



COMP2012H Honors Object-Oriented Programming and Data Structures

Topic 1: Introduction

Prof. Gary Chan

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China



Who Is This Guy?

- Professor, Computer Science and Engineering, HKUST
- Affiliate Professor, Innovation, Policy and Entrepreneurship, HKUST(GZ)
- I build advanced novel systems for applications, many requiring C++ for fast operations
- Office: Room 2539 (Lifts 25/26)
- Email: gchanust.hk
- Phone: 2358 6990
- Office hours: Right after class or by appointment

COMP2012H = COMP2011 + COMP2012

- Bookmark this: <https://course.cse.ust.hk/comp2012h>
- A highly compressed “2-in-1” package deal. If you want to take just COMP2011, don’t take this course!
- Require excellent working knowledge on at least one language
- Covering C++ syntax, object-oriented programming, and how to use it to implement some important data structures and applications
 - ▶ List, queue, stack, hash, (binary) trees, etc.
- Mercilessly fast-paced, expecting large appetite and digestive capacity for knowledge
- A programming-intensive and thinking-intensive course. You need to think smart and program smart in order to keep up.
- The labs, lectures and programming assignments are chasing each other in random sequence over the semester.
 - ▶ Do NOT expect that lectures always lead labs, and labs always lead the programming assignments. In fact, they sometimes do not. You need to self-learn.

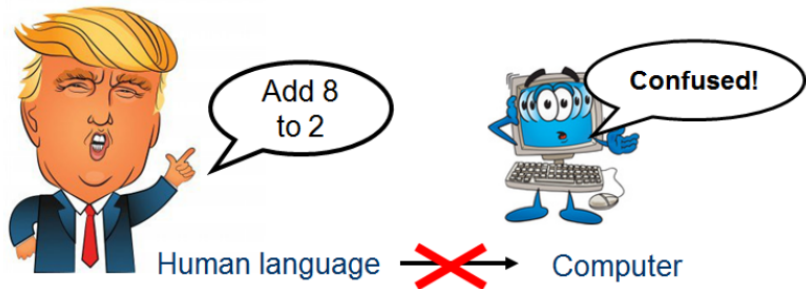
Writing Good Codes: Rise of The Machines

- Know what a machine (compiler) would do



Computer Programming

- Learning computer programming is just like learning a natural language such as English, Japanese, Korean, etc.
- Although they are similar, it doesn't mean they are exactly the same.



- Don't worry. Actually this is good. Since **computer programming** is **more systematic and should be much easier to learn**, in my opinion. ;) (Good news!)

Programming Languages

- Computer programs are written in programming languages.
- Different to those human languages, a programming language defines A SET OF INSTRUCTIONS in SPECIFIC FORMAT that can be given to a computer.
- Two important issues on writing programs:
 1. Program syntax – Is the grammar of the instructions correct?
 2. Program logic – Is the program able to solve the problem?

