
判断终止条件是 i>=j，即全部字符已经判断过。

我们可以把以上的描述封装成一个函数，如果匹配成功返回 true，失败则返回 false
```

关键功能的算法步骤

关键步骤在于我们的判断是否是回文串的函数

```
哨兵并且赋初始值  int i = 0 , int j = list.length() - 1 //注意是 length（） -
1                                                    //不是 length
 进行判断
while( i < j) //终止条件是  i >= j
   if(list[i] != list[j]) return false
   else
   i++ ; j --; //i 后移 ，  j 前移
```

算法性能分析

该算法计算主要消耗在

1.根据指针把 curr 移动到对应位置上 O(n)的复杂度

2.while 循环来判断字符串是否是回文串 循环 n 次

所以**时间复杂度为** O（n^2）