A Handout For Students in Kan Chiao

Handsome Liu

April 17, 2018

Contents

Part I

Introduction

Chapter 1

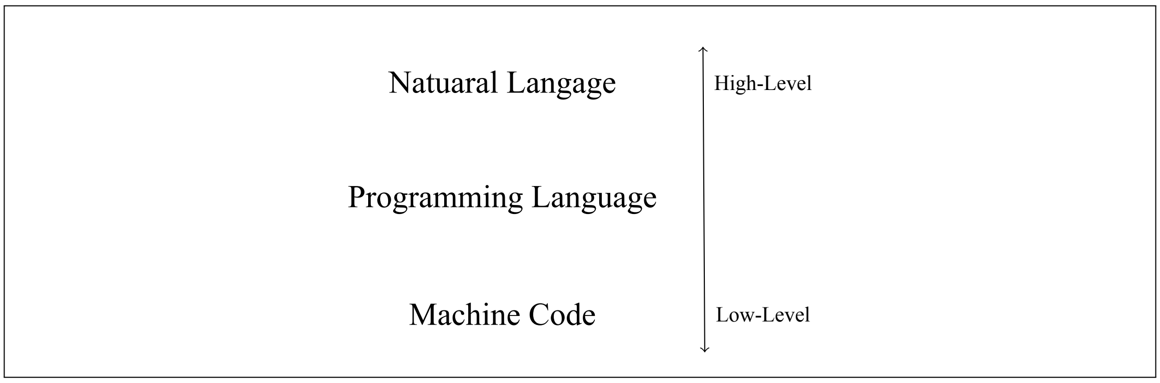
C/C++ Programming Language

1.1 Programming Language

1.1.1 Programming Language and Natural Language

Different from **natural language**, the language that we using to communicate with other *human beings*, **programming language** is a kind of language that we using to communicate with *machines*.

As you may expect, computers are not smart enough to comprehend C/C++ programming language directly. The language used by CPUs is called *machine code*. As a result, we need a **compiler** to achieve such translation. The concept of **compile** and **interpret** will be introduced in later section.



1.1.2 Source File

The code you wrote is called the **source code**, and the file containing your source code is called the **source file**.

1.1.3 Compile and Interpret

To **compile** means to translate *the whole source file* into another language, say, machine code.

To **interpret** means to translate the source code *line by line* and execute it immediately.

Programming languages that needs to be compiled are called **compiled language**, while those that needs to be interpreted are called **interpreted language**.

1.2 C/C++ Language

C programming language is a programming language developed by *Dennis Ritchie* in 1969 1973. It is general-purpose, efficient and easy-to-read, which makes it the first choice for informatics competition.

Based on the syntax of C, *Bjarne Stroustrup* create C++ programming language to achieve *OOP (object-oriented programming)*.

1.2.1 Source File and Header File

Sometimes, it's kind of stupid to define all things in a single file. In C/C++, you can define things in some other files. The file that contains the `main()` function is called **source file**, and the files you used to define things in them are called **header files**. In this book, we will not learn how to write your own header files, but we will learn how to use others' header files.

1.2.2 Compiler for C/C++

C and C++ are both **compiled languages**, which means we need a compiler to compile your source code in order to execute it. The most-used one in Windows system is called `mingw`, while the most-used one in Linux system is called `gcc`.

1.3 IDE (Intergrated Development Environment)

IDE is a kind of application that provides facilities to programmers for coding. An IDE usually consists of:

- a code editor (with syntax highlighter)
- automatically building tools (call the compiler to compile your source code, and execute it automatically)
- a debugger (tells you which line of your code disobeys the rule of syntax)

1.3.1 Code::Blocks

Code::Blocks is a free, open-source IDE that supports many programming languages. Actually, Code::Blocks itself is developed in C++.

1.4 Run on Your PC

1.4.1 IDE and Compiler Installation

Go to `http://www.codeblocks.org/downloads/26`.

