COURSE TECHNOLOGY
CENGAGE Learning

Invitation to Computer Science
5th Edition

Chapter C++
Programming in C++

Objectives

In this chapter, you will learn about:
- Simple C++ programs
- Virtual data storage
- Statement types
- An example of a complete program

Objectives (continued)

- Managing complexity
- Object-oriented programming
- Graphical programming

Introduction

- In this chapter:
  - You will get a sense of what programming in a high-level language is like

A Simple C++ Program

- Comments
  - Anything appearing on a line after the double slash symbol (//)
  - Ignored by the compiler
- Prologue comment
  - Introductory comment
  - Optional
- Include directive (line 5)
  - Refers to the *iostream* library


```
prologue comment     [optional]
include directives   [optional]
using directive      [optional]
functions            [optional]
main function
{
      declarations   [optional]
      main function body
}
```

Figure 2 The Overall Form of a Typical C++ Program

```
1. //Computes and outputs travel time
2. //for a given speed and distance
3. //Written by J. Q. Programmer, 6/15/10
4.
5. #include <iostream>
6. using namespace std;
7.
8. void main()
9. {
10.   int speed;       //rate of travel
11.   double distance; //miles to travel
12.   double time;     //time needed for this travel
13.
14.   cout << "Enter your speed in mph: ";
15.   cin >> speed;
16.   cout << "Enter your distance in miles: ";
17.   cin >> distance;
18.
19.   time = distance/speed;
20.
21.   cout << "At " << speed << " mph, "
22.        << "it will take " << endl;
23.   cout << time << " hours to travel "
24.        << distance << " miles." << endl;
25. }
```

Figure 3 The Program of Figure 1 (line numbers added for reference)

A Simple C++ Program (continued)

- Using directive (line 6)
  - Tells the compiler to look in the *std* namespace for the definition of any names not specifically defined within the program
- Function body (between braces at lines 9 and 25)
  - The heart of the sample program

A Simple C++ Program (continued)

- Syntax
  - The correct form for each component of the language
- C++ is a **free-format language**
  - It does not matter where things are placed on a line

Creating and Running a C++ Program

- First step
  - Type the program into a text editor
- Second step
  - Program must be compiled using a C++ compiler and the resulting object code linked with any C++ library object code
- Third step
  - Load and execute the program file

Creating and Running a C++ Program (continued)

- Integrated Development Environment (IDE)
  - Lets the programmer perform a number of tasks within the shell of a single application program
  - Usually has a GUI (graphical user interface) with menu choices for the different task

Virtual Data Storage

- Identifiers
  - Names in a programming language
  - Cannot be **keywords**
- Case-sensitive
  - Uppercase letters are distinguished from lowercase letters
- Constants
  - Values are fixed and known ahead of time

Virtual Data Storage (continued)

- Variables
  - Values that change as the program executes
- Data type
  - Determines how many bytes will be needed to store the variable
- Variable declaration
  - Consists of a data type followed by a list of one or more identifiers of that type
- Array
  - Groups together a collection of memory locations, all storing data of the same type

int	an integer quantity
double	a real number
char	a character (a single keyboard character, such as 'a')

Figure 4 Some of the C++ Primitive Data Types

Figure 5 A 12-Element Array *hits*

Statement Types

- Input statement
  - Collects a value from the user for a variable within the program
- Output statement
  - Writes a message or the value of a program variable to the user's screen
- Assignment statement
  - Assigns a value to a program variable

Statement Types (continued)

- Control statements
  - Affect the order in which instructions are executed
- Flow of control in the program
  - The path through the program that is traced by following the currently executing statement

Input/Output Statements

- Input stream
  - Sequence of values entered at the keyboard
- Extraction operator
  - Removes the next value from the input stream and stores it in the memory location
- Fixed-point format
  - 11.3362
- Floating-point format
  - 1.13362e+001

The Assignment Statement

- Pseudocode operation
  Set the value of "variable" to "arithmetic expression"
  C++ equivalent
  variable = expression;
- Basic arithmetic operations
  - \+ Addition
  - \- Subtraction
  - * Multiplication
  - / Division

Control Statements

- Control mechanisms
  - **Sequential**: instructions are executed in order
  - **Conditional**: which instruction executes next depends on some condition
  - **Looping**: a group of instructions may be executed many times
- Boolean condition
  - Can be either true or false
  - Often involves comparing the values of two expressions and determining whether they are equal

