

Introduction to C++ Programming

Its Applications in Finance



Thanh Hoang

Claremont Graduate University

October 10, 2012

Today Agenda



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
4. Summary
5. Exercises



General Form of a Class

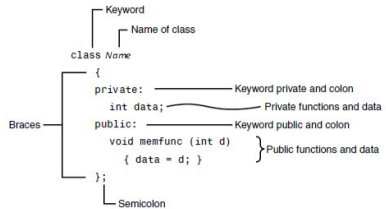
Definition

A class is a template that defines the form of an object. A class specifies:

1. Code
2. Data

General Form

```
class Name {  
    private: // optional  
        // data member declarations  
    public:  
        // member function prototypes  
};
```



Semicolon

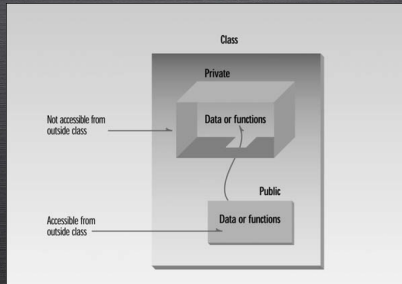


Member Access Control: Private or Public

Definition

We can declare class members, including data items and member functions, either in:

1. **private:** One of the main precepts of OOP is to hide data items into the private section.
2. **public:** The member functions that constitute the class interface go into the public section; otherwise, we cannot call those functions from a program.



A Simple Class

```
1 #include <iostream>
   using namespace std;
3
   // Define a class
5 class simpleClass
   {
7   private:
       // Class data
9       int somedata;
   public:
       // Set data as a member function
11      void setdata(int d) {
13          somedata = d;
15      }
       // Display data
17      void showdata() {
19          cout << "Data is " << somedata << endl;
       }
   };
```

```
int main()
2 {
   // Define two objects
4   simpleClass obj1, obj2;
   // Call member functions
6   obj1.setdata(99);
8   obj2.setdata(100);
   obj1.showdata();
10  obj2.showdata();
12
   return 0;
14 }
```



Define a Simple Class and Create Objects

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

4 class Car { // Declare a Car class
5 {
6     public:
7         double acceleration; // 0-60 MPH acceleration in seconds
8         double fuelcap; // Fuel-tank capacity in gallons
9         double mpg; // Fuel consumption in miles per gallon
10 };

12 int main()
13 {
14     // Create a Car object
15     Car Lexus_RX450h_AWD;
16     double range;

17     // Assign values in Lexus RX450h AWD
18     Lexus_RX450h_AWD.acceleration = 7.4;
19     Lexus_RX450h_AWD.fuelcap = 17.2;
20     Lexus_RX450h_AWD.mpg = 29.0;

21     // Compute the range by assuming a full tank of gas
22     range = Lexus_RX450h_AWD.fuelcap * Lexus_RX450h_AWD.mpg;

23     cout << "Lexus RX450h AWD has " << Lexus_RX450h_AWD.acceleration
24           << " seconds in the 0-60 MPH acceleration performance.\n\n";
25     cout << "With a full tank of gas, " << "Lexus RX450h AWD can run approximately "
26           << range << " miles.\n\n";

27     return 0;
28 }
```



Define a Simple Class and Create Objects (cont.)

```
1  int main()
2  {
3      // Create two Car objects
4      Car Lexus_RX450h_AWD, Ferrari_CA;
5      double rangeLX, rangeFR;
6
7      // Assign values to Lexus RX450h AWD
8      Lexus_RX450h_AWD.acceleration = 7.4;
9      Lexus_RX450h_AWD.fuelcap = 17.2;
10     Lexus_RX450h_AWD.mpg = 29.0;
11
12     // Assign values to Ferrari CA
13     Ferrari_CA.acceleration = 3.8;
14     Ferrari_CA.fuelcap = 20.6;
15     Ferrari_CA.mpg = 17.96;
16
17     // Compute the range assuming a full tank of gas
18     rangeLX = Lexus_RX450h_AWD.fuelcap * Lexus_RX450h_AWD.mpg;
19     rangeFR = Ferrari_CA.fuelcap * Ferrari_CA.mpg;
20
21     cout << "Ferrari California's 0-60 MPH acceleration is greater than "
22     << "Lexus RX450h AWD's approximately "
23     << Lexus_RX450h_AWD.acceleration - Ferrari_CA.acceleration
24     << " seconds.\n\n";
25
26     cout << "However, with a full tank of gas, Lexus RX450h AWD can "
27     << "run further than Ferrari California about "
28     << rangeLX - rangeFR << " miles.\n\n";
29
30     return 0;
31 }
```



Define a Simple Class and Create Objects (cont.)



Member Function Defined Inside a Class