If you are using Windows, please choose *codeblocks-xx.xxmingw-setup.exe*.

If you are using Linux, please choose the one corresponding the operating system you use.

If you are using Mac, you don't have any choice, just press the only link for Mac OS X.

After downloading, follow the installation guide until the installation is complete.

1.4.2 Test

After finishing the installation, press `Ctrl+Shift+N` to open a new file. Paste the code below, and press the button with gear and green triangle to execute it.

```c
#include<stdio.h>
int main(){
    printf("Handsome Liu is so handsome!\n");
}
```

If you've done all the steps above correctly, you should see a windows with text `Handsome Liu is so handsome!` on it. However, if there is not, please check all the steps again.

Part II

Getting Started

Chapter 2

Basic Concepts

In this chapter, we will introduce some basic concepts you need to know.

2.1 Data Type

As you have known, computers stores binary data. However, to we human beings, binary data is hard to manipulate or to read. Data types are some ways to interpret those binary data.

2.1.1 Fundamental Data Type

Fundamental data types, or built-in data types, are provided by the compiler, so you can simply use it without define them.

There are 3 common fundamental data types you need to know, and will be discussed in the later chapters. They are integers (`int`), characters (`char`), and floating point numbers (`double`).

2.1.2 Derived Data Type

Derived data types are some data types that need to be constructed with advanced syntax. Using them, you can create new data types from fundamental data type.

There are many derived data types in C/C++, such as array, structure, union, pointer, and enumeration. We will focus on array and pointer in later chapters.

2.2 Keyword

Keywords, or reserved words, are some words that have fixed meaning in C/C++, and you should NOT name variables after them, just like you should not name a person ”He” or ”They”.

The following list are the keywords in C/C++.

`and`	`and_eq`	`asm`
`auto`	`bitand`	`bitor`
`bool`	`break`	`case`
`catch`	`char`	`class`
`compl`	`const`	`const_cast`
`continue`	`default`	`delete`
`do`	`double`	`dynamic_cast`
`else`	`enum`	`explicit`
`export`	`extern`	`false`
`float`	`for`	`friend`
`goto`	`if`	`inline`
`int`	`long`	`mutable`
`namespace`	`new`	`not`
`not_eq`	`operator`	`or`
`or_eq`	`private`	`protected`
`public`	`register`	`reinterpret_cast`
`return`	`short`	`signed`
`sizeof`	`static`	`static_cast`
`struct`	`switch`	`template`
`this`	`throw`	`true`
`try`	`typedef`	`typeid`
`typename`	`union`	`unsigned`
`using`	`virtual`	`void`
`volatile`	`wchar_t`	`while`
`xor`	`xor_eq`	

2.3 Identifier

Identifier is a kind of string that can be used to name a variable. An legal identifier only contains capital English alphabets, small English alphabets, numeric characters and underscore, and begin with non-numeric characters.

Capital English alphabets: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
Small English alphabets: `abcdefghijklmnopqrstuvwxyz`
Numeric characters: `0123456789`
Underscore: _

Examples:

```
handsomeLiu
HandsomeLiu
HandsomeLiu012
handsome_liu_012
__handsomeLiu
_____
```

Non-examples:

```
012handsomeLiu   (begin with numeric characters)
handsome-liu     (hyphen (-))
handsome liu     (space ( ))
```

2.4 Variable

Variable is a method to store data. You can use the variable as the value it stores.

Before using a variable, you need to declare it, which means to specify its data type and name. After declared, the data type and the name of a variable cannot be changed. A variable should be named after an identifier.

2.5 Statement

Statements are fragments of the program that are executed in sequence. In general cases, the statements will be executed line by line, from up to down. However, there are some statements allows the program execute the statements in different order, which bring us more convenience.

2.5.1 Branching Statements

Branching statements allows the program do different things under different condition. There are two branching statements in C/C++. They are `if-else` and `switch`.

