

PROGRAMMING METHODOLOGY

Lab 1: Fundamental of C Programming Language

1 Introduction

In this lab section, we will concentrate on the fundamental of C programming language, and use it as a tool to implement your algorithms. In the first tutorial, you will be presented a development environment, called Cygwin, to compile your source code to executable program. Besides that, we introduce some basics about C: data types, variables and constants, arithmetic, control flow.

2 Development Environment

In this section, we present a development environment for working with your C program. A development environment is a collection of procedures and tools for developing, testing and debugging an application or program. In this course, we mainly focus on Cygwin¹, a framework contains the number of tools which provide functionality similar to Linux distribution on Windows OS. To install Cygwin on your computer, first, you need download the setup file, which is available on the Cygwin homepage. Note that, you have to choose the right version of your system, either 32-bit or 64-bit.

2.1 Install Cygwin and GCC

After completely download the appropriate setup file, let's begin the installation process. **Figure 1** to **Figure 10** show the step-by-step guide to successfully install Cygwin and GCC environment on your system. Note that, the most important step is in **Figure 7**, you must select the following components to work with C/C++ environment on Cygwin, by clicking "*Skip*" marker:

- `gcc-core`: C compiler sub-package.
- `gcc-g++`: C++ sub-package.

¹ <https://www.cygwin.com/>

- `libgcc1`: C runtime library.
- `gdb`: The GNU Debugger.
- `make`: The GNU version of the 'make' utility.
- `libmpfr4`: A library for multiple-precision floating-point arithmetic with exact rounding

