

《数据结构》

课程设计报告

课程名称：	《数据结构》课程设计
课程设计题目：	约瑟夫环
姓 名：	金宏睿
院 系：	计算机学院
专 业：	计算机科学与技术
班 级：	19052318
学 号：	19051814
指导教师：	葛瑞泉

2020 年 11 月 13 日

一、需求分析

功能需求：

约瑟夫环(Joseph)问题的一种描述是：编号为 $1, 2, 3, \dots, n$ 的 n 个人按顺时针方向围坐一圈，每人持有一个密码(正整数)，一开始任选一个正整数作为报数上限值 m ，从第一个人开始按顺时针方向自 1 开始顺序报数，报到 m 时停止报数，报 m 的人出列，将他的密码作为新的 m 值，从他在顺时针方向上的下一个人开始重新从 1 报数，如此下去，直至所有人全部出列为止。试设计一个程序求出出列顺序。

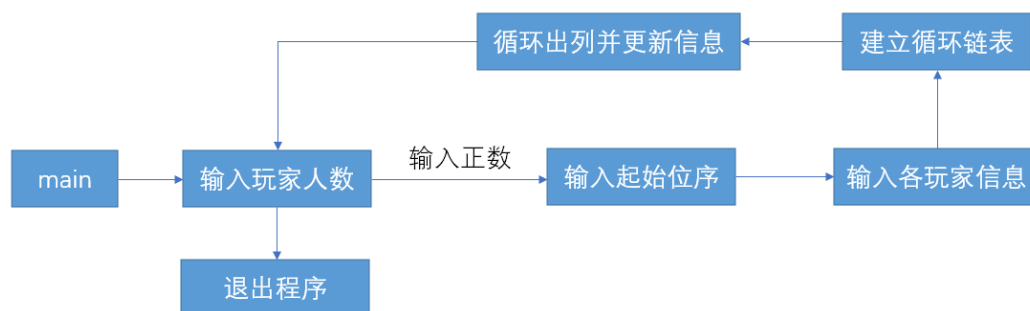
例如：有 5 个人围坐一圈，位序分别为 $1, 2, 3, 4, 5$ 。他们持有的密码分别为 $5, 4, 3, 2, 1$ 。

初始报数值 m 为 1 ，则首先第 1 人出列，以第 1 人的密码 5 为 m 值，顺时针方向继续报数，则下一出列人为 $2, \dots$ 直至所有人全部出列。如此，出列次序为 $1, 2, 3, 4, 5$ 。

界面需求：

通过提示，引导操作者输入玩家数量、开始位置和每个人出列后向后数的位次。可以根据玩家的输入退出程序。

二、概要设计



模块层次结构设计

```
void JosethMain() //程序入口负责
void JosephOutput(Linklist<T> *l, int m) //处理信息并输出出队序列
int JosephBegin(//开始约瑟夫环并引导输入人数
int JosephPlayer(int &n) //引导输入游戏人数
Linklist *CreatJoseph(int num) //引导输入信息，创建约瑟夫 h 环链表
```

接口设计

```
//初始化链表，将链表清空并初始化为空链表
template <typename T> void InitList(Linklist<T> &l)

//返回链表长度
template <class T> int LListLength(Linklist<T> &l)

//按位置创建结点并插入数据在指定结点之后
template <class T> void LListInsert(Linklist<T> &l, int n, T &dt)

//通过序号删除结点
template <class T> void LListDeletByNum(Linklist<T> &l, int pos)

//获取结点在指定链表里的坐标并返回
template <class T> int LListGetPos(Linklist<T> &l, LLnode<T> *ln)
```

数据结构设计

```
//玩家保存为一个结构体作为结点的 data。包含玩家的序号和信息
struct person
{
    int num; //玩家序号
    int mm; //玩家信息
};

//结点类型，用于链表使用
template <class T> struct LLnode
{
    T data;          //结点数据
    LLnode *next; //下一节点的指针
    LLnode *prev; //上一节点的指针
};

//链表结构体，引导一个链表,该链表包含一个空的头结点
template <class T> struct Linklist
{
    Linklist() : len(0) {} //保证链表初始长度为 0
    LLnode<T> *head;      //头结点
    LLnode<T> *tail;      //尾结点
    int len;              //链表长度
};
```

三、详细设计

//每个人作为一个结构体。包含自己的序号和信息

```
struct person
```

```
{
```

```
    int num; //玩家序号
```

```
    int mm; //玩家信息
```

```
};
```

```
template <class T>
```

```
struct LLnode
```

```
{
```

```
    T data;          //结点数据
```

```
    LLnode *next; //下一节点的指针
```

```
    LLnode *prev; //上一节点的指针
```

```
};
```

