Chapter 3: Expressions and Interactivity

3.1 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

- cin converts data to the type that matches the variable:

  ```
  int height;
  cout << "How tall is the room? \n";
  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? \n";
  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.

Reading Strings with `cin`

- Can be used to read in a string
- Must first declare an array to hold characters in string:
  ```cpp
  char myName[21];
  ```
- `myName` is name of array, 21 is the number of characters that can be stored (the size of the array), including the NULL character at the end
- Can be used with `cin` to assign a value:
  ```cpp
  cin >> myName;
  ```

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:
  ```cpp
  circumference = 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), right to left
* / %, in order, left to right
+ -, in order, left to right

In the expression \( 2 + 2 \times 2 \) – 2

1. evaluate first
2. evaluate second
3. evaluate third

<table>
<thead>
<tr>
<th>Table 3-2 Some Expressions</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>6 + 12 * 2 – 4</td>
<td>20</td>
</tr>
<tr>
<td>4 + 17 / 2 – 1</td>
<td>4</td>
</tr>
<tr>
<td>6 – 2 * 2 + 7 – 1</td>
<td>6</td>
</tr>
</tbody>
</table>

Associativity of Operators

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

\[
\begin{align*}
2 + 2 & \times 2 \times 2 = 4 \\
(2 + 2) & \times 2 \times 2 = 6 \\
2 + 2 & \times (2 - 2) = 2 \\
(2 + 2) & \times (2 - 2) = 0
\end{align*}
\]

Grouping with Parentheses

<table>
<thead>
<tr>
<th>Table 3-4 More Expressions</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5 + 3) * 4</td>
<td>28</td>
</tr>
<tr>
<td>14 / 3 (5 – 3)</td>
<td>5</td>
</tr>
<tr>
<td>8 + 12 * (6 - 2)</td>
<td>50</td>
</tr>
<tr>
<td>(4 + 17) + 2 – 1</td>
<td>4</td>
</tr>
<tr>
<td>(6 – 3) * (2 + 7) / 5</td>
<td>9</td>
</tr>
</tbody>
</table>

Algebraic Expressions

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

<table>
<thead>
<tr>
<th>Table 3-5 Algebraic and C++ Multiplication Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algebraic Expression</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>( a \times b )</td>
</tr>
<tr>
<td>( (a)(b) )</td>
</tr>
<tr>
<td>( a \times b \times c \times d )</td>
</tr>
</tbody>
</table>
When You Mix Apples and 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

Lowest: int

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
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

Type Casting

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

C-Style and Prestandard Type Cast Expressions

- C-Style cast: data type name in ()
  ```c++
  cout << ch << " is " << (int)ch;
  ```
- Prestandard C++ cast: value in ()
  ```c++
  cout << ch << " is " << int(ch);
  ```
- Both are still supported in C++, although static_cast is preferred

Program 3.11

```c++
// This program casts a type cast to avoid integer division.
#include <iostream>
int main()
{
    double x = 123.456789;
    double y = 987.654321;
    double m = static_cast<double>(y2-y1)/(x2-x1);
    return 0;
}
```

Named Constants

- C-Style cast: data type name in ()
  ```c++
  cout << ch << " is " << (int)ch;
  ```
- Prestandard C++ cast: value in ()
  ```c++
  cout << ch << " is " << int(ch);
  ```
- Both are still supported in C++, although static_cast is preferred
Named Constants

- **Named constant (constant variable):**
  variable whose content cannot be changed during program execution
- **Used for representing constant values with descriptive names:**
  ```
  const double TAX_RATE = 0.0675;
  const int NUM_STATES = 50;
  ```
- **Often named in uppercase letters**

Constants and Array Sizes

- It is a common practice to use a named constant to indicate the size of an array:
  ```
  const int SIZE = 21;
  char name[SIZE];
  ```
- The sizes of many arrays --sometimes scattered throughout a program-- can be updated with a single change to a constant

const vs. #define

- **#define** -- C-style of naming constants:
  ```
  #define NUM_STATES 50
  ```
  - Note no semicolon at end
  - Interpreted by pre-processor rather than compiler; replaces the text of constant with its value
  - Does not occupy memory location like **const**

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 \textit{sum}.

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 = x + 1; )</td>
<td>Adds 1 to ( x )</td>
<td>10</td>
</tr>
<tr>
<td>( x = x - 2; )</td>
<td>Subtracts 2 from ( x )</td>
<td>3</td>
</tr>
<tr>
<td>( x = x * 10; )</td>
<td>Multiplies ( x ) by 10</td>
<td>60</td>
</tr>
<tr>
<td>( x = x / 2; )</td>
<td>Divides ( x ) by 2</td>
<td>3</td>
</tr>
<tr>
<td>( x = x \mod 4 )</td>
<td>Makes ( x ) the remainder of ( x \div 4 )</td>
<td>2</td>
</tr>
</tbody>
</table>

Combined Assignment Operators

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

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

is equivalent to

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

Formatting Output

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

• Used to control how an output field is displayed

• Some affect 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

---

Stream Manipulators

• Some affect 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

---

Program 3-17

```
// This program displays three rows of numbers.
#include <iostream>
#include <iomanip>  // Required for setw
using namespace std;

int main()
{
    int num1 = 307, num2 = 5, num3 = 113;
    num1 -= 10, num2 += 1, num3 *= 2;
    num1 -= 20, num2 = 100, num3 += 11;
    num1 = 200, num2 = 1000, num3 = 111;

    // Display the first row of numbers
    cout << setw(5) << num1 << num2 << num3 << endl;
    num1 = 25, num2 = 999, num3 = 33;
    // Display the second row of numbers
    cout << setw(5) << num1 << num2 << num3 << endl;
    num1 = 25, num2 = 999, num3 = 33;
    num1 = 25, num2 = 999, num3 = 33;
    // Display the third row of numbers
    cout << setw(5) << num1 << num2 << num3 << endl;
    num1 = 25, num2 = 999, num3 = 33;
    return 0;
}
```

