Assignment 3

概念题

1.1 什么时候需要定义析构函数？

如果对象创建后，自已又额外了资源，可以自定义析构函数来归还他们。

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？

（1）

- 创建对象时显示指出。
- 把对象作为值参数传给函数。
- 把对象作为函数返回值。

（2）

当对象当中有指针类型的数据时，使用默认拷贝构造函数，只能够将指针地址传给，但指针指向的内存为同一块内存地址。

这时，会有如下问题：

- 如果对一个对象操作之后，更改了这块空间的内容，如果不是设计者故意为之，那么就会影响另一个对象
- 当对象a1,a2消亡时，将会分别调用他们的析构函数，这会导致同一个内存区域被同时归还两次，从而导致程序的运行异常。
- 当两个对象有一个消亡时，另一个还没有归还时，则会出现使用已归还的空间的问题。

1.3 请说明C++中 const 和 static 关键词的作用.

c++的函数中，如果使用了const，那么该方法不能够改变该对象中的数据。此外，对于常量对象，之能够调用类中const的成员函数。

对于变量而言，static能够使得该数据在所有对象之间共享，静态数据成员对于该类的所有成员只有一个拷贝。

对于静态的成员函数，只能够访问静态的数据成员。

1.4 简述C++友元的特性以及其利弊.

友元：为了提高在类的外部对类的数据成员的访问效率，在C++中，可以指定某些与一个类密切相关的、又不适合作为该类成员的程序实体直接访问该类的private和protected成员。这些程序实体称为该类的友元。

友元可以是全局函数、其它的类或其它类的某些成员函数。即可以是友元函数，友元类，友元类的函数。

特点：

- 友元关系具有不对称性。
- 友元不具有传递性。
- 友元是数据保护和数据访问效率之间的一种折衷方案。

利弊：

利用 friend 修饰符，可以让一些普通函数 或 另一个类的成员函数 直接对某个类的保护成员和私有成员进行操作，提高了程序的运行效率；同时避免把类的成员都声明为public，最大限度地保护数据成员的安全。

但是，即使是最大限度地保护数据成员，友元也破坏了类的封装性。

Code

改错题

- set_name函数使用了const,但实际上不能够使用const，需要加上const.
- 没有给类的静态成员初始化的定义，需要加上一句

```
int Merchandise::MerchandiseCnt=0;
```

否则编译的时候会出现，undefined reference to Merchandise::Merchandise.

- 没有显示的定义拷贝构造函数，因为数据成员变量有指针，默认的拷贝构造函数只能够拷贝他的地址。但是，实际上所指向的内存空间是一样的，会导致如问题1.2中提到的问题。

```
#include <iostream>
#include <cstring>
using namespace std;
class Merchandise
{
    static int MerchandiseCnt;
    char *name;

public:
    Merchandise(const char *_name);
    ~Merchandise();
    char *get_name() const;
    void set_name(const char *_name) const;
};
Merchandise::Merchandise(const char *_name)
{
    name = new char[strlen(_name) + 1];
    strcpy(name, _name);
    MerchandiseCnt++;
}
Merchandise::~Merchandise()
{
    delete name;
    name = nullptr;
}
char *Merchandise::get_name() const
{
    return name;
}
void Merchandise::set_name(const char *_name) const
{
    delete name;
    name = new char[strlen(_name) + 1];
    strcpy(name, _name);
```

```
}
int main()
{
    {
        Merchandise m1("phone");
        Merchandise m2(m1);
    }
    return 0;
}
```