```
1 #include <iostream>
2 using namespace std;
3
4 // Declare a Car Class
5 class Car
6 {
7 public:
8     // 0-60 MPH acceleration in seconds
9     double acceleration;
10    // Fuel-tank capacity in gallons
11    double fuelcap;
12    // Fuel consumption in miles per gallon
13    double mpg;
14    // Member function
15    double range() {
16        return fuelcap * mpg;
17    }
18 };
```

```
19 int main()
20 {
21     // Create two Car objects
22     Car Lexus_RX450h_AWD, Ferrari_CA;
23     double rangeLX, rangeFR;
24
25     // Assign values to Lexus_RX450h_AWD
26     Lexus_RX450h_AWD.acceleration = 7.4;
27     Lexus_RX450h_AWD.fuelcap = 17.4;
28     Lexus_RX450h_AWD.mpg = 29.0;
29
30     // Assign values to Ferrari_CA
31     Ferrari_CA.acceleration = 3.8;
32     Ferrari_CA.fuelcap = 20.6;
33     Ferrari_CA.mpg = 17.96;
34
35     // Compute the range assuming a full tank of gas
36     rangeLX = Lexus_RX450h_AWD.range();
37     rangeFR = Ferrari_CA.range();
38
39     cout << "Ferrari California's 0-60 MPH acceleration is greater than "
40          << "Lexus RX450h AWD's approximately "
41          << Lexus_RX450h_AWD.acceleration - Ferrari_CA.acceleration
42          << " seconds.\n\n";
43
44     cout << "However, with a full tank of gas, Lexus RX450h AWD can "
45          << "run further than Ferrari California about "
46          << rangeLX - rangeFR << " miles.\n\n";
47
48     return 0;
49 }
```



Member Function Defined Outside a Class

```
1 #include <iostream>
2 using namespace std;
3
4 // Declare a Car Class
5 class Car
6 {
7     public:
8         // 0-60 MPH acceleration in seconds
9         double acceleration;
10        // Fuel-tank capacity in gallons
11        double fuelcap;
12        // Fuel consumption in miles per gallon
13        double mpg;
14        // Member function
15        double range();
16 };
17
18 int main()
19 {
20     .....
21     .....
22     return 0;
23 }
24
25 // Member function
26 double Car::range() {
27     return fuelcap * mpg;
28 }
```

```
1 int main()
2 {
3     // Create two Car objects
4     Car Lexus_RX450h_AWD, Ferrari_CA;
5     double rangeLX, rangeFR;
6
7     // Assign values to Lexus_RX450h_AWD
8     Lexus_RX450h_AWD.acceleration = 7.4;
9     Lexus_RX450h_AWD.fuelcap = 17.2;
10    Lexus_RX450h_AWD.mpg = 29.0;
11
12    // Assign values to Ferrari_CA
13    Ferrari_CA.acceleration = 3.8;
14    Ferrari_CA.fuelcap = 20.6;
15    Ferrari_CA.mpg = 17.96;
16
17    // Compute the range assuming a full tank of gas
18    rangeLX = Lexus_RX450h_AWD.range();
19    rangeFR = Ferrari_CA.range();
20
21    cout << "Ferrari California's 0-60 MPH "
22         << "acceleration is greater than "
23         << "Lexus RX450h_AWD's approximately "
24         << Lexus_RX450h_AWD.acceleration - Ferrari_CA.acceleration
25         << " seconds.\n\n";
26
27    cout << "However, with a full tank of gas, "
28         << "Lexus RX450h_AWD can "
29         << "run further than Ferrari California about "
30         << rangeLX - rangeFR << " miles.\n\n";
31
32    return 0;
33 }
```



Constructors and Destructors

Constructors

A constructor is a special member function that automatically initializes an object when it is created.

Some properties of a constructor are as follow:

1. Same name as its class.
2. Syntactically similar to a function.
3. No explicit return type.

Destructors

With a constructor, its complement is the destructor.

Some properties of a destructor are as follow:

1. Same name as its constructor, preceded by a tilde.
2. Syntactically similar to a function.
3. No explicit return type.
4. Do not have parameters.

Class Declaration

