Introduction to C++ Programming

Its Applications in Finance

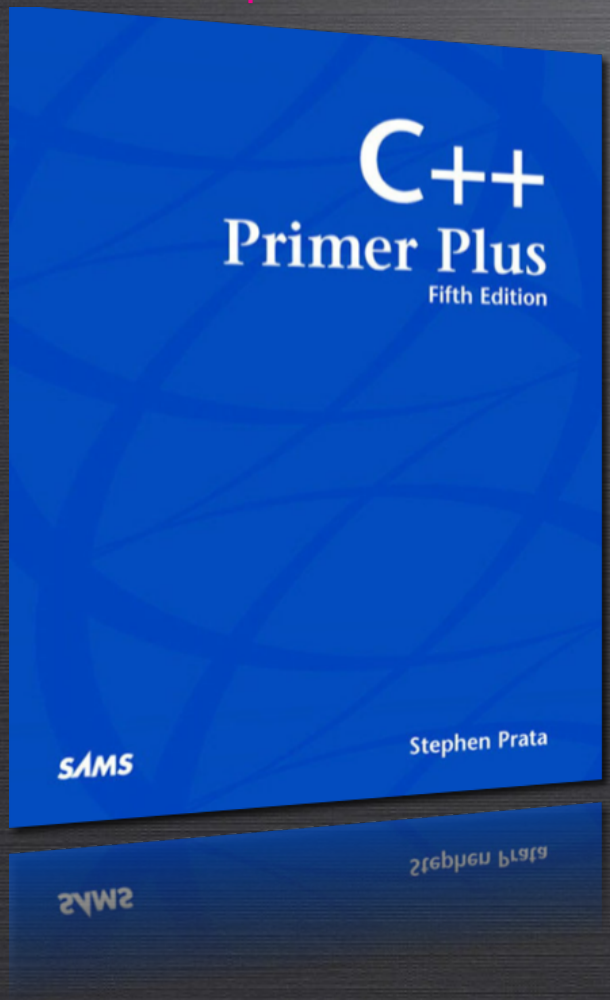
Thanh Hoang

Claremont Graduate University

September 5, 2012

Syllabus

Required Book

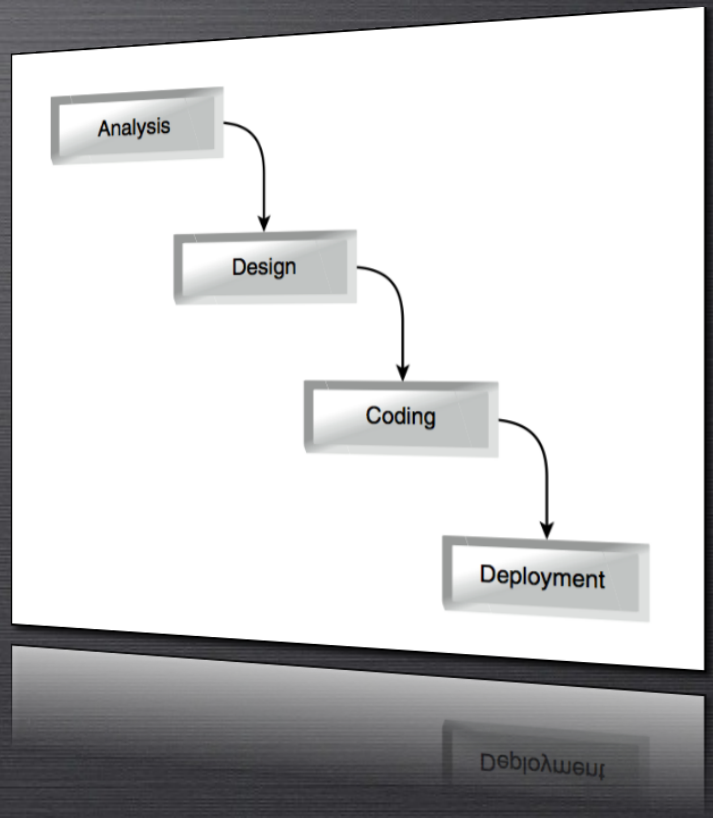


Course Meeting Time

- Time: Wednesday 1:00 - 3:50 PM
- Room: ACB 108
- Lecturer: Thanh Hoang
- E-mail: Thanh.Hoang@cgu.edu
- Office hours: Wednesday 11:00 - 12:30 PM
- Location: Computer Lab, Math House North

