IM 1003: Programming Design

Introduction

Ling-Chieh Kung

Department of Information Management
National Taiwan University

February 17, 2014

Outline

- **Computer programming**
- Our first C++ program: basic structure and **`cout`**
- Our second C++ program: variable declaration and **`cin`**
- Our third C++ program: the **`if`** and **`while`** statements
- Formatting a C++ program

Computer programming

- What are **computer programs**?
  - The elements working in computers.
  - Also known as **software**.
  - A structured combination of data and instructions used to operate a computer to produce a specific result.
- Strength: High-speed computing, large memory, etc.
- Weakness: People (programmers) need to tell them what to do.
- How may a programmer tell a computer what to do?
  - Programmers use "**programming languages**" to write codes line by line and construct "computer programs".
- Running a program means executing the instructions line by line and (hopefully) achieve the programmer's goal.

Programming languages

- People and computers talk in programming languages.
- A programming language may be a machine language, an assembly language, or a high-level language (or something else).
- A **machine language** uses 0s and 1s to form instructions.
  - For example, under the MIPS architecture, each instruction is 32-bit long.
  - "00000000001000100011000000100000" means "adding the registers 1 and 2 and placing the result in register 4."
- An **assembly language** label these instructions as words.
  - `ADD ax, bx`

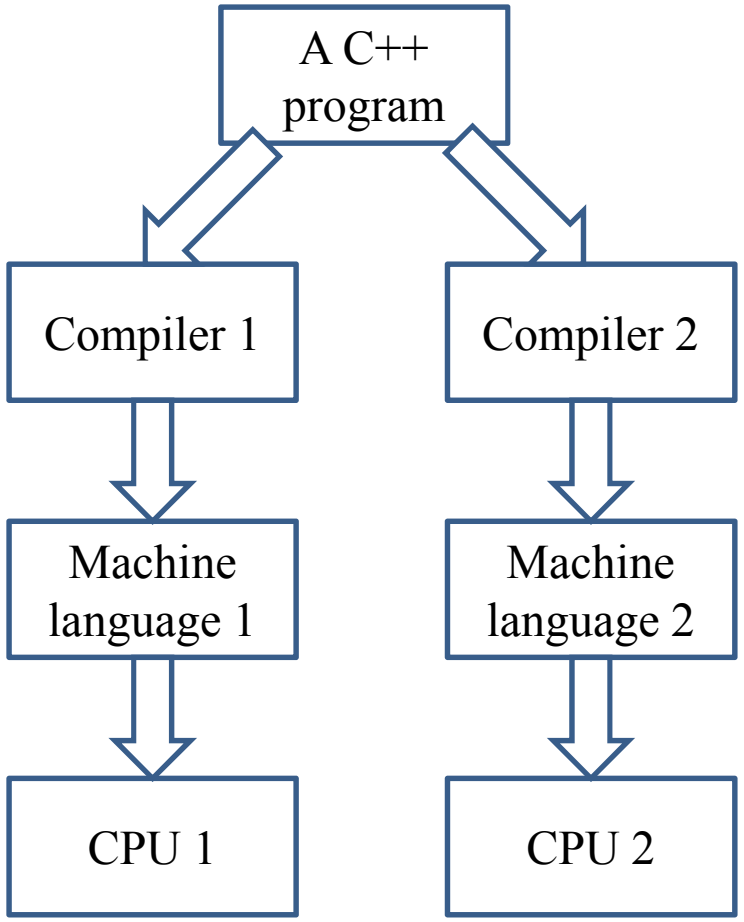
    `MOV cx, ax`
  - An **assembler** then translate an assembly program into a machine program.

Programming languages

- Machine and assembly languages control devices directly.
  - Devices include computers, routers, smart phones, wearable devices, etc.
- Two main drawbacks:
  - **Too inefficient** for developing large-scale programs.
  - Device-dependent and thus **not portable**.
- Most application software are developed in **high-level languages**.
  - The language we study in this course, C++, is a high-level language.
  - Some others: Basic, Quick Basic, Visual Basic, Fortran, COBOL, Pascal, Delphi, C, Perl, Python, Java, C#, PHP, Matlab, etc.
- A **compiler** translates C++ programs into assembly programs.
  - For other high-level programs, an **interpreter** may be used instead.

