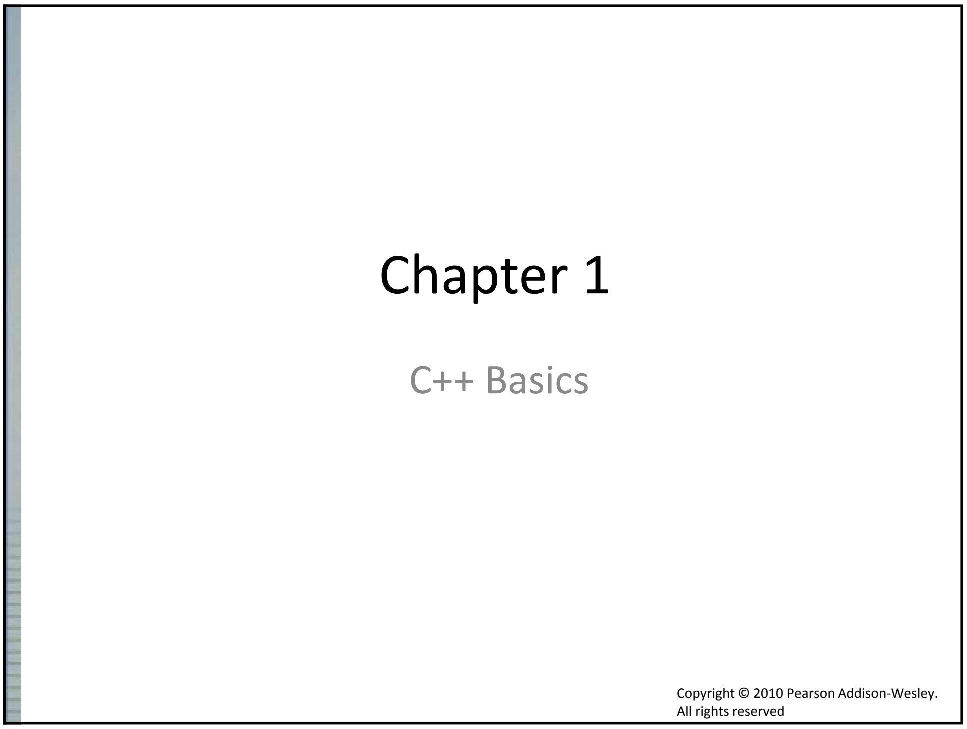
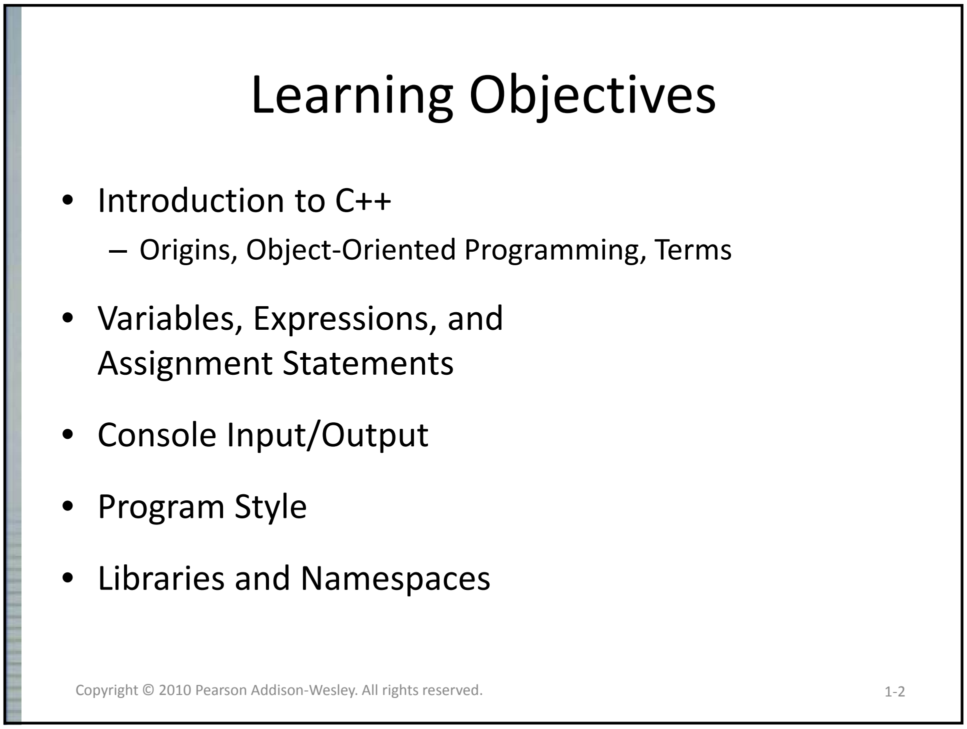


Chapter 1

C++ Basics



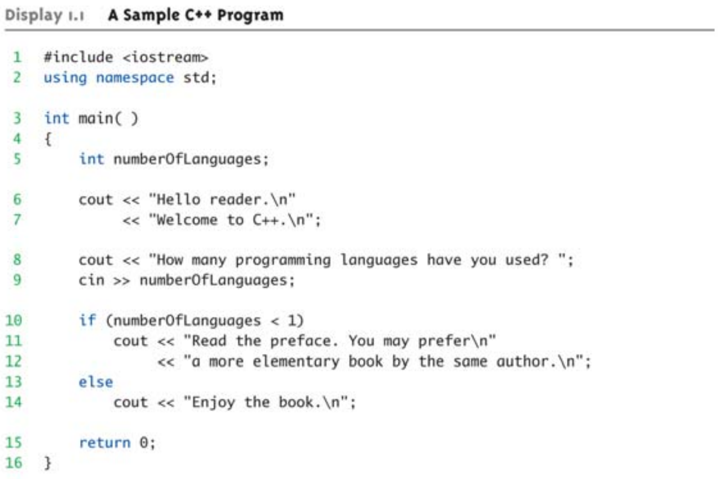
Learning Objectives

- Introduction to C++
  - Origins, Object-Oriented Programming, Terms
- Variables, Expressions, and Assignment Statements
- Console Input/Output
- Program Style
- Libraries and Namespaces

Introduction to C++

- C++ Origins
  - Low-level languages
    - Assembly
  - High-level languages
    - C, C++, Java, COBOL, FORTRAN
  - Object-Oriented-Programming (encapsulation, inheritance, and polymorphism) in C++
- C++ Terminology
  - *Programs* and *functions*
  - Basic Input/Output (I/O) with cin and cout

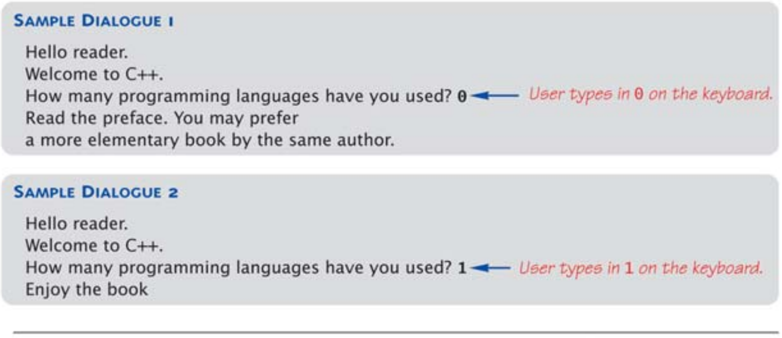
Display 1.1
A Sample C++ Program (1 of 2)



Display 1.1 **A Sample C++ Program**

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

int main( )
{
    int numberOfLanguages;

    cout << "Hello reader.\n"
         << "Welcome to C++.\n";

    cout << "How many programming languages have you used? ";
    cin >> numberOfLanguages;

    if (numberOfLanguages < 1)
        cout << "Read the preface. You may prefer\n"
             << "a more elementary book by the same author.\n";
    else
        cout << "Enjoy the book.\n";

    return 0;
}
```

Display 1.1
A Sample C++ Program (2 of 2)



SAMPLE DIALOGUE 1

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 0 ← User types in 0 on the keyboard.
Read the preface. You may prefer
a more elementary book by the same author.
```