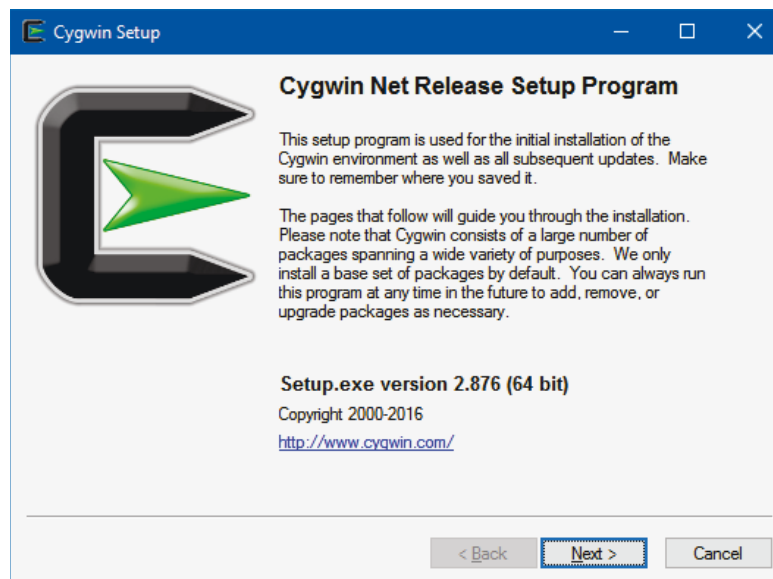


Figure 1 Welcome screen of Cygwin installation

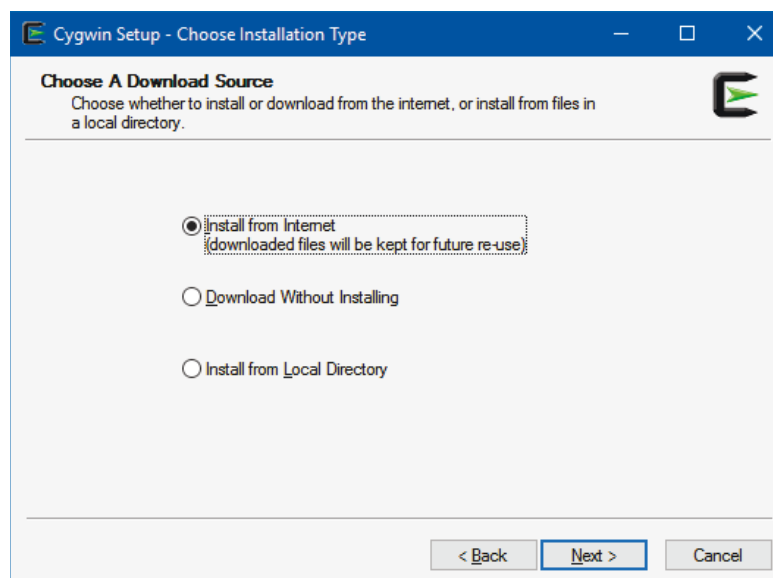


Figure 2 Choose the way to download the installation source

