Introduction to C++ Programming

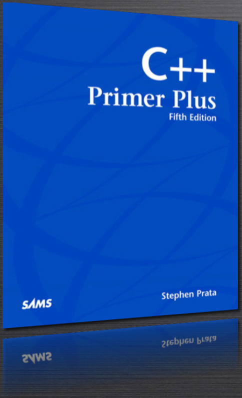
Its Applications in Finance

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

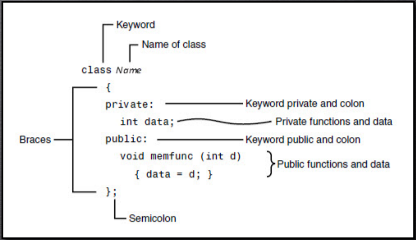
Definition

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
};
```

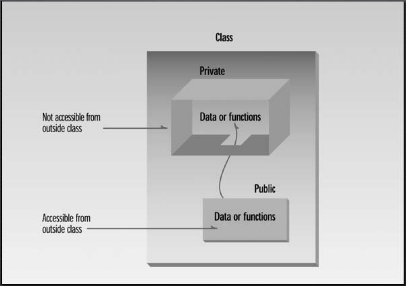


Member Access Control: Private or Public

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

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

// Define a class
class simpleClass
{
private:
    // Class data
    int somedata;
public:
    // Set data as a member function
    void setdata(int d) {
        somedata = d;
    }

    // Display data
    void showdata() {
        cout << "Data is " << somedata << endl;
    }
};
```

```cpp
int main()
{
    // Define two objects
    simpleClass obj1, obj2;

    // Call member functions
    obj1.setdata(99);
    obj2.setdata(100);

    obj1.showdata();
    obj2.showdata();

    return 0;
}
```

Define a Simple Class and Create Objects

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

class Car // Declare a Car class
{
    public:
        double acceleration; // 0-60 MPH acceleration in seconds
        double fuelcap; // fuel-tank capacity in gallons
        double mpg; // fuel consumption in miles per gallon
};

int main()
{
    // Create a Car object
    Car Lexus_RX450h_AWD;
    double range;

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

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

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

    return 0;
}
```

Define a Simple Class and Create Objects (*cont.*)

```cpp
int main()
{
    // Create two Car objects
    Car Lexus_RX450h_AWD, Ferrari_CA;
    double rangeLX, rangeFR;

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

    // Assign values in LexusRX450h
    Ferrari_CA.acceleration = 3.8;
    Ferrari_CA.fuelcap = 20.6;
    Ferrari_CA.mpg = 17.96;

    // Compute the range assuming a full tank of gas
    rangeLX = Lexus_RX450h_AWD.fuelcap * Lexus_RX450h_AWD.mpg;
    rangeFR = Ferrari_CA.fuelcap * Ferrari_CA.mpg;

    cout << "Ferrari California s 0-60 MPH acceleration is greater than "
    << " Lexus RXh400 AWD s approximately "
    << Lexus_RX450h_AWD.acceleration - Ferrari_CA.acceleration
    << " seconds.\n\n";

    cout << "However, with a full tank of gas, Lexus RX450h AWD can "
    << "run further than Ferrari California about "
    << rangeLX - rangeFR << " miles.\n\n";

    return 0;
}
```

Define a Simple Class and Create Objects (*cont.*)

Member Function Defined Inside a Class

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

// Declare a Car Class
class Car
{
public:
    // 0-60 MPH acceleration in seconds
    double acceleration;
    // Fuel-tank capacity in gallons
    double fuelcap;
    // Fuel consumption in miles per gallon
    double mpg;
    // Member function
    double range() {
        return fuelcap * mpg;
    }
};
```

```cpp
int main()
{
    // Create two Car objects
    Car Lexus_RX450h_AWD, Ferrari_CA;
    double rangeLX, rangeFR;

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

    // Assign values in LexusRX450h
    Ferrari_CA.acceleration = 3.8;
    Ferrari_CA.fuelcap = 20.6;
    Ferrari_CA.mpg = 17.96;

    // Compute the range assuming a full tank of gas
    rangeLX = Lexus_RX450h_AWD.range();
    rangeFR = Ferrari_CA.range();

    cout << "Ferrari California's 0-60 MPH accelleration is greater than "
        << " Lexus RXh400 AWD's approximately "
        << Lexus_RX450h_AWD.acceleration - Ferrari_CA.acceleration
        << " seconds.\n\n";

    cout << "However, with a full tank of gas, Lexus RX450h AWD can "
        << "run further than Ferrari California about "
        << rangeLX - rangeFR << " miles.\n\n";

    return 0;
}
```

