C++ BASICS & DATA TYPES

5TH WEEK LECTURE

엄현상(Eom, Hyeonsang)
School of Computer Science and Engineering
Seoul National University

Outline

- C++ Basics
    - Imperative Language (vs. Declarative Lang.)
    - Object-Oriented Design
    - Six Phases of C++ Programs
    - Examples
- Data Types

Imperative Language

- Language for Computation in Terms of Statements That Change a Program State
  - Expressing Commands to Take Action
  - Defining Sequences of Commands for the Computer to Perform

- Procedural Programming Language
  - Structured Programming Language
  - Modular Programming Language
    - Object-Oriented Programming Languages as extended ones

Declarative Language for Expressing What the Program Should Accomplish as Opposed to Imperative Language for Expressing How

Objects

- Reusable Software Components That Model Real-World Items
  : e.g., Babies, Cars, etc.
  - Have Attributes
    - Size, shape, color, weight, etc.
  - Exhibit Behaviors
    - Babies cry, crawl, sleep, etc.; cars accelerate, brake, turn, etc.

Object-Oriented Design (OOD)

- Modeling Real-World Objects in Software

- Modeling Communication among Objects

- Encapsulating Attributes and Operations (Behaviors)
  - Information Hiding
  - Communication through Well-Defined Interfaces

Object-Oriented Analysis and Design (OOAD)

- Analyzing Program Requirements, Then Developing Solution
  - Essential for Large Programs
  - Planning in Pseudocode or UML
    - UML (Unified Modeling Lang.: currently, ver. 2)
      - Graphical representation scheme used to approach OOAD
        » Enabling developers to model object-oriented systems
      - Flexible and extendible
      - Object Management Group (OMG) supervised

Object-Oriented Language

- Programming in Object-Oriented Languages Is Called Object-Oriented Programming (OOP)
- Allowing Programmers to Create User-Defined Types Called Classes
  - Containing Data Members (Attributes) and Member Functions (Behaviors)

C++

- Object-Oriented Programming Language
  - C++ Programs Built from Pieces Called Classes and Functions
    - User-defined ones
    - C++ Standard Library
      - Rich collections of existing classes and functions
        » Reusable in new applications
    - Various popular third-party libraries

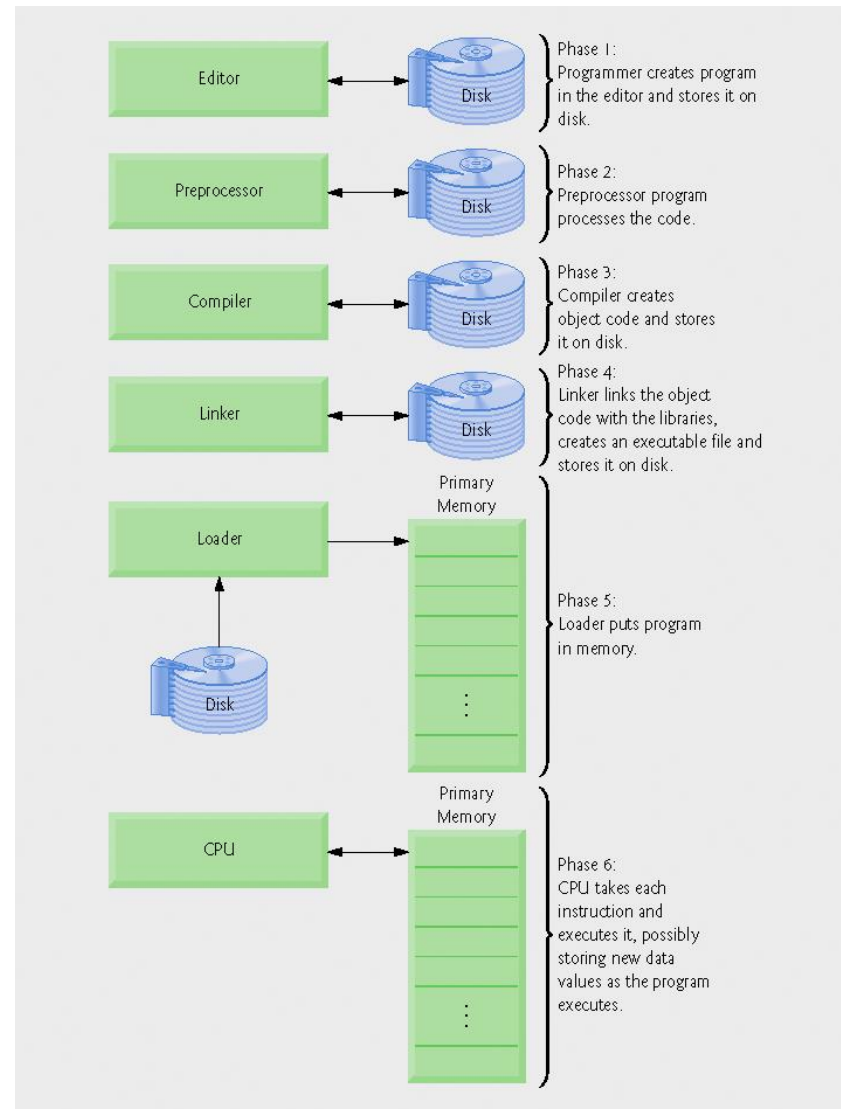Reusable Software
Possibly More Efficient