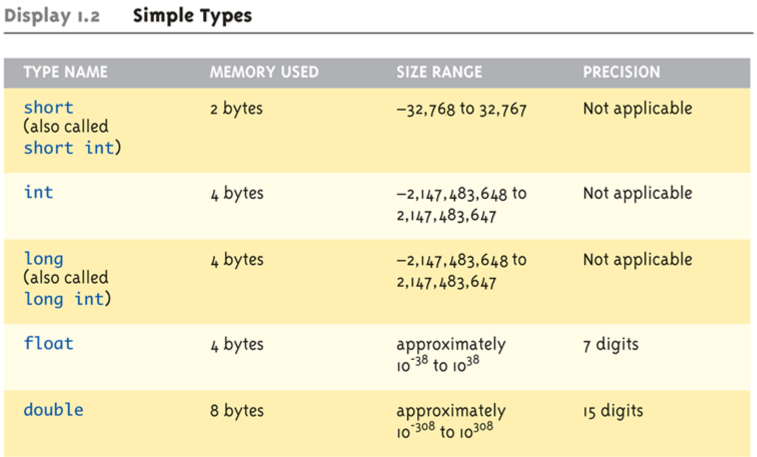
SAMPLE DIALOGUE 2

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 1 ← User types in 1 on the keyboard.
Enjoy the book
```

C++ Variables

- C++ Identifiers
  - Keywords/reserved words vs. Identifiers
    - Start with either a letter or the underscore symbol, and all the rest of the characters must be letters, digits, or the underscore symbol.
  - Case-sensitivity and validity of identifiers
  - Meaningful names!
- Variables
  - A memory location to store data for a program
  - Must declare all variables before use in program

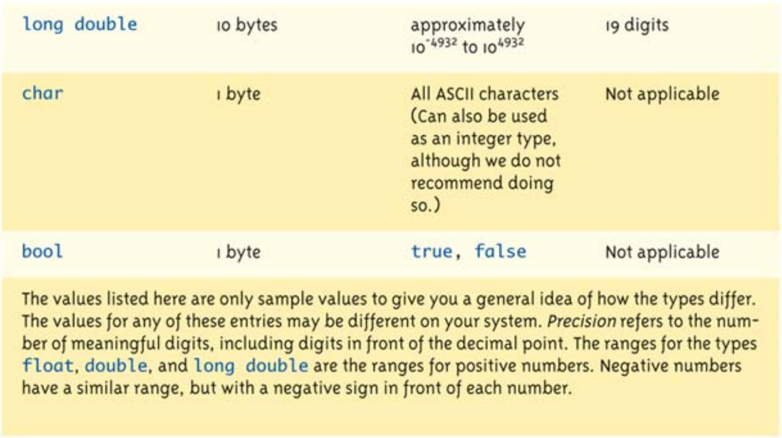
Data Types:
Display 1.2 Simple Types (1 of 2)



Display 1.2 **Simple Types**

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
short (also called short int)	2 bytes	−32,768 to 32,767	Not applicable
int	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
long (also called long int)	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
float	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
double	8 bytes	approximately 10^{-308} to 10^{308}	15 digits

Data Types:
Display 1.2 Simple Types (2 of 2)



long double	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
char	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
bool	1 byte	true, false	Not applicable

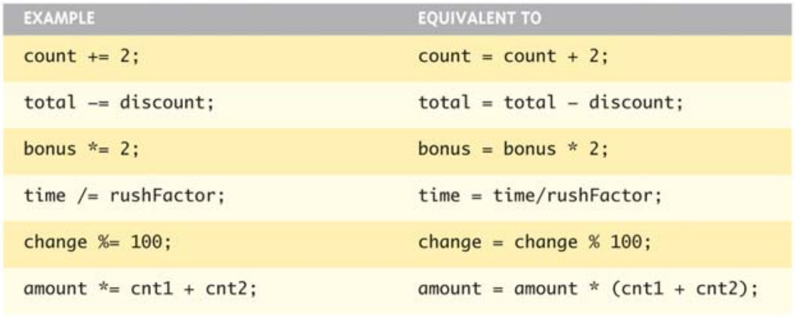
The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types float, double, and long double are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

Assigning Data

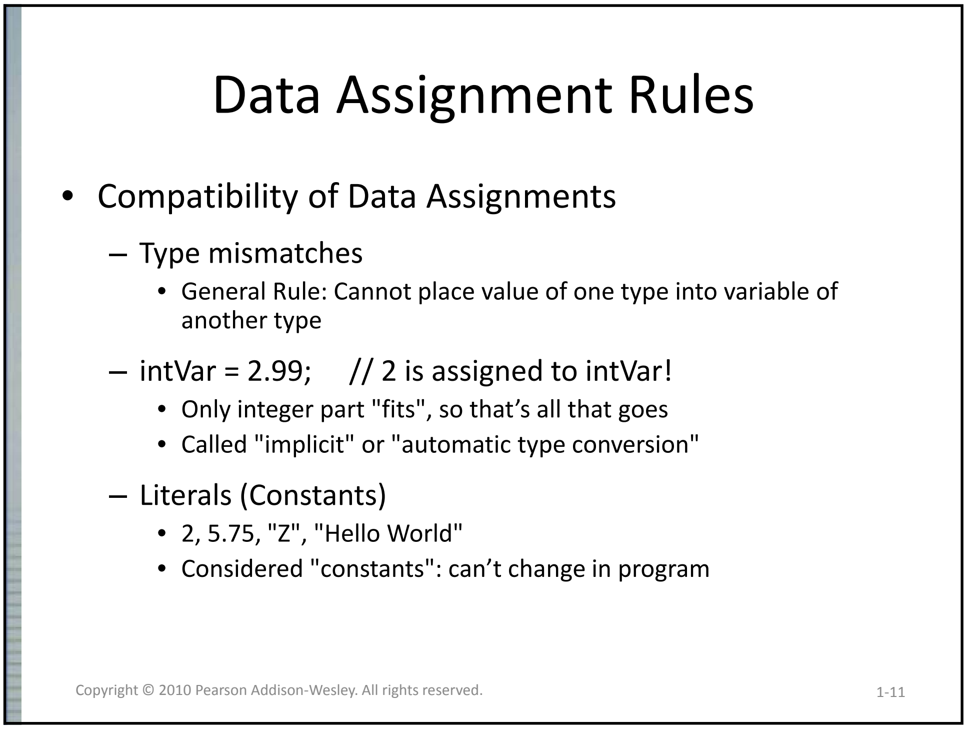
- Initializing data in declaration statement
  - Results "undefined" if you don’t!
    - int myValue = 0;
- Assigning data during execution
  - Lvalues (left-side) & Rvalues (right-side)
    - Lvalues must be variables
    - Rvalues can be any expression
    - Example:
      distance = rate * time;
      Lvalue: "distance"
      Rvalue: "rate * time"
- n = m = 2;

Assigning Data: Shorthand Notations

- Display, page 14

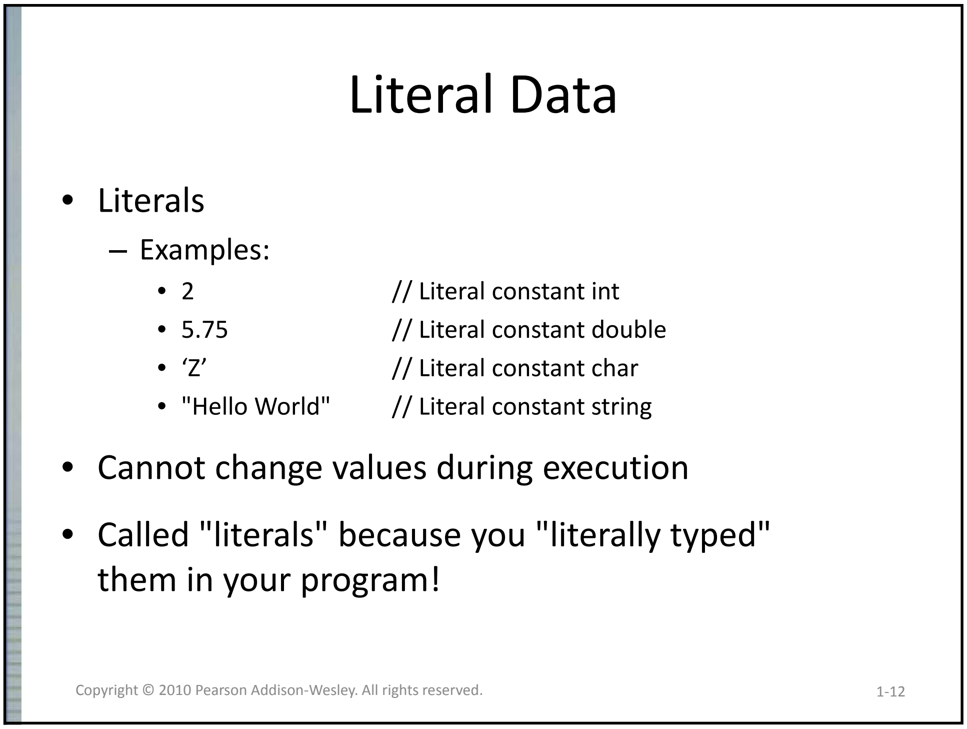


EXAMPLE	EQUIVALENT TO
`count += 2;`	`count = count + 2;`
`total -= discount;`	`total = total - discount;`
`bonus *= 2;`	`bonus = bonus * 2;`
`time /= rushFactor;`	`time = time/rushFactor;`
`change %= 100;`	`change = change % 100;`
`amount *= cnt1 + cnt2;`	`amount = amount * (cnt1 + cnt2);`



Data Assignment Rules

- Compatibility of Data Assignments
  - Type mismatches
    - General Rule: Cannot place value of one type into variable of another type
  - intVar = 2.99;   // 2 is assigned to intVar!
    - Only integer part "fits", so that's all that goes
    - Called "implicit" or "automatic type conversion"
  - Literals (Constants)
    - 2, 5.75, "Z", "Hello World"
    - Considered "constants": can't change in program



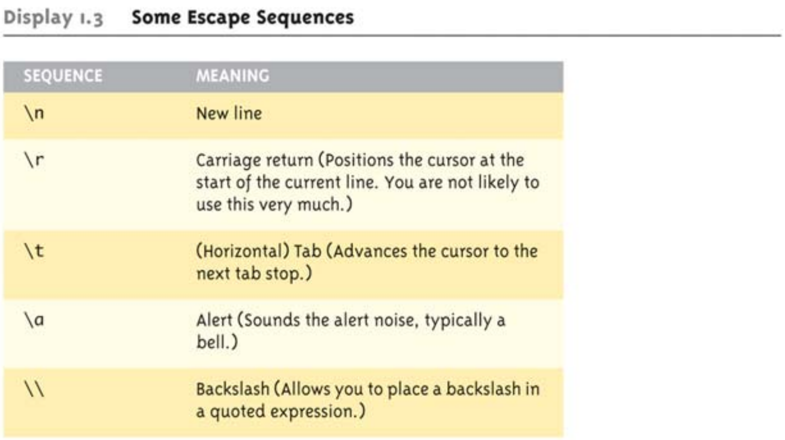
Literal Data

- Literals
  - Examples:
    - 2   // Literal constant int
    - 5.75   // Literal constant double
    - 'Z'   // Literal constant char
    - "Hello World"   // Literal constant string
- Cannot change values during execution
- Called "literals" because you "literally typed" them in your program!

Escape Sequences

- "Extend" character set
- Backslash, \ preceding a character
  - Instructs compiler: a special "escape character" is coming
  - Following character treated as "escape sequence char"
  - Display 1.3 next slide

Display 1.3
Some Escape Sequences (1 of 2)

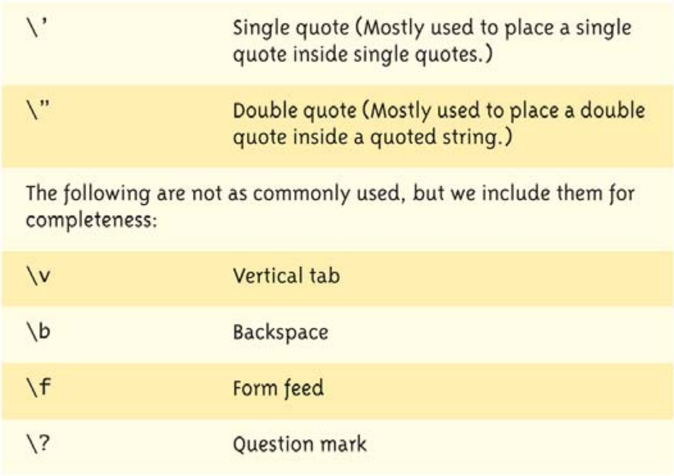


Display 1.3 **Some Escape Sequences**

SEQUENCE	MEANING
\n	New line
\r	Carriage return (Positions the cursor at the start of the current line. You are not likely to use this very much.)
\t	(Horizontal) Tab (Advances the cursor to the next tab stop.)
\a	Alert (Sounds the alert noise, typically a bell.)
\\	Backslash (Allows you to place a backslash in a quoted expression.)

Display 1.3

Some Escape Sequences (2 of 2)



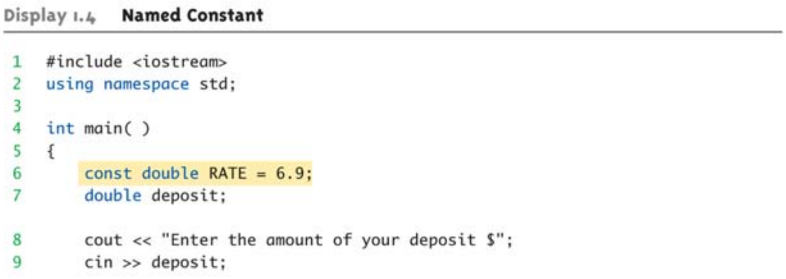
\'	Single quote (Mostly used to place a single quote inside single quotes.)
\"	Double quote (Mostly used to place a double quote inside a quoted string.)
The following are not as commonly used, but we include them for completeness:	
\v	Vertical tab
\b	Backspace
\f	Form feed
\?	Question mark

Constants

- Naming your constants
  - Literal constants are "OK", but provide little meaning
    - e.g., seeing 24 in a pgm, tells nothing about what it represents
- Use named constants instead
  - Meaningful name to represent data
    const int NUMBER_OF_STUDENTS = 57;
    - Called a "declared constant" or "named constant"
    - Now use it’s name wherever needed in program
    - Added benefit: changes to value result in one fix

Arithmetic Operators: **Display 1.4** Named Constant (1 of 2)

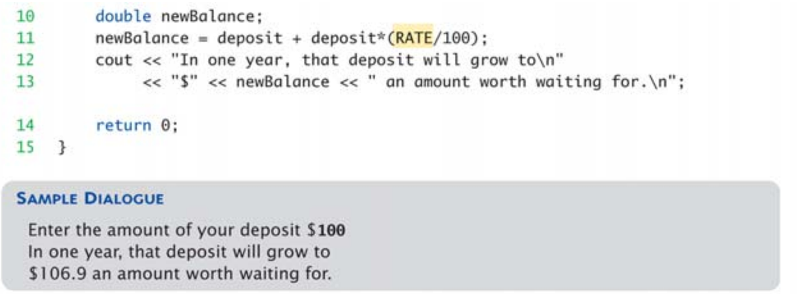
- Standard Arithmetic Operators
  - Precedence rules – standard rules



Display 1.4 **Named Constant**

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

int main( )
{
    const double RATE = 6.9;
    double deposit;

    cout << "Enter the amount of your deposit $";
    cin >> deposit;
```

