```
////////////////////////////////////////////////////////////////////////////
//////////////////////
//   线性链表类的实现
//   Author：Melissa M.CAO
//   Belong：Section of software theory，School of Computer Engineering & Science，
Shanghai University
//   Version： 1.0
////////////////////////////////////////////////////////////////////////////
//////////////////////

#include "StdAfx.h"
#include "LinkList.h"

////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*************************************************************************
*****************************
//以下部分为线性链表结点类的最基本的操作
////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*************************************************************************
*****************************

////////////////////////////////////////////////////////////////////////////
//////////////////////
//     构造函数
////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>node<type>::node(node<type>* pnext)
{
      next = pnext;
}

template<class type>node<type>::node(const type &item,node<type>* pnext)
{
    data = item;
    next = pnext;
}

////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*************************************************************************
*****************************
//以下部分为线性链表类的最基本的操作
```

```
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*****************************************************************************
*****************************

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//     构造函数，空表
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type>LinkList<type>::LinkList( )
{
    head = pcurrent = new node<type>( );
    head->next = NULL;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//     构造函数，利用数组构造链表
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> LinkList<type> :: LinkList(type a[], int n)
{
    int i;
    head = pcurrent = new node <type> ( );
    node<type> *p = head;

    for (i = 0; i < n; i++) {
        p->next = new node<type>(a[i]);
        p = p->next;
    }
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//     析构函数，空表
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type>LinkList<type>::~LinkList( )
{
      Clear( );
      delete head;
}
```

```
//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     求链表长度函数
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>int LinkList<type>::GetLength( )const
{
    node<type>* p = head->next;
    int len = 0;
    while(p != NULL)
    {
        len++;
        p = p->next;
    }
    return len;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     取当前元素值
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>type LinkList<type>::GetCurrent( )const
{
    if (pcurrent == head|| pcurrent == NULL)
    {
        cout << "未取到数据值" << endl;
        exit(1);
    }
    return pcurrent->data;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表定位函数
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>node<type>* LinkList<type>:: Locate(type &x)
{
    pcurrent = head->next;
    while(pcurrent != NULL)
    {
        if( pcurrent->data == x)
            break;
```

```
        else
            pcurrent = pcurrent->next;
    }

    return pcurrent;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表插入函数，在 pcurrent 前插入 x
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> void LinkList<type>::InsertBefor(const type &x)
{
    if (pcurrent == NULL) {
        cout << "您要插入的位置为空，无法插入" << endl;
        return;
    }
    if (head->next == NULL)
    {
        node<type>* newnode = new node<type>(x,NULL);
        head->next = pcurrent = newnode;
    }
    else
    {
        node<type>* newnode = new node<type>(x,pcurrent);
        node<type>* p = head;
        while(p->next != pcurrent && p != NULL)
            p = p->next;
        if (p == NULL)
        {
            cout <<"程序出错了，当前指针所指位置居然不在链表中，无法插入" <<endl;
            return;
        }
        p->next = newnode;
        pcurrent = newnode;
    }
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表插入函数，在 pcurrent 后插入 x
//////////////////////////////////////////////////////////////////////////////
//////////////////////
```

```
template<class type>void LinkList<type>::InsertAfter(const type &x)
{
    if (pcurrent == NULL) {
        cout << "您要插入的位置为空，无法插入" << endl;
        return;
    }
    if(head->next == NULL)
    {
        node<type>* newnode = new node<type>(x,NULL);
        head->next = pcurrent = newnode;
    }
    else
    {
        node<type>* newnode = new node<type>(x, pcurrent->next);
        pcurrent->next = newnode;
        pcurrent = newnode;
    }
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     删除链表当前元素函数
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> type LinkList<type>::DeleteCurrent( )
{
    if (pcurrent == head|| pcurrent == NULL)
    {
        cout << "链表已空!" << endl;
        exit(1);
    }
    node<type>* p = head;
    while(p->next != pcurrent && p != NULL)
        p = p->next;
    if (p == NULL)
    {
        cout <<"程序出错了，当前指针所指位置居然不在链表中，无法删除入" << endl;
        return NULL;
    }
    p->next = pcurrent->next;
    type data1 = pcurrent->data;
    delete pcurrent;
    pcurrent = p->next;
    return data1;
```

```
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     判断链表是否为空
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> int LinkList<type>::IsEmpty( ) const
{
    if (head->next == NULL)
        return 1;
    else
        return 0;
}
//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     清空链表
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>void LinkList<type>::Clear( )
{
    node<type> *p,*q;
    p = head->next;
    while(p != NULL)
    {
        q = p;
        p = p->next;
        /*（1）保持链表头指针最终指向空，但存在效率问题，修改为（2）
        head->next = p; */
        delete q;
    }
    /*（2）保持链表头指针最终指向空，提高效率 */
    head->next = p;

    pcurrent = head;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     将链表当前元素指针指向其下一个结点
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> node<type>* LinkList<type>::CurrentToNext( )
{
```

```
    if (pcurrent != NULL)
        pcurrent = pcurrent->next;
