**中南大学**

**数据结构实验报告**

题 目 线性表

学生姓名 谭哲文

学 号 8202191123

指导教师 余腊生

学 院 计算机学院

专业班级 计科2105

完成时间 2023年5月6日

指导教师评定： 签 名：

单向链表操作的实现

1、需求分析

* 用尾插法建立带头结点的单向链表。
* 对已建立的单向链表实现插入、删除、查询等基本操作。

（1）输入的形式：不同数字代表不同功能

（2）输出的形式：

构造/析构:显示构造/析构成功

判空/满：显示是否为空/满的结果

获取表长：显示当前表长

插入/删除：显示插入是否成功及当前链表

（3）程序所能达到的功能

完成单链表的初始化

检查链表是否为空

建立单链表

求链表长度

遍历链表

按位置插入元素

按位置删除元素

销毁链表。

2、概要设计

（1）链表的抽象数据类型

ADT List

{

    Data object：D＝{ ai | ai ∈ElemType, i=1,2,...,n, n≥0 }

    Relation ：R1＝{ <ai-1 ,ai >|ai-1 ,ai∈D, i=2,...,n }

    Operations ：

InitList( &L )

//初始化一个链表

DestroyList( &L )

//初始条件：链表存在

//操作结果：链表销毁

ListEmpty( L )

//判断链表为空

ListLength( L )

//求链表长度

ListTraverse(L, visit( ))

//遍历链表

ListInsert( &L, i, e )

//往链表里插入元素

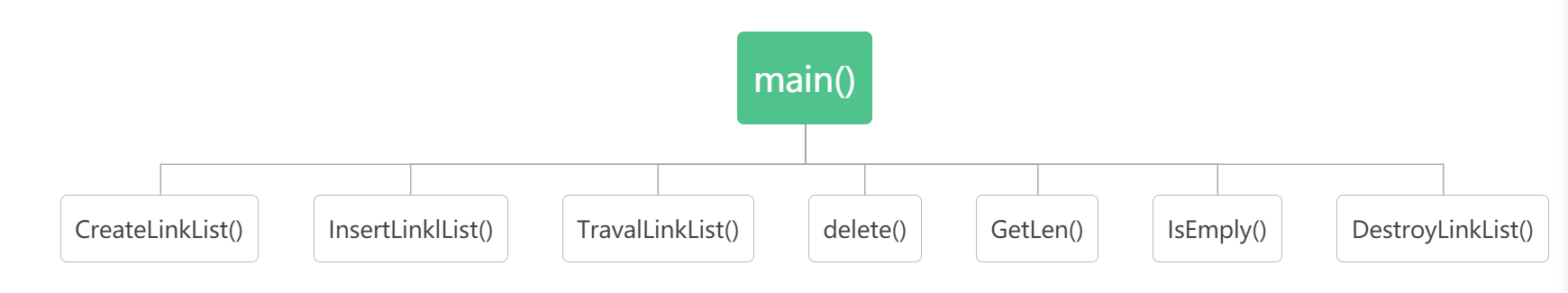
ListDelete(&L, i, &e）

//初始条件：链表存在

//从链表里删除元素

} ADT List

（2）主程序的流程



3、详细设计

（1）有头结点，尾插建表

int LinkList::CreateLinkList(int n)

{

    if (n < 0) {

        printf("error\n");

        return -1;

    }

    Node\* ptemp = NULL;

    Node\* pnew = NULL;

    this->size = n;

    ptemp = this->head;

    for (int i = 0; i < n; i++)

    {

        pnew = new Node;

        pnew->next = NULL;

        cout << "输入第" << i + 1 << "个节点值" << endl;

        cin >> pnew->data;

        ptemp->next = pnew;

        ptemp = pnew;

    }

    cout << "创建完成" << endl;

    return 0;

}

（2）从第一结点开始逐个销毁从而销毁链表

int LinkList::DestroyLinkList()

{

    Node\* ptemp;

    if (this->head == NULL)

    {

        cout << "链表原本就为空" << endl;

        return -1;

