Student Name: 马越
Student ID: 2022141460097

C++ 面向对象程序设计

Assignment 7

Implementation of Ingredient Class

在基类中，实现函数`get_price_unit()`、`get_units()`与`price()`以减少派生类中的重复代码。基类中包含虚函数，因此声明一个虚析构函数，以避免在通过基类指针析构派生类时发生内存泄漏。定义一个纯虚函数`clone()`，以解决无法通过虚基类指针深拷贝派生类对象的问题。

Listing 1: ingredient.h

```h
#ifndef INGREDIENT_H
#define INGREDIENT_H

#include <string>

class Ingredient {
public:
    double get_price_unit() { return price_unit; }
    size_t get_units() { return units; }
    double price() { return price_unit * units; }

    virtual std::string get_name() = 0;
    virtual ~Ingredient() = default;

    virtual Ingredient* clone() const = 0;

protected:
    Ingredient(double price_unit, size_t units) {
        this->price_unit = price_unit;
        this->units = units;
    }

    Ingredient() = default;
    Ingredient(const Ingredient& other) = default;
    Ingredient& operator=(const Ingredient& other) = default;

    double price_unit = -1;
    size_t units = -1;
    std::string name = "Unspecified";
};
```

```
31
32 #endif // INGREDIENT_H
```

A1: 如果将基类中的构造函数与成员声明为`private`，会导致基类与派生类无法创建实例，派生类无法继承与访问这些成员。

Sub Ingredients

从`Ingredient`类中派生出需要的成分。利用宏提高代码复用率。

Listing 2: sub_ingredient.h

```
1  #ifndef SUB_INGREDIENTS_H
2  #define SUB_INGREDIENTS_H
3
4  #include "ingredient.h"
5
6  #define DEFCLASS(NAME, PRICE) class NAME: public Ingredient \
7  {                                                           \
8  public:                                                     \
9      explicit NAME(size_t units) : Ingredient(PRICE, units)  \
10     {                                                       \
11         this->name = #NAME;                                 \
12     }                                                       \
13                                                             \
14     virtual std::string get_name() override {return name;}  \
15     virtual ~NAME() {}                                      \
16                                                             \
17     virtual NAME* clone() const override {                  \
18         return new NAME(*this);                             \
19     }                                                       \
20 }                                                           \
21
22 DEFCLASS(Cinnamon, 5);
23 DEFCLASS(Chocolate, 5);
24 DEFCLASS(Sugar, 1);
25 DEFCLASS(Cookie, 10);
26 DEFCLASS(Espresso, 15);
27 DEFCLASS(Milk, 10);
28 DEFCLASS(MilkFoam, 5);
29 DEFCLASS(Water, 1);
30
31 #endif // SUB_INGREDIENTS_H
```

EspressoBased Class

在复制`Ingredient`类的派生类实例时，调用`clone()`方法。

Listing 3: espresso_base.h

```h
#ifndef ESPRESSO_BASED_H
#define ESPRESSO_BASED_H

#include <vector>
#include <string>
#include <numeric>

#include "ingredient.h"

class EspressoBased {
public:
    std::string get_name();
    double price();
    std::vector<Ingredient*>& get_ingredients();

    [[maybe_unused]] void brew() {}

    virtual ~EspressoBased();

protected:
    EspressoBased() = default;
    EspressoBased(const EspressoBased& other);
    void operator=(const EspressoBased& other);

    std::vector<Ingredient*> ingredients;
    std::string name;
};

#endif // ESPRESSO_BASED_H
```

Listing 4: espresso_base.cpp

```cpp
#include "espresso_based.h"

EspressoBased::~EspressoBased() {
    for (auto& ingredient: ingredients)
        delete ingredient;
    ingredients.clear();
}

std::vector<Ingredient *>& EspressoBased::get_ingredients() {
    return ingredients;
```

```cpp
}

EspressoBased::EspressoBased(const EspressoBased &other) {
    name = other.name;
    for(const auto& ingredient : other.ingredients)
        ingredients.push_back(ingredient->clone());
}

void EspressoBased::operator=(const EspressoBased &other) {
    if (this == &other)
        return;
    for (auto& ingredient: ingredients)
        delete ingredient;
    ingredients.clear();
    ingredients.reserve(other.ingredients.size());
    for(const auto& ingredient : other.ingredients)
        ingredients.push_back(ingredient->clone());
}

double EspressoBased::price() {
    return std::accumulate(ingredients.begin(), ingredients.end(), 0.0,
                           [](double sum, Ingredient* i) { return sum + i->price(); });
}

std::string EspressoBased::get_name() {
    return name;
}
```

A2: 如果将基类的析构函数声明在`protected`部分，会导致析构函数只能被派生类调用，如果创建基类实例将无法被正确析构，可能会引发内存泄漏。

Cappuccino Class

从`EspressoBased`类中派生出`Cappuccino`类，构造时默认在`ingredients`中加入预置的成分，并实现`side_items`相关的成员与方法。