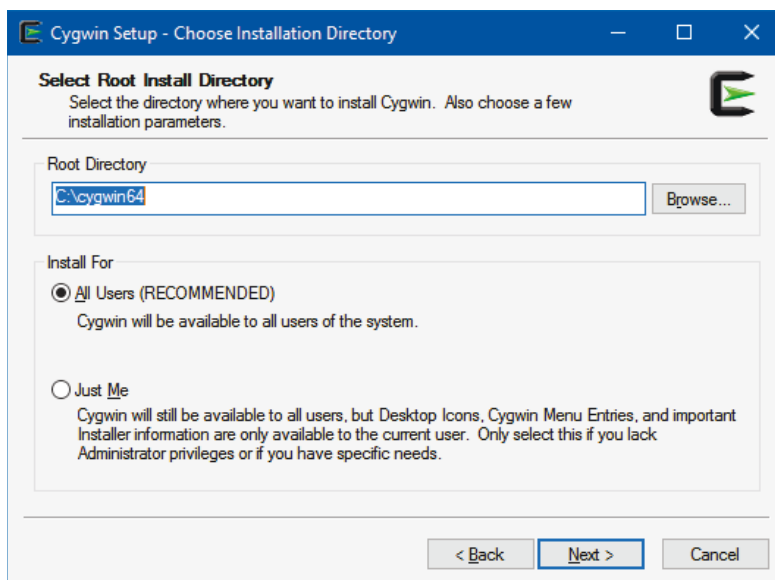


Figure 3 Select root install directory

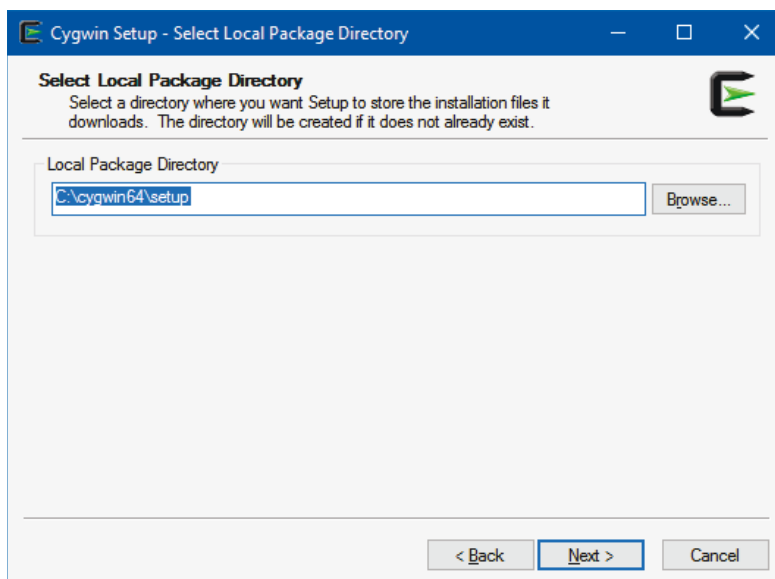


Figure 4 Select the directory of local package

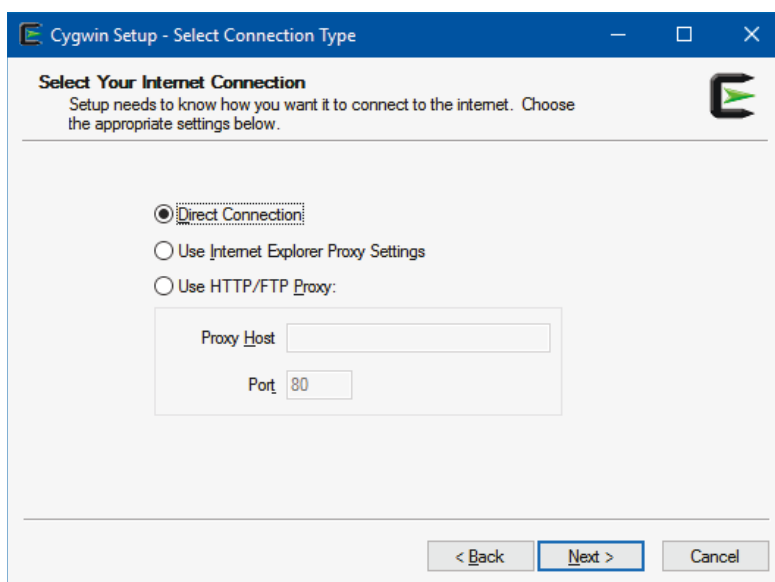


