Ch 3: Expressions and Interactivity

- Output and Input
- Formatting Output
- Character Input
- Mathematical Expressions
- Data Type Conversion
- Underflow and Overflow
- Assignment Operators
- Mathematical Library Functions

Output

- **cout** object used to display information to screen
  - **c**onsole **out**put
  - Standard output object

```
#include <iostream>
using namespace std;
```

- Stream insertion operator sends data to console
  - `cout << "Hello world!";`
  - `cout << "The area is " << area;`

arrows point in direction of data flow

Output

- String escape sequence modifies output
  - Backslash with control character
  - Treated as single character
  - **`cout << "Hello world!\n";`**

Table 2-2 Common Escape Sequences

Escape Sequence	Name	Description
\n	Newline	Causes the cursor to go to the next line for subsequent printing.
\t	Horizontal tab	Causes the cursor to skip over to the next tab stop.
\a	Alarm	Causes the computer to beep.
\b	Backspace	Causes the cursor to back up, or move left one position.
\r	Return	Causes the cursor to go to the beginning of the current line, not the next line.
\\	Backslash	Causes a backslash to be printed.
\'	Single quote	Causes a single quotation mark to be printed.
\"	Double quote	Causes a double quotation mark to be printed.

Formatting Output

- Data sent to output can be formatted to display in different ways
  - `#include <iomanip>`
- Manipulators change the way data is displayed

Table 3-12

Stream Manipulator	Description
`setw(n)`	Establishes a print field of *n* spaces.
`fixed`	Displays floating-point numbers in fixed point notation.
`showpoint`	Causes a decimal point and trailing zeroes to be displayed, even if there is no fractional part.
`setprecision(n)`	Sets the precision of floating-point numbers.
`left`	Causes subsequent output to be left justified.
`right`	Causes subsequent output to be right justified.

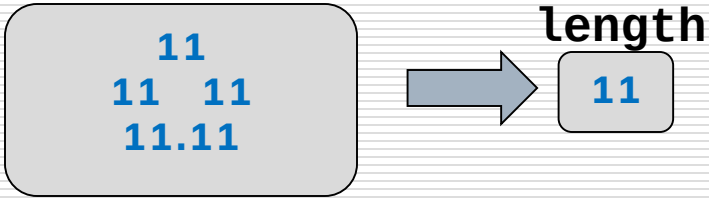
Input

- **cin** object used to read data typed at keyboard
  - console input
  - Standard input object
  - `#include <iostream>`
  - `using namespace std;`
- Stream extraction operator reads data from console (*or keyboard*)
  - `cin >> length;`
  - `cin >> length >> width;`

arrows point in direction of data flow

Input

- Data read should be same as variable data type
  - Stream *may be* put in error state if not
- Each extraction operation reads until whitespace or invalid character (*for data type*) encountered
  - `int length;`
  - `cin >> length;`



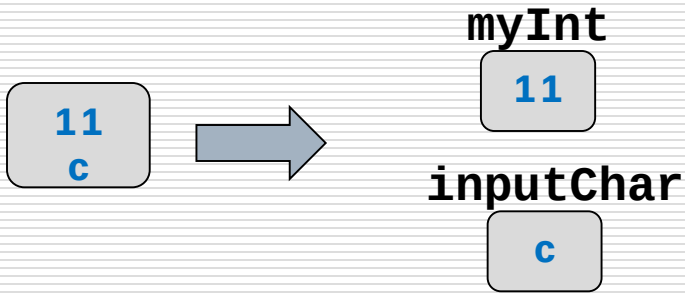
Character Input

- Use of >> with strings will read input until first whitespace
  - Use 'getline' to read entire line
    - `#include <string>`
    - `string inputLine;`
    - `getline(cin, inputLine);`

- Use of >> with character will *not* read whitespace
  - Use 'get' to read *any* single character
    - `char inputChar;`
    - `cin.get(inputChar);`
    - `inputChar = cin.get();`

Character Input

- Use of >> and ‘get’ can cause trailing newline to be left in stream
  - Use ‘cin.ignore’ to remove trailing newline before next read
    - `int myInt;`
    - `char inputChar;`
    - `cin >> myInt;`
    - `cin.ignore();`
    - `cin.get(inputChar);`



Mathematical Expressions

- Combination of constants, variables, and operators that has a value
  - Used for display or storage
- Expression with *multiple* operators are evaluated in steps:
  - higher precedence operators are evaluated before operators with lower precedence
  - same precedence operators are evaluated using associativity

Mathematical Expressions

Appendix C

The operators are shown in order of precedence, from highest to lowest.