集合实现

```
//set.h
#include <iostream>
#include <assert.h>
#include <cstring>
#define SIZE_ONCE 10
class FloatSet
{
    private:
    float *numbers;
    int cnt;
    int capbility;

    public:
    FloatSet();
    FloatSet(const FloatSet& s);
    ~FloatSet();
    bool is_empty() const;
    int size() const;
    bool is_element(float e)const;
    bool is_subset(const FloatSet& s) const;
    bool is_equal(const FloatSet& s) const;
    bool insert(float e);
    bool remove(float e);
    void display() const;
    FloatSet union2(const FloatSet &s) const;
    FloatSet intersection2(const FloatSet &s) const;
    FloatSet difference2(const FloatSet& s) const;
    FloatSet operator+(const FloatSet &s) const ;
    FloatSet operator-(const FloatSet&s) const;
    FloatSet operator*(const FloatSet &s) const;


};
```

```
//set.cpp
#include "set.h"

FloatSet::FloatSet()
{
    cnt=0;
    numbers=new float[SIZE_ONCE];
    capbility=SIZE_ONCE;
}
```

```
FloatSet::FloatSet(const FloatSet &s)
{
    cnt=s.cnt;
    capbility=s.capbility;
    assert(capbility%SIZE_ONCE==0);
    numbers=new float[capbility];
    memcpy(numbers,s.numbers,sizeof(float)*capbility);
}
FloatSet::~FloatSet()
{
    delete []numbers;
}
bool FloatSet::is_empty() const
{
    return cnt==0;
}
int  FloatSet::size() const
{
    return cnt;
}
bool FloatSet::is_element(float e) const
{
    for (int i=0;i<cnt;i++)
    {
        if (numbers[i]==e)
        {
            return true;
        }
    }
    return false;
}
bool FloatSet::is_subset(const FloatSet &s) const
{
    for (int i=0;i<s.cnt;i++)
    {
        if (!is_element(s.numbers[i]))
        {
            return false;
        }
    }
    return true;
}
bool FloatSet::is_equal(const FloatSet &s) const
{
    if (s.cnt!=cnt) return false;
    //大小相同并且是子集。
    return is_subset(s);
}
bool FloatSet::insert(float e)
{
    if (is_element(e))
    {
        return false;
    }
    if (cnt==capbility)
    {
        capbility+=SIZE_ONCE;
        float *tmp=new float[capbility];
```

```
        memcpy(tmp,numbers,(capbility-SIZE_ONCE)*sizeof(float));
        delete []numbers;
        numbers=tmp;
    }
    numbers[cnt]=e;
    cnt++;
    return true;
}
bool FloatSet::remove(float e)
{
    for (int i=0;i<cnt;i++)
    {
        if (numbers[i]==e)
        {
            for (int j=i;j<cnt-1;j++)
            {
                numbers[j]=numbers[j+1];
            }
            cnt--;
            if (cnt==capbility-SIZE_ONCE)
            {
                capbility-=SIZE_ONCE;
                float *tmp=new float[capbility];
                memcpy(tmp,numbers,capbility*sizeof(float));
                delete []numbers;
                numbers=tmp;
            }
            return true;
        }
    }

    return false;
}
void FloatSet::display() const
{
    for (int i=0;i<cnt;i++)
    {
        printf("%f , ",numbers[i]);
    }
    printf("\n");
}

FloatSet FloatSet::union2(const FloatSet &s) const
{
    FloatSet ans(*this);
    for (int i=0;i<s.cnt;i++)
    {
        if (!is_element(s.numbers[i]))
        {
            ans.insert(s.numbers[i]);
        }
    }
    return ans;


}
FloatSet FloatSet::intersection2(const FloatSet &s) const
{
```

```
    FloatSet ans(*this);
    for (int i=0;i<cnt;i++)
    {
        if (!s.is_element(numbers[i]))
        {
            ans.remove(numbers[i]);
        }
    }
    return ans;
}
FloatSet FloatSet::difference2(const FloatSet &s) const
{
    FloatSet ans;
    for (int i=0;i<cnt;i++)
    {
        if (!s.is_element(numbers[i]))
        {
            ans.insert(numbers[i]);
        }
    }
    return ans;
}
FloatSet FloatSet::operator+(const FloatSet &s) const
{
    return union2(s);
}
FloatSet FloatSet::operator*(const FloatSet &s) const
{
    return intersection2(s);
}
FloatSet FloatSet::operator-(const FloatSet &s) const
{
    return difference2(s);
}
```

```
//main.cpp
#include "set.h"

int main()
{
    FloatSet s1;
    FloatSet s2;

    for (int i=0;i<15;i++)
    {
        s1.insert(i);
    }
    for (int i=10;i<20;i++)
    {
        s2.insert(i);
    }
    s1.display();
    s2.display();
    FloatSet s3(s1+s2);
    s3.display();
```

```
    FloatSet s4(s1-s2);
    s4.display();
    FloatSet s5(s1*s2);
    s5.display();

//打印结果：
/*********************************************************************************
*
// 0.000000 , 1.000000 , 2.000000 , 3.000000 , 4.000000 , 5.000000 , 6.000000 ,
7.000000 , 8.000000 , 9.000000 , 10.000000 , 11.000000 , 12.000000 , 13.000000 ,
14.000000 ,
// 10.000000 , 11.000000 , 12.000000 , 13.000000 , 14.000000 , 15.000000 ,
16.000000 , 17.000000 , 18.000000 , 19.000000 ,
// 0.000000 , 1.000000 , 2.000000 , 3.000000 , 4.000000 , 5.000000 , 6.000000 ,
7.000000 , 8.000000 , 9.000000 , 10.000000 , 11.000000 , 12.000000 , 13.000000 ,
14.000000 , 15.000000 , 16.000000 , 17.000000 , 18.000000 , 19.000000 ,
// 0.000000 , 1.000000 , 2.000000 , 3.000000 , 4.000000 , 5.000000 , 6.000000 ,
7.000000 , 8.000000 , 9.000000 ,
// 10.000000 , 11.000000 , 12.000000 , 13.000000 , 14.000000 ,
*********************************************************************************
/
}
```