**中南大学**

**数据结构实验报告**

题 目 线性表

学生姓名 翟衍博

学 号 8207200203

指导教师 余腊生

学 院 计算机学院

专业班级 计算机2006

完成时间 2020年3月27日

指导教师评定： 签 名：

# 单链表操作的实现

1. **需求分析**

单链表的基本操作包括了创建、插入、删除、查找、遍历。

1. 输入的形式和输入值的范围：程序是按数字键选择对应功能的，在各项功能中，创建、插入、删除、查找需要输入数据。
2. 输出的形式：在单链表初始化后显示初始化成功，在检查链表是否为空时显示是否为空的结果，在将清空链表后显示链表已清空，在获取链表长度后会显示链表的长度，在遍历后会显示所有的元素，在查找后会显示指定的元素。
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，清空单链表，求链表长度，遍历链表，按位置查找元素，查找指定元素的位置，按位置插入元素，在指定元素后面插入，按位置删除元素，按值删除元素。
4. 测试数据：

* 初始化后执行创建一个长度为5的单链表，依次输入1,2,4,3,5；
* 选择求链表长度；
* 选择检查链表是否为空；
* 选择按位置查找元素，输入1；
* 选择按位置插入元素，依次输入1，6；
* 选择按位置删除元素，输入1；
* 遍历链表；

1. **概要设计**
2. 抽象数据类型：

ADT LinkedList{

数据对象 

数据关系 

基本操作

InitList(&L)

操作结果：创建了一个空链表

DestroyList(&L)

初始条件：链表已存在

操作结果：链表被销毁

ClearList(&L)

初始条件：链表已存在

操作结果：将链表清为空链表

ListEmpty(L)

初始条件：链表已存在

操作结果：如果链表为空则返回1，链表非空则返回0

ListLength(L)

初始条件：链表已存在

操作结果：返回链表的长度

Insert(&L)

初始条件：链表已存在

操作结果：插入一个元素到链表中

Find(&L)

初始条件：链表已存在

操作结果：查找链表中的元素

Delete(&L)

初始条件：链表已存在

操作结果：删除链表中的元素

}ADT LinkedList

1. 本程序包含的函数：

2.1 cpp文件包含的函数：

主函数main( )

显示菜单showmenu( )

通过数组创建一个链表createL ( )

销毁一个链表destoryL( )

获取链表的长度lengthL( )

判断链表是否为空emptyL( )

插入一个元素insertL( )

获取元素的值getL( )

显示链表showL( )

删除一个元素deleteL( )

查找元素的位置findL( )

2.2 .h文件包含的函数：

构造函数LinkedList( )

析构函数~ LinkedList( )

销毁函数 destroy( )

通过数组创建链表creatLinkedList(T nums[],int n)

获取链表长度getLength()

判断链表是否为空isEmpty()

获得链表元素的值getVal(int index)

显示链表所有元素的值display()

在指定位置插入元素insert(T num,int index)

删除指定位置的元素del(int index)

查询元素find(T target)

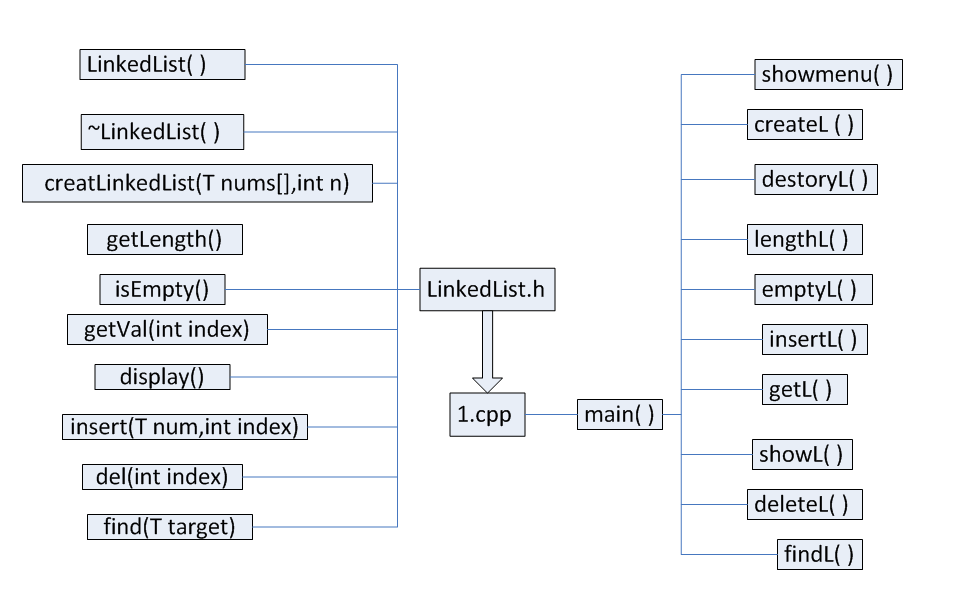


图1 函数模块及关系图

在LinkedLIst.h文件中使用C++模板类实现链表。确定所需提供的可供用户及外部程序方位的公共接口的函数原型，以及相关实现细节部分。

* 构造函数与析构函数
* 基本操作的公共接口，包括引用型操作和修改型操作。
* 数据成员。使用一个指针保存链表的头结点的存储位置及状态

1. **详细设计**
2. 公共接口。根据需求分析的结果，得到的单链表模板类的公共接口如下：

template<typename T>

class ListNode{

public:

