实习报告

题目：编制一个解决约瑟夫环问题的程序

班级：计科三班　姓名：裴启智　学号：PB18111793　完成日期：2019/10/23

一、实验要求

1. 约瑟夫问题的一种描述是：编号为 1,2，…,n（n<=30）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。
2. 利用单向循环链表存储结构模拟此过程，按照出列的顺序印出个人的编号。
3. 基本要求：使用命令行进行传参（命令行格式为<可执行程序名><人数 n><初始的报数上限><密码 1>…<密码 n>），当除可执行程序名外没有参数时，将继续执行程序并提示用户输入这些参数。输入正确的参数后，将在屏幕上输出相应的结果，并将输出结果导出到文件中（为便于查找和查看，文件保存路径为 C:\Users\dell\Desktop\sequence.txt）。
4. 选做要求：当命令行参数不全或输入不正确时，会报错，需重新运行程序。
5. 选做要求：用顺序表实现约瑟夫环。
6. 测试数据：
   （1）　输入：7 20 3 1 7 2 4 8 4 输出 6 1 4 7 2 3 5
   （2）　输入　6 15 1 5 6 7 8 2　　输出 3 4 1 2 5 6

二、设计思路

①循环链表部分

为实现上述程序功能，应用循环链表表示排成一圈的 n 个人。为此需要抽象数据类型：循环链表

1.循环链表的抽象数据类型定义为：

ADT CircularList{

数据对象：D={ai | ai∈ElemSet,i=1,2,…,n,n>=0}

数据关系：R1={<ai-1,ai> | ai-1,ai∈D, ,i=2,…,n,n>=0}

基本操作：

LinkInsert（a[ ], n ）

操作结果：根据传入的数据建立循环链表，并记录每个人的密码，返回指向头结点的指针。

}

2.本程序包含四个模块：

1）主程序模块

```
int main（）{
  初始化;
  调用函数;
  写入文件;
  }
```

2）循环链表单元模块——实现循环链表抽象数据类型

3）joseph 函数模块——模拟报数，计算出列的顺序

各模块之间的调用关系如下：

②顺序表部分（选做）

为实现上述程序功能，应用顺序表表示排成一圈的 n 个人，用带余除法模拟环中的操作。为此需要抽象数据类型：顺序表

1.顺序表的抽象数据类型定义为：

ADT List{

数据对象：D={ai | ai∈ElemSet,i=1,2,…,n,n>=0}

数据关系：R1={<ai-1,ai> | ai-1,ai∈D, ,i=2,…,n,n>=0}

基本操作：

Delete（a[ ][2] , k ,length）

操作结果：根据传入的数据删除当前顺序表的第 k 个结点

2.本程序包含二个模块：

1）主程序模块

int main（）{

初始化；

while（长度不为 0）{

模拟约瑟夫环；

调用删除函数；}

写入文件；

}

2）Delete 函数模块——删去传入顺序表的第 k 个结点

各模块之间的调用关系如下：

三、详细设计

①循环链表部分

1. 结点类型、指针类型

```
typedef struct LNode{

    int t; //每个人手中的密码

    int num; //每个人的序号

    struct LNode*next;//下一个链表的指针域
}LNode,*LinkList;//结点类型，指针类型
```

2. 有序链表设头结点和尾结点为同一个结点。

```
LinkList LinkInsert(int a[],int n)//根据传入的数组 a 建立链表，并读入每
个人的密码，返回指向头结点的指针。n 为结点的数量（即人数）。
{
    int i;//定义计数变量 i
    LinkList begin=NULL,tail;//定义头结点、尾结点
    LNode *p; //定义指向循环链表结点的指针

    for(i=n-1;i>=0;--i) //通过循环对每一个结点进行如下操作
    {
        p=(LinkList)malloc(sizeof(LNode));//初始化 p 指针
        p->t=a[i];//读入当前结点的密码
        p->num=i+1;//读入当前结点的序号
        p->next=begin;//将 p 的指针域指向头结点
        begin=p;//将头结点指向 p，如此循环倒着建立链表
        if(i==n-1) tail=p;//判断，令最后一个节点为尾结点
    }
    tail->next=begin;//使尾结点指向头结点
    return begin;//返回指向头结点的指针
}
```