C/C++
C#
SQL Server
Apache
Visual Basic
Java
ASP
IIS
Java Servlets



Machine Code: Can You Understand This?

```
0000100100101110011001100110100101101100011001010000100100100010011011000110010101100011011101000111
0101011100100110010100110001001011100110001100100010000010100110011101100011011000110011001001011111
011000110110111011011010111000001101001011011000110010101100100001011100011101000001010001011100111
001101100101011000110111010001101001011011101101110000010010010001000101110011101000110010101111000
0111010000100010000010100000100100101110011000010110110001101001011001110110111000100000001101000000
10100000100100101110011001110110110001101110110001000100000101101001000001011010110000101101001
01101110000010100000100100101110011101000111001011100000110010100001001001000000110110101000010110
10010110111000101100001000110110011001110101101110011000110111010001101110110111000001010
0000100100101110011100000110010011011101100110000100100110000001101000000101001101101011000010110
1001011011100011101000001010000010010010000100100011010100000101001001001111010011000100111101000111
0101010101000101001000110010000000110000000010100000100101110011011000010111011001100101001000000010
010101110011011100000001010000101100110001000110000011100000101100001001011100110111000000001010
00001010000010010010000100100011010100000100100100111010011000100111010001110101011010010001010010
0011001000000011000100001010000010010110110101101110110110001000000011000100101100001001010110111
00110000000010100000100101110011011101000010000000100101011100110000001011000100101010110
01100111000000101101001100100011000001011101000010100000100101101101011011101101100010000000110010
00101100001001010110111001100000000101000001001011100110110100001000000010010101110011000000010
1100010110110010101100110011100000010110011001001010000101101000001010000010010110110001100100
00100000010110110010010110110011100000010110100110010001100000010110100101100000100101011110011
0000000010100000100101101100011001000010000001011011001001010110011001110000001011010011001000110100
010111010010110000100101011011100110001000010100000100101100001011001000110010000100000001001010110
11110011000000101100001001010110111001100010010110000100101011011100110000000010100000100101110011
0111010000100000001001010110111001100000010100010110110010010101100110011100000010110100110010011
100001011101000010100000100101101101011011101110110001000000011000000101100001001010100100110000
000010100000100101100010001000000010111001001100010011000011000100001010000010010110111001101110111
000000001010001011100100110001001100001100010011101000001010000010010111010011001010111010000001010
000010010111001001100101011100110111010001101110111001001100101000010100010111001001100010011000110
0110011001010011000100111010000010100000100100101110011100110110100101111010011001010000100100100000
01101101011000010110010110111000101100001011001001100001001100110010010011000100100101011011010
11010110000101101001011011100000101000001001001011100110100101100100011001001011101100110000001001
00100010010001110100001101000011001110100010000000101000010001110100111001010101001001001000000011
0010001011100011100000101110001100010010001000001010
```

Assembly Language: How About This?

```
main:
```

```
    !#PROLOGUE# 0
```

```
    save %sp,-128,%sp
```

```
    !#PROLOGUE# 1
```

```
    mov 1,%o0
```

```
    st %o0,[%fp-20]
```

```
    mov 2,%o0
```

```
    st %o0,[%fp-24]
```

```
    ld [%fp-20],%o0
```

```
    ld [%fp-24],%o1
```

```
    add %o0,%o1,%o0
```

```
    st %o0,[%fp-28]
```

```
    mov 0,%i0
```

```
    nop
```



High-Level Language: Is This Better Now?

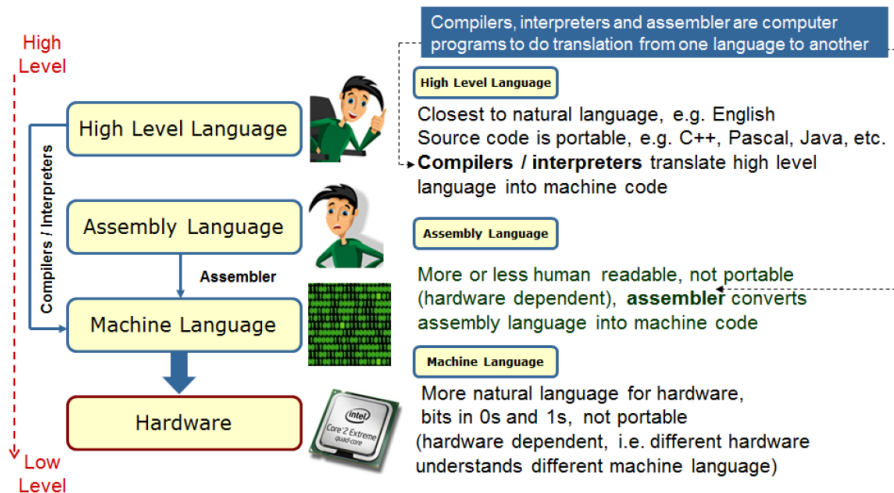
```
int main( )  
{  
    int x, y, z;  
  
    x = 1;  
    y = 2;  
    z = x+y;  
  
    return 0;  
}
```