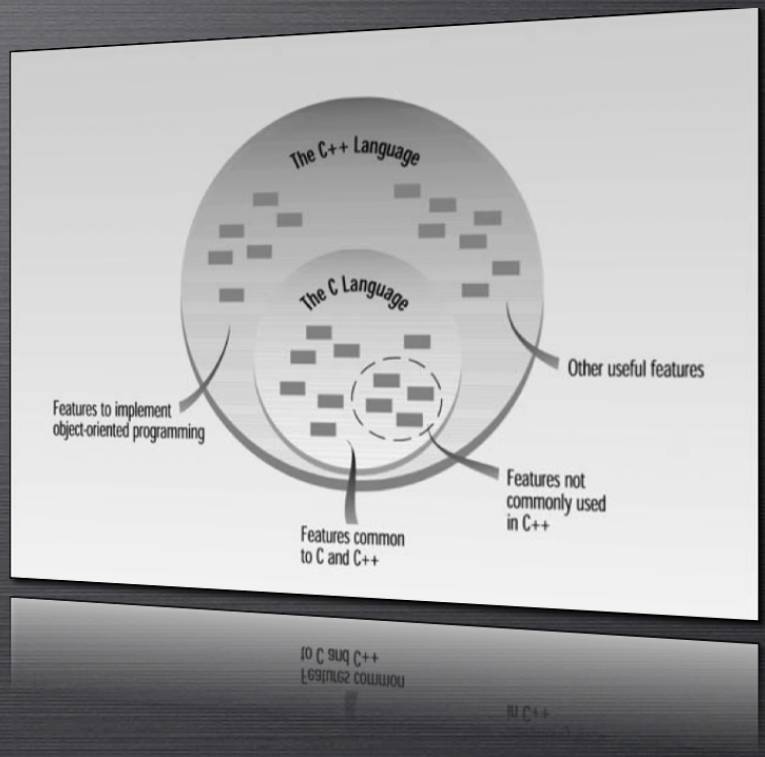
Today Agenda

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*
5. Summary



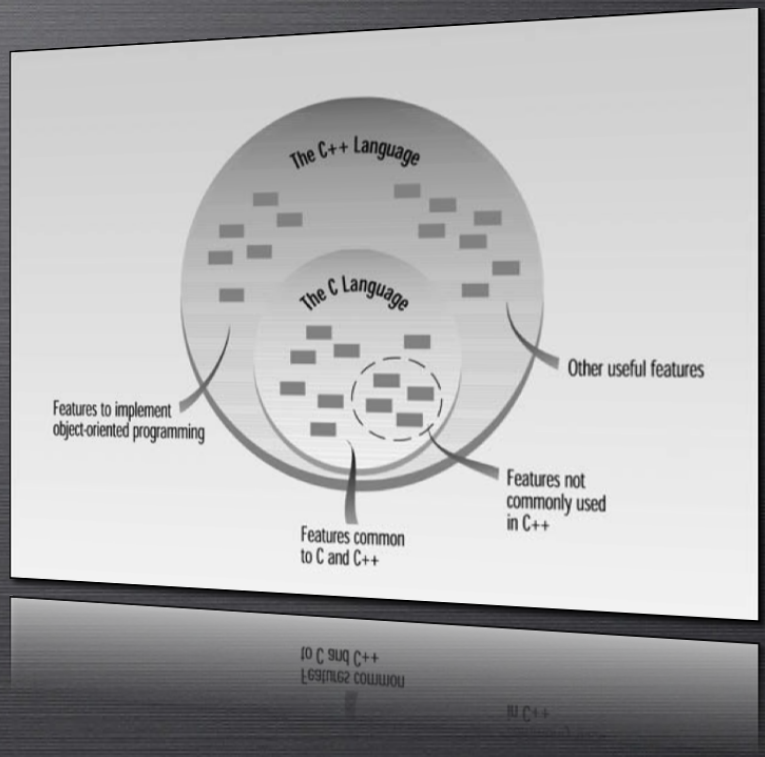
A brief history of C++

1. The foundation of C
   - C – Dennis Ritche (1970s)
   - BCPL – Martin Richards
   - B – Ken Thompson
2. The need for C++
   - Why was C++ invented?
   - Invented by Stroustrup in 1979.
3. The evolution of C++
4. Relationship of C++ with
   - Java
   - C#



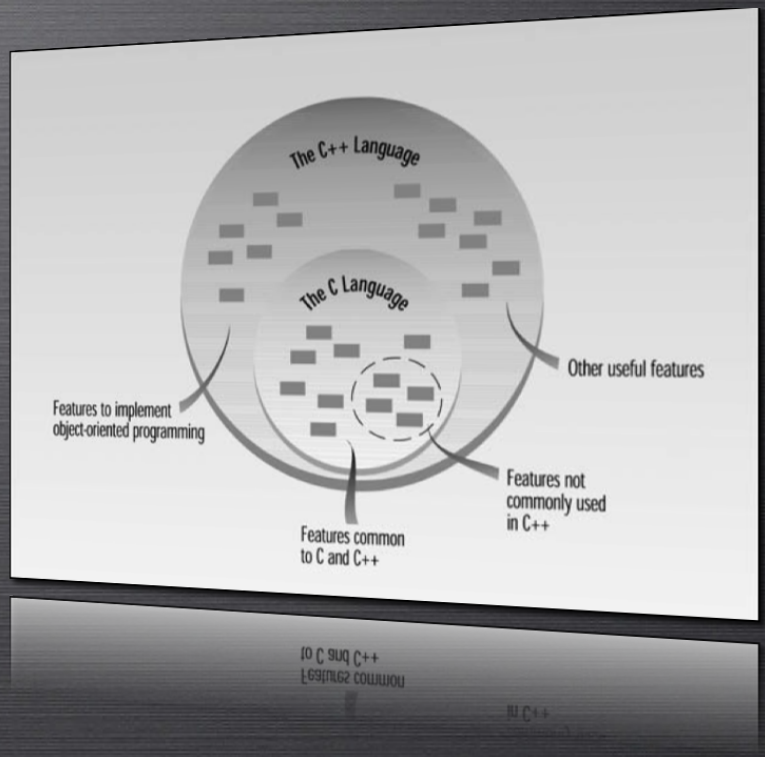
A brief history of C++

1. The foundation of C
   - C – Dennis Ritche (1970s)
   - BCPL – Martin Richards
   - B – Ken Thompson
2. The need for C++
   - Why was C++ invented?
   - Invented by Stroustrup in 1979.
3. The evolution of C++
4. Relationship of C++ with
   - Java
   - C#



