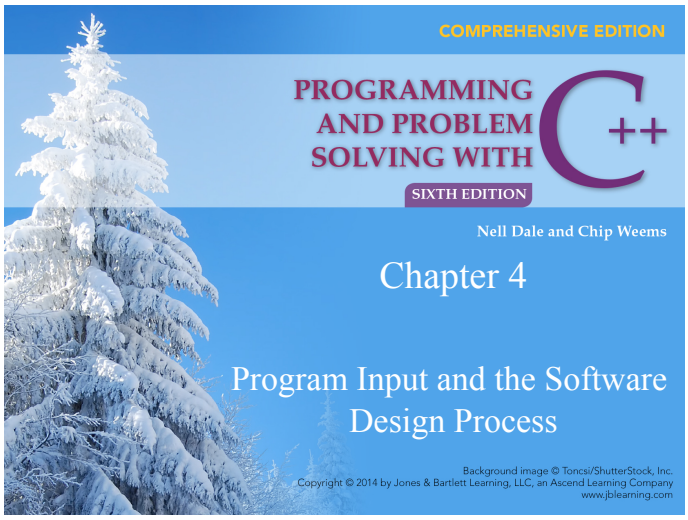


Chapter 4 Topics

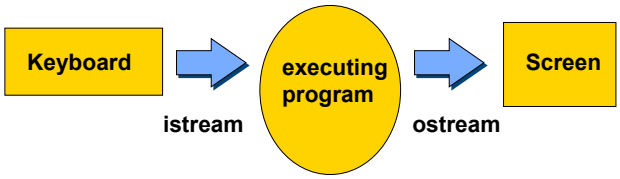
- Input Statements to Read Values into a Program using >>, and functions get, ignore, getline
- Prompting for Interactive Input/Output (I/O)
- Using Data Files for Input and Output

Chapter 4 Topics

- Object-Oriented Design Principles
- Functional Decomposition Methodology
- Software Engineering Tip Documentation

C++ Input/Output

- No built-in I/O in C++
- A library provides input stream and output stream



<iostream> Header File

Access to a library that defines 3 objects

- An istream object named cin (keyboard)
- An ostream object named cout (screen)
- An ostream object named cerr (screen)

Giving a Value to a Variable

In your program you can assign (give) a value to the variable by using the assignment operator =

```
ageOfDog = 12;
```

or by another method, such as

```
cout << "How old is your dog?";
cin  >> ageOfDog;
```

>> Operator

>> is called the input or extraction operator

>> is a binary operator

>> is left associative

Expression	Has value
`cin >> age`	`cin`

Statement

```
cin >> age >> weight;
```


Extraction Operator (>>)

- Variable cin is predefined to denote an input stream from the standard input device (the keyboard)
- The extraction operator >> called "get from" takes 2 operands; the left operand is a stream expression, such as cin--the right operand is a variable of simple type

Extraction Operator (>>)

- Operator >> attempts to extract (inputs) the next item from the input stream and to store its value in the right operand variable
- >> "skips over" (actually reads but does not store anywhere) leading white space characters as it reads your data from the input stream(either keyboard or disk file)

Input Statements

SYNTAX

cin >> *Variable* >> *Variable* . . .;

These examples yield the same result.

```
cin >> length;
cin >> width;
```

```
cin >> length >> width;
```

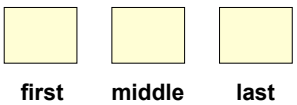

Whitespace Characters Include . . .

- blanks
- tabs
- end-of-line (newline) characters
- newline character created by:
  - hitting Enter or Return at the keyboard or
  - by using the manipulator endl or by using the symbols "\n" in the program

At keyboard you type:

`A[space]B[space]C[Enter]`

```
char   first;
char   middle;
char   last;
```



first middle last

```
cin  >>   first ;
cin  >>   middle ;
cin  >>   last ;
```

first middle last

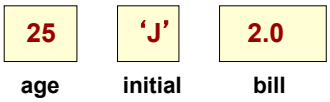
NOTE: A file reading marker is left pointing to the newline character after the 'C' in the input stream

At keyboard you type:

