Introduction to C++ Programming

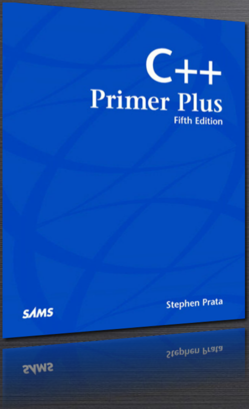
Its Applications in Finance

Thanh Hoang

Claremont Graduate University

October 17, 2012

Today Agenda



1. Standard C++ *string* Class
   - □ Define and Assign *string* Objects
   - □ Input/Output with *string* Objects
   - □ Find, Modify, Compare *string* Objects
   - □ Access *characters* in *string* Objects
   - □ Other *string* Member Functions
2. Friend Function
   - □ Define a Friend Function
   - □ Friend Functions as Bridges
3. Friend Class
4. Structure vs. Class
5. Operator Overloading
   - □ Operator Overloading: + and =
   - □ Comparison Operators
6. Summary
7. Course Summary
8. Final Exam

String Fundamentals

Definition

The most common use for one-dimensional arrays is to create a character string. There are two types of strings:

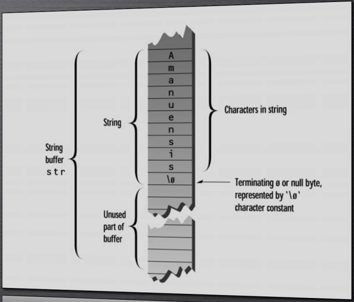
1. C-style string (*null-terminated*)
2. The String Class

General Form of C-style String

```
char mystring[size];
```

General Form of the String Class

```
#include <string>
..................
string strobj;
```



Define and Assign *string* Objects

```cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
    // Initializes strings
    string str1("Beauty"), str2 = "Beast", str3;

    // Assigns str1 to str3
    str3 = str1;
    cout << str3 << endl;

    // Concatenates str2 to str3
    str3 += " and the " + str2;
    cout << str3 << endl;

    // Swaps str1 and str2
    str1.swap(str2);
    cout << str1 << " and the " << str2 << endl;

    return 0;
}
```

Output

```
Beauty
Beauty and the Beast
Beast and the Beauty
```

Input/Output with *string* Objects

```cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
    // Creates string objects
    string fullName, nickname, address;
    string greeting("\nHello, ");

    cout << "\nEnter your full name: ";
    getline(cin, fullName);
    cout << "\nEnter your nickname: ";
    getline(cin, nickname);

    // Appends name to greeting
    greeting += nickname;
    cout << greeting << endl;

    cout << "\nEnter your address on seperate lines\n";
    cout << "Terminate with $\n\n";
    // Reads multiple lines
    getline(cin, address, "$");

    cout << "\nYour address is: " << address << endl;

    return 0;
}
```

```
// Output:
Enter your full name: Thanh Hoang

Enter your nickname: Thanh

Hello, Thanh

Enter your address on seperate lines
Terminate with $

1000 Foothill Blvd
Claremont, CA 91711$

Your address is: 1000 Foothill Blvd
Claremont, CA 91711
```

Find *string* Objects

```cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string str1("You can do it!");
    cout << str1 << endl << endl;

    unsigned long n = str1.find("can");
    cout << "We found \"can\" at " << n
    << endl << endl;

    n = str1.find_first_of("ndt");
    cout << "First of  n ,  d , or  t  at "
    << n << endl << endl;

    n = str1.find_first_not_of("aeiouAEIOU");
    cout << "First consonant at " << n
    << endl << endl;

    return 0;
}
```

```
// Output:
You can do it!

We found "can" at 4

First of  n ,  d , or  t  at 6

First consonant at 0
```

Modify *string* Objects

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string str1("Nah! You cannot do it.");
    string str2("can");
    string str3("Sure, ");

    // displays str1
    cout << str1 << endl << endl;

    // removes "Nah! "
    str1.erase(0, 5);
    // replaces  Y  with  y
    str1.replace(0, 1, "y");
    // replaces "cannot" with "can"
    str1.replace(4, 6, str2);
    // inserts "Sure, " at beginning
    str1.insert(0, str3);
    // removes  .
    str1.erase(str1.size()-1,1);
    // appends "!!!"
    str1.append(3,  ! );
```

```
    // displays a modified string
    cout << str1 << endl << endl;

    // finds a space
    unsigned long x = str1.find(   );

    while( x < str1.size() )
    {
        // replaces a space with a slash
        str1.replace(x, 1, "/");
        x = str1.find(   );
    }

    cout << str1 << endl << endl;

    return 0;
}

// Output:
Nah! You cannot do it.

