Assignment_2 约瑟夫生死游戏

一、项目简介：

约瑟夫生者死者游戏的大意是：30个旅客同乘一条船，因为严重超载，加上风高浪大危险万分；因此船长告诉乘客，只有将全船一半的旅客投入海中，其余人才能幸免于难。无奈，大家只得统一这种方法，并议定30个人围成一圈，由第一个人开始，依次报数，数到第9人，便将他投入大海中，然后从他的下一个人数起，数到第9人，再将他投入大海，如此循环，直到剩下15个乘客为止。问哪些位置是将被扔下大海的位置。

本游戏的数学建模如下：假如N个旅客排成一个环形，依次顺序编号1, 2, ..., N。从某个指定的第S号开始。沿环计数，每数到第M个人就让器出列，且从下一个人开始重新计数，继续进行下去。这个过程一直进行到剩下K个旅客为止。

本游戏要求用户输入的内容包括：1、 旅客的个数，也就是N的值；2、 离开旅客的间隔书，也就是M的值；3、 所有旅客的序号作为一组数据要求存放在某种数据结构中。

本游戏要求输出的内容是包括：1. 离开旅客的序号；2. 剩余旅客的序号。

二、构建设想：

因为是乘客要构成一个环形，所以很容易的就让人联想到了直接用循环链表来解决这个问题，然后要注意到循环链表的插入删除所需要注意的选项就可以实现系统的大部分功能了。最后注意一些异常情况的拍错，提高程序的健壮性，优化代码的可读性。

三、程序设计：

1. 程序的类与结构组织

约瑟夫生死游戏一共有两个类，因为要使用链表，所以首先要做的是对于链表的节点进行定义，如下代码块就是对链表节点类的定义,其代表了整个乘客的数据：

```
//每一个乘客
class Passenger{
public:
    int         _position;    //乘客当前的位置
    Passenger*  _next;        //下一位乘客
public:
    Passenger(int position): _position(position){} //初始化一个乘客的位置
};
```

第二个类就是DeadGame类，其实现循环链表的初始化，插入与删除功能，并且最后将还在活着的人可以打印出来，如下是类的主要代码

```
//游戏类，主要由循环链表存储实现
class DeadGame{
private:
    int           _deadCountM;    //死亡人数
    int           _stillAlive;    //还活着的人的人数
    Passenger*    _initPos;       //最初的位置
    Passenger*    _curPos;        //目前的位置

public:
    DeadGame(int N, int beginPosition,int M);    //初始化整个游戏
    void showAlive();                            //展示还活着的人的序号
    void killHim(Passenger* prePos);             //kill this man!
    Passenger* findHim();                        //找到要死的人
    int getN() { return _stillAlive; }           //返回还活着的人数

};
```

2.程序类中函数的实现

(1) 构造函数 DeadGame():

构造函数的功能主要是构建一个初始化的循环链表,利用一个循环来初始化链表，并且对乘客的位置从1开始标注到结束,如下是具体的代码实现:

```
//初始化整个游戏
DeadGame::DeadGame(int N, int beginPosition,int
M):_deadCountM(M),_stillAlive(N)
{
    _initPos = new Passenger(1);
    auto begin = _initPos;
    _curPos = _initPos;

    for(int i = 2; i <= N; i++)
    {
        auto temp = new Passenger(i);
        if(i == beginPosition)
            _initPos = temp;

        _curPos->_next = temp;
        _curPos = temp;
        _curPos ->_next = nullptr;
    }

    _curPos->_next = begin;
    _curPos = _initPos;
}
```

(2) 查找函数 findHim() :

查找函数的主要思路是把链表从开始的位置开始按照M计数来找到下一个要被推下船的人，找到的了之后就返回这个人所在的位置(返回一个指针),如下是代码的具体实现:

```
//找到要死的人
Passenger* DeadGame::findHim()
{
    Passenger* p;
    for(int i = 0; i < _deadCountM - 1; i++)
    {
        p = _curPos;
        _curPos = _curPos->_next;
    }
    return p;
}
```

(3) 删除函数 killHim() :