Six Phases of C++ Programs

- Edit
  - Writing Program (and Storing Source Code on Disk)
- Preprocess
  - Performing Certain Manipulations Before Compilation
- Compile
  - Translating C++ Programs into Machine Languages

Six Phases of C++ Programs Cont'd

- Link
  - Linking Object Code with Missing Functions and Data
- Load
  - Transferring Executable Image to Memory
- Execute
  - Executing the Program One Instruction at a Time

Six Phases of C++ Programs Cont'd

		Phase 1: Programmer creates program in the editor and stores it on disk.
Editor	Disk	
Preprocessor	Disk	Phase 2: Preprocessor program processes the code.
Compiler	Disk	Phase 3: Compiler creates object code and stores it on disk.
Linker	Disk	Phase 4: Linker links the object code with the libraries, creates an executable file and stores it on disk.
Loader	Primary Memory	Phase 5: Loader puts program in memory.
	Disk	
CPU	Primary Memory	Phase 6: CPU takes each instruction and executes it, possibly storing new data values as the program executes.

Examples of C++ Programs

- Five Examples Demonstrate:
  - How to Display Messages on the Screen
  - How to Obtain Information from the User
  - How to Perform Arithmetic Calculations
  - How to Make Decisions by Comparing Numbers
    - Equality and Relational Operators

Printing a Line of Text

- Simple Program
  - Printing a Line of Text
  - Illustrating Several Important Features of C++

Printing a Line of Text Cont'd

- Comments
  - Explaining Programs to Programmers
  - Improving Program Readability
  - Ignored by Compiler
  - Single-Line Comment
    - Beginning with //
    - Example
      - // This is a text-printing program.
  - Multi-Line Comment
    - Starting with /*
    - Ending with */

Printing a Line of Text Cont'd

```
1   // Fig. 2.1: fig02_01.cpp
2   // Text-printing program.
3   #include <iostream>
4
5   // function main be
6   int main()
7   {
8       std::cout << "Welcome to
9
10      return 0; // in
11
12  } // end function main
```

```
Welcome to C++!
```

Single-line comments

Function **main** returns an

Left brace **{** begins function body

directive to

exactly once in every C++ program

Statements end with a semicolon **;**

Corresponding right brace **}** ends fu

Namespace **std**

Stream insertion operator

Keyword **return** is one of several means to exit a function; value **0** indicates that the program terminated successfully

Good Programming Practice 1

Every program should begin with a comment that describes the purpose of the program, author, date and time.

Printing a Line of Text Cont'd

- Preprocessor Directives Beginning w/ #
  - Processed by preprocessor before compiling
  - Example
    - #include <iostream>
      - Tells preprocessor to include the input/output stream header file <iostream>

- White Space
  - Blank lines, space characters and tabs
  - Used to make programs easier to read
  - Ignored by the compiler

Common Programming Error 1

Forgetting to include the <iostream> header file in a program that inputs data from the keyboard or outputs data to the screen causes the compiler to issue an error message, because the compiler cannot recognize references to the stream components (e.g., cout).

Good Programming Practice 2

Use blank lines and space characters to enhance program readability.