Member Function Defined Outside a Class

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

// Declare a Car Class
class Car
{
    public:
        // 0-60 MPH acceleration in seconds
        double acceleration;
        // Fuel-tank capacity in gallons
        double fuelcap;
        // Fuel consumption in miles per gallon
        double mpg;
        // Member function
        double range();
};

int main()
{
      .....................
      .....................
      return 0;
}

// Member function
double Car::range() {
    return fuelcap * mpg;
}
```

```cpp
int main()
{
    // Create two Car objects
    Car Lexus_RX450h_AWD, Ferrari_CA;
    double rangeLX, rangeFR;

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

    // Assign values in LexusRX450h
    Ferrari_CA.acceleration = 3.8;
    Ferrari_CA.fuelcap = 20.6;
    Ferrari_CA.mpg = 17.96;

    // Compute the range assuming a full tank of gas
    rangeLX = Lexus_RX450h_AWD.range();
    rangeFR = Ferrari_CA.range();

    cout << "Ferrari California s 0-60 MPH "
         << "accelleration is greater than "
         << " Lexus RXh400 AWD s approximately "
         << Lexus_RX450h_AWD.acceleration - Ferrari_CA.acceleration
         << " seconds.\n\n";

    cout << "However, with a full tank of gas, "
         << "Lexus RX450h AWD can "
         << "run further than Ferrari California about "
         << rangeLX - rangeFR << " miles.\n\n";

    return 0;
}
```

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 tidle.
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

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

class Counter
{
private:
    // Private data
    unsigned count;

public:
    // Constructor
    Counter() {
        count = 0;
    }

    // Increase count
    void increase_count() {
        count++;
    }

    // Return count
    int get_count() {
        return count;
    }
};
```

```cpp
int main()
{
    // Create two objects
    Counter obj1, obj2;

    // Display
    cout << obj1.get_count() << endl;
    cout << obj2.get_count() << endl;

    // Increment obj1
    obj1.increase_count();

    // Increment obj2
    obj2.increase_count();
    obj2.increase_count();

    // Display again
    cout << obj1.get_count() << endl;
    cout << obj2.get_count() << endl;

    return 0;
}
```

An Example of Constructors (*cont.*)

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

class Counter
{
    private:
        unsigned count;

    public:
        // Constructor
        Counter() : count(0)
        { /* Empty body */ }

        // Increase count
        void increase_count() {
            count++;
        }

        // Return count
        int get_count() {
            return count;
        }
};
```

```cpp
int main()
{
    // Create two objects
    Counter obj1, obj2;

    // Display
    cout << obj1.get_count() << endl;
    cout << obj2.get_count() << endl;

    // Increase obj1
    obj1.increase_count();

    // Increase obj2
    obj2.increase_count();
    obj2.increase_count();

    // Display again
    cout << obj1.get_count() << endl;
    cout << obj2.get_count() << endl;

    return 0;
}
```

An Example of Destructors

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

// Define a simple Class
class simpleClass
{
public:
    int x;

    // Declare constructor and destructor
    simpleClass(int i): x(i)
    { /* Empty body */ }

    ~simpleClass() {
        cout << "Destructing object whose "
             << "value is " << x << endl;
    }
};
```

```cpp
int main()
{
    // Passing agruments to the
    // simpleClass constructor
    simpleClass obj1(5), obj2(10);

    cout << obj1.x << " " << obj2.x << endl;

    return 0;
}
```

