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

#include "StdAfx.h"
#include "CirList.h"

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//无参构造函数
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> CirList<type>::CirList(void)
{
    head = new Node<type>();// 构造头结点
    head->next = head;// 空循环链表的头结点后继为头结点本身
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//析构函数
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> CirList<type> ::~CirList(void)
{
    Clear();            // 清空线性表
    delete head;        // 释放头结点所点空间
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//清除链表
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> void CirList<type>::Clear( )
{
    Node<type> *p = head->next;
    Node<type> *q;
```

```
    while ( p!= head)
    {
        q = p;
        p = p->next;
        delete q;
    }
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//求链表长度
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> int CirList<type>::Length() const
{
    int count = 0;
    Node<type> *p = head->next;
    while (p != head)
    {
        count++;
        p = p->next;
    }
    return count;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
// 操作结果：由线性表 copy 构造新线性表——复制构造函数模板
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> CirList<type>::CirList(const CirList<type> &copy)
{
    Node<type> *p = copy.head->next, *q, *h;
    head = new Node<type>();// 构造头指针
    head->next = head;// 空循环链表的头结点后继为头结点本身
    h = head;

    while (p != copy.head)
    {
        q = new Node<type>(p->data);
        h->next = q;
        h = q;
        p = p->next;
    }
```

```
    h->next = head;
}

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

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

    p->next = head;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表定位函数
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> Node<type>* CirList<type>:: Locate(type &x)
{
    Node<type> *pcurrent = head->next;
    while(pcurrent != head)
    {
        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 CirList<type>::InsertBefor(Node<type> *pcurrent, 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 == head)
    {
        Node<type>* newNode = new Node<type>(x,head);
        head->next = newNode;
    }
    else
    {
        Node<type>* newNode = new Node<type>(x,pcurrent);
        Node<type>* p = head;
        while(p->next != pcurrent && p->next != head)
            p = p->next;
        if (p->next == head)
        {
            cout << "理论上不允许在后结点前面插入，无法插入！" << endl;
            return;
        }
        p->next = newNode;
    }
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表插入函数，在 pcurrent 后插入 x
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type>void CirList<type>::InsertAfter(Node<type> *pcurrent, 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 == head)
    {
```

```
        Node<type>* newNode = new Node<type>(x,head);
        head->next = newNode;
    }
    else
    {
        Node<type>* newNode = new Node<type>(x, pcurrent->next);
        pcurrent->next = newNode;
    }
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//    打印当前链表中元素
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type> void CirList<type>::PrintList( )
{
    Node<type>* p = head;
    if ( p->next == head)
    {
        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 != head)
    {
        p = p->next;
        cout << p->data << "; ";
    }
    cout << endl;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     //追加元素到链尾
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class type>  void CirList<type> :: Append(const type &x)
{
    Node<type> *p = head;
    Node<type> *pcurrent = head->next;
    while (pcurrent != head)
    {
        p = pcurrent;
```

```
        pcurrent = pcurrent->next;
    }

    Node<type>* newNode = new Node<type>(x, p->next);
    p->next = newNode;
}

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

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表插入函数，在 y 后插入 x
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>void CirList<type>::InsertAfter(const type & x, const type &y)
{
    Node<type> *pcurrent = head->next;
    while (pcurrent != head && pcurrent->data != y)
        pcurrent = pcurrent->next;
    if (pcurrent == head) {
        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;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     链表插入函数，在第 i 个元素后插入 x
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template<class type>void CirList<type>::InsertAfter(const type & x, const int i)
{
    int k = 1;
```

```
    Node<type> *pcurrent = head->next;
    while (pcurrent != head && k != i)
    {
        pcurrent = pcurrent->next;
        k++;
    }
    if (pcurrent == head) {
        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;
}

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

    if ((pcurrent == head) && (pcurrent->next == head)) {
        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;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//返回第 i 个元素的地址
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
```

```
template<class type> Node<type>* CirList<type>::GetElement(int i)
{
    Node<type> *p = head->next;
    int k = 0;
    while ((p != head) && ++k != i)
        p = p->next;
    if (p != head)
        return p;
    else
        return NULL;
}

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//操作结果：非空有序递增表，t 指针初始时等于 t，查找策略：K 与 t 的关键字相比，若相等，查找成功；若小于 t 的关键字，
//          从 h 所指的结点开始查找；若大于 t，从 t 的后继开始找。不成功的条件是查找“一圈”或找到第一大于 K 值的结点
//          为利用已定义的单循环链表，h 即为 head->next，记录 t；
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> Node<type>* CirList<type>::SearchAccordingToCurrentT(type k,
Node<type>*t)
{
    Node<type> *p, *q;

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

    cout << "当前 t 所指向的结点的值为：" << t->GetData() << endl;
//书上第八章习题，请同学们自己完成
}

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//操作结果：n 个人围成一个圆圈，首先第 1 个人从 1 开始一个人一个人顺时针报数，报到第 m 个人，令其出列。
//然后再从下一个人开始，从 1 顺时针报数报到第 m 个人，再令其出列，…，如此下去，直到圆圈中只剩一个人为止。
//此人即为优胜者。即约瑟夫环问题。
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> void Josephus(int n, int m)
```

```
{
    CirList<int> la;            // 定义空循环链表
    int position = 0, len;      // 报数到的人在链表中序号
    type out, winner;
    int i, j, k;
    for (k = 1; k <= n; k++)
        la.Append(k);// 建立数据域为1,2,..,n的循环链表
    //请同学们自己完成
    winner = (la.GetElement(1))->GetData();          // 剩下的一个人为优胜者
    cout << endl << "优胜者:" << winner << endl;
}

////////////////////////////////////////////////////////////////////////////
//////////////////////////
//操作结果：n个人围成一个圆圈，首先第k(1<=k<=n)个人从1开始一个人一个人顺时针报数，报到第m(第k个人的密码)个人，
//令其出列。然后再以下一个人的密码为m，从1（或从该人）顺时针报数报到第m个人，再令其出列，…，如此下去，
//直到圆圈中只剩一个人为止。此人即为优胜者。即带密码的约瑟夫环问题。
//Test: 8 -1 pwd: 3 10 7 1 4 8 4 5 -1  K: 7
//以下为解法一：另外建立一个同步的密码循环连链表；可以解法二：循环链表中的数据为结构，分别包含号码和密码
////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class type> void Josephus(int n, int begin,  int a[])

{
    CirList<int> la, lb;            // 定义空循环链表
    int position, len;      // 报数到的人在链表中序号
    type out, winner;
    int i, j, k, m;
    //请同学们自己完成
    winner = (la.GetElement(1))->GetData();          // 剩下的一个人为优胜者
    cout << endl << "优胜者:" << winner << endl;
}
```