# Chapter 3

## Input/Output

# Objectives

In this chapter, you will:

- Learn what a stream is and examine input and output streams
- Explore how to read data from the standard input device
- Learn how to use predefined functions in a program
- Explore how to use the input stream functions `get`, `ignore`, `putback`, and `peek`

# Objectives (continued)

- Become familiar with input failure
- Learn how to write data to the standard output device
- Discover how to use manipulators in a program to format output
- Learn how to perform input and output operations with the `string` data type
- Become familiar with file input and 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 (continued)

- Use `iostream` header file to extract (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 (continued)

- To use `cin` and `cout`, the preprocessor directive `#include <iostream>` must be used
- Variable declaration is similar to:
  - `istream cin;`
  - `ostream cout;`
- 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 >> (continued)

- 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 >> (continued)

**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 >> (continued)

- 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 >> (continued)

**EXAMPLE 3-1**

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

| | 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 >> ch >> z;` | `57 A 26.9` | `a = 57, ch = 'A', z = 26.9` |
| 10 | `cin >> a >> ch >> z;` | `57 A 26.9` | `a = 57, ch = 'A', z = 26.9` |

## EXAMPLE 3-1

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

| | | | |
|---|---|---|---|
| 11 | `cin >> a >> ch >> z;` | 57<br>A<br>26.9 | a = 57, ch = 'A',<br>z = 26.9 |
| 12 | `cin >> a >> ch >> z;` | 57A26.9 | a = 57, ch = 'A',<br>z = 26.9 |
| 13 | `cin >> z >> ch >> a;` | 36.78B34 | z = 36.78, ch = 'B',<br>a = 34 |
| 14 | `cin >> z >> ch >> a;` | 36.78<br>B34 | z = 36.78, ch = 'B',<br>a = 34 |
| 15 | `cin >> a >> b >> z;` | 11 34 | a = 11, b = 34,<br>computer waits for the next number |
| 16 | `cin >> a >> z;` | 46 32.4 68 | a = 46, z = 32.4, 68 is held for later input |
| 17 | `cin >> a >> z;` | 78.49 | a = 78, z = 0.49 |
| 18 | `cin >> ch >> a;` | 256 | ch = '2', a = 56 |
| 19 | `cin >> a >> ch;` | 256 | a = 256, computer waits for the input value for ch |
| 20 | `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 (continued)

- 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 (continued)

- 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`

## EXAMPLE 3-2

```
// How to use predefined functions.
#include <iostream>
#include <cmath>
#include <string>

using namespace std;

int main()
{
    double u, v;
    string str;

    cout << "Line 1: 2 to the power of 6 = "
         << pow(2, 6) << endl;                          //Line 1

    u = 12.5;                                           //Line 2
    v = 3.0;                                            //Line 3
    cout << "Line 4: " << u << " to the power of "
         << v << " = " << pow(u, v) << endl;            //Line 4

    cout << "Line 5: Square root of 24 = "
         << sqrt(24.0) << endl;                         //Line 5

    u = pow(8.0, 2.5);                                  //Line 6
    cout << "Line 7: u = " << u << endl;                //Line 7

    str = "Programming with C++";                       //Line 8

    cout << "Line 9: Length of str = "
         << str.length() << endl;                       //Line 9

    return 0;
}
```

# Using Predefined Functions in a Program (continued)

**Sample Run**:

```
Line 1: 2 to the power of 6 = 64
Line 4: 12.5 to the power of 3 = 1953.13
Line 5: Square root of 24 = 4.89898
Line 7: u = 181.019
Line 9: Length of str = 20
```

# `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 (parameter) of the function

# Example: **cin.get()**

```
char ch1, ch2;
int num;
```

and the input:

```
A 25
```

Now consider the following statement:

```
cin >> ch1 >> ch2 >> num;
```

When the computer executes this statement, `'A'` is stored in `ch1`, the blank is skipped by the extraction operator `>>`, the character `'2'` is stored in `ch2`, and `5` is stored in `num`. However, what if you intended to store `'A'` in `ch1`, the blank in `ch2`, and `25` in `num`? It is clear that you cannot use the extraction operator `>>` to input this data.