    T val;

    ListNode<T>\* next;

};

template<typename T>

class LinkedList

{

private:

    /\* data \*/

    ListNode<T>\* head;

public:

    /\* function \*/

    LinkedList<T>();                           //创建

    ~LinkedList<T>();

    bool creatLinkedList(T nums[],int n);

    void destoryLinkedList();

    int getLength();                        //获得长度

    bool isEmpty();                         //判断是否为空

    T getVal(int index=0);                    //获得指定位置的元素，索引从0开始

    void display();                         //打印链表

    void insert(T num,int index=0);         //在指定位置插入元素，默认为头插

    void del(int index=0);                  //删除指定元素，默认为头删

    int find(T target);                     //查找元素，找到返回元素位置，找不到返回-1

};

1. 数据成员：类中只有一个指针成员head用于标识单链表的头结点，属于类的实现细节，不能被用户直接访问，为私有成员。
2. 主要操作的实现。

//函数名称：构造函数

//功能描述：创建一个带头结点的空的单链表，表长存储在头结点中

//预备条件：无

//参数列表：无

//后置条件：堆区链表创建

template <typename T>

LinkedList<T>::LinkedList()

{

    head=new ListNode<T>;

    head->val=0;

    head->next=NULL;

}

//函数名称：析构函数

//功能描述：释放链表所占用的空间

//预备条件：无

//参数列表：无

//后置条件：堆区空间释放

template <typename T>

LinkedList<T>::~LinkedList<T>()

{

    destoryLinkedList();

}

//函数名称：销毁函数

//功能描述：释放链表所占用的空间

//预备条件：无

//参数列表：无

//后置条件：堆区空间释放

template <typename T>

void LinkedList<T>::destoryLinkedList()

{

    ListNode<T>\* pre;

    ListNode<T>\* temp;

    pre=head;

    while(pre)

    {

        temp=pre->next;

        pre->next=NULL;

        delete pre;

        pre=temp;

    }

}

//函数名称：创建函数

//功能描述：使用尾插法根据数组创建一个带头结点的单链表，表长存储在头结点中

//预备条件：存在一个非空数组

//参数列表：输入参数为指向数组元素的指针、元素的个数

//后置条件：堆区链表创建

template<typename T>

bool LinkedList<T>::creatLinkedList(T nums[],int n)

{

    head->val=n;

    head->next=NULL;

    ListNode<T>\* rear=head;              //使用的是尾插法，因此要使用一个尾指针

    for(int i=0;i<n;i++)

    {

        ListNode<T>\* temp=new ListNode<T>();

        if(! temp)  return false;       //申请失败

        temp->val=nums[i];                //数据域赋值

        rear->next=temp;

        rear=temp;                      //更新尾指针的值

    }

    rear->next=NULL;

    return true;

}

//函数名称：获得链表长度的函数

//功能描述：获得链表长度

//预备条件：无

//参数列表：无

//返回值 ：链表中元素的个数，链表为空返回0

//后置条件：无

template<typename T>

int LinkedList<T>::getLength()

{

    return int(head->val);

}

//函数名称：判空函数

//功能描述：判断链表是否为空

//预备条件：无

//参数列表：无

//返回值 ：链表为空返回true，非空返回false

//后置条件：无

template<typename T>

bool LinkedList<T>::isEmpty()

{

    return int(head->val)==0;

}

//函数名称：获得元素的值

//功能描述：获得链表特定位置元素的值

//预备条件：链表非空，指定位置不能小于0也不能大于等于表长

//参数列表：元素位置index

//返回值 ：索引为index的元素的值

//后置条件：无

template<typename T>

T LinkedList<T>::getVal(int index)

{

    if(index<0||index>=int(head->val))

    {

        cout<<"getVal函数传递的参数不合理"<<endl;

        exit(0);

    }

    ListNode<T>\* p= head->next;

    while(index--)

    {

        p=p->next;

    }

    return p->val;

}

//函数名称：显示函数

//功能描述：遍历链表并逐一显示元素的值

//预备条件：无

//参数列表：无

//后置条件：控制台显示链表所有元素并换行

template<typename T>

void LinkedList<T>::display()

{

    if(this->isEmpty())

    {

        cout<<"此链表为空"<<endl;

        return ;

    }

    ListNode<T>\* p=head->next;

    cout<<"链表的值为：";

    while(p)