    }

    while (this->head)

    {

        ptemp = head->next;

        free(head);

        head = ptemp;

    }

    cout << "销毁链表完成" << endl;

    return 0;

}

（3）从头到尾遍历结点并显示

int LinkList::TravalLinkList()

{

    Node\* ptemp = this->head->next;

    if (this->head == NULL)

    {

        cout << "链表为空" << endl;

        return -1;

    }

    while (ptemp)

    {

        cout << ptemp->data << "->";

        ptemp = ptemp->next;

    }

    cout << "NULL" << endl;

    return 0;

}

（4）在指定位置插入一个元素

int LinkList::InsertLinklList(Node\* data, int pos)

{

        Node\* ptemp;

        if (this->head == NULL)

        {

            cout << "链表为空" << endl;

            return -1;

        }

        if (data == NULL)

        {

            cout << "插入节点为空" << endl;

            return -1;

        }

        /\*头插\*/

        if (pos < 2)

        {

            Node\* pnew = new Node;

            pnew->data = data->data;

            pnew->next = this->head->next;

            this->head->next = pnew;

            this->size++;

            cout << "插入成功" << endl;

            return 0;

        }

        /\*尾插\*/

        if (pos == this->size)

        {

            ptemp = this->head;

            while (ptemp->next != NULL) {

                ptemp = ptemp->next;

            }

            Node\* pnew = new Node;

            pnew->data = data->data;

            pnew->next = NULL;

            ptemp->next = pnew;

            this->size++;

            cout << "插入成功" << endl;

            return 0;

        }

        /\*中间后插\*/

        if(pos>=2&&pos<=this->size)  {

            ptemp = this->head;

            for (int i = 1; i < pos; i++) {

                ptemp = ptemp->next;

            }

            Node\* pnew = new Node;

            pnew->data = data->data;

            pnew->next = ptemp->next;

            ptemp->next = pnew;

            this->size++;

            cout << "插入成功" << endl;

            return 0;

        }

        else

        {

            cout<<"插入失败，输入数据不正确"<<endl;

        }

}

（5）求单链表的长度

int LinkList::GetLen()

{

    return this->size;

}

（6）判断链表是否为空

bool LinkList::IsEmply()

{

    if (this->head == NULL) {

        return true;

    }

    else {

        return false;

    }

}

4、调试分析

（1）调试过程中遇到的主要问题及解决过程：

主要是一些情况的考虑不周全，比如开始涉及的时候没有考虑其他功能要在初始化的基础上，导致一系列的问题。

其中最为突出的是非数字输入导致程序陷入死循环，查阅了很多资料找到了

cin.clear();

                cin.ignore();

两个函数，再利用goto beging，来返回菜单，可以解决此问题。

（2）分析程序代码的质量

正确性：实现大部分输入数据都能有理想的输出

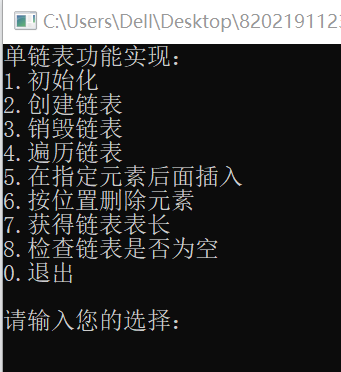
健壮性：在一定的数据输入范围内，该程序能较好地实现链表的操作。

5、使用说明

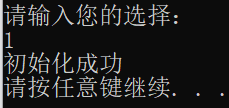
单链表模板类的全部实现内容均包含在一个独立的C++头文件SingleLinkList.h中，要在程序中使用这个类只需包含该头文件即可。这个类实现了创建、插入、删除、查找、遍历等操作。按值查找和删除时都只能对从头结点开始第一个值相同的元素进行操作。清空链表后需要重新初始化。