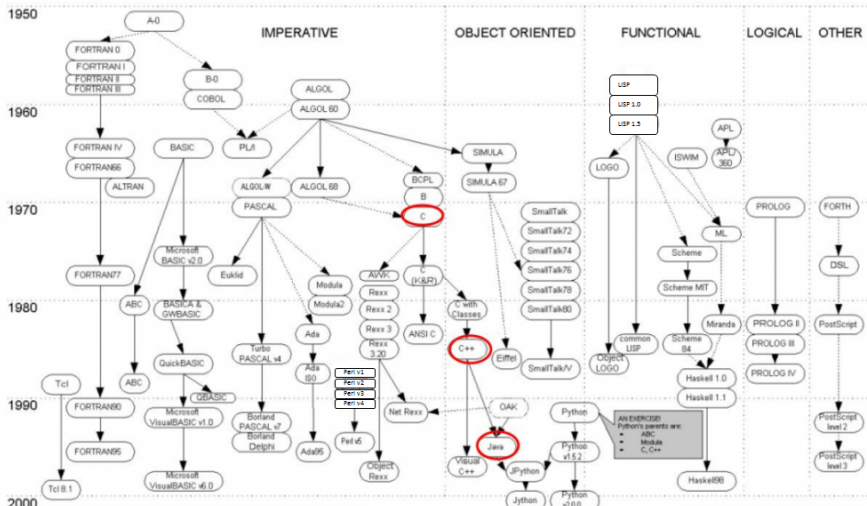
Write a Program to Sum 2 Numbers

- There are 3 integer-value-holding objects: x, y, and z.
- x and y have the value of 1 and 2 respectively.
- z's value is the sum of x's and y's.

Levels of Programming Languages



Chronology of Some High Level Programming Languages



The big truth includes ~2000 languages & very complex associations!
Read for more: <http://perso.wanadoo.fr/levenez/lang/history.html>

<http://amstel.science.uva.nl/~fotisg/python>

Fotis Georgatos <gef@ceid.upatras.gr>
Amstel Institute, Amsterdam, June 2002

Which Programming Language Are We Going to Use?

We are going to use C++ in this course!

- Why C++?

Read the FAQ from the designer of C++, Bjarne Stroustrup.

- Which C++?

- ▶ The language has been **evolving**:

C++ 1983 \Rightarrow C++ 1998 \Rightarrow C++ 2003 \Rightarrow C++ 2011 $\Rightarrow \dots$

- ▶ We will learn C++11 (but not all the new features).

- Which compiler?

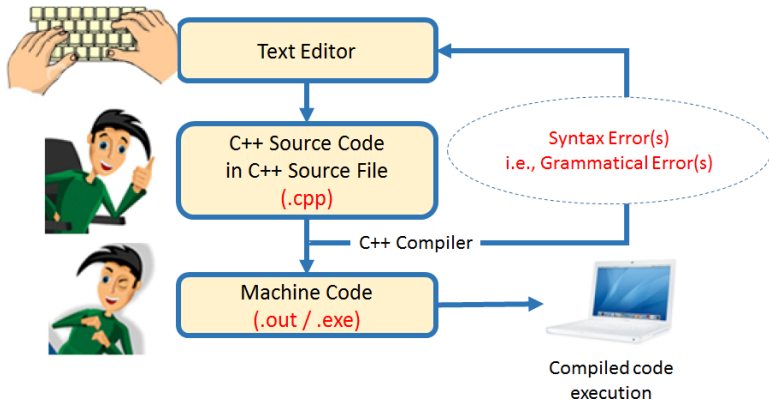
GNU gcc/g++. It is free.

(The compiler you will use is C++11-compliant.)

- Which IDE (integrated development environment) for writing programs?

VSCode. It is free and supported by many operating systems such as Windows, Mac OS, and Linux.

Development Cycle of a C++ Program



- A **compiler** translates **source programs** into **machine codes** that run directly on the target computer.
- For example, `a.cpp` \rightarrow `a.out` (or `a.exe`).
- Some C++ compilers: `gcc/g++`, `VC++`.