Figure 6 Sequential Flow of Control

Comparison	Symbol	Example	Example Result
the same value as	==	2 == 5	false
less than	<	2 < 5	true
less than or equal to	<=	5 <= 5	true
greater than	>	2 > 5	false
greater than or equal to	>=	2 >= 5	false
not the same value as	!=	2 != 5	true

Figure 7 C++ Comparison Operators

Operator	Symbol	Example	Example Result
AND	&&	(2 < 5) && (2 > 7)	false
OR	||	(2 < 5) || (2 > 7)	true
NOT	!	!(2 == 5)	true

Figure 8 C++ Boolean Operators

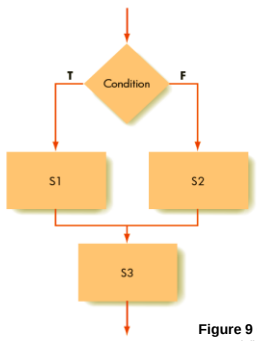


Figure 9 Conditional Flow of Control (if-else)

Control Statements (continued)

- Compound statement
  - Can be used anywhere a single statement is allowed
  - Example

```
{
  cout << "This is the first statement." << endl;
  cout << "This is the second statement." <<
  endl;
  cout << "This is the third statement." << endl;
}
```

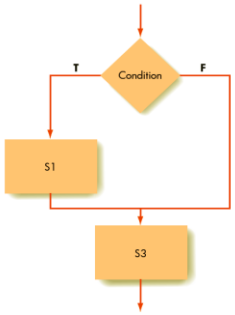



Figure 10 If-Else with Empty Else


```
//Computes and outputs travel time
//for a given speed and distance
//Written by J. Q. Programmer, 6/28/10

#include <iostream>
using namespace std;

void main()
{
    int speed;          //rate of travel
    double distance;    //miles to travel
    double time;        //time needed for this travel
    int hours;          //time for travel in hours
    int minutes;        //leftover time in minutes
    char choice;        //choice of output as
                        //decimal hours
                        //or hours and minutes

    cout << "Enter your speed in mph: ";
    cin >> speed;
    cout << "Enter your distance in miles: ";
    cin >> distance;
    cout << "Enter your choice of format"
         << " for time, " << endl;
    cout << "decimal hours (D) "
         << "or hours and minutes (M): ";
    cin >> choice;
```

Figure 11 The TravelPlanner Program with a Conditional Statement

```
    if (choice == 'D')
    {
       time = distance/speed;
       cout << "At " << speed << " mph, "
            << "it will take " << endl;
       cout << time << " hours to travel "
            << distance << " miles." << endl;
    }
    else
    {
       time = distance/speed;
       hours = int(time);
       minutes = int((time - hours)*60);
       cout << "At " << speed << " mph, "
            << "it will take " << endl;
       cout << hours << " hours and "
            << minutes << " minutes to travel "
            << distance << " miles." << endl;
    }
}
```

Figure 11 The TravelPlanner Program with a Conditional Statement (continued)

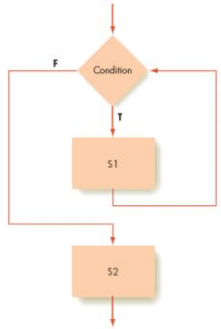


Figure 12 While Loop

Control Statements (continued)

- Initialization of variables
  - Using assignment statements to set the values of certain variables before they are used by the program
- Sentinel value
  - One extra integer that is not part of the legitimate data but is instead a signal that there *are no more data*
- Infinite loop
  - The condition, once true, would remain true forever, and the loop body would be endlessly executed

```
//Computes and outputs travel time
//for a given speed and distance
//Written by J. Q. Programmer, 7/05/10

#include <iostream>
using namespace std;

void main()
{
    int speed;              //rate of travel
    double distance;        //miles to travel
    double time;            //time needed for this travel
    int hours;              //time for travel in hours
```

Figure 13 The TravelPlanner Program with Looping

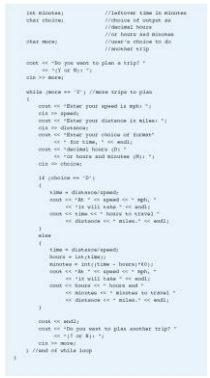


Figure 13 The TravelPlanner Program with Looping (continued)

