面向对象程序设计
实验3 交互依赖

屈熙宸

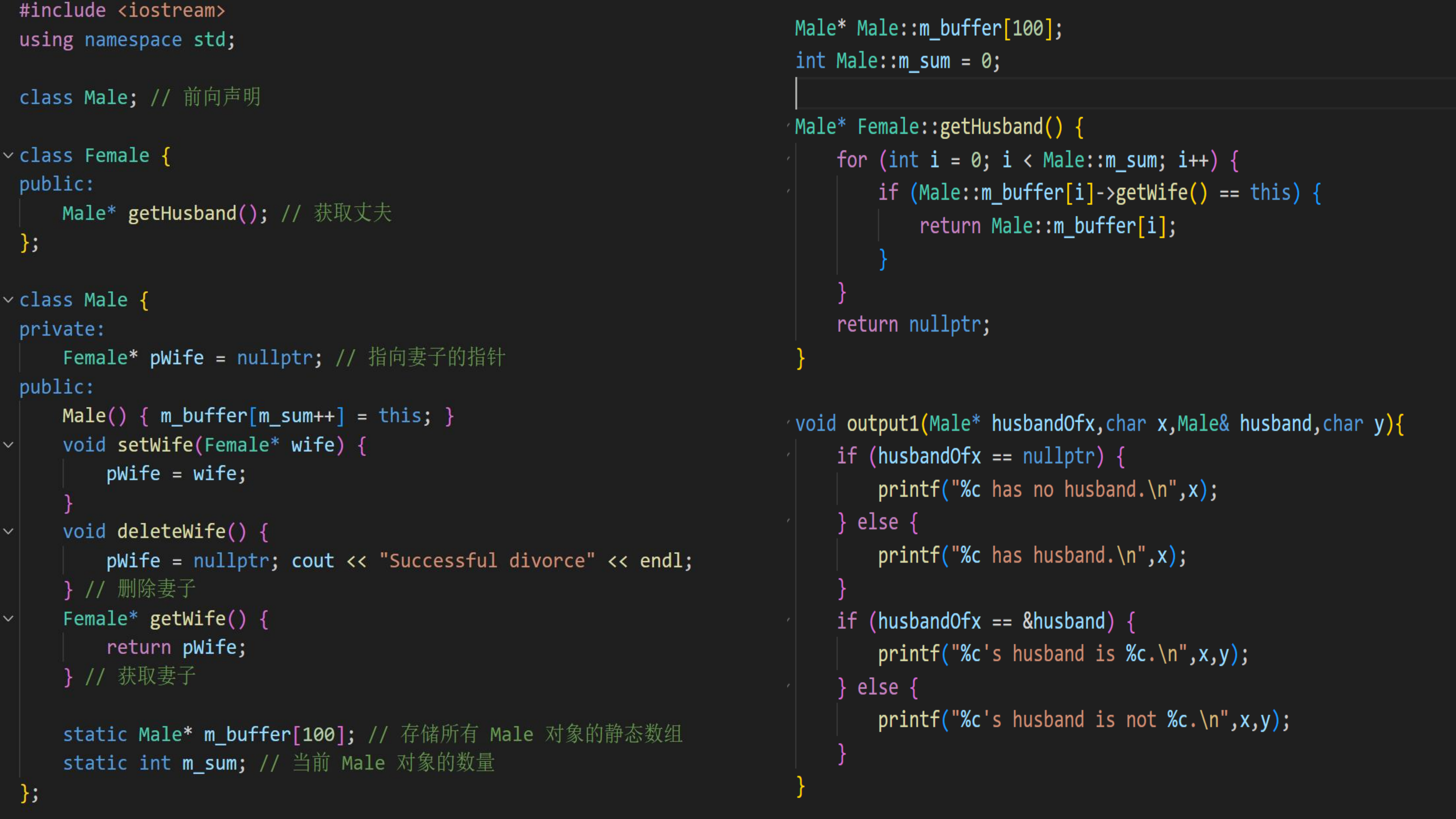
55240425

任务：

调试PPT中双向关联改为单向关联后的程序，给出main函数调用过程

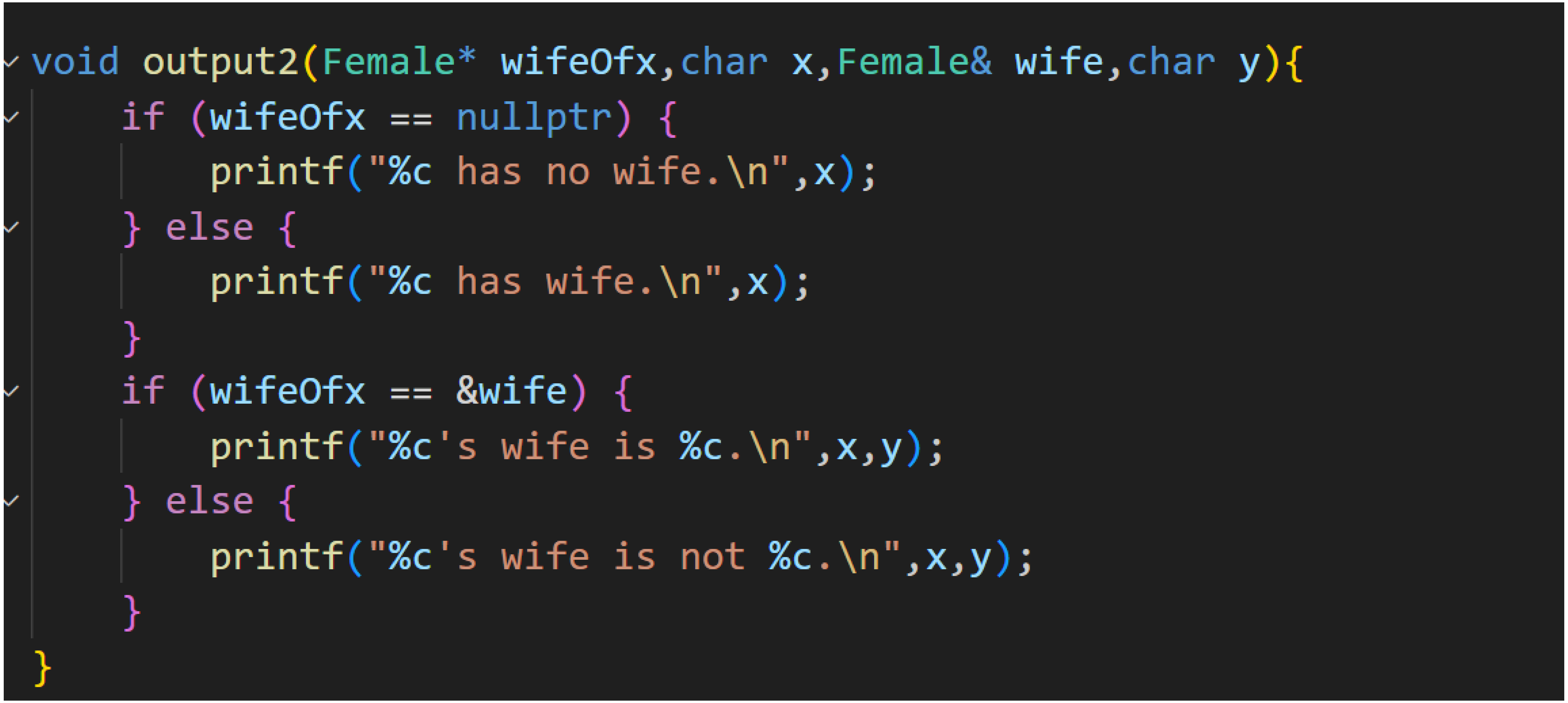
- 分析题目要求:
  - 要求将原本丈夫和妻子的双向关联关系改为单向关联

程序源码+运行结果

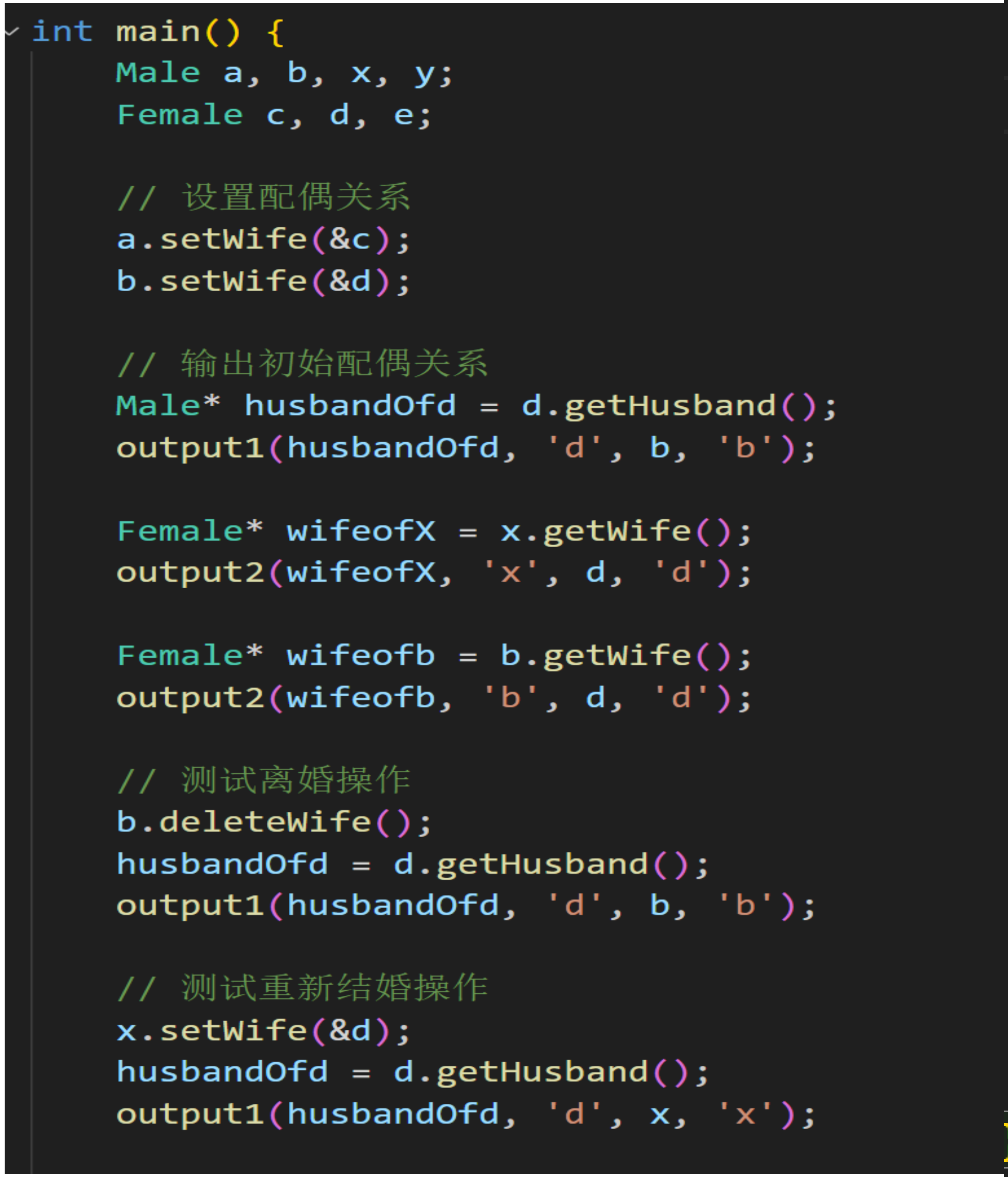


```cpp
#include <iostream>
using namespace std;

class Male; // 前向声明

class Female {
public:
    Male* getHusband(); // 获取丈夫
};

class Male {
private:
    Female* pWife = nullptr; // 指向妻子的指针
public:
    Male() { m_buffer[m_sum++] = this; }
    void setWife(Female* wife) {
        pWife = wife;
    }
    void deleteWife() {
        pWife = nullptr; cout << "Successful divorce" << endl;
    } // 删除妻子
    Female* getWife() {
        return pWife;
    } // 获取妻子

    static Male* m_buffer[100]; // 存储所有 Male 对象的静态数组
    static int m_sum; // 当前 Male 对象的数量
};
```