6、测试程序的运行结果

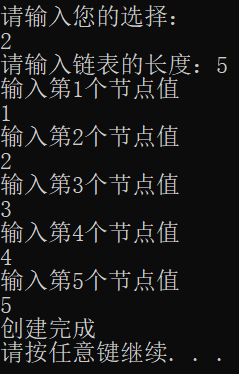
菜单界面



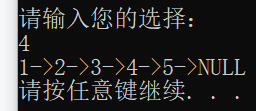
（1）初始化：选择1，输出初始化成功



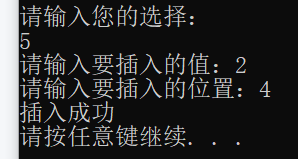
1. 创建单链表：选择2，输入5，然后依次输入1 2 3 4 5，输出创建成功；



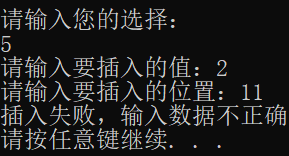
（3）遍历链表：选择4，显示链表



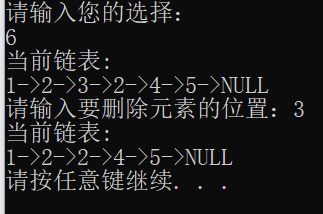
（4）在指定元素后面插入：选择5，依次输入2，4，输出插入成功



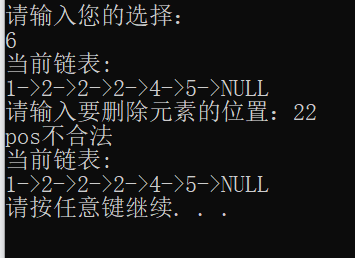
选择5，依次输入2，11，输出插入失败，输入数据不正确



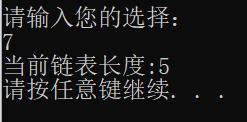
（5）按位置删除元素：选择6，输出显示全表，输入3，输出删除指定元素后的表



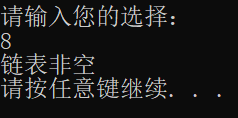
输入22，输出pos不合法



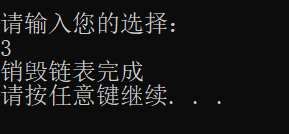
（7）获得链表表长：选择7，输出链表表长



（8）检查链表是否为空：选择8，输出链表不为空



（9）销毁链表：选择3输出链表销毁完成



7、心得体会

1. 通过对于链表的撰写，一方面加深了对于该数据结构的理解和应用，一方面在进行代码编写的过程中学会了写标注，可以在程序出现问题时进行查找漏洞，此次实验中在程序出现问题后，通过自己所写的标注，很快便可以进行修正，就可以很好地降低后期检查的繁琐程度。