2.5.2 Iteration Statements

Iteration, or loop, means to do a procedure repeatedly. To start a iteration, you can use iteration statements. These statements provide different ways to control how many times the procedure will be executed. There are three iteration statements in C/C++. They are `while`, `for`, and `do`.

2.5.3 Jump Statements

Jump statements cause a jump to another statement elsewhere in the code. Generally, they are used to interrupt `switch` statement and iteration staements. The jump statements are `break`, `continue` and `goto`.

2.6 Operand and Operator

The operand is the operated object, while the operator is the symbol specifying the type of operation.

For example, in the operation 8+9, both 8 and 9 are operand, and + is the operator.

There are four kinds of operators in C/C++. They are arithmetic operators, relational operators, logical operators, and bitwise operators.

Arithmetic operators are used to perform mathematical computation, such as addition, substraction, and so on.

Relational operators are used for comparison the values of two operands, such as equality, inequality and so on.

Logical operators are used to manipulate some conditions, such as ’and’, ’or’, and ’not’.

Bitwise operators are used to perform operations on binary numbers, which will not be discussed in this book.

Chapter 3

Structure of a C/C++ Program

We will use the following program as an example.

```c
#include<stdio.h>
int main(){
    printf("Handsome Liu is so handsome!\n");
    /*This
      is
      a
      comment*/
    //This is also a comment
}
```

3.1 Including the Header Files

```c
#include<stdio.h>
```

As mentioned before, some functions are defined in other header files. `stdio.h` is a header file provided by the compiler. `stdio` stands for STanDard Input and Output. Since `printf` is defined in `stdio.h`, we need to specify that we want to use `stdio.h`. The syntax for using a header file is

```
#include< <header_file> >
```

3.2 The `main` Function

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

The `main` function is the starting point of every program. A program always start to execute from the `main` function.

3.3 Comments

Comments are written for human beings, not compilers. Compilers will ignore all contents inside comments.

A comment can be construct by two ways, `//` or a pair of `/*` and `*/`.

All contents after `//` will be viewed as comments until the end of the line.
All contents after `/*` will be viewed as comments until `*/`.

Chapter 4

Basic Syntaxes

In this chapter, we will learn about how to implement the concepts in chapter 2.

4.1 Value and Literal

We have learned that there are many different type of value, such as integers, characters, and floating point numbers. A literal is a representation of the value. For example, an integer 3 and a character 3 should have different representations.

4.1.1 Integer Literal

To represent a decimal integer, just type it down like in mathematics. For example, `123` represents the integer one hundred and twenty-three.

4.1.2 Character Literal

To represent a character, enclose the character with a pair of single quotation marks. For example, `'A'` represents the character ”capital A”, while `'3'` represents the character ”3”.

4.1.3 Floating Point Number Literal

TBD.

4.1.4 String Literal

We haven't learn string data type yet, but we can view string as a sequence of characters. To represent a string, enclose the characters sequence with a pair of double quotation marks. For example, `"Handsome Liu is so handsome!"` represents the string ”Handsome Liu is so handsome!”.

4.2 Variables

As mentioned before, a variable can be used as the value it stores, but you should declare it before using it. In addition, you can modify the value it stores by assigning value to it.

4.2.1 Declaration

To declare a variable, you need to specify its data type and name. The syntax for declaring a variable is

```
<data_type> <name>;
```

For example, if you want to declare a variable of type integer named `number`, the syntax is `int number;`. Note that if a variable has been declared, you cannot re-declare it.

4.2.2 Assignment

We use an equals sign (=) to assign a value to a variable. Thn syntax for assigning is

```
<variable_name> = <value>;
```

For example, if you want to assign a character ”capital C” to a variable named `c`, the syntax is `c='C';`.

4.2.3 Usage

A variable can be used as the value it stores. For example, suppose we have an integer variable named `num` storing `5`, then `num+3` is equivalent to `5+3`.

