

**课 程 实 验 报 告**

**课程名称：面向对象程序设计**

**实验名称：继承对象的整型队列编程**

**院 系 ：计算机科学与技术**

**专业班级 ： CS1503班**

**学 号 ： U201514559**

**姓 名 ： 周铭昊**

**指导教师 ： 纪俊文**

**2017 年 11 月 28 日**

1. **需求分析**
2. **题目要求**

**整型队列是一种先进后出的存储结构，对其进行的操作通常包括判断队列是否为空、向队列顶添加一个整型元素、出队列等。整型队列类型及其操作函数采用面向对象的C++语言定义，请将完成上述操作的所有函数采用C++编程。然后写一个main函数对队列的所有操作函数进行测试，要求main按照《关于C++实验自动验收系统说明》给定的方式工作。注意，请用实验三的SATCK继承形成新的类QUEUE。分析说明除构造函数以外的函数，加virtual说明与不加virtual说明有无区别。并说明为什么不将s2也作为基类。**

**class QUEUE：public STACK{**

**STACK s2;**

**public:**

**QUEUE(int m); //每个栈最多m个元素，要求实现的队列最多能入2m个元素**

**QUEUE(const QUEUE&s); //用队列s拷贝初始化队列**

**virtual operator int ( ) const; //返回队列的实际元素个数**

**virtual int full ( ) const; //返回队列是否已满，满返回1，否则返回0**

**virtual int operator[ ](int x)const; //取下标为x的元素，第1个元素下标为0**

**virtual QUEUE& operator<<(int e); //将e入队列,并返回队列**

**virtual QUEUE& operator>>(int &e); //出队列到e,并返回队列**

**virtual QUEUE& operator=(const QUEUE&s); //赋s给队列,并返回被赋值的队列**

**virtual void print( ) const; //打印队列**

**virtual ~QUEUE( ); //销毁队列**

**};**

**注意：虽然实现的队列 最多能入2m个元素，但是由于模拟中入出操作序列的限制，即操作不能违背先进先出的原则，可能还没入2m个元素队列就“满”了，所以要注意full( )函数的实现：不能简单判断队列是否装了2m个元素。**

**在完成上述程序及测试无误后，请使用队列解决如下舞伴问题，此时main用非命令行的方式工作。假定在一次舞会上，男士排成一队，女士排成另一队。每次舞曲响起时，从男队和女队的队头各出一人，配成舞伴跳完此曲，跳完后各自进入自己队列的尾部。若男女两队的初始人数分别为M和F（M和F均为素数，且M!=F），男队中排在位置m（m<=M）的男士，非常想和女队位置f（f<=F）的女士跳舞，问他在第几支曲舞曲才能和该女士跳舞?请编程解决上述问题。**

1. **需求分析**

题目有两种工作方式，因此在main函数通过判断命令行是否有参数来选择执行功能1—栈模拟队列 还是 功能2—舞伴问题。同时需要输出文件，输出的txt文件名与可执行文件名相同。

1. **系统设计**
2. **概要设计**

设计思路为将整个程序分为两个大模块：栈模拟队列、舞伴问题。栈模拟队列模块又可以细分为栈类模块、构造队列模块、入队列模块、出队列模块、队列赋值模块、输出模块，总体结构如图2.1所示

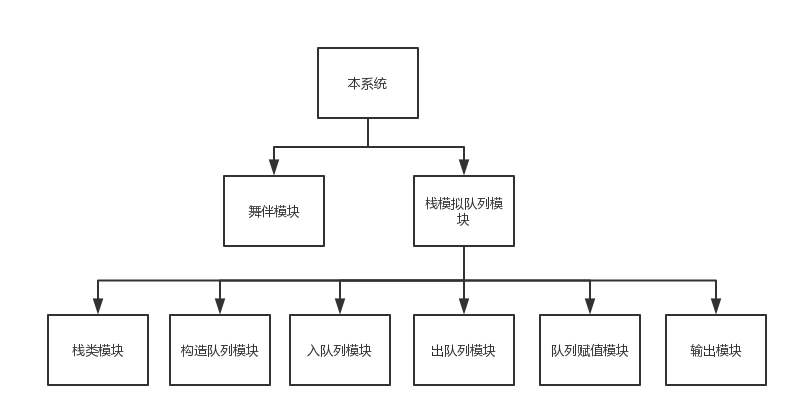


图2.1 系统总体结构图

总体流程为首先判断命令行是否有参数，无则执行舞伴问题，有则执行栈模拟队列问题。舞伴问题求解过程为：先输入具体参数，建立队列模型求解。栈模拟队列问题求解过程为：根据命令行参数选择功能执行对应模块，执行完一个功能后调用输出模块，如此循环直到命令行中所有命令都被执行完。总体流程如图2.2所示