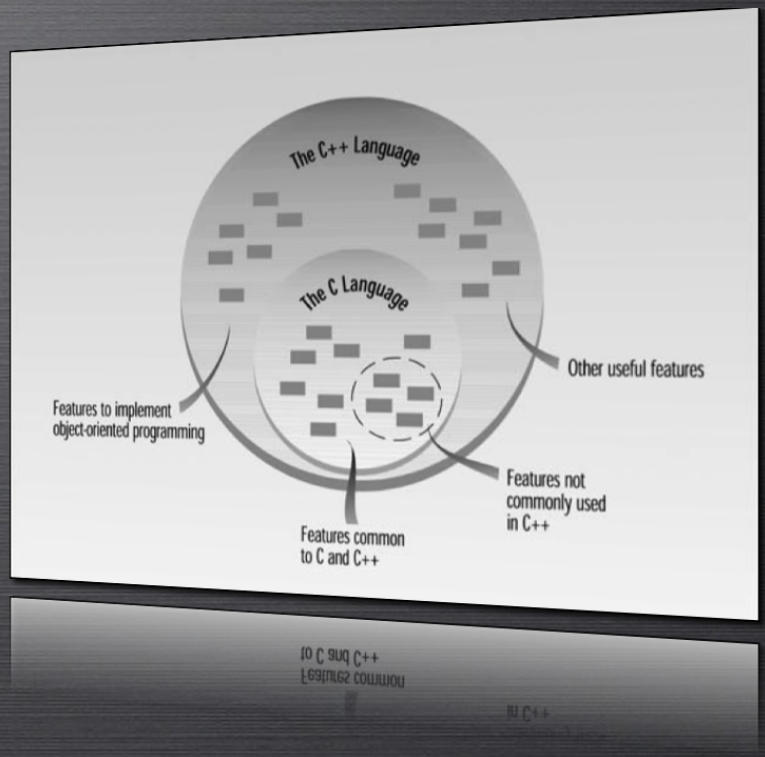
A brief history of C++

1. The foundation of C
   - C – Dennis Ritche (1970s)
   - BCPL – Martin Richards
   - B – Ken Thompson
2. The need for C++
   - Why was C++ invented?
   - Invented by Stroustrup in 1979.
3. The evolution of C++
4. Relationship of C++ with
   - Java
   - C#



A brief history of C++

1. The foundation of C
   - C – Dennis Ritche (1970s)
   - BCPL – Martin Richards
   - B – Ken Thompson
2. The need for C++
   - Why was C++ invented?
   - Invented by Stroustrup in 1979.
3. The evolution of C++
4. Relationship of C++ with
   - Java
   - C#



Object-Oriented Programming (OOP)

1. What is OOP?

Class and Object

A class is a specification describing such a new data form, and an object is particular data structure constructed according to that plan.

2. A simple example of Bank Account

Class	First Name	Last Name	Acc Number	Current Status	Current Balance
Object	Kevin	Kelly	83245-67487	Active	USD 24,657

3. Three traits of C++:
   - Encapsulation
   - Polymorphism
   - Inheritance

Object-Oriented Programming (OOP)

1. What is OOP?

Class and Object

A class is a specification describing such a new data form, and an object is particular data structure constructed according to that plan.

2. A simple example of Bank Account

Class	First Name	Last Name	Acc Number	Current Status	Current Balance
Object	Kevin	Kelly	83245-67487	Active	USD 24,657

3. Three traits of C++:
   - Encapsulation
   - Polymorphism
   - Inheritance

Object-Oriented Programming (OOP)

1. What is OOP?

Class and Object

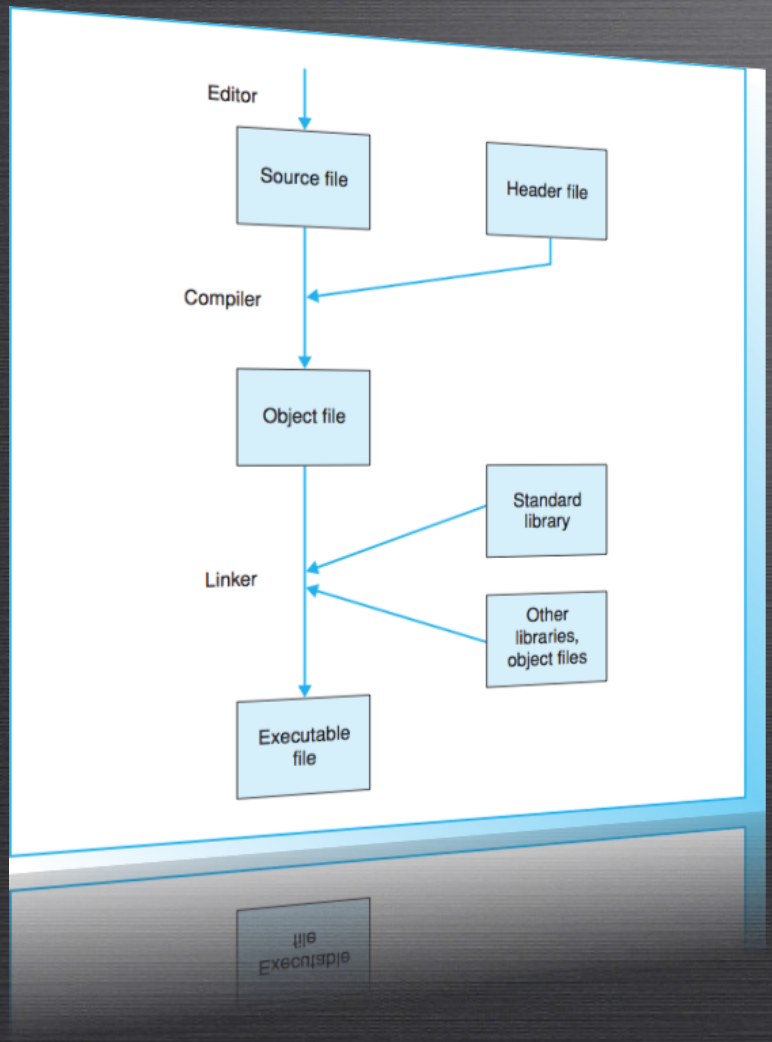
A class is a specification describing such a new data form, and an object is particular data structure constructed according to that plan.

