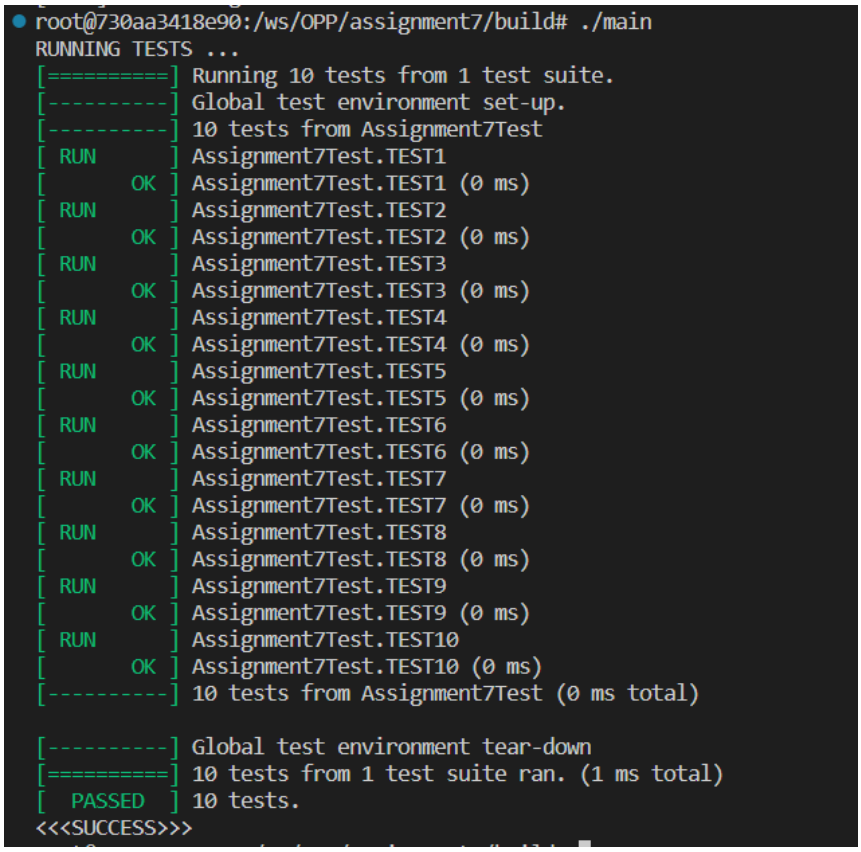
1 单元测试通过截图



2 核心代码解释

```
class Ingredient{
public:
    virtual ~Ingredient(){}//这里要用虚函数，不然用基类指针指向子类的时候无法析构
    double get_price_unit(){
        return price_unit;
    }
    size_t get_units(){
        return units;
    }
    std::string get_name(){
        return name;
    }
    double price(){
        return price_unit*units;
    }
    Ingredient* Copy(){ //new一份一样的出来
        auto p= new Ingredient(price_unit,units);
        p->name=name;
        return p;
    }

protected:
```

```
    Ingredient(double price_unit_, size_t
units_):price_unit(price_unit_),units(units_){}
    double price_unit;
    size_t units;
    std::string name;
};
```

```
class EspressoBased{
    ......
    virtual ~EspressoBased(); //析构需要用虚函数
    ......

};
```

```
Cappuccino::~Cappuccino() //最后析构的时候先调子类的再调基类的
{
    for(const auto& i : side_items){
        delete i;
    }
    side_items.clear();
}
Cappuccino::Cappuccino(const Cappuccino& cap):EspressoBased{cap}{
    //先调用基类构造函数 再调用子类的
    for(auto i : cap.side_items){
        this->side_items.push_back(i->Copy());
    }
}
void Cappuccino::operator=(const Cappuccino& esp){
    if((&esp)==this) return; //判断这俩是不是同一个对象
    this->name=esp.name;
    this->ingredients.clear();//清空vector
    for(auto i : esp.ingredients){
        this->ingredients.push_back(i->Copy());//new一个出来加入vector
    }
    this->side_items.clear();//清空vector
    for(auto i : esp.side_items){
        this->side_items.push_back(i->Copy());
    }
}
double Cappuccino:: price(){
    double ans=0;
    for(auto i : ingredients){//原料价格
        ans+=i->price();
    }
    for(auto i : side_items){//小料价格
        ans+=i->price();
    }
    return ans;
}
```

3 问答题

Question. why do you think the the constructor and the variables are defined as `protected` and not `private`? answer the question in your report after you completed your code.

Answer. 变量需要被子类继承，基类构造函数不被继承但是会在构造派生类对象时被调用，如果是 private 这两项都无法满足，所以不行。

Question. what happens if you define the destructor i.e. `~EspressoBased()` in the protected section? explain your answer in your report.

Answer. 用一个例子把这个问题具体化：

```
#include<iostream>
using namespace std;
class A {
protected:
    A() { std::cout << "A()" << std::endl; }
    virtual ~A() { std::cout << "~A()" << std::endl; }
};

class B : public A{
public:
    B() { std::cout << "B()" << std::endl; }
    ~B() { std::cout << "~B()" << std::endl; }
};

int main(){
    /*
    一般来说，一个类如果做父类，那么它应该有析构函数并且这个析构函数都应该是一个虚函数。
    什么情况下父类中可以没有析构函数或者析构函数可以不为虚函数：
    1)子类并没有析构函数（不需要在析构函数中释放任何new出来的数据）。
    2)代码中不会出现父类指针指向子类对象（多态）的情形。
    为防止用父类指针new一个子类对象，可以把父类的析构函数用protected来修饰*/
    A* p = new B();
    delete p; //错误,protected继承时，~B无法调用~A,即无法访问，所以会编译失败。
    return 0;
}
```