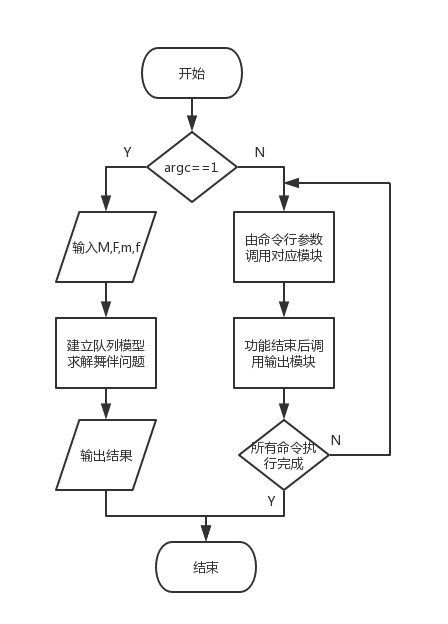


图2.2 系统总体流程图

1. **详细设计**
   1. 舞伴模块

本模块的功能为处理舞伴问题。

首先输入男队长度M、女队长度F、男士位置m、女士位置f，输入完成后构造一个长度为M的队列Mqueue代表男队，再构造一个长度为F的队列Fqueue代表女队，然后按照顺序入队，用1标识普通男性女性，用2标识选定位置的男士女士。在一个循环中，男队出队一个到male，女队出队一个到female，当male=female=2时，跳出循环并输出此时的dance(初始值为1)，否则继续循环且dance加1。流程图如图2.3所示

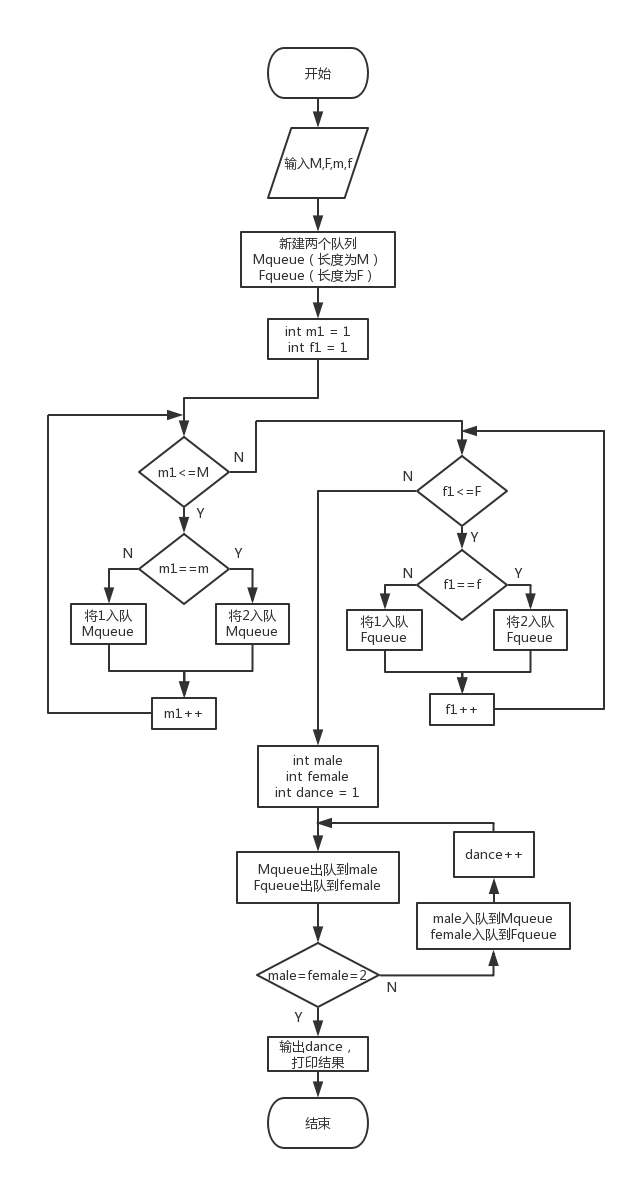


图2.3 舞伴模块流程图

* 1. 栈类模块

本模块的功能为声明一个如下的栈类STACK并定义其成员函数功能：

class STACK{

int \*const elems; //申请内存用于存放栈的元素

const int max; //栈能存放的最大元素个数

int pos; //栈实际已有元素个数，栈空时pos=0;

public:

STACK(int m); //初始化栈：最多m个元素

STACK(const STACK&s); //用栈s拷贝初始化栈

virtual int size ( ) const; //返回栈的最大元素个数max

virtual operator int ( ) const; //返回栈的实际元素个数pos

virtual int operator[ ] (int x) const; //取下标x处的栈元素

virtual STACK& operator<<(int e); //将e入栈,并返回栈

virtual STACK& operator>>(int &e); //出栈到e,并返回栈

virtual STACK& operator=(const STACK&s); //赋s给栈,并返回被赋值的栈

virtual void print( ) const; //打印栈

virtual ~STACK( ); //销毁栈

};

队列类QUEUE继承于栈类STACK，同时还有一个STACK类型的成员s2。

* 1. 构造队列模块

本模块功能为构造新队列：

第一种方式是类QUEUE的构造函数，传入参数为整形数m，用初始化列表来调用父类STACK的构造函数以及成员变量s2的构造函数。

第二种方式是类QUEUE的拷贝构造函数，传入参数为QUEUE类型的引用，初始化列表来调用父类STACK的拷贝构造函数以及成员变量s2的拷贝构造函数。

* 1. 入队列模块

本模块功能为把一个整形元素入队列：

首先判断该队列中继承于父类STACK的栈是否已经满，如果不满，直接入栈。如果满了，再判断成员变量s2是否为空栈，如果为空栈，则将该队列继承的栈中的所有元素倒入s2中，再入栈。如果该队列继承的栈已满，s2也不为空栈，则返回一个长度为0的队列，代表队列已满，无法入队。

* 1. 出队列模块

本模块功能为把一个整形元素出队列：

首先判断栈s2是否为空，如果不为空，直接将s2中的栈顶元素出栈；如果s2为空栈，本队列继承的栈不为空，则把该队列继承的栈中的所有元素倒入s2中，再从s2出栈。其他情况返回一个长度为0的队列，代表队列为空，无法出队。

* 1. 队列赋值模块

本模块功能为把一个队列赋值给另一个队列：

通过调用栈类STACK重载的运算符 = 来实现赋值。

* 1. 输出模块

本模块功能为输出操作结果并同时打印到文件和屏幕上：

首先判断完成的操作。如果为I O C A操作，操作结束后按照顺序输出队列中的所有元素；如果为N操作，结束后输出当前队列剩余元素个数；如果为G操作，输出队列中位置为x的元素。

* 1. 分析说明除构造函数以外的函数，加virtual说明与不加virtual说明有无区别。并说明为什么。

当函数加上virtual说明时（动态链编），代表该函数为虚函数，子类继承的函数为虚函数时，若子类对父类的虚函数进行函数重写，在运行时根据具体对象的不同执行不同的函数。而不加virtual说明时（静态链编），只调用父类的成员函数，无法调用子类中重写的函数，因而不能实现多态。

实际上当类中声明虚函数时，编译器会在类中生成一个虚函数表，存储的是类中的virtual成员函数指针。而用类定义对象时，C++编译器会在对象中添加一个vptr指针（可以通过sizeof类对象发现，加了virtual说明之后对象会增加4个字节，即为vptr指针所占用的内存空间） ，该指针指向虚函数表，在调用虚函数时，编译器会根据对象的vptr指针，在所指向的虚函数表中查找对应函数并调用，因此在子类对象调用某虚函数时可以执行子类中重写的虚函数，而不是父类中定义的函数。