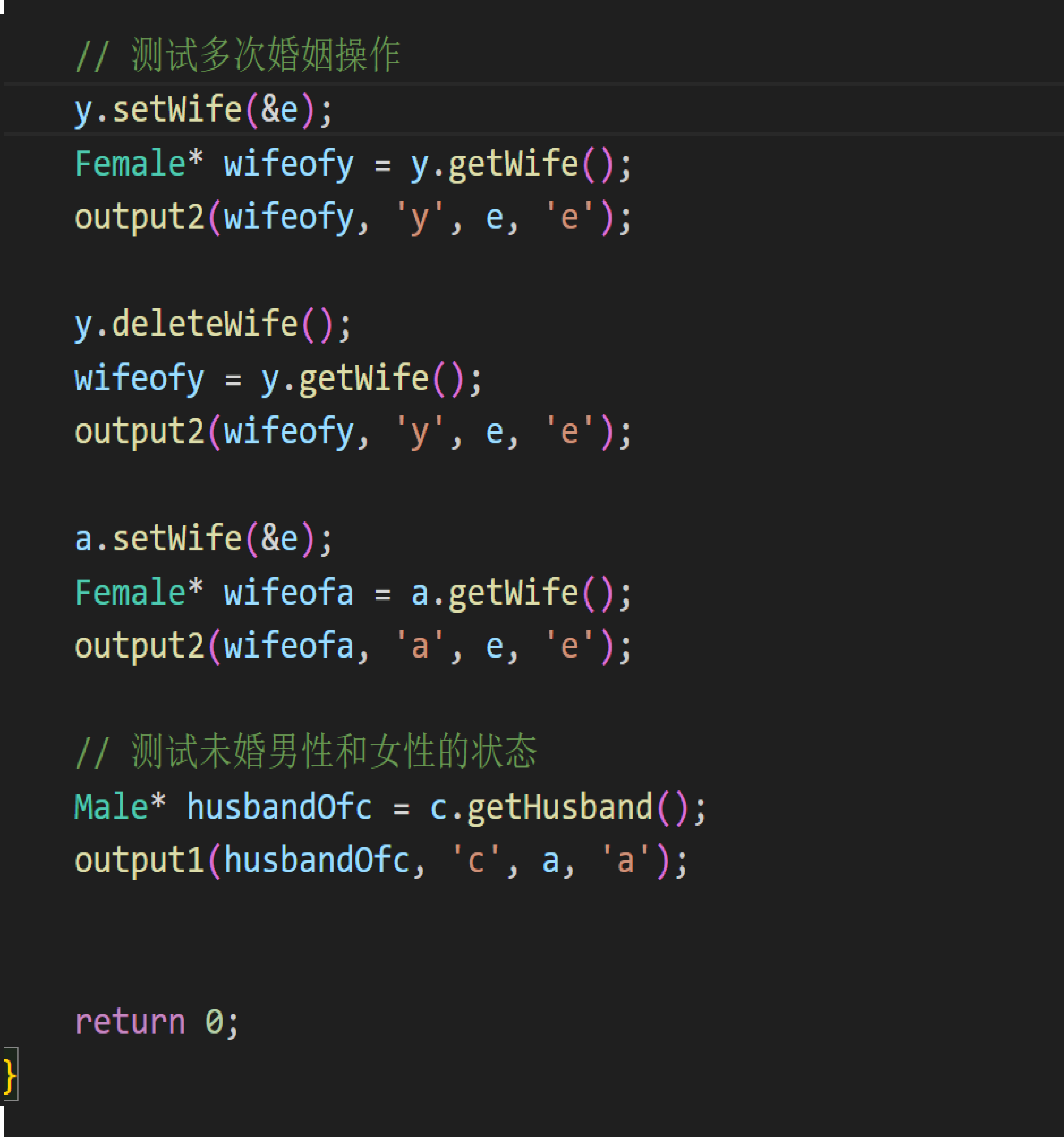
```cpp
Male* Male::m_buffer[100];
int Male::m_sum = 0;

Male* Female::getHusband() {
    for (int i = 0; i < Male::m_sum; i++) {
        if (Male::m_buffer[i]->getWife() == this) {
            return Male::m_buffer[i];
        }
    }
    return nullptr;
}

void output1(Male* husbandOfx,char x,Male& husband,char y){
    if (husbandOfx == nullptr) {
        printf("%c has no husband.\n",x);
    } else {
        printf("%c has husband.\n",x);
    }
    if (husbandOfx == &husband) {
        printf("%c's husband is %c.\n",x,y);
    } else {
        printf("%c's husband is not %c.\n",x,y);
    }
}
```