[space]25[space]J[space]2[Enter]

```
int     age;
char  initial;
float   bill;
```

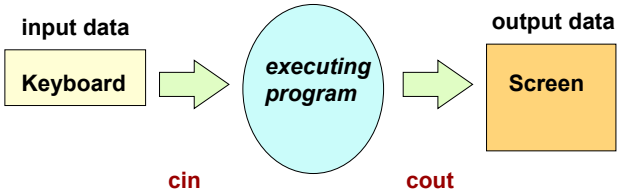

```
cin  >>  age;
cin  >>  initial;
cin  >> bill;
```



NOTE: A file reading marker is left pointing to the newline character after the 2 in the input stream

Keyboard and Screen I/O

```
#include <iostream>
```



(of type istream) (of type ostream)

Another example using >>

NOTE: ▯ shows the location of the file reading marker

STATEMENTS	CONTENTS	MARKER POSITION
int i; char ch; float x; cin >> i;	i: ▯ ch: ▯ x: ▯	25 A\n 16.9\n
	i: 25 ch: ▯ x: ▯	25 A\n 16.9\n
cin >> ch;	i: 25 ch: 'A' x: ▯	25 A\n 16.9\n
cin >> x;	i: 25 ch: 'A' x: 16.9	25 A\n 16.9\n

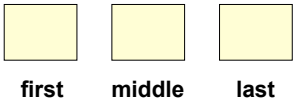
Another Way to Read char Data

- The **`get()`** function can be used to read a single character.
- **`get()`** obtains the very next character from the input stream without skipping any leading whitespace characters

At keyboard you type:

A[space]B[space]C[Enter]

```
char  first;
char  middle;
char  last;
```




```
cin.get(first);
cin.get(middle);
cin.get(last);
```

NOTE: The file reading marker is left pointing to the space after the 'B' in the input stream

Use function ignore() to skip characters

The **ignore()** function is used to skip (read and discard) characters in the input stream

The call:

```
cin.ignore(howMany, whatChar);
```

will skip over up to **howMany** characters or until **whatChar** has been read, whichever comes first

An Example Using cin.ignore()

NOTE: ▯ shows the location of the file reading marker

STATEMENTS	CONTENTS	MARKER POSITION
int a; int b; int c;	a b c	957 34 1235\n 128 96\n
cin >> a >> b;	a: 957, b: 34, c:	957 34 1235\n 128 96\n
cin.ignore(100, '\n');	a: 957, b: 34, c:	957 34 1235\n 128 96\n
cin >> c;	a: 957, b: 34, c: 128	957 34 1235\n 128 96\n

Another Example Using cin.ignore()

NOTE: ▯ shows the location of the file reading marker

STATEMENTS	CONTENTS	MARKER POSITION
int i; char ch;	i ch	A 22 B 16 C 19\n
cin >> ch;	i, ch: 'A'	A 22 B 16 C 19\n
cin.ignore(100, 'B');	i, ch: 'A'	A 22 B 16 C 19\n
cin >> i;	i: 16, ch: 'A'	A 22 B 16 C 19\n

String Input in C++

Input of a string is possible using the extraction operator >>

Example

```
string   message;
cin  >>  message;
cout <<  message;
```

However . . .

>> Operator with Strings

Using the extraction operator(>>) to read input characters into a string variable

- **The >> operator skips any leading whitespace characters such as blanks and newlines**
- **It then reads successive characters into the string**
- **>> operator then stops at the first trailing whitespace character (which is not consumed, but remains waiting in the input stream)**

String Input Using >>

```
string   firstName;
string   lastName;
cin  >>  firstName >> lastName;
```

Suppose input stream looks like this:

▯▯Joe▯Hernandez▯23

What are the string values?

