# Chapter 3

Input/Output

# I/O Streams and Standard I/O Devices (1 of 3)

- 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 the source to the 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 (2 of 3)

- 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

# cin and the Extraction Operator >> (1 of 7)

- 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

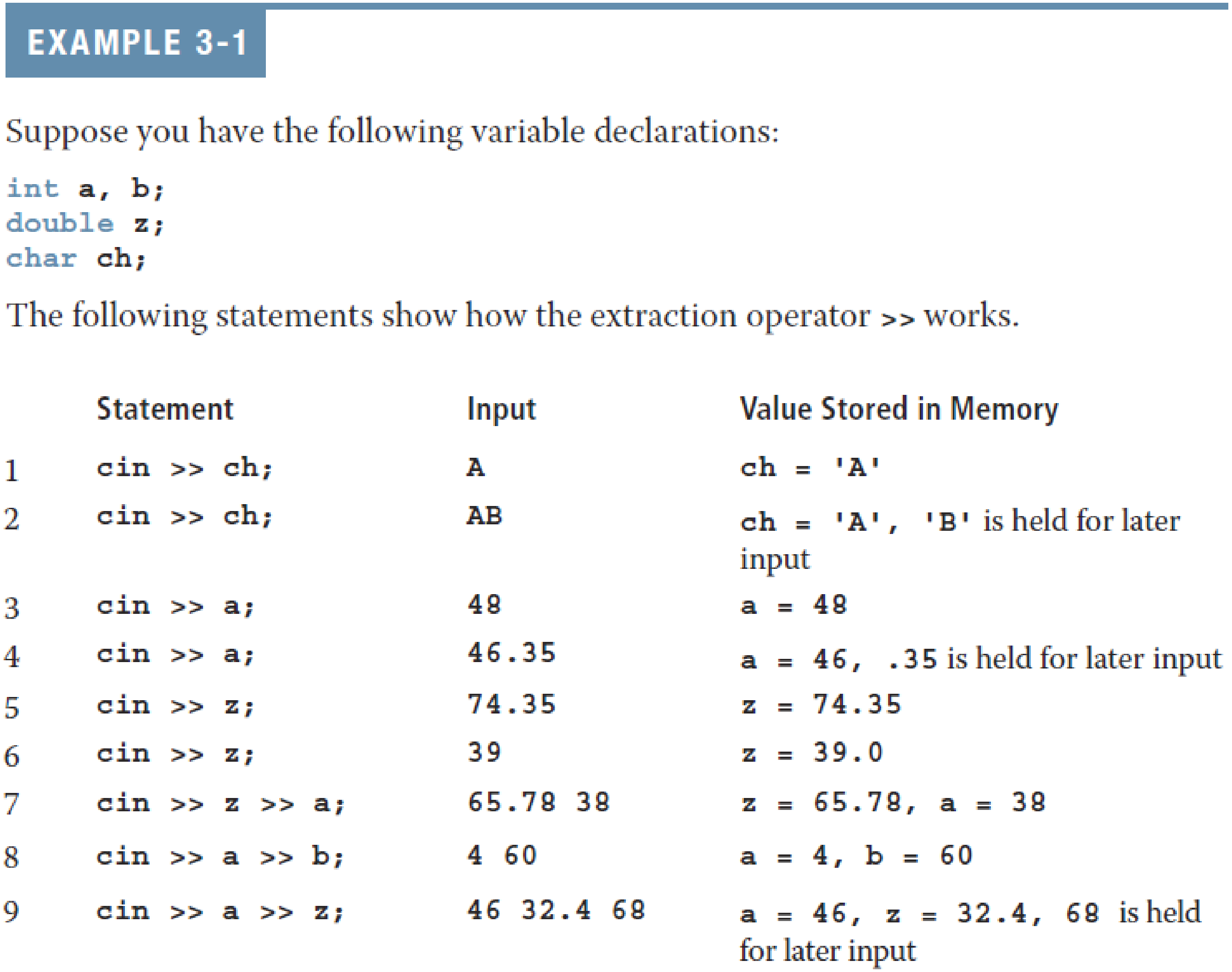
# cin and the Extraction Operator >> (3 of 7)