    {

        cout<<p->val<<",";

        p=p->next;

    }

    cout<<endl;

}

//函数名称：插入函数

//功能描述：在指定位置插入一个元素

//预备条件：指定位置不能大于表长也不能小于0

//参数列表：输入参数为待插入元素、指定位置的下标

//返回值 ：无

//后置条件：链表元素和长度发生改变

template<typename T>

void LinkedList<T>::insert(T num,int index)

{

    if(index<0||index>int(head->val))

    {

        cout<<"insert函数传递的参数不合理"<<endl;

        return;

    }

    ListNode<T>\* p=head;

    while(index--)                          //找到待插入位置的前驱

    {

        p=p->next;

    }

    ListNode<T>\* temp=new ListNode<T>();    //申请新节点

    temp->val=num;                          //新节点的数据域赋值

    temp->next=p->next;                     //原先节点的后继作为新节点的后继

    p->next=temp;                           //新节点作为原先节点的后继

    head->val++;

    //this->display();

}

//函数名称：删除函数

//功能描述：删除指定的元素

//预备条件：指定的元素存在才能成功

//参数列表：指定元素的值

//返回值 ：无

//后置条件：链表元素和长度发生改变

template<typename T>

void LinkedList<T>::del(int index)

{

    if(index<0||index>int(head->val))

    {

        cout<<"del函数传递的参数不合理"<<endl;

        return;

    }

    ListNode<T>\* p=head;

    while(index--)         //找到待删除节点的前驱，使用后删

    {

        p=p->next;

    }

    ListNode<T>\* temp=p->next;  //记录被删节点

    //T delval=temp->val;       //记录被删元素

    p->next=p->next->next;      //拆链

    delete temp;                //释放被删节点

    temp=NULL;                  //!delete后一定要指控

    head->val--;

    //this->display();

}

//函数名称：查找函数

//功能描述：查找指定位置的元素并

//预备条件：指定位置要有结点存在

//参数列表：输入参数为指定位置

//返回值 ：待查找元素的索引，找不到返回-1

//后置条件：无

template<typename T>

int LinkedList<T>::find(T target)

{

    int res=0;

    ListNode<T>\* p=head->next;

    while(p!=NULL&&p->val!=target)

    {

        p=p->next;

        res++;

    }

    return p==NULL?0:res;

}

1. **调试分析**
2. 调试过程中遇到的主要问题及解决过程。在调试过程中曾发现在一些函数结束后会自动调用析构函数，我起初表示很奇怪，因为我并没有手动调用析构函数，而且测试链表的生命周期也没有结束，后来才发现是在有些函数体内部我定义了新的链表对象，函数结束了这个对象的生命周期就结束了，自然也就调用了它的析构函数。
3. 分析程序代码的质量

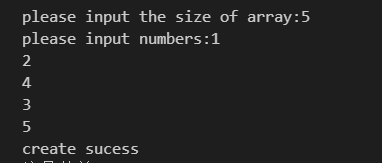
* 正确性。在一定测试数据集上表现正确。
* 健壮性。在一定的数据输入范围内，该程序能较好地实现链表的操作。但是如果输入数据非法，该程序还是可能会产生一些预想不到的输出结果。本实验的重点是数据结构的实现，所以程序的健壮性方面我考虑的并不是很全面。

1. **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LinkedList.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。在1.c文件中引入该头文件，并编写对应的函数和菜单函数，就可以让用户满足基本的单链表操作。

1. **测试程序的运行结果**
2. 创建单链表

输入5，然后依次输入1 2 4 3 5；



1. 求链表长度

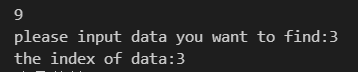
输出5

1. 检查链表是否为空

输出链表非空

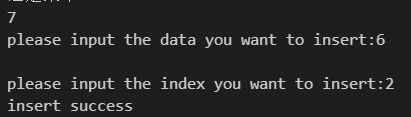
1. 按位置查找元素

输入3，输出3



1. 按位置插入元素

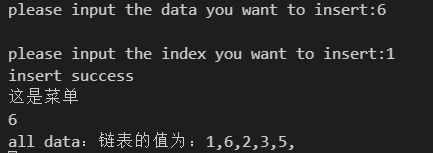
依次输入6，2



1. 按位置删除元素

输入3

1. 遍历链表；



1. **心得体会**
2. 学到了c++类模板，以前写数据结构都是默认int，这样不具备通用性，这次学了c++类模板并付诸使用。
3. 程序调试过程是一个漫长的过程，当自己写完程序后在运行的过程中总会和自己预想的有些偏差，这是就需要单步调试来找错了。在其间令我印象深刻的是在用c++delete temp的时候要先把temp的后继指向空，不这样就会运行出错，也是找到了学长数据结构报告里的一个小错误。

