3.1 The cin Object

The cin Object

- Standard input object
- Like cout, requires iostream file
- Used to read input from keyboard
- Information retrieved from cin with `>>`
- Input is stored in one or more variables

The cin Object in Program 3-1

```
int height;
cout << "How tall is the room? ";
cin >> height;
```

Displaying a Prompt

- A prompt is a message that instructs the user to enter data.
- You should always use `cout` to display a prompt before each `cin` statement.

```
cout << "How tall is the room? ";
cin >> height;
```
The cin Object

- Can be used to input more than one value:
  
  ```cpp
  cin >> height >> width;
  ```

- Multiple values from keyboard must be separated by spaces

- Order is important: first value entered goes to first variable, etc.

The cin Object Gathers Multiple Values in Program 3-2

```cpp
Program 3-2

// This program asks the user to enter the length and width of 
// a rectangle. It calculates the rectangle's area and displays 
// the value on the screen.
#include <iostream>
using namespace std;

int main()
{
  int length, width, area;
  cout << "This program calculates the area of a rectangle\n";
  cout << "Enter the length and width of the rectangle:\n";
  cin >> length >> width;
  area = length * width;
  cout << "The area of the rectangle is: " << area << endl;
  return 0;
}
```

Program Output with Example Input Shown in Bold

This program calculates the area of a rectangle.

10 20 [Input]

The area of the rectangle is 200

The cin Object Reads Different Data Types in Program 3-3

```cpp
Program 3-3

// This program demonstrates how cin can read multiple values
// of different data types.
#include <iostream>
using namespace std;

int main()
{
  int whole;
  double fractional;
  char letter;
  cout << "Enter an integer, a double, and a character: \n";
  cin >> whole >> fractional >> letter;
  cout << "The integer is: " << whole << endl;
  cout << "The double is: " << fractional << endl;
  cout << "The character is: " << letter << endl;
  return 0;
}
```

Program Output with Example Input Shown in Bold

Enter an integer, a double, and a character: 4.57 b [Input]

Whole: 4
Fractional: 5.7
Letter: b

3.2

Mathematical Expressions

- Can create complex expressions using multiple mathematical operators
- An expression can be a literal, a variable, or a mathematical combination of constants and variables
- Can be used in assignment, cout, other statements:
  - `area = 2 * PI * radius;`
  - `cout << "border is: " << 2*(l+w);`

Order of Operations

In an expression with more than one operator, evaluate in this order:
- (unary negation), in order, left to right
- / %, in order, left to right
- + -, in order, left to right

In the expression `2 + 2 * 2 - 2`

1. Evaluate second
2. Evaluate first
3. Evaluate third
Order of Operations

Table 3.2 Some Simple Expressions and Their Values

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 + 2 + 4</td>
<td>13</td>
</tr>
<tr>
<td>10 / 2 + 3</td>
<td>2</td>
</tr>
<tr>
<td>8 + 12 * 2 - 4</td>
<td>28</td>
</tr>
<tr>
<td>4 + 17 % 2 - 1</td>
<td>0</td>
</tr>
<tr>
<td>6 - 3 * 2 + 7 - 2</td>
<td>6</td>
</tr>
</tbody>
</table>

Associativity of Operators

- (unary negation) associates right to left
- *, /, %, +, - associate right to left
- parentheses ( ) can be used to override the order of operations:
  - \((2 + 2) \times (2 - 2) = 0\)
  - \((2 + 2) \times 2 - 2 = 6\)
  - \(2 + 2 \times (2 - 2) = 2\)

Grouping with Parentheses

Table 3.4 More Simple Expressions and Their Values

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5 + 2) * 4</td>
<td>28</td>
</tr>
<tr>
<td>10 / (5 + 3)</td>
<td>5</td>
</tr>
<tr>
<td>8 + 12 * (6 - 2)</td>
<td>56</td>
</tr>
<tr>
<td>(4 + 17) % 2 - 1</td>
<td>0</td>
</tr>
<tr>
<td>(6 - 3) * (2 + 7) / 3</td>
<td>9</td>
</tr>
</tbody>
</table>

Algebraic Expressions

- Multiplication requires an operator: \(Area=lw\) is written as \(Area = l \times w\);
- There is no exponentiation operator: \(Area=s^2\) is written as \(Area = pow(s, 2)\);
- Parentheses may be needed to maintain order of operations:
  - \(m=(y_2-y_1)/(x_2-x_1)\) is written as \(m = \frac{y_2-y_1}{x_2-x_1}\);

Algebraic Expressions

Table 3.5 Algebraic and C++ Multiplication Expressions

<table>
<thead>
<tr>
<th>Algebraic Expression</th>
<th>Operation</th>
<th>C++ Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x/3))</td>
<td>x / 3</td>
<td>x / 3</td>
</tr>
<tr>
<td>((x+y))</td>
<td>x + y</td>
<td>x + y</td>
</tr>
<tr>
<td>((x-y))</td>
<td>x - y</td>
<td>x - y</td>
</tr>
</tbody>
</table>

3.3

When You Mix Apples with Oranges: Type Conversion
When You Mix Apples with Oranges: Type Conversion

- Operations are performed between operands of the same type.
- If not of the same type, C++ will convert one to be the type of the other.
- This can impact the results of calculations.