Example: Hello World!

```
/*  
 * File: hello-world.cpp  
 * A common program used to demo a new language  
 */  
  
#include <iostream>           // Load info of a Standard C++ library  
using namespace std;         // Standard C++ namespace  
  
int main()                    // Program's entry point  
{  
    /* Major program codes */  
    cout << "Hello World!" << endl;  
  
    return 0;                 // A nice ending  
}
```



Write, Compile, and Run a Program in a Terminal

STEP 1 : Write the program using an **editor**.

e.g., **VSCode**, **vi** (Unix/Linux), **MS Word** (Windows)

STEP 2 : Save the program into a file called **hello-world.cpp**.

STEP 3 : Compile the program using **g++** compiler.

```
g++ -o hello-world hello-world.cpp
```

If you don't specify the output filename using the “**-o**” option, the default is **a.out**.

```
g++ hello-world.cpp
```

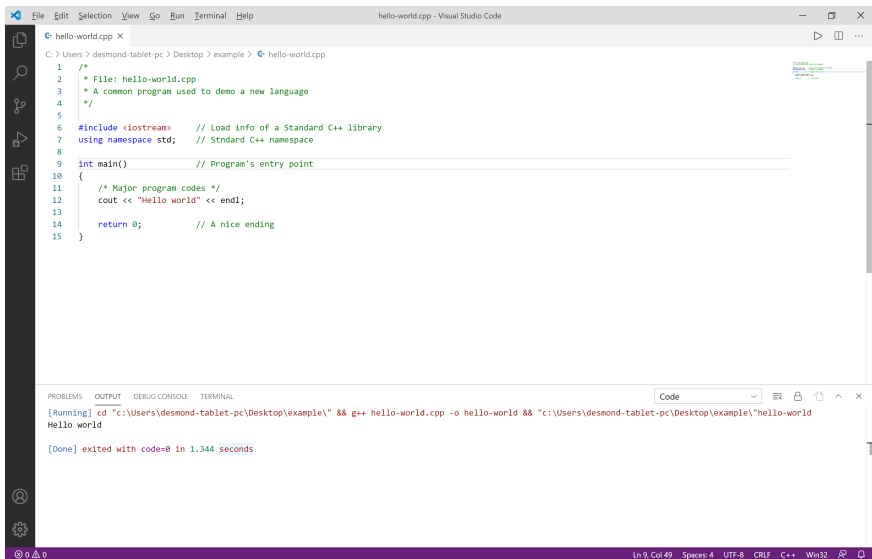
STEP 4 : Run the program in a terminal (command window):

```
linux:: hello-world  
Hello World!
```

VSCode IDE for C/C++

LAB

In the lab, you will use VSCode (similar to MS Visual Studio).



The screenshot displays the Visual Studio Code interface. The main editor window shows a file named `hello-world.cpp` with the following C++ code:

```
1  /*  
2   * File: hello-world.cpp  
3   * A common program used to demo a new language  
4   */  
5  
6  #include <iostream>    // Load info of a Standard C++ library  
7  using namespace std;  // Standard C++ namespace  
8  
9  int main()             // Program's entry point  
10 {  
11     /* Major program codes */  
12     cout << "Hello world" << endl;  
13  
14     return 0;          // A nice ending  
15 }
```

Below the editor, the TERMINAL panel shows the command to compile and run the program, and the output:

```
[Running] cd "c:\Users\desmond-tablet-pc\Desktop\example\" && g++ hello-world.cpp -o hello-world && "c:\Users\desmond-tablet-pc\Desktop\example\hello-world  
Hello world  
  
[Done] exited with code=0 in 1.344 seconds
```

The status bar at the bottom indicates the current position in the file: Ln 9, Col 49, Spaces: 4, UTF-8, CRLF, C++, Win32.