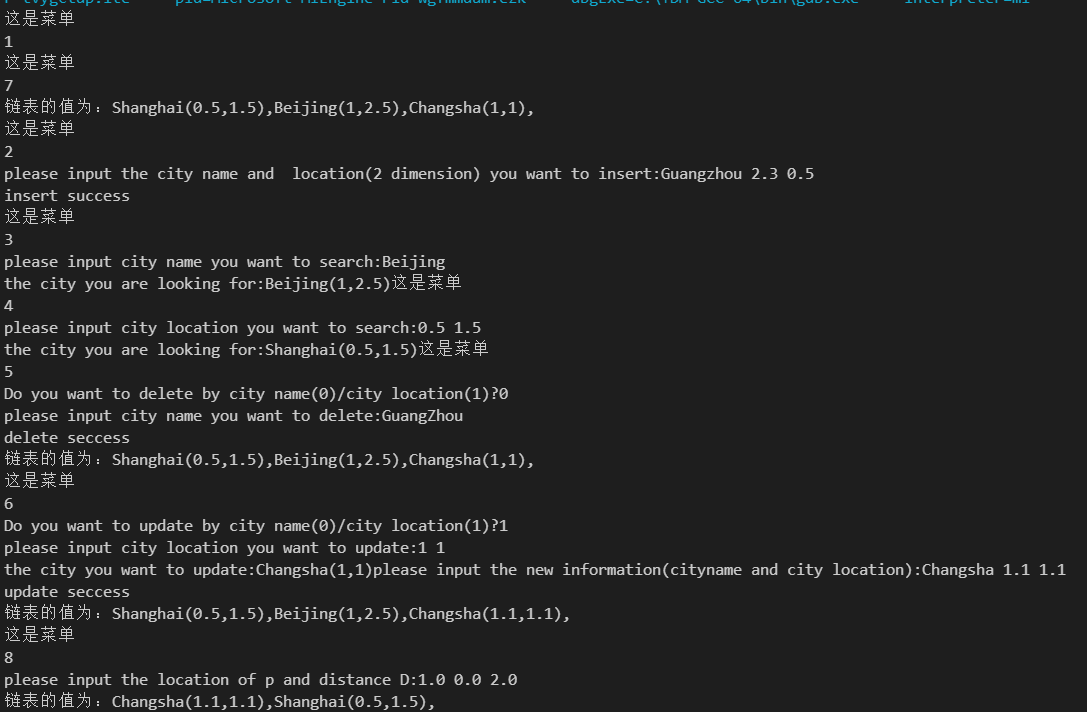
# 城市链表

1. **需求分析**

城市链表的基本操作包括了显示、插入、删除、查找、更新、范围查找。

1. 输入的形式和输入值的范围：制作一个菜单，程序是按数字键选择对应功能的，在各项功能中，插入、删除、查找、范围查找需要输入数据，其中城市名称为字符串类型，城市相对位置为浮点数，规定相对位置为非负数，其中城市相对位置设为二维。
2. 输出的形式：输出格式为“城市名（城市位置）”，如“Changsha(1,1)”。
3. 程序所能达到的功能：完成创建城市链表、插入城市数据、根据城市名称或者城市位置删除城市数据、根据城市名称和城市位置查找城市数据、更新城市经纬度、查找距某一特定位置的距离不大于某一距离的城市。
4. 测试数据：

* 选择1,创建城市链表，依次输入3 Changsha 1.0 1.0 Beijing 1.0 2.5 Shanghai 0.5 1.5
* 选择7，显示链表数据，输出“Shanghai(0.5,1.5),Beijing(1,2.5),Changsha(1,1)”
* 选择2，插入城市数据，输入“GuangZhou 2.3 0.5”
* 选择7，显示链表数据，输出“Guangzou(2.3，3.5)，Shanghai(0.5,1.5),Beijing(1,2.5),Changsha(1,1)”
* 选择3，按城市名称查找城市，输入Beijing,输出“the city you are looking for：Beijing(1,2.5)”
* 选择4，按城市位置查找城市，输入0.5 1.5，输出“the city you are looking for: Shanghai(0.5,1.5)”
* 选择5，删除城市数据，选择0，按城市名称删除，输入GuangZhou,输出“delete success”，并展示删除后的链表
* 选择6，更新城市数据，选择1，按城市位置更改，输入Changsha 1.1 1.1,输出“delete success”，并展示更改后的链表
* 选择8，查找距某一特定位置的距离不大于某一距离的城市，依次输入指定地点P与指定位置D，输入1.0 1.0 2.0，输出满足条件的城市：“Changsha(1.1,1.1),Shanghai(0.5,1.5)”



1. **概要设计**
2. 抽象数据类型：

ADT CityList{

数据对象 

数据关系 

基本操作

InitList(&L，filename)

操作结果：导入数据创建了一个城市链表

Insert(&L,newCity)

初始条件：链表已存在

操作结果：插入一个城市数据到链表中

Find(&L，cityName)