不将s2也作为基类的原因：因为已经有一个基类STACK了，而s2的类型也为STACK，一个子类不能同时继承两个相同的父类，会产生二义性。

1. **软件开发**

采用Windows10开发环境，开发使用IDE为VS2017 ，在其中编译链接生成可执行文件。

1. **软件测试**

4.1 舞伴问题

首先按照提示输入四个参数M、F、m、f，单击回车后显示问题答案，测试样例如下表4.1所示，测试结果如图4.1所示，程序正确运行。

表4.1 舞伴问题测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 男队人数M | 女队人数F | 男士位置m | 女士位置f | 正确答案 | 实际答案 |
| 5 | 3 | 1 | 1 | 1 | 1 |
| 5 | 3 | 3 | 2 | 8 | 8 |
| 7 | 5 | 4 | 4 | 4 | 4 |
| 13 | 11 | 7 | 5 | 137 | 137 |

|  |  |
| --- | --- |
| C:\Users\lnsqr\Desktop\CPP_PROJECT\REPORT\实验六\U201514559_6.exe  (a) | C:\Users\lnsqr\Desktop\CPP_PROJECT\REPORT\实验六\U201514559_6.exe  (b) |
| C:\Users\lnsqr\Desktop\CPP_PROJECT\REPORT\实验六\U201514559_6.exe  (c) | C:\Users\lnsqr\Desktop\CPP_PROJECT\REPORT\实验六\U201514559_6.exe  (d) |

图4.1 舞伴问题测试结果

4.2 栈模拟队列测试

本项测试用了老师提供的 C++实验自动验收系统V3.0，自动测试结果如图4.2所示，程序正确通过测试。

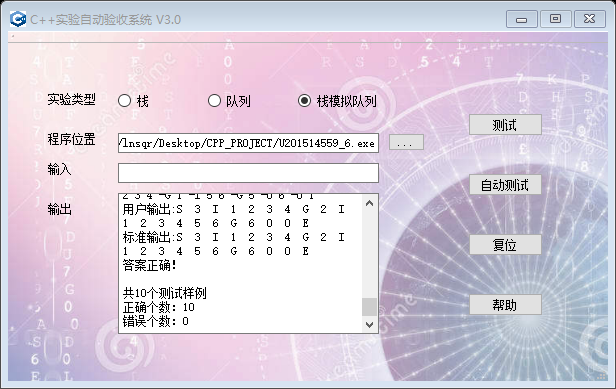


图4.2 栈模拟队列测试结果

1. **特点与不足**
2. **技术特点**

本程序的特点主要是采取了命令行的方式工作，在获取到诸如-S -I之类的命令时，对当前操作用一个变量opt\_flag记录，然后继续读取命令行参数，根据opt\_flag执行对应程序。收到下一个-S类的命令或者所有参数读取完毕后，代表本次操作结束，调用一次PrintToFile函数，把操作结果同时输出到屏幕和文件里。

1. **不足和改进的建议**

不足之处在于没有对命令行的参数做出比较详尽的错误检测，输入参数不规范时容易产生错误。改进的时候可以多加一些限制条件，同时对程序进行一些优化。

1. **过程和体会**
2. **遇到的主要问题和解决方法**

主要遇到的问题是对const类型的修改，由于栈类STACK的成员变量max和elems都是const类型，在初始化的时候可以用初始化列表来赋值，但在assign函数中对栈赋值时，const类型并不是一个有效的左值。因此我的解决办法是用强制类型转化，取const类型变量的地址，强制转化为对应的类型，再改变这个地址所指向的内存空间即可。

1. **课程设计的体会**

通过本次课程设计，我对栈和队列两种数据结构的了解更加深刻了，认识到很多实际用这两种数据结构来解决是十分方便快捷的，就比如舞伴问题，用队列模型很容易求解。同时也熟悉了C++语言的各种用法，感受到面向对象程序设计的灵活性。逐渐从面向过程转变编程思想，是我本次课程设计最大的收获。

1. **源码和说明**
2. **文件清单及其功能说明**

执行文件为U201514559\_6.exe，源码文件为U201514559\_6.txt

1. **用户使用说明书**

本程序有两种运行方式，一种直接双击可执行文件，用无命令行参数形式运行，另外一种通过cmd输入命令行参数调用本程序。

1. **源代码**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*STACK类定义\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class STACK {

int \*const elems; //申请内存用于存放栈的元素

const int max; //栈能存放的最大元素个数

int pos; //栈实际已有元素个数，栈空时pos=0;

public:

STACK(int m); //初始化栈：最多m个元素

STACK(const STACK&s); //用栈s拷贝初始化栈

virtual int size() const; //返回栈的最大元素个数max

virtual operator int() const; //返回栈的实际元素个数pos

virtual int operator[ ] (int x) const; //取下标x处的栈元素

virtual STACK& operator<<(int e); //将e入栈,并返回栈

virtual STACK& operator>>(int &e); //出栈到e,并返回栈

virtual STACK& operator=(const STACK&s); //赋s给栈,并返回被赋值的栈

virtual void print() const; //打印栈

virtual ~STACK(); //销毁栈

};

STACK stack0 = 0;//用于标识异常情况

STACK::STACK(int m) :max(m), elems(new int[m])//const类型用初始化列表赋值

{

pos = 0;

}

STACK::STACK(const STACK&s): max(s.max), elems(new int[s.max])//拷贝构造函数(新建栈)

{

for (int i = 0;i<=s.pos;i++)//复制栈元素

{

elems[i] = s.elems[i];

}

pos = s.pos;

//cout << "copy finished" << endl;

}

int STACK::size() const

{

return max;

}

STACK::operator int() const//返回栈的实际元素个数pos

{

return pos;

}

int STACK::operator[ ] (int x) const //取下标x处的栈元素

{

if (x <= pos)

return elems[x];

else

return -1984;

}

STACK& STACK::operator<<(int e) //将e入栈,并返回栈

{

if (pos < max)//判断当前栈空间大小是否足够

{

elems[pos] = e;

pos++;

return (\*this);

}

else

{

//cout << "栈空间已满！" << endl;

return stack0;//返回一个大小为0的栈，表示栈满

}

}

STACK& STACK::operator>>(int &e) //出栈到e,并返回栈

{

if (pos == 0)

{

//cout << "空栈！" << endl;

return stack0;//返回一个大小为0的栈，表示栈空

}

else

{

e = elems[pos - 1];//把栈顶元素出栈到e

elems[pos - 1] = 0;//栈顶元素赋0

pos--;

return (\*this);

}

}

STACK& STACK::operator=(const STACK&s) //赋s给栈,并返回被赋值的栈

{

\*((int\*)(&max))= s.max;

delete elems;

\*((int\*\*)(&elems))=NULL;

\*((int\*\*)(&elems))= new int[s.max];

for (int i = 0;i<=s.pos;i++)//复制栈元素

{

elems[i] = s.elems[i];

}

pos = s.pos;

return (\*this);

}

void STACK::print() const

{

for (int i = 0; i < int(\*this); i++)

{

cout << " " << elems[i];

}

}

STACK::~STACK()

{

if(elems!=NULL)

delete[]elems;

\*((int\*)(&max)) = 0;

pos = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*STACK类结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int x = 0;//元素下标

ofstream MyFile("U201514559\_6.txt");//文件

int opt\_flag = 0;//操作标志

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*QUEUE类定义\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class QUEUE:public STACK{

STACK s2;//s2为出队，栈顶表示队首。

public:

QUEUE(int m); //每个栈最多m个元素，要求实现的队列最多能入2m个元素

QUEUE(const QUEUE&s); //用队列s拷贝初始化队列

virtual operator int ( ) const; //返回队列的实际元素个数

virtual int full ( ) const; //返回队列是否已满，满返回1，否则返回0

virtual int operator[ ](int x)const; //取下标为x的元素，第1个元素下标为0

virtual QUEUE& operator<<(int e); //将e入队列,并返回队列

virtual QUEUE& operator>>(int &e); //出队列到e,并返回队列

virtual QUEUE& operator=(const QUEUE&s); //赋s给队列,并返回被赋值的队列

virtual void print( ) const; //打印队列

virtual ~QUEUE( ); //销毁队列

};

QUEUE queue0 = 2;//用于标识异常情况

QUEUE::QUEUE(int m):STACK(m), s2(m)

{

}

QUEUE::QUEUE(const QUEUE&s):STACK(s),s2(s.s2)//可以用父类的地方也可以用子类

{

}

QUEUE::operator int() const//返回队列的实际元素个数

{

return (STACK::operator int ()+int(s2));

}

int QUEUE::full ( ) const//返回队列是否已满，满返回1，否则返回0

{

if(int(\*this)>=STACK::size()+s2.size())//判断当前队列空间与最大空间大小

{

return 1;

}

return 0;