3. joseph 函数

```
void joseph(LinkList &L,int b[],int m,int n)//根据循环链表求出出列顺
序，存放在传入的数组 b 中，m 为初始密码，n 为人数（即循环链表的结点数）
{
    int i,j=n;//定义计数变量
    LinkList p,q;//定义指向循环链表结点的指针
    p=L;//p 指向循环链表的头指针
```

```
    b[0]=m%n;//求出第一个出列的人的序号
    while(j!=0)
    {
        for(i=1;i<m;++i)
        {
            q=p;
            p=p->next;

        }//通过循环，使指针 q 指向要删除的 p 节点的前驱节点，p 指向要删除的结
点

        m=p->t;//将 m 设置为新的密码

        b[n-j]=p->num;//将当前结点的序号存入保存结果的数组中

        q->next=p->next;//将当前 p 指向的结点删除

        free(p);//释放 p 结点的内存

        p=q->next;//继续找下一个出列者对应的序号

        j--;
    }
}
```

4. 主函数的算法

```
int main(int argc , char *argv[])//用命令行传参，即 main 函数具有参数，
其中 argc 为输入的数量，argv[0]为程序 exe 文件的路径，argv[1]、argv[2]等等
为输入。不同的输入中间要用空格隔开
{
    int i,n,m;//i 为计数变量，n 为人数，m 为报数上限值

    int code[30];//存放每个人的密码

    int sequence[30];//存放出列的顺序

    LinkList L;//初始化头结点
```

```
    bool flag= false;//定义标记变量

    cout<<"please input parameters"<<endl;//提示输入

    for(i=1;;++i)//输入不符合要求将会报错

    {
        for(int j=0;j<strlen(argv[i]);++j)
        {
            if(!isdigit(argv[i][j]))
            {
                flag=true;

                break;}//如果输入的有字母，则将标记变量置为 true，跳出循环

        }
        if(flag)
        {

            cout<<"illegal input"<<endl;//报错

            break;
        }

        if(i==argc-1) break;//检查完每个输入后跳出循环

    }
    sscanf(argv[1],"%d",&n);

    sscanf(argv[2],"%d",&m);//用 sscanf 将输入格式化转换为整形存入 n 和 m

中

    if(n!=(argc-3))//如果输入的人数不够，则报错

    {
        cout<<"illegal input"<<endl;
        exit(0);
    }

    if(n>30||n<0)//如果输入的人数超过 30 或为负数，则报错

    {
        cout<<"illegal input"<<endl;
        exit(0);
    }

    if(m<0)//若初始密码为负数，则报错

    {
```

```
        cout<<"illegal input"<<endl;
        exit(0);
    }
    for(i=3;i<=argc;++i)//将每个人的密码读入 code 数组中，以便于后续操作
    {
        sscanf(argv[i],"%d",&code[i-3]);//用 sscanf 将输入格式化转换为整
形存入 code[i-3]中
        if(code[i-3]<0){//如果某个人的密码为负数，则报错
            cout<<"illegal input"<<endl;
            exit(0);
        }
    }
    L=LinkInsert(code,n);//调用 LinkInsert 函数初始化循环链表，链表长度
为 n
    joseph(L,sequence,m,n);//调用 joseph 函数计算出列顺序，存入 sequence
数组中，m 为初始密码，n 为人数
    FILE *fp;//定义文件指针

if((fp=fopen("C:\\Users\\dell\\Desktop\\sequence.txt","w"))==NULL)//
若文件打开失败，则报错，文件路径为 C:\Users\dell\Desktop\sequence.txt
    {
        cout<<"file cannot open"<<endl;
        exit(0);
    }
    for(i=0;i<n;++i)
    {
        cout<<sequence[i]<<" ";//输出出列的顺序到屏幕
        fprintf(fp,"%d ",sequence[i]);//输出出列的顺序到 sequence.txt
文件
    }
```

```
    fclose(fp);//关闭文件
    return 0;
}
```

②顺序表实现（选做）

1 .全局变量 length

```
int length;//定义数组当前的长度，由于分函数会改变长度，故定义为全局变量
```

2. Delete 函数

```
   void Delete(int a[][2],int k,int length){//删除保存密码的数组中的元素，k 为
要删除的位置，length 为数组的当前长度
    for(int i=k;i<length;++i){//删除顺序表中的某个节点，将后面的结点全部向前移动
一位
        a[i-1][0]=a[i][0];
        a[i-1][1]=a[i][1];
    }
}
```