```
class Name
{
    // Private data and functions
public:
    // Declares constructors and destructor
    Name()
    { /* body of code */ }
    ~Name()
    { /* body of code */ }
    // public data and functions
};
```



An Example of Constructors

```
1  #include <iostream>
   using namespace std;
3
   class Counter
5  {
   private:
7      // Private data
       unsigned count;
9
   public:
11     // Constructor
       Counter() {
13         count = 0;
       }
15
       // Increase count
17     void increase_count() {
19         count++;
       }
21
       // Return count
       int get_count() {
23         return count;
       }
25 };
```

```
1  int main()
   {
3      // Create two objects
       Counter obj1, obj2;
5
       // Display
7      cout << obj1.get_count() << endl;
       cout << obj2.get_count() << endl;
9
       // Increment obj1
11     obj1.increase_count();
13
       // Increment obj2
15     obj2.increase_count();
       obj2.increase_count();
17
       // Display again
19     cout << obj1.get_count() << endl;
       cout << obj2.get_count() << endl;
21
       return 0;
   }
```



An Example of Constructors (*cont.*)

```
1  #include <iostream>
2  using namespace std;
3
4  class Counter
5  {
6      private:
7          unsigned count;
8
9      public:
10         // Constructor
11         Counter() : count(0)
12         { /* Empty body */ }
13
14         // Increase count
15         void increase_count() {
16             count++;
17         }
18
19         // Return count
20         int get_count() {
21             return count;
22         }
23 };
```

```
1  int main()
2  {
3      // Create two objects
4      Counter obj1, obj2;
5
6      // Display
7      cout << obj1.get_count() << endl;
8      cout << obj2.get_count() << endl;
9
10     // Increase obj1
11     obj1.increase_count();
12
13     // Increase obj2
14     obj2.increase_count();
15     obj2.increase_count();
16
17     // Display again
18     cout << obj1.get_count() << endl;
19     cout << obj2.get_count() << endl;
20
21     return 0;
22 }
```



An Example of Destructors

```
1 #include <iostream>
2 using namespace std;
3
4 // Define a simple Class
5 class simpleClass
6 {
7 public:
8     int x;
9
10    // Declare constructor and destructor
11    simpleClass(int i): x(i)
12    { /* Empty body */ }
13
14    ~simpleClass() {
15        cout << "Destructing object whose "
16              << "value is " << x << endl;
17    }
18 };
```

```
1 int main()
2 {
3     // Passing arguments to the
4     // simpleClass constructor
5     simpleClass obj1(5), obj2(10);
6
7     cout << obj1.x << " " << obj2.x << endl;
8
9     return 0;
10 }
```

```
5 10
2 Destructing object whose value is 10
Destructing object whose value is 5
```



Add a Constructor to the Car Class

```
1 #include <iostream>
2 using namespace std;
3
4 // Define a Car class
5 class Car
6 {
7     public:
8         // 0-60 MPH acceleration in seconds
9         double acceleration;
10        // Fuel-tank capacity in gallons
11        double fuelcap;
12        // Fuel consumption in miles per gallon
13        double mpg;
14
15        Car(double a, double f, double m) : ←
16            acceleration(a), fuelcap(f), mpg(m)
17        { /* Empty body */ }
18
19        // Declare a member function
20        double range() {
21            return mpg * fuelcap;
22        }
23 };
```

```
1 int main()
2 {
3     // Passing values to Car() constructor
4     Car Lexus_RX450h_AWD(7.4, 17.2, 29.0);
5     Car Ferrari_CA(3.8, 20.6, 17.96);
6
7     double rangeLX, rangeFR;
8
9     // Compute the ranges assuming a full tank of gas
10    rangeLX = Lexus_RX450h_AWD.range();
11    rangeFR = Ferrari_CA.range();
12
13    cout << "Ferrari California's 0-60 MPH acceleration "
14          << "is greater than Lexus RX450h AWD's approximately "
15          << Lexus_RX450h_AWD.acceleration - Ferrari_CA.acceleration
16          << " seconds.\n\n";
17
18    cout << "However, with a full tank of gas, "
19          << "Lexus RX450h AWD can "
20          << "run further than Ferrari California about "
21          << rangeLX - rangeFR << " miles.\n\n";
22
23    return 0;
24 }
```



Overloaded Constructors

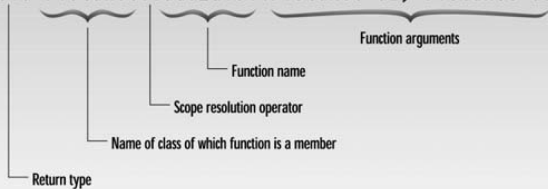
```
1 #include <iostream>
2 using namespace std;
3
4 class Distance // Define a Distance Class
5 {
6     private:
7         int feet;
8         float inches;
9     public:
10        // Constructor (no argument)
11        Distance() : feet(0), inches(0.0)
12        { /* Empty body */ }
13
14        // Constructor (two arguments)
15        Distance(int ft, float in) : feet(ft), inches(in)
16        { /* Empty body */ }
17
18        // Obtain length from the user
19        void getdist();
20        // Display distance
21        void showdist();
22        // Declare function prototype
23        void add_dist(Distance, Distance);
24 };
25
26 int main()
27 {
28     Distance dist1, dist3;
29     Distance dist2(11, 6.25);
30
31     dist1.getdist();
32     dist3.add_dist(dist1, dist2);
```