Sure, you can do it!!!

Sure,/you/can/do/it!!!
```

Compare *string* Objects

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string aName = "Tom";
    string userName;

    cout << "\nEnter your first name: ";
    getline(cin, userName);

    if(userName == aName)
        cout << "Greeting, Tom\n";
    else if(userName < aName)
        cout << "You come before Tom\n";
    else
        cout << "You come after Tom\n";

    // compare() function
    int n = userName.compare(0, 2, aName, 0, 2);

    cout << "The first two letters of your name \""
      << userName.substr(0,2);
```

```
    if(n==0)
        cout << "\" match ";
    else if (n < 0)
        cout << "\" come before ";
    else
        cout << "\" come after ";

    cout << "\"" << aName.substr(0,2)
    << "\"." << endl;

    return 0;
}

// Output:
Enter your first name: Thanh
You come before Tom
The first two letters of your name
 "Th" come before "To".
```

Access Characters in *string* Objects

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    char strarr[80];
    string word;

    cout << "\nEnter a word: ";
    getline(cin, word);

    // length of string object
    unsigned long length = word.length();

    cout << "One character at a time: ";
    for(int i=0; i<length; i++)
        cout << word.at(i) << " ";
```

```
    word.copy(strarr, length, 0);

    // terminate with \0
    strarr[length] = 0;

    cout << "\nArray contains: "
   << strarr << endl;

    return 0;
}


// Output:
Enter a word: Excellent
One character at a time: E x c e l l e n t
Array contains: Excellent
```

Other *string* Member Functions

Member functions	Definition
append	Extend the current string by adding an additional appending string at its end
assign	Assign a new content to the string replacing its current content
at	Return the character at a specified position in the string
capacity	Return the size of the allocated storage space in the string object
clear	Erase any previous content
compare	Compare the content of a string object to the content of a comparing string
empty	Return whether the string is empty: *true* if the string size is zero, *false* otherwise
erase	Erase a part of the string content, and shorten the string length
find	Search the specified content of a string, and return the position
insert	Insert some additional content at a specific location in the string
length	Return a number of characters in the string
replace	Replace a section of the current string by some other content
substr	Return a substring of the current string object
swap	Swap the content of a string with the content of another string

Define a *friend* Function

```cpp
#include <iostream>
using namespace std;

class simpleClass
{
private:
    int num1, num2;

public:
    // constructor
    simpleClass( int a, int b ) : num1(a), num2(b)
    { /* empty body */ }

    // friend function
    friend void commonDenom( simpleClass x );
};

// define a friend function
void commonDenom( simpleClass x )
{
    int max;
    if (x.num1 > x.num2)
        max = x.num1;
    else
        max = x.num2;
```

```cpp
    for (int j = 2; j <= max; j++)
        if (!(x.num1 % j) && !(x.num2 % j)) {
            cout << j <<     ;
        }

    cout << endl;
    return;
}

int main()
{
    simpleClass group(6, 30);

    cout << "\nCommon denominator(s): ";
    commonDenom(group);
    cout << endl;

    return 0;
}

// Output:
Common denominator(s): 2 3 6
```

Define a *friend* Function (*cont.*)

```cpp
#include <iostream>
#include <cmath>
using namespace std;

class Distance
{
private:
    int feet;
    float inches;

public:
    // constructor (no argument)
    Distance() : feet(0), inches(0.0)
    { /* empty body */ }

    // constructor (two arguments)
    Distance(int ft, float in) : feet(ft), inches(in)
    { /* empty body */ }

    // display distance
    void showdist()
    { cout << feet << "-" << inches << " ; }