初始条件：链表和城市已存在

操作结果：查找链表中匹配的城市并返回其值

Delete(&L，cityName)

初始条件：链表和城市已存在

操作结果：删除链表中的城市

Update(&L, newData)

初始条件：链表和城市已存在

操作结果：更新链表中城市的数据

Circle(&L,distance)

初始条件：链表存在，距离不小于0

操作结果：创建一个新的城市链表

}ADT CityList

1. 本程序包含的函数：

主函数 main( )

菜单函数 showMenu( )

创建城市链表：createC()

插入一个城市 insertC( )

通过城市名查找：searchByName()

通过城市位置查找：searchByLoc()

删除一个城市 deleteC( )

更新城市信息 updateC( )

运算符重载函数 operator( )

类型转换重载函数 operator int( )

距离函数 getDistance( )

1. **详细设计**
2. 公共接口。根据需求分析的结果，进一步加入对实现细节的考虑，例如在查找城市时使用部分匹配比使用全部匹配在交互上更加友好。另外，加上C++类设计中的正规函数（包括构造函数、析构函数等），得到的城市类的公共接口如下：

//构造函数

    City()

    {

        this->cname="undefined";

        this->p.x=0;

        this->p.y=0;

    }

    City(string cname, double x, double y)

    {

        this->cname=cname;

        this->p.x=x;

        this->p.y=y;

    }

    //更新位置坐标

    void update(double x,double y)

    {

        this->p.x=x;

        this->p.y=y;

    }

    //重载==、！=运算符，方便后后续的查找需要

    bool operator==(City c)

    {

        return this->cname==c.cname||this->p.x==c.p.x&&this->p.y==c.p.y;

    }

    bool operator!=(City c)

    {

        return !(this->cname==c.cname||this->p.x==c.p.x&&this->p.y==c.p.y);

    }

    //重载输入运算符

    friend istream& operator>>(istream& in,City &c)

    {

         in >> c.cname >> c.p.x>>c.p.y;

        return in;

    }

    //重载输出运算符，为了兼容cout<<val

    friend ostream& operator<<(ostream& out , City &c)

    {

        cout<<c.cname<<"("<<c.p.x<<","<<c.p.y<<")";

        return out;

    }

   //以下4个运算符重载（对++、--、=、int（））都是为了兼容头节点的数据域表示链表长

    //重载++、--运算符

    City operator++(int n)      //有参数n的为++后置形式

    {

        this->p.x++;

        return \*this;

    }

    City operator--(int n)      //有参数n的为--后置形式

    {

        this->p.x--;

        return \*this;

    }

    //定义City到int的类型转换

    operator int()

    { return int(this->p.x); }

    //重载赋值运算

    City operator=(int x)

    { this->p.x = x;  return \*this; }

    //距离函数

    double getDistance(Location p2)

    {

        return sqrt((this->p.x-p2.x)\*(this->p.x-p2.x)+(this->p.y-p2.y)\*(this->p.y-p2.y));

    }

1. 数据成员：类中有两个私有成员城市名和坐标。

struct Location

{

    double x;

    double y;

    //构造函数

    Location(){};

    Location(double x,double y)

    {

        this->x=x;

        this->y=y;

    }

};

private:

    string cname;   //城市名称

    Location p;     //位置坐标

主要操作的实现：

void showMenu()

{

    // system("cls");

    // cout<<"menu：\n"

    //     <<"1.creat city linkedlist\n"

    //     <<"2.insert city node\n"

    //     <<"3.search by city name\n"

    //     <<"4.search by city location\n"

    //     <<"5.delete city (by city name or by city location)\n"

    //     <<"6.update city (by city name or by city location)\n"

    //     <<"7.display all city information\n"

    //     <<"8.show all cities whose distance from location P below D\n"

    //     <<"0.exit\n"

    //     <<"------------------------------------------------------------------\n"

    //     <<"please input a integer:";

    cout<<"这是菜单"<<endl;

}

//功能1：创建城市链表

void createC(LinkedList<City>& cities)

{

    // system("cls");

    cout<<"please input the number of cities:";

    int num=0;

    cin>>num;

    cout<<"please input"<<num<<" cities name and  location(2 dimension) one by one:";

    while(num--)

    {

        City c;

        cin>>c;

        cities.insert(c);

    }

    cout<<"creat success"<<endl;

}

//功能2：插入城市节点

void insertC(LinkedList<City>& cities)

{

    // system("cls");

    cout<<"please input the city name and  location(2 dimension) you want to insert:";

    City c;

    cin>>c;

    cities.insert(c);

    cout<<"insert success"<<endl;

}

//功能3：通过城市名查找

void searchByName(LinkedList<City>& cities)

{

    // system("cls");

    string targetname;

    cout<<"please input city name you want to search:";

    cin>>targetname;