```
1 // Display all lengths
2 cout << "Distance 1 = ";
3 dist1.showdist();
4 cout << "Distance 2 = ";
5 dist2.showdist();
6 cout << "Distance 3 = ";
7 dist3.showdist();
8 cout << endl;
9
10 return 0;
11 }
12
13 // Obtain length from the user
14 void Distance::getdist(){
15     cout << "Enter feet: ";
16     cin >> feet;
17     cout << "Enter inches: ";
18     cin >> inches;
19 }
20
21 // Display distance
22 void Distance::showdist(){
23     cout << feet << " - " << inches << " " << endl;
24 }
25
26 // Add lengths - d1 and d2
27 void Distance::add_dist(Distance d1, Distance d2){
28     // Add the inches
29     inches = d1.inches + d2.inches;
30     feet = 0;
31     if (inches >= 12.0)
32     {
33         inches -= 12.0;
34         feet++;
35     }
36     feet += d1.feet + d2.feet;
37 }
```



Scope Resolution Operator

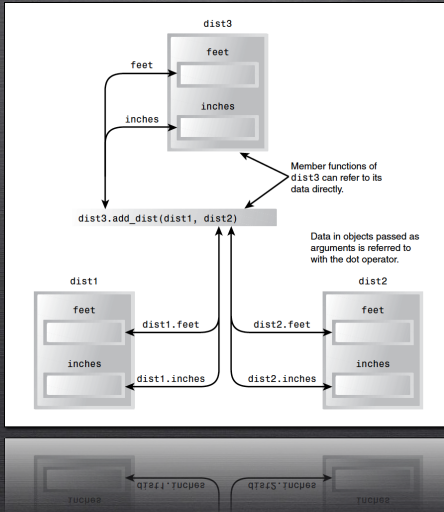
```
void Distance::add_dist(Distance d2, Distance d3)
```



Return type



Overloaded Constructors



Return Objects from Functions

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

3
4 // Define a Distance Class
5 class Distance
6 {
7 private:
8     int feet;
9     float inches;
10 public:
11     // Constructor (no arguments)
12     Distance() : feet(0), inches(0.0)
13     { /* Empty body */ }

14     // Constructor (two arguments)
15     Distance(int ft, float in) : feet(ft), inches(in)
16     { /* Empty body */ }

17     // Obtain length from the user
18     void getdist();
19     // Display distance
20     void showdist();
21     // Declare function prototype
22     Distance add_dist(Distance);
23 };

24
25 int main()
26 {
27     Distance dist1, dist3;
28     Distance dist2(11, 6.25);

29     dist1.getdist();
30     dist3 = dist1.add_dist(dist2);
```

```
1     // Display all lengths
2     cout << "Distance 1 = ";
3     dist1.showdist();
4     cout << "Distance 2 = ";
5     dist2.showdist();
6     cout << "Distance 3 = ";
7     dist3.showdist();
8     cout << endl;

9     return 0;
10 }

11 // Obtain length from the user
12 void Distance::getdist() {
13     cout << "Enter feet: ";
14     cin >> feet;
15     cout << "Enter inches: ";
16     cin >> inches;
17 }

18 // Display distance
19 void Distance::showdist() {
20     cout << feet << " - " << inches << " " << endl;
21 }

22 // Add this distance to d2, return the sum
23 Distance Distance::add_dist(Distance d2) {
24     // Creates a temporary object
25     Distance temp;
26     // Adds the inches
27     temp.inches = inches + d2.inches;
28     if (temp.inches >= 12.0)
29     {
30         temp.inches -= 12.0;
31         temp.feet += 1;
32     }
33     temp.feet += feet + d2.feet;
34     return temp;
35 }
```