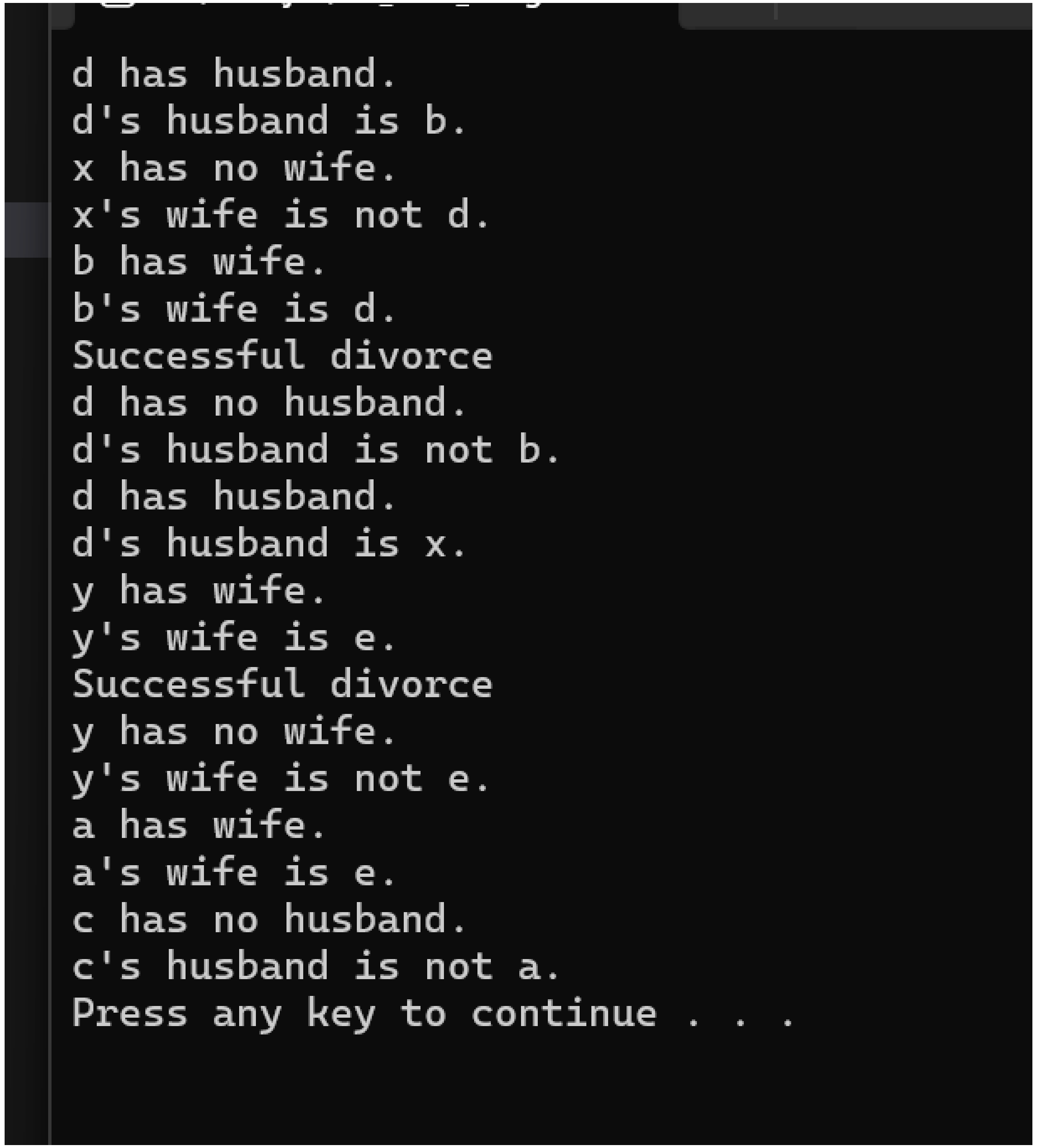
```
void output2(Female* wifeOfx,char x,Female& wife,char y){
    if (wifeOfx == nullptr) {
        printf("%c has no wife.\n",x);
    } else {
        printf("%c has wife.\n",x);
    }
    if (wifeOfx == &wife) {
        printf("%c's wife is %c.\n",x,y);
    } else {
        printf("%c's wife is not %c.\n",x,y);
    }
}
```



```cpp
int main() {
    Male a, b, x, y;
    Female c, d, e;

    // 设置配偶关系
    a.setWife(&c);
    b.setWife(&d);

    // 输出初始配偶关系
    Male* husbandOfd = d.getHusband();
    output1(husbandOfd, 'd', b, 'b');

    Female* wifeofX = x.getWife();
    output2(wifeofX, 'x', d, 'd');

    Female* wifeofb = b.getWife();
    output2(wifeofb, 'b', d, 'd');

    // 测试离婚操作
    b.deleteWife();
    husbandOfd = d.getHusband();
    output1(husbandOfd, 'd', b, 'b');

    // 测试重新结婚操作
    x.setWife(&d);
    husbandOfd = d.getHusband();
    output1(husbandOfd, 'd', x, 'x');
```

