IM 1003: Programming Design

Introduction

Ling-Chieh Kung

Department of Information Management
National Taiwan University

February 18, 2013

Outline

- **Computer programs**
- The C++ programming language
- The basic structure of C++ programs
- Formatting a C++ program

Computer programs

- 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: They cannot "think" (at least at this moment).
  - People (programmers) need to tell them what to do.

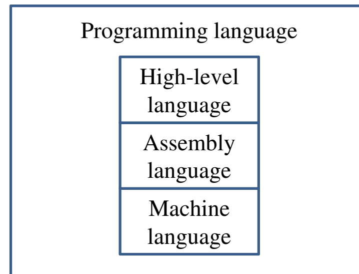
Computer programming

- 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.

```
int a = 0, b = 5;
int c = a + b;
cout << c;     // c must be 5
```

Programming languages

- People and computers talk in programming languages.
- A programming languages may be one of the following:



Machine languages

- A machine language contains only binary values like 01011011….
- Machines can follow instructions written in machine languages.
  - 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."
- Machine languages are machine-dependent: A program written in one machine language can only run on one type of machine (CPU).
- Machines can only read machine languages.
- Though people can program in machine language directly (with a very huge dictionary), it is too inefficient.

Assembly languages

- Instead of writing numbers directly, we may label operations, registers, memory addresses, and anything else by readable items.
  - Label 000000 as **`ADD`**, 000010 as **`JUMP`**, etc.
- Then we can write programs by these items:
  - ```
    ADD ax, bx
    MOV cx, ax      // then register cx contains ax + bx
    ```
- The collection of these readable items and the associated grammar forms an assembly language.
  - The first programming language is an assembly language.

Machine and assembly languages

- To program in assembly languages, we rely on an "assembler".
  - An assembler translates an assembly-language instruction into the corresponding machine-language binary codes.
  - The mapping is "one-to-one".
- In developing large-scale software, programming in assembly languages is still not efficient enough.
- Moreover, machine and assembly languages are not portable.
  - Different types of machines need different machine/assembly languages.
    ```

High-level languages

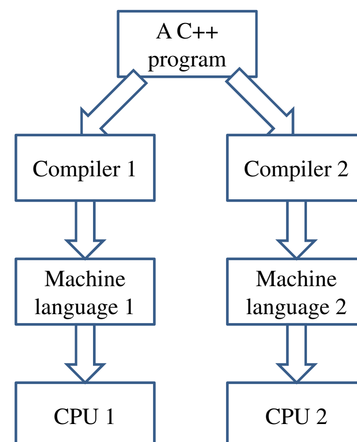
- Most application software are developed in high-level languages.
  - A high-level language looks more like human languages.
  - More tools helping programmer increase efficiency are added.
- There are many many many high-level languages:
  - Some others: Basic, Quick Basic, Visual Basic, Fortran, COBOL, Pascal, Delphi, C, Perl, Python, Java, C#, PHP, Matlab, etc.
  - The language we study in this course, C++, is also a high-level language.

Interpreters and compilers

- Programmers rely on "interpreters" and "compilers" to translate instructions written in high-level languages to machine-language binary codes.
  - C++ is a compiled language.
  - Basic and Perl, for example, are interpreted languages.
- An interpreter translates instructions one by one. Once an instruction is translated, it is executed immediately.
- A compiler first reads the whole program and then translate all instructions at once. All translated instructions are executed after the translation.
- In this course, we focus on compilers.

Portability of high-level languages

- Most high-level languages allow portability.
- For example, 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.

  - Remark: Unix is written in C and MS Windows is written in C++.
  - Some application developers mix assembly codes in their program to enhance efficiency.

High-level and assembly languages

- C/C++ is sometimes called a "mid-level" language.
  - It allows a C++ programmer to "access" the memory.
  - We will see this when we study pointers.
- With such low-level functionality, C/C++ is very powerful.
  - And dangerous…
- In this course, we will study only C++ as a high-level language.
  - In the next semester, you may get some training in assembly languages in Computer Organization and Structure in the IM department.
  - You are encouraged to take Computer Organization and Assembly Languages, Computer Architecture, Systems Programming, and Compiler Design in the CSIE department to learn more about low-level languages and computer system architecture.

Outline

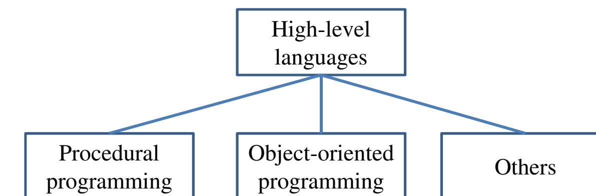
- Computer programs
- **The C++ programming language**
- The basic structure of C++ programs
- Formatting a C++ program

The C++ programming language

- C++ is developed by Bjarne Stroustrup starting in 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 the functionalities of classes and objects (and many more) into C.
- C++ is (almost) a superset of C.
  - Most C programs can be complied by a C++ compiler.

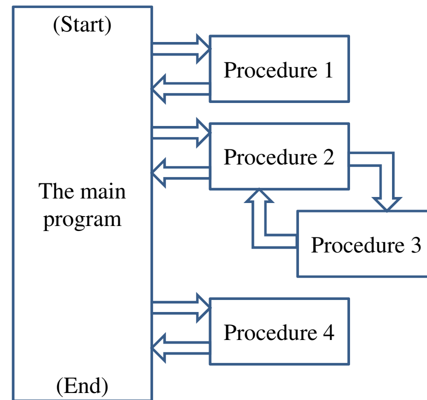
Perspectives of designing programs

- High-level programming languages can be categorized according to the perspectives of designing the program.
  - In this course, we talk about procedural and object-oriented languages.
  - There are many more that will be introduced in Programming Languages in the IM department.



