Chapter 3

Numeric Types, Expressions, and Output
Chapter 3 Topics

- Constants of Type int and float
- Evaluating Arithmetic Expressions
- Implicit Type Coercion and Explicit Type Conversion
- Calling a Value-Returning Function
- Using Function Arguments
- Using C++ Library Functions in Expressions
- Calling a Void Function
- C++ Manipulators to Format Output
- String Operations: length, find, and substr
length Function

- Function **length** returns an unsigned integer value that equals the number of characters currently in the string.

- Function **size** returns the same value as function **length**.

- You must use **dot notation** in the call to function **length** or **size**.
**find Function**

- Function **find** returns an unsigned integer value that is the beginning position for the first occurrence of a particular substring within the string.

- The **substring** argument can be a string constant, a string expression, or a char value.

- If the **substring** was not found, function **find** returns the special value `string::npos`.
**substr Function**

- Function `substr` returns a particular substring of a string

- The first argument is an unsigned integer that specifies a **starting position** within the string

- The second argument is an unsigned integer that specifies the **length** of the desired substring

- **Positions** of characters within a string are numbered starting from 0, not from 1
Chapter 4

Program Input and the Software Design Process

Dale/Weems
Chapter 4 Topics

- Input Statements to Read Values into a Program using `>>`, and functions `get`, `ignore`, `getline`
- Prompting for Interactive Input/Output
- Using Data Files for Input and Output
- Object-Oriented Design Principles
- Functional Decomposition Methodology
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 =

```cpp
ageOfDog = 12;
```

or by another method, such as

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

>> is called the input or extraction operator
>> is a binary operator
>> is left associative

Expression

\[
\text{cin} \ >> \ \text{age}
\]

Has value

\[
\text{cin}
\]

Statement

\[
\text{cin} \ >> \ \text{age} \ >> \ \text{weight};
\]
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.
- Operator `>>` attempts to extract the next item from the input stream and to store its value in the right operand variable.
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

The newline character is created by hitting Enter or Return at the keyboard, or by using the manipulator endl or “\n” in a program.
Extraction Operator >>

>> “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)
At keyboard you type:

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

char first;
char middle;
char last;

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

‘A’ ‘B’ ‘C’

NOTE: A file reading marker is left pointing to the newline character after the ‘C’ in the input stream
At keyboard you type:

```c
[space] 25 [space] J [space] 2 [Enter]
```

```c
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>

input data

Keyboard

executing program

output data

Screen

(cin (of type istream))

(cout (of type ostream))
Another example using `>>`

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i;</td>
<td></td>
<td>25 A\n</td>
</tr>
<tr>
<td>char ch;</td>
<td></td>
<td>16.9\n</td>
</tr>
<tr>
<td>float x;</td>
<td></td>
<td>16.9\n</td>
</tr>
<tr>
<td>cin &gt;&gt; i;</td>
<td></td>
<td>25 A\n</td>
</tr>
<tr>
<td></td>
<td>i</td>
<td>16.9\n</td>
</tr>
<tr>
<td>cin &gt;&gt; ch;</td>
<td></td>
<td>25 A\n</td>
</tr>
<tr>
<td></td>
<td>'A'</td>
<td>16.9\n</td>
</tr>
<tr>
<td>cin &gt;&gt; x;</td>
<td></td>
<td>25 A\n</td>
</tr>
<tr>
<td></td>
<td>'A'</td>
<td>16.9\n</td>
</tr>
<tr>
<td></td>
<td>16.9</td>
<td>16.9\n</td>
</tr>
</tbody>
</table>

NOTE: □ shows the location of the file reading marker.
String Input in C++

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

Example

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

However . . .
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, and **stops at the first trailing whitespace** character (which is not consumed, but remains waiting in the input stream)
String Input Using >>

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

Suppose input stream looks like this:

```
Joe Hernandez 23
```

What are the string values?
string firstName;
string lastName;
cin >> firstName >> lastName;

Result

"Joe"
firstName

"Hernandez"
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

```cpp
string message;
ggetline(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`

```cpp
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`

```cpp
string firstName;
string lastName;
ggetline(cin, firstName);
ggetline(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.
// 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
totalPrice = quantity * unitPrice;
cout << "Part # " << partNumber << endl;
cout << "Quantity: " << quantity << endl;
cout << "Unit Cost: $ " << setprecision(2) << unitPrice << endl;
cout << "Total Cost: $ " << totalPrice << endl;
Disk Files for I/O

#include <fstream>

input data

disk file “myInfile.dat”

executing program

your variable
(of type ifstream)

output data

disk file “myOut.dat”

your variable
(of type ofstream)
Disk I/O

To use disk I/O

- **Access** include `<fstream>`
- **Choose** valid identifiers for your filestreams and declare them
- **Open** the files and associate them with disk names
- **Use** your filestream identifiers in your I/O statements (using `>>` and `<<`, manipulators, `get`, `ignore`)
- **Close** the files
#include <fstream>

ifstream myInfile;       // Declarations
ofstream myOutfile;

myInfile.open(“myIn.dat”);  // Open files
myOutfile.open(“myOut.dat”);

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

- **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
- **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


#include <string>
// Contains conversion function c_str

ifstream inFile;
string fileName;

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

// Convert string fileName to a C string type
inFile.open(fileName.c_str());
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
- Open files
- Get social security number
- Get name
- Write data in proper formats
- Close files

Open Files
- 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
Set initial to middleName.substr(0, 1) + period
Close files
inData.close()
outData.close()
C++ Program

```c++
// 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
```

//*******************************************************************************

```c++
// 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;