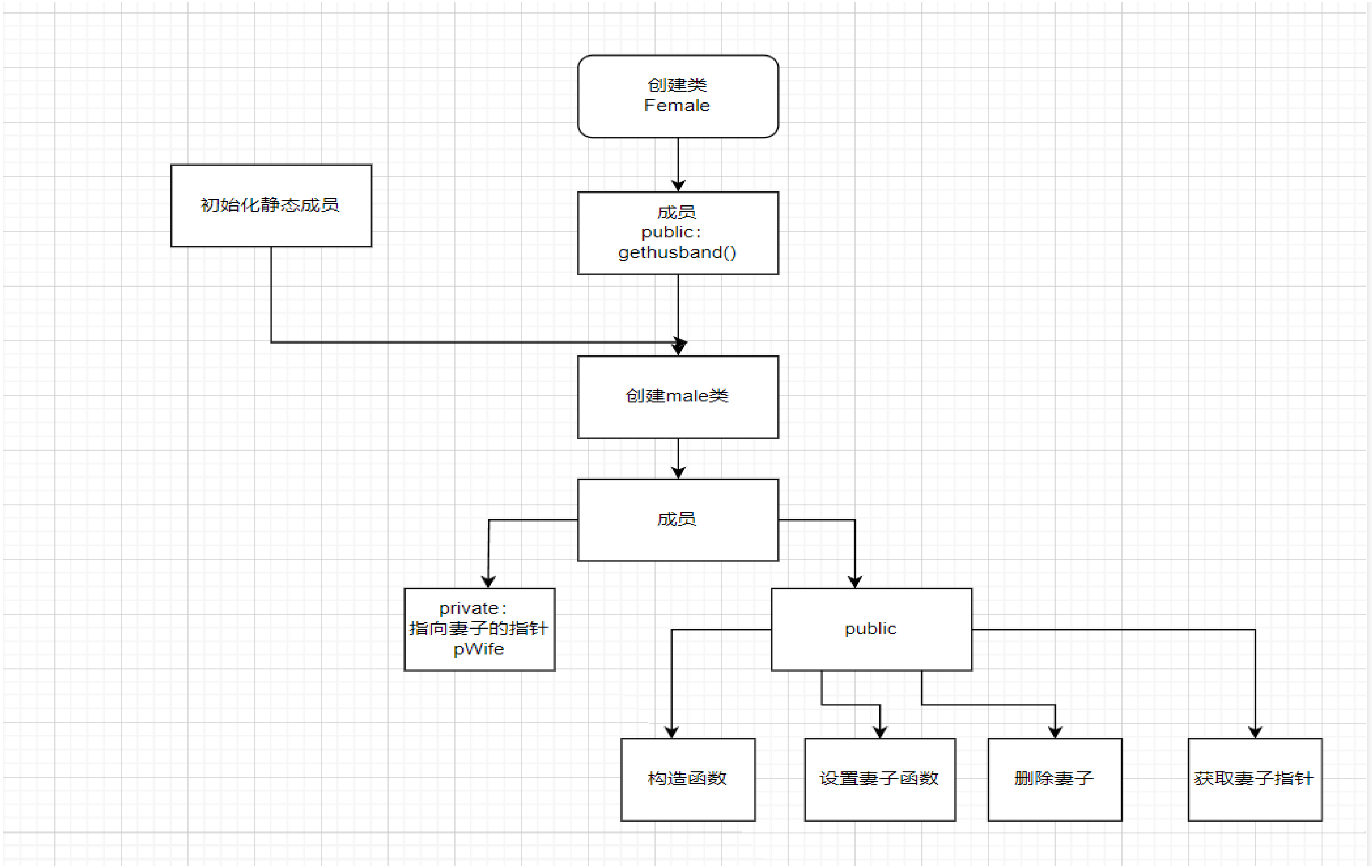


```cpp
    // 测试多次婚姻操作
    y.setWife(&e);
    Female* wifeofy = y.getWife();
    output2(wifeofy, 'y', e, 'e');

    y.deleteWife();
    wifeofy = y.getWife();
    output2(wifeofy, 'y', e, 'e');

    a.setWife(&e);
    Female* wifeofa = a.getWife();
    output2(wifeofa, 'a', e, 'e');

    // 测试未婚男性和女性的状态
    Male* husbandOfc = c.getHusband();
    output1(husbandOfc, 'c', a, 'a');


    return 0;
}
```



```
d has husband.
d's husband is b.
x has no wife.
x's wife is not d.
b has wife.
b's wife is d.
Successful divorce
d has no husband.
d's husband is not b.
d has husband.
d's husband is x.
y has wife.
y's wife is e.
Successful divorce
y has no wife.
y's wife is not e.
a has wife.
a's wife is e.
c has no husband.
c's husband is not a.
Press any key to continue . . .
```

2.流程图


创建类
Female
初始化静态成员
成员
public：
gethusband()
创建male类
成员
private：
指向妻子的指针
pWife
public
构造函数
设置妻子函数
删除妻子
获取妻子指针

3.分析难点

分析难点:

在于如何寻找妻子的丈夫，如若遍历寻找丈夫地址则难以进行

解决方法:

建立关于丈夫的线性表，作为静态成员，给妻子遍历一个方位

4.分析

- 优点：
- 代码实现了基本的配偶设置、解除和查询功能，涵盖了婚姻关系的主要操作。
- 缺点：
- 没有检查重复婚姻或非法操作（如一个 Male 对象同时拥有多个妻子，或一个 Female 对象同时有多个丈夫）。

5.收获

- 更好的理解类与类的关联
- 切身体会了构造函数的重要用途