**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 >> (5 of 7)



## EXAMPLE 3-1

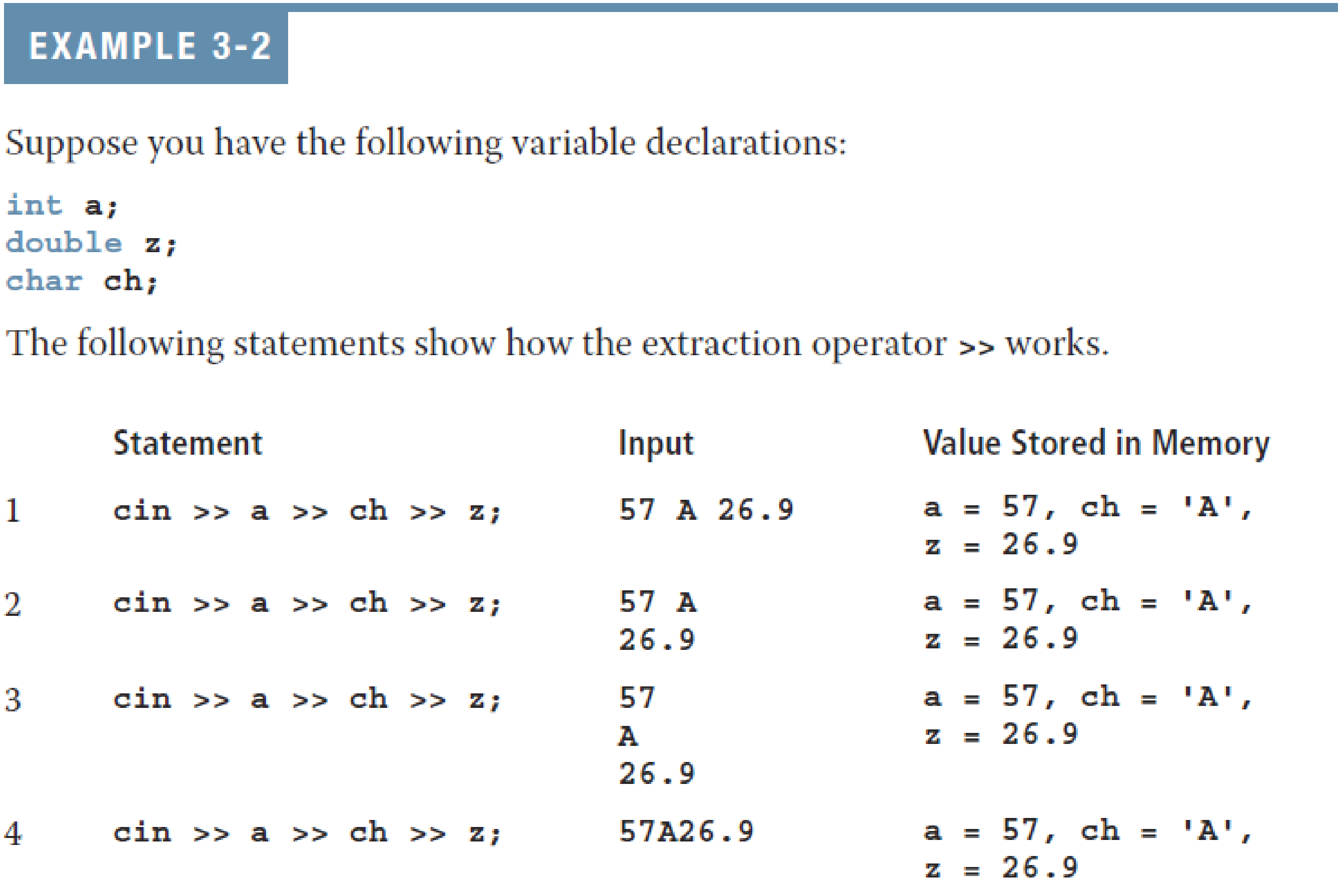
Suppose you have the following variable declarations:

```
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 >> (6 of 7)



**EXAMPLE 3-2**

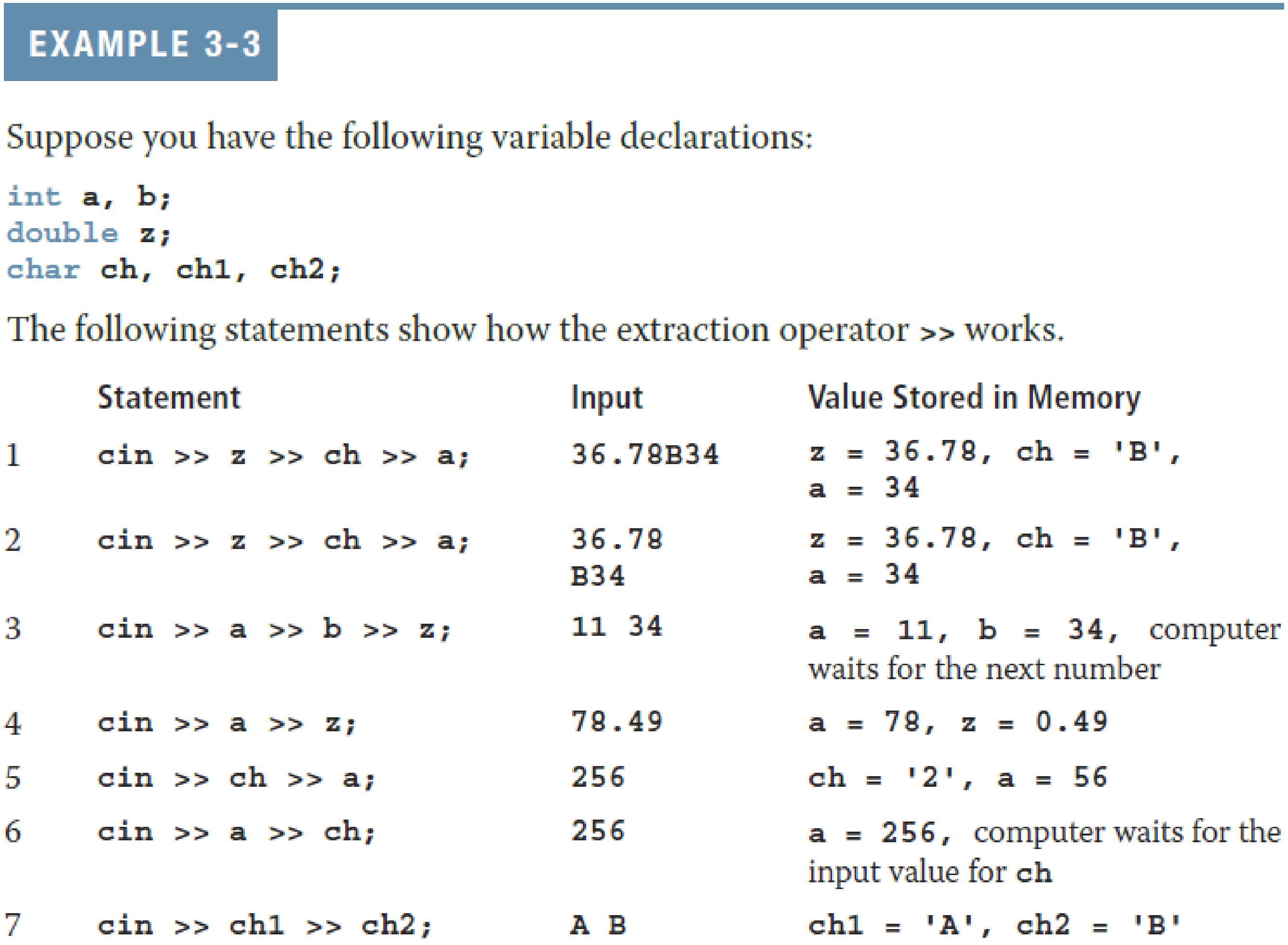
Suppose you have the following variable declarations:

```
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',`<br>`z = 26.9` |
| 2 | `cin >> a >> ch >> z;` | `57 A`<br>`26.9` | `a = 57, ch = 'A',`<br>`z = 26.9` |
| 3 | `cin >> a >> ch >> z;` | `57`<br>`A`<br>`26.9` | `a = 57, ch = 'A',`<br>`z = 26.9` |
| 4 | `cin >> a >> ch >> z;` | `57A26.9` | `a = 57, ch = 'A',`<br>`z = 26.9` |

# cin and the Extraction Operator >> (7 of 7)



**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.

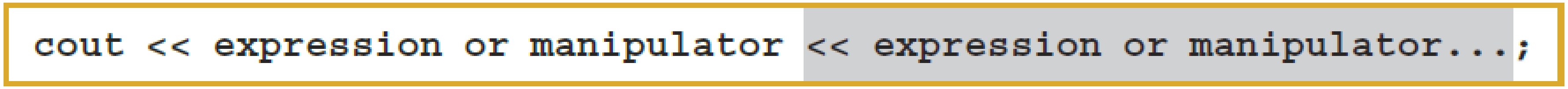
| | Statement | Input | Value Stored in Memory |
|---|---|---|---|
| 1 | `cin >> z >> ch >> a;` | `36.78B34` | `z = 36.78, ch = 'B',`<br>`a = 34` |
| 2 | `cin >> z >> ch >> a;` | `36.78`<br>`B34` | `z = 36.78, ch = 'B',`<br>`a = 34` |
| 3 | `cin >> a >> b >> z;` | `11 34` | `a = 11, b = 34,` computer 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'` |

# Input Failure

- Things can go wrong during execution
- If input data does not match corresponding variables, the 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

