课程编号：100072107　　北京理工大学 2021－2022 学年第二学期（<u>页面中所有文字不要删除</u>）

2022 年《面向对象技术与方法》试卷

班级：<u>　07112002　</u> **学号：**<u>　1120200822　</u> **姓名：**<u>　郑子帆　　　</u>　　　　**卷面成绩：**______________

注意：一共有三道编程题，同学们根据本课程《考试须知》中的要求，在规定的截止时间之前把此答卷发到授课老师的电子邮箱。对每一道编程题目，都要写出自己的源代码（注释可用中文）、以及运行结果的截屏。（Please send the answer sheets to your teacher in 2 hours）

考试诚信承诺书

我承诺已成功下载本次考试试卷，并自己完成答题，诚信考试。I promise that I have successfully downloaded the examination paper and to answer the questions myself.

此处贴上你的签名(signature)：郑子帆

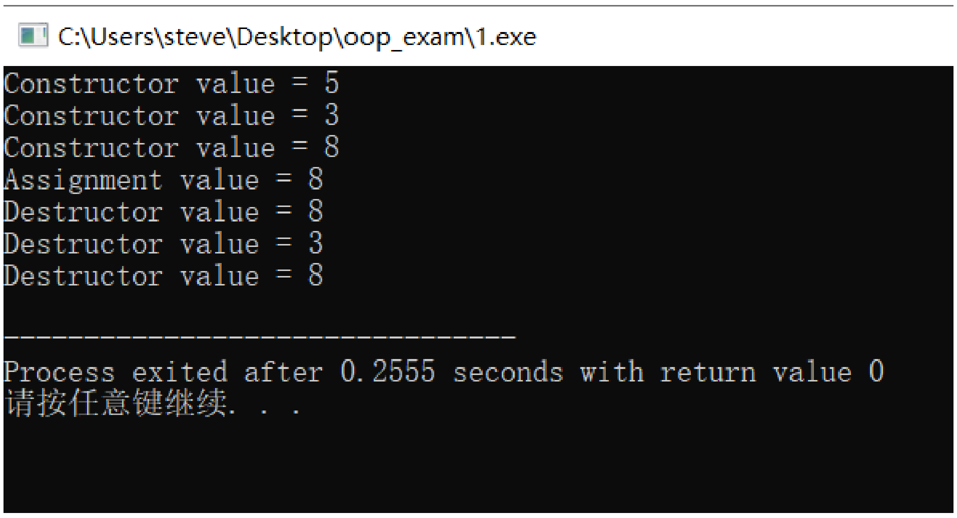
1 (35 points) Complete the class Calculator.

#include <iostream>	**The outputs:**
using namespace std;	
class Calculator	Constructor value = 5
{	Constructor value = 3
private: int value;	Constructor value = 8
public: // your functions:	Assignment value = 8
……	Destructor value =8
};	Destructor value = 3
int main()	Destructor value = 8
{	
Calculator m(5), n;	
m = m + n;	
return 0;	
}	

//Your codes with necessary explanations:

```
#include <iostream>

using namespace std;

class Calculator{
    private:
        int value;
    public:
        Calculator(int val = 3){
            // 带有默认参数值的有参构造函数，默认参数值为 3
            value = val;
            // 将参数值赋值给 Calculator 类对象的 value 数据成员
            cout << "Constructor value = " << value << endl;
            // 输出 Constructor value =
        }
        ~Calculator(){
            // 析构函数
            cout << "Destructor value = " << value << endl;
            // 输出 Destructor value =
        }
        Calculator operator + (const Calculator &a){
            // 重载 + 运算，参数为 const 修饰的 Calculator 类对象的引用
            return Calculator(value + a.value);
            // 返回二者 value 相加所构造的 Calculator 类对象
        }
        Calculator& operator = (const Calculator &a){
            /*
                重载 = 运算（赋值运算），参数为 const 修饰的 Calculator 类对象的引用
                返回类型为Calculator的引用,使得Calculator类对象在赋值时可以连等(非
必需）
            */
            value = a.value;
            // 将 a.value 赋值给当前对象的 value
            cout << "Assignment value = " << value << endl;
            // 输出 Assignment value =
            return *this;
        }
};

int main(){
    Calculator m(5), n;
    m = m + n;
    return 0;
}
```

//Screen capture of running result



C:\Users\steve\Desktop\oop_exam\1.exe

```
Constructor value = 5
Constructor value = 3
Constructor value = 8
Assignment value = 8
Destructor value = 8
Destructor value = 3
Destructor value = 8

--------------------------------
Process exited after 0.2555 seconds with return value 0
请按任意键继续. . .
```

2 (30 points) Complete the class Animal, Wolf and Tiger.

```
#include <iostream>
#include <string>
using namespace std;
class Food
{
    string FoodName;
public:
    Food(string s) : FoodName(s) { };
    string GetFoodName() { return FoodName; }
};
class Animal    // abstract class
{
    string AnimalName;
    Food& food;
public: // your functions:
        ……
};
class Wolf : public Animal
{
public: // your functions:
        ……
};
```