    City temp(targetname,-1,-1);    //创建临时的city对象，便于查找,

                                    //因为location的两个分量只有正数才有意义，所以一定是由于城市名相等才返回的

    int index=cities.find(temp);

    //未找到

    if(index==-1)

    {

        cout<<"no information you are looking for."<<endl;

        return;

    }

    //找到了

    City tempc=cities.getVal(index);

    cout<<"the city you are looking for:"<<tempc;

}

//功能4：通过城市位置查找

void searchByLoc(LinkedList<City>& cities)

{

    // system("cls");

    double targetx,targety;

    cout<<"please input city location you want to search:";

    cin>>targetx>>targety;

    City temp("\*#\*#\*",targetx,targety); //创建临时的city对象，便于查找,

    int index=cities.find(temp);

    //未找到

    if(index==-1)

    {

        cout<<"no information you are looking for."<<endl;

        return;

    }

    //找到了

    City tempc=cities.getVal(index);

    cout<<"the city you are looking for:"<<tempc;

}

//功能5：删除城市信息（按城市名称/城市位置）

void deleteC(LinkedList<City>& cities)

{

    // system("cls");

    cout<<"Do you want to delete by city name(0)/city location(1)?";

    bool ch;

    cin>>ch;

    int index;

    //按城市名称删除

    if(!ch)

    {

        string targetname;

        cout<<"please input city name you want to delete:";

        cin>>targetname;

        City temp(targetname,-1,-1);    //创建临时的city对象，便于查找,

                                        //因为location的两个分量只有正数才有意义，所以一定是由于城市名相等才返回的

        int index=cities.find(temp);

        //未找到

        if(index==-1)

        {

            cout<<"no existence."<<endl;

            return;

        }

        cities.del(index);

        cout<<"delete seccess"<<endl;

    }

    //按城市位置删除

    else

    {

        double targetx,targety;

        cout<<"please input city location you want to delete:";

        cin>>targetx>>targety;

        City temp("\*#\*#\*",targetx,targety); //创建临时的city对象，便于查找

        int index=cities.find(temp);

        //未找到

        if(index==-1)

        {

            cout<<"no existence."<<endl;

            return;

        }

        cities.del(index);

        cout<<"delete seccess"<<endl;

    }

    cities.display();

}

//功能6：更新城市信息（按城市名称/城市位置）

void updateC(LinkedList<City>& cities)

{

    // system("cls");

    cout<<"Do you want to update by city name(0)/city location(1)?";

    bool ch;

    cin>>ch;

    int index;

    //按城市名称更新

    if(!ch)

    {

        string targetname;

        cout<<"please input city name you want to update:";

        cin>>targetname;

        City temp(targetname,-1,-1);    //创建临时的city对象，便于查找,

                                        //因为location的两个分量只有正数才有意义，所以一定是由于城市名相等才返回的

        int index=cities.find(temp);

        //未找到

        if(index==-1)

        {

            cout<<"no existence."<<endl;

            return;

        }

        City tempc=cities.getVal(index);

        cout<<"the city you want to update:"<<tempc;

        cout<<"please input the new information(cityname and city location):";

        cin>>tempc;

        //先删再插

        cities.del(index);

        cities.insert(tempc,index);

        cout<<"update seccess"<<endl;

    }

    //按城市位置更新

    else

    {

        double targetx,targety;

        cout<<"please input city location you want to update:";

        cin>>targetx>>targety;

        City temp("\*#\*#\*",targetx,targety); //创建临时的city对象，便于查找

        int index=cities.find(temp);

        //未找到

        if(index==-1)

        {

            cout<<"no existence."<<endl;

            return;

        }

        City tempc=cities.getVal(index);

        cout<<"the city you want to update:"<<tempc;

        cout<<"please input the new information(cityname and city location):";

        cin>>tempc;

        //先删再插

        cities.del(index);

        cities.insert(tempc,index);

        cout<<"update seccess"<<endl;

    }

    cities.display();

}

//功能7：显示城市链表

void displayC(LinkedList<City>& cities)

{

    // system("cls");

    cities.display();

}

//功能8：显示距位置P的距离小于D的所有城市

void dis\_below(LinkedList<City>& cities)

{

    double px,py,d;

    cout<<"please input the location of p and distance D:";

    cin>>px>>py>>d;

    Location P(px,py);

    LinkedList<City>\* closer=new LinkedList<City>();

    for (int i = 0; i < cities.getLength(); i++)

    {

        City c=cities.getVal(i);

        //如果在指定范围内就插入匹配链表

        if (c.getDistance(P) <= d)

        {

            closer->insert(c);

        }

    }

    closer->display();

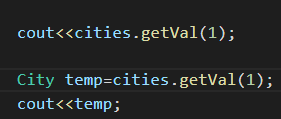
}

1. **调试分析**
2. 调试过程中遇到的主要问题及解决过程。

在编写求距离的函数的时候，发现不能访问类的私有数据成员，解决方法是将该函数声明为City类的友元函数，声明为友元函数后vscode仍有下滑波浪线提示有错误，但是程序却可以编译运行，在网上查阅资料一些用户也遇到过这样的问题，后来通过将距离函数编写为类的成员函数而解决。