Another Example

- Example
  - Write a program to assist SportsWorld, a company that installs circular swimming pools
- To estimate costs for swimming pool covers or for fencing
  - SportsWorld needs to know the area or circumference of a pool, given its radius

```
Get value for user's choice about continuing
While user wants to continue, do the following steps
    Get value for pool radius
    Get value for choice of task
    If task choice is circumference
        Compute pool circumference
        Print output
    Else (task choice is area)
        Compute pool area
        Print output
    Get value for user's choice about continuing
Stop
```

Figure 14 A Pseudocode Version of the SportsWorld Program

Figure 15 The SportsWorld Program

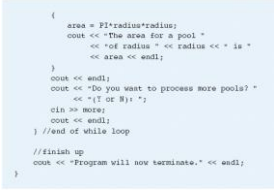


Figure 15 The SportsWorld Program (continued)

```
Do you want to process a pool? (Y or N): Y
Enter the value of the radius of a pool: 2.7
Enter your choice of task.
C to compute circumference, A to compute area: C
The circumference for a pool of radius 2.70 is 16.96
Do you want to process more pools? (Y or N): Y
Enter the value of the radius of a pool: 2.7
Enter your choice of task.
C to compute circumference, A to compute area: A
The area for a pool of radius 2.70 is 22.90
Do you want to process more pools? (Y or N): Y
Enter the value of the radius of a pool: 14.53
Enter your choice of task.
C to compute circumference, A to compute area: C
The circumference for a pool of radius 14.53 is 91.29
Do you want to process more pools? (Y or N): N
Program will now terminate.
```

Figure 16 A Sample Session Using the Program of Figure 15

Managing Complexity

- Divide and conquer
  - A problem-solving approach and not just a computer programming technique
- Figure 17(a)
  - An example of a **structure chart** or **structure diagram**

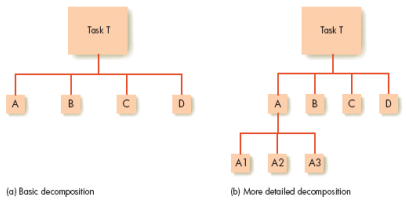


Figure 17 Structure Charts

Using Functions

- Functions
  - Each function in a program should do one and only one subtask
- Argument list
  - List of identifiers for variables pertinent to that function

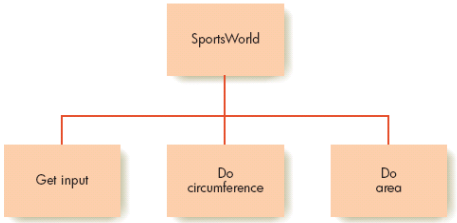


Figure 18 Structure Chart for the SportsWorld Task


```
Get value for user's choice about continuing
While the user wants to continue
  Do the input subtask
  If (Task = 'C') then
    do the circumference subtask
  else
    do the area subtask
  Get value for user's choice about continuing
```

Figure 19 A High-Level Modular View of the SportsWorld Program

```
void main()
{
    double radius;      //radius of a pool - given
    char taskToDo;      //holds user choice to
                        //compute circumference or area
    char more;          //controls loop for
                        //processing more pools

    cout.setf(ios::fixed);
    cout.precision(2);

    cout << "Do you want to process a pool? (Y or N): ";
    cin >> more;
    cout << endl;
    while (more == 'Y') //more circles to process
    {
        getInput(radius, taskToDo);

        if (taskToDo == 'C') //compute circumference
            doCircumference(radius);
        else //compute area
            doArea(radius);

        cout << endl;
        cout << "Do you want to process more pools? (Y or N): ";
        cin >> more;
        cout << endl;
    }

    //finish up
    cout << "Program will now terminate." << endl;
}
```

Figure 20 The Main Function in a Modularized Version of the SportsWorld Program

Writing Functions

- Any function can invoke another function
- A function can even invoke itself
- The function header consists of three parts
  - A return indicator
  - The function identifier
  - A parameter list


```
function header
{
   local declarations   [optional]
   function body
}
```

Figure 21 The Outline for a C++ Function

Writing Functions (continued)

- Argument is **passed by value**
  - If the value is one that the function must know to do its job but should not change
- Argument **passed by reference**
  - If value passed to the function is one that the function should change, and the main function should know the new value

Writing Functions (continued)