Hierarchy of Types

Highest: `long double`  
`double`  
`float`  
`unsigned long`  
`long`  
`unsigned int`  
`int`  

Lowest: 
Ranked by largest number they can hold

Type Coercion

- **Type Coercion**: automatic conversion of an operand to another data type
- **Promotion**: convert to a higher type
- **Demotion**: convert to a lower type

Coercion Rules

1) `char`, `short`, `unsigned short` automatically promoted to `int`
2) When operating on values of different data types, the lower one is promoted to the type of the higher one.
3) When using the `=` operator, the type of expression on right will be converted to type of variable on left

Overflow and Underflow

- Occurs when assigning a value that is too large (overflow) or too small (underflow) to be held in a variable
- Variable contains value that is ‘wrapped around’ set of possible values
- Different systems may display a warning/error message, stop the program, or continue execution using the incorrect value
3.5 Type Casting

- Used for manual data type conversion
- Useful for floating point division using ints:
  ```
  double m;
  m = static_cast<double>((y2-y1)/(x2-x1));
  ```
- Useful to see int value of a char variable:
  ```
  char ch = 'C';
  cout << ch << " is " << static_cast<int>(ch);
  ```

3.6 Multiple Assignment and Combined Assignment

- The = can be used to assign a value to multiple variables:
  ```
  x = y = z = 5;
  ```
- Value of = is the value that is assigned
- Associates right to left:
  ```
  x = (y = (z = 5));
  ```
Combined Assignment

- Look at the following statement:

  \[ \text{sum} = \text{sum} + 1; \]

  This adds 1 to the variable \text{sum}.

Combined Assignment

- The combined assignment operators provide a shorthand for these types of statements.
- The statement

  \[ \text{sum} = \text{sum} + 1; \]

  is equivalent to

  \[ \text{sum} += 1; \]

Other Similar Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>What It Does</th>
<th>Value of ( x ) After the Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x += y )</td>
<td>Adds ( y ) to ( x )</td>
<td>( x )</td>
</tr>
<tr>
<td>( x *= y )</td>
<td>Multiplies ( x ) by ( y )</td>
<td>( x )</td>
</tr>
<tr>
<td>( x /= y )</td>
<td>Divides ( x ) by ( y )</td>
<td>( x )</td>
</tr>
<tr>
<td>( x %= y )</td>
<td>Makes ( x ) the remainder of ( x \mod y )</td>
<td>( x )</td>
</tr>
</tbody>
</table>

Combined Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example Usage</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>( x += y )</td>
<td>( x = x + y )</td>
</tr>
</tbody>
</table>
| *=       | \( x *= y \) | \( x = x 
| /=       | \( x /= y \) | \( x = x / y \) |
| %=       | \( x %= y \) | \( x = x \mod y \) |

Formatting Output

- Can control how output displays for numeric, string data:
  - size
  - position
  - number of digits
- Requires \text{iomanip} header file

3.7

Formatting Output
Stream Manipulators

- Used to control how an output field is displayed
- Some effect just the next value displayed:
  - `setw(x)`: print in a field at least x spaces wide. Use more spaces if field is not wide enough

The `setw` Stream Manipulator in Program 3-13

```
Program 3-13
1: // This program displays three rows of numbers.
2: #include <iostream>
3: #include <iomanip>  // Required for set
4: using namespace std;
5: int main()
6: {
7:   double real1 = 397.5, real2 = 272.3, real3 = 123.45;
8:   real1 += 36, real2 += 7, real3 += 12;
9:   // Display the first row of numbers
10:   cout << setw(10) << real1 << setw(10) << real2 << setw(10) << real3 << endl;
11:   // Display the second row of numbers
12:   cout << setprecision(2) << real1 << endl;
13:   // Display the third row of numbers
14:   cout << fixed << real2 << endl;
15:   cout << showpoint << real3 << endl;
16: return 0;
17: }
```

Stream Manipulators

- Some effect values until changed again:
  - `fixed`: use decimal notation for floating-point values
  - `setprecision(x)`: when used with `fixed`, print floating-point value using x digits after the decimal. Without `fixed`, print floating-point value using x significant digits
  - `showpoint`: always print decimal for floating-point values