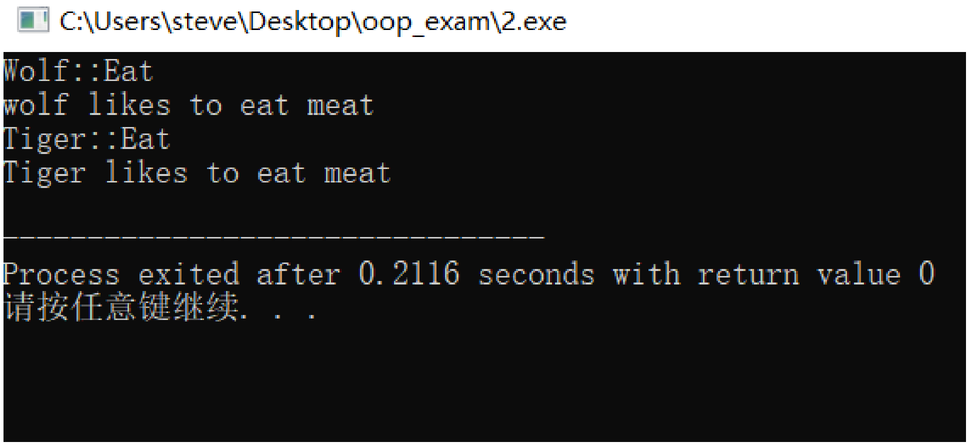
```
class Tiger : public Animal
{
public: // your functions:
        ……
};

int main()
{
    Food meat("meat");
    Animal* panimal = new Wolf("wolf", meat);

    panimal->Eat();                  // display: Wolf::Eat
    cout << *panimal << endl;        // display: Wolf likes to eat meat.
    delete panimal;

    panimal = new Tiger("Tiger", meat);
    panimal->Eat();                  // display: Tiger::Eat
    cout << *panimal << endl;        // display: Tiger likes to eat meat.
    delete panimal;

    return 0;
}
```

//Your codes with necessary explanations:

```
#include <iostream>
#include <string>
using namespace std;
class Food
{
    string FoodName;
    public:
        Food(string s) : FoodName(s) { };
        string GetFoodName() { return FoodName; }
};

class Animal  // abstract class
{
    string AnimalName;
    // string 类型字符串记录动物的名字
    Food &food;
    // 组合，将 Food 类对象的引用作为 Animal 类的数据成员，记录动物的食物
```

```cpp
    public:
        Animal(string name, Food &fd):food(fd){AnimalName = name;}
        /*
            有参构造函数，其中 name 赋值给 AnimalName
            而由于数据成员中有 Food 类对象的引用，故要用参数初始化列表进行初始化
            参数中采用了 Food 类对象的引用
        */
        virtual void Eat() = 0;
        /*
            由于 Animal 为抽象类，故我们设计 Eat()函数为纯虚函数
            这也使得在主函数中调用 Animal 类的指针指向派生类对象的地址时，
           可以通过 vtable 调用派生类的 Eat()
        */
        friend ostream& operator << (ostream &os, const Animal &a){
            // 重载流插入运算符，输出对象中是什么动物爱吃什么食物，为友元函数
            os << a.AnimalName << " likes to eat " << a.food.GetFoodName();
            // 输出时调用 Food 类中的 GetFoodName()接口得到食物名称
            return os;
        }
};

class Wolf : public Animal
// 派生类 Wolf，继承自 Animal 类
{
    public:
        Wolf(string wname, Food &fd):Animal(wname, fd){}
        /*
            Wolf 类的有参构造函数，传入参数 名字 和 食物
            通过初始化列表对基类进行初始化
        */
        virtual void Eat(){
            // 重写(override) Eat()函数
            cout << "Wolf::Eat" << endl;
        }
};

class Tiger : public Animal
// 派生类 Tiger，继承自 Animal 类
{
    public:
        Tiger(string tname, Food &fd):Animal(tname, fd){}
        // 有参构造函数，与 Wolf 类类似，对 Tiger 类对象进行初始化
        virtual void Eat(){
            // 重写 Eat()函数
            cout << "Tiger::Eat" << endl;
        }
};

int main(){
    Food meat("meat");
    Animal* panimal = new Wolf("wolf", meat);
    // 指针指向动态分配的 Wolf 类对象空间


    panimal->Eat();            // display: Wolf::Eat
    cout << *panimal << endl;   // display: Wolf likes to eat meat.
    delete panimal;

    panimal = new Tiger("Tiger", meat);
    panimal->Eat();           // display: Tiger::Eat
    cout << *panimal << endl;  // display: Tiger likes to eat meat.
    delete panimal;

    return 0;
}
```