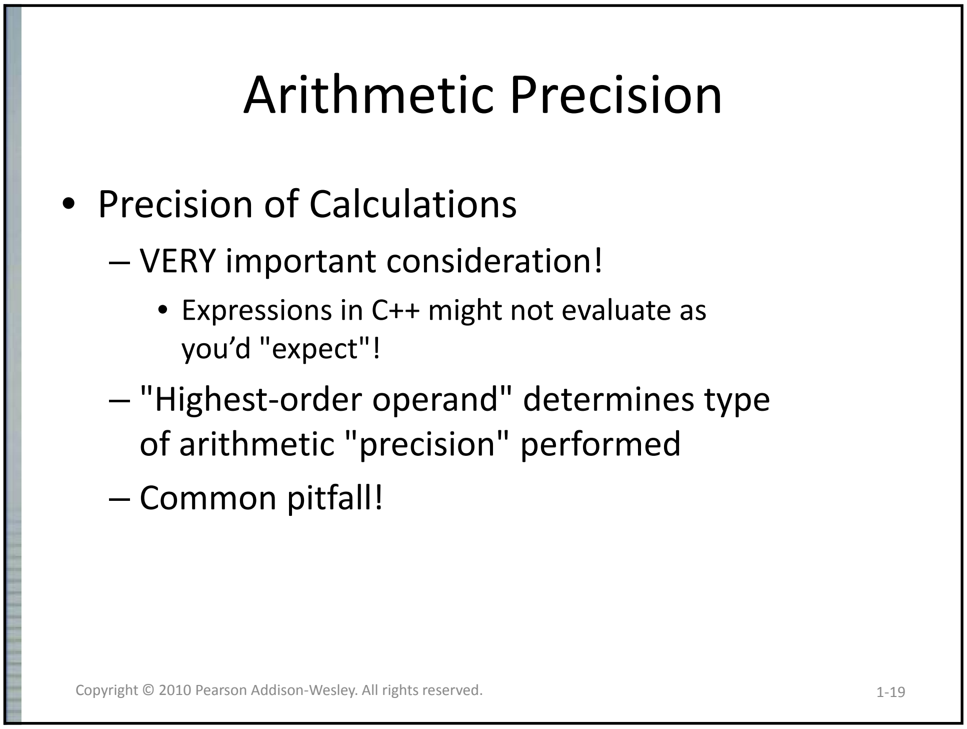
Arithmetic Operators: **Display 1.4** Named Constant (2 of 2)



```
    double newBalance;
    newBalance = deposit + deposit*(RATE/100);
    cout << "In one year, that deposit will grow to\n"
         << "$" << newBalance << " an amount worth waiting for.\n";

    return 0;
}
```

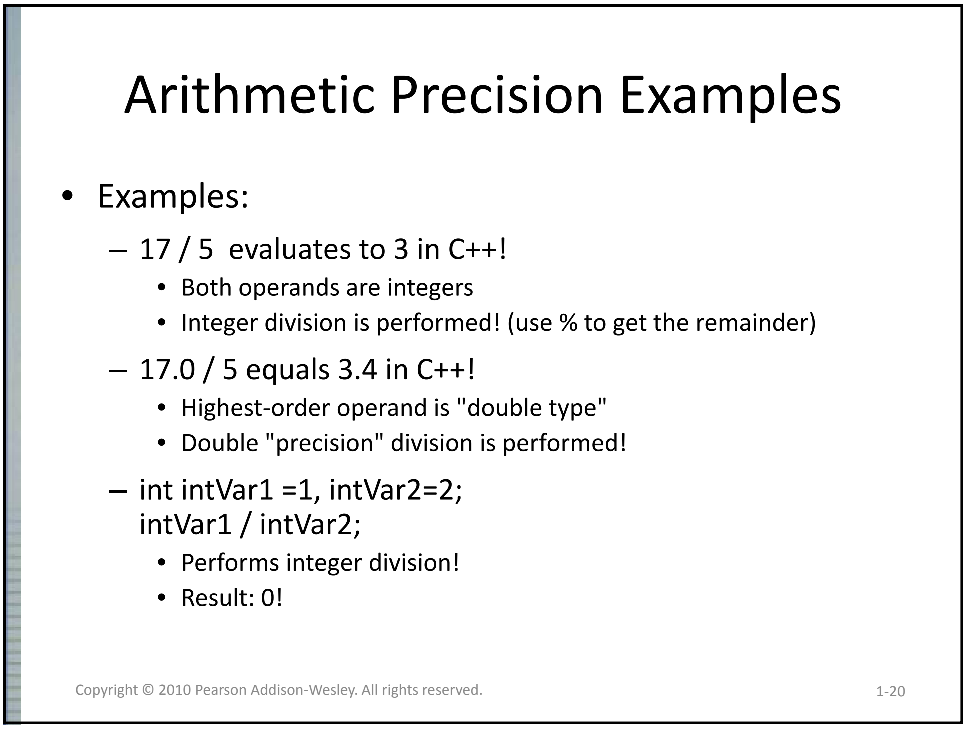
Sample Dialogue

Enter the amount of your deposit $**100**
In one year, that deposit will grow to
$106.9 an amount worth waiting for.