- By default
  - Arguments in C++ are passed by value, which protects them from change by the function
- *getInput* function
  - Both *radius* and *taskToDo* are values that *getInput* obtains from the user

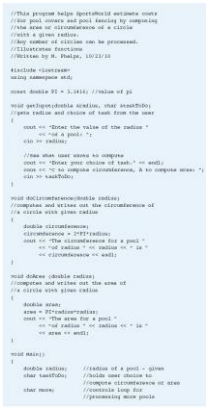

```
void getInput(double &radius, char &taskToDo)
//gets radius and choice of task from the user
{
    cout << "Enter the value of the radius "
         << "of a pool: ";
    cin >> radius;

    //See what user wants to compute
    cout << "Enter your choice of task." << endl;
    cout << "C to compute circumference, A to compute area: ";
    cin >> taskToDo;
}
```

Figure 22 The getInput Function

```
void doCircumference(double radius)
//computes and writes out the circumference of
//a circle with given radius
{
    double circumference;
    circumference = 2*PI*radius;
    cout << "The circumference for a pool "
         << "of radius " << radius << " is "
         << circumference << endl;
}
```

Figure 23 The doCircumference Function

Figure 24 The Complete Modularized SportsWorld Program




```
    cout.setf(ios::fixed);
    cout.precision(2);

    cout << "Do you want to process a pool? (Y or N): ";
    cin >> more;
    cout << endl;
    while (more == 'Y') //more circles to process
    {
      getInput(radius, taskToDo);
      if (taskToDo == 'C') //compute circumference
         doCircumference(radius);
      else  //compute area
         doArea(radius);

      cout << endl;
      cout << "Do you want to process more pools? (Y or N): ";
      cin >> more;
      cout << endl;
    }

    //finish up
    cout << "Program will now terminate." << endl;
}
```

Figure 24 The Complete Modularized SportsWorld Program (continued)

Writing Functions (continued)

- Modularizing a program is useful for:
  - Planning
  - Coding
  - Testing
  - Modifying
  - Reading
- Nonvoid function
  - A function that returns a single value to the section of the program that invoked
  - Must contain a return statement

Figure 25 The SportsWorld Program Using Nonvoid Functions

```
//This program helps SportsWorld estimate costs
//for pool covers and pool fencing by computing
//the area or circumference of a circle
//with a given radius.
//Any number of circles can be processed.
//Illustrates nonvoid functions
//Written by M. Phelps, 10/23/10

#include <iostream>
using namespace std;

const double PI = 3.1416; //value of pi

void getInput(double &radius, char &taskToDo);
//gets radius and choice of task from the user
{
    cout << "Enter the value of the radius "
         << "of a pool: ";
    cin >> radius;

    //See what user wants to compute
    cout << "Enter your choice of task." << endl;
    cout << "C to compute circumference, A to compute area: ";
    cin >> taskToDo;
}

double docircumference(double radius);
//computes the circumference of a circle with given radius
{
    return 2*PI*radius;
}

double doArea(double radius);
//computes the area of a circle with given radius
{
    return PI*radius*radius;
}

void main();
{
    double radius; //radius of a circle - given
    char taskToDo; //holds user choice to
                   //compute circumference or area
    char more;     //controls loop for processing more circles

    cout.setf(ios::fixed);
    cout.precision(2);

    cout << "Do you want to process a pool? (Y or N): ";
    cin >> more;
    cout << endl;
    while (more == 'Y') //more circles to process
```



```
    {
        getInput(radius, taskToDo);
        if (taskToDo == 'C') //compute circumference
        {
            cout << "The circumference for a pool "
                 << "of radius " << radius << " is "
                 << docircumference(radius) << endl;
        }
        else              //compute area
        {
           cout << "The area for a pool "
                << "of radius " << radius << " is "
                << doArea(radius) << endl;
        }
        cout << endl;
        cout << "Do you want to process more pools? (Y or N): ";
        cin >> more;
        cout << endl;
    }

    //finish up
    cout << "Program will now terminate." << endl;
}
```

Figure 25 The SportsWorld Program Using Nonvoid Functions (continued)

Term	Meaning	Term	Meaning
Local variable	Declared and known only within a function	Global constant	Declared outside any function and known everywhere
Argument passed by value	Function receives a copy of the value and can make no permanent changes in the value	Argument passed by reference	Function gets access to memory location where the value is stored; changes it makes to the value persist after control returns to main function
Void function	Performs a task, function invocation is a complete C++ statement	Nonvoid function	Computes a value; must include a return statement; function invocation is used within another C++ statement

