Object Oriented Programming in C++ with Real Example

# Objective

## Get familiar with OOP with some real-world project

## Get some understanding of some advanced concept

## Brief the first project build a Binomial Tree Pricer to price Option

## Lay a foundation of the final project

### Review Date Class

Date class is a class to present the Date object in this program. It contains data member of year, month and date. It can also compute the date difference in fraction of year. Refer to file Date.h and Date.cp.

A computer screen shot of a program

AI-generated content may be incorrect.

The first thing to be noted is the Marco of #ifndef DATE\_H #define DATE\_H, then end with #endif. This is meant to avoid function or class within the header file to be copied into compiling unit more than one time, which will leads to error.

The modern way of doing this is to use macro of #prama once

Another thing to noted is the operator overloading to define the “subtraction” operator between 2 Date objects.

A computer screen shot of a program code

AI-generated content may be incorrect.

Similarly, the operator of “<<” to cout or “>>” to “cin”, need to also be re-defined. This is due to original CIN or COUT within STL function has no knowledge of what Date() class is. So in order to output user defined class object to I/O, the operator must be overloaded. This is the power of C++, since developer has the freedom to do almost anything.

## Market data class and other types of market data

### RateCurve class: a class of yield curve

A screen shot of a computer code

AI-generated content may be incorrect.

### VolCurve class: a class of volatility by option expiry

A screen shot of a computer

AI-generated content may be incorrect.

### Market Class: a class of market data, contains all curve and vols

A screen shot of a computer program

AI-generated content may be incorrect.

Refer to sample code in Market.h and Market.cpp

Key points summary:

* The #ifndef #define #endif macros avoid duplicated declaration when included by other files. This is similar to #program once, which is widely used but not in C++ standard (yet).
* Member function of a class manipulates it’s member variables, including public, private members
* We would like a member function Print() to display the market information
* We would like to overload the >> and << operators so that our market is stream-able
* Market.h -> declaration, and Market.Cpp implementation, good practice to separate declaration and implementation for larger project.

### Notice on Operator Overloading

Looks strange at first sight. It is the same as function overloading: reuse the same function name but takes a different set of argument types. Can be used in the normal function call form

(operator>>) (cin, mkt);

or the infix form:

cin >> market; // first Arg operator second Arg

The keyword operator simply makes the latter possible.

### Default Constructor

A default constructor is a constructor with no arguments

Declaration:

ClassName ();

Definition:

ClassName::ClassName () {...}; or

ClassName::ClassName () : m\_A(..), m\_B(..) {...};

Either defined with an empty parameter list, or with default arguments provided for every parameter

If there is no user-defined constructor, the compiler will always declare a dummy default constructor as an inline public member of its class, e.g., our Market class.

When is default constructor called? Market mkt;

### Copy Constructor

Let have a look at another example of market class declaration. Refer to Market.h

Copy constructor is a constructor and Its parameter is T& (seldom used, why?), or const T&, where T is the type of the class

Declaration: ClassName (const ClassName&);

Definition: ClassName (const ClassName&) { // some code here }

It is used to

* Initialize one object from another of the same type
* Copy an object to pass it as an argument to a function
* Copy an object to return it from a function

When is it used?

* *Direct initialization*: T a(b);, where a and b are both of type T
* *Assignment initialization*: T a = b;

It is often having similar usage of operator overloading const T& operator=(const T& other) {}

**If there is no user defined copy constructor, compiler will make a default one: copy everything by value — not desirable for pointers. Here is an example.**

### Copy Assignment Constructor

Assignment constructor will be called when assigning one object value to another object. Its syntax behaves more like a operator”=” overloading but returning the Class& type.

ClassName& operator = (const ClassName&) { // some code here }

A screenshot of a computer program

AI-generated content may be incorrect.

Take note assignment constructor only called when two objects are already instantiated. If not instantiated, then copy constructor is called.

Market mkt1;

Market mkt2;

Mkt2 = mkt1; // assignment constructor will be called.

