Chapter 3: Input/Output

I/O Streams and Standard I/O Devices

- I/O: sequence of bytes (stream of bytes) from source to destination
  - Bytes are usually characters, unless program requires other types of information
  - Stream: sequence of characters from source to destination
  - Input stream: sequence of characters from an input device to the computer
  - Output stream: sequence of characters from the computer to an output device

I/O Streams and Standard I/O Devices (cont'd.)

- Use `iostream` header file to receive data from keyboard and send output to the screen
  - Contains definitions of two data types:
    - `istream`: input stream
    - `ostream`: output stream
  - Has two variables:
    - `cin`: stands for common input
    - `cout`: stands for common output

I/O Streams and Standard I/O Devices (cont'd.)

- Variable declaration is similar to:
  - `istream cin;`
  - `ostream cout;`
- To use `cin` and `cout`, the preprocessor directive `#include <iostream>` must be used
- Input stream variables: type `istream`
- Output stream variables: type `ostream`

cin and the Extraction Operator >>

- The syntax of an input statement using `cin` and the extraction operator >> is:

```
cin >> variable >> variable...;
```

- The extraction operator >> is binary
  - Left-side operand is an input stream variable
    - Example: `cin`
  - Right-side operand is a variable

cin and the Extraction Operator >> (cont'd.)

- No difference between a single `cin` with multiple variables and multiple `cin` statements with one variable
- When scanning, `>>` skips all whitespace
  - Blanks and certain nonprintable characters
- `>>` distinguishes between character `2` and number `2` by the right-side operand of `>>`
  - If type `char` or `int` (or `double`), the `2` is treated as a character or as a number `2`

cin and the Extraction Operator >> (cont'd.)

TABLE 3-1 Valid Input for a Variable of the Simple Data Type

Data Type of a	Valid Input for a
char	One printable character except the blank
int	An integer, possibly preceded by a + or – sign
double	A decimal number, possibly preceded by a + or – sign. If the actual data input is an integer, the input is converted to a decimal number with the zero decimal part.

- Entering a `char` value into an `int` or `double` variable causes serious errors, called input failure

`cin` and the Extraction Operator >> (cont'd.)

- When reading data into a `char` variable
  - `>>` skips leading whitespace, finds and stores only the next character
  - Reading stops after a single character
- To read data into an `int` or `double` variable
  - `>>` skips leading whitespace, reads + or - sign (if any), reads the digits (including decimal)
  - Reading stops on whitespace non-digit character

cin and the Extraction Operator >> (cont'd.)

EXAMPLE 3-1

Suppose you have the following variable declarations:

```
int a, b;
double z;
char ch;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> ch;`	A	`ch = 'A'`
2	`cin >> ch;`	AB	`ch = 'A'`, `'B'` is held for later input
3	`cin >> a;`	48	`a = 48`
4	`cin >> a;`	46.35	`a = 46`, `.35` is held for later input
5	`cin >> z;`	74.35	`z = 74.35`
6	`cin >> z;`	39	`z = 39.0`
7	`cin >> z >> a;`	65.78 38	`z = 65.78`, `a = 38`
8	`cin >> a >> b;`	4 60	`a = 4`, `b = 60`
9	`cin >> a >> z;`	46 32.4 68	`a = 46`, `z = 32.4`, `68` is held for later input

cin and the Extraction Operator >> (cont'd.)

EXAMPLE 3-2

Suppose you have the following variable declarations:

```
int a;
double z;
char ch;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> a >> ch >> z;`	`57 A 26.9`	`a = 57, ch = 'A', z = 26.9`
2	`cin >> a >> ch >> z;`	`57 A` `26.9`	`a = 57, ch = 'A', z = 26.9`

3	`cin >> a >> ch >> z;`	`57`
		`A`
		`26.9`
			`a = 57, ch = 'A', z = 26.9`

4	`cin >> a >> ch >> z;`	`57A26.9`	`a = 57, ch = 'A', z = 26.9`

cin and the Extraction Operator >> (cont'd.)