    // friend function
    friend float square(Distance);
};
```

```cpp
int main()
{
    Distance dist(4, 6.25);
    float sqft = square(dist);

    cout << "\nDistance = ";
    dist.showdist();
    cout << "\nSquare = " << sqft << " square feet\n";

    return 0;
}

// function definition
float square(Distance d)
{
    float fltfeet = d.feet + d.inches/12;
    float feetsqrd = pow(fltfeet,2);

    return feetsqrd;
}

// Output:
Distance = 4-6.25
Square = 20.4379 square feet
```

Friend Function as a Bridge

```cpp
#include <iostream>
using namespace std;

// Declares a class before using it
class beta;

class alpha
{
private:
    int data;

public:
    // alpha constructor (no argument)
    alpha() : data(3)
    { /* empty body */ }

    // friend function
    friend int sumFunc(alpha, beta);
};

class beta
{
private:
    int data;
```

```cpp
public:
    // beta constructor (no argument)
    beta() : data(6)
    { /* empty body */ }

    // friend function
    friend int sumFunc(alpha, beta);
};

int main()
{
    alpha a;
    beta b;

    cout << "\nSum = " << sumFunc(a, b) << endl;

    return 0;
}

int sumFunc(alpha a, beta b)
{ return( a.data + b.data ); }

// Output:
Sum = 9
```

Friend Class

```cpp
#include <iostream>
using namespace std;

class alpha
{
private:
    int data;

public:
    // constructor (no argument)
    alpha() : data(10)
    { /* empty body */ }

    // friend class
    friend class beta;
};
```

```cpp
class beta
{
public:
    void showFunc(alpha a)
    {
        cout << "\nData = " << a.data;
    }
};

int main()
{
    alpha a;
    beta b;

    b.showFunc(a);
    cout << endl;

    return 0;
}

// Output:
Data = 10
```

Define a simple Structure

```cpp
#include <iostream>
using namespace std;

struct simple
{

    int getnum()
    { return num; }

    void putnum(int i)
    { num = i; }

private:
    int num;
};

int main()
{
    simple a;

    a.putnum(10);
    cout << a.getnum() << endl;

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

class simple
{
    int num;

public:
    int getnum()
    { return num; }

    void putnum(int i)
    { num = i; }
};

int main()
{
    simple a;

    a.putnum(10);
    cout << a.getnum() << endl;

    return 0;
}
```

Nested Structures or Classes

```
#include <iostream>
using namespace std;

struct Distance
{
    int feet;
    float inches;
};

struct Room
{
    Distance length;
    Distance width;
};

int main()
{
    Room dining;

    dining.length.feet = 14;
    dining.length.inches = 5.5;
```

```
    dining.width.feet = 10;
    dining.width.inches = 4.5;

    float l = dining.length.feet +
    dining.length.inches/12;

    float w = dining.width.feet +
    dining.width.inches/12;

    cout << "Dining room area is "
      << l * w << " square feet." << endl;

    return 0;
}

// Output:
Dining room area is 150.005 square feet.
```

Object 3D

```cpp
#include <iostream>
using namespace std;

class Obj3D
{
private:
    // 3D coordinates
    int x, y, z;

public:
    // constructor (no argument)
    Obj3D() : x(0), y(0), z(0)
    { /* empty body */ }

    // constructor (three arguments)
    Obj3D(int a, int b, int c) : x(a), y(b), z(c)
    { /* empty body */ }

    Obj3D operator + (Obj3D obj);
    Obj3D operator = (Obj3D obj);

    void show();
};
```

```cpp
// Overload operator +
Obj3D Obj3D::operator + (Obj3D obj)
{
    Obj3D temp;

    temp.x = x + obj.x;
    temp.y = y + obj.y;
    temp.z = z + obj.z;

    return temp;
}

// Overload operator =
Obj3D Obj3D::operator = (Obj3D obj)
{
    x = obj.x;
    y = obj.y;
    z = obj.z;

    // return the modified object
    // by using a temporary Obj3D
    return Obj3D(x,y,z);
}

// Show x, y, z coordinates
void Obj3D::show()
{ cout << x << ", " << y << ", " << z << endl; }
```

Object 3D (*cont.*)