Portability of high-level languages

- Most high-level languages allow **portability**.
- A C++ program following the standard can run on computers with different types of CPU.
  - As long as we have the right compilers for both computers.



High-level and assembly languages

- Which one should a programmer adopt?

High-level languages	Assembly languages
Easier to program	Harder to program
Portable (typically)	Not portable
Hard (if not impossible) to control hardware directly	Can control hardware directly
Suitable for application software, web services, operating systems, etc.	Suitable for drivers, video cards, embedded systems, etc.

  - Some application developers mix assembly codes in their program to enhance efficiency.

The C++ programming language

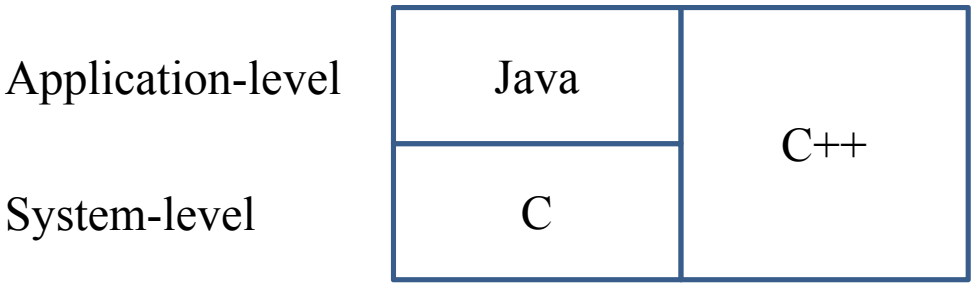
- C/C++ is sometimes called a "**mid-level**" language.
  - It allows a C++ programmer to "access" the **memory**.
- With such low-level functionality, C/C++ is powerful.
  - And dangerous…
- In this course, we focus on high-level programming.
- Some (not all) courses related to low-level programming:
  - In the IM department: Computer Organization and Structure, Operating Systems, Computer Networking, etc.
  - In the CSIE department: Computer Organization and Assembly Languages, Computer Architecture, Systems Programming, Compiler Design, etc.

The C++ programming language

- C++ is developed by Bjarne Stroustrup starting from 1979 at AT&T Bell Labs.
- C++ originates from another programming language C.
  - C is a **procedural** programming language.
  - C++ is an **object-oriented** programming (OOP) language.
- Roughly speaking, C++ is created by adding object-oriented functionalities into C.
  - For **teams** to build **large** software systems requiring a **long** time.
- C++ is (almost) a superset of C.
  - Most C programs can be complied by a C++ compiler.

Why C++?

- C++ is broader than C (or other non-OO languages):
  - For C, we do not need to study objects-oriented functionalities.
- C++ is broader than Java (or other pure high-level languages):
  - For Java, we do not need to study memory-related functionalities.
- C++ is powerful!



Outline

- Computer programming
- **Our first C++ program: basic structure and `cout`**
- Our second C++ program: variable declaration and **`cin`**
- Our third C++ program: the **`if`** and **`while`** statements
- Formatting a C++ program

Our first C++ program

- As in most introductory computer programming courses, let's start from the "Hello World" example:

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

int main()
{
  cout << "Hello World! \n";
  return 0;
}
```

- Let's try to compile this source code and run it!

Our first C++ program

- The program can be decomposed into four parts.
  - The preprocessor.
  - The namespace.
  - The main function block.
  - The statements.
- Some words are colored because they are C++ **reserved words** (**keywords**), which serve for special purposes.
  - We will talk about them soon.

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

int main()
{
  cout << "Hello World! \n";
  return 0;
}
```

The preprocessor and namespace

- At this moment, let's ignore the first two lines.
  - They are doing some preparations before you may write your own instructions.
  - To be discussed later.
- For now, just copy them.