EXAMPLE 3-3

Suppose you have the following variable declarations:

```
int a, b;
double z;
char ch, ch1, ch2;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> z >> ch >> a;`	`36.78B34`	`z = 36.78, ch = 'B', a = 34`
2	`cin >> z >> ch >> a;`	`36.78` `B34`	`z = 36.78, ch = 'B', a = 34`
3	`cin >> a >> b >> z;`	`11 34`	`a = 11, b = 34,` computer waits for the next number
4	`cin >> a >> z;`	`78.49`	`a = 78, z = 0.49`
5	`cin >> ch >> a;`	`256`	`ch = '2', a = 56`
6	`cin >> a >> ch;`	`256`	`a = 256,` computer waits for the input value for `ch`
7	`cin >> ch1 >> ch2;`	`A B`	`ch1 = 'A', ch2 = 'B'`

Using Predefined Functions in a Program

- Function (subprogram): set of instructions
  - When activated, it accomplishes a task
- `main` executes when a program is run
- Other functions execute only when called
- C++ includes a wealth of functions
  - Predefined functions are organized as a collection of libraries called header files

Using Predefined Functions in a Program (cont'd.)

- Header file may contain several functions
- To use a predefined function, you need the name of the appropriate header file
  - You also need to know:
    - Function name
    - Number of parameters required
    - Type of each parameter
    - What the function is going to do

Using Predefined Functions in a Program (cont'd.)

- To use `pow` (power), include `cmath`
  - Two numeric parameters
  - Syntax: `pow(x,y)` = x^y
    - `x` and `y` are the arguments or parameters
  - In `pow(2,3)`, the parameters are `2` and `3`

cin and the get Function

- The get function
  - Inputs next character (including whitespace)
  - Stores in memory location indicated by its argument
- The syntax of cin and the get function:

```
cin.get(varChar);
```

- varChar
  - Is a char variable
  - Is the argument (or parameter) of the function

cin and the ignore Function

- ignore function
  - Discards a portion of the input
- The syntax to use the function ignore is:

```
cin.ignore(intExp, chExp);
```

  - intExp is an integer expression
  - chExp is a char expression
- If intExp is a value m, the statement says to ignore the next m characters or all characters until the character specified by chExp

cin and the ignore Function (cont'd.)

EXAMPLE 3-5

Consider the declaration:

```
int a, b;
```

and the input:

```
25 67 89 43 72
12 78 34
```

Now consider the following statements:

```
cin >> a;
cin.ignore(100, '\n');
cin >> b;
```

The first statement, `cin >> a;`, stores `25` in `a`. The second statement, `cin.ignore(100, '\n');`, discards all of the remaining numbers in the first line. The third statement, `cin >> b;`, stores `12` (from the next line) in `b`.

putback and peek Functions

- `putback` function
  - Places previous character extracted by the `get` function from an input stream back to that stream
- `peek` function
  - Returns next character from the input stream
  - Does not remove the character from that stream

putback and peek Functions (cont'd.)

- The syntax for `putback`:

```
istreamVar.putback(ch);
```

  - `istreamVar`: an input stream variable (`cin`)
  - `ch` is a `char` variable

- The syntax for `peek`:

```
ch = istreamVar.peek();
```

  - `istreamVar`: an input stream variable (`cin`)
  - `ch` is a `char` variable

The Dot Notation Between I/O Stream Variables and I/O Functions

- A precaution
  - In the statement

    ```
    cin.get(ch);
    ```

    `cin` and `get` are two separate identifiers separated by a dot
  - Dot separates the input stream variable name from the member, or function, name
  - In C++, dot is the member access operator

Input Failure

- Things can go wrong during execution
- If input data does not match corresponding variables, program may run into problems
- Trying to read a letter into an `int` or `double` variable will result in an input failure
- If an error occurs when reading data
  - Input stream enters the fail state

The `clear` Function

- Once in a fail state, all further I/O statements using that stream are ignored
- The program continues to execute with whatever values are stored in variables
  - This causes incorrect results
- The `clear` function restores input stream to a working state