Printing a Line of Text Cont'd

- Function main
  - A part of every C++ program
    - Exactly one function in a program must be main
  - Can return a value
  - Example
    - int main()
      - This main function returns an integer (whole number)
  - Body is delimited by braces ({ })

Printing a Line of Text Cont'd

- Statements
  - Instruct the program to perform an action
  - All statements end with a semicolon (;)

Printing a Line of Text Cont'd

- Namespace
  - std::
    - Specifies using a name that belongs to "namespace" std
    - Can be removed through the use of using statements
- Standard output stream object
  - std::cout
    - "Connected" to screen
    - Defined in input/output stream header file <iostream>

Printing a Line of Text Cont'd

- Stream insertion operator <<
  - Value to right (right operand) inserted into left operand
  - Example
    - std::cout << "Hello";
      - Inserts the string "Hello" into the standard output
        » Displays to the screen

Printing a Line of Text Cont'd

- Escape characters
  - A character preceded by "\"
    - Indicates "special" character output
  - Example
    - "\n"
      - Cursor moves to beginning of next line on the screen

Common Programming Error 2

Omitting the semicolon at the end of a C++ statement is a syntax error. (Again, preprocessor directives do not end in a semicolon.) The syntax of a programming language specifies the rules for creating a proper program in that language. A syntax error occurs when the compiler encounters code that violates C++'s language rules (i.e., its syntax).

Common Programming Error 2

Syntax errors are also called compiler errors, compile-time errors or compilation errors, because the compiler detects them during the compilation phase. You will be unable to execute your program until you correct all the syntax errors in it. As you'll see, some compilation errors are not syntax errors.

Printing a Line of Text Cont'd

- return statement
  - One of several means to exit a function
  - When used at the end of main
    - The value 0 indicates the program terminated successfully
    - Example
      - return 0;

Good Programming Practice 3

Many programmers make the last character printed by a function a newline (\ n). This ensures that the function will leave the screen cursor positioned at the beginning of a new line. Conventions of this nature encourage software reusability—a key goal in software development.

Escape sequences

Escape sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
\\	Backslash. Used to print a backslash character.
\'	Single quote. Use to print a single quote character.
\"	Double quote. Used to print a double quote character.

Good Programming Practice 4

Indent the entire body of each function one level within the braces that delimit the body of the function. This makes a program's functional structure stand out and helps make the program easier to read.

Good Programming Practice 5

Set a convention for the size of indent you prefer, then apply it uniformly. The tab key may be used to create indents, but tab stops may vary. We recommend using either 1/4-inch tab stops or (preferably) three spaces to form a level of indent.

Modifying the 1st C++ Program

- Two examples
  - Print text on one line using multiple statements
    - Each stream insertion resumes printing where the previous one stopped
  - Print text on several lines using a single statement
    - Each newline escape sequence positions the cursor to the beginning of the next line
    - Two newline characters back-to-back output a blank line

Printing a Line of Text Cont'd

```
1   // Fig. 2.3: fig02_03.cpp
2   // Printing a line of text with multiple statements.
3   #include <iostream> // allows program to output data to the screen
4
5   // function main begins program execution
6   int main()
7   {
8       std::cout << "Welcome ";
9       std::cout << "to C++!\n";
10
11      return 0; // indicate that program ended successfully
12
13  } // end function main
```
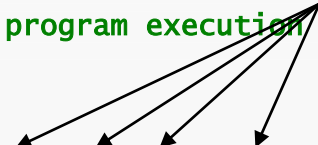
Multiple stream insertion statements produce one line of output because line 8 ends without a newline

```
Welcome to C++!
```

Printing Lines of Text

```
1   // Fig. 2.4: fig02_04.cpp
2   // Printing multiple lines of text with a single statement.
3   #include <iostream> // allows program to output data to the screen
4
5   // function main begins program execution
6   int main()
7   {
8       std::cout << "Welcome\nto\n\nC++!\n";
9
10      return 0; // indicate that program ended successfully
11
12  } // end function main
```