3.主函数

```
int main(int argc,char *argv[]) {//用命令行传参，即 main 函数具有参数，其中 argc
为输入的数量，argv[0]为程序 exe 文件的路径，argv[1]、argv[2]等等为输入。不同的输
入中间要用空格隔开
    int i,m,n,num=1;//定义计数变量 i，初始密码 m，人数（顺序表的长度）n，当前序号
num
    int code[30][2],sequence[30];//定义存入二维数组 code[30][2],用来对应的密码
和序号，定义数组 sequence 数组存放出列的顺序
    bool flag= false;//定义标记变量
    cout<<"please input parameters"<<endl;//提示输入
    for(i=1;;++i)//输入不符合要求将会报错
```

```
    {
        for(int j=0;j<strlen(argv[i]);++j)
        {
            if(!isdigit(argv[i][j]))
            {
                flag=true;
                break;}//如果输入的有字母，则将标记变量置为 true，跳出循环
        }
        if(flag)
        {
            cout<<"illegal input"<<endl;//报错
            break;
        }
        if(i==argc-1) break;//检查完每个输入后跳出循环
    }
    sscanf(argv[1],"%d",&n);
    sscanf(argv[2],"%d",&m);//用 sscanf 将输入格式化转换为整形存入 n 和 m 中
    if(n!=(argc-3))//如果输入的人数不够，则报错
    {
        cout<<"illegal input"<<endl;
        exit(0);
    }
    if(n>30||n<0)//如果输入的人数超过 30 或为负数，则报错
    {
        cout<<"illegal input"<<endl;
        exit(0);
    }
    if(m<0)//若初始密码为负数，则报错
    {
        cout<<"illegal input"<<endl;
        exit(0);
    }
    for(i=3;i<=argc;++i)//将每个人的密码读入 code 数组的第二维的第一位中，以便于
后续操作
    {
```

```
        sscanf(argv[i],"%d",&code[i-3][0]);//用 sscanf 将输入格式化转换为整形存
入 code[i-3][0]中
        if(code[i-3][0]<0){//如果某个人的密码为负数，则报错
            cout<<"illegal input"<<endl;
            exit(0);
        }
    }


    for(i=0;i<n;++i){
        code[i][1]=i+1;
    }//将每个人的序号读入 code 数组的第二维的第二位中，以便于后续操作
    length=n;//将 length 置为 n。（length 为当前顺序表长度，后续会发生改变）
    //m 为当前密码，length 为当前顺序表长度，num 为当前需要出列的序号，n 为总长度
（固定）
    while(length!=0){//通过循环进行如下的操作
        if(m%length==0){
            num+=length-1;//若当前密码恰好为当前人数的倍数，则返回当前顺序表的最
后个元素，注意序号需要-1
        }
        else{
            num+=m%length-1;//否则，计算下一个出列的人的序号（用带余除法模拟环状
结构），注意序号需要-1
        }
        if(num>length) num%=length;//如果当前出列的序号超过了当前长度，则除以当
前长度并取余数即为当前出列的序号
        sequence[n-length]=code[num-1][1];//将出列序号存入存放结果的序列中
        m=code[num-1][0];//将 m 设为新的密码
```

```
        Delete(code,num,length);//调用 Delete 函数删去顺序表的当前节点

        --length;//长度-1
    }

    FILE *fp;//定义文件指针

    if((fp=fopen("C:\\Users\\dell\\Desktop\\sequence.txt","w"))==NULL)//若文
件打开失败，则报错，文件路径为 C:\Users\dell\Desktop\sequence.txt
    {
        cout<<"file cannot open"<<endl;
        exit(0);
    }

    for(i=0;i<n;++i){

        cout<<sequence[i]<<" ";//输出出列的顺序到屏幕

        fprintf(fp,"%d ",sequence[i]);//输出出列的顺序到 sequence.txt 文件
    }
    fclose(fp);//关闭文件


    return 0;
}
```

四、调试分析

1. 刚开始建立链表和读入数据是分开进行的，后来经过调试发现可以把它们综合在一起，这样更加高效。
2. 在用循环链表实现时，刚开始没有在结构体里定义序号这一数据域，导致得到当前人的序号很麻烦，需要推导数学公式来进行计算。后来在结构体中加入序号这一数据域后便可以直接将序号存入 sequence 数组。这也启发了我在选做时采用二维数组存放序号和密码，避免复杂的数学推导和运算。
3. 初始化指针之后要记得释放空间，malloc、realloc 和 free 函数要成对使用。
4. 在调试命令行的输入时要选择合适的方法将字符串转化为整形数字，同时要理解 argc、argv[ ]的含义。
5. 在进行判断输入是否正确时，要考虑全各种情况以及在什么情况下要跳出循环，否则容易出现死循环的情况。
6. 在用顺序表实现时，运用带余除法模拟环状操作时要注意恰好整除的情况，以及什么时候需要+1 或者-1。带余除法可实现从表尾到表头的操作。

7. 用顺序表实现时，要注意区分当前长度和总长度，否则容易出现一些意想不到的错误。
8. 后来发现若用指针对顺序表进行操作，可能会省去改变当前表长的麻烦。
9. 时空复杂度分析