```
istreamVar.clear();
```

Output and Formatting Output

- Syntax of `cout` when used with `<<`

```
cout << expression or manipulator << expression or manipulator...;
```

- `expression` is evaluated
- `value` is printed
- `manipulator` is used to format the output
  - Example: `endl`

setprecision Manipulator

- Syntax:

```
setprecision(n)
```

- Outputs decimal numbers with up to `n` decimal places
- Must include the header file `iomanip`:
  - `#include <iomanip>`

`fixed` Manipulator

- `fixed` outputs floating-point numbers in a fixed decimal format
  - Example: `cout << fixed;`
  - Disable by using the stream member function `unsetf`
    - Example: `cout.unsetf(ios::fixed);`
- `scientific` manipulator: outputs floating-point numbers in scientific format

`showpoint` Manipulator

- `showpoint` forces output to show the decimal point and trailing zeros
- Examples:
  - `cout << showpoint;`
  - `cout << fixed << showpoint;`

setw

- Outputs the value of an expression in a specified number of columns
  - `cout << setw(5) << x << endl;`
- If number of columns exceeds the number of columns required by the expression
  - Output of the expression is right-justified
  - Unused columns to the left are filled with spaces
- Must include the header file `iomanip`

Additional Output Formatting Tools

- Additional formatting tools that give you more control over your output:
  - `setfill` manipulator
  - `left` and `right` manipulators
  - `unsetf` manipulator

`setfill` Manipulator

- Output stream variables can use `setfill` to fill unused columns with a character

```
ostreamVar << setfill(ch);
```

- Example:
  - `cout << setfill('#');`

`left` and `right` Manipulators

- `left`: left-justifies the output

```
ostreamVar << left;
```

- Disable `left` by using `unsetf`

```
ostreamVar.unsetf(ios::left);
```

- `right`: right-justifies the output

```
ostreamVar << right;
```

Types of Manipulators

- Two types of manipulators:
  - With parameters
  - Without parameters
- Parameterized: require `iomanip` header
  - `setprecision`, `setw`, and `setfill`
- Nonparameterized: require `iostream` header
  - `endl`, `fixed`, `showpoint`, `left`, and `flush`

Input/Output and the `string` Type

- An input stream variable (`cin`) and `>>` operator can read a string into a variable of the data type `string`
- Extraction operator
  - Skips any leading whitespace characters
  - Reading stops at a whitespace character
- The function `getline`
  - Reads until end of the current line

```
getline(istreamVar, strVar);
```

Debugging: Understanding Logic Errors and Debugging with `cout` statements

- Syntax errors
  - Reported by the compiler
- Logic errors
  - Typically not caught by the compiler
  - Spot and correct using `cout` statements
  - Temporarily insert an output statement
  - Correct problem
  - Remove output statement

File Input/Output

- File: area in secondary storage to hold info
- File I/O is a five-step process
  1. Include `fstream` header
  2. Declare file stream variables
  3. Associate the file stream variables with the input/output sources
  4. Use the file stream variables with >>, <<, or other input/output functions
  5. Close the files

Summary

- Stream: infinite sequence of characters from a source to a destination
  - Input stream: from a source to a computer
  - Output stream: from a computer to a destination
  - `cin`: common input
  - `cout`: common output
  - To use `cin` and `cout`, include iostream header

Summary (cont'd.)

- `get` reads data character-by-character
- `ignore` skips data in a line
- `putback` puts last character retrieved by `get` back to the input stream
- `peek` returns next character from input stream, but does not remove it
- Attempting to read invalid data into a variable causes the input stream to enter the fail state

Summary (cont'd.)

- The manipulators `setprecision`, `fixed`, `showpoint`, `setw`, `setfill`, `left`, and `right` can be used for formatting output
- Include `iomanip` for the manipulators `setprecision`, `setw`, and `setfill`
- Header `fstream` contains the definitions of `ifstream` and `ofstream`