2. A simple example of Bank Account

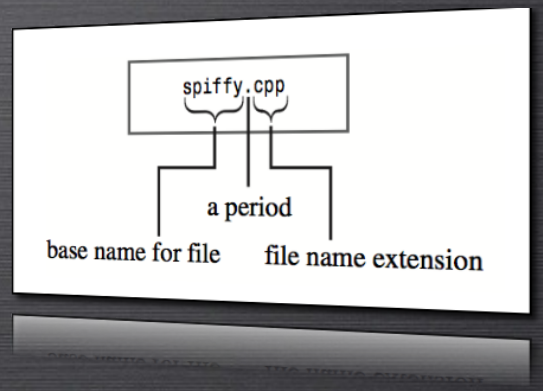
Class	First Name	Last Name	Acc Number	Current Status	Current Balance
Object	Kevin	Kelly	83245-67487	Active	USD 24,657

3. Three traits of C++:
   - Encapsulation
   - Polymorphism
   - Inheritance

Programming steps



Source code	Extension
Unix	c, cc, cxx
Borland C++	cpp
MS Visual C++	cpp, cxx, cc

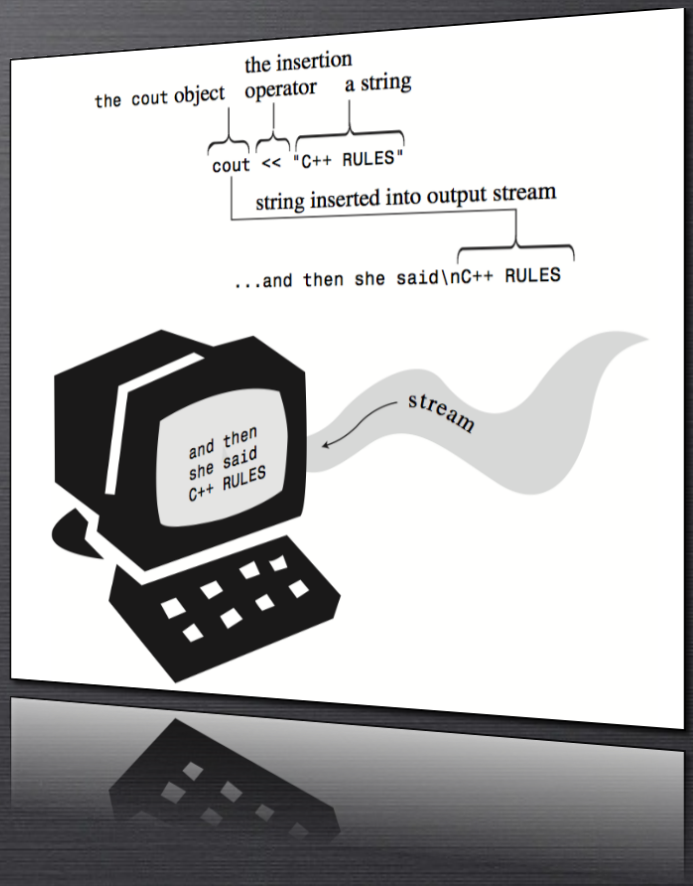


2 types of errors

- Compile errors (missing ; , mis-spelling, linking error)
- Run-time errors (logic error, roundoff, calculations, a program doesn't end)

A simple program

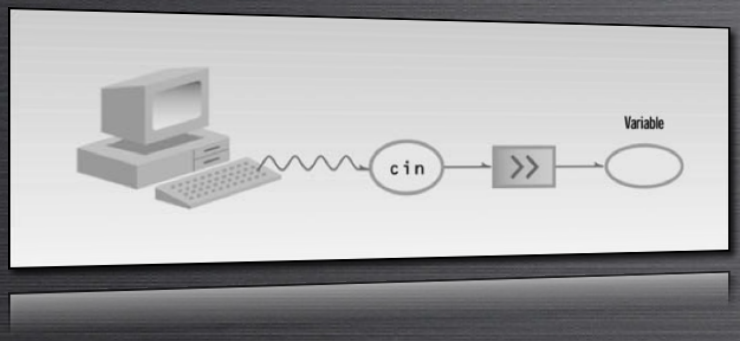
```
/*
   This is a simple C++ program
   to display a message
 */

#include <iostream>
using namespace std;

int main()
{
    cout << "C++ is powerful.";
    cout << endl;

    cout << "I love C++." << endl;

    return 0;
}
```

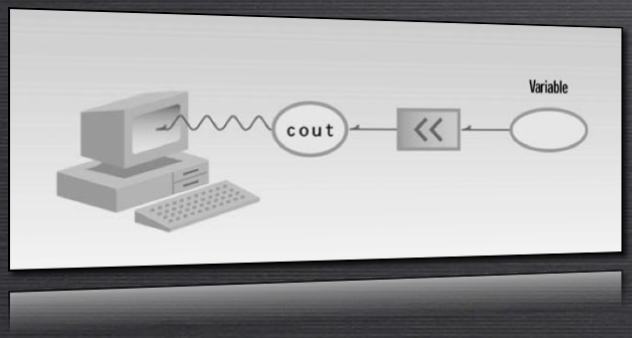


Carrots

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

int main()
{
    // declare an integer variable
    int carrots;

    // assign a value to the variable
    carrots = 25;

    // display the value of the variable
    cout << "I have ";
    cout << carrots;
    cout << " carrots." << endl;

    // modify the variable
    carrots -= 1;
    cout << "Crunch, crunch. Now I have " << carrots << " carrots." << endl;

    return 0;
}
```

cout and *cin*






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

int main()
{
      int carrots;

      cout << "How many carrots do you have? ";
      cin >> carrots; // C++ input

      cout << "Here are two more. ";
      carrots = carrots + 2;

      // the next line concatenates output
      cout << "Now you have " << carrots
            << " carrots." << endl;

      return 0;
}
```

Arithmetic operators

```
// Compute the area of a rectangle
// given its length and width.

#include <iostream>
using namespace std;

int main()
{
    int length;
    int width;
    int area;

    length = 7;
    width = 5;

    area = length *  width;

    cout << "The area is ";
    cout << area << endl;

    return 0;
}
```