### Class Data Member as Pointer

* When we have a pointer as a member variable: default copy constructor will copy the address the pointer is pointing to, it ends up having a new pointer points to same underlying resource in the memory, which is multiple pointers point to same underlying data.
* Have 2 or more pointers point to same resource is a very bad practice, why? Recall the pointer delete mechanism. When 1st pointer delete function is called, complier will try to free the underlying data memory first, then recycle the pointer memory. After this action, we have 2nd pointer points to invalid data memory as far as complier knows. This is called “dangling pointer”.
* What is more dangerous is that, when 2nd pointer is out of scope (auto delete), or deleted explicitly, the underlying data memory is already freed (which current thread has no access to). This will create a run time error of memory access violation.
* What shall we do in this case?
  1. Normally we shall avoid using raw pointer as data member of class
  2. If raw pointer is used, then we shall implement the copy constructor, and in which we will define the “deep copy” behaviour. Refer to example in source code of Market.h.

A screen shot of a computer

AI-generated content may be incorrect.

### Destructor Revisit

This is exception safe — destructor of an object will always be called when the object goes out of scope or deleted. The purpose of destructor is to recycle the memory occupied by object back to OS, or free the memory. If pointer is used for class data member, then check NULL is to avoid deleting uninitialized pointer. So it is always good practice to check NULL, since not all constructor functions will initialize the pointer member.

A screenshot of a computer

AI-generated content may be incorrect.

## Summary on Constructor and Destructor

The following three special member functions always come together:

* Copy constructor
* Destructor: remember virtual
* Copy assignment operator =, similar to copy constructor, but called when assigning value of one variable to another existing variable

Note the difference of

T a = b; // copy constructor is called, same as T a(b);

T c;

c = b; // copy assignment operator is called

They all have a default version by the compiler, when one needs to be customized, all three need to be customized.

## Trade Class

Serves as a base class of all different type of trade. Refer to Trade.h.

Pay attention to:

1. Payoff() function is a virtual function with implementation disabled at this class. This means it must be implemented at derived class. It also has implication of we cannot instantiate a Trade class object.
2. Const modifier at end of member function, which means this function cannot change class data member
3. Virtual destructor
4. Protected accessor for data member, which gives the access of these data to derived class

A computer screen with text on it

AI-generated content may be incorrect.

Later we will show case of two child class of trade being created from Trade class, i.e, Bond and Swap class.

**Bond Class**

A **bond** is a fixed-income security that represents a loan made by an investor to a borrower, typically a corporation or government. The issuer of the bond promises to pay back the principal amount at maturity and makes periodic interest payments, known as coupons, until then. Bonds are used for raising capital and are considered a relatively safer investment compared to stocks, depending on the creditworthiness of the issuer.

Valuation of bond:

* MTM for the market traded price x number of lot
* Theoretical price: face value x Pv of future cash flows from a yield curve

## A screen shot of a computer AI-generated content may be incorrect.

## Swap Class

An **interest rate swap** is a financial contract between two parties who exchange cash flows based on different interest rate structures. Typically, one party pays a fixed interest rate while receiving a floating rate, and the other does the opposite. These swaps are used to manage interest rate risk, hedge against fluctuations, or improve financing conditions. They are commonly used by corporations and financial institutions to adjust their exposure to changing market rates.

Valuation of swap:

PV of future cashflows, sum of PV of fixed leg and floating leg.

A math equations and formulas

AI-generated content may be incorrect.

Annuity: is defined as PV of fixed leg where assuming fixed (swap) rate is 1 or 100%.

This gives to NPV of swap equals to Annuity x (market rate of swap – traded rate of swap) for any give valuation date.

Now let’s implement this swap class and compute the NPV using annuity approach.

Pay attention to:

* Inline key word allowing us to implement Payoff function in header file directly.
* We put annuity function as private, which means we do not want outsider to access this function, it is not necessary.

A screenshot of a computer

AI-generated content may be incorrect.

## TreeProduct Class, an interface class for Option type of trade

This is a child class inherited from Trade class, refer to TreeProduct.h

A screen shot of a computer screen

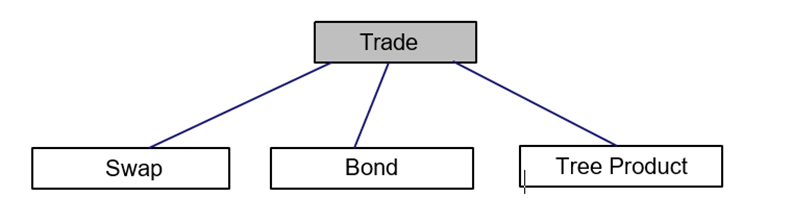
AI-generated content may be incorrect.