}

int QUEUE::operator[ ](int x)const //取下标为x的元素,第1个元素下标为0

{

int i = 0;

int j = 0;

if(x>=(STACK::operator int ()+int(s2)))//超出队列元素范围

{

return -1984;

}

else

{

if(x<=(int(s2)-1))//x可从s2中取得

{

i = (int(s2)-1);//第一个元素下标

for(j=0;j<x;j++)

{

i--;

}

return s2[i];

}

else//x从s1中取

{

x=x-int(s2);

return STACK::operator [](x);

}

}

}

QUEUE& QUEUE::operator<<(int e)//将e入队列,并返回队列

{

int p = 0;

if ((STACK::operator<<(e)).size() != 0)//如果栈s1不满，直接入栈s1中

return \*this;

else

if(int(s2)== 0)//如果s1满了，s2为空，则把s1中的元素弹栈到s2中，再入栈s1

{

while((STACK::operator>>(p)).size()!=0)

{

s2<<p;

}

STACK::operator<<(e);

return \*this;

}

else

if(int(s2) < s2.size())//如果s1满了，s2中存在一些元素且不满(中途进行过出栈操作)

{

return queue0;//这样设置为无法入队列

}

else //s1和s2都满了

return queue0;//返回一个大小为0的队列，表示队列满

}

QUEUE& QUEUE::operator>>(int &e)//出队列到e,并返回队列

{

int p = 0;

if ((s2>>e).size() != 0)//如果s2不为空，直接弹出

return \*this;

else

if(STACK::operator int ()!=0)//s2为空，s1不为空，则把s1中的元素弹到s2中，再从s2出栈

{

while((STACK::operator>>(p)).size()!=0)

{

s2<<p;

}

s2>>e;

return \*this;

}

else//s1和s2都空

return queue0;//返回一个大小为0的队列，表示队列空

}

QUEUE& QUEUE::operator=(const QUEUE&s) //赋s给队列,并返回被赋值的队列

{

STACK::operator=(s);

s2 = s.s2;

return (\*this);

}

void QUEUE::print() const//打印队列

{

int i =0;

if (opt\_flag <5)//完成的操作为I 0 C A

{

if(int(s2)!=0)//s2不为空，输出

{

for (i = (int(s2)-1) ; i >= 0; i--)//栈底的下标为0,从栈顶输出到栈底，s2的栈顶为队列头

{

cout << " " << s2[i];

MyFile << " " << s2[i];

}

}

if(STACK::operator int ()!=0)//s2中所有元素输出结束后，若s1不为空，输出

{

for (i = 0 ; i < STACK::operator int (); i++)//栈底的下标为0,从栈顶输出到栈底，s1的栈顶为队列尾

{

cout << " " << STACK::operator [](i);

MyFile << " " << STACK::operator [](i);

}

}

}

if (opt\_flag == 5)//当前队列的剩余个数 N

{

cout << " " << int(\*this);

MyFile << " " << int(\*this);

}

if (opt\_flag == 6)//数组下标为x的元素 G

{

cout << " " << (\*this)[x];

MyFile << " " << (\*this)[x];

}

}

QUEUE::~QUEUE()

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*QUEUE类结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char\* argv[])

{

if (argc==1)//处理舞伴问题

{

int M,F,m,f;

cout << "请输入男队人数M：";

cin >>M;

cout << "请输入女队人数F：";

cin >>F;

cout << "请输入男士位置m：";

cin >>m;

cout << "请输入女士位置f：";

cin >>f;

if(M==F || m>M || f>F)

{

cout << "输入不合规范，已退出程序" <<endl;

return -1;

}

else

{

QUEUE Mqueue = M;//申请男队队列

QUEUE Fqueue = F;//申请女队队列

for(int m1=1;m1<=M;m1++)//给男队赋初值

{

if(m1 == m)

Mqueue<<2;//被选中的男生

else

Mqueue<<1;//表示普通男生

}

for(int f1=1;f1<=F;f1++)//给女队赋初值

{

if(f1 == f)

Fqueue<<2;//被选中的女生

else

Fqueue<<1;//表示普通女生

}

int male,female;//在跳舞的男女

int dance = 1;//跳舞计数

while(1)

{

Mqueue>>male;

Fqueue>>female;

if(male == 2 && female == 2)

{

cout << "他在第 "<< dance <<" 支舞曲可以和那位女士跳舞"<<endl;

system("pause");

return 0;