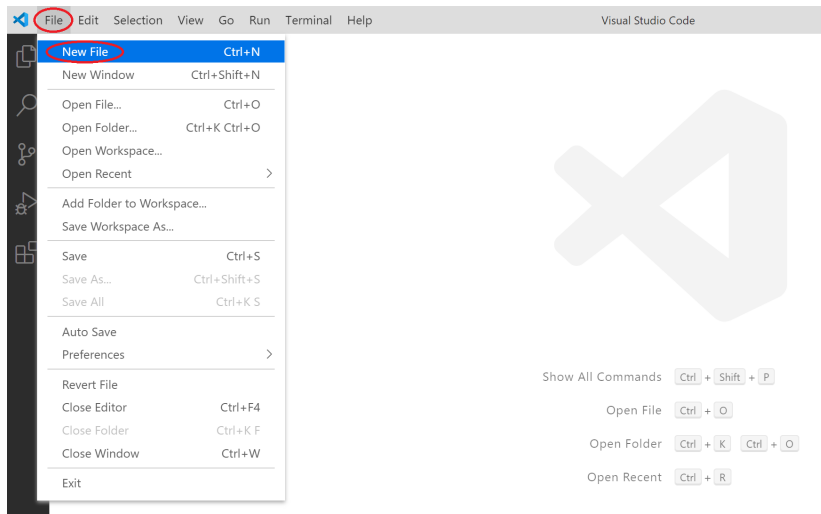


Step 1	Create a new file	Ctrl-N (or Click File → New File)
Step 2	Write program	VSCode built-in editor
Step 3	Save program	Ctrl-S (or Click File → Save)
Step 4	Compile and run program	F1 → “Run code”

Step 1a: Create a new file

LAB

- Ctrl-N (or Click “File” → “New File”)