Pay attention:

* It also contains 2 more interface function need to be implemented in derived class.
* In the constructor, it has access of parent class data tradeType field.
* Revisit the pure virtual function definition:

The virtual and const = 0 make this class an abstract class, which means the function should be implemented in derived class, otherwise there will be compiling error if there is any line of code want to instantiate any real object of TreeProduct class or derived class.

## Summary of trade hierarchy



## Further into Options Trade

### EuropeanTrade class: TreeProduct

Now let’s see the example of how derived class EuropeanTrade is implemented.

A black background with colorful text

AI-generated content may be incorrect.

A few key points:

1. There is a data member optType with OptionType, which is a Enum. This is meant for type of option.

A screenshot of a computer

AI-generated content may be incorrect.

1. Initializer list, why?
   1. **Performance**: Using initializer lists can lead to better performance in certain scenarios. When you use an initializer list, the compiler can often optimize the construction of objects or elements more effectively than if you were to use assignment in the constructor body.
   2. **Avoiding Default Construction**: In some cases, using initializer lists allows you to avoid default construction of objects, which can be useful when default construction is expensive or unnecessary.
   3. **Immutable Members**: If you have **const** or reference members in your class, initializer lists are the only way to initialize them, as they cannot be assigned to after construction.
   4. **Member Initialization Order**: The order in which members are initialized matters in C++. Initializer lists allow you to specify the order explicitly, which can help avoid potential issues related to the order of member initialization.
   5. **Aggregate Initialization**: Initializer lists can be used for aggregate initialization of arrays and structures, making the code more concise and readable.
2. Is EuropeanTrade a complete class or abstract class, i.e., can we create an object of this class such as:

EuropeanTrade\* eurTrade = new EuropeanTrade();

1. Pay attention to the virtual key word, and function overridden along the hierarchy of classes.
2. Revisit to the public key word in class inheritance

Class Member Accessibility

|  |  |  |  |
| --- | --- | --- | --- |
| Members | Self | Derived Class | Other Class |
| public | ✓ | ✓ | ✓ |
| protected | ✓ | ✓ | × |
| private | ✓ | × | × |

1. Why we are making Payoff() function a const:
   1. a const function is not allowed to change any of its members
   2. Discussion point: why we want const?
2. Pay attention to the function return types of const type&.

* Avoiding Copy Overhead – When returning an object by value, a copy of the object is made, which can be costly for large objects. Returning by reference eliminates this unnecessary duplication.
* Allowing Direct Modification – If a function returns a reference to an existing object, the caller can modify it directly without creating a separate copy.

### SpreadOption class

Further inherited from EuropeanTrade for this specific type of option trade.

A screen shot of a computer screen

AI-generated content may be incorrect.

### Payoff function in payoff.h

In this file, two functions are implemented using namespace PAYOFF. Recall namespace is an encapsulation tool which allows accessible between difference files. Assessing using “::”.

VanillaOption() function gives terminal payoff of a European style option.

A screen shot of a computer program

AI-generated content may be incorrect.

CallSpread function gives the terminal payoff of a call spread option.

A screen shot of a computer program

AI-generated content may be incorrect.

### Pricer Implementation

We have implemented Market, also Trade classes, now we need to implement the Pricer class to define the Pricing analytics logic. This is not mandatory but a good practice since each class is only in charge of their own task. In programming, this is the concept of delegation, which normally means we shall avoid having a large piece of code doing everything, and instead, we shall achieve more “modular” design. This will save a lot of trouble for error debugging, if anything went wrong.

Now let’s have a look of how to implement pricer class.

A black screen with white text

AI-generated content may be incorrect.

Discussion:

Why we have 2 pricing function, one is public and one is private?

A screen shot of a computer code

AI-generated content may be incorrect.

Points to note:

1. From external scope, we can only call Price(), price function will take in Trade\*, which is a base class pointer. It will try to design the type of trade and then call the actual pricing function accordingly.
2. Pay attention to dynamic cast for type determination in C++ in run type.

PriceTree() is meant for any trade types which is inherited from TreeProduct.

Now let’s take a look at how PriceTree() is implemented in this pricer class.