//链表结构体，引导一个链表,该链表包含一个空的头结点

```
template <class T>
```

```
struct Linklist
```

```
{
```

```
    Linklist() : len(0) {} //保证链表初始长度为 0
```

```
    LLnode<T> *head;      //头结点
```

```
    LLnode<T> *tail;      //尾结点
```

```
    int len;              //链表长度
```

```
};
```

//初始化链表，将链表清空并初始化为空链表

```
template <typename T>
```

```
void InitList(Linklist<T> &l)
```

```
{
```

```
    //若链表为非空，清空链表并释放内存
```

```
    if (l.len != 0)
```

```
    {
```

```
        LLnode<T> *p, *tp;
```

```
        p = l.head;
```

```
        //遍历清空并释放空间
```

```
        for (int i = 0; i <= l.len; i++)
```

```
        {
```

```
            tp = p->next;
```

```
            delete p;
```

```
            p = tp;
```

```

    }
}

//创建空链表
LLnode<T> *tp = new LLnode<T>;
l.len = 0;
l.head = tp;
tp->prev = l.head;
l.tail = l.head;
l.head->next = NULL;
}

//返回链表长度
template <class T>
int LListLength(Linklist<T> &l) {
    return l.len;
}

//按位置创建结点并插入数据在指定结点之后
template <class T>
void LListInsert(Linklist<T> &l, int n, T &dt) {
    //若插入位置超过链表长度，报错并结束函数
    if (n > l.len) {
        cout << "LinkList is too short" << endl;
        return;
    }
    else if (n == l.len) {
        LLnode<T> *tp = new LLnode<T>;
        tp->data = dt;
        tp->prev = l.tail;
        l.tail->next = tp;
        tp->next = NULL;
        l.tail = tp;
        l.len++;
        return;
    }

    LLnode<T> *p = l.head;
    //循环至相应节点处
    for (int i = 0; i < n; i++){
        p = p->next;
    }
    //创建新结点插入节点
    LLnode<T> *tp = new LLnode<T>;

```

```

    tp->next = p->next;
    tp->prev = p;
    p->next->prev = tp;
    p->next = tp;
    tp->data = dt;
    l.len++; //链表长度加一
}

```

//通过序号删除结点

```
template <class T>
```

```
void LLDeleteByNum(Linklist<T> &l, int pos){
```

```
    //若插入位置超过链表长度，报错并结束函数
```

```
    if (pos > l.len){
```

```
        cout << "LinkList is too short" << endl;
```

```
        return;
```

```
    }
```

```
    //链表下标从 1 开始算， 小于等于 0 直接退出函数
```

```
    if (pos <= 0){
```

```
        cout << "posision must be greater than 0" << endl;
```

```
        return;
```

```
    }
```

```
    //如果是尾结点，清除即可
```

```
    if (pos == l.len){
```

```
        LLnode<T> *p = l.tail->prev;
```

```
        delete l.tail;
```

```
        l.tail = p;
```

```
        l.len--;
```

```
        l.tail->next = NULL;
```

```
        return;
```

```
    }
```

```
    LLnode<T> *p = l.head;
```

```
    //循环至相应结点
```

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

```
        p = p->next;
```

```
    }
```

```
    LLnode<T> *tp = p->next; //保存应删结点的指针
```

```
    tp->next->prev = p;
```

```
    p->next = tp->next; //当前结点 next 指向应删结点的 next
```

```
    delete tp;          //释放空间
```

```
    l.len--;            // 链表长度减一
```

```
}
```

//获取结点在指定链表里的坐标并返回

```
template <class T>
int LListGetPos(Linklist<T> &l, LLnode<T> *ln){
    LLnode<T> *tp = l.head;
    if(l.len == 1) if(tp->next==ln) return 1;
    for(int i=1;i<l.len;i++){
        tp = tp->next;
        if(tp==ln)
            return i;
    }
    return -1;
}
```

//引导输入信息，创建约瑟夫 h 环链表

```
template <class T>
Linklist<T> *CreatJoseph(int num){
    cout<<"          please input the message of each player : ";
    Linklist<person> *l = new Linklist<person>;
    InitList(*l);
    for (int i = 1; i <= num; i++){
        int n;
        cin >> n;
        person p;
        p.num = i;
        p.mm = n;
        LListInsert(*l, LListLength(*l), p);
    }
    cout<<endl;
    return l;
}
```

//引导输入游戏人数

```
int JosephPlayer(int &n){
    cout << endl;
    cout << "          please input the positive integer of players or other to exit : ";
    cin >> n;
    cout<<endl;
    return n > 0;
}
```

//开始约瑟夫环并引导输入游戏开始的坐标

```
int JosephBegin(){
    int m;
    cout << "          please input the number of beginning of game : " ;
```

```

        cin >> m;
        cout << endl;
        return m;
    }

