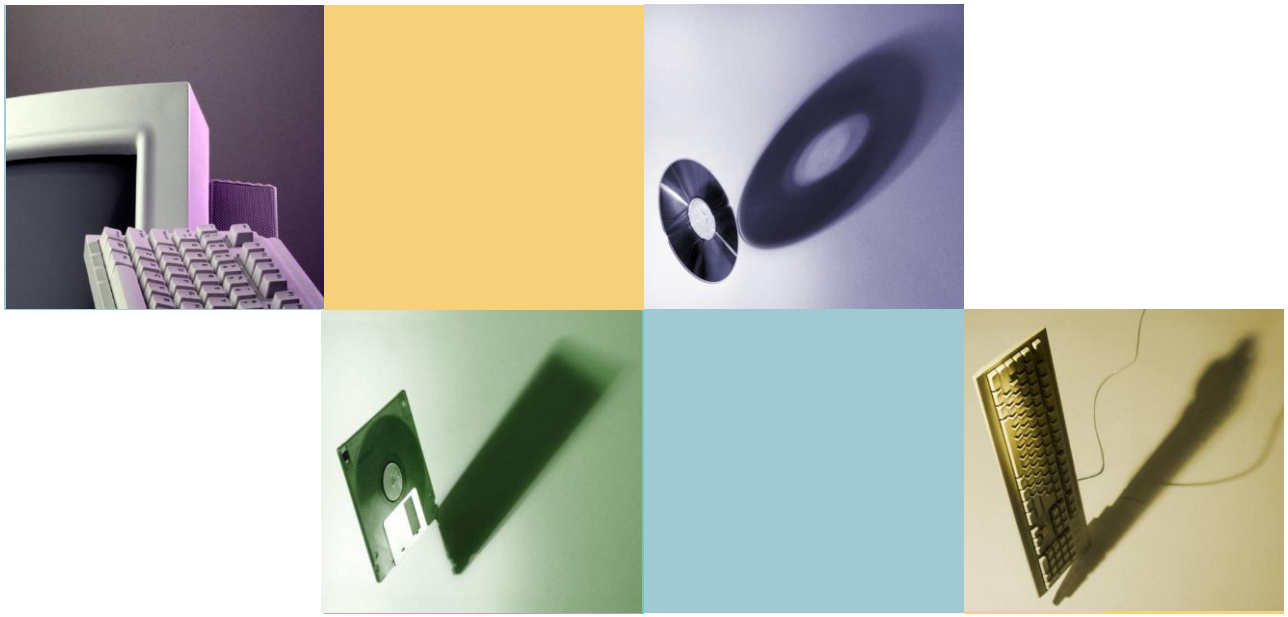
Object-Oriented Programming



Chuan-Kang Ting

Dept of Computer Science and Information Engineering
National Chung Cheng University

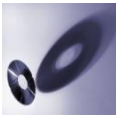
Chapter 1

C++ Basics



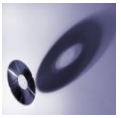
From C to C++

- **C++ = C + classes**
  + (other modern features)
  - Developed by Bjarne Stroustrup of AT&T Bell Laboratories in the early 1980s
    - http://www.stroustrup.com/
  - C is a subset of C++
    - Make it easy for programmers to migrate to C++
  - C++ has facilities for classes
    - Can be used for OOP


C++ and OOP

- **The characteristics of OOP – Pie**
  - **E**ncapsulation
    - information hiding and abstraction
  - **I**nheritance
    - code reusability
  - **P**olymorphism
    - a single name with multiple meanings in inheritance

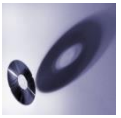

C++ and OOP (cont'd)

- **The characters of C++**
  - Connection to C: makes C++ a traditional look with an object-oriented spirit
  - Classes: allow C++ to be used as an OOL
  - Overloading of functions and operators
  - Template
  - Namespace
  - Exception handling
  - C++ style of memory management



C++ Terminology

- **Functions: all procedure-like entities**
  - May be called *procedures*, *methods*, *functions*, or *subprograms* in other language
- **Program: a C++ program is just a function called `main`**
  - Invoked *automatically* by system

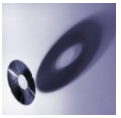




A Sample C++ Program (1 of 2)

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  }
```

Note the C++ style!


Variables

- **A memory location to store data for a program**
- **Every variable in a C++ program must be declared before it is used**
- **Declare: tell the compiler what kind (type) of data you will store in the variable**
- **e.g.**

```
int numStudents;
double score;
```



Data Types: Simple Types

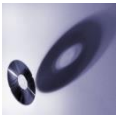
Display 1.2 Simple Types

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
`short` (also called `short int`)	2 bytes	–32,767 to 32,767	Not applicable
`int`	4 bytes	–2,147,483,647 to 2,147,483,647	Not applicable
`long` (also called `long int`)	4 bytes	–2,147,483,647 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



Initializing & Assigning Data

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

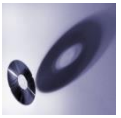




Assignment Statement (1)

- **Assignment**
  - "Change" the value of a variable
  - Syntax

    Variable (Lvalue) = *Expression* (Rvalue)
  - e.g.