Output

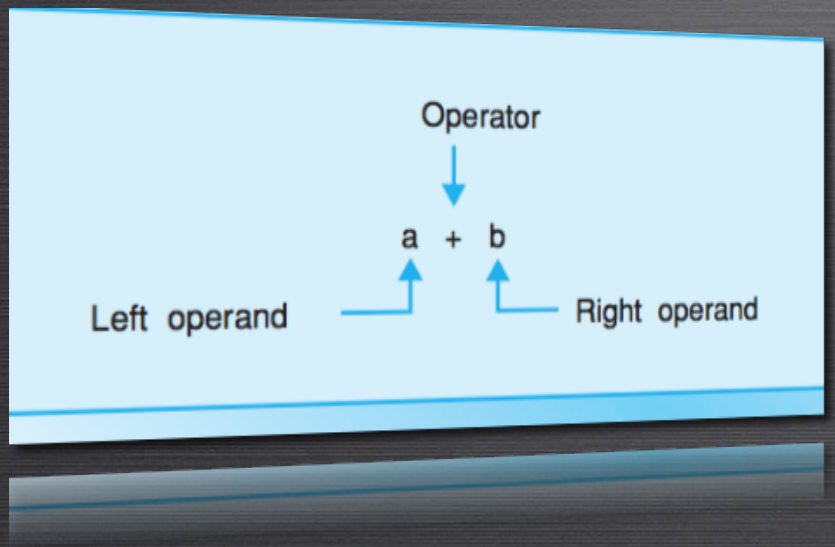
```
The area is 35
```

C++ supports a full range of arithmetic operators that enable you to manipulate numeric values used in a program.

Operator	Significance
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Remainder

A simple program computes the area of a rectangle with three variables: length, width, and area.

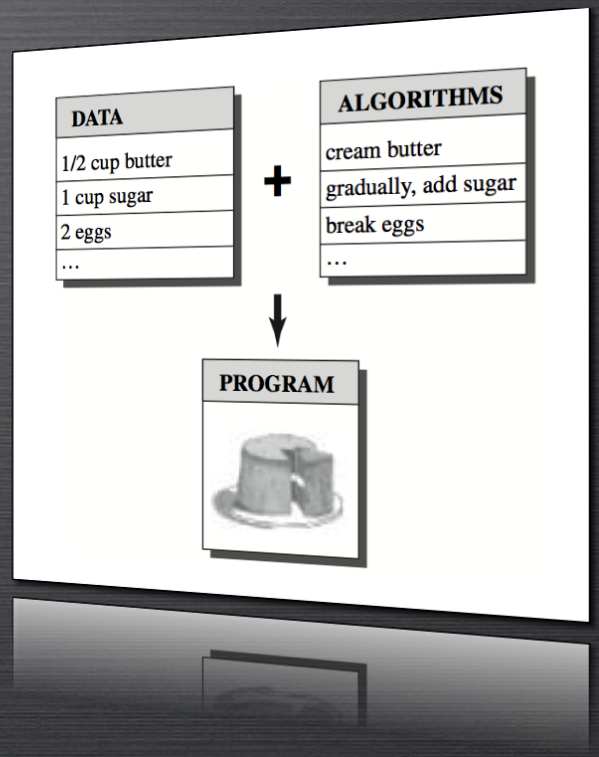
A simplified version




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

int main()
{
    // int length;
    // int width;
    // int area;
    // length = 7;
    // width = 5;

    int length = 7, width = 5;

    // area = length *  width;
    // cout << "The area is " << area;

    cout << "The area is " <<
            length * width << endl;

    return 0;
}
```

An interactive program

```
/*
 *  An interactive program that
 *  computes the area of a rectangle.
 */

#include <iostream>
using namespace std;

int main()
{
     int length, width;

     cout << "Enter the length: ";
     cin >> length;

     cout << "Enter the width: ";
     cin >> width;

     cout << "The area is " << length * width;
     cout << endl;

     return 0;
}
```



Another data type

```
/*
 *   This program illustrates the differences
 *   between integer and double data types
 */

#include <iostream>
using namespace std;

int main()
{
    int ivar = 100;
    double dvar = 100.0;

    cout << "Original value of ivar: " << ivar << endl;
    cout << "Original value of dvar: " << dvar << endl;

    cout << endl;

    // Divide both by 3
    ivar = ivar / 3;
    dvar = dvar / 3;

    cout << "ivar after division: " << ivar << endl;
    cout << "dvar after division: " << dvar << endl;

    return 0;
}
```

When a fractional component is required, the integer type (*int*) cannot be used.

C++ defines other floating type: *float* and *double*. For example,

Double

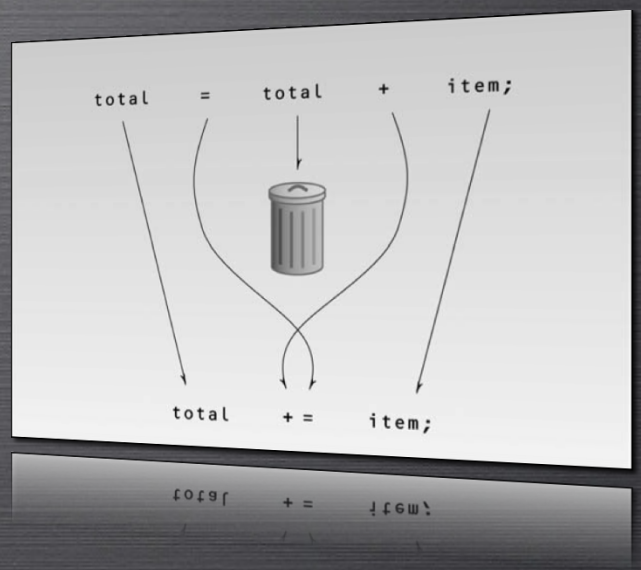
```
double result;
```

Output