Show All Commands `Ctrl` + `Shift` + `P`

Open File `Ctrl` + `O`

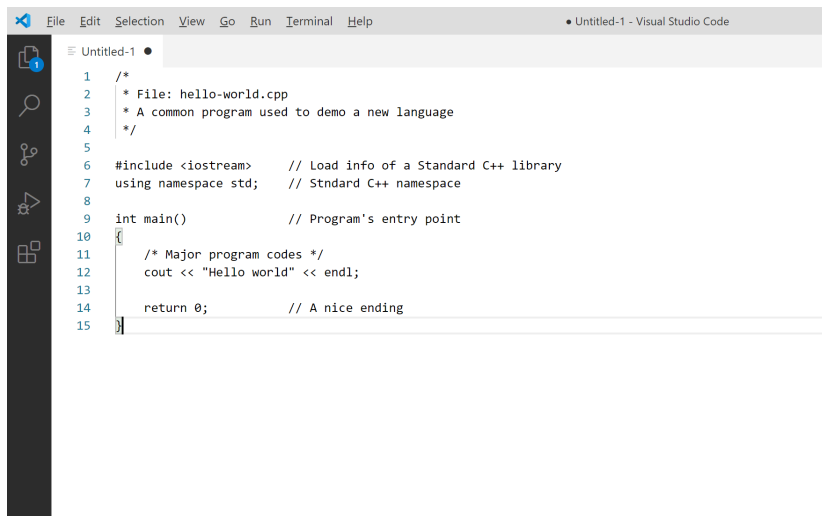
Open Folder `Ctrl` + `K` `Ctrl` + `O`

Open Recent `Ctrl` + `R`

Step 2: Write program

LAB

- Write your program



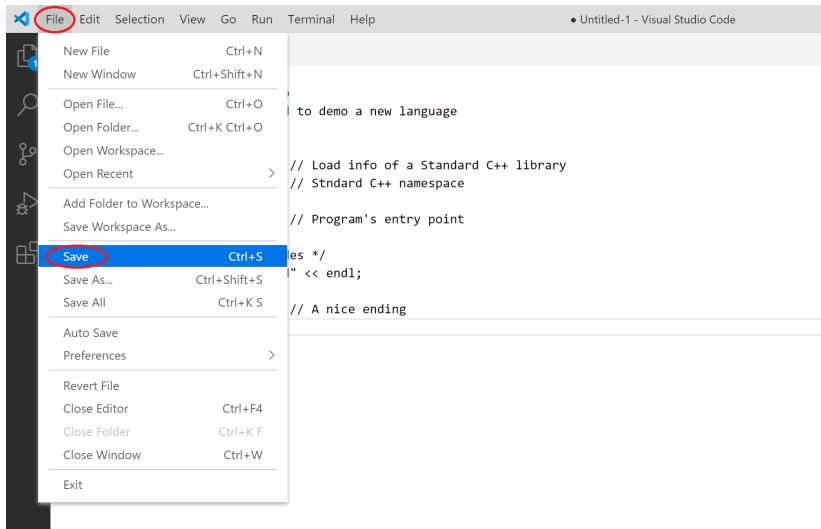
The screenshot shows the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates 'Untitled-1 - Visual Studio Code'. The left sidebar contains icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The main editor area displays a C++ program in a file named 'Untitled-1'. The code is as follows:

```
1  /*
2  * File: hello-world.cpp
3  * A common program used to demo a new language
4  */
5
6  #include <iostream>    // Load info of a Standard C++ library
7  using namespace std;  // Standard C++ namespace
8
9  int main()            // Program's entry point
10 {
11     /* Major program codes */
12     cout << "Hello world" << endl;
13
14     return 0;          // A nice ending
15 }
```

Step 3a: Save program

LAB

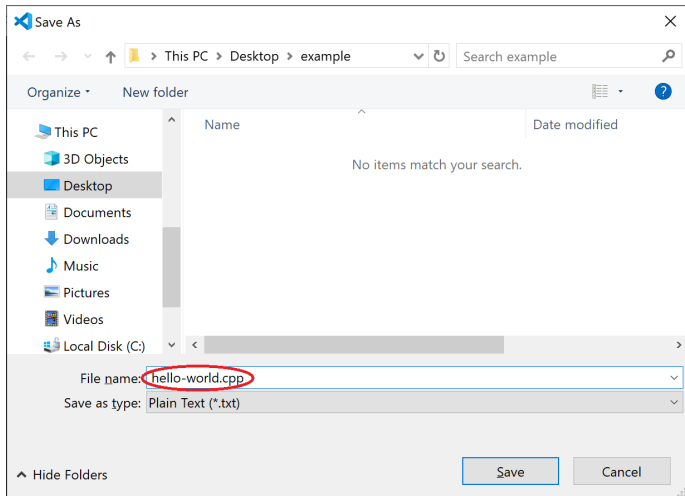
- Ctrl-S (or File → Save)



Step 3b: Save program

LAB

- Choose a location, give it a proper name, and click “Save”.
Note: make sure the name is something which ends with .cpp to indicate that is a C++ source file (e.g., hello-world.cpp).



Step 4: Compile and run program

LAB

- F1 → click “Run code”

The screenshot shows the Visual Studio Code interface with the file `hello-world.cpp` open. The code is a simple C++ program that prints "Hello world". A right-click context menu is open over the code, with the **Run Code** option highlighted. Below the main menu, there are sections for "recently used" and "other commands" with their respective keyboard shortcuts. At the bottom, the **OUTPUT** panel is active, showing the command executed to compile and run the program, and the output "Hello world" which is circled in red.

```
File Edit Selection View Go Run Terminal Help
hello-world.cpp - Visual Studio Code

hello-world.cpp x
C: > Users > desmond-tablet-pc > Desktop > example
1  /*
2   * File: hello-world.cpp
3   * A common program used to demo a
4   */
5
6  #include <iostream>    // Load in
7  using namespace std;  // Standard
8
9  int main()            // Program
10 {
11     /* Major program codes */
12     cout << "Hello world" << endl;
13
14     return 0;         // A nice e
15 }
```

Run Code (Ctrl + Alt + N) recently used

other commands

- Ctrl + Alt + UpArrow
- Ctrl + Alt + DownArrow
- Shift + Alt + I
- Ctrl + K
- Ctrl + C
- Ctrl + D

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

[Running] cd "c:\Users\desmond-tablet-pc\Desktop\example\" && g++ hello-world.cpp -o hello-world

Hello world

[Done] exited with code=0 in 1.072 seconds

Main: the Entry Point

- Every program must have exactly one and only one `main()` function.