Figure 26 Some C++ Terminology

Object-Oriented Programming

- A program is considered a simulation of some part of the world that is the domain of interest
  - "Objects" populate this domain
- When an object-oriented program is executed
  - The program generates requests for services that go to the various objects
- Terms associated with object-oriented programming
  - Encapsulation
  - Inheritance
  - Polymorphism

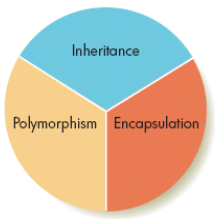


Figure 27 Three Key Elements of OOP

C++ and OOP

- Member variables
  - Properties of a class
- Member functions
  - Services that any object of the class can perform
- Objects
  - **Instances of** classes


```
//This program helps SportsWorld estimate costs
//for pool covers and pool fencing by computing
//the area or circumference of a circle
//with a given radius.
//Any number of circles can be processed.
//Uses class Circle
//Written by M. Phelps, 10/23/10

#include <iostream>
using namespace std;

const double PI = 3.1416; //value of pi

//class interface
class Circle
{
public:
    void setRadius(double value);
    //sets radius equal to value

    double getRadius();
    //returns current radius

    double doCircumference();
    //computes and returns circumference of a circle

    double doArea();
    //computes and returns area of a circle

private:
    double radius;
};  //end of class interface
```

Figure 28 An Object-Oriented SportsWorld Program


```
void getInput(double &newRadius, char &taskToDo)
//gets radius and choice of task from the user
{

    cout << "Enter the value of the radius "
         << "of a pool: ";
    cin >> newRadius;

    //See what user wants to compute
    cout << "Enter your choice of task." << endl;
    cout << "C to compute circumference, A to compute area: ";
    cin >> taskToDo;
}

//main function
void main()
{
    double radius;     //radius of a pool - given
    char taskToDo;     //holds user choice to
                       //compute circumference or area
    char more;         //controls loop for
                       //processing more pools
    Circle swimmingPool;  //create a Circle object
    cout.setf(ios::fixed);
    cout.precision(1);

    cout << "Do you want to process a pool? (Y or N): ";
    cin >> more;
    cout << endl;
    while (more == 'Y') //more circles to process
    {
      getInput(radius, taskToDo);
      swimmingPool.setRadius(radius);
      if (taskToDo == 'C') //compute circumference
        cout << "The circumference for a pool "
             << "of radius " << swimmingPool.getRadius();
             << " is " << swimmingPool.doCircumference() << endl;
      else     //compute area
        cout << "The area for a pool "
             << "of radius is " << swimmingPool.getRadius();
             << " is " << swimmingPool.doArea() << endl;

      cout << endl;
      cout << "Do you want to process more pools? (Y or N): ";
      cin >> more;
      cout << endl;
    }

    //finish up
    cout << "Program will now terminate." << endl;
}

//class implementation
```

Figure 28 An Object-Oriented SportsWorld Program (continued)

```
void Circle::setRadius(double value)
//sets radius equal to value
{
    radius = value;
}

double Circle::getRadius()
//returns current radius
{
    return radius;
}

double Circle::doCircumference()
//computes and returns circumference of a circle
{
    return 2*PI*radius;
}

double Circle::doArea()
//computes and returns area of a circle
{
    return PI*radius*radius;
}
```

Figure 28 An Object-Oriented SportsWorld Program (continued)

One More Example

- In Figure 29
  - *Circle* object has a radius property
  - *Rectangle* object has a width property and a height property
  - Any *Circle* object can set the value of its radius and can compute its area
  - A *Square* object has a side property
  - *Square2* object doesn't have any properties or any way to compute its area

Figure 29 A C++ Program with Polymorphism and Inheritance

Figure 29 A C++ Program with Polymorphism and Inheritance (continued)

Figure 29 A C++ Program with Polymorphism and Inheritance (continued)


```
The area of a circle with radius 23.5 is 1734.95
The area of a rectangle with dimensions 12.4 and 18.1 is 224.44
The area of a square with side 3 is 9
The area of a square with side 4.2 is 17.64
```

Figure 30 Output from the Program of Figure 29

One More Example (continued)