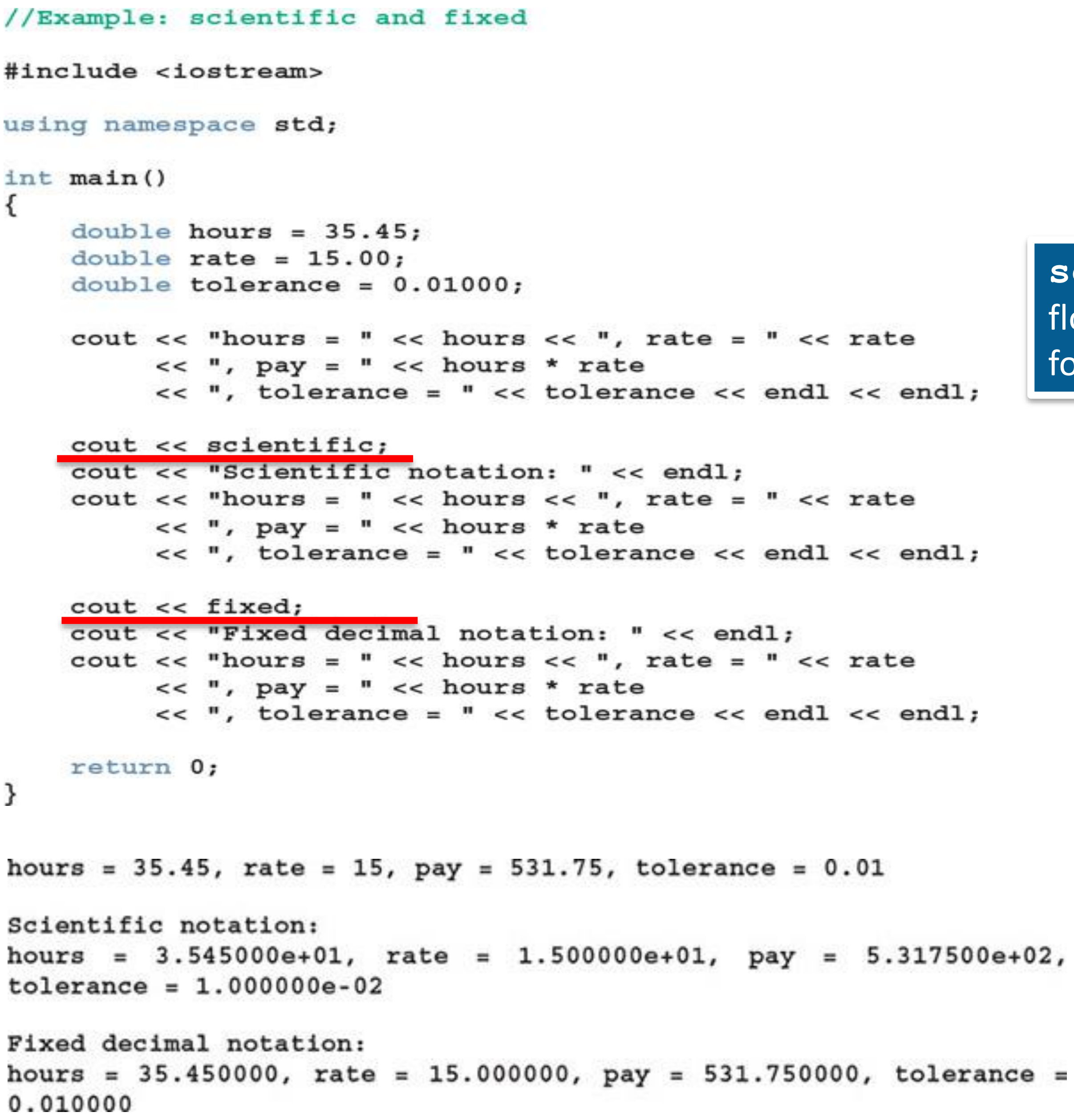
# Output and Formatting Output

- 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`**

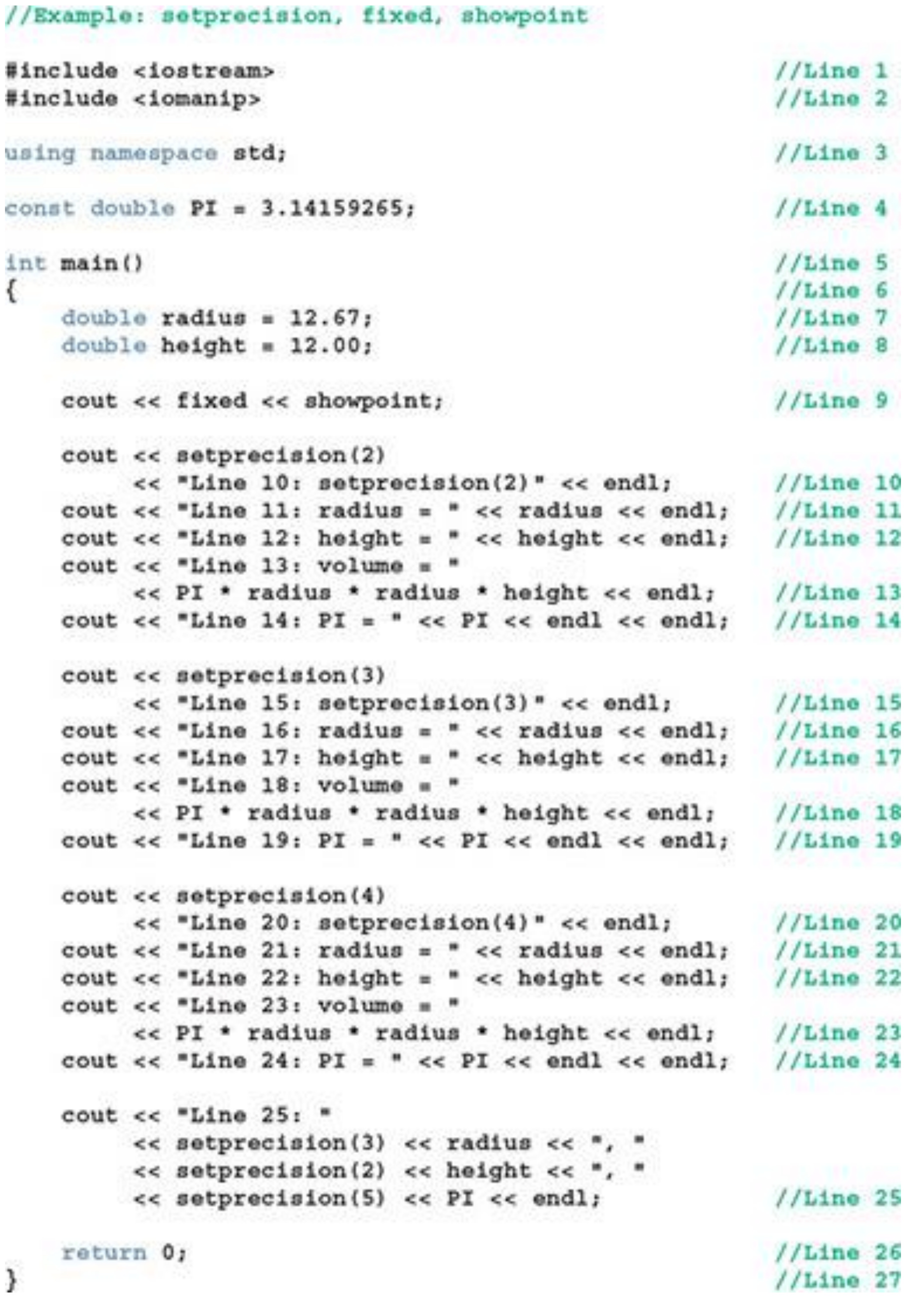


```
//Example: scientific and fixed

#include <iostream>

using namespace std;

int main()
{
    double hours = 35.45;
    double rate = 15.00;
    double tolerance = 0.01000;

    cout << "hours = " << hours << ", rate = " << rate
         << ", pay = " << hours * rate
         << ", tolerance = " << tolerance << endl << endl;

    cout << scientific;
    cout << "Scientific notation: " << endl;
    cout << "hours = " << hours << ", rate = " << rate
         << ", pay = " << hours * rate
         << ", tolerance = " << tolerance << endl << endl;

    cout << fixed;
    cout << "Fixed decimal notation: " << endl;
    cout << "hours = " << hours << ", rate = " << rate
         << ", pay = " << hours * rate
         << ", tolerance = " << tolerance << endl << endl;

    return 0;
}
```

```
hours = 35.45, rate = 15, pay = 531.75, tolerance = 0.01

Scientific notation:
hours = 3.545000e+01, rate = 1.500000e+01, pay = 5.317500e+02,
tolerance = 1.000000e-02

Fixed decimal notation:
hours = 35.450000, rate = 15.000000, pay = 531.750000, tolerance =
0.010000
```