```
Original value of ivar: 100
Original value of dvar: 100
ivar after division: 33
dvar after division: 33.3333
```

Compound assignment operators

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

int main()
{
      int ans = 27;

      ans += 10; // ans = ans + 10
      cout << ans << ", ";

      ans -= 7; // ans = ans - 7
      cout << ans << ", ";

      ans *= 2; // ans = ans * 2
      cout << ans << ", ";

      ans /= 3; // ans = ans / 3
      cout << ans << ", ";

      ans %= 3; // ans = ans % 3
      cout << ans << endl;

      return 0;
}
```



Output

Here is the output: 37, 30, 60, 20, 2

Two benefits of the compound assignment:

1. More compact than the tradition way.
2. More efficient executable code.

Increment and decrement

$$x{+}{+} \;:\; x = x + 1$$

$$x{-}{-} \;:\; x = x - 1$$

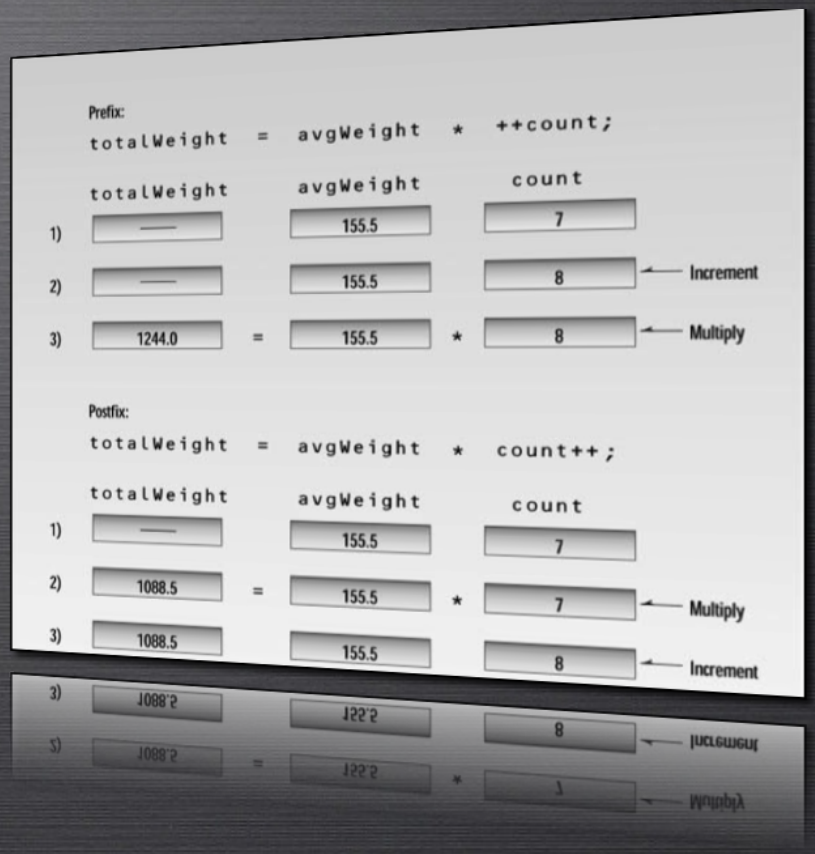
The increment ++ operator adds 1 to its operand, and the decrement −− operator subtracts 1. What is the difference between x++ and ++x?

For example, we have:

1. Case 1: x = 10; y = ++x;
2. Case 2: x = 10; y = x++;

Answer:

1. Case 1: x = 11; y = 11;
2. Case 2: x = 11; y = 10;



Increment and decrement

$$x{+}{+} \;:\; x = x + 1$$

$$x{-}{-} \;:\; x = x - 1$$

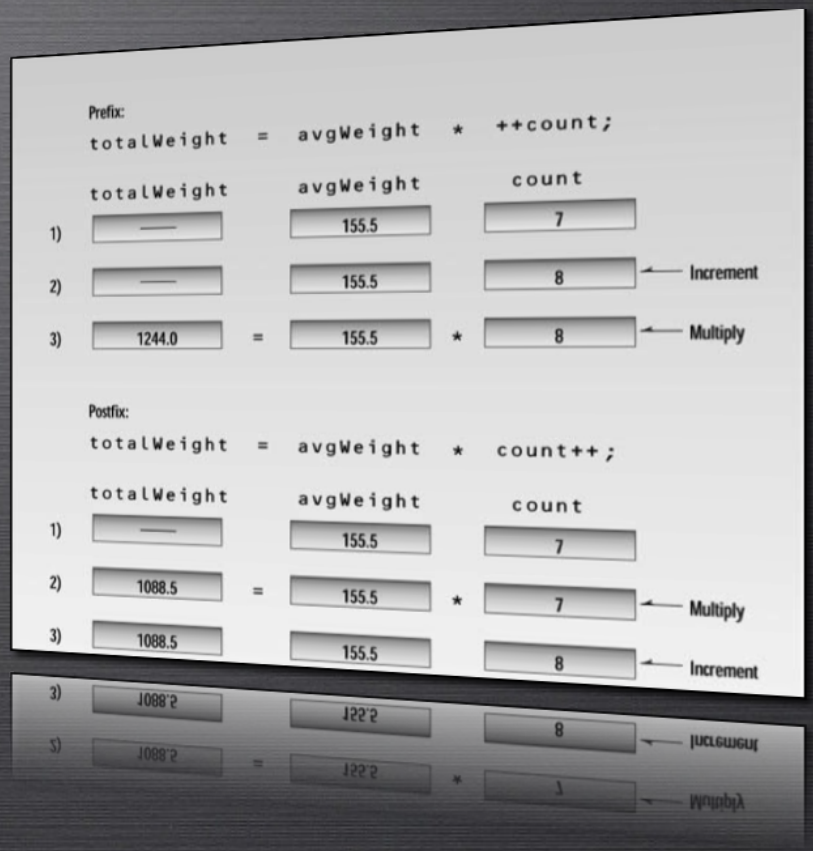
The increment ++ operator adds 1 to its operand, and the decrement −− operator subtracts 1. What is the difference between x++ and ++x?

