Standard Version of Starting Out with C++, 4th Edition

Chapter 3

Expressions and Interactivity

Topics

3.1 The `cin` Object

3.2 Mathematical Expressions

3.3 When You Mix Apples and Oranges: *Type Conversion*

3.4 Overflow and Underflow

3.5 Type Casting

3.6 Named Constants

Topics (continued)

3.7 Multiple Assignment and Combined Assignment

3.8 Formatting Output

3.9 Formatted Input

3.10 More About Member Functions

3.11 More Mathematical Library Functions

3.12 Hand Tracing a Program

3.14 Introduction to File Input and Output

3.1 The `cin` Object

- Standard input object
- Like `cout`, requires `iostream` file
- Used to read input from keyboard
- Often used with `cout` to display a user prompt first
- Information retrieved from `cin` with `>>`
- Input information stored in one or more variables

The `cin` Object

- User input goes from keyboard to keyboard buffer
- cin converts information to the type that matches the variable:

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

The `cin` Object

- Can be used to input > 1 value:

```
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

- Can be used to read in a string
- Must first declare an array to hold characters in string:

```
char myName[20];
```

- `myName` is name of array, `20` 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:

```
cin >> myName;
```

3.2 Mathematical Expressions

- Can create complex expressions using multiple mathematical operators
- An expression can be a constant, a variable, or a mathematical combination of constants and variables
- Can be used in assignment, `cout`, other statements:

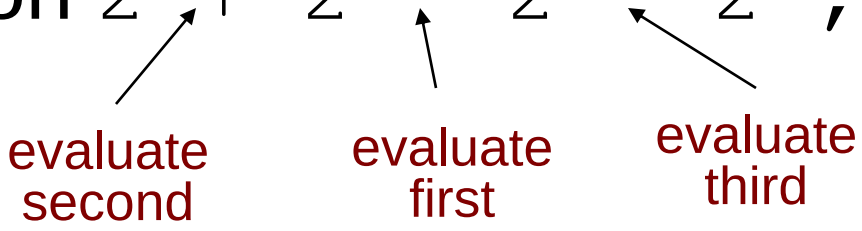
```
area = 2 * PI * radius;
cout << "border is: " << 2*(l+w);
```

Order of Operations

In an expression with > 1 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` ,



Associativity of Operators

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

```
  2 + 2  *  2 - 2  = 4
(2 + 2) *  2 - 2  = 6
  2 + 2  * (2 - 2) = 2
(2 + 2) * (2 - 2) = 0
```

Algebraic Expressions

- Multiplication requires an operator:

  $Area = lw$ is written as `Area = l * 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:

  $$m = \frac{y_2 - y_1}{x_2 - x_1}$$ is written as

  ```
  m = (y2-y1) /(x2-x1);
  ```

3.3 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

3.4 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:

```
float m = static_cast<float>(y2-y1)
                              /(x2-x1);
```

- Useful to see `int` value of a `char` variable:

```
char ch = 'C';
cout << ch << " is "
     << static_cast<int>(ch);
```

C-Style and Prestandard Type Cast Expressions

- C-Style cast: data type name in `()`

```
cout << ch << " is " << (int)ch;
```

- Prestandard C++ cast: value in `()`

```
cout << ch << " is " << int(ch);
```

- Both are still supported in C++, although `static_cast` is preferred

3.6 Named Constants

- Named constant (constant variable): variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:

```
const float TAXRATE = 0.0675;
const int NUMSTATES = 50;
```

- Often named in uppercase letters

`const` vs. `#define`

- `#define` – C-style of naming constants:

```
#define NUMSTATES 50
```

  - Note no ; at end
- Interpreted by pre-processor rather than compiler
- Does not occupy memory location like `const`

3.7 Multiple Assignment and Combined Assignment

- = is an operator that can be used > 1 time in an expression:

```
x = y = z = 5;
```

- Value of = is the value that is assigned
- Associates right to left:

```
x = (y = (z = 5));
```

value is 5 value is 5 value is 5

Combined Assignment

- C++ shorthand for common mathematical operations:

```
sum = sum + newnum;
```

  - Note: not an algebraic equation!
- Operators: `+=, -=, *=, /=, %=`
- `sum += newnum;` short for expression above

3.8 Formatting Output

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

Stream Manipulators

- Used to control features of an output field
- 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
- 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

Manipulator Examples

```
const float e = 2.718;
cout << setw(8) << e;    // 2.718 in a
                         // field 8 wide
cout << setprecision(2);
cout << e;               // 2.7
float price = 25.0, discount = 0.6;
cout << fixed << setprecision(2);
cout << price * discount; // 15.00
```

3.9 Formatted Input

- Can format field width for use with `cin`
- Useful when reading string data to be stored in a character array:

```
char fName[10];
cout << "Enter your name: ";
cin >> setw(10) >> fName;
```

- `cin` reads one less character than specified in `setw()` directive

Formatted Input

- To read an entire line of input, use `cin.getline()`:

```
char address[81];
cout << "Enter your address: ";
cin.getline(address, 81);
```

- `cin.getline` takes two arguments:
  - Name of array to store string
  - Size of the array

Formatted Input

- To read a single character:
  - Use `cin`:

```
char ch;
cout << "Strike any key to continue";
cin >> ch;
```

    Problem: will skip over blanks, tabs, `<CR>`
  - Use `cin.get()`:

```
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()`:

```
cin.ignore(); // skip next char
cin.ignore(10, '\n'); // skip the next
                      // 10 char. or until a '\n'
```

3.10 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`

3.11 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`

3.12 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

3.14 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:

```
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:

```
outfile << "Inventory report";
```

- Can use input file object and >> to copy data from file to variables:

```
infile >> partNum;
infile >> qtyInStock >> qtyOnOrder;
```

Closing Files

- Use the `close` member function:

```
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

Standard Version of Starting Out with C++, 4th Edition

Chapter 3

Expressions and Interactivity