**`scientific`** manipulator outputs floating-point numbers in scientific format

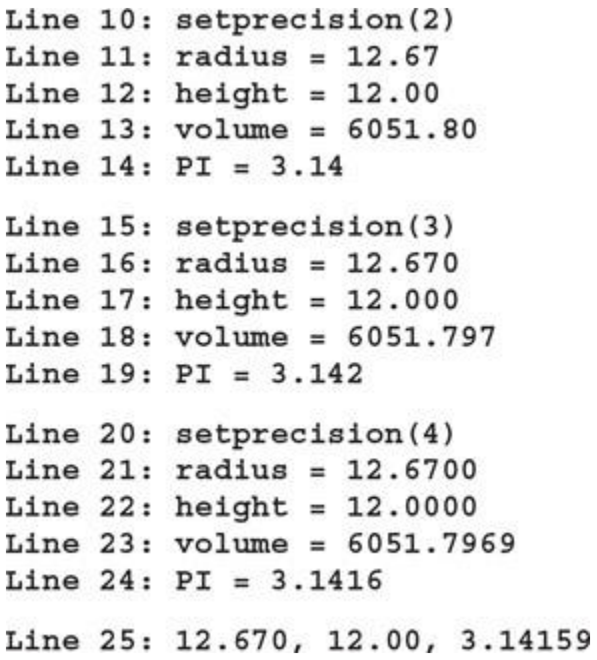
**`fixed`** outputs floating-point numbers in a fixed decimal format



```
//Example: setprecision, fixed, showpoint

#include <iostream>                                 //Line 1
#include <iomanip>                                  //Line 2

using namespace std;                                //Line 3

const double PI = 3.14159265;                       //Line 4

int main()                                          //Line 5
{                                                   //Line 6
    double radius = 12.67;                          //Line 7
    double height = 12.00;                          //Line 8

    cout << fixed << showpoint;                     //Line 9

    cout << setprecision(2)
         << "Line 10: setprecision(2)" << endl;     //Line 10
    cout << "Line 11: radius = " << radius << endl; //Line 11
    cout << "Line 12: height = " << height << endl; //Line 12
    cout << "Line 13: volume = "
         << PI * radius * radius * height << endl;  //Line 13
    cout << "Line 14: PI = " << PI << endl << endl; //Line 14

    cout << setprecision(3)
         << "Line 15: setprecision(3)" << endl;     //Line 15
    cout << "Line 16: radius = " << radius << endl; //Line 16
    cout << "Line 17: height = " << height << endl; //Line 17
    cout << "Line 18: volume = "
         << PI * radius * radius * height << endl;  //Line 18
    cout << "Line 19: PI = " << PI << endl << endl; //Line 19

    cout << setprecision(4)
         << "Line 20: setprecision(4)" << endl;     //Line 20
    cout << "Line 21: radius = " << radius << endl; //Line 21
    cout << "Line 22: height = " << height << endl; //Line 22
    cout << "Line 23: volume = "
         << PI * radius * radius * height << endl;  //Line 23
    cout << "Line 24: PI = " << PI << endl << endl; //Line 24

    cout << "Line 25: "
         << setprecision(3) << radius << ", "
         << setprecision(2) << height << ", "
         << setprecision(5) << PI << endl;          //Line 25

    return 0;                                       //Line 26
}                                                   //Line 27
```

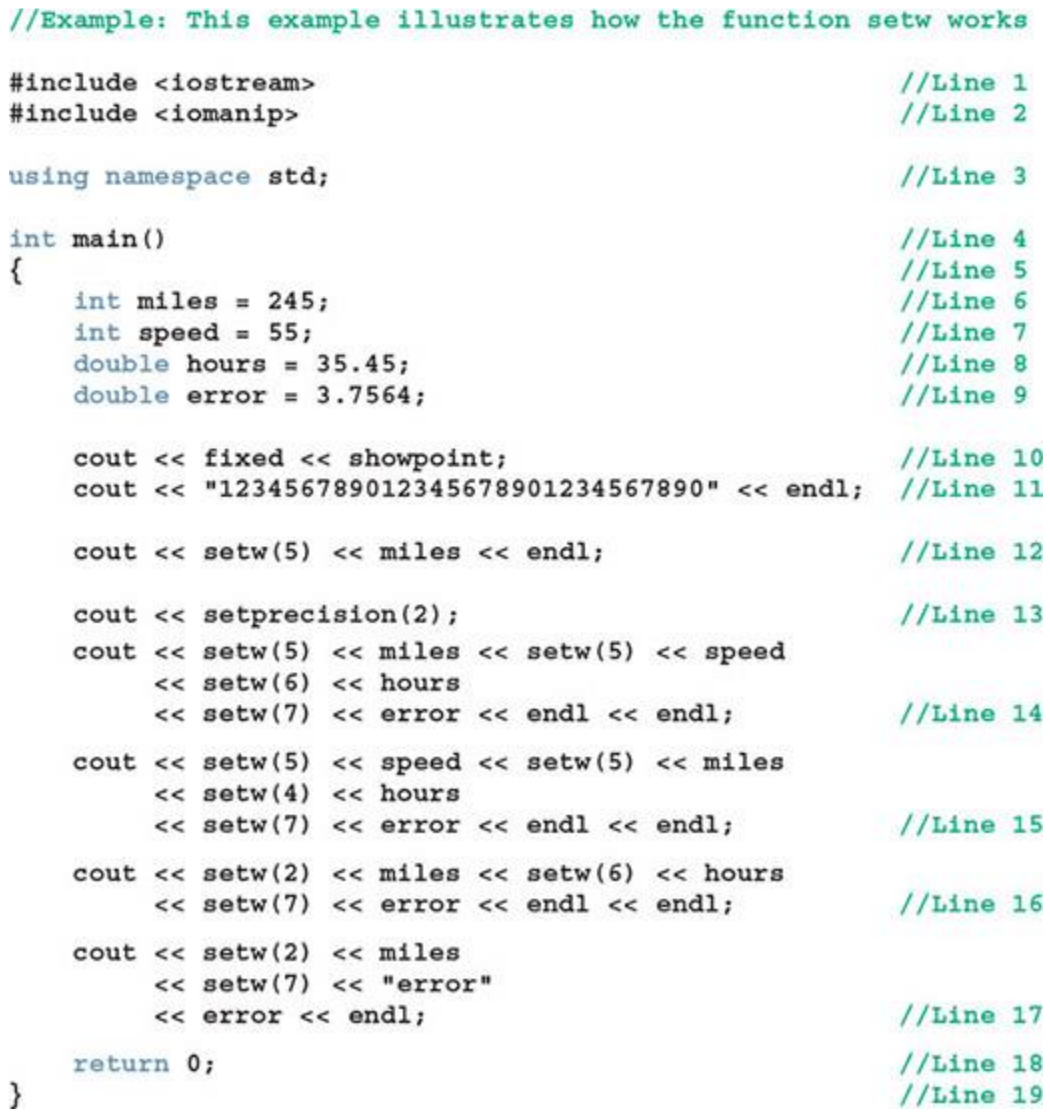
**`setprecision(n)`** outputs decimal numbers with up to **`n`** decimal places

