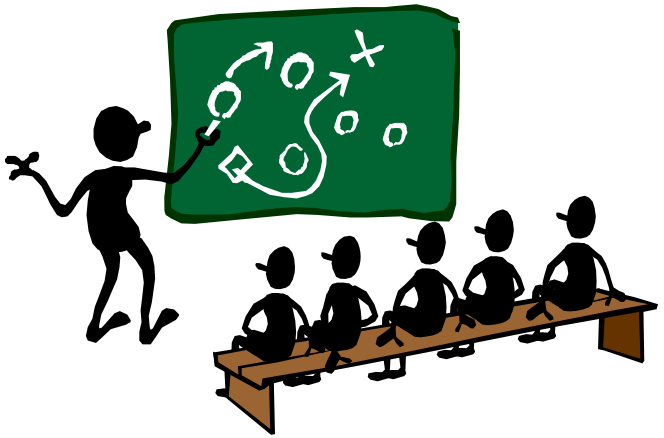
C++ Programming Language
Chapter 1 C++ Basics



Juinn-Dar Huang

Associate Professor

jdhuang@mail.nctu.edu.tw

February 2011

Learning Objectives

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

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Introduction to C++

- C programming language
  - developed by D. Ritchie, AT&T Bell Lab., 1970s
  - Structured Programming (or Modular Programming)
- C++ programming language
  - developed by B. Stroustrup, AT&T Bell Lab., 1980s
  - to be a better “C”
  - Object-Oriented Programming (OOP)

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C++ and OOP

- Main characteristics of OOP
  - encapsulation (Chap 6)
    - information hiding, abstraction
  - inheritance (Chapter 14)
    - code reuse
  - polymorphism (Chapter 15)
    - a single name can have multiple meanings within a class hierarchy (inheritance)
  - template (Chapter 16)
    - code reuse
    - generic programming

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

A Sample C++ Program (1/2)

Display 1.1 A Sample C++ Program

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

3   int main( )
4   {
5       int numberOfLanguages;

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

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

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

15      return 0;
16  }
```

A Sample C++ Program (2/2)

SAMPLE DIALOGUE 1

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

User types in `0` on the keyboard.

SAMPLE DIALOGUE 2

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 1
Enjoy the book
```

User types in `1` on the keyboard.

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C++ Identifiers

- An identifier is a name of variable, constant, …
- A C++ identifier
  - consists of a sequence of letters, digits, and the underscore character (_)
  - must start with either a letter or an underscore character // avoid doing so in general
  - is case-sensitive
    - e.g., abc and AbC are two different identifiers
  - can be of any length // sadly, NOT true in reality
- Keywords are special identifiers (Appendix 1)
  - e.g., if, for, char, …
  - cannot be used for user-defined entities

C++ Variables

- Variables
  - its name is of course an identifier
  - is a memory location to store data
  - must be declared before its use

```
int number;                 // declaration & definition
double width, length;       // declaration & definition
extern int count;           // declaration ONLY, discuss later
```

  - meaningful names!
  - naming convention: starting with a lowercase letter
    - e.g., weight, total_weight, totalWeight, …

Fundamental Data Types (1/2)

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

Fundamental Data Types (2/2)

`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.

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Be More Precise (1/3)

- Integral types
  - Boolean (bool) , character (char) , integer (int)
- Floating-point types
  - float, double, long double

- Character types
  - size: almost universally a byte (8 bits)
  - hold a character mainly; can also hold an integral value
  - char, signed char, unsigned char are 3 different types
  - signed char: –128~127; unsigned char: 0~255
  - char: implementation-dependent (either signed or unsigned)

```
char ch;
signed char sch;
unsigned char uch;
```

Be More Precise (2/3)

- Integer types
  - [signed | unsigned] [long | short] int : 9 combinations!
  - an integer type is signed unless “unsigned” is explicitly specified
  - i.e., actually 6 different types!

int
unsigned int
short int
unsigned short int
long int
unsigned long int

int can be omitted

  - size
    - short int: 2 bytes typically
    - int: 4 bytes typically
    - long int: 4 bytes typically
    - guarantee:

      $1 = \text{sizeof(char)} \le \text{sizeof(short)} \le \text{sizeof(int)} \le \text{sizeof(long)}$

      $\text{sizeof(T)} = \text{sizeof(signed T)} = \text{sizeof(unsigned T)}$

      $\text{char} \ge 1 \text{ byte}; \text{ short} \ge 2 \text{ bytes}; \text{ long} \ge 4 \text{ bytes}$

Be More Precise (3/3)

- Floating-point types
  - hold floating-point data
  - size: sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)
  - always signed! ; no short double!

