M.Sc. in High-Performance Computing
5614. C++ Programming
Assignment 3

R. Morrin <rmorrin@maths.tcd.ie>

February 23, 2025

Instructions

- Gather all your code and pdf/text files into a single tar-ball.
- Submit this tar-ball via `Blackboard` before 23:59 Fri 14th Mar..
- Any questions, please ask me in class or email me at rmorrin@maths.tcd.ie. Don't wait until right before the deadline!
- Late submissions without prior arrangement or a valid explanation will result in reduced marks.
- All non-code answers should be submitted in a single pdf or text file. If you write and scan in, please make sure I can read it!
- The non-code questions only need short answers - not an essay!
- Read notes at end of this document.
- Remember to use comments when appropriate.

Overview of Assignment 3

The main purpose of the assignment is to introduce inheritance and dynamic polymorphism in C++.

Some of you will be taking the Financial Applications course, but you don't need to be taking it for the assignment. The examples here are trivial. Recall that the basic principle when using OOP is that your class design should represent the real world problem you are solving.

The assignment consists of a simple toy example where you have a portfolio of three different types of financial instruments that you invested in. All three different types have different payoffs depending on their parameters. So we will have three different classes to represent each type. Also, because each of the three **is-a** trade, we will also create a base `Trade` class. We can then pass around pointers (or references) to the derived classes as pointers (or references) to the base class.

All of your trades will mature on the same day. And you want to calculate the payoff from your portfolio, and the profit, given a particular level of the underlying.

Note on comments

10% of available marks will be allocated for comments where appropriate. Similar to earlier assignments, please try to use some Doxygen comments. You can see some basics here `http://doxygen.nl/manual/docblocks.html`. Doxygen is more useful for C++ than C. I can show my generated documentation for this code in the next class if you want to see it. The generated class inheritance diagram for the for my code for the assignment is shown in Fig. 1.

General

The classes in this assignment are trivial. In real-world code you might follow the rule of zero[1] for these. Because this is a assignment for practice, we will add destructors which print simple statements to screen.

The best rule to follow in real-world situations is C.21 of the `C++ Core Guidelines`[2] but for this assignment, the code you write should follow the Rule of Five Defaults[3]. The difference between those approaches is that for the latter you effectively always start off with a "boilerplate" where all five are defaulted, whereas for the former you will still try to follow the Rule of Zero, but once you do something with one, then you default the others.

Note that because we mark some member variables as const, that means that the classes cannot be assigned to. So we will delete the assignment operators.

Forward Contract

A forward agreement is an agreement to pay a forward price F in order to receive the underlying asset at maturity. The payoff is given by

$$S_T - F \tag{1}$$

where S_T is the underlying asset price at maturity.

Call Option

A call option gives the owner the right, but not the obligation, to purchase the underlying asset for a fixed price called the strike price K. The payoff from a call option is given by

$$\max(S_T - K, 0) \tag{2}$$

Put Option

A call option gives the owner the right, but not the obligation, to sell the underlying asset for a fixed price called the strike Price K. The payoff from a put option is given by

$$\max(K - S_T, 0) \tag{3}$$

Q1. Makefile (5%)

Write a very simple `Makefile` to take care of your assignment. Your file will have four targets:

1. portfolio.o
2. assignment3.o
3. assignment3

[1]`http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c20-if-you-can-avoid-defining-default-operations-do`

[2]`http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c21-if-you-define-or-delete-any-copy-move-or-destructor-function-define-or-delete-them-all`

[3]`http://scottmeyers.blogspot.com/2014/03/a-concern-about-rule-of-zero.html`

4. clean

(Although you can merge the second and third if you want).
`make clean` should just delete the executable and any intermediate files. Don't forget to mark it as a phony target. You can also add an additional `all` target which will build `assignment3`.
For full marks, use some Makefile variables such as

```
CXX         := g++
CXXFLAGS    := -Wextra -Wall -std=c++23
```

Try to use some automatic variables such as `$@` and `$<` in your makefile commands. You can use pattern rules if you have covered them elsewhere but you will need to remember to take care of the header file dependencies (I didn't cover any pattern rules).

Q2. Dynamic Polymorphism (30%)

Base class: `Trade`

You are given a partially complete definition of the `Trade` class in instruments.h

```
class Trade
{
public:
        // TODO:
        // (1) Define copy and move constructors as default
        // (2) Delete assignment operators

        Trade() : cost {0}{
                std::cout << "Trade (base class) Constructor (Default)\n";
        }

        Trade (double const cost) : cost {cost} {
                std::cout << "Trade (base class) Constructor (overloaded)\n";
        }

        virtual ~Trade (){
                std::cout << "Trade (base class) Destructor\n";
        }

        // TODO:
        // (3) declare payoff as a pure virtual constant member function
        // (4) Grant access to private member of this class
        //     to any non-member functions that need it

private:
        double const cost;      ///< Holds premium, or cost to enter the trad
};
```

You will need to add some lines where `TODO` is marked in the comments, but you will do this as you are completing the sections below.