我设计了一个城市类city，成员变量有cityname、x、y三个变量，为了满足头节点能存放链表长度，我重载了int（）运算符，也就是对一个city对象进行int强制类型转换得到的int（x）；然后我为了方便城市对象的输出又重载了<<运算符，"<<+city对象"可以输出cityname、x、y,三个信息，但是尴尬的是我"<<+city对象"的时候，它自动调用了int()强制转换，然后输出了这个对象的x取整，而不是我想要的输出三个变量的值。

这里的cities是一个值为city类的链表，T getval（int n）是获取链表中索引为n的值。上面那种直接写法就不行，输出的是一个数，下面那种写法就可以，可以输出cityname、x、y三个值。我感到有些奇怪。



1. 分析程序代码的质量

* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。可以判断一些非法输入，但不能保证程序十分健壮，因为我还是将侧重点放在功能的实现以及链表这种数据结构的体会上，没有放太多精力在系统的健壮性上。

1. **使用说明**

城市类及部分操作函数包含在一个C++头文件CityLinkedList.h中，CityLinkedList.h又使用LinkedList.h头文件，在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类定义、计算距离的函数，而城市链表的创建、删除、查找、遍历等操作则是由LinkedList.h实现，所以也被包含在这个头文件中。具体菜单对应的各个函数的实现在2.cpp文件中。

1. **心得体会**
2. 了解了运算符重载的妙处。比如在模板类的数据结构定义中，我们若采用头节点存储链表长的方法时，在类模板函数的实现中就会出现出现把头节点的数据当作整数进行的一些操作，如赋整数的值、自增、自减运算。在我们想要使用类模板时，就需要兼容头节点的这些操作，而运算符重载恰好可以解决这样的兼容问题。
3. 体会了逐模块调试的有效。
4. 重新复习了友元函数的相关知识。

# 约瑟夫环

1. **需求分析**

本问题使用循环链表求解约瑟夫环问题。

1. 输入的形式和输入值的范围：输入正整数上限值m与人数n（n<=30）与每个人的密码。
2. 输出的形式：按出列顺序输出每个人的编号。
3. 程序所能达到的功能：程序应该模拟约瑟夫环的功能。
4. 测试数据：

依次输入20 7 3 1 7 2 4 8 4

1. **概要设计**
2. 抽象数据类型：

ADT CirLinkedList{

数据对象 

数据关系 

基本操作

InitCircle(&C)

操作结果：创建一个循环链表

DestoryCircle(&C)

操作结果：销毁一个循环链表

DoJosephus(&L)

初始条件：存活人数大于1

操作结果：将出队的结果顺序输出

}ADT CirLinkedList

1. 本程序包含的函数：

主函数 main( )

构造函数 CirLinkedList::CirLinkedList()

析构函数 CirLinkedList::~CirLinkedList()

创建循环链表 void CirLinkedList::creatCir(int n)

获取循环链表的长度 int CirLinkedList::getlen()

判断循环链表是否为空 bool CirLinkedList::empty()

删除指定号码的节点，并返回其密码 int CirLinkedList::del(int number)

约瑟夫函数 void doJosephus(CirLinkedList& c,int m)

1. **详细设计**
2. 公共接口：根据需求分析的结果，进一步加入对实现细节的考虑，另外，加上C++类设计中的正规函数（包括构造函数、析构函数等），得到的循环链表类的公共接口如下：

class CirLinkedList

{

private:

    /\* data \*/

    CirListNode\* rear;  //带头结点的有尾指针的循环链表

public:

    CirLinkedList();

    ~CirLinkedList();

    void creatCir(int n);

    int getlen();

    bool empty();

    int del(int number);

    //友元函数，进行约瑟夫操作

    friend void doJosephus(CirLinkedList& c,int m);

};

1. 数据成员：只有一个尾指针rear作为私有成员，能被友元函数和自带函数访问。用于标记循环的起始位置和插入结点时的尾部。
2. 主要操作的实现：

//构造函数

CirLinkedList::CirLinkedList()

{

    rear=nullptr;

}

//析构函数

CirLinkedList::~CirLinkedList()

{

    CirListNode\* p=rear;

    CirListNode\* q=rear->next;

    while(p!=q)

    {

        p->next=p->next->next;

        q->next=nullptr;

        delete q;

        q=p->next;

    }

    q->next=nullptr;

    delete q;

}

//创建循环链表

void CirLinkedList::creatCir(int n)

{

    rear=new CirListNode();

    if(!rear)

    {

        cout<<"申请空间失败！"<<endl;

        return ;

    }

    rear->next=new CirListNode();   //头节点

    if(!rear->next)