4.3 Operator

4.3.1 Arithmetic Operator

Syntax in C/C++	Symbol in Mathematics	Explanation
+	$+$	Addition
−	-	Substraction
*	$\times$	Multiplication
/	$\div$	Division
%	mod	Modular

While performing arthimetic operation on non-negative integer values, the result of addition, substraction and multiplicaton is just like what you've learned in your math class. For division(, and modular, *resp.*), the results are the quotient(, and remainder, *resp*) The results of all the five operations are integers.

For example, $11 \div 3 = 3...2$, so `11/3==3`, and `11%3==2`

While performing arthimetic operation on character values, the character values will be converted into integer values according to the ASCII table firstly, then complete the integer arthimetic operation. ASCII table will be introduced in the later parts.

4.3.2 Relational Operator

Syntax in C/C++	Symbol in Mathematics	Explanation
<	$<$	Less than
==	$=$	Equal to
>	$>$	Greater than
>=	$\geq$	Greater than or eual to
!=	$\neq$	Not equal to
<=	$\leq$	Less than or equal to

The results of these relational operators will be either `true` or `false`.

4.3.3 Logical Operator

Syntax in C/C++	Symbol in Mathematics	Explanation
`!`	$\neg$	Not
`&&`	$\wedge$	And
`||`	$\vee$	Or

The operands of logical operators should be either `true` or `false`.
For example, `(1<2) || (5==6)` is a reasonable logical operation, while `'A' && 123` is not. ”One less than two, or five equals to six” does make sense (even if it’s false actually), but ”A and one hundred and twenty-three” is nonsense.

Logical operators are often used to combined the results of relational operators.

4.4 Formatting Inputting and Outputting

`scanf` and `printf` are functions provided by compiler and defined in `stdio.h`.

4.4.1 The `printf()` function

`printf()`, stands for printing formatted data. To output a string, say, ”Handsome Liu is so handsome!”, just simply put it inside the parentheses. Recall that a string is represented as a sequence of characters enclosed by a pair of double quotation marks.

```c
printf("Handsome Liu is so handsome!");
//outputting "Handsome Liu is so handsome!"
```

This is straightforward, but what if we need to output the value of a variable, say, an integer `num`? As expected, putting `num` inside the string does not work.

```c
int num=5;
printf("num");//outputting "num" instead of "5".
```

As a result, we need format specifier to tell the compiler you want to output a variable.

4.4.2 Format Specifier

Format specifier	Description
`%d`	output an `int` in decimal
`%o`	output an `int` in octal
`%x`	output an `int` in hexadecimal
`%c`	output a `char`
`%f`	output a `double`

4.4.3 `printf()` with Format Specifier

Now, equipped with format specifier, we are able to output a variable. Let's look at the following code.

```
    int a=5;
    int b=3;
%    printf("%d+%d=%d",a,b,a+b); //outputting "5+3=8"
```

As you may discover, the format specifiers are replaced by the value you put after the string. Thus, the first `%d` is replaced by the value of `a`, which is ”5” in decimal. The second `%d` is replaced by the value of `b`, and the third `%d` is replaced by the value of `a+b`.

4.4.4 Control Characters

Now, with `printf()`, we can output almost anything we want. However, there is still a problem. How to start a new line?

For example, you want to output two lines contains `Handsome` and `Liu` respectively.

```
    printf("Handsome");
    printf("Liu");
```

However, the result of the program above will be like this.

```
HandsomeLiu
```

And a program like this

```
    printf("Handsome
    Liu");
```

will be counted as syntax error.

It seems that there's no way to achieve this, unless we define some special ”marks” for starting a new line. These marks are called ”Control Characters”. For example, '\n' is a mark for starting a new line, and will be viewed as a character. Thus, to achieve the goal, we can use `printf` like this

```
    printf("Handsome\nLiu");
```

The output of this program will be

```
Handsome
Liu
```

4.4.5 `scanf()` with Format Specifier

Sometimes it is necessary to control the values of some variables while executing the program. One way to achieve this is using `scanf()` to assign values to integers via inputting while executing.