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

int main()
{
  cout << "Hello World! \n";
  return 0;
}
```

The main function block

- A C++ Program always runs from the first line of "**the main function block**" to the last line.
  - The function is named `main()`.
  - One program, one main function.
- A pair of braces (curly brackets) defines a **block**.
  - Within `{` and `return 0;`, we write our statements to tell the program what to do.
- For now, just copy them.

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

int main()
{
  cout << "Hello World! \n";
  return 0;
}
```

Statements

- There are always some **statements** in the main function.
  - `return 0;` is also a statement.
  - The computer executes the first statement, then the second, then the third….
- Each C++ statement is ended with a **semicolon** (`;`).
  - There are two statements in this main function.

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

int main()
{
  cout << "Hello World! \n";
  return 0;
}
```

cout and <<

```
cout << "Hello World! \n";
```

- cout is a pre-defined object for "console output".
  - It sends whatever data passed to it to the standard display device.
  - Typically the computer screen in the console mode.
- The **insertion operator <<** marks the direction of data flow.
  - Data flow like streams.
- "Hello world! \n" is a **string**.
  - Characters within a pair of **double quotation marks** form a string.
- cout << "Hello world! \n":
  - Let the string "Hello world! \n" flow to the screen. The character H first, then e, then l….

The escape sequence \n

```
cout << "Hello World! \n";
```

- But wait… what is that "**\n**"?
- In C++, the **slash** symbol "****" starts an **escape sequence**.
  - An escape sequence represents a "special character".
  - **\n** in C++ means "change to a new line".
  - To see this, try the following codes:

```
cout << "Hello World! \n";
cout << "I love C++\n  so much!";
```

Escape sequences

- Some common escape sequences are listed below:

Escape sequence	Effect	Escape sequence	Effect
`\n`	A new line	`\\`	A slash: \
`\t`	A horizontal tab	`\'`	A single quotation: '
`\b`	A backspace	`\"`	A double quotation: "
`\a`	A sound of alert		

Concatenated data streams

- The insertion operator << can be used to **concatenate** multiple data streams in one single statement.
  - The two statements

```
cout << "Hello World! \n";
cout << "I love C++\n  so much!";
```

    and this statement

```
cout << "Hello World! \n" << "I love C++\n  so much!";
```

    display the same thing.
- Note that the statement

```
"Hello World!" >> cout;
```

  is wrong!

Our first C++ program as a whole

- This is our first C++ program:

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

int main()
{
  cout << "Hello World! \n";
  return 0;
}
```

- Go modify the statements by yourself!

Outline

- Computer programming
- Our first C++ program: basic structure and **`cout`**
- **Our second C++ program: variable declaration and `cin`**
- Our third C++ program: the **`if`** and **`while`** statements
- Formatting a C++ program

The `cin` object

- The **`cout`** object prints out data to the console output.
- Another object, **`cin`**, accepts data **input** (by the user or other programs) from the console input (typically the keyboard).
- In order to use the **`cin`** object, we need to first prepare a "**container**" for the input data. The thing we need is a **variable**.
- When we use a single variable to receive the data, the syntax is

```
cin >> variable;
```

  - In this week, you will use the **`cin`** object to **interact** with your program.
  - In the future, you will use **`cin`** to read testing data of your program.
- Let's first learn how to **declare variables**.

Variables and data types

- A variable is a container that store a value.
  - Once we declare a variable, the system allocate a **memory space** for it.
  - A value may then be stored in that space.
- In C++, each variable must be specified a **data type**.
  - It tells the system how to allocate memory spaces.
  - It tells the system how to interpret those 0s and 1s stored there.
- The data type will also determine how **operations** are performed on the variable.

Basic data types

- There are ten **basic** (or **built-in** or **primitive**) data types in C++.
  - They are provided as part of the C++ standard.