Simple Form of the `main` Function

```
int main () { ... }
```

General Form of the `main` Function

```
int main (int argc, char** argv) { ... }
```

(We'll talk about `argc` and `argv` later.)

- Between the braces “{” and “}” are the program codes consisting of zero or more program **statements**.
- Each simple C++ statement ends in a semicolon “;”.

C++ Comments

- Use `/* ... */` for **multiple-line comments**.

```
/*  
 * A common program used to demo a new language  
 */
```

- **Single-line comments** start with `//`.

```
// Program's entry point
```

- Comments are just for human to read.
- They will **not** be translated by the compiler into machine codes.

#include and Standard C++ Libraries

- `#include` will include information of a **library** — a collection of sub-programs. e.g. `#include <iostream>` gets the information of the **standard C++ library** called **iostream** that deals with I/O:
 - ▶ **cin**: an object to read, e.g., from the keyboard or file
 - ▶ **cout**: an object to print out, e.g., to the screen or file
 - ▶ **cerr**: an object to print error message, e.g., to the screen or file

Examples

```
// endl means "end of a line"
cout << "Einstein: God does not play dice." << endl;

// You may also break down the message in several lines
cerr << "Error: "
      << "There is no stress and tension in HKUST!"
      << endl;
```

- These library information files are called **header files**.

#include and User-defined Libraries

- You may also define your *own* library.
- Again you need to use `#include` to include its information into your sub-programs.
- Example: `#include "drawing.h"` gets the information of a *user-defined C++ library* about drawing.
- By convention, the *header file* of a *user-defined library* ends in `".h"` or `".hpp"`, while *Standard C++ library* header files have *no* file suffix.
- Also by convention, the *header file* of a *user-defined library* is delimited using double-quotes `"..."`, while *Standard C++ library* header files use `< ... >`.

C++ is a Free Format Language

- Extra blanks, tabs, lines are ignored.
- Thus, codes may be indented in any way to enhance readability.
- More than one statement can be on one line.
- Here is the same Hello World program:

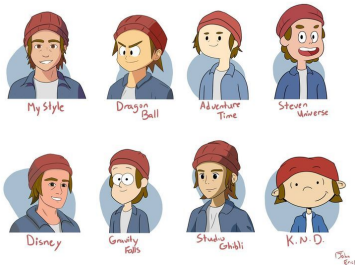
```
#include <iostream> /* File: hello-world-too.cpp */  
using namespace std; int main (int argc,  
    char** argv) { cout<<"Hello World!"<<endl;return 0;}
```

- On the other hand, a single statement may be spread over several lines.

```
cout << "Hello World!"  
    << endl;
```

Good Programming Style

- Place **each** statement on a line by itself.
- For **long** statements
 - ▶ if possible, break it down into several shorter statements.
 - ▶ wrap it around with proper indentation (since extra space doesn't matter!)
- Use blank lines to **separate** sections of related codes that together perform some action.
- **Indent consistently**. Use the same indentation for the same block of codes.



Programming as Problem Solving



- **Understand** and **define** the problem clearly.
 - ▶ What are the input(s) and output(s)?
 - ▶ Any constraints?
 - ▶ Which information is essential?
- **Develop** a solution.
 - ▶ Construct an algorithm.
- **Translate** the algorithm into a C++ program.
- **Compile** the program.
- **Test** the program.
- **Debug** the program.
- **Document** the program as you write the program.
- **Maintain** the program
 - ▶ modify the codes when conditions change.
 - ▶ enhance the codes to improve the solution.

What Makes a Good Program?

- Correctness
 - ▶ Meets the problem requirements
 - ▶ Produces correct results
- Easy to read and understand
- Easy to modify
- Easy to debug
- Efficient
 - ▶ Fast
 - ▶ Requires less memory



That's all!

Any questions?