Use newline characters to print on multiple lines

```
Welcome
to

C++!
```

Adding Integers

- Variable
  - Is a location in memory where a value can be stored
  - Common data types (fundamental, primitive or built-in)
    - int – for integer numbers
    - char – for characters
    - double – for floating point numbers
  - Declare variables with data type and name before use

```cpp
1  // Fig. 2.5: fig02_05.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second intege
11    int sum; // sum of number1 a
12
13    std::cout << "Enter first int
14    std::cin >> number1; // read first integer from user into number1
15
16    std::cout << "Enter second integer: "; // prompt user for data
17    std::cin >> number2; // read second integer from user into number2
18
19    sum = number1 + number2; // add the numbers; store result in sum
20
21    std::cout << "Sum is " << sum << std::endl; // display sum; end line
22
23    return 0; // indicate that program ended successfully
24
25 } // end function main
```

Declare integer variables

Use stream extraction operator with standard input stream to obtain user input

Stream manipulator **std::endl** outputs a newline, then "flushes" output buffer

Concatenating, chaining or cascading stream insertion operations

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Adding Integers Cont'd

- Variables (Cont'd)
  - You can declare several variables of same type in one declaration
    - Comma-separated list
    - int integer1, integer2, sum;
  - Variable name
    - Must be a valid identifier
      - Series of characters (letters, digits, underscores)
      - Cannot begin with digit
      - Case sensitive (uppercase letters are *different* from lowercase letters)

Good Programming Practice 6

Place a space after each comma (,) to make programs more readable.

Good Programming Practice 7

Some programmers prefer to declare each variable on a separate line. This format allows you to place a descriptive comment next to each declaration.

Portability Tip 1

C++ allows identifiers of any length, but your C++ implementation may impose some restrictions on the length of identifiers. Use identifiers of 31 characters or fewer to ensure portability.

Good Programming Practice 8

Choosing meaningful identifiers helps make a program *self-documenting*—a person can understand the program simply by reading it rather than having to refer to manuals or comments.

Good Programming Practice 9

Avoid using abbreviations in identifiers. This promotes program readability.

Good Programming Practice 10

Avoid identifiers that begin with underscores and double underscores, because C++ compilers may use names like that for their own purposes internally. This will prevent names you choose from being confused with names the compilers choose.

Error-Prevention Tip 1

Languages like C++ are "moving targets." As they evolve, more keywords could be added to the language. Avoid using "loaded" words like "object" as identifiers. Even though "object" is not currently a keyword in C++, it could become one; therefore, future compiling with new compilers could break existing code.

Good Programming Practice 11

Always place a blank line between a declaration and adjacent executable statements. This makes the declarations stand out in the program and contributes to program clarity.

Good Programming Practice 12

If you prefer to place declarations at the beginning of a function, separate them from the executable statements in that function with one blank line to highlight where the declarations end and the executable statements begin.

Adding Integers Cont'd

- Input stream object
  - std::cin from <iostream>
    - Usually connected to keyboard
    - Stream extraction operator >>
      - Waits for user to input value, press *Enter* (*Return*) key
      - Stores a value in the variable to the right of the operator
        » Converts the value to the variable's data type
    - Example
      - std::cin >> number1;
        » Reads an integer typed at the keyboard
        » Stores the integer in variable number1

Error-Prevention Tip 2

Programs should validate the correctness of all input values to prevent erroneous information from affecting a program's calculations.

Adding Integers Cont'd

- Assignment operator =
  - Assigns the value on the right to the variable on the left
  - Binary operator (two operands)
  - Example:
    - sum = variable1 + variable2;
      - Adds the values of variable1 and variable2
      - Stores the result in the variable sum

Adding Integers Cont'd

- Stream manipulator std::endl
  - Outputs a newline
  - Flushes the output buffer

Good Programming Practice13

Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.

Adding Integers Cont'd

- Concatenating stream insertion operations
  - Use multiple stream insertion operators in a single statement
    - Stream insertion operation knows how to output each type of data
  - Also called chaining or cascading

Adding Integers Cont'd

- Concatenating stream insertion operations (Cont'd)
  - Example
    - std::cout << "Sum is " << number1 + number2 << std::endl;
      - Outputs "Sum is "
      - Then outputs the sum of variables number1 and number2
      - Then outputs a newline and flushes the output buffer

Memory Concepts

- Variable names
  - Correspond to actual locations in the computer's memory
    - Every variable has a name, a type, a size and a value
  - When a new value placed into a variable, the new value overwrites the old value
    - Writing to memory is "destructive"