Category	Type	Bytes	Type	Bytes
Integers	`bool`	1	`long`	4
	`char`	1	`unsigned int`	4
	`int`	4	`unsigned short`	2
	`short`	2	`unsigned long`	4
Fractional numbers	`float`	4	`double`	8

- There are ten basic data types, belonging to two categories.
  - The number of bytes is **compiler-dependent**.
- Today let's use integer and Boolean variables only.

Variable declaration

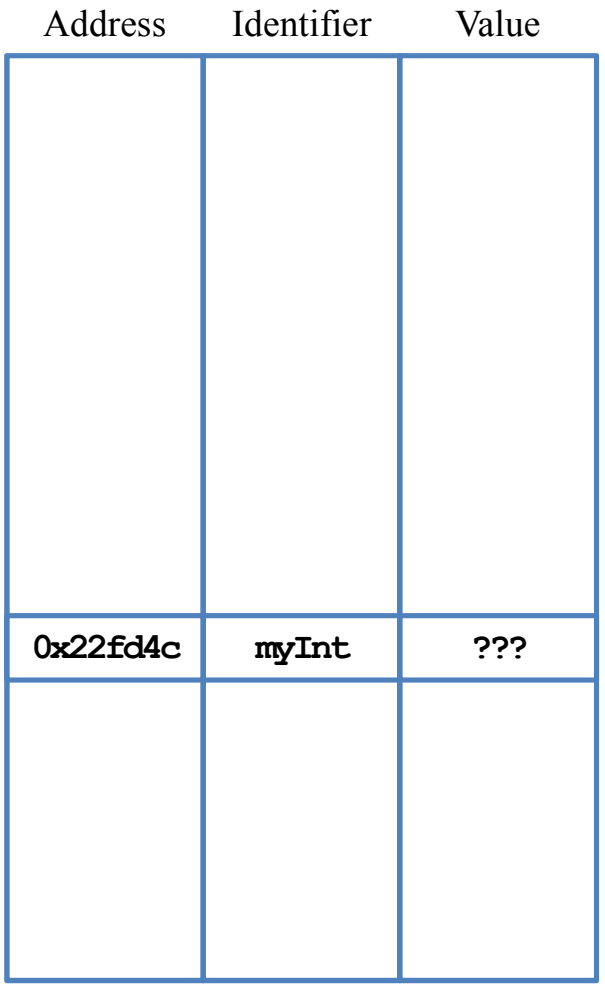
- Before we use a variable, we must first **declare** it.
  - We need to specify its **name** and **data type**.
- The syntax of a variable declaration statement is

```
type variable name;
```

  - For example,

```
int myInt;
```

    declares an integer variable called **`myInt`**.
- A variable name is an **identifier**.
  - We do not need to memorize the memory address (which is a sequence of numbers).
  - We access the space through the variable name.