- Boolean type (bool)
  - only two values: true and false

```
bool answer;
answer = true;
answer = false;
```

  - bool to int: true is converted to 1 ; false to 0
  - int to bool: nonzero to true; zero to false

```
int a = true; // a = 1;
int b = false; // b = 0;

bool c = -12; // c = true;
bool d = 0; // d = false;
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Assignment Statements (1/2)

- Initialize a variable before using it!
  - Results is undefined if you don't!
- Initializing variables in declarations is allowed
  - int myValue = 0, limit(10) ;
- Assigning data during execution
  - lvalues (left-hand side) & rvalues (right-hand side)
    - lvalues MUST be variables
    - rvalues can be any expressions
    - e.g.,

      distance = rate * time;
      lvalue: "distance"
      rvalue: "rate * time"

Variable = Expression

```
a = 3;
b = c;
d = time * rate;
sum = sum + value;
n = m = 2;
```

Assignment Statements (2/2)

- Shorthand Notations

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);`

- Also

  >>=, <<=, &=, |=, ^=

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Data Assignment Rules

- Compatibility of data assignments
  - Type mismatches
    - General Rule: avoid placing value of one type into variable of another type
  - int 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
    - 2, 5.75, "Z", "Hello World"
    - considered "constants": can’t change in program

Literals

- Literals
  - Examples:
    - ‘Z’ // character literal
    - "Hello World" // string literal
    - 2 // integer literal (in decimal)
    - 0x1f // integer literal (in hexadecimal)
    - 5.75 // floating-point literal of the type double
- Cannot change values during execution
- Called "literals" because you "literally typed" them in your program!

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Escape Sequences

- "Extend" character set
- Backslash “\” precedes a character
  - Instructs compiler: a special "escape character" is coming

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Some Escape Sequences (1/2)

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.)

Some Escape Sequences (2/2)

\'	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

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Constants

```
double money *= (1 + 0.05);  // What is 0.05?
```

```
const double RATE = 0.05; // convention: all uppercase letters
double money *= (1 + RATE); // better readability
RATE = 0.1;  // compilation error
```

- Named constants or declared constants (e.g., RATE)
  - better readability
  - better maintainability
    - if the interest rate raises to 7% ➔ just replace 0.05 with 0.07 ONCE!
  - change attempts result in compilation errors!
- Named constants MUST be initialized

```
const int myWeight; // compilation error!
```

Named Constants (1/2)

Display 1.4 **Named Constant**

```
1   #include <iostream>
2   using namespace std;
3
4   int main( )
5   {
6       const double RATE = 6.9;
7       double deposit;

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


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Named Constants (2/2)

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

14        return 0;
15    }
```

SAMPLE DIALOGUE

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

- Operator precedence rules
  - specify the execution order of variant operators
  - details in Chapter 2

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Arithmetic Precision (1/2)

- 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!

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Arithmetic Precision (2/2)

- Examples:
  - 17 / 5 evaluates to 3 in C++!
    - Both operands are integers
    - Integer division is performed!
  - 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 (1/2)

- Casting for Variables
  - Can add ".0" to literals to force precision arithmetic, but what about variables?
    - We can't use "myInt.0"!
  - C style ➔ double dvar = (double) ivar;
  - C++ style ➔ double dvar = static_cast<double>(ivar) ;

static_cast<type>(expression)

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Type Casting (2/2)

- Two kinds
  - 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 static_cast operator

      ```
      int m;
      static_cast<double>(m) / 5.5
      ```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

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;

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Shorthand Operators: Two Options

- Post-increment
  intVar++
  - uses current value of variable, THEN increments it
- Pre-increment
  ++intVar
  - increments variable first, THEN uses new value
- No difference if "alone" in statement:
  intVar++; and ++intVar; → identical result

- The above ideas also apply to decrement operators

- Whenever both forms get the same result
  → prefer prefix to postfix

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

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

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; // discuss later
  - Tells C++ compiler 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

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

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, and 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
- You must explicitly tell C++ compiler how to output numbers in your programs!

Formatting Numbers

- "Magic Formula" to force decimal sizes:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- These statements force all future cout'ed values:
  - To have exactly two digits after the decimal point
  - Example:
    cout << "The price is $" << price << endl;
    - Now results in the following:
      The price is $78.50
- Can modify precision "as you want" as well!

More details in Chapter 12

Error Output

- Output with cerr
  - cerr works same as cout
  - provides mechanism for distinguishing between regular output and error output
- Re-direct output streams
  - most systems allow cout and cerr to be "redirected" to other devices
    - e.g., line printer, output file, error console, etc.

Input Using cin

- cin for input, cout for output, cerr for error output
- Differences:
  - ">>" (extraction operator) points opposite
    - Think of it as "pointing toward where the data goes"
  - "cin" is used instead of "cout"
  - no literals allowed for cin
    - Must input to a **variable**
    - cin >> 23; // compilation error!

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

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

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 the specified library file to your program
  - called "preprocessor directive"
    - Executes before compilation, and simply “copies” library file into your program file
- C++ has many libraries
  - Input/output, math, strings, …

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

- The notion of namespace is for large-scale SW project
  - avoid name collisions

More details in Chapter 11

Summary (1/2)

- C++ is case-sensitive
- Use meaningful names
  - for variables and constants
- Variables must be declared before use
  - should also be initialized before use
- More cares in numeric manipulation
  - type casting, precision, parentheses, precedence (order of operations)
- #include C++ libraries as needed

Summary (2/2)

- Object cin
  - used for console input
- Object cout
  - used for console output
- Object cerr
  - used for error messages
- Use comments to aid understanding of your program

C++ Reference Books

- If you are serious in studying C++
  - forget about those “Teach yourself C++ in XX days” books (throw them into your trash can)
- *The C++ Programming Language*, special (3rd) edition, B. Stroustrup, Addison-Wesley, 2000
- *C++ Primer*, 4th edition, S. B. Lippman et al, Addison-Wesley, 2005