The `payoff` constant member function takes ones parameter, S_T, and calculates and returns the payoff for that trade at that underlying price. Declare the function as a *pure virtual* function.

(A) `Forward` class

Write a class `Forward` which derives from `Trade`. It should have one private member variable `double const forward_price` which stores the forward price. You should override the base class `payoff` function so that, when called,

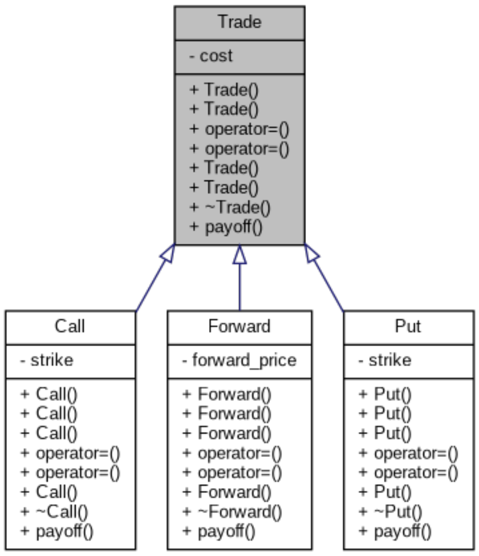

Figure 1: Inheritance hierarchy for Q2

it returns the payoff. from the contract (See Eq. 1). You do not want to further override this function in more derived classes if you later add any.

Explicitly delete the default `Forward` constructor. This will have no practical effect to your code (it wouldn't be generated anyway) but serves as a comment to show that you intended for there to be no default constructor.

The constructor which you write will take one parameter which is the forward price. `Forward (double fp)`. The forward contracts will have zero cost.

(B) `Call` class

Similarly, write a class `Call` which derives from `Trade`. It should have one private member variable `double const strike` which stores the strike price. You should override the base class `payoff` function so that, when called, it returns the payoff from the contract (See Eq. 2). You do not want to further override this function in more derived classes if you later add any. Similar to above, explicitly delete the default `Call` constructor.

The constructor which you write will take two parameters - strike price and cost. `Call (double cost, double k)`.

(C) `Put` class

Write a third class `Put` which also derives from `Trade`. It should have one private member variable `double const strike` which stores the strike price. You should override the base class `payoff` function so that, when called, it returns the payoff from the contract (See Eq. 3). You do not want to further override this function in more derived classes if you later add any. Again explicitly delete the default `Put` constructor.

The constructor which you write will take two parameters - strike price and cost. `Put (double cost, double k)`.

Q3. Non-member functions (10%)

Download the file portfolio.h which contains two function declarations. You will write the corresponding function definitions in portfolio.cc.

(A) Function to compute portfolio payoff (5)

Write

```
double portfolio_payoff(std::vector<Trade const *> const & trades, double const S_T)
```

which iterates through the container of `Trade*`, and calculates, and returns, the overall payoff for the portfolio for a given underlying asset price S_T.

(B) Function to calculate portfolio profit (5)

Write the function definition for

```
double portfolio_profit(std::vector<Trade const *> const & trades, double const S_T)
```

which calculates the overall profit for the portfolio.

Q4. Putting it all together. (5%)

The `main` function is provided in assignment3.cc. It is copied below. Check that it compiles and works as expected.

assignment3.cc

```
/**
 * @file assignment3.cc
 * @brief Main function for 5614 Assignment 3 2025
 * @author R. Morrin
 * @version 5.0
 * @date 2025-02-22
 */

#include <iostream>
#include <vector>
#include "instruments.h"
#include "portfolio.h"

int main()
{
        std::vector<Trade const *> trades;

        // Add some trades
        trades.push_back(new const Forward {6.0});
        trades.push_back(new const Forward {2.0});
        trades.push_back(new const Call {1.0, 6.0});
        trades.push_back(new const Call {5.5, 3.0});
        trades.push_back(new const Put {4.0, 7.0});
        trades.push_back(new const Put {4.5, 6.0});

        std::cout <<  "\nPortfolio:\nS_T" << '\t' << "Payoff" << '\t' << "Profit\n";
        for (int stock_price = 0; stock_price <= 15; ++stock_price) {
                std::cout <<  stock_price << '\t' << portfolio_payoff(trades, stock_price)
                        << '\t' << portfolio_profit(trades, stock_price) << "\n";
        }
        std::cout << '\n';

        // Delete allocated trades manually
        for (auto i = 0u; i < trades.size(); ++i) {
                delete trades[i];
        }

        return 0;
}
```