城市链表

1、需求分析

* 将城市若干信息置于结点中，组成城市链表，其基本操作包括了显示、插入、删除、查找、更新、范围查找。

（1）输入的形式：不同数字代表不同功能

（2）输出的形式：

添加城市：显示添加成功/失败

删除城市：显示删除成功/失败

修改城市：显示修改成功/失败

查询城市：显示所有城市

坐标查询：显示所有在指定范围里的城市

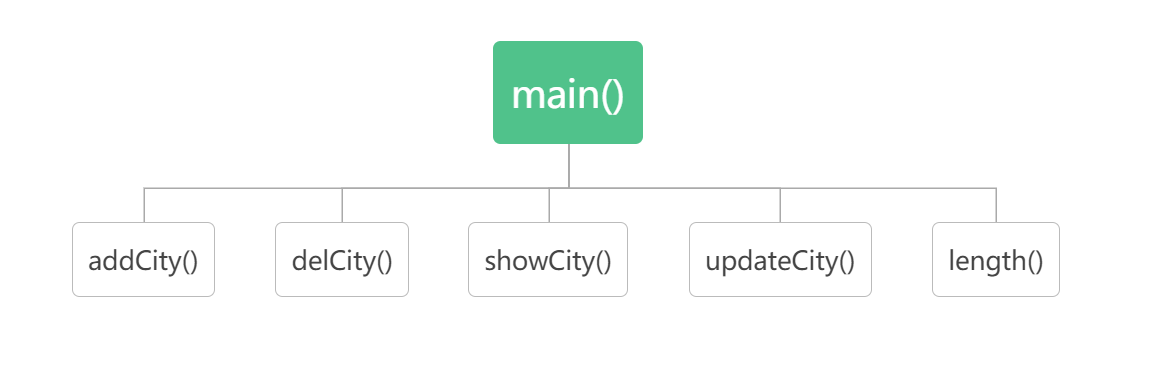
（3）程序所能达到的功能

完成城市的导入，插入城市数据，删除城市数据，查找城市信息，更新城市信息，查找指定范围内的城市。

2、概要设计

（1）链表的抽象数据类型

（2）主程序的流程



3、详细设计

（1）添加城市

void CityList::addCity()

{

    City \*newCity = new City();

    City \*temp = this->head;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    cout << "----------- 添加一个城市 ----------" << endl;

    cout << "请输入城市名：";

    cin >> newCity->name;

    cout << "请输入面积（平方千米）：";

    cin >> newCity->area;

    cout << "请输入人口数(万人)：";

    cin >> newCity->peopleCount;

    cout << "请输入城市X坐标:";

    cin >> newCity->x;

    cout << "请输入城市y坐标:";

    cin >> newCity->y;

    cout << "请输入城市等级：[1-5]:";

    cin >> newCity->top;

    while (newCity->top > 5 || newCity->top < 1)

    {

        cin.clear();

        cin.ignore();

        cout << "城市等级输入错误,请重新输入：";

        cin >> newCity->top;

    }

    cout << endl;

    cout << "请输入城市特色：";

    cin >> newCity->feature;

    newCity->next = NULL;

    temp->next = newCity;

    cout << "添加成功!" << endl;

}

（2）遍历城市

int CityList::showCity()

{

    City\* temp = head->next;

    if (temp == NULL) {

        cout << "城市为空，无法输出!" << endl;

        return -1;

    }

    cout << "---------- 正在生成城市清单 ---------" << endl;

    while (temp != NULL) {

        cout << "城市名：" << temp->name << endl;

        cout << "城市人口（万人）：" << temp->peopleCount << endl;

        cout << "城市面积（平方千米）：" << temp->area << endl;

        cout << "城市等级：" << temp->top << endl;

        cout << "城市X坐标：" << temp->x << endl;

        cout << "城市Y坐标：" << temp->y << endl;

        cout << "城市特色：" << temp->feature << endl;

        cout << "------------------------------" << endl;

        temp = temp->next;

    }

    return 0;

}

（3）删除城市

int CityList::delCity()

{

    if (this->head->next == NULL)

    {

        cout << "城市链表为空，请先添加城市。。。" << endl;

        return -1;

    }

    cout << "------------------------------" << endl;

    string cityName;

    char choose;

    cout << "请输入你要删除的城市名：";

    cin >> cityName;

    // 找到要删除城市的前一个城市

    City\* ctemp = head;

    while (ctemp->next->name != cityName) {

        ctemp = ctemp->next;

        if (ctemp->next == NULL) {

            cout << "没有你想要删除的城市，程序即将返回主页面" << endl;

            return -1;

        }

    }

    City\* temp = head; // 删除城市的前一个城市

    City\* del = head->next; // 删除城市

    cout << "你将删除的城市信息：" << endl;

    cout << "城市名：" << del->name << endl;

    cout << "城市人口（万人）：" << del->peopleCount << endl;

    cout << "城市面积（平方千米）：" << del->area << endl;

    cout << "城市等级：" << del->top << endl;

    cout << "城市X坐标：" << del->x << endl;

    cout << "城市Y坐标：" << del->y << endl;

    cout << "城市特色：" << del->feature << endl;

    cout << "------------------------------" << endl;

    cout << "你确定要删除[Y-N]：";

    cin >> choose;

    if (choose == 'Y') {

        temp->next = del->next;

        free(del);

        cout << "删除成功" << endl;

        return 0;

