```
///////////////////////////////////////////////////////////////////////////////////////////////////////////
//   静态链表及其结点的类实现
//   Author：Melissa M.CAO
//   Belong：Section of software theory，School of Computer Engineering & Science，Shanghai University
//   Version： 1.0
///////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "StdAfx.h"
#include "StaticLinkList.h"

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

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

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

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

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

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//   静态链表的构造函数定义
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
template<class type> StaticLinkList<type>::StaticLinkList(int size):maxsize(size)
{
    //申请最大空间
    info = new StaticNode<type>[maxsize];
    if (info == NULL)
    {
        cout << "动态存储分配失败" << endl;
        exit(1);
    }

    //形成“链”关系
    for (int i = 0; i < maxsize; i++)
    {
        info[i].next = i+1;
    }
    info[maxsize-1].next = -1;  //最后一个结点

    head = 0;   //带有头结点
    avail = 1;  //可用空间
}

/*template<class type> StaticLinkList<type>::~StaticLinkList(void)
{
}*/

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//     求链表长度函数
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> int StaticLinkList<type>::GetLength( ) const
{
    int len = 0;
    int current = info[head].next;
    while (current != avail)
    {
        len++;
        current = info[current].next;
    }
    return len;
}

////////////////////////////////////////////////////////////////////////////////
/////////////////////////
//     链表定位函数
////////////////////////////////////////////////////////////////////////////////
////////////////////////////
template<class type> int StaticLinkList<type>::Locate(type &x)
{
    int pos = info[head].next;
    while (pos != avail)
    {
        if( info[pos].data == x)
            break;
        else
            pos = info[pos].next;
    }

    return pos;
}

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//     清空链表
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> void StaticLinkList<type>::Clear( )
{
    //形成“链”关系
    for (int i = 0; i < maxsize; i++)
    {
        info[i].next = i+1;
    }
    info[maxsize-1].next = -1;  //最后一个结点

    head = 0;   //带有头结点
    avail = 1;  //可用空间
}
```

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//    打印当前链表中元素
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> void StaticLinkList<type>::PrintList( )
{
    int current = info[head].next;
    cout << "目前静态链表中的链表内容为：";
    while (current != avail)
    {
        if (info[head].next  != current)
            cout << "，";
        cout << info[current].data;
        current = info[current].next;
    }

    cout << endl << "目前可用空间情况为：";
    current = avail;
    if (avail == -1)
        cout << "" << endl;
    while (current != -1)
    {
        cout << "下标为" << current << "的空间可用；";
        current = info[current].next;
    }
    cout << endl;
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////
//     //追加元素到链尾
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class type>  void StaticLinkList<type> :: Append(const type &x)
{
    if (avail == -1)
    {
        cout << "因为是静态链表，申请的空间已经用完，无法添加！" << endl;
        return;
    }
    int first = head, now = info[head].next;

    while (now != avail)
    {
        first = now;
        now = info[now].next;
    }

    info[now].data = x;         //追加元素
    avail = info[avail].next;   //修改可用表
}

////////////////////////////////////////////////////////////////////////////////////////////////////////
//     链表插入函数，flag为P，表示在第i个元素前插入x，flag为N，表示在第i个元素后插入x。
//////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class type>void StaticLinkList<type>::Insert(const type & x, const int i, char flag)
{
    if (avail == -1)
    {
        cout << "因为是静态链表，申请的空间已经用完，无法添加！" << endl;
        return;
    }

    if (i < 1)
    {
        cout << "您指定的插入位置不存在！" << endl;
        return;
    }
    int first = head, now = info[head].next, k = 1;
    while (now != avail && k != i)
    {
        first = now;
        now = info[now].next;
        k++;
    }   //找到指定的位置
    if (now == avail) { //空表或指定位置大于表的长度
        cout << "您要在第" << i << "个元素前或后插入" << x << "，但链表长度小于" << i << "，或者是空表，无法插入！" << endl;
        return;
    }
    //插入
    int last = GetLast();
    int newpos = avail;
    avail = info[avail].next;
    info[newpos].data = x; //“新申请”一个位置
    info[last].next = avail;
    if (flag == ’P’ || flag == ’p’)
    {
        info[first].next = newpos;
        info[newpos].next = now;
    }
    else
    {
        if (flag == ’N’ || flag == ’n’)
        {
            info[newpos].next = info[now].next;
            info[now].next = newpos;
        }
        else
        {
            cout << "您指定的插入方式错误，要么之前，要么之后！" << endl;
            return;
        }
    }
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////
//      链表插入函数，flag为P，表示在y前插入x，flag为N，表示在y后插入x。
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type>void StaticLinkList<type>::Insert(const type & x, const type &
y, char flag)
{
    if (avail == -1)
    {
        cout << "因为是静态链表，申请的空间已经用完，无法添加！" << endl;
        return;
    }

    int first = head, now = info[head].next;
    while (now != avail && info[now].data != y)
    {
        first = now;
        now = info[now].next;
    }   //找到指定的元素或空表或表中无指定元素
    if (now == avail) { //空表或指定位置大于表的长度
```

```
        cout << "您要在" << y << "前或后插入" << x << "，但链表中不存在y，或者是空表，无法插入！" << endl;
        return;
    }
    //插入
    int newpos = avail;
    int last = GetLast();
    avail = info[avail].next;
    info[newpos].data = x; //“新申请”一个位置
    if (flag == 'P' || flag == 'p')
    {
        info[first].next = newpos;
        info[newpos].next = now;
    }
    else
    {
        if (flag == 'N' || flag == 'n')
        {
            info[newpos].next = info[now].next;
            info[now].next = newpos;
        }
        else
        {
            cout << "您指定的插入方式错误，要么之前，要么之后！" << endl;
            return;
        }
    }
    info[last].next = avail;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////
//     链表中删除第i个元素
//////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class type> type StaticLinkList<type>::Remove(const int i)
{
    type data1;
    int k = 1, current, first;
    first = head;
    current = info[head].next;
    while (current != avail && k != i)
    {
        first = current;
        current = info[current].next;
        k++;
    }

    if (current == avail) {
        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;
    }

    //删除过程
    int last = GetLast();
    data1 = info[current].data;
    if (current != last)
    {
        info[first].next = info[current].next;
        info[current].next = avail;
        avail = current;
        //原表尾的 next 指向 avail，现在 avail 发生变化，表尾的 next 应该指向新的 avail
        info[last].next = avail;
    }
    else
    {
        info[current].next = avail;
        avail = last;
    }

    return data1;
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////
//      返回链表中最后一个元素的位置
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> int StaticLinkList<type>::GetLast()
{
    int pos = -1;
    int current = head;
    while (info[current].next != avail)
    {
        current = info[current].next;
    }

    return current;
}
```