For example, we have:

1. Case 1: x = 10; y = ++x;
2. Case 2: x = 10; y = x++;

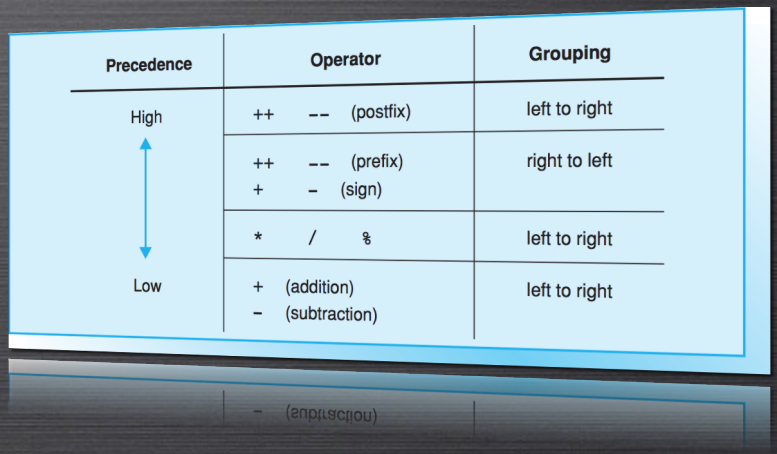
Answer:

1. Case 1: x = 11; y = 11;
2. Case 2: x = 11; y = 10;



Prefix and postfix example

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

int main()
{
     int count = 10;

     // Displays 10
     cout << "count = " << count << endl;

     // Displays 11 (prefix)
     cout << "count = " << ++count << endl;

     // Displays 11
     cout << "count = " << count << endl;

     // Displays 11 (postfix)
     cout << "count = " << count++ << endl;

     // Displays 12
     cout << "count = " << count << endl;

     return 0;
}
```



Precedence	Operator	Grouping
High	++ -- (postfix)	left to right
	++ -- (prefix) + - (sign)	right to left
	* / %	left to right
Low	+ (addition) - (subtraction)	left to right

Relational and logical operators

Operator	Significance
<	less than
<=	less than or equal to
>	greater than
>=	geater than or equal to
==	equal
!=	unequal

A	B	A && B	A || B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

A	!A
true	false
false	true

Demonstrate relational operators

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

int main()
{
    int num;

    cout << "Enter a number: ";
    cin >> num;

    cout << num << " < 10 is " << (num < 10) << endl;
    cout << num << " > 10 is " << (num > 10) << endl;
    cout << num << " == 10 is " << (num==10) << endl;

    return 0;
}
```

Output

```
Enter a number: 99
99 < 10 is 0
99 > 10 is 1
99 = 0 is 0
```

The *if* statement

The simplest form

if (*condition*) statement;

If the condition is true (*non-zero*), then the statement will execute.

If the condition is false (*zero*), then the statement will not execute.

Operator	Meaning
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal to
! =	Not equal

Demonstrate the *if* statement

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

int main()
{
    int a = 2, b = 3, c, d;

    if (a < b) cout << "a is less than b" << endl;
    if (a == b) cout << "a is equal to b" << endl;

    c = a - b;
    cout << "c is " << c << endl;
    if (c >= 0) cout << "c is non-negative." << endl;
    if (c < 0) cout << "c is negative." << endl;

    d = b - a;
    cout << "d is " << d << endl;
    if (d >= 0) cout << "d is non-negative." << endl;
    if (d < 0) cout << "d is negative." << endl;

    return 0;
}
```

The *for* loop statement

The simplest form

for (initialization; *condition*; increment) statement;

1. One of powerful C++ loop constructs.
2. Initialization sets a loop control variable to an initial value.
3. Condition is an expression that is tested each time the loop repeats. As long as condition is true (*non-zero*), the loop keeps running.
4. Increment is an expression that determines how the loop control variable is incremented each time the loop repeats.

Demonstrate the *for* loop statement

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

int main()
{
      for (int count=1; count <= 100; count = count+1)
            cout << count << "  ";

      return 0;
}
```

C++ professional coding

NOT: count = count+1 ⟶ count++

for (int count=1; count <= 100; count++) cout << count << " ";

Demonstrate the *for* loop statement

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

int main()
{
      for (int count=1; count <= 100; count = count+1)
            cout << count << " ";

      return 0;
}
```

C++ professional coding

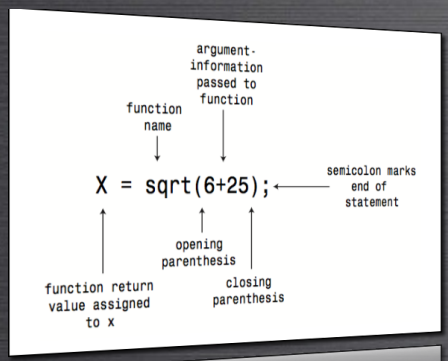
NOT: count = count+1 ⟶ count++

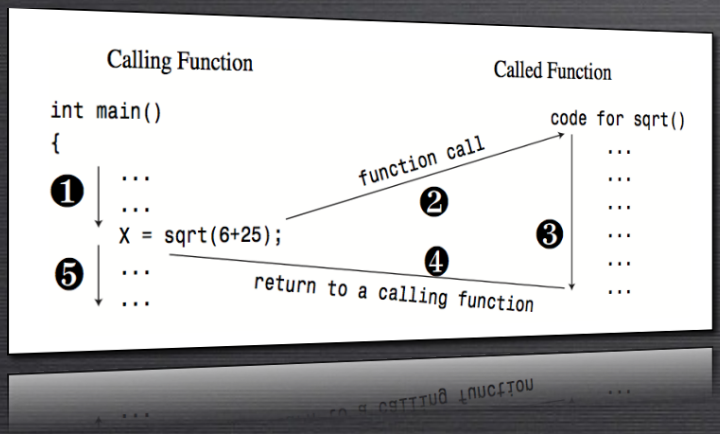
for (int count=1; count <= 100; count++) cout << count << " ";

Demonstrate a block of code

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

int main()
{
      double n, d;

      cout << "Enter a numerator: ";
      cin >> n;
      cout << "Enter a denominator: ";
      cin >> d;

      if (d != 0)
      {
            cout << "A denominator does not equal zero, so division is OK" << endl;
            cout << n << "/" << d << " is " << n / d << endl;
      }

      return 0;
}
```