```
int main()
{
    Obj3D obj1(1, 2, 3), obj2(4, 5, 6);
    Obj3D obj3, obj4;

    cout << "\nOriginal value of object 1: ";
    obj1.show();
    cout << "\nOriginal value of object 2: ";
    obj2.show();

    // add obj1 and obj2
    obj3 = obj1 + obj2;
    cout << "\nValue of object 3 after adding object 1
                and object 2: ";
    obj3.show();

    // add obj1, obj2 and obj3
    obj4 = obj1 + obj2 + obj3;
    cout << "\nValue of object 4 after adding object 1,
                object 2, and object 3: ";
    obj4.show();

    // demonstrate multiple assignment
    obj4 = obj3 = obj2 = obj1;
    cout << "\nValue of object 3 after modified: ";
    obj3.show();
    cout << "\nValue of object 4 after modified: ";
    obj4.show();
    cout << endl;

    return 0;
}
```

```
// Output:
Original value of object 1: 1, 2, 3

Original value of object 2: 4, 5, 6

Value of object 3 after adding object 1
        and object 2: 5, 7, 9

Value of object 4 after adding object 1,
        object 2, and object 3: 10, 14, 18

Value of object 3 after modified: 1, 2, 3

Value of object 4 after modified: 1, 2, 3

Program ended with exit code: 0
```

Distance

```cpp
#include <iostream>
using namespace std;

class Distance
{
private:
    int feet;
    float inches;

public:
    // constructor (no argument)
    Distance() : feet(0), inches(0.0)
    { /* empty body */ }
    // constructor (two arguments)
    Distance(int ft, float in) : feet(ft), inches(in)
    { /* empty body */ }

    // member function
    void getdist()
    {
        cout << "\nEnter feet: ";
        cin >> feet;
        cout << "Enter inches: ";
        cin >> inches;
    }

    void showdist()
    {
        cout << feet << "'-" << inches << '"';
    }

    Distance operator + (Distance);
};
```

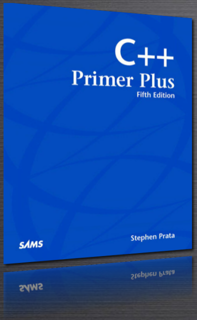
```cpp
Distance Distance::operator + (Distance obj)
{
    int f = feet + obj.feet;
    float i = inches + obj.inches;
    if(i >= 12.0)
    {
        i -= 12.0;
        f++;
    }
    return Distance(f,i);
}

Distance Distance::operator = (Distance obj)
{
    feet = obj.feet;
    inches = obj.inches;

    return Distance(feet,inches);
}

int main()
{
    Distance dist1, dist3, dist4;
    Distance dist2(11, 6.25);

    dist1.getdist();
    dist3 = dist1 + dist2;
    dist4 = dist1 + dist2 + dist3;

    cout << "\nDistance 1 = "; dist1.showdist(); cout << endl;
    cout << "\nDistance 2 = "; dist2.showdist(); cout << endl;
    cout << "\nDistance 3 = "; dist3.showdist(); cout << endl;
    cout << "\nDistance 4 = "; dist4.showdist(); cout << endl;

    return 0;
}
```

Comparison Operators

```
#include <iostream>
using namespace std;

class Distance
{
private:
    int feet;
    float inches;

public:
    // constructor (no argument)
    Distance() : feet(0), inches(0.0)
    { /* empty body */ }

    // constructor (two arguments)
    Distance(int ft, float in) : feet(ft), inches(in)
    { /* empty body */ }

    // member function
    void getdist()
    {
        cout << "\nEnter feet: ";
        cin >> feet;
        cout << "Enter inches: ";
        cin >> inches;
    }
```

```
    // display distance
    void showdist()
    { cout << feet << "'-" << inches << '"' ; }