Memory Concepts (Cont'd)

- Variable names (Cont'd)
  - Reading variables from memory is nondestructive
  - Example
    - sum = number1 + number2;
      - Although the value of sum *is overwritten*
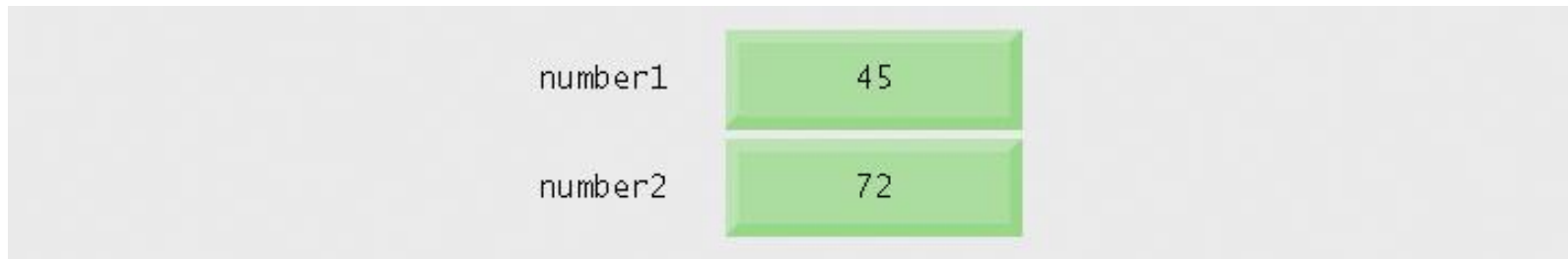   - The values of number1 and number2 *remain intact*

Memory Concepts (Cont'd)

number1   45

Memory location showing the name and value of variable number1.

Memory Concepts (Cont'd)

Memory locations after storing values for number1 and number2.

Memory Concepts (Cont'd)

Memory locations after calculating and storing the sum of number1 and number2.

Arithmetic

- Arithmetic operators
  - *
    - Multiplication
  - /
    - Division
    - Integer division truncates (discards) the remainder
      - 7 / 5 evaluates to 1
  - %
    - The modulus operator returns the remainder
      - 7 % 5 evaluates to 2

Common Programming Error 3

Attempting to use the modulus operator (%) with noninteger operands is a compilation error.

Arithmetic (Cont'd)

- Straight-line form
  - Required for arithmetic expressions in C++
  - All constants, variables and operators appear in a straight line

Arithmetic (Cont'd)

- Grouping subexpressions
  - Parentheses are used in C++ expressions to group subexpressions
    - In the same manner as in algebraic expressions
  - Example
    - a * (b + c)
      - Multiply a times the quantity b + c

Arithmetic Operators

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm or $b \cdot m$	`b * m`
Division	/	$x\,/\,y$ or $\dfrac{x}{y}$ or $x \div y$	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Arithmetic (Cont'd)

- Rules of operator precedence
  - Operators in parentheses are evaluated first
    - For nested (embedded) parentheses
      - Operators in innermost pair are evaluated first
  - Multiplication, division and modulus are applied next
    - Operators are applied from left to right
  - Addition and subtraction are applied last
    - Operators are applied from left to right

Precedence of Arithmetic Operators

| Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*	Multiplication	Evaluated second. If there are several, they are evaluated left to right.
/	Division	
%	Modulus	
+	Addition	Evaluated last. If there are several, they are evaluated left to right.
–	Subtraction	

Common Programming Error 4

Some programming languages use operators ** or ^ to represent exponentiation. C++ does not support these exponentiation operators; using them for exponentiation results in errors.

Good Programming Practice 14

Using redundant parentheses in complex arithmetic expressions can make the expressions clearer.