### BinomialTree Class

Inherited from Pricer Class, this is meant to implement a simple version of Binomial tree model. Some revisit on Binomial Tree model and procedure.

The binomial tree model is a popular method for pricing options and other financial derivatives. It works by modeling the possible price movements of an asset over discrete time steps. At each step, the asset price can either move up or down, creating a branching tree structure.

**Key Concepts**:

* Tree Structure – The model starts with an initial asset price and evolves through multiple time steps, where at each step, the price moves up or down by a predetermined factor.
* Risk-Neutral Valuation – Probabilities are assigned to the up and down movements based on market conditions, ensuring that expected returns align with the risk-free rate.
* Option Pricing – By working backward from the final payoffs at expiration, the model calculates the present value of the option, determining its fair price.

A computer screen with text on it

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A bit of summary so far on Pricer. So we have base Pricer() class which can price trade like Swap or Bond, which is not a TreeProduct. In this case, payoff() function is called directly and NPV is computed.

* Pricer::price() -> if not Tree Product -> payoff(): gives the NPV
* Pricer::price() -> Tree product -> priceTree(): which creates a Binomial Tree model to price option type of tree on Tree nodes.

### Extension of Binomial Tree Model

CRR model

The **Cox-Ross-Rubinstein (CRR) model** is a widely used **binomial tree model** for pricing options. It was introduced in 1979 by John Cox, Stephen Ross, and Mark Rubinstein as a discrete-time approach to valuing derivatives.

**Key Features of the CRR Model**

* **Binomial Tree Structure:** The model assumes that the underlying asset price can move **up** or **down** at each time step.
* **Risk-Neutral Valuation:** It calculates option prices using probabilities that ensure expected returns align with the risk-free rate.
* **Flexibility:** The model can price **American** and **European** options, as well as options with dividends.

The only thing different to our 1st implementation in this case is the u and d choice. In CRR model, u and d is given by

u = exp(sigma \* Sqrt(dT))

d = exp(-sigma \* Sqrt(dT))

p = (exp(dT\*sigma) – d ) / (u-d)

## A computer screen shot of a program code AI-generated content may be incorrect.

## A computer screen with text and numbers AI-generated content may be incorrect.

## Pricing American Option

An **American option** is a type of financial derivative that gives the holder the right, but not the obligation, to buy or sell an underlying asset at a predetermined price (**strike price**) at any time **before or on** its expiration date.

**Key Features of American Options**

* **Early Exercise** Unlike European options, which can only be exercised at expiration, American options allow exercise at any point before expiration.
* **Higher Value** Because of the flexibility to exercise early, American options tend to be more valuable than European options.
* **Common Examples** Many stock options traded on U.S. exchanges are American-style.

**Pricing Complexity**

Valuing American options is more complex than European options because early exercise decisions depend on market conditions, dividends, and interest rates. They are often priced using:

* **Binomial Tree Model** (e.g., Cox-Ross-Rubinstein model)
* **Finite Difference Methods**
* **Monte Carlo Simulations**

Now we add American trade class to handle two types of American option into curent Tree product class hierachy.

TreeProduct

AmericanTrade

EuropeanTrade

AmerCallSpread

AmericanOption

EuroCallSpread

EuropeanOption

Discussion point:

* Can we price the American option in the same way of European option? If not, what would be the procedure Pricer need to have?
* What are the procedure like if we price American option on a Tree model?

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

### Pricing American Option with Tree Model

Tree was not invented for European payoff. There are plenty of more efficient methods for pricing European products

* Closed-form solution (analytic pricer)
* Numerical integration (semi-analytic)

Tree is the most natural way to handle American payoff due to the backward iteration. At any point in time, or any node of the tree, we know the continuation value of the product — through calculating the conditional expectation.

American product allows the option holder to exercise the option any time. This translates to the choice to make at any node of the tree: continue holding the option or take the intrinsic value

* Continue holding the option: the option worth its conditional expectation
* Exercise now: the option worth its intrinsic value (terminal payoff)

Optimal exercise strategy is to exercise when intrinsic value worth more than the continuation value: this implies that we will take the max between continuation value and exercise value when we roll backward pricing on the tree nodes.