}

Mqueue<<male;

Fqueue<<female;

dance++;

}

}

return 0;

}

else//栈自动测试

{

int size\_1 = 0;

int size\_2 = 0;

int push\_e = 0;//入队列元素

int pop\_e = 0;//出队列元素

int pop\_num = 0;//出队列元素个数

queue0<<1984;

//判断是否设定栈或队列大小，-S

if ((strcmp("-S", argv[1]) == 0) || (strcmp("-s", argv[1]) == 0))

{

//正确输入-S后

//判断是否输入了栈和队列大小

cout << "S";

MyFile << "S";//在文件中写入S操作

if (argv[2] == NULL)

{

//cout << "未输入栈队或队列大小！" << endl;

return -3;

}

sscanf(argv[2], "%d", &size\_1);//获取命令行参数

QUEUE queue1 = size\_1;//调用构造函数

cout << " " << size\_1;//返回max

MyFile << " " << size\_1;//在文件中写入max

//循环判定操作

for (int i = 3; i < argc; i++)

{

//检测到操作符

if ((strcmp("-I", argv[i]) == 0) || (strcmp("-i", argv[i]) == 0))

{

queue1.print();//操作完成后调用

opt\_flag = 1;

cout << " " << "I";

MyFile << " " << "I";//在文件中写入I

continue;

}

if ((strcmp("-O", argv[i]) == 0) || (strcmp("-o", argv[i]) == 0))

{

queue1.print();//操作完成后调用

opt\_flag = 2;

cout << " " << "O";

MyFile << " " << "O";//在文件中写入O

continue;

}

if ((strcmp("-C", argv[i]) == 0) || (strcmp("-c", argv[i]) == 0))

{

queue1.print();//操作完成后调用

opt\_flag = 3;

cout << " " << "C";

MyFile << " " << "C";//在文件中写入C

//不需要输入参数，直接调用拷贝构造函数

QUEUE queue2 = queue1;

queue1 = queue2;//以后操作新队列

continue;

};

if ((strcmp("-A", argv[i]) == 0) || (strcmp("-a", argv[i]) == 0))

{

queue1.print();//操作完成后调用

opt\_flag = 4;

cout << " " << "A";

MyFile << " " << "A";//在文件中写入A

continue;

}

if ((strcmp("-N", argv[i]) == 0) || (strcmp("-n", argv[i]) == 0))

{

queue1.print();//操作完成后调用

opt\_flag = 5;

cout << " " << "N";

MyFile << " " << "N";//在文件中写入N

continue;

}

if ((strcmp("-G", argv[i]) == 0) || (strcmp("-g", argv[i]) == 0))

{

queue1.print();//操作完成后调用

opt\_flag = 6;

cout << " " << "G";

MyFile << " " << "G";//在文件中写入G

continue;

}

switch (opt\_flag)

{

case 0:

{

break;

}

case 1://入队列

{

sscanf(argv[i], "%d", &push\_e);

if ((queue1<<push\_e)[0] == 1984)//判断入栈结果返回值

{

cout << " " << "E";

MyFile << " " << "E";

return -1;

}

//cout << push\_e << " 成功入栈!" << endl;

break;

}

case 2://出栈

{

sscanf(argv[i], "%d", &pop\_num);

for (int j = 0; j < pop\_num; j++)

{

if ((queue1>>pop\_e)[0] == 1984)//调用出栈函数

{

cout << " " << "E";

MyFile << " " << "E";

return -2;

}

//cout << pop\_e << "成功出栈!" << endl;

}

break;

}

case 3://深拷贝构造

{

//无参数输入

break;

}

case 4://深拷贝赋值

{

sscanf(argv[i], "%d", &size\_2);

QUEUE queue3 = size\_2;//new

queue3 = queue1;//assign

queue1 = queue3;//把新栈赋值给旧栈，以后操作新栈

break;

}

case 5://剩余元素个数

break;

case 6://获得下标为x的元素

sscanf(argv[i], "%d", &x);

if (-1984 == queue1[x])

{

cout << " " << "E";

MyFile << " " << "E";

return -3;

}

break;

default:

break;

}

}

queue1.print();//命令结束后后调用

}

else

{

//cout << "未设定栈队或队列大小！" << endl;

return -2;

}

MyFile.close();//关闭文件

return 0;

}

}