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

Add a Constructor to the Car Class

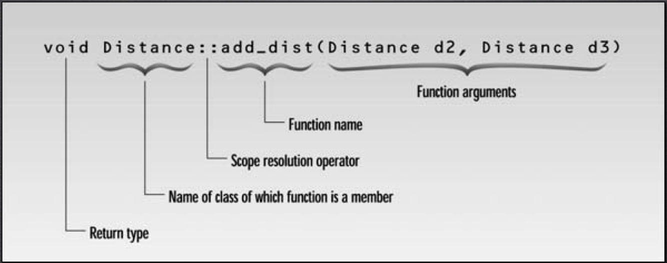
```
#include <iostream>
using namespace std;

// Define a Car class
class Car
{
    public:
        // 0-60 MPH acceleration in seconds
        double acceleration;
        // Fuel-tank capacity in gallons
        double fuelcap;
        // Fuel consumption in miles per gallon
        double mpg;

        Car(double a, double f, double m) : ↩
            acceleration(a), fuelcap(f),  mpg(m)
        { /* Empty body */ }

        // Declare a member function
        double range() {
            return mpg * fuelcap;
        }
};
```

```
int main()
{
    // Passing values to Car() constructor
    Car Lexus_RX450h_AWD(7.4, 17.2, 29.0);
    Car Ferrari_CA(3.8, 20.6, 17.96);

    double rangeLX, rangeFR;

    // Compute the ranges assuming a full tank of gas
    rangeLX = Lexus_RX450h_AWD.range();
    rangeFR = Ferrari_CA.range();

    cout << "Ferrari California s 0-60 MPH acceleration "
        << "is greater than Lexus RXh400 AWD s approximately "
        << Lexus_RX450h_AWD.acceleration - Ferrari_CA.acceleration
        << " seconds.\n\n";

    cout << "However, with a full tank of gas, "
        << "Lexus RX450h AWD can "
        << "run further than Ferrari California about "
        << rangeLX - rangeFR << " miles.\n\n";

    return 0;
}
```

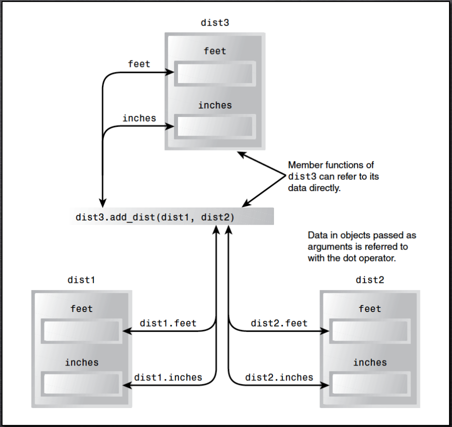
Overloaded Constructors

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

class Distance // Define a Distance Class
{
    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 */ }

        // Obtain length from the user
        void getdist();
        // Display distance
        void showdist();
        // Declare function prototype
        void add_dist(Distance, Distance);
};


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

    dist1.getdist();
    dist3.add_dist(dist1, dist2);
    // Display all lengths
    cout << "Distance 1 = ";
    dist1.showdist();
    cout << "Distance 2 = ";
    dist2.showdist();
    cout << "Distance 3 = ";
    dist3.showdist();
    cout << endl;

    return 0;
}

// Obtain length from the user
void Distance::getdist(){
    cout << "Enter feet: ";
    cin >> feet;
    cout << "Enter inches: ";
    cin >> inches;
}

// Display distance
void Distance::showdist(){
    cout << feet << "'-" << inches << '"' << endl;
}

// Add lengths: d1 and d2
void Distance::add_dist(Distance d1, Distance d2){
    // Adds the inches
    inches = d1.inches + d2.inches;
    feet = 0;
    if (inches >=12.0)
    {
        inches -= 12.0;
        feet++;
    }
    feet += d1.feet + d2.feet;
}
```

Scope Resolution Operator



Overloaded Constructors



Return Objects from Functions

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

// Define a Distance Class
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 */ }

    // Obtain length from the user
    void getdist();
    // Display distance
    void showdist();
    // Declare function prototype
    Distance add_dist(Distance);
};

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

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