Figure 5 Select your internet connection

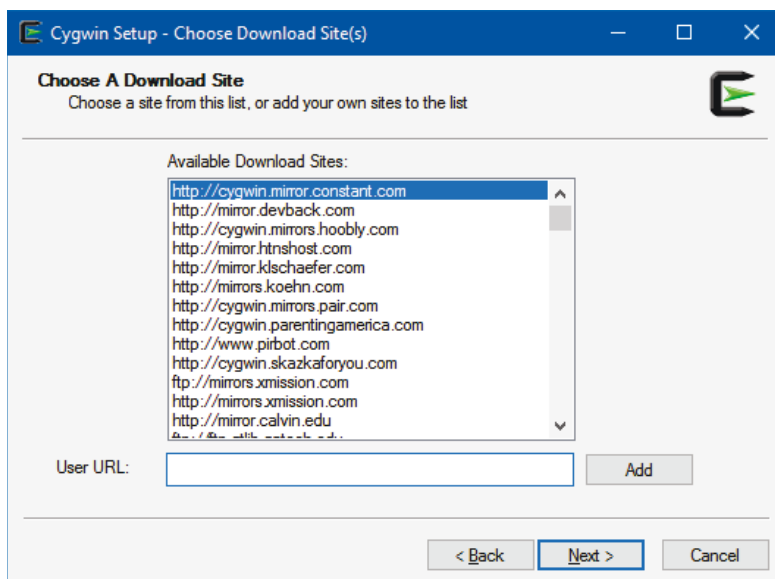


Figure 6 Select a download site

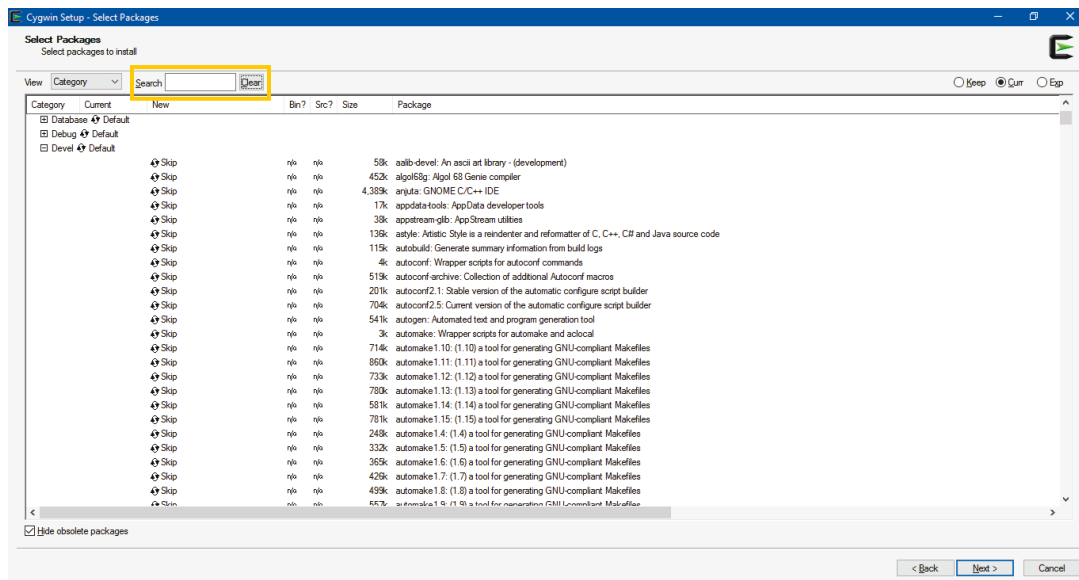


Figure 7 Select package (*)