Results Using >>

```
string   firstName;
string   lastName;
cin  >>  firstName >> lastName;
```

Result

"Joe"	"Hernandez"
firstName	lastName

getline() Function

- Because the extraction operator stops reading at the first trailing whitespace, >> cannot be used to input a string with blanks in it
- Use the `getline` function with 2 arguments to overcome this obstacle
- First argument is an input stream variable, and second argument is a string variable

Example

```
string    message;
getline(cin,  message);
```


getline(inFileStream, str)

- `getline` does not skip leading whitespace characters such as blanks and newlines
- `getline` reads successive characters(including blanks) into the string, and stops when it reaches the newline character '\n'
- The newline is consumed by `getline`, but is not stored into the string variable

String Input Using getline

```
string    firstName;
string    lastName;
getline(cin,  firstName);
getline(cin,  lastName);
```

Suppose input stream looks like this:

□□Joe□Hernandez□23

What are the string values?

Results Using getline

```
string    firstName;
string    lastName;
getline(cin,  firstName);
getline(cin,  lastName);
```

" Joe Hernandez 23"	?
firstName	lastName

Interactive I/O

- In an interactive program the user enters information while the program is executing
- Before the user enters data, a prompt should be provided to explain what type of information should be entered
- The amount of information needed in the prompt depends on
  - the complexity of the data being entered, and
  - the sophistication of the person entering the data

Prompting for Interactive I/O

```
// Pattern: cout(prompt) cin(read value)
cout  <<  "Enter part number :"  <<  endl;
cin    >>  partNumber;
cout  <<  "Enter quantity ordered : "  <<
endl;
cin    >>  quantity;
cout  <<  "Enter unit price : " <<  endl;
cin    >>  unitPrice;
// Calculate and print results
```

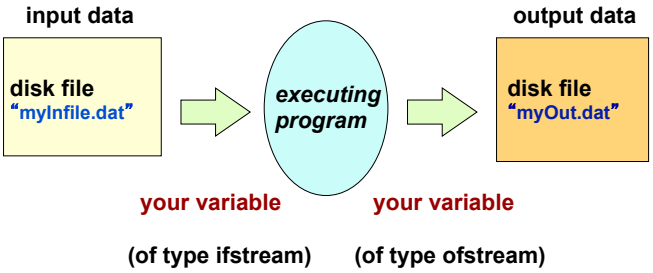
Prompting for Interactive I/O, cont...

```
totalPrice  =  quantity * unitPrice;
cout  <<  "Part # " <<  partNumber  <<  endl;
cout  <<  "Quantity: "  <<  quantity
      <<  endl;
cout  <<  "Unit Cost: $ "  <<  setprecision(2)
      <<  unitPrice  <<  endl;
cout  <<  "Total Cost: $ " <<  totalPrice
      <<  endl;
```


USING DATA FILES FOR INPUT AND OUTPUT

Disk Files for I/O

#include <fstream>



Disk I/O

To use disk I/O

- **Access** #include <fstream>
- **Choose** valid identifiers for your file streams and declare them
- **Open** the files and associate them with disk names

Disk I/O, cont...

- **Use** your file stream identifiers in your I/O statements(using >> and << , manipulators, get, ignore)
- **Close** the files

Disk I/O Statements

```
#include <fstream>

ifstream   myInfile;        // Declarations
ofstream   myOutfile;

myInfile.open("myIn.dat"); // Open files
myOutfile.open("myOut.dat");
// Verify that they are open
myInfile.close();           // Close files
myOutfile.close();
```

Opening a File

Opening a file

- Associates the C++ identifier for your file with the physical(disk) name for the file
  - If the input file does not exist on disk, open is not successful
  - If the output file does not exist on disk, a new file with that name is created
  - If the output file already exists, it is erased

Opening a File

Opening a file

- Places a file reading marker at the very beginning of the file, pointing to the first character in the file

Stream Fail State

- When a stream enters the fail state,
  - Further I/O operations using that stream have no effect at all
  - The computer does not automatically halt the program or give any error message

Stream Fail State

- Possible reasons for entering fail state include:
  - Invalid input data (often the wrong type)
  - Opening an input file that doesn't exist
  - Opening an output file on a disk that is already full or is write-protected

Run Time File Name Entry

```
#include <string>
// Contains conversion function c_str

ifstream  inFile;
string    fileName;

// Prompt:
cout << "Enter input file name: " << endl;
cin  >>  fileName;

// Convert string fileName to a C string type
inFile.open(fileName.c_str());
```