这个函数的作用主要是承接上一个函数,在发现要被推出的人的位置之后，根据地址将链表相应的节点删除，如下是具体实现的代码:

```
//kill this man!
void DeadGame::killHim(Passenger *prePos)
{
    auto p = _curPos;

    prePos->_next = _curPos->_next;
    _curPos = _curPos->_next;
    cout << p->_position << endl;

    _stillAlive--;
    delete p;
}
```

(4) 打印函数 showAlive():

在游戏结束之后要做的就是展示还活着的乘客的人数,以及他们所在的序号，大体上就是将链表遍历一遍，但是值得注意的是要从最小的开始输出，如下是程序的具体实现代码:

```
//展示还活着的人的序号
void DeadGame::showAlive()
{
    //找到序号最小的存活乘客
    while(_curPos->_position < _curPos->_next->_position)
        _curPos = _curPos->_next;

    _curPos = _curPos->_next;
    for(int i = 0; i < this->_stillAlive; i++)
    {
        cout << _curPos->_position << "\t";
        _curPos = _curPos->_next;
    }
    cout << endl;
}
```

3.主函数main()的实现:

主函数主要承载一个输入输出的功能，在每删除一个人的时候要打印出死亡的信息，如下是主函数的具体代码的实现:

```
int main(int argc, const char * argv[])
{
    cout <<"现有N个人围成一圈，从第S个人开始依次报数，报M的人出局，再由下一人开始报数，如此循环，直至剩下K个人为止"<<endl;
    cout << endl;


    int N,S,M,K;
    cout << "请输入生死游戏的总人数N: " ;
    cin >> N;
    cout << "请输入游戏开始的位置S: " ;
    cin >> S;
    cout << "请输入死亡数字M: " ;
    cin >> M;
    cout << "请输入剩余的生者人数K: " ;
    cin >> K;
    cout << endl << endl << endl;
    if(N <= 0 || S <= 0 || M <= 0 || S > N || K < 0)
    {
        cout << "输入不符合条件,请重新开始" << endl;
        return 0;
    }


    DeadGame game(N,S,M);
    int i = 1;
    while(game.getN() > K)
    {
        auto prePosition = game.findHim();
        cout << "第" << i << "个死者的位置是: \t";
        game.killHim(prePosition);
        i++;
    }
    cout << endl << endl << endl;
    cout << "最后剩下:\t\t" << game.getN() <<"人" << endl;
    cout << "剩余的生者位置是为:";
    game.showAlive();

    return 0;
}
```