    // compare distances
    bool operator < (Distance);
};

bool Distance::operator < (Distance obj)
{
    float fltft1 = feet + inches/12;
    float fltft2 = obj.feet + obj.inches/12;

    return (fltft1 < fltft2) ? true : false;
}

int main()
{
    Distance dist1, dist2(11, 6.25);

    dist1.getdist();
    cout << "\nDistance 1 = ";  dist1.showdist();
    cout << "\nDistance 2 = ";  dist2.showdist();

    if(dist1 < dist2)
        cout << "\nDistance 1 is less than distance 2." << endl;
    else
        cout << "\nDistance 2 is less than distance 1." << endl;

    return 0;
}
```

Summary

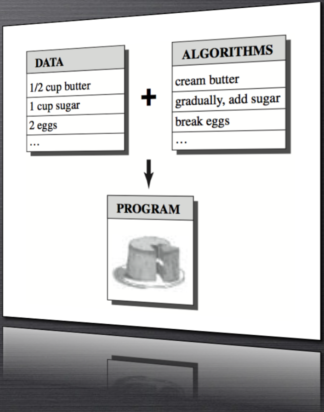


Stephen Prata.
C++ Primer Plus, 5th Edition,
Chapter 4, 11, 15
SAMS Publishing, 2004.

1. Standard C++ *string* Class
   - Define and Assign *string* Objects
   - Input/Output with *string* Objects
   - Find, Modify, Compare *string* Objects
   - Access *characters* in *string* Objects
   - Other *string* Member Functions
2. Friend function
   - Define a Friend Function
   - Friend Functions as Bridges
3. Friend Class
4. Structure vs. Class
5. Operator Overloading
   - Operator Overloading: + and =
   - Comparison Operators
6. Summary
7. Course Summary
8. Final Exam

Lecture 1 – Introduction to C++ Programming

1. A brief history of C++
2. Object-Oriented Programming
3. C++ operators
    - Arithmetic operators
    - Assignment operators
    - Relational and logical operators
4. Control statements
    - Control statement *if*
    - Control statement *for*



DATA
1/2 cup butter
1 cup sugar
2 eggs
...

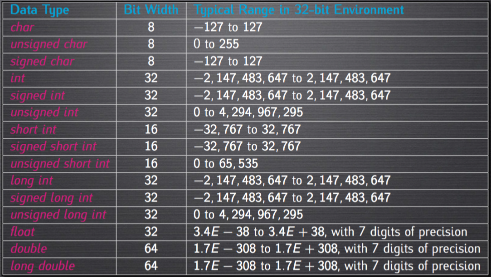
+

ALGORITHMS
cream butter
gradually, add sugar
break eggs
...

PROGRAM

Lecture 2 – Data Types and Operators

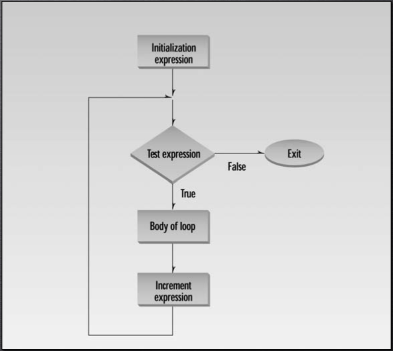
1. C++ Built-in Data Types
   - Character (*char*)
   - Integer (*int*)
   - Floating Point Types (*float*)
   - Logical and Relational Type (*bool*)
   - Escape Sequence Codes
2. C++ Variables
   - Fundamental Properties