FUNCTIONAL DECOMPOSITION

Functional Decomposition

- A technique for developing a program in which the problem is divided into more easily handled subproblems
- The solutions of these subproblems create a solution to the overall problem

Functional Decomposition

In functional decomposition, we work from the abstract (a list of the major steps in our solution) to the particular (algorithmic steps that can be translated directly into code in C++ or another language)

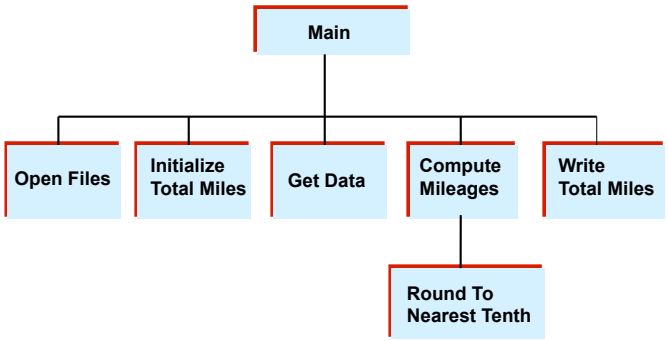
Functional Decomposition

- Focus is on actions and algorithms
- Begins by breaking the solution into a series of major steps; process continues until each subproblem cannot be divided further or has an obvious solution

Functional Decomposition

- Units are *modules* representing algorithms
  - A module is a collection of concrete and abstract steps that solves a subproblem
  - A module structure chart (hierarchical solution tree) is often created
- Data plays a secondary role in support of actions to be performed

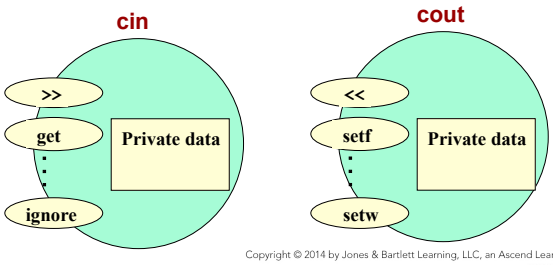
Module Structure Chart



OBJECT-ORIENTED DESIGN

Object-Oriented Design

A technique for developing a program in which the solution is expressed in terms of objects -- self-contained entities composed of data and operations on that data



More about OOD

- Languages supporting OOD include: C++, Java, Smalltalk, Eiffel, CLOS, and Object-Pascal
- A ***class*** is a programmer-defined data type and objects are variables of that type

More about OOD

- In C++, **cin** is an object of a data type (class) named istream, and **cout** is an object of a class ostream.
- Header files iostream and fstream contain definitions of stream classes
- A class generally contains **private** data and **public** operations (called ***member functions***)

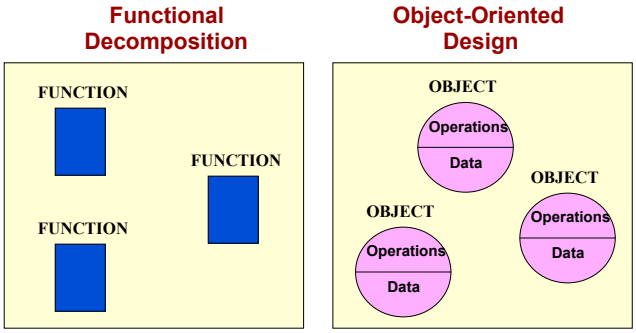
Object-Oriented Design (OOD)

- **Focus** is on entities called objects and operations on those objects, all bundled together
- **Begins** by identifying the major objects in the problem, and choosing appropriate operations on those objects

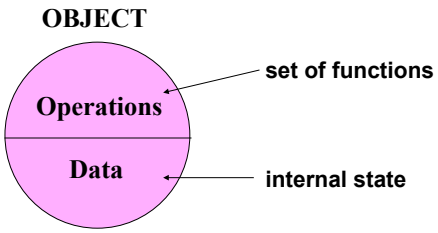
Object-Oriented Design (OOD)

- **Units** are ***objects***; programs are collections of objects that communicate with each other
- **Data** plays a leading role; algorithms are used to implement operations on the objects and to enable object interaction

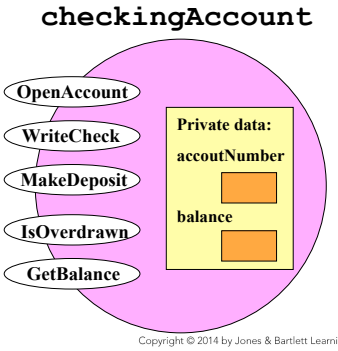
Two Programming Methodologies



What is an object?