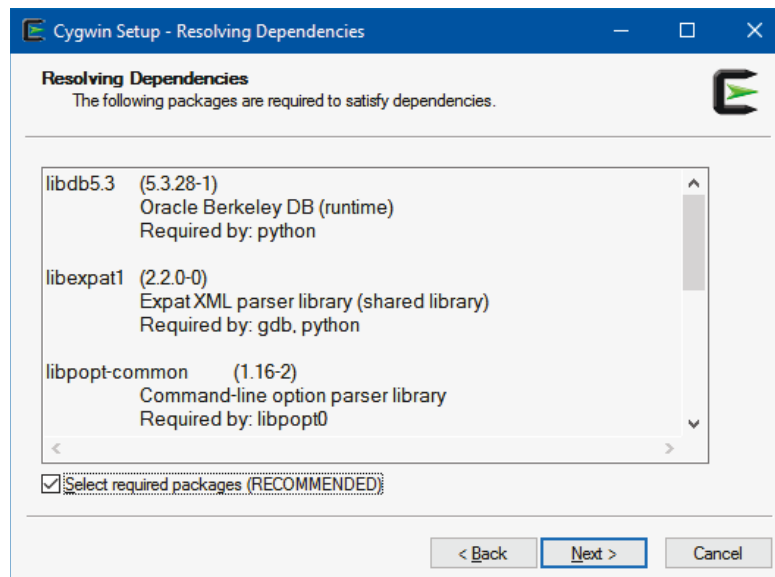


Figure 8 Resolving dependencies for setup process

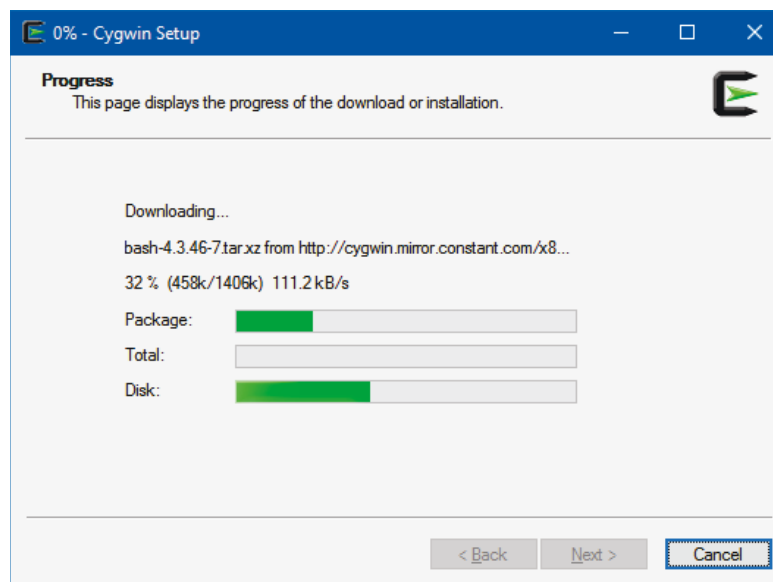


Figure 9 Installing Cygwin and components

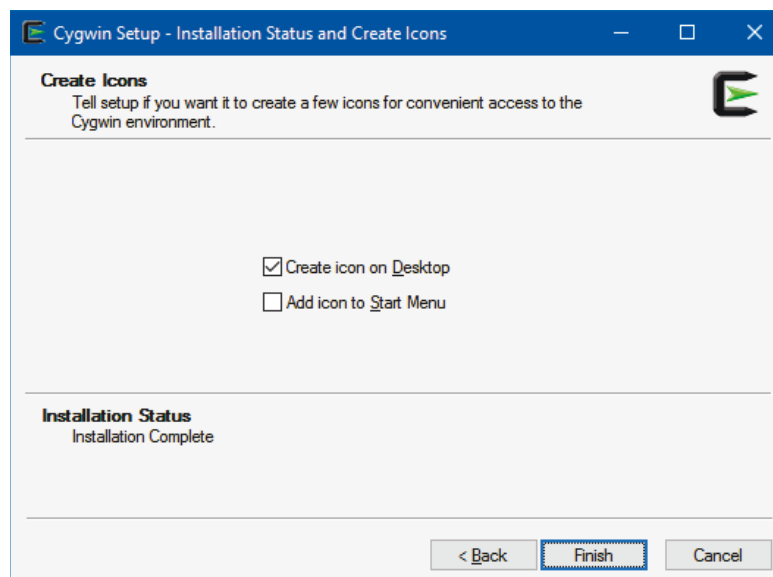


Figure 10 Installation complete screen

2.2 Add Cygwin to System Path

Supposing you have successfully all requirement components to work with C/C++ on Cygwin environment, but you can only call Cygwin commands in Cygwin Terminal. The question is, can we call Cygwin commands in within either Windows CMD² command-line or

² <https://en.wikipedia.org/wiki/Cmd.exe>

Windows PowerShell³. Absolutely, the answer is "Yes", and we need few more steps to do this. First, you need to navigate to System Properties windows (right-click on "Computer" icon on desktop → select "Properties" → select "Advanced system settings" → select "Environment Variables..." → select "Path" and then "Edit..." → select "New" → type the path of Cygwin "bin" directory⁴). **Figure 11** and **Figure 12** visualize the steps to add Cygwin to system path. It will be different from interface if you use other Windows editions, in this tutorial, we use Windows 10 Enterprise edition.