The usage of `scanf()` is like this.

```
scanf("<Format Specifier 1><Format Specifier 2>......",
            &variable1, &variable2, ......)
```

Please note that every variable name has to be preceded by a `&` for some reason.

Let's look at the following code.

```
#include<stdio.h>
int main(){
    printf("Hello!\n");
    int a, b;
    scanf("%d%d",&a, &b);
    printf("%d %d\n",a, b);
}
```

After executing this program, the result will be like this.

```
Hello!

```

Recall that the program will be executed line by line. After line 3 is executed, a `scanf` is encountered. Thus, the program will suspend and wait for your input.

To input an integer, simply type an integer and press <enter>. For example, after input, say, 7 and 25, the program will resume, and the result will be like this.

```
Hello!
7
25
7 25
```

Note that the `7` and `25` in the second and third line is the integers you just typed, and the `7 25` in the fourth line is the output of line 6.

4.5 Branching Statements

4.5.1 `if-else` statement

`if`

The syntax of `if` is quite simple.

```
if(condition){
    <statement1>;
    <statement2>;
    ...
}
```

It's concept is also simple. If the condition is true, then go inside the curly brackets and continue executing from `<statement1>`. Otherwise,skip the curly brackets and continue executing from the first line below the curly brackets.

`else`

An `else` must follow an `if`. Combining with `else`, the syntax of `if-else` becomes

```
if(condition){
    <statement1>;
    <statement2>;
    ...
}
else{
    <statement3>;
    <statement4>;
    ...
}
```

If the condition is true, then go inside the first curly brackets, continue execute from `<statement1>`, and skip the second curly bracket. Otherwise, skip the first curly brackets, go inside the second curly brackets, and continue executing from `<statement3>`.

Note that you should not put semicolons at the end of `if-else` statements.

For example, the following code is a program for checking if an integer `num` is even or odd.

```
    if(num%2==0){
        printf("%d is even.\n",num);
    }
    else{
        printf("%d is odd.\n",num);
    }
```

4.6 Iteration Statements

4.6.1 while

The syntax of `while` is as follows.

```
while(condition){
    <statement1>;
    <statement2>;
}
```

A `while` statement can be viewed as an `if` statement executed repeatedly until the condition become false. To be more specific, the procedure will be like this.
1. Check if the condition is true. If not, goto 3.
2. execute the statements inside the curly brackets, and go back to 1.
3. Continue executing the statements below the curly brackets.

4.6.2 for

The syntax of `for` is a little bit more complicated than `while`.

```
for(initialization; condition; updating){
    <statement1>;
    <statement2>;
}
```

While using `while` statement, there are three things you need to do almost every time. To initialize some variables, to set the condition, and to update some variables.

For example, if you want to output ”Handsome Liu is so handsome !\n” for 10 times using `while` statement, you may write a program like this.

```
    int counter=0;//initialization
    while(counter<10){//setting condition
        printf("Handsome Liu is so handsome !\n");
        counter=counter+1;//updating
    }
```

4.7 Array

An array is a sequence of elements. The elements must be of same data type. The data type and the length of the array cannot be changed after declared.

4.7.1 Declaration

To declare an array, you need to specify its name, its length, and the data type of its elements. The syntax for declaring an array is

```
<data_type> <name>[<length]
```

For example, if you want to declare an charater array of length 100 named `c`, the syntax is `char c[100];`

4.7.2 Indexing

In mathematics, the elements of a sequence a of length n is written as $a_1, a_2, ..., a_n$.

In C/C++ and most of the programming languages, the elements of an array `a` of length `n` is represented as `a[0], a[1], ..., a[n-1].`

4.7.3 Assignment