约瑟夫生者死者游戏

1.项目背景简介

> 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。

2.项目功能要求(要求采用单循环链表)

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：
- 旅客的个数，也就是N的值
- 离开旅客的间隔书，也就是M的值
- 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
- 离开旅客的序号
- 剩余旅客的序号

3.项目核心代码及功能介绍

- 创建的类
    - Person
      单个人信息单元做为链表的节点

```
class Person{
public:
 Person(){}
 ~Person(){}
public:
 int number;                    //每个人的序号
 Person* next;                  //指向下一个人
};
```

    - People
      保存链表头节点和尾节点以及输入信息，并提供成员函数接受输入并模拟过程输出结果

```
class People{
public:
 People();
 ~People(){};
 void Game();                   //开始进行模拟生死游戏
 bool Check();                  //检查初始输入是否合理

public:
 int N;                         //开始时参与该游戏总人数
 int S;                         //开始时从第S个人开始模拟
 int M;                         //每数到第M个人就让该人出列
 int K;                         //最后剩余K人时输出剩余人信息
```

```
        Person* Head;
        Person* Rear;
    };
```

- 接受输入函数（在构造People时进行）
  对于这个函数我采用了非常严谨的输入错误检查机制以免后期模拟时出现错误，这个检查函数为Check函数，在下一个模块介绍。然后根据输入构造一个循环单链表，模拟初始的N个人

```
People::People() {
    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，再由下一个人开始报数。如此"
          "循环，直至只剩下K个人为止"<<endl;
    bool check;
    do{
        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;
        check=Check();
    }while(!check);
    /*构造一个循环单链表*/
    auto ptr1=new Person;
    ptr1->number=1;
    Head=ptr1;
    for(int i=2;i<=N;i++){
        auto ptr2=new Person;
        ptr2->number=i;
        ptr1->next=ptr2;
        if(i==N){
            ptr2->next=Head;
            Rear=ptr2;
        }
        ptr1=ptr2;
    }
}
```

- 检查输入函数
  这里考虑了6种非正常情况：
    - 总人数N小于0
    - 开始位置S小于0
    - 跳过人数M小于0
    - 剩余人数K小于0
    - 开始位置S大于总人数N
    - 剩余人数K大于总人数N

```
bool People::Check(){
    if(N<=0){
        cout<<"The total number N should be positive!"
              "Please REINPUT YOUR DATA!"<<endl;
        return false;
    }
    else if(S<=0){
        cout<<"The begin point S should be positive!"
              "Please REINPUT YOUR DATA!"<<endl;
        return false;
    }
    else if(M<=0){
        cout<<"The dead number M should be positive!"
              "Please REINPUT YOUR DATA!"<<endl;
        return false;
    }
    else if(K<=0){
        cout<<"The residual number K should be positive!"
              "Please REINPUT YOUR DATA!"<<endl;
        return false;
    }
    else if(S>N){
        cout<<"The begin point S should less than "
              "the total number N!"
```

```
                "Please REINPUT YOUR DATA!"<<endl;
        return false;
    }
    else if(K>N){
        cout<<"The residual number K must less than "
                "the total number N!"
                "Please REINPUT YOUR DATA!"<<endl;
        return false;
    }
    else{
        return true;
    }
}
```

- 模拟游戏函数
  这里我的想法是用两个指针，一个指向当前位置，一个指向当前位置的前一个，因为是单链表，所以无法通过当前位置直接获得它的前驱，所以我就加了一个指针让二者同时迭代，这样每次删除元素链接链表时就十分方便了。

```
void People::Game(){
    auto ptr=Head;
    auto ptrFormer=Head;            //指向当前位置的前一个人
    if(S==1){                       //如果当前初始位置为1，那么前一个人就是最后一个
        ptrFormer=Rear;
    }
    else{                           //如果不是就顺序遍历到第S-1位置
        for(int i=0;i<S-2;i++){
            ptrFormer=ptrFormer->next;
        }
    }
    for(int i=0;i<S-1;i++){         //将ptr指向初始位置也就是S位置
        ptr=ptr->next;
    }
    for(int i=1;i<=N-K;i++){        //进行N-K次模拟删除节点
        for(int j=0;j<M-1;j++){     //模拟数到第M个人(将ptr和ptrFormer一起迭代)
            ptr=ptr->next;
            ptrFormer=ptrFormer->next;
        }
        auto ptrNext=ptr->next;     //找到当前位置的下一个元素
        ptrFormer->next=ptrNext;    //将当前位置的前一个元素链接至当前位置的下一个元素
        cout<<"第"<<i<<"个死者的位置是: "<<(ptr->number)<<endl;
        if(ptr==Head){              //如果当前位置为头节点就更新头节点的值（因为最后输出时要用到head）
            Head=ptr->next;
        }
        delete(ptr);                //删除该位置节点
        ptr=ptrNext;                //继续模拟操作
    }
    cout<<"最后剩下"<<K<<"人"<<endl;
    cout<<"剩余的生者位置为";
    auto alive=Head;
    for(int i=0;i<K;i++){
        cout<<alive->number<<"  ";
        alive=alive->next;
    }
    cout<<endl;
}
```

4.项目演示

```
/Users/kirito/CLionProjects/untitled/cmake-build-debug/untitled
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，再由下一个人开始报数。如此循环，直至只剩下K个人为止
请输入生死游戏的总人数N:30
请输入游戏开始的位置S:1
请输入死亡数字M:9
请输入剩余的生者人数K:15
第1个死者的位置是: 9
第2个死者的位置是: 18
第3个死者的位置是: 27
第4个死者的位置是: 6
第5个死者的位置是: 16
第6个死者的位置是: 26
```

```
第7个死者的位置是: 7
第8个死者的位置是: 19
第9个死者的位置是: 30
第10个死者的位置是: 12
第11个死者的位置是: 24
第12个死者的位置是: 8
第13个死者的位置是: 22
第14个死者的位置是: 5
第15个死者的位置是: 23
最后剩下15人
剩余的生者位置为1  2  3  4  10  11  13  14  15  17  20  21  25  28  29
press any key to continue:
```