    {

        cout<<"申请空间失败！"<<endl;

        return ;

    }

    rear->next->next=rear;

    rear->next->code=n;

    rear->next->number=0;

    cout<<"please input "<<n<<" codes one by one:";

    rear->number=1;

    cin>>rear->code;

    //第一个节点在上面已经申请，因此从第二个结点开始

    for(int i=2;i<=n;i++)

    {

        //采用后插

        CirListNode\* temp=new CirListNode();    //1.申请新节点

        if(!temp)

        {

            cout<<"申请空间失败！"<<endl;

            return ;

        }

        cin>>temp->code;                        //2.数据域赋值

        temp->number=i;

        temp->next=rear->next;                  //3.新节点的后继作为原先节点的后继

        rear->next=temp;                        //4.新节点作为原先节点的后继

        rear=rear->next;           //更新尾指针的

    }

}

//获取循环链表的长度

int CirLinkedList::getlen()

{

    return rear->next->code;

}

//判断循环链表是否为空

bool CirLinkedList::empty()

{

    return rear->next->code==0;

}

//删除指定号码的节点，并返回其密码

int CirLinkedList::del(int number)

{

    //异常情况，链表为空

    if(!rear)    return -1;

    CirListNode\* p=rear;

    while(p->next->number!=number)

    {

        p=p->next;          //这是p指向了待删除元素的前驱节点

    }

    //后删节点

    CirListNode\* q=p->next; //获取被删节点

    int res=q->code;        //获取被删元素

    p->next=q->next;        //拆链

    q->next=nullptr;

    //后删可能要更新尾指针，这是在一步步调试中发现的问题

    if(rear==q)

        rear=p;

    delete q;               //释放被删节点

    rear->next->code--;

    return res;

}

//约瑟夫函数

void doJosephus(CirLinkedList& c,int m)

{

    cout<<"the order of departure:";

    CirListNode\* pt=c.rear->next;

    int cnt=1;

    while(c.getlen()>1)

    {

        if(!pt->number)  //是头节点，直接跳过，指数器不加1

        {

            pt=pt->next;

        }

        else

        {

            pt=pt->next;

            cnt++;

            if(!pt->number)  //是头节点，直接跳过，指数器不加1

            {

                pt=pt->next;

            }

        }

        if(cnt==m)

        {

            cout<<pt->number<<" ";  //输出

            CirListNode\* temp=pt->next; //临时保存pt

            m=c.del(pt->number);    //更新m

            pt=temp;

            if(!pt->number)  //是头节点，直接跳过，指数器不加1

            {

                pt=pt->next;

            }

            cnt=1;                 //更新cnt

        }

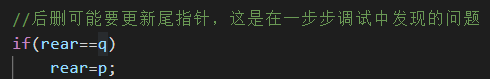
    }

    //最后留下来的数

    cout<<c.rear->number;

}

1. **调试分析**
2. 调试过程中遇到的主要问题及解决过程。在执行约瑟夫函数后，发现程序退出异常,经单步调试后发现是在约瑟夫函数中调用了删除函数del时，在进行后删节点的时候，没有移动rear指针，然后delete释放掉了rear，因此在后续操作是用到rear指针就报错了。解决方法是在delete节点是，先判断是不是rear指针指向的节点，如果是，则更新rear指针的值，让其指向它的前驱。



1. 分析程序代码的质量

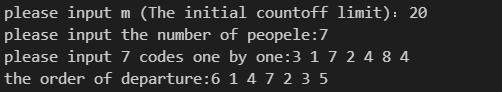
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。该程序能够处理输入数据非法的情况，将输入限制在合法数据内，健壮性目前没有问题。

1. **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CirLinkedList.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等一些基本操作以及约瑟夫环的模拟函数

1. **测试程序的运行结果**

依次输入20 7 3 1 7 2 4 8 4，输出：6 1 4 7 2 3 5



1. **心得体会**

通过此实验体会到了循环链表的特点，就是“循环”这一特性，模拟约瑟夫环问题可以算是它的一个典型应用了。学习数据结构要掌握的一个关键就是明白什么时候该使用哪种数据结构，经过实践可以体会到在问题涉及到循环往复、多插删时可以考虑使用循环链表。

此外，还体会到了计算机模拟的过程，像约瑟夫问题，在数据规模很小的时候，我们人脑凭借规则可以比较容易地得出答案，而将我们的思维过程编写成代码却并不是一件易事；但是，当数据量和复杂性一旦加大到一定规模，计算机的优越性就体现出来了，它可以很快地给出答案。

**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src子目录。

文件清单如下：

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

1.cpp·······················单链表测试程序代码

CityLinkedList.h·················城市类定义实现及相关操作，c++头文件

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

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

3.cpp·····················约瑟夫问题测试程序代码

1.exe······················单链表测试程序

2.exe·························城市链表测试程序

3exe·························约瑟夫问题测试程序

Readme.txt······················说明文件