Listing 5: cappuccino.h

```cpp
#ifndef CAPPUCCINO_H
#define CAPPUCCINO_H

#include "sub_ingredients.h"
#include "espresso_based.h"

class Cappuccino: public EspressoBased {
public:
```

```
    Cappuccino();
    Cappuccino(const Cappuccino& other);
    ~Cappuccino() override;
    void operator=(const Cappuccino& other);

    void add_side_item(Ingredient* side);
    std::vector<Ingredient*>& get_side_items();

private:
    std::vector<Ingredient*> side_items;
};

#endif // CAPPUCCINO_H
```

Listing 6: cappuccino.cpp

```
#include "cappuccino.h"

Cappuccino::Cappuccino(): EspressoBased() {
    name = "Cappuccino";
    ingredients = {
        new Espresso(2),
        new Milk(2),
        new MilkFoam(1),
    };
}

Cappuccino::Cappuccino(const Cappuccino &other): EspressoBased(other) {
    for (const auto &item: other.side_items)
        side_items.push_back(item->clone());
}

Cappuccino::~Cappuccino() {
    for (auto& ingredient: ingredients)
        delete ingredient;
    ingredients.clear();
    side_items.clear();
}

void Cappuccino::operator=(const Cappuccino &other) {
    if (this == &other)
        return;
    for (auto &item : side_items)
        delete item;
    side_items.clear();
    side_items.reserve(other.side_items.size());
    for (const auto &item : other.side_items)
```

```
        side_items.push_back(item->clone());
}

void Cappuccino::add_side_item(Ingredient *side) {
    ingredients.push_back(side);
    side_items.push_back(side);
}

std::vector<Ingredient *> &Cappuccino::get_side_items() {
    return side_items;
}
```

Mocha Class

Mocha类的实现类似Cappuccino类。

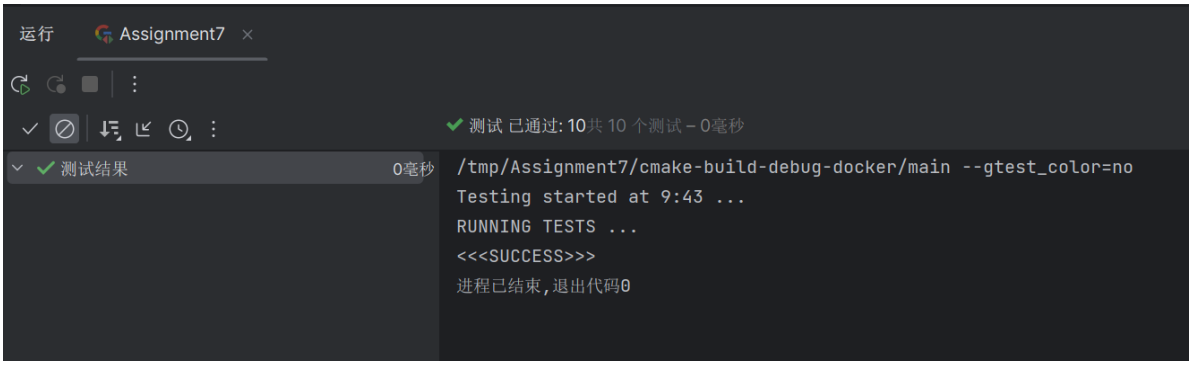
Listing 7: mocha.h

```
#ifndef MOCHA_H
#define MOCHA_H

#include "sub_ingredients.h"
#include "espresso_based.h"

class Mocha: public EspressoBased {
public:
    Mocha();
    Mocha(const Mocha& other);
    ~Mocha() override;
    void operator=(const Mocha& other);

    void add_side_item(Ingredient* side);
    std::vector<Ingredient*>& get_side_items();

private:
    std::vector<Ingredient*> side_items;
};

#endif // MOCHA_H
```

Listing 8: mocha.cpp

```
#include "mocha.h"

Mocha::Mocha(): EspressoBased() {
    name = "Mocha";
```

```
    ingredients = {
        new Espresso(2),
        new Milk(2),
        new MilkFoam(1),
        new Chocolate(1),
    };
}

Mocha::Mocha(const Mocha &other): EspressoBased(other) {
    for (const auto &item: other.side_items)
        side_items.push_back(item->clone());
}

Mocha::~Mocha() {
    for (auto& ingredient: ingredients)
        delete ingredient;
    ingredients.clear();
    side_items.clear();
}

void Mocha::operator=(const Mocha &other) {
    if (this == &other)
        return;
    for (auto &item : side_items)
        delete item;
    side_items.clear();
    side_items.reserve(other.side_items.size());
    for (const auto &item : other.side_items)
        side_items.push_back(item->clone());
}

void Mocha::add_side_item(Ingredient *side) {
    ingredients.push_back(side);
    side_items.push_back(side);
}

std::vector<Ingredient *> &Mocha::get_side_items() {
    return side_items;
}
```


运行
Assignment7
测试 已通过: 10共 10 个测试 – 0毫秒
测试结果
0毫秒
/tmp/Assignment7/cmake-build-debug-docker/main --gtest_color=no
Testing started at 9:43 ...
RUNNING TESTS ...
<<<SUCCESS>>>
进程已结束,退出代码0