Order in Which a Second-Degree Polynomial Is Evaluated
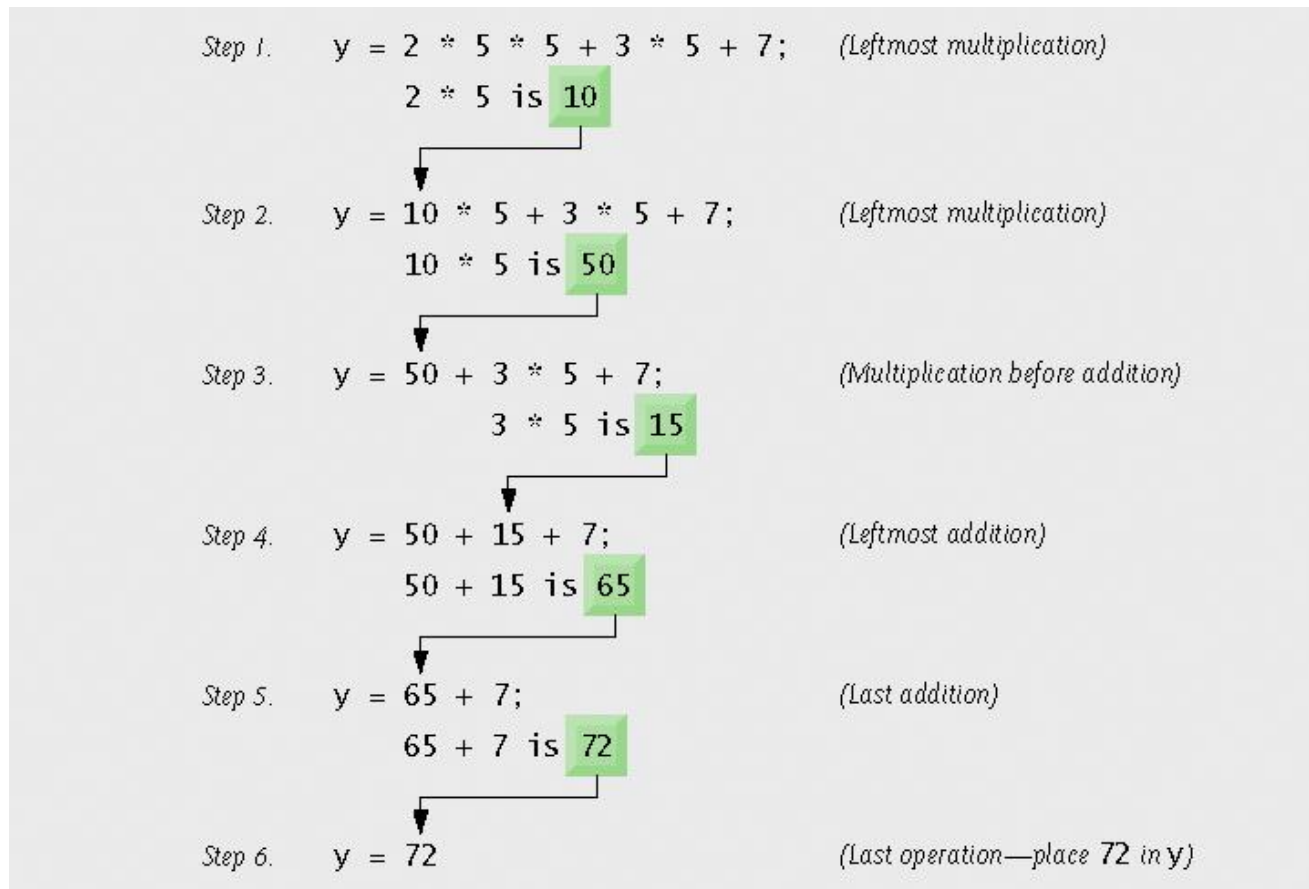
Decision Making

- Condition
  - Expression can be either true or false
  - Can be formed using equality or relational operators
- if statement
  - If the condition is true, the body of the if statement executes
  - If the condition is false, the body of the if statement does not execute

Equality and relational operators

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Common Programming Error 5

A syntax error will occur if any of the operators =, !=, >= and <=appears with spaces between its pair of symbols.

Common Programming Error 6

Reversing the order of the pair of symbols in any of the operators !=, >= and <= (by writing them as =!, => and =<, respectively) is normally a syntax error. In some cases, writing != as =! will not be a syntax error, but almost certainly will be a *logic error* that has an effect at execution time. (cont'd…)

Common Programming Error 6

You will understand why when you learn about logical operators. A *fatal logic error* causes a program to fail and terminate prematurely. A *nonfatal logic error* allows a program to continue executing, but usually produces incorrect results.