# cin and the ignore Function

- ignore: 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

# Example: **cin.ignore()**

Consider the declaration:

```
char ch1, ch2;
```

and the input:

```
Hello there. My name is Mickey.
```

a. Consider the following statements:

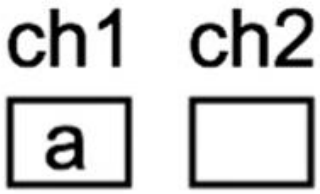
```
cin >> ch1;
cin.ignore(100, '.');
cin >> ch2;
```

The first statement, `cin >> ch1;`, stores `'H'` in `ch1`. The second statement, `cin.ignore(100, '.');`, results in discarding all characters until . (period). The third statement, `cin >> ch2;`, stores the character `'M'` (from the same line) in `ch2`. (Remember that the extraction operator >> skips all leading whitespace characters. Thus, the extraction operator skips the space after . [period] and stores `'M'` in `ch2`.)

# `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

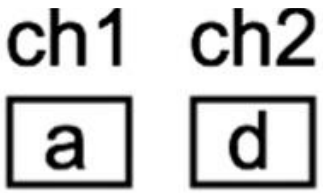
# How Peak & Putback Works!





```
char ch1, ch2;
ch1 = cin.peek();
cin.get(ch2);
cin.putback('d');
cin.get(ch2);
```

# How Peak & Putback Works!





```
char ch1, ch2;
ch1 = cin.peek();
cin.get(ch2);
cin.putback('d');
cin.get(ch2);
```

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

- 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();
```

# `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);`
- The manipulator `scientific` is used to output 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 specific 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 and reading stops at a whitespace character
- The function `getline`
  - Reads until end of the current line

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

# 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

# Programming Example: Movie Ticket Sale and Donation to Charity

- A theater owner agrees to donate a portion of gross ticket sales to a charity
- The program will prompt the user to input:
  - Movie name
  - Adult ticket price
  - Child ticket price
  - Number of adult tickets sold
  - Number of child tickets sold
  - Percentage of gross amount to be donated

# Programming Example: I/O

- Inputs: movie name, adult and child ticket price, # adult and child tickets sold, and percentage of the gross to be donated
- Program output:

```
_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
Movie Name: ..... Journey to Mars
Number of Tickets Sold: .....      2650
Gross Amount: ..... $ 9150.00
Percentage of Gross Amount Donated:     10.00%
Amount Donated: ..... $  915.00
Net Sale: ..... $ 8235.00
```

# Programming Example: Problem Analysis

- The program needs to:
  1. Get the movie name
  2. Get the price of an adult ticket price
  3. Get the price of a child ticket price
  4. Get the number of adult tickets sold
  5. Get the number of child tickets sold

# Programming Example: Problem Analysis (continued)

6. Calculate the gross amount

```
grossAmount = adultTicketPrice *
noOfAdultTicketsSold + childTicketPrice *
noOfChildTicketsSold;
```

7. Calculate the amount donated to the charity

```
amountDonated = grossAmount *
percentDonation / 100;
```

8. Calculate the net sale amount

```
netSale = grossAmount – amountDonated;
```

9. Output the results

# Programming Example: Variables

```
string movieName;
double adultTicketPrice;
double childTicketPrice;
int    noOfAdultTicketsSold;
int    noOfChildTicketsSold;
double percentDonation;
double grossAmount;
double amountDonated;
double netSaleAmount;
```

# Programming Example: Formatting Output

- First column is left-justified
  - When printing a value in the first column, use `left`
- Numbers in second column are right-justified
  - Before printing a value in the second column, use `right`
- Use `setfill` to fill the empty space between the first and second columns with dots

# Programming Example: Formatting Output (continued)

- In the lines showing gross amount, amount donated, and net sale amount
  - Use blanks to fill space between the `$` sign and the number
- Before printing the dollar sign
  - Use `setfill` to set the filling character to blank

# Programming Example: Main Algorithm

1. Declare variables
2. Set the output of the floating-point to:
   - Two decimal places
   - Fixed
   - Decimal point and trailing zeros
3. Prompt the user to enter a movie name
4. Input movie name using `getline` because it might contain spaces
5. Prompt user for price of an adult ticket

# Programming Example: Main Algorithm (continued)

6. Input price of an adult ticket
7. Prompt user for price of a child ticket
8. Input price of a child ticket
9. Prompt user for the number of adult tickets sold
10. Input number of adult tickets sold
11. Prompt user for number of child tickets sold
12. Input the number of child tickets sold

# Programming Example: Main Algorithm (continued)

13. Prompt user for percentage of the gross amount donated
14. Input percentage of the gross amount donated
15. Calculate the gross amount
16. Calculate the amount donated
17. Calculate the net sale amount
18. Output the results

# 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 (continued)

- `get` reads data character-by-character
- `putback` puts last character retrieved by `get` back to the input stream
- `ignore` skips data in a line
- `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 (continued)

- 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`
- File: area in secondary storage to hold info
- Header `fstream` contains the definitions of `ifstream` and `ofstream`