**`showpoint`** forces output to show the decimal point and trailing zeros

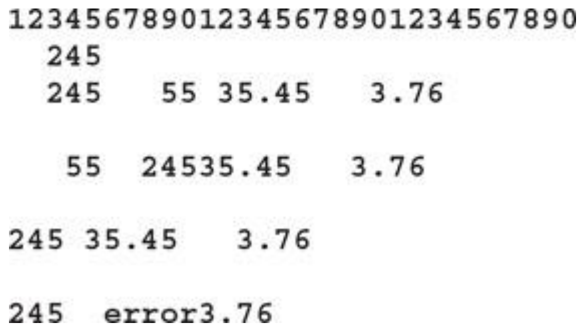


```
Line 10: setprecision(2)
Line 11: radius = 12.67
Line 12: height = 12.00
Line 13: volume = 6051.80
Line 14: PI = 3.14

Line 15: setprecision(3)
Line 16: radius = 12.670
Line 17: height = 12.000
Line 18: volume = 6051.797
Line 19: PI = 3.142

Line 20: setprecision(4)
Line 21: radius = 12.6700
Line 22: height = 12.0000
Line 23: volume = 6051.7969
Line 24: PI = 3.1416

Line 25: 12.670, 12.00, 3.14159
```

**setw(n)** outputs the value of an expression in n number of columns



```
//Example: This example illustrates how the function setw works

#include <iostream>                                 //Line 1
#include <iomanip>                                  //Line 2

using namespace std;                                //Line 3

int main()                                          //Line 4
{                                                   //Line 5
    int miles = 245;                                //Line 6
    int speed = 55;                                 //Line 7
    double hours = 35.45;                           //Line 8
    double error = 3.7564;                          //Line 9

    cout << fixed << showpoint;                     //Line 10
    cout << "123456789012345678901234567890" << endl; //Line 11

    cout << setw(5) << miles << endl;               //Line 12

    cout << setprecision(2);                        //Line 13
    cout << setw(5) << miles << setw(5) << speed
         << setw(6) << hours
         << setw(7) << error << endl << endl;       //Line 14

    cout << setw(5) << speed << setw(5) << miles
         << setw(4) << hours
         << setw(7) << error << endl << endl;       //Line 15

    cout << setw(2) << miles << setw(6) << hours
         << setw(7) << error << endl << endl;       //Line 16

    cout << setw(2) << miles
         << setw(7) << "error"
         << error << endl;                          //Line 17

    return 0;                                       //Line 18
}                                                   //Line 19
```

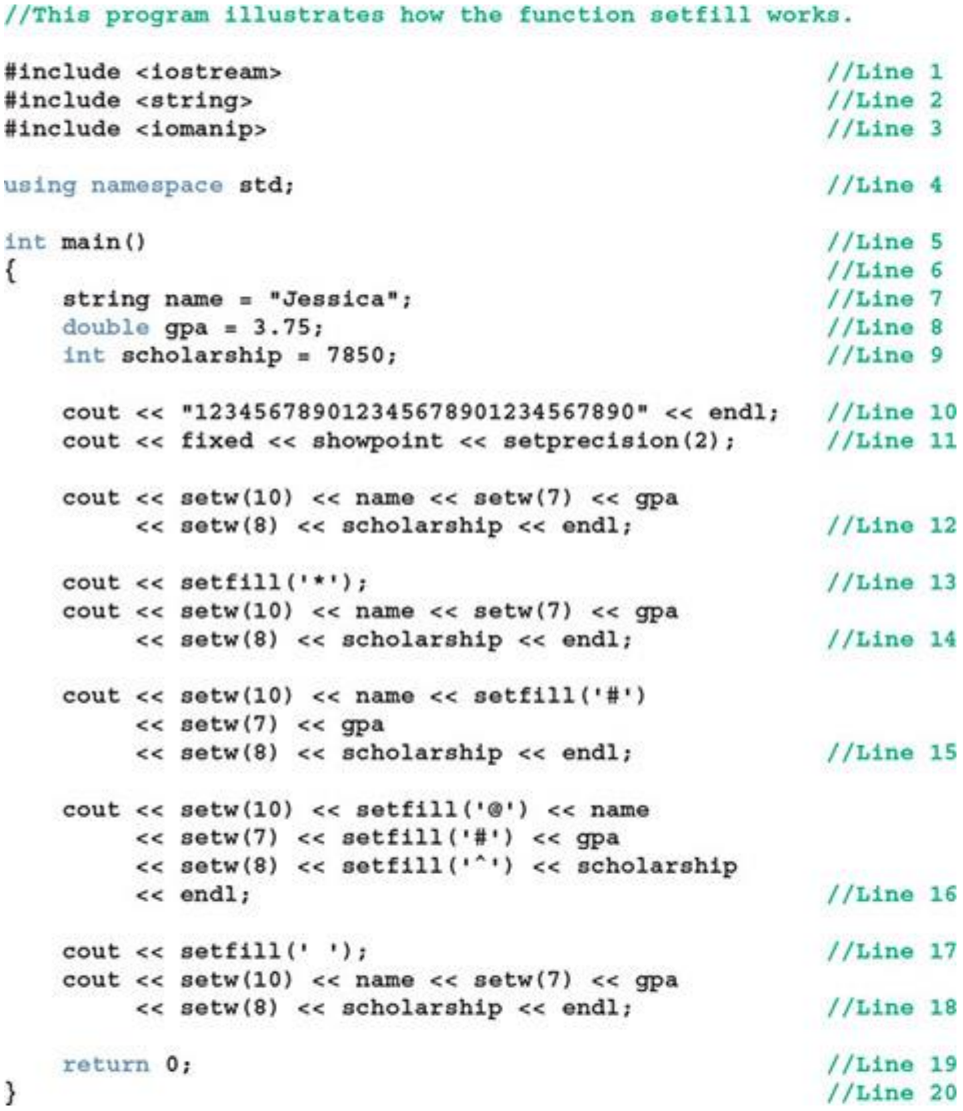


```
123456789012345678901234567890
  245
  245   55 35.45   3.76

   55  24535.45   3.76

245 35.45   3.76

245  error3.76
```