    }

    else {

        cout << "删除取消" << endl;

        return-1;

    }

}

（4）修改城市

int CityList::updateCity()

{

    if (this->head->next== NULL)

    {

        cout << "城市链表为空，请先添加城市。。。" << endl;

        return -1;

    }

    cout << "<--- 这里是修改城市信息服务! --->" << endl;

    string name;

    cout << "请输入你要修改城市的名字：";

    cin >> name;

    City\* ctemp = head;

    while (ctemp->name != name)

    {

        ctemp = ctemp->next;

        if (ctemp == NULL)

        {

            cout << "没有此城市，将退出此功能。。。" << endl;

            return -1;

        }

    }

    City\* temp = head;

    cout << "----------- 修改城市信息 ----------" << endl;

    cout << "请输入城市名：";

    cin >> temp->name;

    cout << "请输入面积（平方千米）：";

    cin >> temp->area;

    cout << "请输入人口数(万人)：";

    cin >> temp->peopleCount;

    cout << "请输入城市X坐标:";

    cin >> temp->x;

    cout << "请输入城市y坐标:";

    cin >> temp->y;

    cout << "请输入城市等级：[1-5]:";

    cin >> temp->top;

    while (temp->top > 5 || temp->top < 1)

    {

        cout << "城市等级输入错误,请重新输入：";

        cin >> temp->top;

    }

    cout << "请输入城市特色：";

    cin >> temp->feature;

    cout << "\*\*\*\*\*\*\*\* 修改信息成功 \*\*\*\*\*\*\*\*" << endl;

    return 0;

}

（5）求指定范围所有城市

void CityList::length()

{

    if (this->head->next == NULL)

    {

        cout << "城市链表为空，请先添加城市。。。" << endl;

        return;

    }

    City\* cityList = head->next;

    cout << "\*\*\*\*\*\*\*\*\* 搜索范围之内的城市 \*\*\*\*\*\*\*\*\*" << endl;

    int x, y, D;

    cout << "请输入坐标：" << endl;

    cout << "x =";

    cin >> x;

    cout << "y =";

    cin >> y;

    cout << "请输入距离 D：";

    cin >> D;

    cout << "正在为你返回符合城市：" << endl;

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

    int ads = 0;

    while (cityList != NULL)

    {

        if (((cityList->x -x) \* (cityList->x - x)) + ((cityList->y - y) \* (cityList->y - y)) <= D \* D)

        {

            ads++;

            cout << "城市名：" << cityList->name << endl;

            cout << "城市人口（万人）：" << cityList->peopleCount << endl;

            cout << "城市面积（平方千米）：" << cityList->area << endl;

            cout << "城市等级：" << cityList->top << endl;

            cout << "城市X坐标：" << cityList->x << endl;

            cout << "城市Y坐标：" << cityList->y << endl;

            cout << "城市特色：" << cityList->feature << endl;

            cout << "------------------------------" << endl;

        }

        cityList = cityList->next;

    }

    if (ads == 0)

    {

        cout << "没有符合的城市" << endl;

    }

}

4、调试分析

（1）调试过程中遇到的主要问题及解决过程：

过程中遇到修改城市时，第一次修改输入错误后，第二系统报错的问题，在删除城市的判断之后也出现同样情况，经过对比二者的异同分析，才发现二者都经历了循环过程，指针的位置都发生了改变，在尝试将指针地址复制给辅助指针之后，成功解决相应问题。

（2）分析程序代码的质量

正确性：实现大部分输入数据都能有理想的输出

健壮性：在一定的数据输入范围内，该程序能较好地实现链表的操作。

5、使用说明

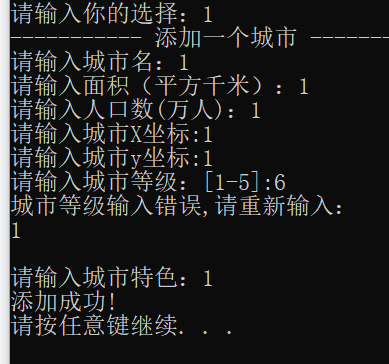
城市类及部分操作函数的全部实现内容均包含在一个独立的C++头文件CitiesManagement.h中，要在程序中使用这个类只需包含该头文件即可。这个头文件实现了城市类的更新数据、导入数据、计算距离、统计城市等操作。按值查找和删除时都只能对从头结点开始第一个匹配的元素进行操作。