   - Bits and Bytes
   - Hexadecimal and Octal
3. The *const* Qualifier
4. Data Type Conversion
5. Modulus Operator



Data Type	Bit Width	Typical Range in 32-bit Environment
char	8	−127 to 127
unsigned char	8	0 to 255
signed char	8	−127 to 127
int	32	−2, 147, 483, 647 to 2, 147, 483, 647
signed int	32	−2, 147, 483, 647 to 2, 147, 483, 647
unsigned int	32	0 to 4, 294, 967, 295
short int	16	−32, 767 to 32, 767
signed short int	16	−32, 767 to 32, 767
unsigned short int	16	0 to 65, 535
long int	32	−2, 147, 483, 647 to 2, 147, 483, 647
signed long int	32	−2, 147, 483, 647 to 2, 147, 483, 647
unsigned long int	32	0 to 4, 294, 967, 295
float	32	$3.4E-38$ to $3.4E+38$, with 7 digits of precision
double	64	$1.7E-308$ to $1.7E+308$, with 7 digits of precision
long double	64	$1.7E-308$ to $1.7E+308$, with 7 digits of precision

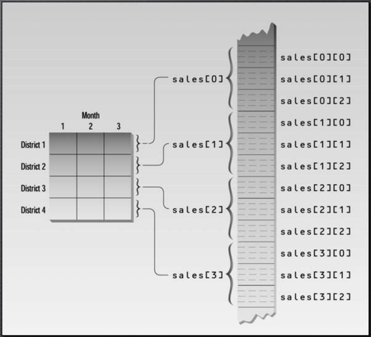
Lecture 3 - Program Control Statements

1. Random Number
2. Selection Statements
   - *if* statement
   - *switch* statement
3. Iteration Statements
   - *for* loop statement
   - *while* loop statement
   - *do-while* loop statement
4. Jump Statements
   - *break* statement
   - *continue* statement
   - *return* statement
   - *goto* statement



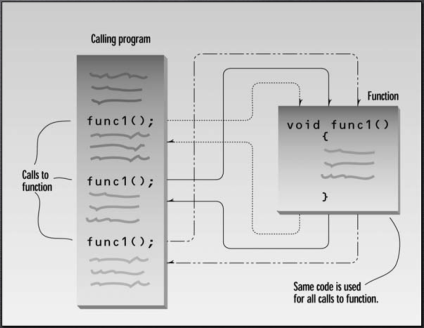
Lecture 4 - Arrays, Strings and Pointers

1. Arrays
   - One-Dimensional (*1D*) Array
   - Two-Dimensional (*2D*) Array
   - Multi-Dimensional Array
2. Strings
   - String Functions
   - Array of Strings
3. Pointers
   - Pointer Operators
   - Pointer Expression
   - Pointers and Arrays
4. File Input and Output



Lecture 5 – Functions

1. Function Fundamentals
   - General Form
   - Create a Function
   - Using Arguments and *return* Statement
2. Function Prototypes
3. Scope
   - Local Scope
   - Global Scope
4. Passing to Functions
   - Pointers in Functions
   - Arrays in Functions
   - Strings in Functions
5. Recursion
6. Inline Functions
7. Overloaded Functions

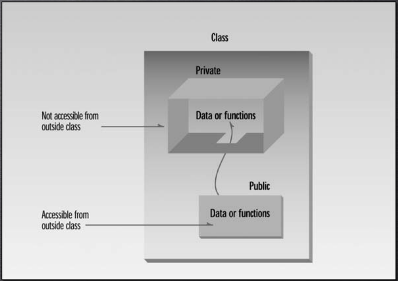