An object contains data and operations



OOD Used with Large Software Projects

- Objects within a program often model real-life objects in the problem to be solved
- Many libraries of pre-written classes and objects are available as-is for re-use in various programs

OOD Used with Large Software Projects

- The OOD concept of inheritance allows the customization of an existing class to meet particular needs without having to inspect and modify the source code for that class
- This can reduce the time and effort needed to design, implement, and maintain large systems

SOFTWARE ENGINEERING

Software Engineering Tip Documentation

- Documentation includes the written problem specification, design, development history, and actual code of a problem
- Good documentation helps other programmers read and understand a program
- Good documentation invaluable when software is being debugged and modified (maintained)

Software Engineering Tip Documentation

- Documentation is both external and internal to the program
- External documentation includes the specifications, development history, and the design documents
- Internal documents includes the program format and self-documenting code--meaningful identifiers and comments

Software Engineering Tip Documentation

- Comments in your programs may be sufficient for someone reading or maintaining your programs
- However, if the program is to be used by non-programmers, then you must also provide a user's manual
- Keep documentation up-to-date and indicate any changes you made in pertinent documentation

CASE STUDY

Names in Multiple Formats

Problem

You are beginning to work on a problem that needs to output names in several formats along with the corresponding social security number.

As a start, you decide to write a short C++ program that inputs a social security number and a single name and displays it in the different formats, so you can be certain that all of your string expressions are correct.

Algorithm

Main Module **Level 0**

Open files
Get social security number
Get name
Write data in proper formats
Close files

Open Files **Level 1**

inData.open("name.dat")
outData.open("name.out")

Get Name

Get first name
Get middle name or initial
Get last name

Write Data in Proper Formats

Write first name, blank, middle name, blank, last name, blank, social security number
Write last name, comma, first name, blank, middle name, blank, social security number
Write last name, comma, blank, first name, blank, middle initial, period, blank, social security number
Write first name, blank, middle initial, period, blank, last name

Middle initial — Level 2

Set initial to middleName.substr(0, 1) + period

Close files

inData.close()
outData.close()

C++ Program

```
//************************************************************
// Format Names program
// This program reads in a social security number, a first name
// a middle name or initial, and a last name from file inData.
// The name is written to file outData in three formats:
//    1. First name, middle name, last name, and social security
//    number.
//    2. last name, first name, middle name, and social
//    security number
//    3. last name, first name, middle initial, and social
//    security number
//    4.  First name, middle initial, last name
//************************************************************
```



```
#include <fstream>          // Access ofstream
#include <string>           // Access string
using namespace std;

int main()
{
    // Declare and open files
    ifstream inData;
    ofstream outData;
    inData.open("name.dat");
    outData.open("name.out");
```



```
// Declare variables
    string socialNum;      // Social security number
    string firstName;      // First name
    string lastName;            // Last name
    string middleName;     // Middle name
    string initial;        // Middle initial
```



```
// Read in data from file inData
inData >> socialNum >> firstName >> middleName
       >> lastName;
// Access middle initial and append a period
initial = middleName.substr(0, 1) + '.';
```

```
// Output information in required formats
    outData << firstName << ' ' << middleName << ' '
            << lastName << ' ' << socialNum << endl;
    outData << lastName << ", " << firstName << ' '
            << middleName << ' ' << socialNum  << endl;
    outData << lastName << ", " << firstName << ' '
            << initial << ' ' << socialNum  << endl;
    outData << firstName << ' ' << initial << ' '
            << lastName;
```



```
    // Close files
       inData.close();
       outData.close();
       return 0;


}
```