- *Square*
  - A stand-alone class with a side property and a *doArea* function
- *Square2* class
  - Recognizes the fact that squares are special kinds of rectangles
  - Subclass of the *Rectangle class*
  - Inherits the width and height properties from the "parent" *Rectangle* class

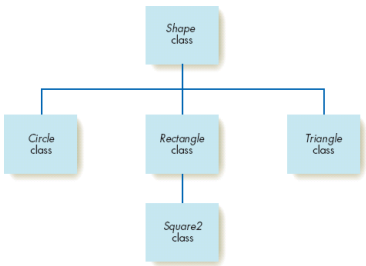


Figure 31 A Hierarchy of Geometric Classes

What Have We Gained?

- Reasons why OOP is a popular way to program
  - Software reuse
  - A more natural "worldview"
- Software reuse
  - Useful class that has been implemented and tested becomes a component available for use in future software development

A More “Natural” Worldview

- Object-oriented programming
  - Recognizes that in the “real world,” tasks are done by entities (objects)
  - Allows the programmer to come closer to modeling or simulating the world as we see it
- Object-oriented program design
  - Begins by identifying objects that are important in the domain of the program

Graphical Programming

- Graphics
  - Make it easier to manage tasks of the operating system
  - Can help us visualize and make sense of massive amounts of output produced by programs that model complex physical, social, and mathematical systems

Graphics Primitives

- Bitmapped display
  - Screen is made up of thousands of individual picture elements, or **pixels,** laid out in a two-dimensional grid
- High-resolution terminals
  - Terminals with a high density of pixels
- Frame buffer
  - Memory that stores the actual screen image

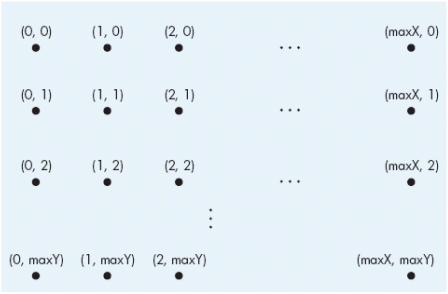


Figure 32 Pixel-Numbering System in a Bitmapped Display

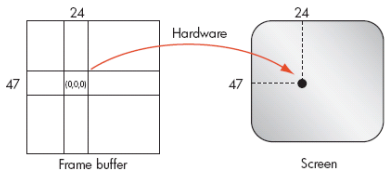


Figure 33 Display of Information on the Hardware Terminal

Graphics Primitives (continued)

- Graphics library
  - Collection of functions
- Functions available in the *Invitation to Computer Science* package
  - *clearscreen(l) , moveto(x, y)*
  - *getmaxx( ), getmaxy( )*
  - *setcolor(l), lineto(x, y)*
  - *rectangle(x1, y1, x2, y2)*
  - *circle(x, y, r), writedraw(value, x, y)*
  - *getmouse(x, y)*

An Example of Graphics Programming

- Titled windows
  - Are part of just about every graphical interface
- Commands to draw four lines in the desired position

```
moveto(50, 60); lineto(150, 60);
moveto(50, 70); lineto(150, 70);
moveto(250, 60); lineto(350, 60);
moveto(250, 70); lineto(350, 70);
```



```
clearscreen(0);
setcolor(1);
rectangle(50, 50, 350, 80);
rectangle(50, 80, 350, 300);
moveto(50, 60);
lineto(150, 60);
moveto(50, 70);
lineto(150, 70);
moveto(250, 60);
lineto(350, 60);
moveto(250, 70);
lineto(350, 70);
writedraw("Title", 180, 70);
```

Figure 34 Commands to Produce a Titled Window

Summary

- Comments
  - Anything appearing on a line after the double slash symbol (//)
- Syntax
  - The correct form for each component of the language
- Integrated Development Environment (IDE)
  - Lets the programmer perform a number of tasks within the shell of a single application program
- Statement types
  - Input statement, output statement, and assignment statement

Summary (continued)

- Control mechanisms
  - Sequential, conditional, and looping
- Divide and conquer
  - A problem-solving approach and not just a computer programming technique
- Functions
  - Each function in a program should do one and only one subtask

Summary (continued)

- Object-oriented programming
  - Recognizes that in the "real world," tasks are done by entities (objects)
- Graphics
  - Make it easier to manage tasks of the operating system