```
distance = rate * time;
count = count + 2;
```

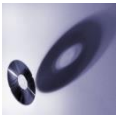




Assignment Statement (3)

- **Assignment compatibility**
  - General rule: You cannot store a value of one type in a variable of another type
  - e.g. type mismatch

```
int intVar;
intVar = 2.99;
```

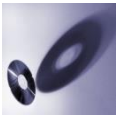
  - Mostly, the compiler will give intVar the value 2, not the value 3
  - Not all compilers will react the same way → confusing and less portable


Initializing Variables

- **Initialization**
  - Initialize a variable by **giving** it an “initial” value
  - A variable has no meaningful value until initialized
    - An uninitialized variable may cause errors
  - Initialize:
    - int score = 60, numStudents = 50;
    - int score(60), numStudents(50);

Initialization
vs. Assignment


Constants

- **Variables: changeable**
- **Constants: unchangeable**
  - Declaration: modifier **`const`**
  - e.g.

```
const int CLOSED_WINDOWS = -1;
const int DOUBLE_CLICK = 2;
```

  - Practical manner
    - Writing declared constants in all uppercase letters

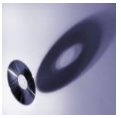


Arithmetic Operators & Expression (1)

- **Precision of calculations**
  - VERY important!
    - Expressions in C++ might not evaluate as you would "expect“
    - Common pitfall
  - Highest-order operand determines the precision
    - e.g.

```
z = x + y;
```

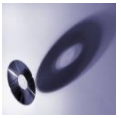
      - If **all** the types are integer types, the result will be the integer type
      - If **at least one** of the sub-expression is of a floating-point type, the result will be a floating-point type


Arithmetic Operators & Expression (2)

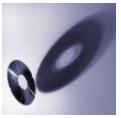
- **Division**
  - Floating-point division
    - Luckily, everything behaves as you expect
  - Integer division
    - Integer division discards the part after the decimal point. e.g. 11/3 = 3 (not 3.666 or 4)
    - Notice: the number is NOT rounded
    - Common Pitfall:

```
int feet = 15000;
totalPrice = 5000   * (feet / 5280.0);  //$14,200
totalPrice = 5000   * (feet / 5280);    //$10,000
totalPrice = 5000.0 * (feet / 5280);    //?
```


Type Casting

- **A way of changing a value of one type to a value of another type**
- **Implicit (automatic):**
  - Type coercion, e.g. `double d = 5;` (5 → 5.0 → d)
- **Explicit (manual):**
  - static_cast<*Type*>(*Expr.*)
    - e.g. `double ans = n / static_cast<double>(m);`
    - Older form: `(double)42` or `double(42)`
  - const_cast<*Type*>(*Expr.*)
  - dynamic_cast<*Type*>(*Expr.*)
  - reinterpret_cast<*Type*>(*Expr.*)


Console Input/Output

- **I/O Objects**
  - **`cin`**: input from standard console (e.g. keyboard)
  - **`cout`**: output to standard console (e.g. screen)
  - **`cerr`**: ouput to standard error output stream (e.g. screen)
- **Declaration**

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

- **Operator << and >>**
  - Assign (insertion/extraction)
  - Direction (to/from)



Console Input/Output (cont'd)

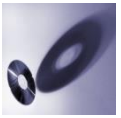
- **An example**

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

int main()
{
  int numPerson;
  double cost;

  cout << "Enter the number of persons \n"
       << "followed by the cost per person. \n";
  cin >> numPerson
      >> cost;

  cout << "Total cost = " << (cost * numPerson) << endl;

  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(2);                  //affect all subsequent lines
  cout << "Total cost = " << (cost * numPerson) << endl;

  return(0);
}
```

```
Enter the number of persons
followed by the cost per person.
16
27.826
Total cost = 445.216
Total cost = 445.21
```

Styling Your Code

- **To make your code easy to read and easy to modify**
- **Comments**
  - for line: //
  - for block: /* */
- **Rule of thumb**
  - constants: ALL_UPPER_CASE
  - variables / functions: lowerToUpper
  - comments: “just enough”
  - General Rule: Name it meaningfully


Libraries & Namespaces

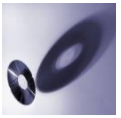
- **Libraries**
  - Include a library by the include directive:

```
#include <Lib_name>
```

    - Called "**preprocessor** directive"
      - Executes before compiler, and simply "copies"

- **Namespaces**
  - Collection of name definitions
  - To distinguish names, such as function names, in different places
  - All the C++ standard libraries place their definitions in the **`std`** (standard) namespace:

```
using namespace std;
```


Summary (1)

- **Variables, Expression, and Assignment Statements**
  - must be declared before they are used
  - an uninitialized variable may cause errors
  - declare by `const` the constants, which cannot be changed
  - be careful about the data type in expression to avoid unexpected errors → type casting
  - Occasions for pre- and post-increment/decrement



Summary (2)

- **Console Input/Output**
  - I/O objects: cin, cout, cerr
  - include directive and namespace

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

- **Program Style**
  - constants: ALL_UPPER_CASE
  - variables/functions: lowerToUpper
  - comments: “just enough”
  - meaningful names!