Sample output using provided `main` function

Output

```
> ./assignment3
Trade (base class) Constructor (Default)
Constructor for Forward with forward price 6
Trade (base class) Constructor (Default)
Constructor for Forward with forward price 2
Trade (base class) Constructor (overloaded)
Creating Call with strike 6. Premium paid 1
Trade (base class) Constructor (overloaded)
Creating Call with strike 3. Premium paid 5.5
Trade (base class) Constructor (overloaded)
Creating Put with strike 7. Premium paid 4
Trade (base class) Constructor (overloaded)
Creating Put with strike 6. Premium paid 4.5

Portfolio:
S_T     Payoff  Profit
0       5       -10
1       5       -10
2       5       -10
3       5       -10
4       6       -9
5       7       -8
6       8       -7
7       11      -4
8       15      0
9       19      4
10      23      8
11      27      12
12      31      16
13      35      20
14      39      24
15      43      28

Deleting Forward with forward price 6
Trade (base class) Destructor
Deleting Forward with forward price 2
Trade (base class) Destructor
Destroying Call with strike 6
Trade (base class) Destructor
Destroying Call with strike 3
Trade (base class) Destructor
Destroying Put with strike 7
Trade (base class) Destructor
Destroying Put with strike 6
Trade (base class) Destructor
```

Q5. Written Questions (30%)

(A) `virtual` & `override` (5)

What does the keyword virtual mean in the base-class `Trade`? Explain, or give a simple example, of how using the keyword virtual would change behaviour in a piece of code vs. not using it.

Is there any negative effect of having a virtual function in your class?

You were asked to declare `payoff()` as pure virtual. What is the effect of doing this?

What is the effect of using the specifier override in a derived class. All else equal, will there be any difference in the resulting binary code when using override vs. not using it? If yes, how? If no, why not?

(B) Polymorphism (5)

What is the difference between *static* and *dynamic* polymorphism? Explain in your own words (only a few sentences) how dynamic polymorphism is used in the code you wrote for the `Trade` examples.

(C) Compiler issues and optimisations (10)

Download the file widget1.cc. The main parts of the code are shown below.

widget1.cc

```cpp
 1 struct Widget
 2 {
 3         Widget (std::size_t in)
 4                 : vals (in)
 5                 , original_N {in} {
 6                         // Populate with 0,...,in-1
 7                         std::iota(vals.begin(), vals.end() ,0);
 8                   }
 9
10         ///     ~Widget(){ std::cout << "Deleting Widget\n";}    // (α)
11
12         std::list<int> vals;
13
14         std::size_t original_N;                                   // (β)
15         ///     std::size_t const original_N;                     // (γ)
16 };
17
18 int main()
19 {
20         Widget X {10};
21         Widget Y {X};
22         Widget Z {std::move(X)};
23         Z=Y;
24         X=std::move(Z);
25
26         for (auto const & i : X.vals) {
27                 std::cout << i << '\n';
28         }
29
30         return 0;
31 }
```

Compile the code and dump out and filter from the symbol table. You should get a result similar to:

```
$ g++ --std=c++23   -Wextra -Wall  -o widget1 widget1.cc
$ nm -C widget1 | grep Widget | sort | uniq
00000000000015a8 W Widget::Widget(unsigned long)
000000000000163c W Widget::~Widget()
0000000000001658 W Widget::Widget(Widget const&)
0000000000001736 W Widget::Widget(Widget&&)
000000000000176c W Widget::operator=(Widget const&)
00000000000017a6 W Widget::operator=(Widget&&)
0000000000001a89 W std::remove_reference<Widget&>::type&& std::move<Widget&>(Widget&)
```

Defining own destructor

When you uncomment `Line 10` (α) above, and recompile, you should see something like:

```
$ g++ --std=c++23   -Wextra -Wall  -o widget1 widget1.cc
$ nm -C widget1 | grep Widget | sort | uniq
nm -C widget1 | grep Widget | sort | uniq
0000000000001450 W Widget::Widget(unsigned long)
00000000000014e4 W Widget::~Widget()
0000000000001518 W Widget::Widget(Widget const&)
000000000000154e W Widget::operator=(Widget const&)
0000000000001845 W std::remove_reference<Widget&>::type&& std::move<Widget&>(Widget&)
```

Explain what has happened here and why. How could this affect the efficiency of your program? How are `Line 22` and `Line 24` still managing to work?

Changing to `const`

Comment `Line 14` (β) and uncomment `Line 15` (γ). Try to compile the code. Does it compile? If not, why not?