Arrays of Objects

```
1 #include <iostream>
2 using namespace std;
3
4 // Define a Simple Class
5 class simpleClass
6 {
7     private:
8         int x;
9
10    public:
11        void set_x(int i) {
12            x = i;
13        }
14
15        int get_x() {
16            return x;
17        }
18 };
```

```
1 int main()
2 {
3     // Create an array of objects
4     simpleClass arr[5];
5
6     for(int j=0; j<5; j++) arr[j].set_x(j);
7
8     for(int j=0; j<5; j++)
9         cout << "arr[" << j << "].get_x(): "
10             << arr[j].get_x() << endl;
11
12     return 0;
13 }
```

```
1 arr[0].get_x(): 0
2 arr[1].get_x(): 1
3 arr[2].get_x(): 2
4 arr[3].get_x(): 3
5 arr[4].get_x(): 4
```



One-Dimensional Array of Objects

```
1 #include <iostream>
   using namespace std;
3
   class simpleClass
   {
7       private:
           int x;
9
       public:
           simpleClass(int i) : x(i)
11              { /* Empty body */ }
13
           int get_x() {
15               return x;
           };
};
```

```
int main()
{
   // Initialize an array of objects
4   simpleClass arr[5] = {-4, -3, -2, -1, 0};
6
   for (int j=0; j<5; j++)
       cout << "arr[" << j << "] get_x() : "
8           << arr[j].get_x() << endl;
10
   return 0;
}
```

```
1 arr[0].get_x() : -4
   arr[1].get_x() : -3
3   arr[2].get_x() : -2
   arr[3].get_x() : -1
5   arr[4].get_x() : 0
```



Two-Dimensional Array of Objects

```
1 #include <iostream>
2 using namespace std;
3
4 class simpleClass
5 {
6 private:
7     int x;
8 public:
9     simpleClass(int i) : x(i)
10     { /* Empty body */ }
11
12     int get_x()
13     { return x; }
14 };
```

```
1 int main()
2 {
3     simpleClass arr[4][2] = {
4         1, 2,
5         3, 4,
6         5, 6,
7         7, 8
8     };
9
10    for(int i=0; i<4; i++) {
11        for(int j=0; j<2; j++) {
12            cout << arr[i][j].get_x() << " ";
13        }
14        cout << endl;
15    }
16
17    return 0;
18 }
```



A Simple Example Using an Object Pointer

```
1 #include <iostream>
2 using namespace std;
3
4 class simpleClass
5 {
6     private:
7         int x;
8     public:
9         void getnum(int i) {
10             x = i;
11             return; // optional
12         }
13
14         void shownum() {
15             cout << x << endl;
16             return; // optional
17         }
18 };
```

```
1 int main()
2 {
3     // Declare an object and a pointer
4     simpleClass obj, *objptr;
5
6     // Call member functions directly
7     obj.getnum(10);
8     obj.shownum();
9
10    // objptr points to obj by assigning
11    // objptr the address of obj
12    objptr = &obj;
13
14    // Call member functions via a pointer to obj
15    // Access member function by using an arrow operator
16    objptr -> shownum();
17
18    objptr -> getnum(20);
19    obj.shownum();
20
21    return 0;
22 }
```



Increment and Decrement an Object Pointer

```
1 #include <iostream>
2 using namespace std;
3
4 class simpleClass
5 {
6     private:
7         int x;
8     public:
9         void getnum(int i) {
10             x = i;
11             return;
12         }
13
14         void shownum() {
15             cout << x << endl;
16             return;
17         }
18 };
```

```
1 int main()
2 {
3     // Declare two objects and a pointer
4     simpleClass obj[2], *objptr;
5
6     // Call member functions directly on obj
7     obj[0].getnum(10);
8     obj[1].getnum(20);
9
10    // objptr points to the first element
11    objptr = &obj[0];
12    // Show value of obj[0] via a pointer
13    objptr -> shownum();
14    // Move forward to next object
15    objptr++;
16    // Show value of obj[1] via a pointer
17    objptr -> shownum();
18    // Moves backward to previous object
19    objptr--;
20    // Show value of obj[0] via a pointer
21    objptr -> shownum();
22
23    return 0;
24 }
```



Summary

1. Classes and Objects

- General Form of a Class
- Member Access Control
- Define a Class and Create Objects
- Include Functions in a Class
 - Inside the Class
 - Outside the Class

2. Constructors and Destructors

- Constructors
- Destructors
- Overloaded Constructors

3. Arrays of Objects

- Initialize an Array of Objects
- Pointers to Objects

Reading



Stephen Prata.