```
func1();
func1();
func1();
```

```
void func1()
{
}
```

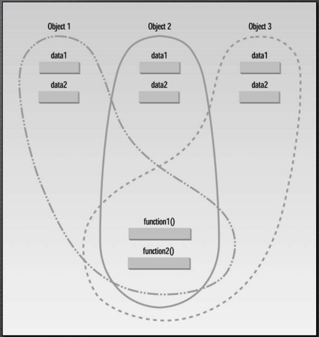
Lecture 6 – Classes and Objects

1. Classes and Objects
   - General Form of a Class
   - Member Access Control
   - Define a Class and Create Objects
   - Include Functions in a Class
     - Inside a Class
     - Outside a Class
2. Constructors and Destructors
   - Constructors
   - Destructors
   - Overloaded Constructors
3. Arrays of Objects
   - Initialize an Array of Objects
   - Pointers to Objects


Class
Private
Data or functions
Not accessible from outside class
Public
Data or functions
Accessible from outside class

Lecture 7 – Classes and Objects (*cont.*)

1. Standard C++ *string* Class
   - Define and Assign *string* Objects
   - Input/Output with *string* Objects
   - Find, Modify, Compare *string* Objects
   - Access *characters* in *string* Objects
   - Other *string* Member Functions
2. Friend Function
   - Define a Friend Function
   - Friend Functions as Bridges
3. Friend Class
4. Structure vs. Class
5. Operator Overloading
   - Operator Overloading: + and =
   - Comparison Operators



Covered Topics in Exercises

General Topics

- Lecture 1: Simple exercises with *if* and *for*
- Homework 1: Fibonacci series
- Lecture 2: Data types and Type conversion
- Homework 2: Prime numbers and Angstrom numbers
- Homework 3: Questions with control statements (*if*, *for* ...)
- Lecture 4: Bubble Sort
- Lecture 4: Cholesky decomposition algorithm
- Homework 4: Approximating a square root of a number
- Lecture 5: Simulating a uniform random number with overloaded functions
- Lecture 5: Simulating π with Monte Carlo simulation
- Homework 5: Finding prime factors of a number
- Lecture 6: Collatz Problems (4 different programming styles)

Covered Topics in Exercises (*cont.*)

Corporate Finance

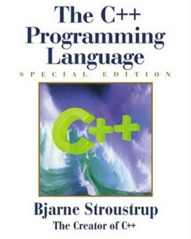
- Lecture 2: Computing Net Present Value (NPV) of projects
- Lecture 2: Comparing NPVs between two projects
- Lecture 2: Computing Profitability Index (PI)
- Lecture 2: Making investment decisions based on NPV and PI
- Lecture 3: Internal Rate of Return (IRR)
- Homework 5: Computing NPV and PI with functions

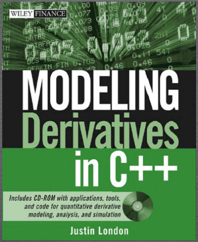
Fixed Income

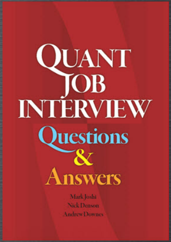
- Lecture 4: Price Bonds with discrete interest rates
- Homework 4: Price Bonds with continuous compounding interest rates
- Homework 4: Calculate Yield to Maturity of Bonds
- Homework 4: Calculate the Macaulay duration of Bonds

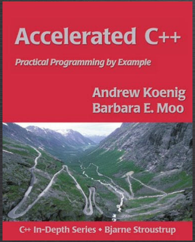
Derivatives

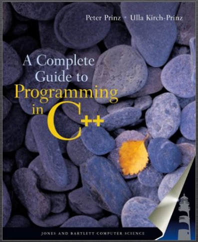
- Lecture 2: Price European Options by Black-Scholes-Merton Models
- Lecture 5: Price American Options by Binomial Approximations
- Lecture 6: Price European Options by Monte Carlo Simulation
- Lecture 7: Price Asian Options with Classes and Objects

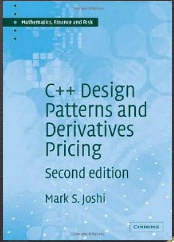

The C++ Programming Language
SPECIAL EDITION
C++
Bjarne Stroustrup
The Creator of C++


WILEY FINANCE
MODELING Derivatives in C++
Includes CD-ROM with applications, tools, and code for quantitative derivative modeling, analysis, and simulation
Justin London


QUANT JOB INTERVIEW Questions & Answers
Mark Joshi
Nick Denson
Andrew Downes


Accelerated C++
Practical Programming by Example
Andrew Koenig
Barbara E. Moo
C++ In-Depth Series • Bjarne Stroustrup


Peter Prinz • Ulla Kirch-Prinz
A Complete Guide to Programming in C++
JONES AND BARTLETT COMPUTER SCIENCE


Mathematics, Finance and Risk
C++ Design Patterns and Derivatives Pricing
Second edition
Mark S. Joshi
CAMBRIDGE

Asian Option

Definition