//Screen capture of running result



```
C:\Users\steve\Desktop\oop_exam\2.exe
Wolf::Eat
wolf likes to eat meat
Tiger::Eat
Tiger likes to eat meat

--------------------------------

Process exited after 0.2116 seconds with return value 0
请按任意键继续. . .
```

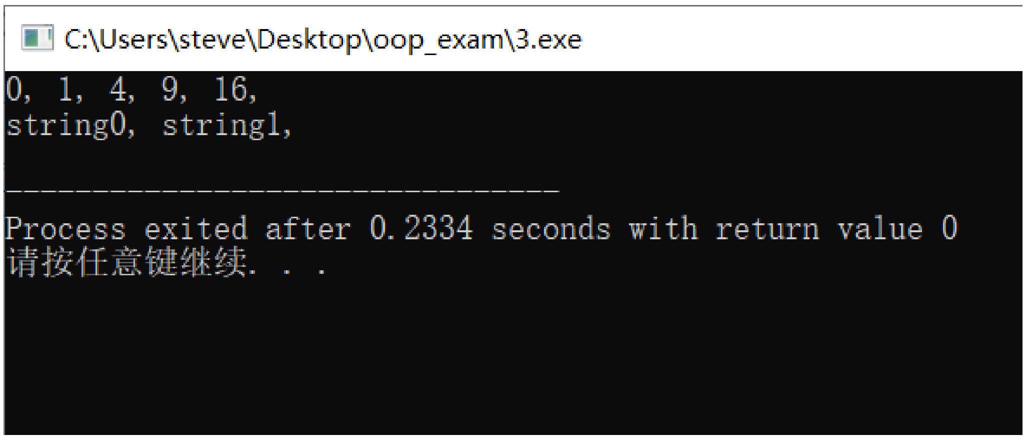
3（35 points) Define an array class template MArray which can be used as in the following main(). (Note: you are not allowed to define MArray based on the templates in the C++ standard library).

#include <iostream>	int main()
# include <string>	{
using namespace std;	MArray<int> intArray(5); // 5 is the number of elements
// **Your definition of MArray:**	for (int i = 0; i<5; i++)
……	intArray[i] = i * i;
	MArray<string> stringArray(2);
	stringArray [0] = ″string0″;
	stringArray [1] = ″string1″;
	MArray<string> stringArray1 = stringArray;
	cout << intArray << endl; ***// display:*** **0, 1, 4, 9, 16,**
	cout << stringArray1 << endl; ***// display:*** **string0, string1,**
	return 0;
	}

//Your codes with necessary explanations:

```
#include <iostream>
#include <string>

using namespace std;

template <typename T> // 类模板，模板名为 T
class MArray{
    private:
        T *a;
        // T 类型的指针 a，用于存放数组内容
        int sz;
        // 表示数组长度大小
    public:
        MArray(int n = 0){
            /*
                带有默认参数值的有参构造函数
                默认值为 0
                初始化 MArray 对象的数组大小
            */
            sz = n;
            a = new T [sz];
            // 为 a 动态分配数据类型为 T 的 大小为 sz 的空间
            // 指针指向首地址
        }
        MArray(const MArray &d_arr){
            /*
                拷贝构造函数
                若不设计此函数，调用默认的赋值函数（浅拷贝）
                这会使得两个对象的 T 类型的指针指向同一个地址
                使得在析构的时候同一个空间被释放后还会“被释放一次”，造成错误
                所以有必要设计此拷贝构造函数
            */
            sz = d_arr.sz;
            // 将 d_arr 中的数组大小赋值给当前对象
            a = new T [sz];
            // 为 a 动态分配空间
            for(int i = 0; i < sz; i ++) a[i] = d_arr.a[i];
            // 将 d_arr 中的数组的数据内容赋值给当前对象
        }
        ~MArray(){
            // 析构函数
            delete [] a;
            // 回收在构造函数中为 a 动态分配的空间
        }
        T& operator [] (int ind){
            // 重载下标运算符，返回类型为 T 的引用，使得其能作为左值被修改
            return a[ind];
        }
        friend ostream& operator << (ostream &os, const MArray &d_arr){
            // 友元函数，重载流插入运算符
            for(int i = 0; i < d_arr.sz; i ++)
                os << d_arr.a[i] << ", ";
            // 从头到尾输出类对象 d_arr 的数组中的内容，中间用", "隔开
            return os;
        }
};

int main(){
    MArray <int> intArray(5);    // 5 is the number of elements
    for (int i = 0; i < 5; i ++)
        intArray[i] = i * i;
    MArray <string> stringArray(2);
    stringArray[0] = "string0";
    stringArray[1] = "string1";
    MArray <string> stringArray1 = stringArray;
    cout << intArray << endl;    // display: 0, 1, 4, 9, 16,
    cout << stringArray1 << endl; // display: string0, string1,
    return 0;
}
```

//Screen capture of running result



```
0, 1, 4, 9, 16,
string0, string1,

--------------------------------
Process exited after 0.2334 seconds with return value 0
请按任意键继续. . .
```