Assignment 3

191220022 丁一凡

一、 概念题

1.1 什么时候需要定义析构函数?

如果对象创建后，又申请了额外的资源（如内存空间），则需要定义析构函数来归还。

1.2 什么时候会调用拷贝构造函数？使用默认的拷贝构造函数有什么需要特别注意的情况?

在三种情况下，会调用拷贝构造函数：

① 创建对象时显式指出；

② 把对象作为值参数传递给函数；

③ 把对象作为函数的返回值。

默认的拷贝构造函数对于非对象成员，会采取普通的拷贝操作，如果类中有指针、引用等数据成员，默认的拷贝构造函数会使得拷贝的对象与原对象中的指针指向同一块内存，从而带来一些难以发现的错误。

1.3 请说明C++中const和static关键词的作用

C++中，被 `const` 关键词修饰过的成员函数会变为常成员函数，在常成员函数中不能修改成员函数的值。

C++中，被 `static` 关键词修饰过的类的数据成员为该类的静态数据成员，对该类的所有对象只有一个拷贝，可以更好地实现数据共享；

被 `static` 修饰过的函数为静态成员函数，只能访问类的静态成员。

1.4 简述C++友元的特性以及其利弊

友元的特性：

① 友元可以在类的外部访问该类的数据成员；

② 友元不是类的成员；

③ 友元关系不具有对称性和传递性。

利：

提高了对类的数据成员的访问效率

弊：

不通过类提供的接口访问，违背了数据封装的思想，不利于数据保护

二、 编程题

2.1

1. 静态成员 `MerchandiseCnt` 记录创建的对象数，所以在析构函数中应该减一 ,对于字符数组，`delete` 时应该添加 `[]`

```
Merchandise::~Merchandise()
{
    delete []name;  //添加[]
    name = nullptr;
    MerchandiseCnt--;   //添加
}
```

2. 函数 `void Merchandise::set_name(const char *_name)` 应该为修改上商品名称的接口，但是加上 `const` 关键词后变为常成员函数，不能修改数据成员 `char* name`

```
void Merchandise::set_name(const char *_name) //删去const
{
    delete name;
    name = new char[strlen(_name) + 1];
    strcpy(name, _name);
}
```

2.2

```
class FloatSet
{
    float *numbers;// 可根据需要添加其他成员变量
    int MaxSize;
    int count;
public:
    FloatSet();
    FloatSet(const FloatSet& s);
    ~FloatSet();
    bool is_empty() const; //判断是否为空集
    int size() const; //获取元素个数
    bool is_element(float e) const; //判断e是否属于集合
    bool is_subset(const FloatSet& s) const; //判断集合是否包含于s
    bool is_equal(const FloatSet& s) const; //判断集合是否相等
    bool insert(float e); //将元素e加入集合，成功返回true，否则返回false(e已属于集合)
    bool remove(float e); //将e从集合中删除，成功返回true，否则返回false(e不属于集合)
```

```
    void display() const; //打印集合所有元素
    FloatSet union2(const FloatSet &s) const; //计算集合和s的并集
    FloatSet intersection2(const FloatSet &s) const; //计算集合和s的交集
    FloatSet difference2(const Floatet& s) const; //计算集合和s的差
};

FloatSet::FloatSet()
{
    count = 0;
    MaxSize = 10;
    numbers = new float[MaxSize];
}
FloatSet::FloatSet(const FloatSet& s)
{
    count = s.count;
    MaxSize = count * 2;
    numbers = new float[MaxSize];
    for(int i=0; i < count; i++)
        numbers[i] = s.numbers[i];
}
FloatSet::~FloatSet()
{
    count = 0;
    MaxSize = 0;
    delete []numbers;
}
bool FloatSet::is_empty() const { return count==0; }
int FloatSet::size() const { return count; }
bool FloatSet::is_element(float e)
{
    for(int i = 0; i < count; i++)
    {
        if(numbers[i] == e)
            return true;
    }
    return false;
}
bool is_subset(const FloatSet& s) const
{
    for(int i = 0; i < count; i++)
    {
        int j = 0;
        for(; j < s.count; j++)
        {
            if(s.numbers[j] == numbers[i])
                break;
        }
        if(j == s.count)
            return false;
    }
    return true;
}
bool FloatSet::is_equal(const FloatSet& s) const
{
    if(s.count != count)
        return false;
    for(int i = 0; i < count; i++)
    {
```

```
        int j = 0;
        for(; j < s.count; j++)
        {
            if(s.numbers[j] == numbers[i])
                break;
        }
        if(j == s.count)
            return false;
    }
    return true;
}
bool FloatSet::insert(float e)
{
    for(int i = 0; i < count; i++)
    {
        if(numbers[i] == e)
            return false;
    }
    if(count == MaxSize)
    {
        MaxSize *= 2;
        float* t = new float[MaxSize];
        for(int i = 0; i < size; i++)
            t[i] = numbers[i];
        delete []numbers;
        numbers = t;
    }
    numbers[count] = e;
    count++;
    return true;
}
bool FloatSet::remove(float e)
{
    for(int i = 0; i < count; i++)
    {
        if(numbers[i] == e)
        {
            for(int j = i + 1; j < count; j++)
                numbers[j - 1] = numbers[j];
            count--;
            return true;
        }
    }
    return false;
}
void FloatSet::display()
{
    for(int i = 0; i < count; i++)
        cout << numbers[i] << " ";
    cout << endl;
}
FloatSet FloatSet::union2(const FloatSet& s) const
{
    FloatSet result(s);
    for(int i = 0; i < count; i++)
    {
         result.insert(numbers[i]);
    }
```

```
    return result;
}
FloatSet FloatSet::intersection2(const FloatSet& s) const
{
    FloatSet result;
    for(int i = 0; i < count; i++)
    {
        if(s.is_element(numbers[i]))
        {
            result.insert(numbers[i]);
        }
    }
    return result;
}
FloatSet FloatSet::difference2(const FloatSet& s) const
{
    FloatSet result(*this);
    for(int i = 0; i < s.count; i++)
        result.remove(s[i]);
    return result;
}
```