### Steps of Price American Option with Binomial Tree

Here are the steps involved in pricing an American option using a binomial tree (CRR model):

1. Set up the Binomial Tree:

* Define the parameters: underlying asset price (S), strike price (K), time to expiration (T), volatility (σ), risk-free interest rate (r), and the number of time steps (n).
* Determine the time step size (Δt) by dividing the total time to expiration (T) by the number of time steps (n).

1. Calculate Up and Down Movements and probability

* Calculate the up factor (u) and down factor (d) based on the volatility and time step size.



*  Calculate probability of each node.

1. At last time step T, calculate option value as excised payoff. Then at t = T- Δt, it value is
2. Early Exercise:

At each node of t = T – i\*Δt , compare the calculated option value with the intrinsic value (the difference between the current stock price and the strike price). If the intrinsic value is higher, the option holder may choose to exercise early.

V(t - Δt ) = Max(V(t) \* Df, exercised value)

1. Backward Induction: Move backward through the tree, updating option values based on the early exercise decisions and the option pricing formula.
2. Final result: the option value at the starting node of the tree is the estimated fair price for the American option.

Points to discuss:

Let’s examine our Binomial Tree pricer and see if we can simply handle the American option. The answer is yes. Why?

Pay attention to function of ValueAtNode for American and European trade.

It seems that the procedure of pricing American and European option on binomial tree are largely same, only difference is the handling of “optimum exercise”, which is handled by ValueAtNode function differently.

A computer screen shot of a program code

AI-generated content may be incorrect.

European trades:



American trade:

A black screen with white text

AI-generated content may be incorrect.

It is quite clear that we have achieved one model implementation but be able to price different kind of option. This is the power of OOP.

* Code re-usability
* Logically clear and easy to read
* Scalable to handle other types

### Review on Tree Algorithm by Below Pseudo Code

1: Set up the tree and parameters

2: Initialize the last time slice with final payoff

3: for *k* = *N* − 1 to 0 do

4: for *i* = 0 to *k* do

5: Calculate the continuation value (discounted expectation)

6: Given the information of the tree node, calculate the option value at Node(k, i)

7: end for

8: end for

## Exercise and Discussion

Using Tree model to price American call/put and European call put at different strikes and draw a graph of PV with strike. Discuss your finding.

Trade example 1Y call and put option on stock, suppose *S*0 = 100, *r* = 3%, *σ* = 10%, no dividend, and time step choose 20.

##### American Call versus European Call

−

20

0

20

40

60

80

100

120

140

160

180

200

220

0

50

100

European call price

American call price

Strike

* American call and European call prices are identical
* It is never optimal to early exercise a call option (Is there any exception?)

##### American Put versus European Put

−

20

0

20

40

60

80

100

120

140

160

180

200

220

0

50

100

European put price

American put price

American put price is above European put price and it is possible to early exercise

## Project Files Chart

main.cpp

fstream

Market.h

TreePricer.h

EuropeanTrade.h

AmericanTrade.h

iostream

Date.h

vector

cmath

TreeProduct.h

cassert

Payoff.h

Types.h

## Extension: JRRN Binomial Model

The constraint is arbitrary, as long as first and second moment matches. Jarrow et al. proposed making

* It can be verified that the second moment matches
* Note that the original Jarrow Rudd (JR) tree enforces the probability to 0.5 and is thus not a risk neutral tree (used for credit risk where the market is not complete)

# Assignment (4 weeks)

1. Setup project: create a project of windows or OS console application, where contains all files of Date, Market, Trade, Pricer and be able to run the program without error.
2. Complete the missing implementation in Date, Market, Bond, Swap class.
3. In main.cpp, create a bond trade with maturity of 2Y with notional of 100,000, choosing any of bond name from bond price.txt, and coupon frequency is semi-annual, coupon rate is 2.5% and traded price 101.5. Then use pricer to compute the NPV of bond.
4. In main.cpp, create a swap trade with maturity of 5y with notional of 1,000,000 and fixed rate of 4.5%, coupon frequency is semi-annual. Then use pricer to compute the NPV of swap.
5. In main.cpp, create an European call option on stock (any stock provided in stock price.txt file), 6M in expiry, strike is 5% ITM. Compute the NPV of the trade using Tree model and compare with Black Scholes price.
6. In main.cpp, create an American call option on stock (any stock provided in stock price.txt file), 6M in expiry, strike is 5% ITM. Compute the NPV of the trade using Tree model.