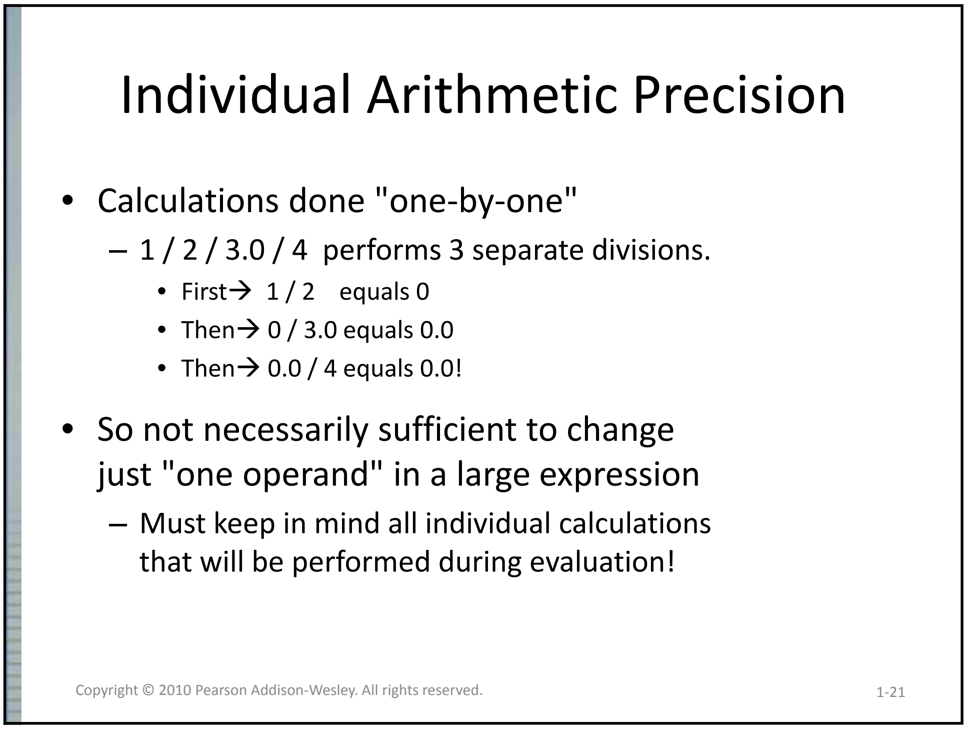
Arithmetic Precision

- Precision of Calculations
  - VERY important consideration!
    - Expressions in C++ might not evaluate as you'd "expect"!
  - "Highest-order operand" determines type of arithmetic "precision" performed
  - Common pitfall!



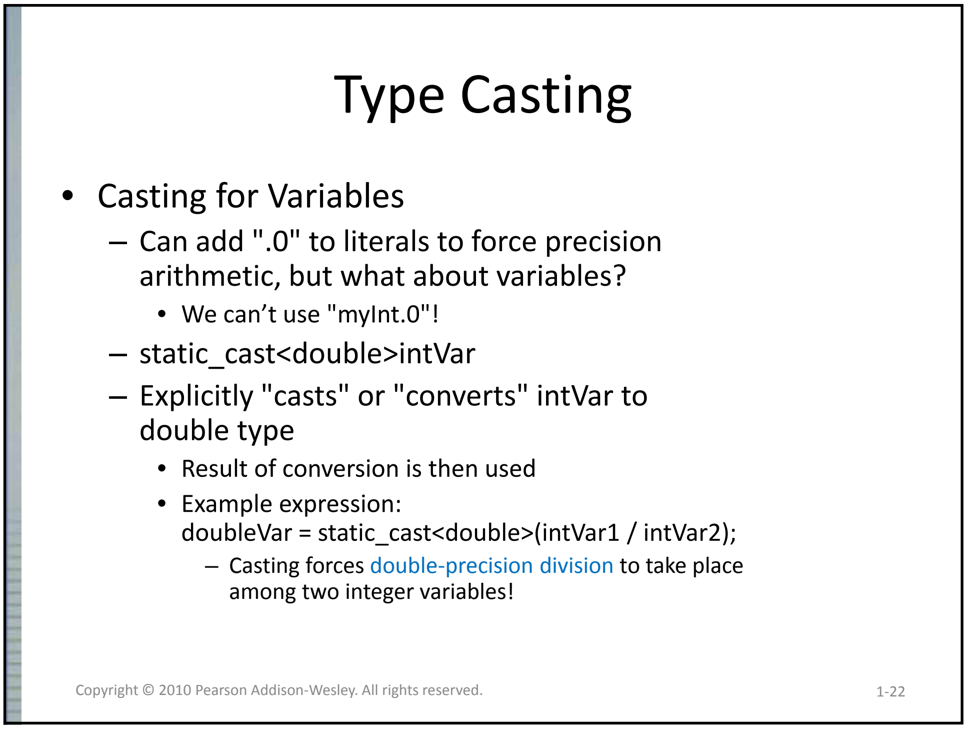
Arithmetic Precision Examples

- Examples:
  - 17 / 5 evaluates to 3 in C++!
    - Both operands are integers
    - Integer division is performed! (use % to get the remainder)
  - 17.0 / 5 equals 3.4 in C++!
    - Highest-order operand is "double type"
    - Double "precision" division is performed!
  - int intVar1 =1, intVar2=2;
    intVar1 / intVar2;
    - Performs integer division!
    - Result: 0!



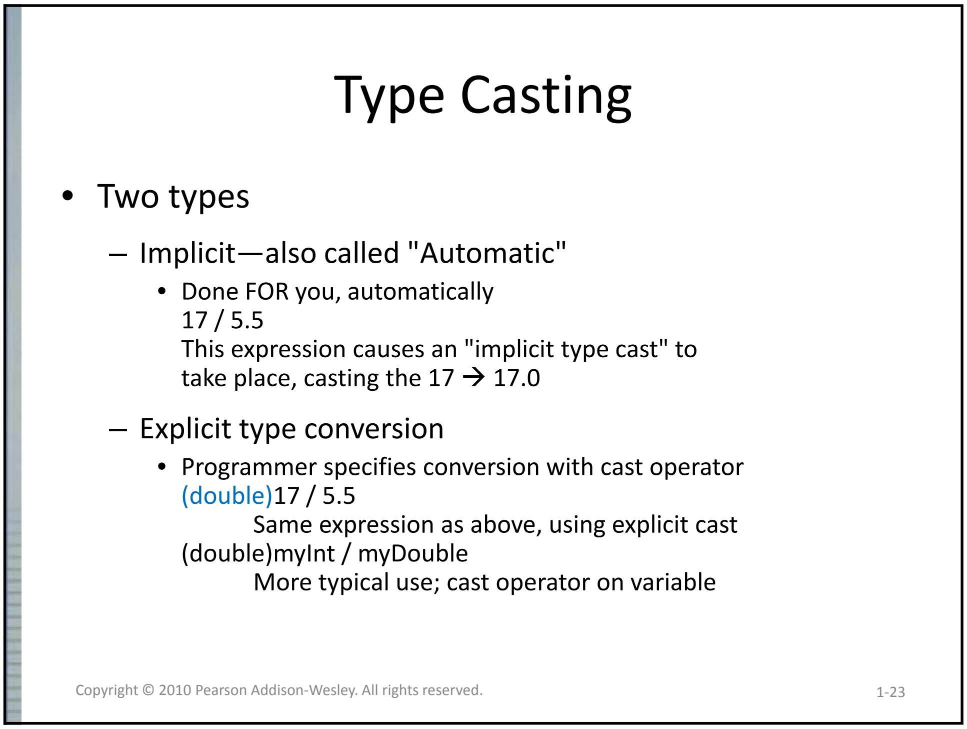
Individual Arithmetic Precision

- Calculations done "one-by-one"
  - 1 / 2 / 3.0 / 4 performs 3 separate divisions.
    - First→ 1 / 2   equals 0
    - Then→ 0 / 3.0 equals 0.0
    - Then→ 0.0 / 4 equals 0.0!
- So not necessarily sufficient to change just "one operand" in a large expression
  - Must keep in mind all individual calculations that will be performed during evaluation!



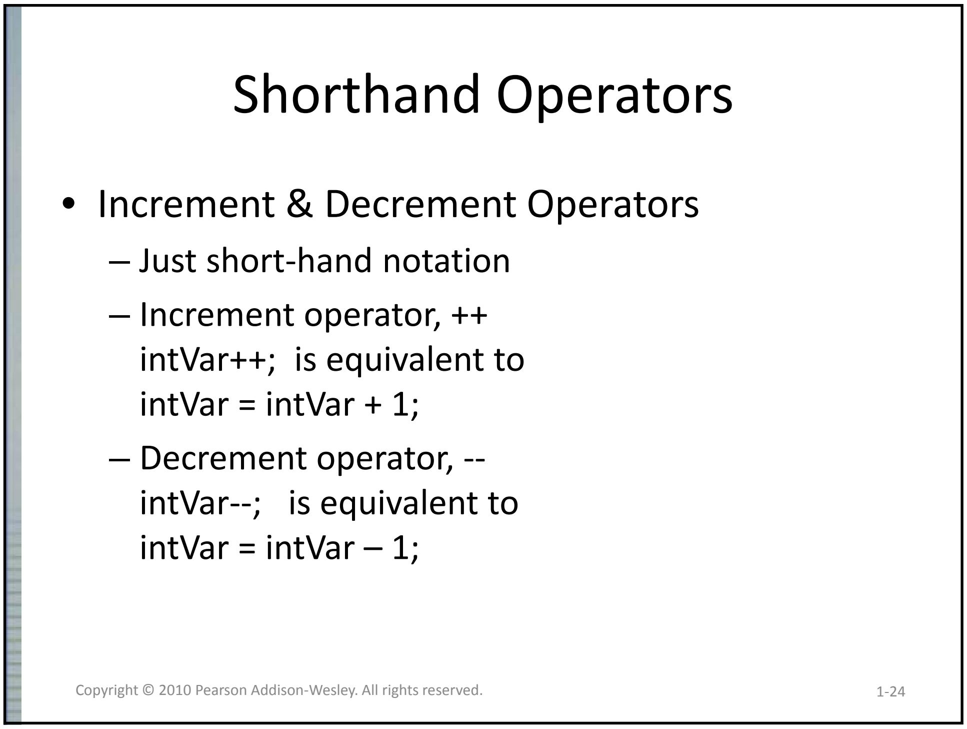
Type Casting

- Casting for Variables
  - Can add ".0" to literals to force precision arithmetic, but what about variables?
    - We can't use "myInt.0"!
  - static_cast<double>intVar
  - Explicitly "casts" or "converts" intVar to double type
    - Result of conversion is then used
    - Example expression:
      doubleVar = static_cast<double>(intVar1 / intVar2);
      - Casting forces double-precision division to take place among two integer variables!