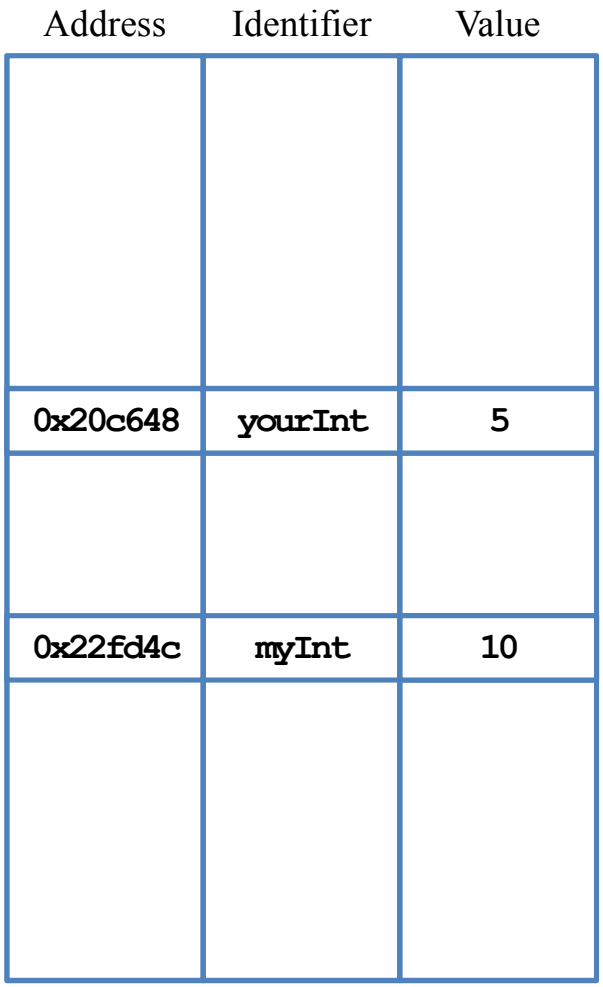
Memory

Declaration and assignment

- Besides declaring a variable, we may also **assign** values to a variable.
  - `int myInt;` declares an integer variable.
  - `myInt = 10;` assigns 10 to `myInt`.
- We may do these together.

`type variable name = initial value;`

  - `int yourInt = 5;` declares an integer variable `yourInt` and assigns 5 to it.
  - The assignment is then called **initialization** if done with declaration.
- Without initialization, the variable may be of **any value** (depending on what was left since the last time this space is used)!



Memory

More about variable declaration

- We may declare multiple variable in the same type together:
  - `int a, b, c;` declares three integers **a**, **b**, and **c**.
- We may initialize all of them also in a single statement:
  - `int a = 1, b = 2, c = 3;`
- A variable's name consists of a consecutive sequence of letters, digits, and the underline symbol "_".
  - It cannot begin with a number.
  - It cannot be the same as a C++ **keywords** (cf. Figure 3.3 of the textbook).
  - It (and the whole C++ world) is **case-sensitive**.
- **Always** initialize your variables (e.g., 0).
- Use meaningful names (e.g., **`yardToInch`** is) better than **`y`**).
- **Capitalize** the fir,st character of each word, but not the very first one.
  - `int yardToInch = 12, avgGrade = 0, maxGrade = 100;`

Our 2nd C++ program (in progress)

- This is our second C++ program (to be completed later):
- We first declare and initialize two integers.
- We then do

```
cout << num1 + num2;
```

- There are two **operations** here:
  - **`num1 + num2`** is an addition operation. The sum will be **returned** to the program.
  - That returned value is then sent to **`cout`** through **`<<`**.
- In effect, **`17`** is displayed on the screen.

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

int main()
{
  int num1 = 13, num2 = 4;
  cout << num1 + num2;

  return 0;
}
```

Our 2nd C++ program (in progress)

- Let's make the output look better:

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

int main()
{
  int num1 = 13, num2 = 4;
  cout << "The sum of " << num1 << " and " << num2 << " is "
       << num1 + num2 << "\n";

  return 0;
}
```

- How would you interpret the program?

Our 2nd C++ program (in progress)

- There are other arithmetic operations:
  - Addition, subtraction, multiplication, division, and modulus.
- What will be displayed on the screen?
- Data types matter!
  - If the inputs of the division operation are both integers, the output will be **truncated** to an integer.
  - We will discuss this in details later in this semester.

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

int main()
{
  int num1 = 13, num2 = 4;

  cout << num1 + num2 << "\n";
  cout << num1 - num2 << "\n";
  cout << num1 * num2 << "\n";
  cout << num1 / num2 << "\n";
  cout << num1 % num2 << "\n";

  return 0;
}
```

Our second C++ program

- Now we are ready to present our second C++ program:

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

int main()
{
  int num1 = 0, num2 = 0;

  cout << "Please enter one number: ";
  cin >> num1;
  cout << "Please enter another number: ";
  cin >> num2;

  cout << "The sum is " << num1 + num2;

  return 0;
}
```