6、测试程序的运行结果

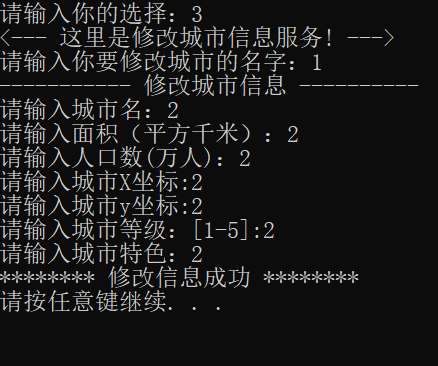
菜单



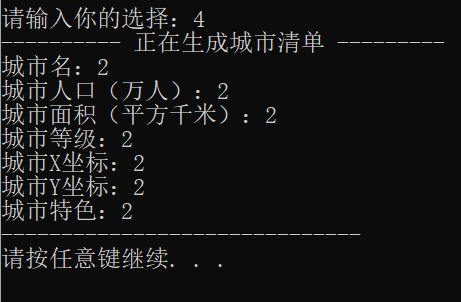
（1）添加城市：选择1，依次输入城市信息，输出添加成功



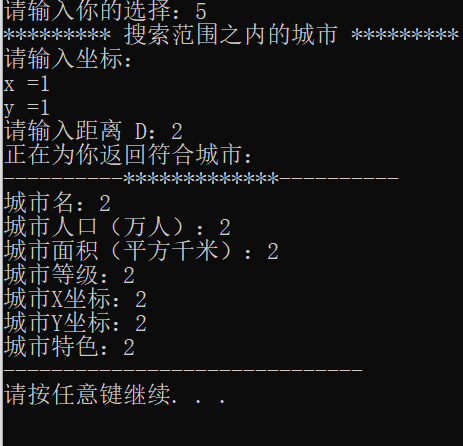
（2）修改城市：选择3，依次输入修改的城市信息，输出修改城市成功

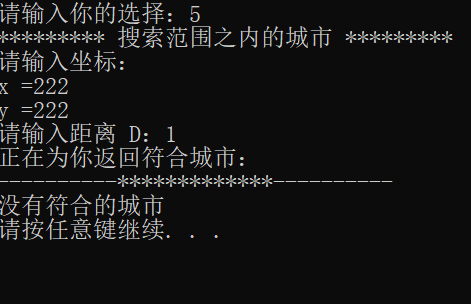


（3）查询城市：选择4，显示全表



1. 坐标查询：选择5，依次输入坐标和范围，显示满足条件的城市





7、心得体会

通过设计城市链表，对于链表进行了实例化的运用，切实感受到了链表的实用性，除了具体的一些细节需要修改之外，基本的增删改查都没有很大的变化，可以说掌握了一种数据结构，就可以掌握非常多实例的运用。

约瑟夫环

1、需求分析

* 本问题是求解一个有终止的递推问题，所以特定情况一定有迭代解法和递归解法以及数学通解。但是由于每次循环终止参数不固定，最简单直接的解法就是使用循环链表。

（1）输入的形式：输入人数、个人密码和初始密码

（2）输出的形式：依次输出退出游戏的人，最后输出胜利的人

（3）程序所能达到的功能：创建、显示循环链表，可视化迭代计算的功能。

2、概要设计

（1）链表的抽象数据类型

ADT CircleList

{

    Data object：D＝{ ai | ai ∈ElemType, i=1,2,...,n, n≥0 }

    Relation ：R1＝{ <ai-1 ,ai >|ai-1 ,ai∈D, i=2,...,n }

    Operations ：

    InitList( &L )