四、程序运行截图：

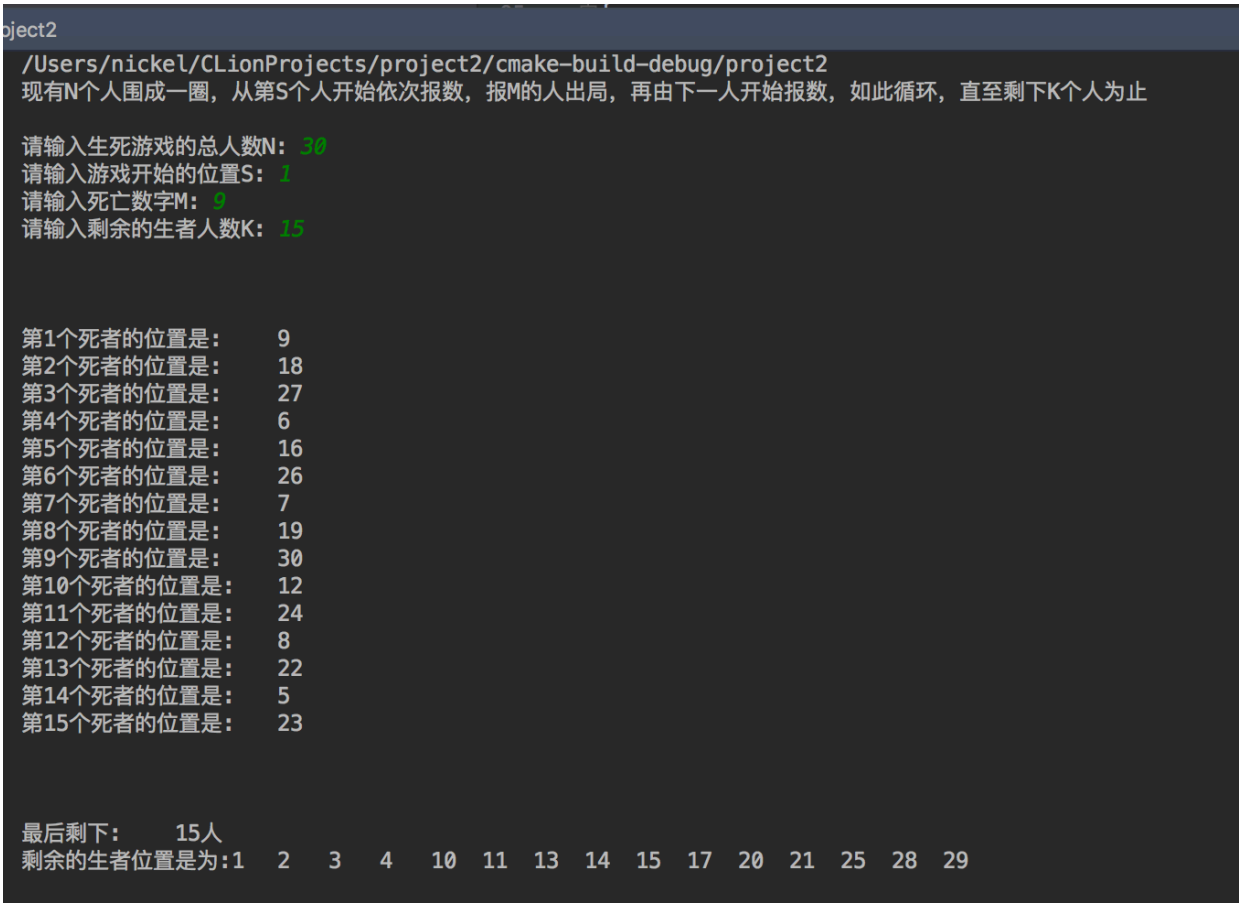
1.运行测试1：

生死游戏总人数N: 30

游戏开始的位置S: 1

死亡数字M: 9

剩余生者人数K: 15



2.运行测试2：

生死游戏总人数N: 100

游戏开始的位置S: 20

死亡数字M: 5

剩余生者人数K: 70

```
请输入生死游戏的总人数N: 100
请输入游戏开始的位置S: 20
请输入死亡数字M: 5
请输入剩余的生者人数K: 70



第1个死者的位置是:    24
第2个死者的位置是:    29
第3个死者的位置是:    34
第4个死者的位置是:    39
第5个死者的位置是:    44
第6个死者的位置是:    49
第7个死者的位置是:    54
第8个死者的位置是:    59
第9个死者的位置是:    64
第10个死者的位置是:   69
第11个死者的位置是:   74
第12个死者的位置是:   79
第13个死者的位置是:   84
第14个死者的位置是:   89
第15个死者的位置是:   94
第16个死者的位置是:   99
第17个死者的位置是:   4
第18个死者的位置是:   9
第19个死者的位置是:   14
第20个死者的位置是:   19
第21个死者的位置是:   25
第22个死者的位置是:   31
第23个死者的位置是:   37
第24个死者的位置是:   43
第25个死者的位置是:   50
第26个死者的位置是:   56
第27个死者的位置是:   62
第28个死者的位置是:   68
第29个死者的位置是:   75
第30个死者的位置是:   81



最后剩下:    70人
剩余的生者位置是为:1   2   3   5   6   7   8   10  11  12  13  15  16  17  18
```

五、程序容错性测试：

1.容错性测试1：

生死游戏总人数N: 0

游戏开始的位置S: 0

死亡数字M: 0

剩余生者人数K: 0

2.容错性测试2：

生死游戏总人数N: 15

游戏开始的位置S: 1

死亡数字M: 0

剩余生者人数K: 0

3.容错性测试3(S > N)：

生死游戏总人数N: 10

游戏开始的位置S: 15

死亡数字M: 5

剩余生者人数K: 1

4.容错性测试4(M > N)：

生死游戏总人数N: 10

游戏开始的位置S: 5

死亡数字M: 15

剩余生者人数K: 5