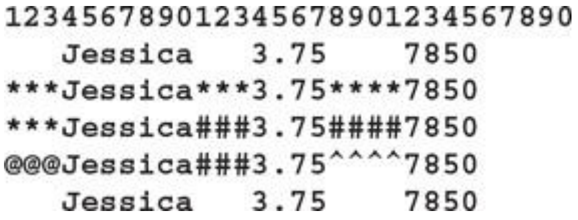
# Additional Output Formatting Tools

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

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



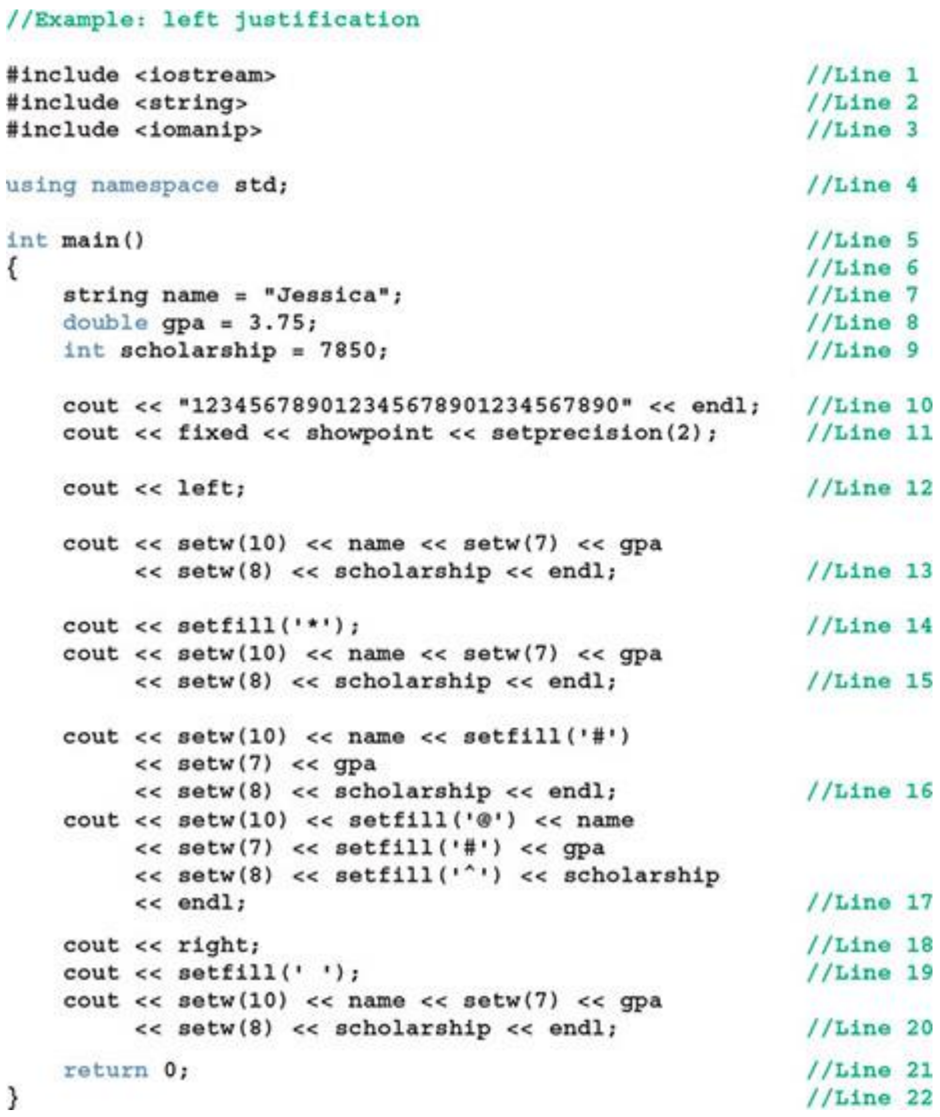
```
//This program illustrates how the function setfill works.

#include <iostream>                                    //Line 1
#include <string>                                      //Line 2
#include <iomanip>                                     //Line 3

using namespace std;                                   //Line 4

int main()                                             //Line 5
{                                                      //Line 6
    string name = "Jessica";                           //Line 7
    double gpa = 3.75;                                 //Line 8
    int scholarship = 7850;                            //Line 9

    cout << "123456789012345678901234567890" << endl;  //Line 10
    cout << fixed << showpoint << setprecision(2);     //Line 11

    cout << setw(10) << name << setw(7) << gpa
         << setw(8) << scholarship << endl;            //Line 12

    cout << setfill('*');                              //Line 13
    cout << setw(10) << name << setw(7) << gpa
         << setw(8) << scholarship << endl;            //Line 14

    cout << setw(10) << name << setfill('#')
         << setw(7) << gpa
         << setw(8) << scholarship << endl;            //Line 15

    cout << setw(10) << setfill('@') << name
         << setw(7) << setfill('#') << gpa
         << setw(8) << setfill('^') << scholarship
         << endl;                                      //Line 16

    cout << setfill(' ');                              //Line 17
    cout << setw(10) << name << setw(7) << gpa
         << setw(8) << scholarship << endl;            //Line 18

    return 0;                                          //Line 19
}                                                      //Line 20
```



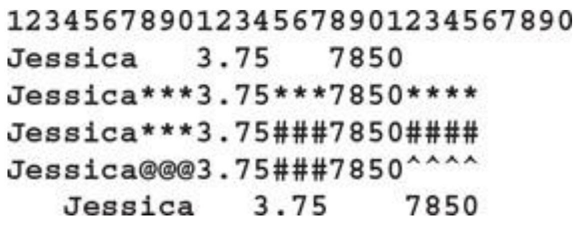
```
123456789012345678901234567890
   Jessica   3.75    7850
***Jessica***3.75****7850
***Jessica###3.75####7850
@@@Jessica###3.75^^^^7850
   Jessica   3.75    7850
```

## `left` manipulator left-justifies the output

## `right` manipulator right-justifies the output



```
//Example: left justification

#include <iostream>                                     //Line 1
#include <string>                                       //Line 2
#include <iomanip>                                      //Line 3

using namespace std;                                    //Line 4

int main()                                              //Line 5
{                                                       //Line 6
    string name = "Jessica";                            //Line 7
    double gpa = 3.75;                                  //Line 8
    int scholarship = 7850;                             //Line 9

    cout << "123456789012345678901234567890" << endl;   //Line 10
    cout << fixed << showpoint << setprecision(2);      //Line 11

    cout << left;                                       //Line 12

    cout << setw(10) << name << setw(7) << gpa
         << setw(8) << scholarship << endl;             //Line 13

    cout << setfill('*');                               //Line 14
    cout << setw(10) << name << setw(7) << gpa
         << setw(8) << scholarship << endl;             //Line 15

    cout << setw(10) << name << setfill('#')
         << setw(7) << gpa
         << setw(8) << scholarship << endl;             //Line 16
    cout << setw(10) << setfill('@') << name
         << setw(7) << setfill('#') << gpa
         << setw(8) << setfill('^') << scholarship
         << endl;                                       //Line 17

    cout << right;                                      //Line 18
    cout << setfill(' ');                               //Line 19
    cout << setw(10) << name << setw(7) << gpa
         << setw(8) << scholarship << endl;             //Line 20

    return 0;                                           //Line 21
}                                                       //Line 22
```



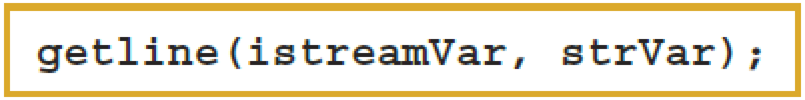
```
123456789012345678901234567890
Jessica   3.75   7850
Jessica***3.75***7850****
Jessica***3.75###7850####
Jessica@@@3.75###7850^^^^
   Jessica   3.75    7850
```

# Types of Manipulators

- Two types of manipulators
  - Those with parameters
  - Those without parameters
- Parameterized stream manipulators require the **`iomanip`** header
  - **`setprecision`**, **`setw`**, and **`setfill`**
- Manipulators without parameters require the **`iostream`** header
  - **`endl`**, **`fixed`**, **`scientific`**, **`showpoint`**, and **`left`**