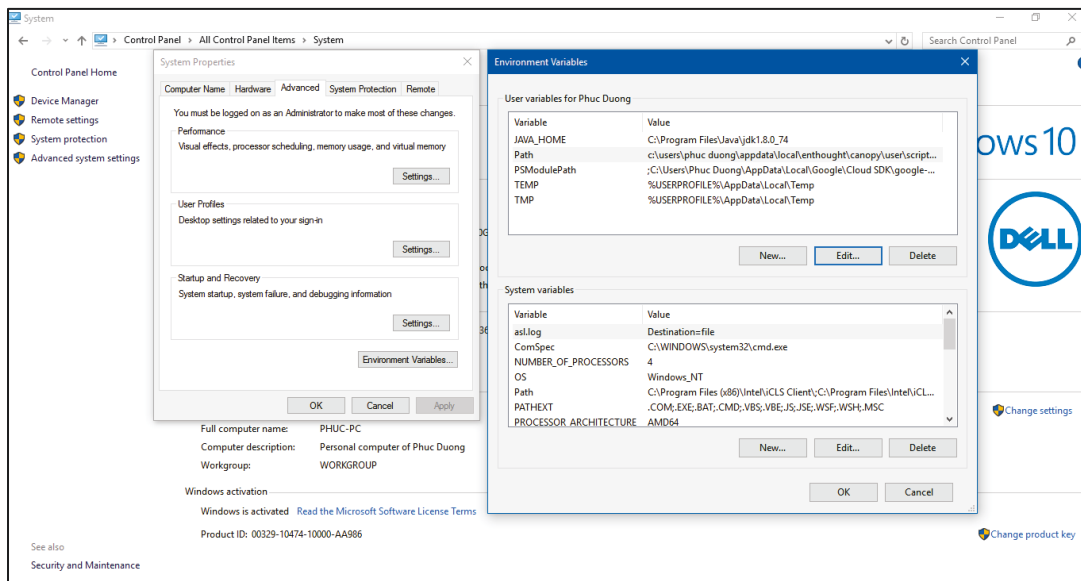


Figure 11 Environment Variables

³ <https://msdn.microsoft.com/en-us/powershell/mt173057.aspx>

⁴ Suppose the "bin" directory is in C:\cygwin64\bin

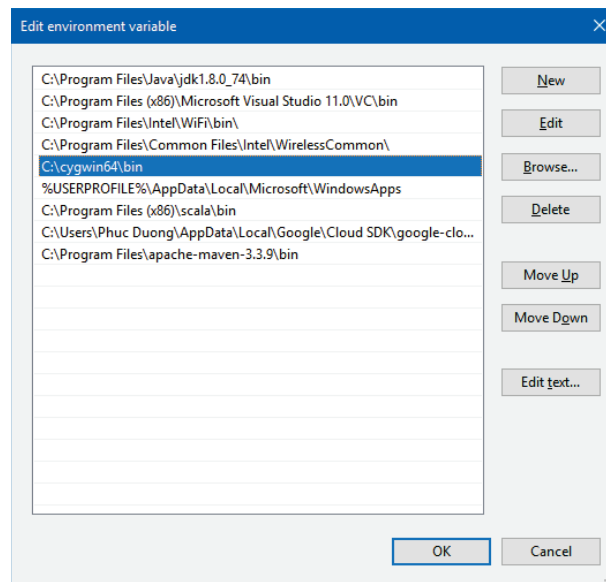


Figure 12 Add Cygwin directory to system path

3 C Programming Language

Before we go deeper into C programming language, let's execute your first C program. **Figure 13** shows an example, which will print a sequence of characters on the screen.

```
1 // hello.c
2 // A first program in C.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main()
7 {
8     printf("Hello World!\n");
9 }
```

Figure 13 A first program in C

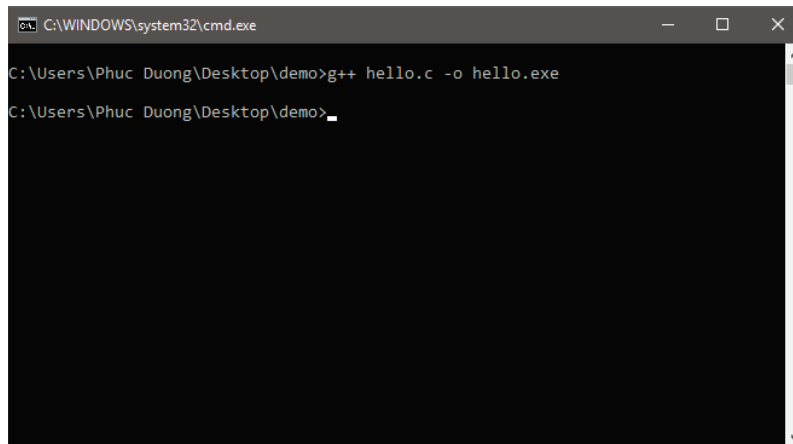
Although this program is simple, it contains several important features of the C/C++ programming language. Lines **1**, **2**, and **5** begin with `//` symbol, indicates that these lines are **comments**. You should insert comments to your program for program readability, and comments are ignored by the compiler. Another way to make comments to your program is using `/* ... */`, a multi-line comments. Line **3** is a pre-processor directive, in this case, we tell the preprocessor to include the contents of the **standard input/output header** in the current program. Line **6**, the main function, is a part of every C/C++ program. The parentheses after main indicate that main is a program building block called a function. C/C++ programs contain one or more functions, one of which

must be `main`. Every program in C/C++ begins executing at the function `main`. The keyword **int** before the **main** function indicates that this function returns an integer value. In line 8, we call a statement, **printf**, to instruct the computer performs an action, in this case, display the string of characters marked by the quotation marks on the screen. An important note is that every statement must end with a semicolon (;), as knowledge as the statement terminator. The last two characters in the argument of **printf** statement is called **escape character**, these characters will not be printed on the screen, **Table 1** shows some common escape characters.

Table 1 Common escape characters

| <i>Escape sequence</i> | <i>Description</i> |
|------------------------|--|
| <code>\n</code> | Newline. Position the cursor at the beginning of the next line. |
| <code>\t</code> | Horizontal tab. Move the cursor to the next tab stop. |
| <code>\a</code> | Alert. Produces a sound or visible alert without changing the current cursor position. |
| <code>\\</code> | Backslash. Insert a backslash character in a string. |
| <code>\"</code> | Double quote. Insert a double-quote character in a string. |