Common Programming Error 7

Confusing the equality operator ==with the assignment operator = results in logic errors. The equality operator should be read "is equal to," and the assignment operator should be read "gets" or "gets the value of" or "is assigned the value of." Some people prefer to read the equality operator as "double equals." Confusing these operators may not necessarily cause an easy-to-recognize syntax error, but may cause extremely subtle logic errors.

```cpp
1   // Fig. 2.13: fig02_13.cpp
2   // Comparing integers using if statements, relational operators
3   // and equality operators.
4   #include <iostream> // allows program to perform input and output
5
6   using std::cout; // program uses cout
7   using std::cin;  // program uses cin
8   using std::endl; // program uses endl
9
10  // function main begins program execution
11  int main()
12  {
13      int number1; // first integer
14      int number2; // second integer
15
16      cout << "Enter two integers to compare: "; // prompt
17      cin >> number1 >> number2; // read two integers
18
19      if ( number1 == number2 )
20          cout << number1 << " == " << number2 << endl;
21
22      if ( number1 != number2 )
23          cout << number1 << " != " << number2 << endl;
24
25      if ( number1 < number2 )
26          cout << number1 << " < " << number2 << endl;
27
28      if ( number1 > number2 )
29          cout << number1 << " > " << number2 << endl;
30
```

using declarations eliminate the need for **std::** prefix

Declaring variables

You can write **cout** and **cin** without **std::** prefix

if statement compares the values of **number1** and

if statement compares values of **number1** and **number2** to test for inequality

If the condition is **true** (i.e., the values are equal), execute this statement

If the condition is **true** (i.e., the values are not equal), execute this statement

Compares two numbers using relational operators **<** and **>**

Decision Making Cont'd

```
31      if ( number1 <= number2 )
32          cout << number1 << " <= " << number2 << endl;
33
34      if ( number1 >= number2 )
35          cout << number1 << " >= " << number2 << endl;
36
37      return 0; // indicate that program ended successfully
38
39 } // end function main
```

Compares two numbers using the relational operators <= and >=

```
Enter two integers to compare: 3 7
3 != 7
3 < 7
3 <= 7
```

```
Enter two integers to compare: 22 12
22 != 12
22 > 12
22 >= 12
```

```
Enter two integers to compare: 7 7
7 == 7
7 <= 7
7 >= 7
```

Good Programming Practice 15

Place **using** declarations immediately after the **#include** to which they refer.

Good Programming Practice 16

Indent the statement(s) in the body of an **if** statement to enhance readability.

Good Programming Practice 17

For readability, there should be no more than one statement per line in a program.

Common Programming Error 8

Placing a semicolon immediately after the right parenthesis after the condition in an `if` statement is often a logic error (although not a syntax error). The semicolon causes the body of the `if` statement to be empty, so the `if` statement performs no action, regardless of whether or not its condition is true.

Common Programming Error 9

It is a syntax error to split an identifier by inserting white-space characters (e.g., writing **main** as **ma in**).

Good Programming Practice 18

A lengthy statement may be spread over several lines. If a single statement must be split across lines, choose meaningful breaking points, such as after a comma in a comma-separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines and left-align the group of indented.

Precedence and Associativity of the Operators Discussed So Far

Operators				Associativity	Type
( )				left to right	parentheses
*	/	%		left to right	multiplicative
+	–			left to right	additive
<<	>>			left to right	stream insertion/extraction
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

Good Programming Practice 19

Refer to the operator precedence and associativity chart when writing expressions containing many operators. Confirm that the operators in the expression are performed in the order you expect. If you are uncertain about the order of evaluation in a complex expression, break the expression into smaller statements or use parentheses to force the order of evaluation, exactly as you would do in an algebraic expression.

Data Types

Data types		
`long double`		
`double`		
`float`		
`unsigned long int`	(synonymous with `unsigned long`)	
`long int`	(synonymous with `long`)	
`unsigned int`	(synonymous with `unsigned`)	
`int`		
`unsigned short int`	(synonymous with `unsigned short`)	
`short int`	(synonymous with `short`)	
`unsigned char`		
`char`		
`bool`		

```
long double: 12 B
double:       8 B
float:        4 B

long:         4 B

int:          4 B

short:        2 B

char:         1 B
```