The cin object

- In this example, we allow the user to enter two numbers.
- We declare two variables to receive the inputs.
- We then use the **cin** object to send input values into the variables.

```
cout << "Please enter one number: ";
cin >> num1;
cout << "Please enter another number: ";
cin >> num2;
```

  - The **cout** statements are **prompts**: a message telling the user what to do.
  - The input of a value ends when the user press "enter".
- The variable can then be used in other statements.

```
cout << "The sum is " << num1 + num2;
```

The cin object

- The **extraction operator >>** is used with the **cin** object.
- One cannot use **cout** with **>>** or **cin** with **<<!**
  - Both statements here are wrong:

```
a >> cout;
b << cin;
```

- The input stream is split into multiple pieces by "enter" and white spaces.
  - Different pieces are sent to different variables.
  - If the number of variables is fewer than the input pieces, pieces will be put in an **input buffer** waiting for future **cin** operations.
- Try to run the program by entering "**4 13**".

Our second C++ program

- Another way to implement this program:

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

int main()
{
  int num1 = 0, num2 = 0;

  cout << "Please enter two numbers, separated by a white space: ";
  cin >> num1 >> num2;

  cout << "The sum is " << num1 + num2;

  return 0;
}
```

- `>>` may send (pieces of) values to multiple variables.
- Data types matter: What if we enter "`1.3 4`"?

Outline

- Computer programming
- Our first C++ program: basic structure and **`cout`**
- Our second C++ program: variable declaration and **`cin`**
- **Our third C++ program: the `if` and `while` statements**
- Formatting a C++ program

Our third C++ program (in progress)

- Would you guess what does this program do?
- We use the **if** statement to control the sequence of executions.

```
if (condition)
{
  statements
}
```

  - If ***condition*** returns **true**, do ***statements*** sequentially.
  - Otherwise, skip those ***statements***.
- What is ==?

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

int main()
{
  int num1 = 0, num2 = 0;

  cout << "Please enter two numbers, "
       << "separated by a white space: ";
  cin >> num1 >> num2;

  if (num1 > num2)
    cout << "The larger one is " << num1;
  if (num1 < num2)
    cout << "The larger one is " << num2;
  if (num1 == num2)
    cout << "The two are equal";

  return 0;
}
```

The comparison operators

- == checks whether the two sides of it are **equal**.
  - Returns a **Boolean** value: true (non-zero) or false (zero).
- = and == are different!
  - When we write `a = 20`, it assigns 20 to `a`. 20 is then returned.
  - When we write `a == 20`, it checks whether `a` equals 20. Either **true** or **false** is then returned.
  - What happens to the following three programs?

```
int a = 0;
if(a == 1)
{
  cout << "here!";
}
```

```
int a = 0;
if(a = 1)
{
  cout << "here!";
}
```

```
int a = 0;
if(a = 0)
{
  cout << "here!";
}
```

- Do distinguish "**becomes**" and "**equals**"!
  - `a = 20` is read as "`a` becomes 20". `a == 20` is read as "`a` equals 20".

The comparison operators

- All the following comparison operators return a Boolean value.
  - `>`: bigger than
  - `<`: smaller than
  - `>=`: not smaller than
  - `<=`: not bigger than
  - `==`: equals
  - `!=`: not equals

The if statement

- In an **if** block, there may be **multiple** statements.
- A pair of **curly brackets** are used to define the block.

```
int a = 0;
if(a == 1)
{
  cout << "he";
  cout << "re!";
}
```

- We may drop { } if, and only if, there is **only one** statement under the if statement.

```
int a = 0;
if(a == 1)
  cout << "here!";
```

- What will happen to this program?

```
int a = 0;
if(a == 1)
  cout << "he";
  cout << "re!";
```

Our third C++ program

- Would you guess what does this program do?
- We use the **while** statement to **repeat** several statements.

```
while (condition)
{
   statements
}
```

  - If *condition* returns **true**, do *statements* sequentially and then go back to check *condition* again.
- What is **num1 = num1 - 1**?