Square root function






```cpp
// How to use a sqrt function

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

int main()
{
      double x, result;

      cout << "Enter a value: ";
      cin >> x;

      result = sqrt(x);
      cout << "The square root of " << x;
      cout << " is " << result << endl;

      return 0;
}
```

Summary

1. C++ solves the complexity
2. *main* function
3. C++ operators
   - Arithmetic operators
   - Assignment operators
   - Relational and logical operators
4. *if* control statement
5. *for* control statement
6. Functions

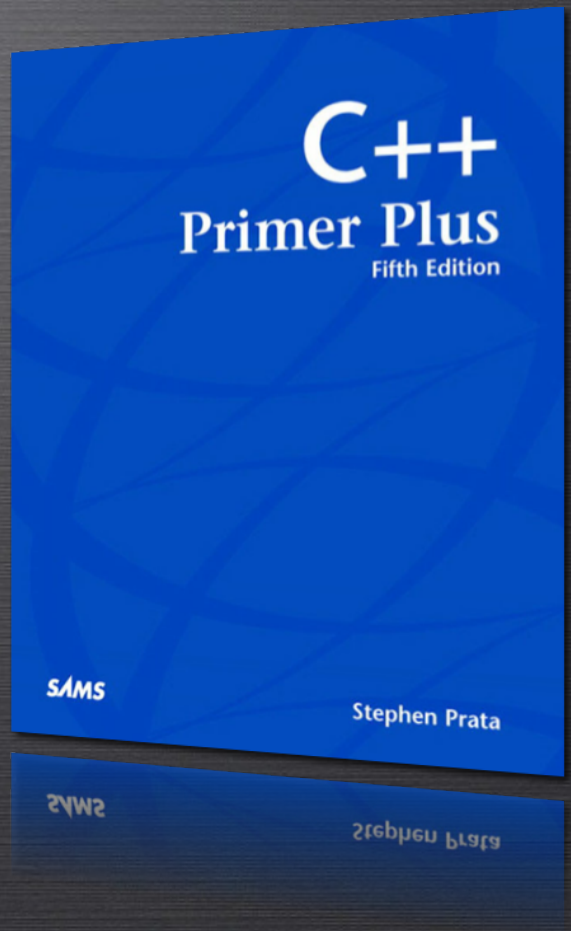
Reading

Stephen Prata.

C++ Primer Plus, 5th Edition,

Chapter 1, 2

SAMS Publishing, 2004.



Exercise 1

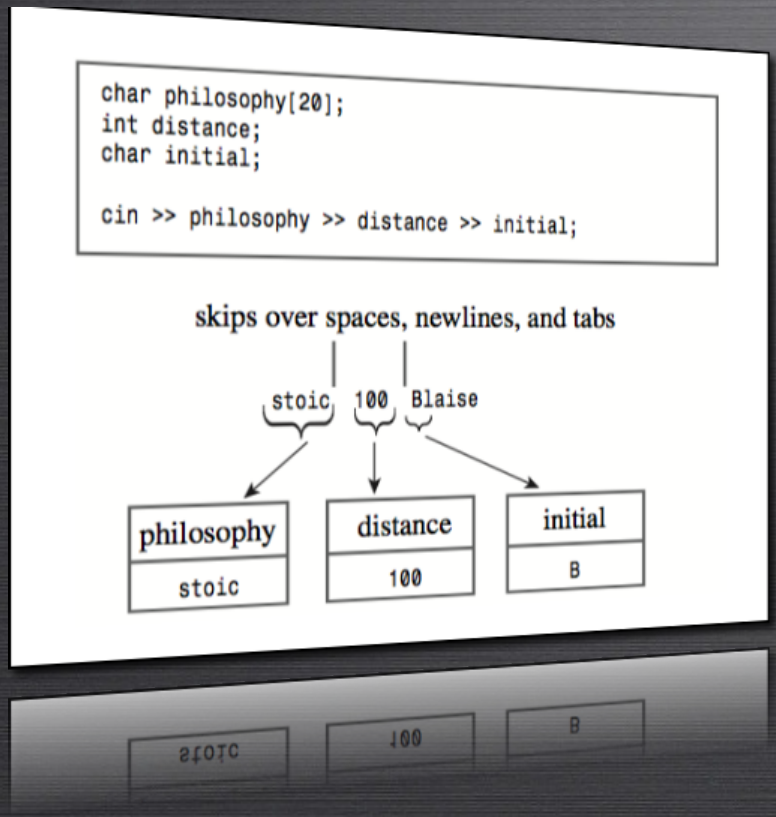
Write a C++ program prompts users to type in a floating-point number representing the *radius* of a circle, then calculates and displays the circle's *area*.

Exercise 2

Write a C++ program that allows users to enter an amount in USD, and displays this value converted to four other monetary units (British Pound, Chinese Yuan, European Euro, and Japanese Yen.)

Given that on March 20, 2012, one British Pound (GBP) was equivalent to 1.5887 USD, one Chinese Yuan (CNY) was 0.1581 USD, one European Euro (EUR) was 1.3269 USD, and one Japanese Yen (JPY) was 0.0120 USD.

An exercise with *cin*



Write a C++ program that allows the user to enter two fractions, and then displays their sum in fractional form. We do not need to reduce it to lowest terms.