①循环链表

LinkInsert 函数实现对链表的初始化和插入数据，由于链表长度为 n，故时间复杂度为 O（n）。不需要辅助空间，故空间复杂度为 O（1）。

Joseph 函数通过循环找到要出列的结点，由于循环的次数和当前的 m 有关，故时间复杂度为 O（max（m））。不需要辅助空间，故空间复杂度为 O（1）。

主函数中判断不符合要求的输出时需要检查从 1 开始的每一个 argv[i],故时间复杂度为 O（argc）。由于数组 code 和 sequence 长度固定，不需要辅助空间，故空间复杂度为 O(1)。

②顺序表

Delete 函数从长为 length 的数组中删去第 k 个元素，时间复杂度为 O（length）。不需要辅助空间，故空间复杂度为 O（1）。

主函数中判断不符合要求的输出时需要检查从 1 开始的每一个 argv[i],故时间复杂度为 O（argc）。由于数组 code 和 sequence 长度固定，不需要辅助空间，故空间复杂度为 O(1)。

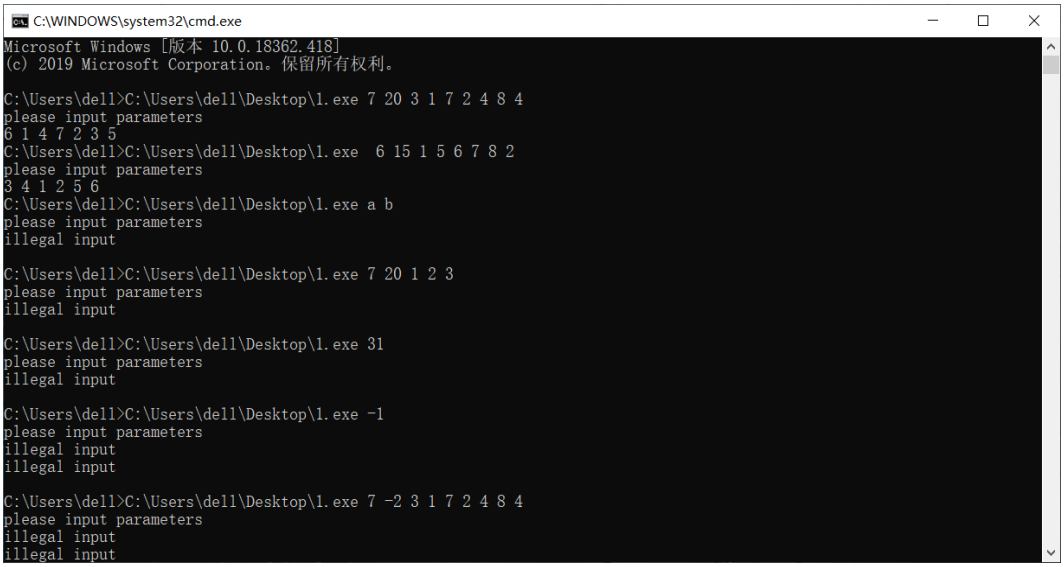
主函数中模拟约瑟夫环的部分循环需要执行 n 次，故时间复杂度为 O(n)。且不需要辅助空间，故空间复杂度为 O（1）。

五、代码测试

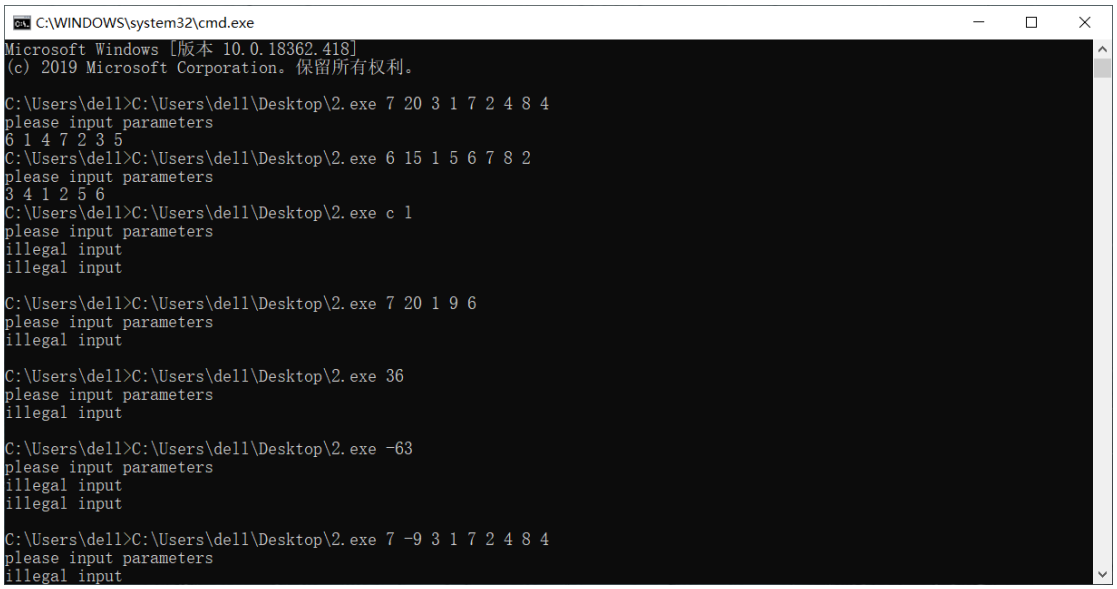
采用命令行输入，在桌面上按开始键+R，在运行框中输入 cmd，即得到命令窗口。输入 exe 文件的路径，后面输入相应的数据，中间用空格隔开。