More Stream Manipulators in Program 3-17

```
Program 3-17
1: // This program adds sales figures for 3 days. The total
2: // sales are calculated and displayed in a table.
3: #include <iostream>
4: using namespace std;
5: int main()
6: {
7:   double day1, day2, day3, total;
8:   // Set the sales for each day.
9:   cout << "Enter the sales for day 1: " ;
10:   cin >> day1;
11:   cout << "Enter the sales for day 2: " ;
12:   cin >> day2;
13:   cout << "Enter the sales for day 3: " ;
14:   cin >> day3;
15:   total = day1 + day2 + day3;
16:   // Calculate the total sales.
17:   cout << "Total sales: " << total << endl;
18: return 0;
```

More Stream Manipulators in Program 3-17

```
Program 3-17
17: // Display the sales figures.
18: cout << "-----------" << endl;
19: cout << "Day 1: " << day1 << endl;
20: cout << "Day 2: " << day2 << endl;
21: cout << "Day 3: " << day3 << endl;
22: cout << "Total: " << total << endl;
```

Program Output with Example Input shown in bold

Enter the sales for day 1: 1521.87
Enter the sales for day 2: 1499.26
Enter the sales for day 3: 1499.77

Sales Figures

<table>
<thead>
<tr>
<th>Day</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day 1</td>
<td>1521.87</td>
</tr>
<tr>
<td>Day 2</td>
<td>1499.26</td>
</tr>
<tr>
<td>Day 3</td>
<td>1499.77</td>
</tr>
<tr>
<td>Total</td>
<td>4510.99</td>
</tr>
</tbody>
</table>
Stream Manipulators

<table>
<thead>
<tr>
<th>Stream Manipulator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setw()</td>
<td>Establishes a prior field of n spaces.</td>
</tr>
<tr>
<td>fill()</td>
<td>Displays leading zeros in fixed-point notation.</td>
</tr>
<tr>
<td>showpoint()</td>
<td>Causes a decimal point and trailing zeros to be displayed, even if there is no fractional part.</td>
</tr>
<tr>
<td>setprecision()</td>
<td>Sets the precision of floating-point numbers.</td>
</tr>
<tr>
<td>left()</td>
<td>Causes subsequent output to be left justified.</td>
</tr>
<tr>
<td>right()</td>
<td>Causes subsequent output to be right justified.</td>
</tr>
</tbody>
</table>

3.8 Working with Characters and String Objects

Using `cin` with the >> operator to input strings can cause problems:
- It passes over and ignores any leading whitespace characters (spaces, tabs, or line breaks)
- To work around this problem, you can use a C++ function named `getline`.

To read a single character:
- Use `cin`:
  ```cpp
cchar ch;
cout << "Strike any key to continue"; cin >> ch;  // will skip over blanks, tabs, <CR>
cin.get(ch);  // will read the next character entered, even whitespace
```

Using `getline` in Program 3-19

```cpp
// This program demonstrates using the getline function.
#include <iostream>
#include <string>
int main()
{ 
    std::string input;
    std::cout << "Enter your name: " << std::endl;
    std::getline(std::cin, input);
    std::cout << "Hello, " << input << std::endl;
    return 0;
}
```

Program Output:
- Enter your name: John
- Hello, John

Using `cin.get()` in Program 3-21

```cpp
// This program demonstrates the use of cin.get().
#include <iostream>
int main()
{ 
    char ch;
    std::cout << "Strike any key to continue"; cin.get(ch);
    return 0;
}
```

Program Output:
- Strike any key to continue. [Enter]
- Enter any character. [Enter]
- Thank you!
Working with Characters and \texttt{string} Objects

- Mixing \texttt{cin >>} and \texttt{cin.get()} in the same program can cause input errors that are hard to detect.
- To skip over unneeded characters that are still in the keyboard buffer, use \texttt{cin.ignore()}:
  - \texttt{cin.ignore(); // skip next char}
  - \texttt{cin.ignore(10, '\n'); // skip the next 10 char. or until a '\n'}

\texttt{string} Member Functions and Operators

- To find the length of a string:
  - \texttt{string state = "Texas"; int size = state.length();}
- To concatenate (join) multiple strings:
  - \texttt{greeting2 = greeting1 + name1;}
  - \texttt{greeting1 = greeting1 + name2;}
  - Or using the += combined assignment operator:
    - \texttt{greeting1 += name2;}

More Mathematical Library Functions

- Require \texttt{cmath} header file
- Take double as input, return a double
- Commonly used functions:
  - \texttt{sin} Sine
  - \texttt{cos} Cosine
  - \texttt{tan} Tangent
  - \texttt{sqrt} Square root
  - \texttt{log} Natural (e) log
  - \texttt{abs} Absolute value (takes and returns an int)

These require \texttt{cstdlib} header file

- \texttt{rand}(): returns a random number (int) between 0 and the largest int the compute holds. Yields same sequence of numbers each time program is run.
- \texttt{srand(x)}: initializes random number generator with unsigned int \texttt{x}