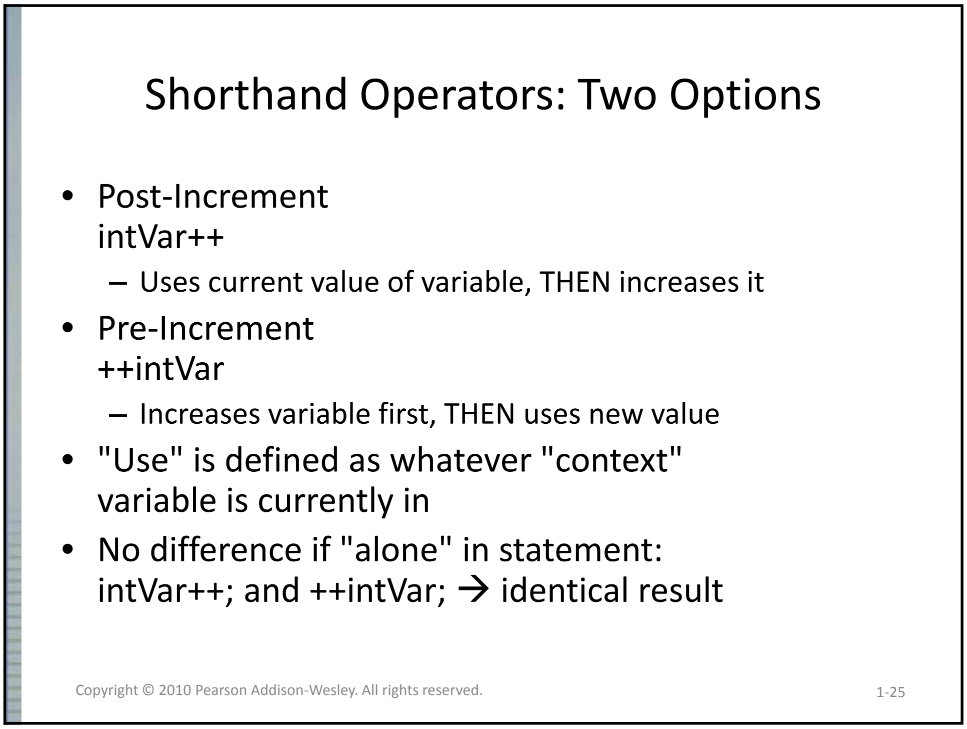
Type Casting

- Two types
  - Implicit—also called "Automatic"
    - Done FOR you, automatically
      17 / 5.5
      This expression causes an "implicit type cast" to take place, casting the 17 → 17.0
  - Explicit type conversion
    - Programmer specifies conversion with cast operator
      (double)17 / 5.5
      Same expression as above, using explicit cast
      (double)myInt / myDouble
      More typical use; cast operator on variable



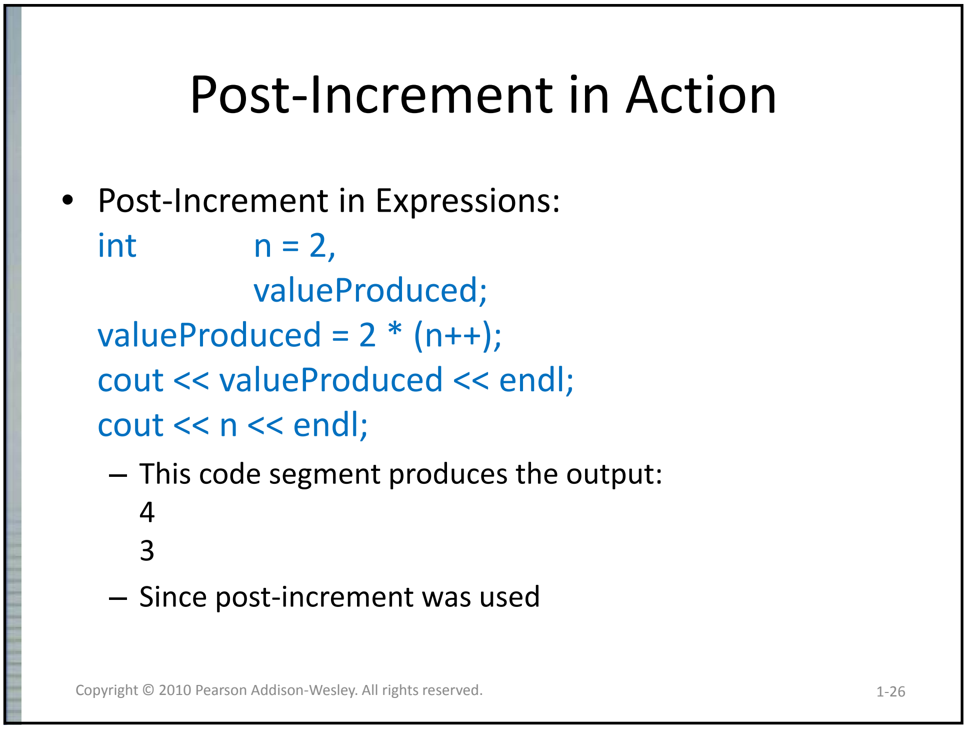
Shorthand Operators

- Increment & Decrement Operators
  - Just short-hand notation
  - Increment operator, ++
    intVar++; is equivalent to
    intVar = intVar + 1;
  - Decrement operator, --
    intVar--;   is equivalent to
    intVar = intVar – 1;



Shorthand Operators: Two Options

- Post-Increment
  intVar++
  - Uses current value of variable, THEN increases it
- Pre-Increment
  ++intVar
  - Increases variable first, THEN uses new value
- "Use" is defined as whatever "context" variable is currently in
- No difference if "alone" in statement:
  intVar++; and ++intVar; → identical result



Post-Increment in Action

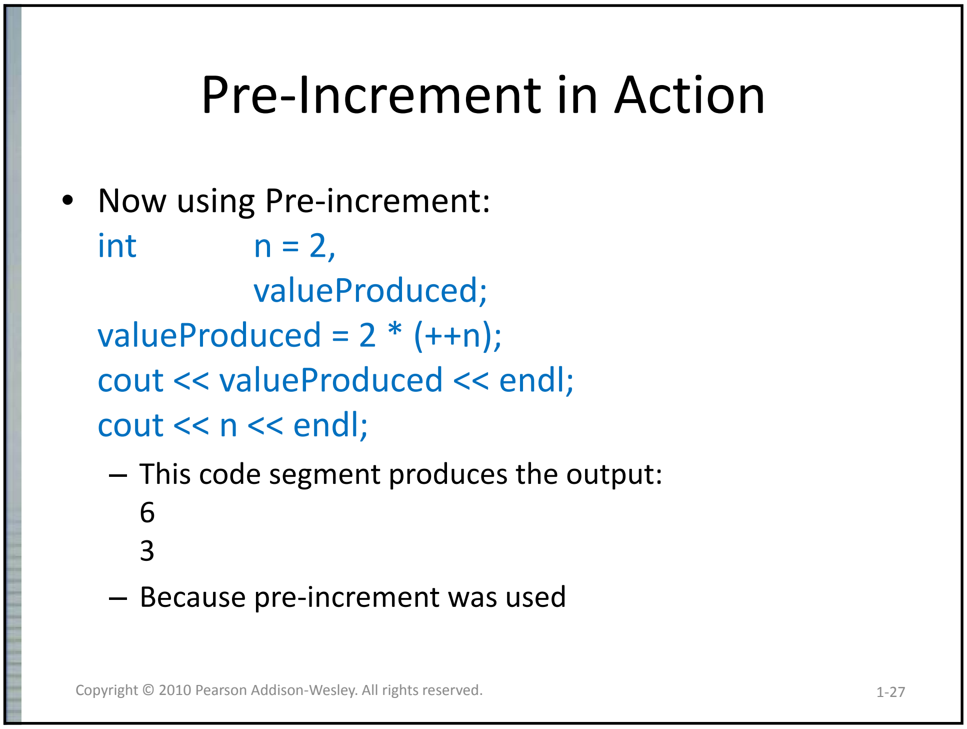
- Post-Increment in Expressions:

```
int        n = 2,
           valueProduced;
valueProduced = 2 * (n++);
cout << valueProduced << endl;
cout << n << endl;
```

  - This code segment produces the output:

```
4
3
```

  - Since post-increment was used



Pre-Increment in Action

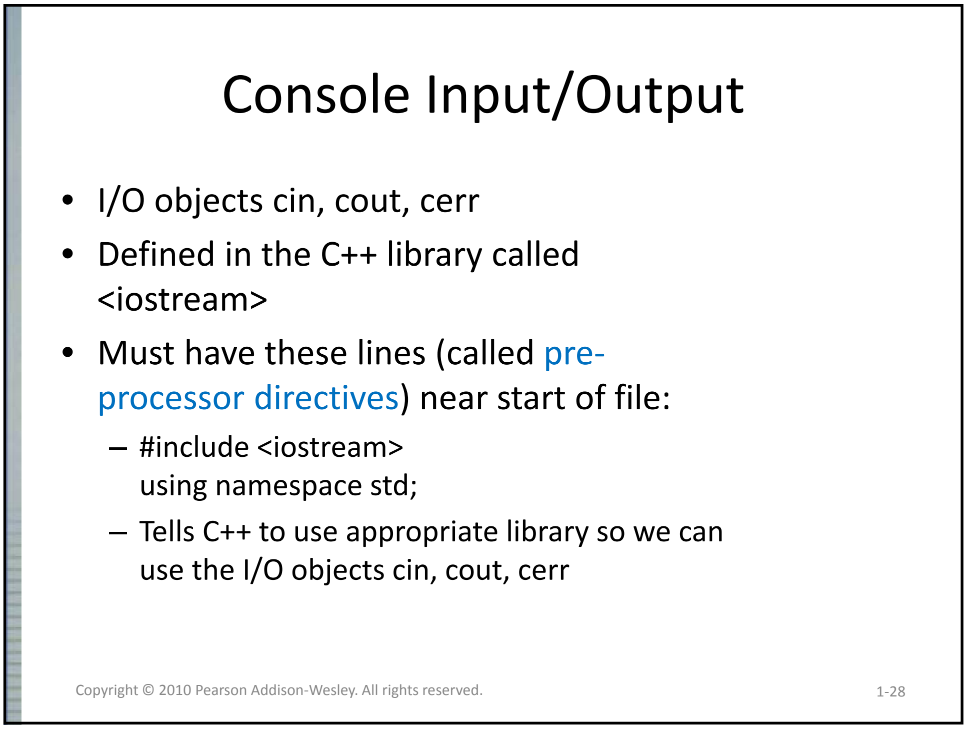
- Now using Pre-increment:

```
int             n = 2,
                valueProduced;
valueProduced = 2 * (++n);
cout << valueProduced << endl;
cout << n << endl;
```