Program 3-21

```
// This program sets the sales figures for 3 days. The total is displayed.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    double d1, d2, d3, total;
    // Set the sales for each day.
    d1 = get the sales for Day 1;
    d2 = get the sales for Day 2;
    d3 = get the sales for Day 3;
    // Display the sales figures.
    cout << "Sales figures:");
    cout << "Day 1: " << d1 << endl;
    cout << "Day 2: " << d2 << endl;
    cout << "Day 3: " << d3 << endl;
    return 0;
}
```

---

Program Output

<table>
<thead>
<tr>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td>750</td>
<td>1500</td>
<td>2000</td>
</tr>
<tr>
<td>300</td>
<td>600</td>
<td>900</td>
</tr>
<tr>
<td>1500</td>
<td>3000</td>
<td>4500</td>
</tr>
</tbody>
</table>

---

Program Output with Example Input Shown in Bold

- Enter the sales for Day 1: 1232.37 [Bold]
- Enter the sales for Day 2: 1600.36 [Bold]
- Enter the sales for Day 3: 1801.77 [Bold]

Sales figures:

- Day 1: 1232.37
- Day 2: 1600.36
- Day 3: 1801.77
Stream Manipulators

Table 3-12

<table>
<thead>
<tr>
<th>Stream Manipulator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed</td>
<td>Displays floating-point numbers in fixed point notation.</td>
</tr>
<tr>
<td>ios::scientific</td>
<td>Causes a decimal point and trailing zeros to be displayed, even if there is no fractional part.</td>
</tr>
<tr>
<td>ios::precision()</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>
</tbody>
</table>

Formatted Input

- **Can format field width for use with cin**
- **Useful when reading string data to be stored in a character array:**
  ```cpp
  const int SIZE = 10;
  char firstName[SIZE];
  cout << "Enter your name: ";
  cin >> setw(SIZE) >> firstName;
  ```
- **cin reads one less character than specified with the setw() manipulator**

- **To read an entire line of input, use cin.getline():**
  ```cpp
  const int SIZE = 81;
  char address[SIZE];
  cout << "Enter your address: ";
  cin.getline(address, SIZE);
  ```
  - Name of array to store string
  - Size of the array

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

- Mixing cin >> and 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 cin.ignore():
  ```cpp
  cin.ignore(); // skip next char
  cin.ignore(10, '\n'); // skip the next 10 char. or until a '\n'
  ```

More About Member Functions

- **Member Function**: procedure that is part of an object
- **cout, cin** are objects
- Some member functions of the cin object:
  - getline
  - get
  - ignore

More Mathematical Library Functions

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

More Mathematical Library Functions

- These require cstdlib header file
- `rand()`: returns a random number (int) between 0 and the largest int the computer holds. Yields same sequence of numbers each time program is run.
- `srand(x)`: initializes random number generator with unsigned int x
Hand Tracing a Program

- Hand trace a program: act as if you are the computer, executing a program:
  - step through and ‘execute’ each statement, one-by-one
  - record the contents of variables after statement execution, using a hand trace chart (table)
- Useful to locate logic or mathematical errors

Introduction to File Input and Output

- Can use files instead of keyboard, monitor screen for program input, output
- Allows data to be retained between program runs
- Steps:
  - Open the file
  - Use the file (read from, write to, or both)
  - Close the file

Files: What is Needed

- Use fstream header file for file access
- File stream types:
  - ifstream for input from a file
  - ofstream for output to a file
  - fstream for input from or output to a file
- Define file stream objects:
  - ifstream infile;
  - ofstream outfile;
Opening Files

• Create a link between file name (outside the program) and file stream object (inside the program)
• Use the `open` member function:
  ```cpp
  infile.open("inventory.dat");
  outfile.open("report.txt");
  ```
• Filename may include drive, path info.
• Output file will be created if necessary; existing file will be erased first
• Input file must exist for `open` to work

Using Files

• Can use output file object and `<<` to send data to a file:
  ```cpp
  outfile << "Inventory report"
  ```
• Can use input file object and `>>` to copy data from file to variables:
  ```cpp
  infile >> partNum;
  infile >> qtyInStock >> qtyOnOrder;
  ```

Closing Files

• Use the `close` member function:
  ```cpp
  infile.close();
  outfile.close();
  ```
• Don’t wait for operating system to close files at program end:
  – may be limit on number of open files
  – may be buffered output data waiting to send to file

Program 3.20

```cpp
1 // This program writes data to a file.
2 #include <iostream>
3 #include <fstream>
4 using namespace std;
5
6 int main()
7 {
8   ofstream outFile;
9   outfile.open("demand.txt");
10   cout << "Writing data to the file.
11   // Write 4 great names to the file
12   outFile << "Beethoven"
13   outFile << "Mendelssohn"
14   outFile << "Mozart"
15   outFile << "Schubert"
16   // Close the file
17   outFile.close();
18   return 0;
19 }
```
Program 3.10 (continued)

```c
18  #include <stdio.h>
19  
20  main()
21  {
22    struct name
23    {
24      char *fname; // Name of the file
25      char *name; // Name to be read
26    } n;
27    
28    // Read name from the file
29    printf("Enter the name of the file: ");
30    scanf("%s", n.fname);
31    printf("Enter the name to be read: ");
32    scanf("%s", n.name);
33    
34    // Close the file
35    printf("Closing the file.");
36    close(n.fname);
37    
38    return 0;
```