C++ Primer Plus, 5th Edition,
Chapter 10

SAMS Publishing, 2004.



Collatz Problem

In 1937, Collatz posed the problem:

Take any integer number as an input. If the number is even, divide it by 2; otherwise, if the number is odd, multiply by 3 and add 1. The conjecture is that, no matter which number taken as an input, the end result is 1.

Algorithm

1. Inputs an integer n
2. Terminates **if** $n = 1$
3. Checks **if** n is even, then $n /= 2$; else $n = n * 3 + 1$;
4. Next number: Go to Step 2

Different Programming Styles

1. No Structured Programming
2. Structured Programming
3. Recursive Programming
4. Object-Oriented Programming



Monte Carlo Simulation

Let $W(t)$, $0 \leq t \leq T$, be a Brownian motion on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, and let $\mathcal{F}(t)$, $0 \leq t \leq T$, be a filtration for this Brownian motion. We consider a stock price process whose differential is:

$$\frac{dS(t)}{S(t)} = \alpha(t)dt + \sigma(t)dW(t)$$

Note: Both the mean rate of return $\alpha(t)$ and the volatility $\sigma(t)$ are allowed to be adapted processes.

Under the risk-neutral measure, the formula becomes:

$$\frac{dS(t)}{S(t)} = r(t)dt + \sigma(t)d\widetilde{W}(t)$$

In the BSM model, we assume a constant volatility σ , a constant interest rate r . Therefore, the price of a European call option, with a time to maturity T is equal to:

$$\begin{aligned} c(t, S(t)) &= \mathbb{E} \left[e^{-r(T-t)} (S(T) - K)^+ | \mathcal{F}(t) \right] \\ S(t) &= S(0) \exp \left(\left(r - \frac{1}{2} \sigma^2 \right) t + \sigma \widetilde{W}(t) \right) \end{aligned}$$



Monte Carlo Simulation (*cont.*)

Since $\widetilde{W}(t)$ is a Brownian motion, $\widetilde{W}(T)$ is distributed as a Gaussian with mean zero and variance T . Thus, with x is a random standard normal variable, we can write:

$$\widetilde{W}(T) = x\sqrt{T}$$

Therefore, we obtain:

$$S(T) = S(0) \exp \left(\left(r - \frac{1}{2} \sigma^2 \right) T + \sigma \widetilde{W}(T) \right) = S(0) \exp \left(\left(r - \frac{1}{2} \sigma^2 \right) T + \sigma x \sqrt{T} \right)$$

The price of a European call option is equal to:

$$\begin{aligned} c(0, S(0)) &= \widetilde{\mathbb{E}} \left[e^{-rT} (S(T) - K)^+ | \mathcal{F}(0) \right] \\ &= \widetilde{\mathbb{E}} \left[e^{-rT} \left(S(0) \exp \left(\left(r - \frac{1}{2} \sigma^2 \right) T + \sigma x \sqrt{T} \right) - K \right)^+ \right] \\ &= e^{-rT} \left(\widetilde{\mathbb{E}} \left[\left(S(0) \exp \left(\left(r - \frac{1}{2} \sigma^2 \right) T + \sigma x \sqrt{T} \right) - K \right)^+ \right] \right) \end{aligned}$$



Monte Carlo Simulation (*cont.*)

Monte Carlo Simulation: Our objective is to approximate the expectation by using the law of large numbers. If Y_i are a sequence of identically distributed independent random variables, then with the probability 1, the sequence:

$$\frac{1}{n} \sum_{i=1}^n Y_i \approx \mathbb{E}(Y)$$

$$c(0, S(0)) = e^{-rT} \left[\tilde{\mathbb{E}} \left[\left(S(0) \exp \left(\left(r - \frac{1}{2} \sigma^2 \right) T + \sigma x \sqrt{T} \right) - K \right)^+ \right] \right]$$

Algorithm:

1. Generates a standard normal random variable x from a $N(0, 1)$ distribution.
2. Computes $S(0) \exp \left(\left(r - \frac{1}{2} \sigma^2 \right) T + \sigma x \sqrt{T} \right) - K$
3. Repeats the computation for a large number of iterations, and then takes the average.
4. Multiplies this average by e^{-rT} to get the price of a European call option.