  - This code segment produces the output:

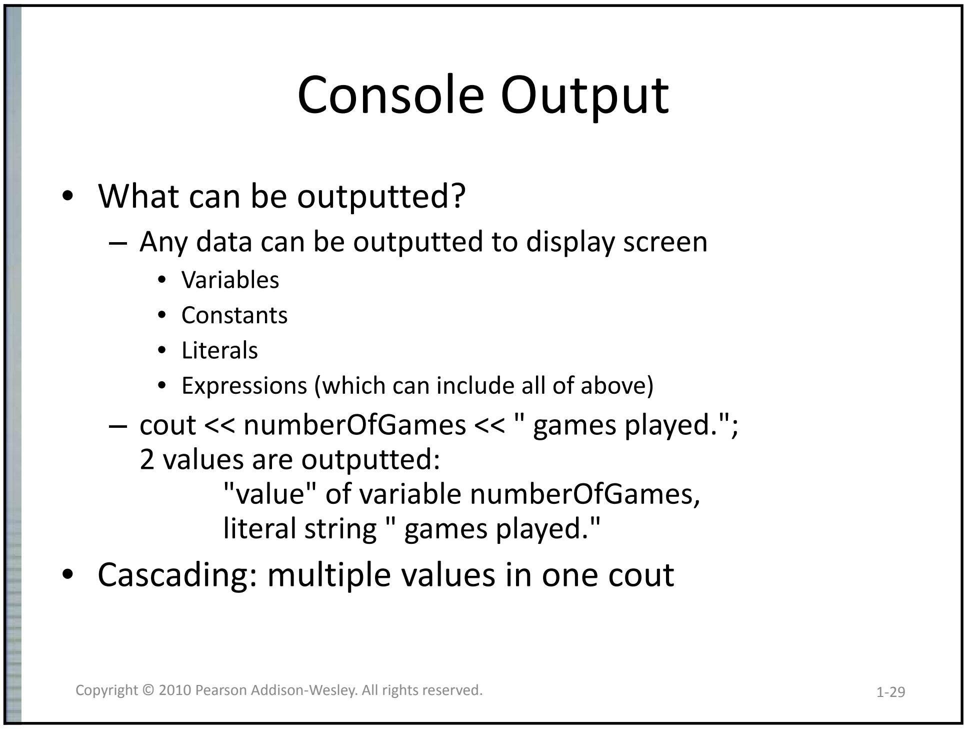
```
6
3
```

  - Because pre-increment was used



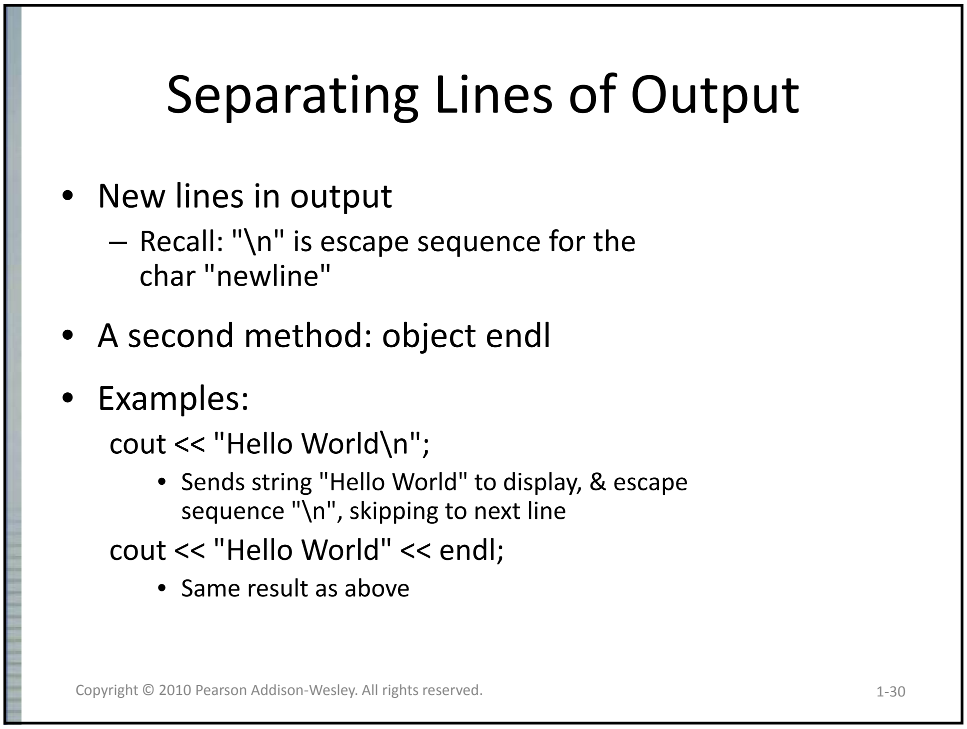
Console Input/Output

- I/O objects cin, cout, cerr
- Defined in the C++ library called <iostream>
- Must have these lines (called pre-processor directives) near start of file:
  - #include <iostream>
    using namespace std;
  - Tells C++ to use appropriate library so we can use the I/O objects cin, cout, cerr



Console Output

- What can be outputted?
  - Any data can be outputted to display screen
    - Variables
    - Constants
    - Literals
    - Expressions (which can include all of above)
  - cout << numberOfGames << " games played.";
    2 values are outputted:
    "value" of variable numberOfGames,
    literal string " games played."
- Cascading: multiple values in one cout

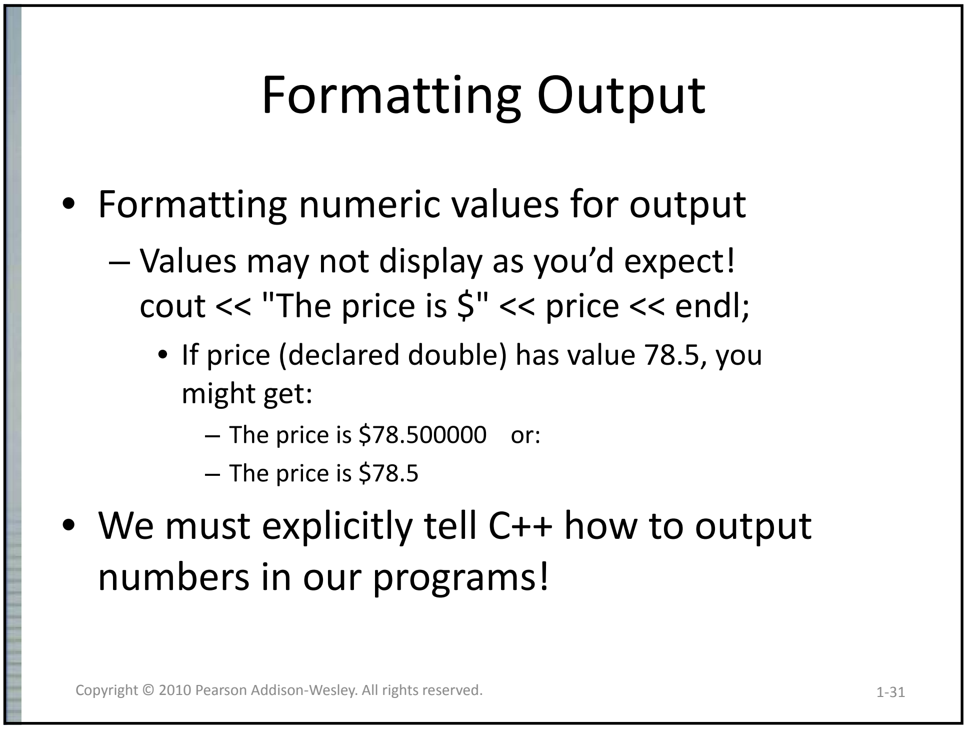


Separating Lines of Output

- New lines in output
  - Recall: "\n" is escape sequence for the char "newline"
- A second method: object endl
- Examples:

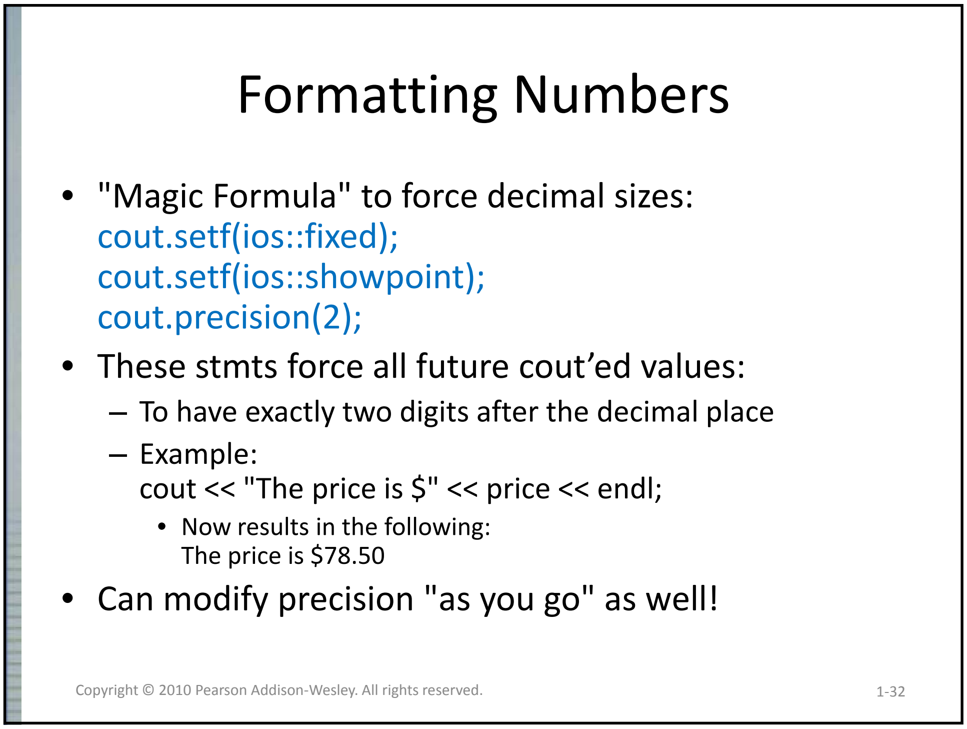
  cout << "Hello World\n";
    - Sends string "Hello World" to display, & escape sequence "\n", skipping to next line