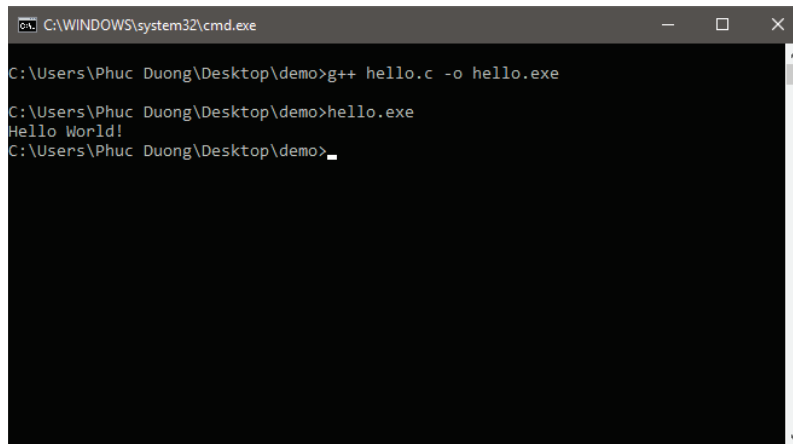
You had an overview of a program in C. Now, we focus on how to compile your document program to an executable program. In order to do this, let's start the Cygwin in the current window which contains your document program by using the following key combination and mouse, **Ctrl** + **Shift** + **Right-click**, and select "Open command window here". Then, when the command window shows up (**Figure 14**), type the command as follow **g++ hello.c -o hello.exe**, and the GCC will compile your document program to an executable program (whose file extension is **.exe**), and place it in the same window of your document program.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Phuc Duong\Desktop\demo>g++ hello.c -o hello.exe
C:\Users\Phuc Duong\Desktop\demo>
```

Figure 14 Compile your program

After compile from source code to an executable program, to run it, we easily call this program on the CMD window, as in **Figure 15**.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Phuc Duong\Desktop\demo>g++ hello.c -o hello.exe
C:\Users\Phuc Duong\Desktop\demo>hello.exe
Hello World!
C:\Users\Phuc Duong\Desktop\demo>
```

Figure 15 Run your program

The rest of this section is organized as follows: (1) datatypes, (2) variables and constant, (3) arithmetic operators, and (4) control flow.

3.1 Datatypes

Table 2 shows several fundamental datatypes⁵ of C, along with the sizes and the ranges of value instances of these types may have. We must note that bool type is not available in C environment, thus, in case you want to use it, you need include the **stdbool.h** header.

⁵ <https://msdn.microsoft.com/en-us/library/s3f49ktz.aspx>

Table 2 C datatypes

| Value type | Byte | Range |
|---------------|------|---|
| int | 4 | 32-bit signed two's complement integer (-2^{31} to $2^{31} - 1$, inclusive) |
| char | 1 | 8-bit signed two's complement integer (-2^7 to $2^7 - 1$, inclusive) |
| float | 4 | 32-bit IEEE 754 single-precision float, 3.4E +/- 38 (7 digits) |
| double | 8 | 64-bit IEEE 754 single-precision float, 1.7E +/- 308 (15 digits) |
| bool | 1 | true or false |

3.2 Variables and Constants

In C, and like the most programming languages, is able to use named variables and their contents. Variables are simply names used to refer to some location in memory – a location that holds a value with which we are working. To declare a variable in C, it has to begin with the data type, then the variable name⁶, the optional part is the assigned content for the variable. Variable names in C are made up of letters (upper and lower case) and digits. The underscore character (" _ ") is also permitted. Names must not begin with a digit and do not use any special prefix characters. Upper and lower case letters are distinct, so *x* and *X* are two different names. Traditional C practice is to use lower case for variable names, and all upper case for symbolic constants.

```

1 // dem01.c
2 // Using variable
3 #include <stdio.h>
4
5 int main()
6 {
7     int age = 18;
8     printf("Happy %d Birthday!\n", age);
9 }
```

Figure 16 Sample program uses variable

The program in **Figure 16** declares a variable, which is **age**, has **int** type and assigns its value is 18. Then, use **printf** statement to print the content of **age** on the screen, between the pre-defined sequence of characters. The **%d** character indicates the compiler to replace its current position to the value of **age**. The following program in **Figure 17** will calculate the

⁶ This name need to be unique in the scope of program code

circumference of a circle by prompting user to input a radius, and then return the circumference of corresponding circle.

```
1 // circle.c
2 #include <stdio.h>
3
4 int main()
5 {
6     const float PI = 3.14;
7     int radius;
8     float circumference;
9
10    printf("Enter the radius: "); // prompt
11    scanf("%d", &radius); // read an integer
12
13    circumference = 2 * PI * radius;
14
15    printf("Circumference: %f\n", circumference);
16 }
```