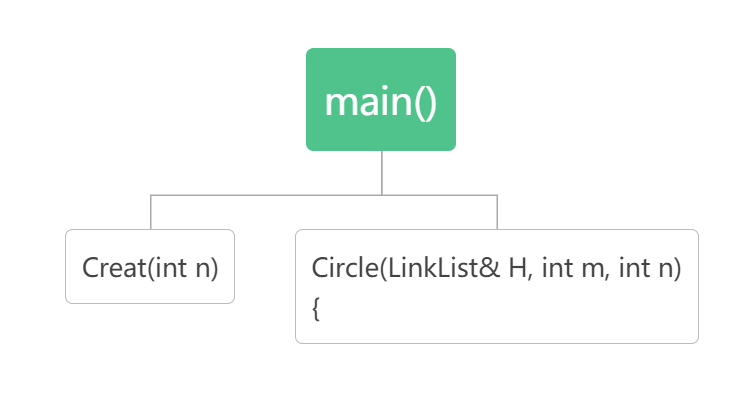
    DestroyList( &L )

    Greate( &L )

    Circle(&L)

} ADT CircleList

（2）主程序的流程



3、详细设计

（1）创建一个大小为n的循环单链表

LinkList Creat(int n)

{

    LinkList L = NULL;

    LNode\* s, \* r = NULL;

    int x;

    for (int i = 1; i <= n; i++)

    {

        LNode\* s = new LNode;

        cin >> x;

        s->data = x;

        s->num = i;

        if (L == NULL) L = s;

        else r->next = s;

        r = s;

    }

    if (r != NULL) r->next = L;

    return L;

}

（2）依次显示退出的人和最后的胜者

void Circle(LinkList& H, int m, int n)

{

    LinkList q, p = H;

    for (int j = 1; j <= n; j++)

    {

        for (int i = 1; i < m; i++)

        {

            p = p->next;

        }

        m = p->data;

        if (p->next == p)

        {

            cout << endl

                 << "最后胜利者为：" << p->num;

        }

        else

            cout << p->num << " ";

        q = p->next;

        p->num = q->num;

        p->data = q->data;

        p->next = q->next;

        free(q);

    }

}

4、调试分析

（1）调试过程中遇到的主要问题及解决过程：无

（2）分析程序代码的质量

正确性：实现大部分输入数据都能有理想的输出

健壮性：在一定的数据输入范围内，该程序能较好地实现链表的操作。

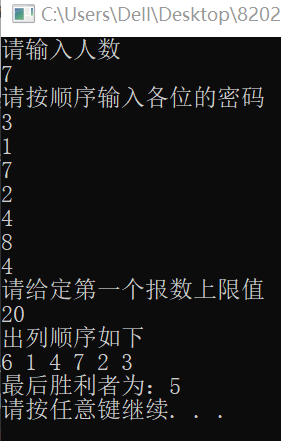
5、使用说明

循环链表类和约瑟夫函数的全部实现内容均包含在一个独立的C++头文件JosephuesProblem.h中，要在程序中使用这个类只需包含该头文件即可。这个头文件实现了循环链表类的创建、显示、约瑟夫迭代操作，从稳定性和逻辑性来看没有问题。

6、测试程序的运行结果



输出结果



7、心得体会

本题实际上也可用其他方法，比如数组取模去循环，但是，远不如循环链表来得直观，同时，动态的释放内存也让循环链表的做法在这一题中更具优势。

**附录：源程序文件清单**

各程序源代码随本实验报告电子版一起打包，存放在lab1子目录。

文件清单如下：

SingleLinkList.h··················单链表定义及实现，c++头文件

SingleLinkList.cpp·······················单链表测试程序代码

CitiesManagement.h···························城市类定义实现及相关操作

CitiesManagement.cpp·······················城市链表测试程序代码

JosephuesProblem.h·····················循环链表定义及实现，约瑟夫问题实现，c++头文件

JosephuesProblem.cpp·······················约瑟夫问题测试程序代码