分别进行两组正确的输入和错误的输入（输入字母、负数、前后数量不匹配、超过 n 的最大数量等）

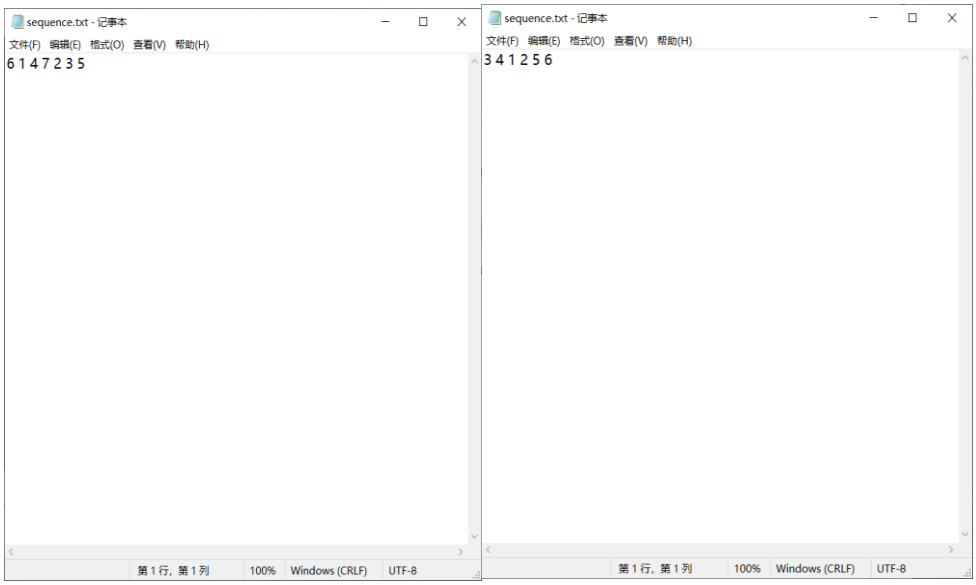
1. 循环链表



2. 顺序表



C:\Users\dell\Desktop\sequence.txt 路径下的 sequence.txt 文件如下图



六、实验总结

通过这次实验，我了解了如何用循环链表和顺序表解决约瑟夫环问题。通过用循环链表以及用顺序表和带余除法模拟环状结构，使我对线性表的一些基本操作如构建、插入、删除等等更加熟悉。在一遍一遍地调试代码中，也发现了很多小的问题，如怎么设置循环控制变量，表长是否需要改变，如何判断输入是否符合要求，如何使用命令行传参以及将字符串转化为数字等等，这些都使我收获颇丰。

在写实验报告时，我又把之前的代码重新解释了一遍，再次深入理解了一些小的细节，使自己的思路更加清晰，也巩固加深了我对这类问题的认识。

在选做部分自己花了比必做更多的时间，但从循环链表中得到了启示，用二维数组记录每个人的序号，省去了在删除等操作时重新计算序号的麻烦。但在处理带余除法运算时遇到了一些困难，需要考虑整除以及当前表的长度不断变化所带来的一系列问题。后来发现通过指针来进行操作而不是在原表上进行操作会更加方便。

通过解决上述问题，我更加熟悉了顺序表及其相关操作。

七、附录

源程序文件名清单

#include<cstdio>	C 标准库中的头文件
#include<cstdlib>	提供一些函数与符号常量
#include <cstring>	用于封装字符串数据结构
#include <iostream>	用于输入输出的 iostream 类库