```cpp
    // Display all lengths
    cout << "Distance 1 = ";
    dist1.showdist();
    cout << "Distance 2 = ";
    dist2.showdist();
    cout << "Distance 3 = ";
    dist3.showdist();
    cout << endl;

    return 0;
}

// Obtain length from the user
void Distance::getdist() {
    cout << "Enter feet: ";
    cin >> feet;
    cout << "Enter inches: ";
    cin >> inches;
}

// Display distance
void Distance::showdist() {
    cout << feet << "'-" << inches << '"' << endl;
}

// Add this distance to d2, return the sum
Distance Distance::add_dist(Distance d2) {
    // Declares a temporary object
    Distance temp;
    // Adds the inches
    temp.inches = inches + d2.inches;
    if(temp.inches >=12.0)
    {
        temp.inches -= 12.0;
        temp.feet = 1;
    }
    temp.feet += feet + d2.feet;
    return temp;
}
```

Arrays of Objects

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

// Define a Simple Class
class simpleClass
{
    private:
        int x;

    public:
        void set_x(int i) {
           x = i;
        }

        int get_x() {
           return x;
        }
};
```

```cpp
int main()
{
    // Create an array of objects
    simpleClass arr[5];

    for(int j=0; j<5; j++) arr[j].set_x(j);

    for(int j=0; j<5; j++)
        cout << "arr[" << j << "].get_x(): "
             << arr[j].get_x() << endl;

    return 0;
}
```

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

One-Dimensional Array of Objects

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

class simpleClass
{
    private:
        int x;

    public:
        simpleClass(int i) : x(i)
        { /* Empty body */ }

        int get_x() {
            return x;
        }
};
```

```cpp
int main()
{
    // Initialize an array of objects
    simpleClass arr[5] = {-4, -3, -2, -1, 0};

    for(int j=0; j<5; j++)
        cout << "arr[" << j << "].get_x(): "
             << arr[j].get_x() << endl;

    return 0;
}
```

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

Two-Dimensional Array of Objects

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

class simpleClass
{
private:
    int x;
public:
    simpleClass(int i) : x(i)
    { /* Empty body */ }

    int get_x()
    { return x; }
};
```

```cpp
int main()
{
    simpleClass arr[4][2] = {
        1, 2,
        3, 4,
        5, 6,
        7, 8
    };

    for(int i=0; i<4; i++) {
        for(int j=0; j<2; j++) {
            cout << arr[i][j].get_x() << " ";
        }
        cout << endl;
    }

    return 0;
}
```

A Simple Example Using an Object Pointer

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

class simpleClass
{
    private:
        int x;
    public:
        void getnum(int i) {
            x = i;
            return; // optional
        }

        void shownum() {
            cout << x << endl;
            return;  // optional
        }
};
```

```
int main()
{
    // Declare an object and a pointer
    simpleClass obj, *objptr;

    // Call member functions directly
    obj.getnum(10);
    obj.shownum();

    // objptr points to obj by assigning
    // objptr the address of obj
    objptr = &obj;

    // Call member functions via a pointer to obj
    // Access member function by using an arrow operator
    objptr -> shownum();

    objptr -> getnum(20);
    obj.shownum();

    return 0;
}
```

Increment and Decrement an Object Pointer

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

class simpleClass
{
    private:
        int x;
    public:
        void getnum(int i) {
            x = i;
            return;
        }

        void shownum() {
            cout << x << endl;
            return;
        }
};
```

```cpp
int main()
{
    // Declare two objects and a pointer
    simpleClass obj[2], *objptr;

    // Call member functions directly on obj
    obj[0].getnum(10);
    obj[1].getnum(20);

    // objptr points to the first element
    objptr = &obj[0];
    // Show value of obj[0] via a pointer
    objptr -> shownum();
    // Move forward to next object
    objptr++;
    // Show value of obj[1] via a pointer
    objptr -> shownum();
    // Moves backward to previous object
    objptr--;
    // Show value of obj[0] via a pointer
    objptr -> shownum();

    return 0;
}
```

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:

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

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:

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

The price of a European call option is equal to:

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

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:

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

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

Algorithm:

1. Generates a standard normal random variable x from a $N(0, 1)$ distribution.
2. Computes $S(0)\exp\left((r - \frac{1}{2}\sigma^2)T + \sigma x\sqrt{T}\right) - K$
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.