  cout << "Hello World" << endl;
    - Same result as above



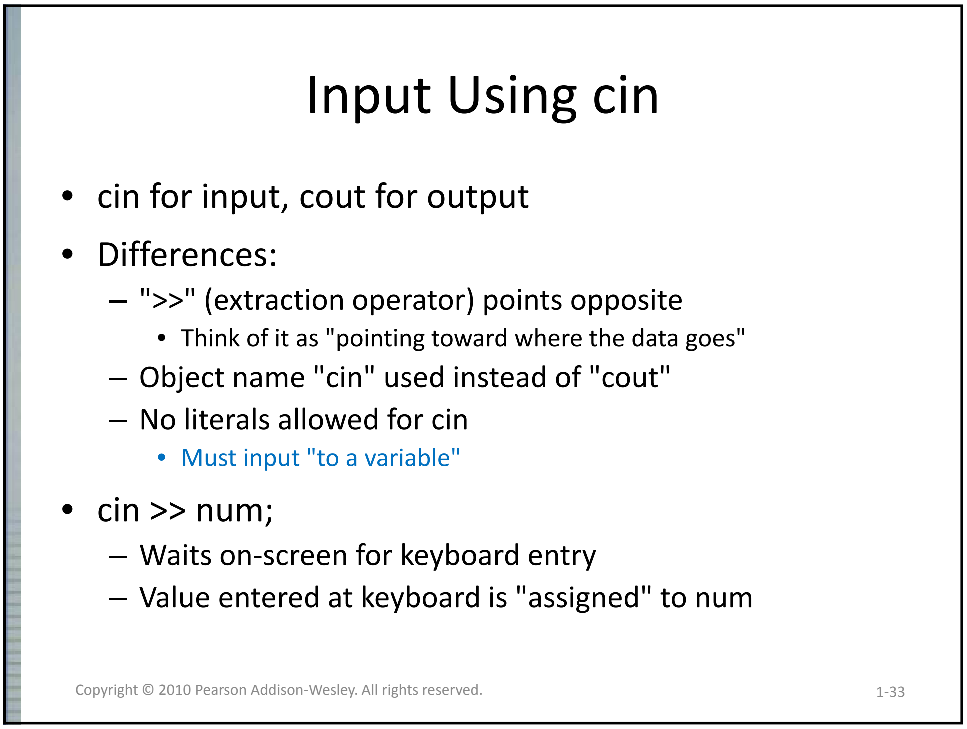
Formatting Output

- Formatting numeric values for output
  - Values may not display as you'd expect!
    cout << "The price is $" << price << endl;
    - If price (declared double) has value 78.5, you might get:
      - The price is $78.500000 or:
      - The price is $78.5
- We must explicitly tell C++ how to output numbers in our programs!



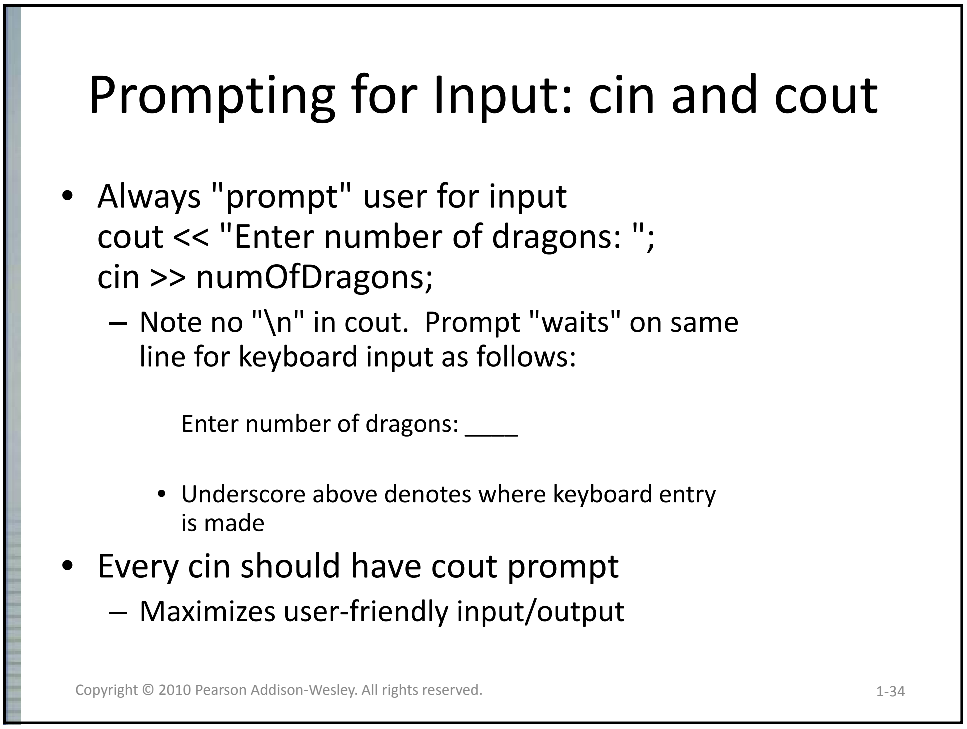
Formatting Numbers

- "Magic Formula" to force decimal sizes:
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(2);
- These stmts force all future cout'ed values:
  - To have exactly two digits after the decimal place
  - Example:
    cout << "The price is $" << price << endl;
    - Now results in the following:
      The price is $78.50
- Can modify precision "as you go" as well!



Input Using cin

- cin for input, cout for output
- Differences:
  - ">>" (extraction operator) points opposite
    - Think of it as "pointing toward where the data goes"
  - Object name "cin" used instead of "cout"
  - No literals allowed for cin
    - Must input "to a variable"
- cin >> num;
  - Waits on-screen for keyboard entry
  - Value entered at keyboard is "assigned" to num

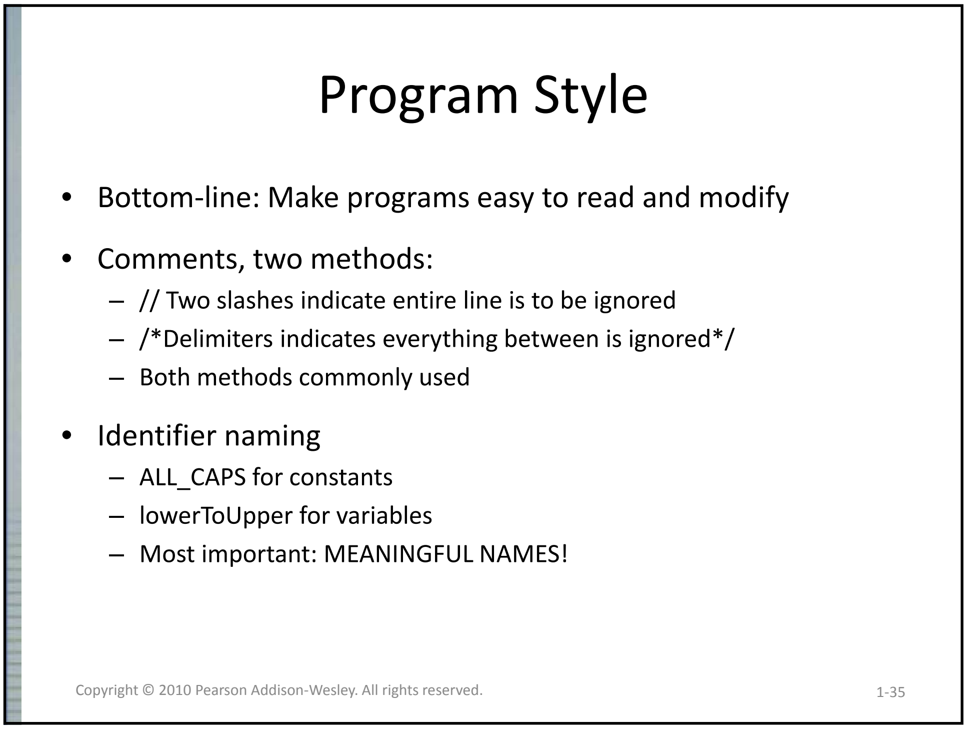


Prompting for Input: cin and cout

- Always "prompt" user for input
  cout << "Enter number of dragons: ";
  cin >> numOfDragons;
  - Note no "\n" in cout. Prompt "waits" on same line for keyboard input as follows:

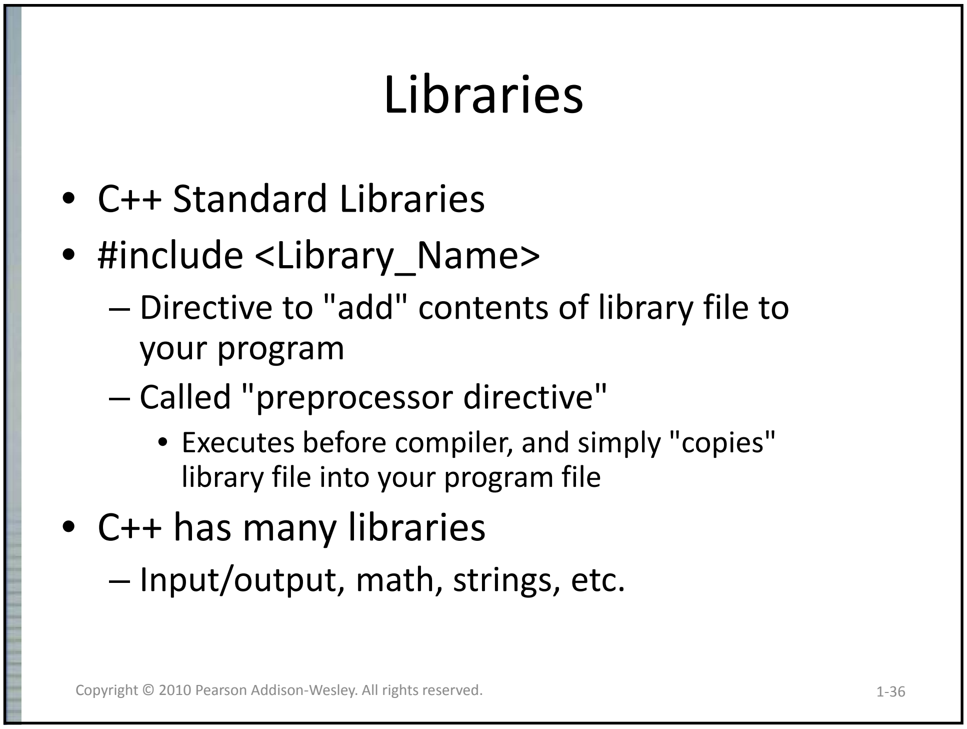
```
Enter number of dragons: ____
```

    - Underscore above denotes where keyboard entry is made
- Every cin should have cout prompt
  - Maximizes user-friendly input/output



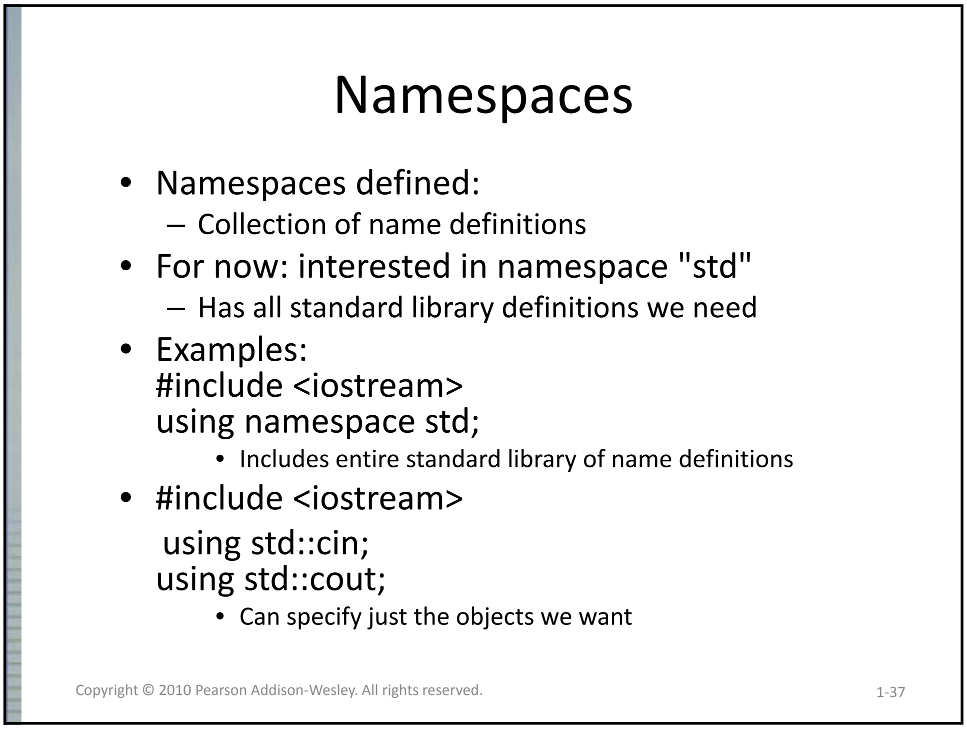
Program Style

- Bottom-line: Make programs easy to read and modify
- Comments, two methods:
  - // Two slashes indicate entire line is to be ignored
  - /*Delimiters indicates everything between is ignored*/
  - Both methods commonly used
- Identifier naming
  - ALL_CAPS for constants
  - lowerToUpper for variables
  - Most important: MEANINGFUL NAMES!



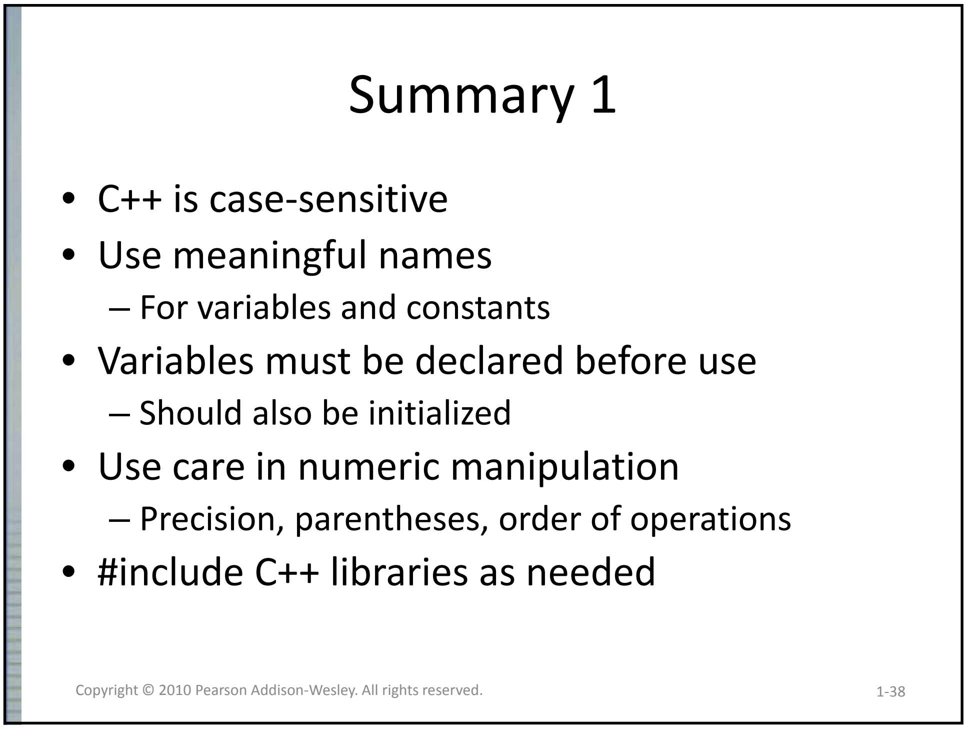
Libraries

- C++ Standard Libraries
- #include <Library_Name>
  - Directive to "add" contents of library file to your program
  - Called "preprocessor directive"
    - Executes before compiler, and simply "copies" library file into your program file
- C++ has many libraries
  - Input/output, math, strings, etc.



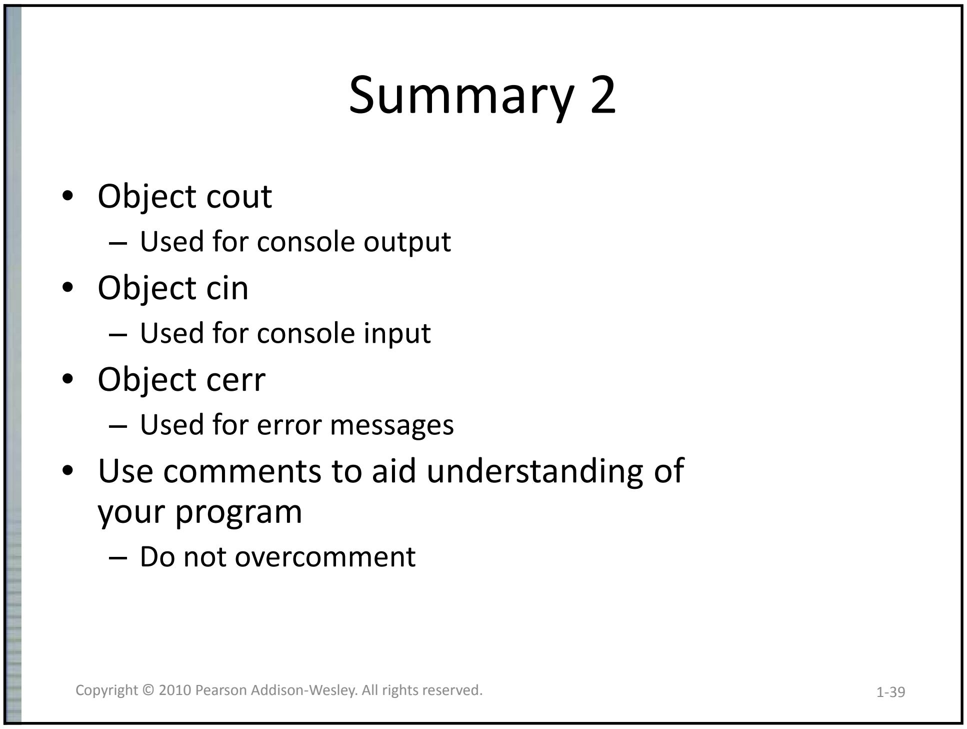
Namespaces

- Namespaces defined:
  - Collection of name definitions
- For now: interested in namespace "std"
  - Has all standard library definitions we need
- Examples:
  #include <iostream>
  using namespace std;
    - Includes entire standard library of name definitions
- #include <iostream>
  using std::cin;
  using std::cout;
    - Can specify just the objects we want



Summary 1

- C++ is case-sensitive
- Use meaningful names
  - For variables and constants
- Variables must be declared before use
  - Should also be initialized
- Use care in numeric manipulation
  - Precision, parentheses, order of operations
- #include C++ libraries as needed



Summary 2

- Object cout
  - Used for console output
- Object cin
  - Used for console input
- Object cerr
  - Used for error messages
- Use comments to aid understanding of your program
  - Do not overcomment