高级程序设计 **2020 春**

实验报告

指导老师：陈家骏 黄书剑 姓名：王晨渊, 学号：181220057

1 概念题

1.1 简述 C++ 中继承的概念，并说明继承有哪些优点。

在定义一个新的类时，将某个或某些已有类的所有特征包含进来，然后在新的类中再定义新的特征或对已包含的特征进行重定义（修改）

继承的优点在于：对实物按层次进行分类；对概念进行组合；支持软件的增量开发。

1.2 C++ 提供几种继承方式？请列举派生类中继承自基类成员的各种访问控制。

C++ 有 public private 和 protected 三种继承方式。

若采用 public 继承

基类 public-> 派生类 public

基类 protected-> 派生类 protected

基类 private-> 派生类不可直接使用

若采用 protected 继承

基类 public-> 派生类 protected

基类 protected-> 派生类 protected

基类 private-> 派生类不可直接使用

若采用 private 继承, 基类的 public protected 和 private 均不能直接使用。

注意，以上的使用指的是派生类对象的调用和派生类的派生类对于基类成员的使用。在派生类中使用基类的成员，其访问控制与继承方式无关，与基类中原始的访问控制保持一致。

1.3 简述 C++ 中转移构造函数与转移赋值函数的作用。

直接转移临时或即将消亡的对象占用的额外系统资源，提高使用临时或即将消亡的对象用于构造或赋值新的对象的效率。

2 编程题

2.1 自动驾驶

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using std::cout;
using std::endl;
class Car
{
public:
    // fRadian:方向盘旋转角度; fSpeed:汽车行驶速度
    // fDeltaTime:每隔多久更新一次行驶状态
    void drive(float fRadian, float fSpeed, float fDeltaTime)
    {
        cout << "Driving..." << endl;
        cout << "Radian:" << fRadian << endl;
        cout << "Speed:" << fSpeed << endl;
        cout << "DeltaTime:" << fDeltaTime << endl;
    }
};

class AutopilotCar:public Car
{
public:
    void autoDrive()
    {
        cout<<"autoDriving..."<<endl;
    }
};


class UpgradedAutopilotCar:public AutopilotCar
{
public:
    void optimizedDrive(float fRadian, float fSpeed, float fDeltaTime)
    {
        drive(fRadian+(int)rand()%5,fSpeed,fDeltaTime);
    }
protected:
    AutopilotCar::drive;
};
```

```
40
41      class PerfectCar:private Car
42      {
43      public:
44          void autoDrive()
45          {
46              cout<<"autoDriving..."<<endl;
47          }
48      };
```

2.2 程序运行结果

我预期的运行结果是

Base()
A()
Derive()
~Derive()
~A()
~Base()
Base()
A()
Derive()
Base()
A()
Derive()
Base()
A()
Derive(const Derive&)
Base()
A()
Derive(const Derive&)
~Derive()
~A()
~Base()
~Derive()
~A()
~Base()
Derive& operator=(const Derive& testDerive)
~Derive()

~A()
~Base()
~Derive()
~A()
~Base()

但实际上并不是，查阅资料了解，g++ 会采取 RVO 优化，即使使用-o0 编译也不能关闭 RVO 优化。RVO 优化导致 testFunc() 在返回时不会调用拷贝构造函数生成临时变量，而是直接利用栈上已有的变量给赋值操作符重载函数传参。真实的运行结果如下。

Base()
A()
Derive()
~Derive()
~A()
~Base()
Base()
A()
Derive()
Base()
A()
Derive()
Base()
A()
Derive(const Derive&)
~Derive()
~A()
~Base()
Derive& operator=(const Derive& testDerive)
~Derive()
~A()
~Base()
~Derive()
~A()
~Base()

想要取消 RVO 优化必须加 -fno-elide-constructors 选项