Operator	Associativity
`::`	unary: left to right
	binary: right to left
`() [] -> .`	left to right
`++ -- + - ! ~ (type) * & sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

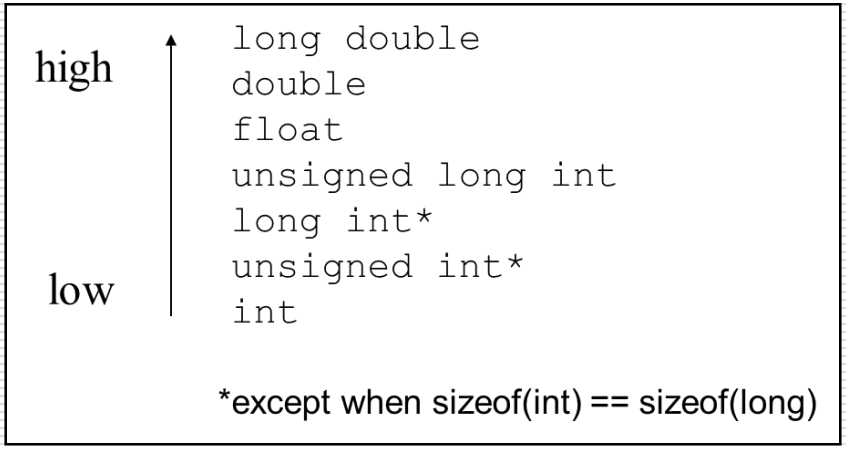
12 + 6 / 3
12 + 2
14

12 + 4 – 7
16 – 7
9

2^4
pow(2.0, 4.0)

Data Type Conversion

- Expressions with different data types will evaluate to a single data type
  - **Implicit** conversion
    - automatically converts a value from one data type to another
    - minimize loss of information, if possible



Data Type Conversion

- Rules of implicit conversion

1. chars, shorts, and unsigned shorts are automatically promoted to int
   - except when sizeof(short) == sizeof(int)
2. an operator working with different data types promotes the lower-ranking type to the higher-ranking type
3. the final value of an expression is converted to the data type of the variable to which it is assigned

```
2 + 2.3F
4.3F
```

```
double d = 4.3F;
```

Data Type Conversion

- Expressions with different data types will evaluate to a single data type
  - **Explicit** conversion/ Type casting
    - programmer inserts statement to explicitly convert a value from one data type to another
    - unary **cast** operator

      ```
      static_cast<DataType>(Value)
      ```

    - C-style

      ```
      (DataType)Value
      ```

    - Prestandard C++

      ```
      DataType(Value)
      ```

  - conversion is only temporary; stored value is not affected

```
static_cast<float>(4.3)
static_cast<float>(2 + 4.3)
```

Overflow and Underflow

- Overflow
  - Variable assigned value that is **too large** for data type
- Underflow
  - Variable assigned value that is **too small** for data type

Program 3-7

```
1  // This program demonstrates integer overflow and underflow.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       // testVar is initialized with the maximum value for a short.
8       short testVar = 32767;
9
10      // Display testVar.
11      cout << testVar << endl;
12
13      // Add 1 to testVar to make it overflow.
14      testVar = testVar + 1;
15      cout << testVar << endl;
16
17      // Subtract 1 from testVar to make it underflow.
18      testVar = testVar - 1;
19      cout << testVar << endl;
20      return 0;
21 }
```

Program Output

```
32767
-32768
32767
```

Assignment Operators

- = simple assignment
  - binary; assigns value of right operand to memory location of left operand
- compound assignment
  - simple assignment operator combined with arithmetic or bitwise operators

Table 3-9

Operator	Example Usage	Equivalent to
+=	`x += 5;`	`x = x + 5;`
-=	`y -= 2;`	`y = y - 2;`
*=	`z *= 10;`	`z = z * 10;`
/=	`a /= b;`	`a = a / b;`
%=	`c %= 3;`	`c = c % 3;`

Mathematical Library Functions

- C++ supports built-in functions for mathematical calculations
  - `#include <cmath>`

Table 3-13

Function	Example	Description
`abs`	`y = abs(x);`	Returns the absolute value of the argument. The argument and the return value are integers.
`cos`	`y = cos(x);`	Returns the cosine of the argument. The argument should be an angle expressed in radians. The return type and the argument are `doubles`.
`exp`	`y = exp(x);`	Computes the exponential function of the argument, which is `x`. The return type and the argument are `doubles`.
`fmod`	`y = fmod(x, z);`	Returns, as a `double`, the remainder of the first argument divided by the second argument. Works like the modulus operator, but the arguments are `doubles`. (The modulus operator only works with integers.) Take care not to pass zero as the second argument. Doing so would cause division by zero.
`log`	`y = log(x);`	Returns the natural logarithm of the argument. The return type and the argument are `doubles`.
`log10`	`y = log10(x);`	Returns the base-10 logarithm of the argument. The return type and the argument are `doubles`.
`sin`	`y = sin(x);`	Returns the sine of the argument. The argument should be an angle expressed in radians. The return type and the argument are `doubles`.
`sqrt`	`y = sqrt(x);`	Returns the square root of the argument. The return type and argument are `doubles`.
`tan`	`y = tan(x);`	Returns the tangent of the argument. The argument should be an angle expressed in radians. The return type and the argument are `doubles`.

Mathematical Library Functions

- C++ supports built-in functions for generating random numbers
  - Seed random number generator with system time
    - `#include <cstdlib> // for srand/rand`
    - `#include <ctime>  // for time func`
    - `unsigned int seed = time(0);`
    - `srand(seed);`

Mathematical Library Functions

- Generate random integer number between 0 and RAND_MAX
  - **`rand()`**
    - RAND_MAX is constant defined in <cstdlib>
- Generate random integer number between *minValue* and *maxValue*
  - **`(rand() % (maxValue – minValue + 1)) + minValue`**

rand() % 100 → range 0 to 99
rand() % 100 + 1 → range 1 to 100
rand() % 30 + 1985 → range 1985 to 2014