# Input/Output and the `string` Type

- An input stream variable (such as **`cin`**) and **>>** operator can read a string into a variable of the data type **`string`**
- The extraction operator:
  - Skips any leading whitespace characters
  - Stops reading at a whitespace character
- The function **`getline`** reads until end of the current line



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

- Example

# Mathematical Library Functions

- Before using a C++ mathematical function, the programmer must know:
  - Name of the desired mathematical function
  - What the function does
  - Type of data required by the function
  - Data type of the result returned by the function
- To access mathematical functions in a program, the header file `cmath` must be used
  - Format: **`#include <cmath>`** <- no semicolon

# Mathematical Library Functions (cont'd.)

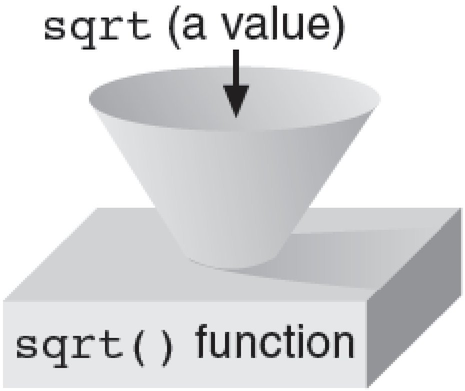


**Figure 3.7** Passing data to the `sqrt()` function

# Mathematical Library Functions (cont'd.)

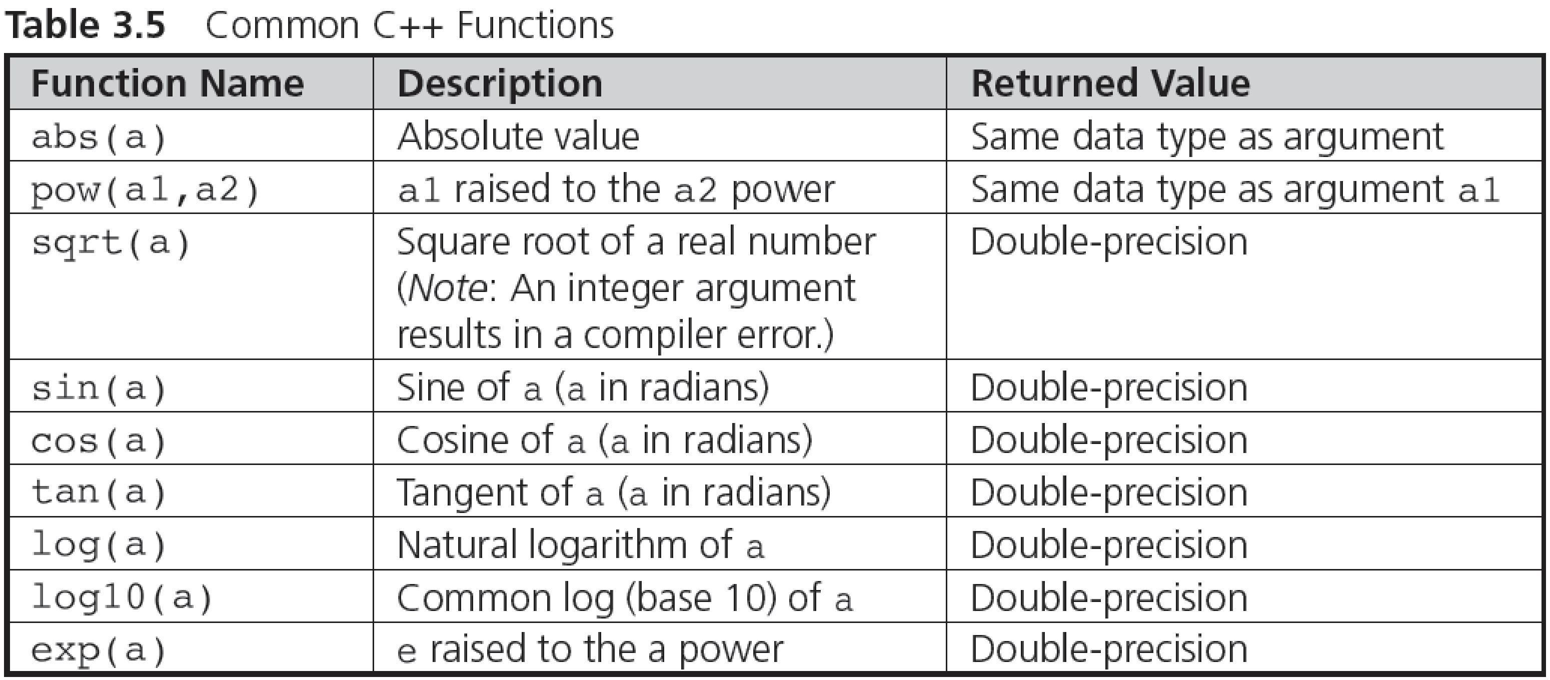
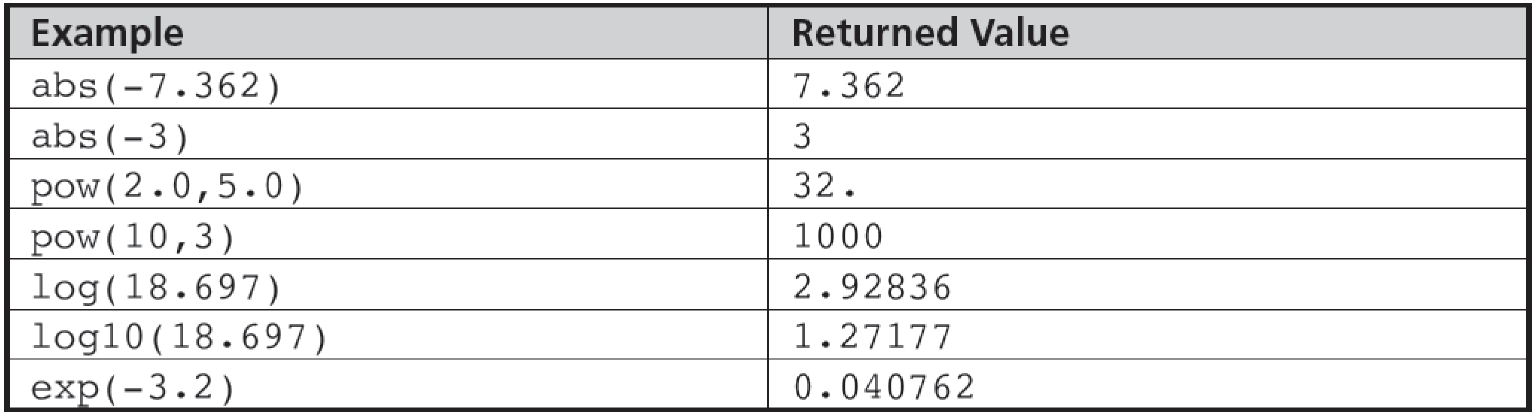