Procedural languages

- The main idea of procedural programming is to construct a program by combining pieces of modules.
  - These modules are generally called procedures.
  - In C/C++, procedures are called functions.



Object-oriented languages

- Some large-scale software have many "items" that are similar.
  - For example, in your MS Windows, there are so many "windows".
  - They may be of different sizes and functions, but the attributes (height, width, caption, etc.) and operations (resizing, maximizing, closing, etc.) they need are all the same.
- Instead of designing software based on defining tasks (i.e., procedures), people may design based on defining these items.
  - In C/C++, these items are called objects.
  - The development of GUI (graphical user interface) is one of the main motivations of OOP (object-oriented programming).
  - Does it make sense now that Unix is written in C and MS Windows is written in C++?

Object-oriented languages

- C++ is an object-oriented language.
  - As it originates in C, a procedural language, it is easier to start with the procedural part.
  - Afterwards, we will study the object-oriented part.
- Some people say that C++ is not a pure object-oriented language.
  - One may write a correct C++ program without using objects.
  - For some other OO languages, such as Java and C#, this is impossible. These languages are sometimes called pure OO languages.

Why C++?

- C++ is harder than C:
  - In C we do not need to study objects, classes, inheritance, polymorphism, operator overloading, etc.
- C++ is harder than Java:
  - In Java we do not need to study pointers and many pointer-related topics.
- But that means C++ is powerful!

Application-level	Java	C++
System-level	C	

C++ is hard!

- Not all of you will program in C++ after you graduate.
- But once you really know C++, it is easy to learn any other procedural or OO languages.
- How to "really" know C++?
  - Being diligent is this course is necessary but not sufficient!
  - Take Data Structures in IM.
  - Take OOP in CSIE or ESOE.
  - Take Computer Organization and Assembly Languages in CSIE (if you are still interested in it after taking Computer Organization and Structure in IM).
  - Study design patterns.
  - And many many more!

Outline

- Computer programs
- The C++ programming language
- **The basic structure of C++ programs**
- Formatting a C++ program

Our first C++ program

- In most computer programming courses, we start with 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 `cout` instruction.

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

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

The preprocessor

- Preprocessor commands, which begins with **#**, performs some actions before the compiler does the translation.
- The **include** command here is to include a header file:
  - Files containing definitions of common variables and functions.
  - Written to be included by other programs.

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

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


The preprocessor

- **#include <iostream>**
  - **iostream** is part of the C++ standard library. It provides functionalities of data input and output.
  - Before the compilation, the compiler looks for the **iostream** header file and copy the codes therein to replace this line.
  - The same thing happens when we include other header files.

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

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


Including header files

- In this program, we include the **iostream** file for the **cout** object.
- With **<** and **>**, the compiler searches for iostream in the C++ standard library.
- We may write our own functions into self-defined header files and include them by ourselves:
  - **#include "C:\myHeader.h";**
  - Use quotation marks instead of angle brackets.
  - A path must be specified.
- We will not use self-defined header files until the second half of this semester.

Namespaces

- What is a namespace?
- Suppose all roads in Taiwan have different names. In this case, we do not need to include the city/county name in our address.
  - This is why we do not need to specify the district for an address in Taipei.

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

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

Namespaces

- However, there are so many roads sharing the same name.
  - So on top of road names, we need "another level of names".
- A C++ namespace is a collection of names.
  - For C++ variables, functions, objects, etc.
  - The object **cout** and all other items defined in the C++ standard library are defined in the namespace **std**.

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

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


The scope resolution operator (::)

- By writing **using namespace std;**, whenever the compiler sees a name, it searches whether it is defined in this program or the namespace **std**.
- Instead, we may specify the namespace of **cout** each time when we use it with the scope resolution operation **::**.