```

//处理信息并输出出队序列

```

template <typename T>
void JosephOutput(Linklist<T> *l, int m){
    cout<<"          the order of outline is : ";
    LLnode<person> *tp = l->head->next;
    while (LListLength(*l))
    {
        for (int i = 1; i < m; i++)
        {
            if (tp->next)
                tp = tp->next;
            else
                tp = l->head->next;
        }
        m = tp->data.mm;
        cout << tp->data.num << " ";
        int pos = LListGetPos(*l, tp);
        if (tp->next)
            tp = tp->next;
        else
            tp = l->head->next;
        LListDeletByNum(*l, pos);
    }
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<"-----"<<endl;
}

```

```

void JosethMain(){
    int n, m;
    Linklist<person> *l;
    while (JosephPlayer(n))
    {
        m = JosephBegin();
        l = CreatJoseph<person>(n);
        JosephOutput(l,m);
    }
}

```



```
}

int main(){
    JosethMain();
    return 0;
}
```

四、调试分析

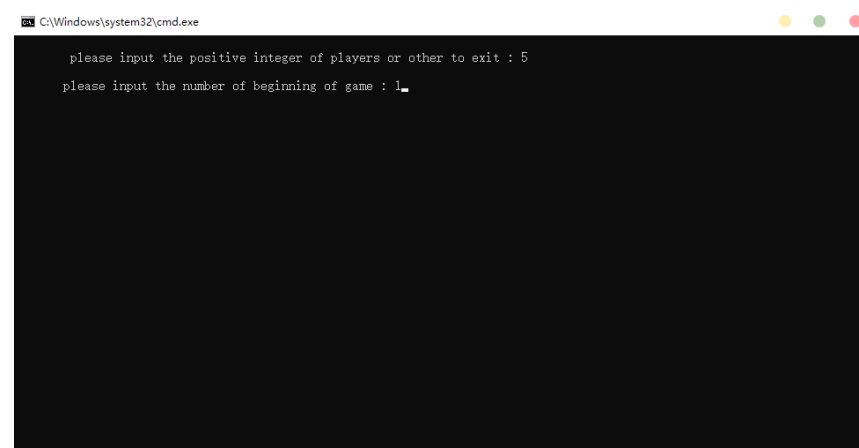
1. 约瑟夫环问题是一个环状问题，考虑到问题的特定，使用循环链表方便处理。
2. 为了保存每个位置的次序和信息，采用了结构体来存储代表玩家的位置
3. 采取了循环链表和循环找坐标的方法，时间复杂度为 $O(n^2)$ 。

五、用户手册

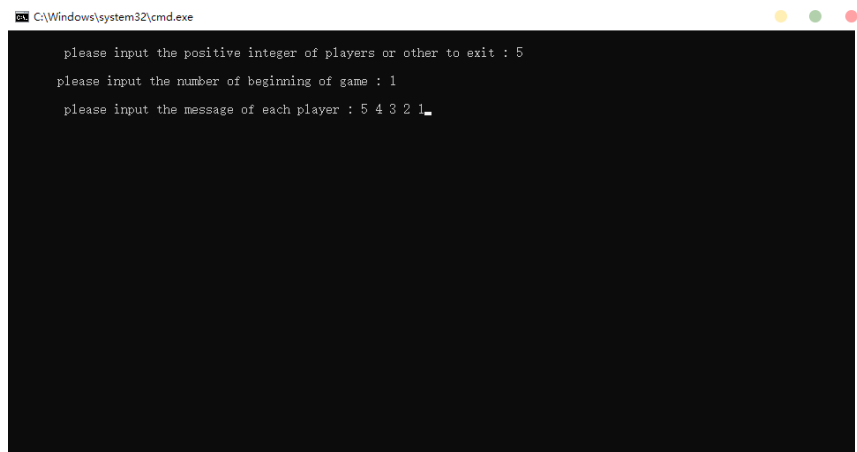
- 1、本程序的执行文件为：joseph.exe
- 2、进入演示程序后，将显示如下的界面, 引导用户输入参与人数



随后出现以下界面，引导输入游戏开始的坐标



然后引导输入每个节点的信息



```
C:\Windows\system32\cmd.exe

please input the positive integer of players or other to exit : 5
please input the number of beginning of game : 1
please input the message of each player : 5 4 3 2 1
```

最后程序输出结果出队序列，并提示下一次输入



```
C:\Windows\system32\cmd.exe

please input the positive integer of players or other to exit : 5
please input the number of beginning of game : 1
please input the message of each player : 5 4 3 2 1
the order of outline is : 1 2 3 4 5
-----
please input the positive integer of players or other to exit : 
```

六、测试结果



```
C:\Windows\system32\cmd.exe

please input the positive integer of players or other to exit : 5
please input the number of beginning of game : 1
please input the message of each player : 5 4 3 2 1
the order of outline is : 1 2 3 4 5
-----
please input the positive integer of players or other to exit : 
```

输入:

5

1

5 4 3 2 1

能正常输出结果且正确

七、附录

源程序文件名清单: joseph.cpp 和 joseph.exe