**Table 3.5** Common C++ Functions

| Function Name | Description | Returned Value |
|---|---|---|
| `abs(a)` | Absolute value | Same data type as argument |
| `pow(a1,a2)` | `a1` raised to the `a2` power | Same data type as argument `a1` |
| `sqrt(a)` | Square root of a real number (*Note*: An integer argument results in a compiler error.) | Double-precision |
| `sin(a)` | Sine of `a` (`a` in radians) | Double-precision |
| `cos(a)` | Cosine of `a` (`a` in radians) | Double-precision |
| `tan(a)` | Tangent of `a` (`a` in radians) | Double-precision |
| `log(a)` | Natural logarithm of `a` | Double-precision |
| `log10(a)` | Common log (base 10) of `a` | Double-precision |
| `exp(a)` | `e` raised to the a power | Double-precision |

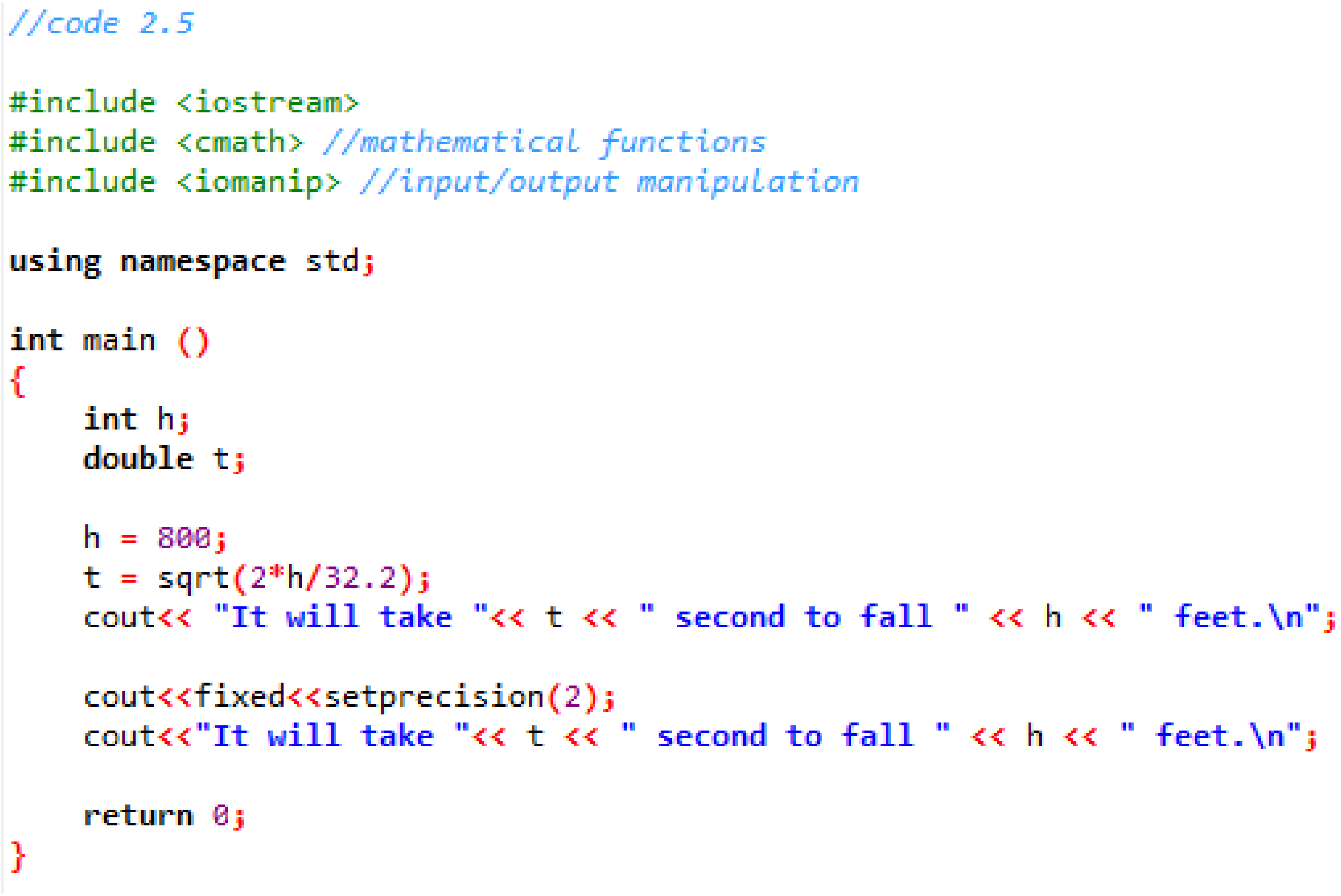
# Mathematical Library Functions (cont'd.)

Table 3.6 Selected Function Examples



| Example | Returned Value |
|---|---|
| abs(-7.362) | 7.362 |
| abs(-3) | 3 |
| pow(2.0,5.0) | 32. |
| pow(10,3) | 1000 |
| log(18.697) | 2.92836 |
| log10(18.697) | 1.27177 |
| exp(-3.2) | 0.040762 |

# Mathematical Library Functions (cont'd.)



```
//code 2.5

#include <iostream>
#include <cmath> //mathematical functions
#include <iomanip> //input/output manipulation

using namespace std;

int main ()
{
    int h;
    double t;

    h = 800;
    t = sqrt(2*h/32.2);
    cout<< "It will take "<< t << " second to fall " << h << " feet.\n";

    cout<<fixed<<setprecision(2);
    cout<<"It will take "<< t << " second to fall " << h << " feet.\n";

    return 0;
}
```

| | |
|---|---|
| $x^y$ | double pow(double x, double y);<br>float powf(float x, float y);<br>long double powl(long double x, long double y); |
| Square root of $x$ $(x^{1/2})$ | double sqrt(double x);<br>float sqrtf(float x);<br>long double sqrtl(long double x); |
| Cubic root of $x$ $(x^{1/3})$ | double cbrt(double x);<br>float cbrtf(float x);<br>long double cbrtl(long double x); |
| Hypotenuse of a right-angled triangle whose legs are $x$ and $y$, $(x^2+y^2)^{1/2}$ | double hypot(double x, double y);<br>float hypotf(float x, float y);<br>long double hypotl(long double x, long double y); |

| | |
|---|---|
| Rounds *x* upward to an integer (ceiling) | `double ceil(double x);`<br>`float ceilf(float x);`<br>`long double ceill(long double x);` |
| Rounds *x* downward to an integer (floor) | `double floor(double x);`<br>`float floorf(float x);`<br>`long double floorl(long double x);` |
| Rounds *x* toward zero | `double trunc(double x);`<br>`float truncf(float x);`<br>`long double truncl(long double x);` |
| Round *x* | `double round(double x);`<br>`float roundf(float x);`<br>`long double roundl(long double x);` |

```
#include <iostream>
#include <cmath>

int main ()
{
  cout << "value\tround\tfloor\tceil\ttrunc\n";
  cout << "-----\t-----\t-----\t----\t-----\n";
  cout << 2.3 << “\t” << round( 2.3) << “\t” << floor( 2.3) << “\t” <<
ceil( 2.3) << “\t” << trunc( 2.3) << endl;
  cout << 3.8 << “\t” << round( 3.8) << “\t” << floor( 3.8) << “\t” <<
ceil( 3.8) << “\t” << trunc( 3.8) << endl;
  cout << 5.5 << “\t” << round( 5.5) << “\t” << floor( 5.5) << “\t” <<
ceil( 5.5) << “\t” << trunc( 5.5) << endl;
return 0;
}
```