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

int main()
{
  int num1 = 0, num2 = 0;
  cout << "Please enter two numbers, "
       << "separated by a white space: ";
  cin >> num1 >> num2;

  while (num1 > num2)
  {
    cout << "number 1 is " << num1 << "\n";
    num1 = num1 - 1;
    if (num1 = num2 * 2)
      break;
  }

  return 0;
}
```

The while statement

- **while** is nothing but an **if** that repeats.
- Consider the assignment operator **=** again:

```
num1 = num1 - 1;
```

  - Read it as "becomes": **num1** becomes **num1** minus 1.
  - First, **num1 - 1** is calculated and returned by the subtraction operator **-**.
  - This value is then assigned to **num1**.
- Now we fully understand this program:

```
while (num1 > num2)
{
  cout << "number 1 is " << num1 << "\n";
  num1 = num1 - 1;
  if (num1 = num2 * 2)
    break;
}
```

Syntax errors vs. logic errors

- A **syntax error** occurs when the program does not follow the standard of the programming language.

```
if (num1 > num2)
  cout << "The larger one is << num1;
if (num1 < num2)
  cout << "The larger one is " << num2
```

  - The compiler detects syntax errors.
- A **logic error** occurs when the program does not run as the programmer expect.

```
if (num1 > num2)
  cout << "The larger one is " << num1;
if (num1 < num2)
  cout << "The larger one is " << num1;
```

  - Programmers must detect logic errors by themselves.
  - The process is called **debugging**.

Outline

- Computer programming
- Our first C++ program: basic structure and **`cout`**
- Our second C++ program: variable declaration and **`cin`**
- Our third C++ program: the **`if`** and **`while`** statements
- **Formatting a C++ program**

Formatting a C++ program

- In a C++ program, semicolons are marks of the end of statements.
- White spaces, tabs, and new lines do not affect the compilation and execution of a C++ program.
  - Except strings and preprocessor commands.
- The following two programs are equivalent:

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

int main()
{
  cout << "Hello World! \n";
  return 0;
}
```

```
#include <iostream>
using namespace
std; int main
(){cout << "Hello
World! \n";return 0;}
```

Formatting a C++ program

- Maintaining the program in a good **format** is very helpful.
- While each programmer may have her own programming style, there are some general guidelines.
  - Move to a new line for each semicolon.
  - Align paired braces vertically.
  - Indent blocks according to their levels.
  - Write comments.

Formatting a C++ program

- Move to a **new line** for each semicolon.
  - Never put two statements in the same line!
- **Align** paired braces vertically.
  - Which one do you prefer?

```
int main()
{
  int a = 5;
  if(a < 5)
  {
    cout << "...";
    cout << "...";
  }
  return 0;
}
```

```
int main()
{
  int a = 5;
  if(a < 5)
  {
    cout << "...";
    cout << "...";
}
  return 0;
  }
```

```
int main() {
  int a = 5;
  if(a < 5) {
    cout << "...";
    cout << "...";
  }
  return 0;
}
```

Indentions

- **Indent** blocks according to their levels.
  - Which one do you prefer?

```
int main()
{
  int a = 5;
  if(a < 5)
  {
    cout << "...";
    cout << "...";
  }
  return 0;
}
```

```
int main()
{
int a = 5;
if(a < 5)
{
cout << "...";
cout << "...";
}
return 0;
}
```

```
int main()
{
  int a = 5;
  if(a < 5)
  {cout << "...";
    cout << "...";
  }
    return 0;
}
```

Comments

- **Comments** are programmers' **notes** and will be ignored by the compiler.
- In C++, there are two ways of writing comments:
  - A single line comment: Everything following a \\ in the same line are treated as comments.
  - A block comment: Everything within * and *\ (may across multiple lines) are treated as comments.

```
/* Ling-Chieh Kung's work
   for the first lecture */

#include <iostream>
using namespace std;

int main()
{
  cout << "Hello World! \n";
  return 0; // the program terminates correctly
}
```