An Asian option is one whose payoff includes a time average of the underlying asset price. The average may be over the entire time period between initiation and expiration or may be over some period of time that begins later than the initiation of the option and ends with the option's expiration.

There are two ways to compute the average:

1. Continuous:

$$S_{\text{avg}} = \frac{1}{T}\int_0^T S(t)dt$$

2. Discrete:

$$S_{\text{avg}} = \frac{1}{m}\sum_{j=1}^{m} S(t_j)$$

The payoffs of an Asian option are:

1. Call option: $\max(0, S_{\text{avg}} - K)$
2. Put option: $\max(0, K - S_{\text{avg}})$

Asian Option – Geometric Average

Assume that the underlying asset price, S, is lognormally distributed and S_{avg} is a geometric average of the S's, which is also lognormally distributed. Consider a newly issued option that will provide a payoff at time T based on S_{avg} calculated between time zero and time T.

Under the risk-neutral assumption, the geometric average price option can be treated like a regular option with the volatility σ_{adj} set equal to $\sigma\sqrt{3}$ and the adjusted dividend yield equal to:

$$q_{\text{adj}} = r - \frac{1}{2}\left(r - q - \frac{\sigma^2}{6}\right) = \frac{1}{2}\left(r + q + \frac{\sigma^2}{6}\right)$$

We can use the Black-Scholes-Merton formula to price geometric average price calls and puts where we need to substitute:

$$\begin{aligned} \sigma_{\text{adj}} &= \sigma\sqrt{3} \\ q_{\text{adj}} &= \frac{1}{2}\left(r + q + \frac{\sigma^2}{6}\right) \end{aligned}$$

Asian Option – Geometric Average (*cont.*)

By using the Black-Scholes-Merton formula, we price geometric average price Asian options:

$$\begin{aligned} c &= S(0)e^{-q_{\text{adj}}T}N(d1) - Ke^{-rT}N(d2) \\ p &= Ke^{-rT}N(-d2) - S(0)e^{-q_{\text{adj}}T}N(-d1) \end{aligned}$$

where:

$$\begin{aligned} \sigma_{\text{adj}} &= \sigma\sqrt{3} \\ q_{\text{adj}} &= \frac{1}{2}\left(r + q + \frac{\sigma^2}{6}\right) \\ d_1 &= \frac{1}{\sigma_{\text{adj}}\sqrt{T}}\left(\log\left(\frac{S(0)}{K}\right) + \left(r - q_{\text{adj}} + \frac{1}{2}\sigma_{\text{adj}}^2\right)T\right) \\ d_2 &= d_1 - \sigma_{\text{adj}}\sqrt{T} \end{aligned}$$

Asian Option – Arithmetic Average

An arithmetic average of a set of lognormal random variables is not itself lognormal. This distribution is complicated, expressed by the German–Yor formulas. Luckily, we can approximate this complicated distribution by a lognormal distribution by using the Turnbull–Wakeman approximation.

The first moment of the continuous arithmetic average price distribution in $[t, T]$

$$M_1 = \frac{e^{(r-q)T} - 1}{(r-q)T} S(0)$$

The second moment of the continuous arithmetic average:

$$M_2 = \frac{2e^{(2(r-q)+\sigma^2)T} S^2(0)}{(r-q+\sigma^2)(2r-2q+\sigma^2)T^2} + \frac{2S^2(0)}{(r-q)T^2}\left(\frac{1}{2(r-q)+\sigma^2} - \frac{e^{(r-q)T}}{r-q+\sigma^2}\right)$$

If we assume the the average asset price is lognormal, the arithmetic average price option can be treated like an option on a future contract, where:

$$\begin{aligned} F_0 &= M_1 \\ \sigma^2_{\text{adj}} &= \frac{1}{T}\log\left(\frac{M_2}{M_1^2}\right) \end{aligned}$$

Asian Option – Arithmetic Average (*cont.*)

By using the Black-Scholes-Merton formula, we price arithmetic average price Asian options:

$$\begin{aligned} c &= e^{-rT}\left(F_0 N(d_1) - KN(d_2)\right) \\ p &= e^{-rT}\left(KN(-d_2) - F_0 N(-d_1)\right) \end{aligned}$$

where:

$$\begin{aligned} F_0 &= M_1 \\ \sigma^2_{\mathrm{adj}} &= \frac{1}{T}\log\left(\frac{M_2}{M_1^2}\right) \\ d_1 &= \frac{1}{\sigma_{\mathrm{adj}}\sqrt{T}}\left(\log\left(\frac{F_0}{K}\right) + \frac{1}{2}\sigma^2_{adj}T\right) \\ d_2 &= d_1 - \sigma_{\mathrm{adj}}\sqrt{T} \end{aligned}$$