(D) Compiler issues and optimisations 2 (5)

Download widget2.cc

widget2.cc

```
 1 struct Widget2
 2 {
 3         Widget2 (std::size_t in)
 4                 : vals (in)
 5                   , original_N {in} {
 6                         std::cout << "Constructor" << std::endl;
 7                         // Populate with 0,...,in-1
 8                         std::iota(vals.begin(), vals.end() ,0);
 9                   }
10
11         Widget2(Widget2 const& rhs){     // Copy Constructor
12                 original_N = rhs.original_N;
13                 vals = vals;
14                 std::cout << "Copy Constructor" << std::endl;
15         }
16
17         Widget2(Widget2 && rhs){         Move Constructor
18                 original_N = rhs.original_N;
19                 vals = std::move(rhs.vals);
20                 std::cout << "Move Constructor" << std::endl;
21         }
22
23         std::list<int> vals;
24         private:
25         std::size_t original_N;
26 };
27
28 Widget2 incremented_copy(Widget2 const& w){
29         std::cout << "Entering incremented Copy" << std::endl;
30         Widget2 incremented {w};
31         for (auto &i : incremented.vals) {
32                 i++;
33         }
34         std::cout << "Leaving incremented Copy" << std::endl;
35         return(incremented);                            //  (δ)
36         // return(std::move(incremented));              //  (ε)
37 }
38
39 int main()
40 {
41         Widget2 X {10};
42         Widget2 W {incremented_copy(X)};
43
44         for (auto const & i : W.vals) {
45                 std::cout << i << '\n';
46         }
47
48         return 0;
49 }
```

When you compile and run this program you should see something like

```
$ g++ --std=c++23  -Wextra -Wall  -o widget2 widget2.cc
$ ./widget2
Constructor
Entering incremented Copy
Copy Constructor
Leaving incremented Copy
```

Now comment out `Line 35` (δ) and uncomment `Line 36` (ϵ). You should get the output similar to the below

```
$ g++ --std=c++23 -o widget2 widget2.cc
$ ./widget2
Constructor
Entering incremented Copy
Copy Constructor
Leaving incremented Copy
Move Constructor
```

(Hint: Turn warnings back on for more information). Next comment out `Lines 17-21` (i.e. the move constructor). Compile and run again

```
$ g++ --std=c++23   -o widget2 widget2.cc
$ ./widget2
Constructor
Entering incremented Copy
Copy Constructor
Leaving incremented Copy
Copy Constructor
```

Explain what is happening in each of these steps. i.e. why don't we see any message about `Move Constructor` or `Copy Constructor` in the first output. Why do we see "`Move Constructor`" in the second output, and "`Copy Constructor`" in the third output. What is your conclusion about using `std::move()` to `return` the function result?

(E) Misc. (5)

(i) Virtual Inheritance

When might you need to specify virtual inheritance when deriving your classes?

(ii) Singleton

What is a singleton pattern? Explain a few sentences how you might implement one in C++. (You don't need an essay - just say what you want to achieve, and how you achieve it. Mention the effect of static etc.).

Q6. AI example for polymorphism (20%)

Have an AI tool generate an example of polymorphism in C++. Include the output in AI_polymorphism.cc in your submission. In your writeup document, say what you tool you used and what you asked it to do i.e. the command/question you used to have it generate the example. (Also include that information in a comment at the start of file with the code.)

Explain what the output is. First generally as if you were explaining it generally to a beginner, and then you can get more specific. Is the base class an abstract base class? Are there explicit constructors/destructors etc. Is the inheritance public etc. What keywords are used and what do they mean? Could you have added anything to the classes or structs that it gives you? What is the point of doing things this way i.e. what is the point/use of polymorphism etc? Are there any practices in the output code that I have told you not to do/use? Don't just use the AI tool to try and answer the questions.

Include the generated code in your writeup if it fits on a page. If you worked with another student from the class for this question, you should reference that in your writeup, but you should both try to generate your own code separately.

Notes:

- Use universal initialisation syntax {} where you can rather than = or () notations.

- 10% of the respective marks available for each of the coding questions will be given for comments. You do not need to go overboard and please make sure the comments are relevant. i.e. you don't need to say obvious things like `int A; /*This is an integer */`. Note that in the lecture slides, I sometimes include comments such as `/* Function Prototype */` - but this is just for the purposes of the lectures. This would not necessarily be a helpful comment on its own in a proper program. Strive to make your code as self-documenting as possible by using appropriate names for variables and functions.

- I tend to use "East const" notation. If any of you check the CPP Core guidelines, this conflicts with `https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rl-const`. There are arguments for both methods. I use "East const" because I think it is more consistent when teaching the C course. For your own code, pick a style and be consistent. In reality, if you are working, you wilkl have in-house style guidelines which you should follow.

- To create your tarball, `cp` everything you want to submit into a directory e.g. `Assignment3`, and then change directory so that you are in the parent directory of `Assignment3`.

  ```
  tar -cvf [username].assignment3.tar Assignment3
  ```

  where you replace `[username]` with your own username. (Don't leave the square brackets in the name).