    return pcurrent;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//    将当前指针置为链表中第 i 个元素
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>node<type>* LinkList<type>::ResetCurrent(int i)
{
    if (head == NULL|| i < 0)
          return NULL;
    if (i == 0)
    {
          pcurrent = head;
          return head;
    }
    node<type>* p = head;
    int k = 0;
    while (p != NULL && k < i)
    {
          p = p->next;
          k++;
    }
    pcurrent = p;
    return p;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//    打印当前链表中元素
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> void LinkList<type>::PrintList( )
{
    node<type>* p = head;
    if ( p->next == NULL)
    {
        cout << "已经是空链表，没有任何数据！" << endl;
        return;
    }
    cout << "查链表中各元素分别是：";
```

```
    while (p->next != NULL)
    {
        p = p->next;
        cout << p->data << "; ";
    }
    cout << endl;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//    当前元素置于第 i 个结点，主要用于多项式
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class type> node <type> * LinkList<type>::Reset( int i )
 {
     if (head == NULL || i< 0)
        return NULL;
     if(i == 0 )
     {
         pcurrent = head;
         return head ;
     }
     node<type> *p = head;
     int k = 0;
     while( p != NULL && k < i )
     {
         p = p->next;
         k++;
     }
     pcurrent = p;
     return p;
 }

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//    当前元素指向下一个结点，主要用于多项式
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class type> node <type> * LinkList<type>::Next( )
{
    if(pcurrent != NULL)
        pcurrent = pcurrent->next;
    return pcurrent;
}
```

```
//////////////////////////////////////////////////////////////////////////////
//////////////////////
//    判断是否达到链表的最后
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> int LinkList <type>::EndofList() const
{
    if ( pcurrent == NULL)
        return 1;
    else
        return 0;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//*****************************************************************************
*****************************
//以下部分为线性链表类的“基本”操作的扩展
//////////////////////////////////////////////////////////////////////////////
//////////////////////
//*****************************************************************************
*****************************

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表插入函数，在 y 后插入 x
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>void LinkList<type>::InsertAfter(const type & x, const type &y)
{
    pcurrent = head->next;
    while (pcurrent != NULL && pcurrent->data != y)
        pcurrent = pcurrent->next;
    if (pcurrent == NULL) {
        cout << "您要在后插入 x，但 y 不存在或者是空表，无法插入" << endl;
        return;
    }
    node<type>* newnode = new node<type>(x, pcurrent->next);
    pcurrent->next = newnode;
    pcurrent = newnode;
}

//////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////
//     链表插入函数，在第 i 个元素后插入 x
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>void LinkList<type>::InsertAfter(const type & x, const int i)
{
    int k = 1;
    pcurrent = head->next;
    while (pcurrent != NULL && k != i)
    {
        pcurrent = pcurrent->next;
        k++;
    }
    if (pcurrent == NULL) {
        cout <<"您要在 i 后插入 x，但链表长度小于 i，或者是空表，无法插入" << endl;
        return;
    }
    node<type>* newnode = new node<type>(x, pcurrent->next);
    pcurrent->next = newnode;
    pcurrent = newnode;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//插入元素作为表中的第一个元素
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> void LinkList<type>::InsertAsFirst(const type &x)
{
    node<type>* newnode = new node<type>(x,head->next);
    head->next = newnode;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表中删除第 i 个元素
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> type LinkList<type>::Remove(const int i)
{
    type data1;
    int k = 0;
    pcurrent = head;
    while (pcurrent->next != NULL && ++k != i)
```

```
        pcurrent = pcurrent->next;

    if (pcurrent == NULL) {
        cout <<"您要删除第 i 个元素,但链表长度小于 i,或者是空表,无法删除"<< endl;
        return NULL;
    }

    node<type>* q = pcurrent->next;
    pcurrent->next = q->next;
    data1 = q->data;
    delete q;
    return data1;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表逆置
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class type>  int LinkList<type> :: Reverse ( )
{
    if (head->next == NULL) // empty list
    {
         cout << "空表, 无需逆置" << endl;
         return 1;
    }
    node <type> * p = head->next;
    node <type> * q = NULL;
    node <type> * r = NULL;
    while (p!=NULL)
    {
        r = q;
        q = p;
        p = p->next;
        q->next = r;
    }
    head->next = q;
    return 1;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     //追加元素到链尾
//////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////
template <class type>  void LinkList<type> :: Append(const type &x)
{
    node<type> *p = head;
    pcurrent = head->next;
    while (pcurrent != NULL)
    {
        p = pcurrent;
        pcurrent = pcurrent->next;
    }

    node<type>* newnode = new node<type>(x, p->next);
    p->next = newnode;
    pcurrent = newnode;
}


////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*****************************************************************************
*****************************
//以下部分为线性链序表的习题
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*****************************************************************************
*****************************
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//      通过输入的方式建立有序链表的构造函数。
//      参数：2--非递增重复；4--非递增不重复；
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class type> void LinkList<type> :: OrderMerge(int t, LinkList<type> &b)
{
    node<type> *pa, *pb, *q, *p, *r = NULL;
    pa = head->next;
    pb = b.head->next;
    if (t == 2 || t == 4)
        head->next = NULL; //结果链表初始化
    //请同学们自己完成一书上习题 13

    //不加处理，b 析构时有问题
    //b.head->next = NULL;--这一句不起作用，必须是下面的 public 函数，直接赋值可能
没用
```

```
    b.head->SetNext(NULL);
}
```