```
#include <iostream>

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


Namespaces

- Most programmers do not need to define their own namespaces.
  - Unless you really want to name your own variable/object as **cout**.
- Typically a **using namespace std;** instruction suffices.
- We will revisit namespaces and the scope resolution operator later in this semester.

The main function

- A C++ Program always runs from the first line of the main function to the last line.
  - The main function is always named **main()**.
  - One program, one main function.
- A pair of braces (curly brackets) defines a block.
  - Here within **{** and **}** instructions of the main function is specified.

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

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

The main function

- **int main()**:
  - The function header line.
  - **int** (stands for integer) specifies that the function should return an integer as the returned value.
  - **main**: the function name.
  - argument for a function is included within **(** and **)**.
- **return 0**:
  - Return the integer 0 as the returned value.
  - Tell the operating system that everything is fine.

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

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


Statements

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

```
#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 symbol **<<** marks the direction of data flow.
  - Data flow like streams.
- **"Hello world! \n"** is a string.
  - Characters contained in 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… where is that \n?
- In C++, the slash symbol \ marks the beginning of an escape sequence.
  - An escape sequence is some kind of special "characters".
  - **\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	Display
`\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 the one statement

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

  display the same thing.
- Remark: the following statement

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

  is wrong!

Our first C++ programs as a whole

- Now we fully understand our first C++ program.

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

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

- Remark: Some words are colored because they are C++ reserved words (keywords), which serve for special purposes. We will talk about them soon.

Outline

- Computer programs
- The C++ programming language
- The basic structure of C++ programs
- **Formatting a C++ program**

Formatting a C++ program

- Recall that in C++ 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.
  - Let the editor color the codes.
  - Move to a new line for each semicolon.
  - Align paired braces vertically.
  - Indent blocks according to the levels.
  - Write comments.

Comments

- Comments are programmers' note for the program.
- They 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.

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
}
```

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 << "...";
  }
  return 0;
}
```

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

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


Indentions

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

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

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

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

  - Remark: Indentions are typically done with tabs. We use two white spaces on slides because we need to save spaces.

Coloring

- Most modern C++ editors color the codes.
  - My style:

```
/* 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
}
```

  - Feel free to create your own coloring style.