Figure 17 Sample program uses const and variables

The program in **Figure 17** declares a constant variable **PI** (line 6) and two variables (line 7, 8), which are **radius** and **circumference**. To declare a constant variable, we have to begin with **const** keyword; from that point, it becomes a read-only variable, that means, you can't edit its value. Next, prompt the user to input the radius of circle at line 10, and then get and assign the input's value to **radius** variable (line 11). You should note that, since the radius is an integer number, we use **%d**. At line 13, we calculate the circumference of the current circle on the right of "=" symbol, then assign its value to the left variable, which is **circumference**. Finally, print the result on the screen, but **circumference** is float number, we use **%f** to indicate the computer to print a floating-point number at corresponding position.

3.3 Arithmetic operators

To calculate the result of an arithmetic expression, C provides some basic arithmetic operators. The arithmetic operators are *binary* operators, that means, the operator + has to have two operands, for example, "3 + 7".

Table 3 Arithmetic operators

| Arithmetic operator | Operation | Algebraic expression | C expression |
|---------------------|----------------|--------------------------------------|--------------|
| + | Addition | $x + 7$ | x + 7 |
| - | Subtraction | $y - z$ | y - z |
| * | Multiplication | $a \times b$ | a * b |
| / | Division | x/y or $\frac{x}{y}$ or $x \div y$ | x / y |
| % | Remainder | $r \bmod s$ | r % s |

3.4 Control flow

The control-flow of a language specify the order in which computations are performed. In the above sample programs, we have many declarations and statements, which ended by a semicolon. To group those into a block, we use braces "{" and "}", thus they are syntactically equivalent to a single statement. And there is no semicolon after the right brace that ends a block. The if-else statement is used to express decisions, whose syntax is:

```
if(true-false-expression) {
    // Statement1
}
else {
    // Statement2
}
```

The **true-false-expression** is evaluated; if it is **true**, **Statement1** is executed, if it is **false**, **Statement2** is executed instead. Conditions in if statements are formed by using the equality operators and relational operators summarized in **Table 4**. The relational operators all have the same level of precedence and they associate left to right. The equality operators have a lower level of precedence than the relational operators and they also associate left to right.

Table 4 Equality and relational operators

| <i>Algebraic operator</i> | <i>C operator</i> | <i>Example</i> | <i>Meaning</i> |
|-----------------------------|-------------------|------------------|---------------------------------|
| <i>Equality operators</i> | | | |
| = | == | x == y | x is equal to y |
| ≠ | != | x != y | x is not equal to y |
| <i>Relational operators</i> | | | |
| > | > | x > y | x is greater than y |
| < | < | x < y | x is less than y |
| ≥ | >= | x >= y | x is greater than or equal to y |
| ≤ | <= | x <= y | x is less than or equal to y |

3.5 Else-If Statement

In Section 3.4, we introduce the control statement "if – else", which evaluates only one condition at a time, in order to sequentially evaluate the conditions, we use "else – if" statement, as follows:

```
if(expression1) {
    // Statement1
}
else if(expression2) {
    // Statement2
}
else if(expression3) {
    // Statement3
}
else {
    // Statement4
}
```

By using this structure, the expressions are evaluated in order, if an expression is true, the statement associated with it is executed, and this terminates the whole chain. The last **else** part handles the "none of the above" or default case, that means, if none of the other conditions is satisfied, the default statement is executed. For example, see the program in **Figure 18**.

```
1 // conditional-statements.c
2 #include <stdio.h>
3
4 int main()
5 {
6     float score = 8.4;
7
8     if(score < 5) {
9         printf("You need to improve your score!");
10    }
11    else if(score < 8) {
12        printf("That's OK!");
13    }
14    else if(score < 9) {
15        printf("Good!");
16    }
17    else {
18        printf("Excellent!");
19    }
20 }
```

Figure 18 Sample program using else-if

4 Exercises

1. Write a C program to print your name, date of birth, and mobile number.
2. Write a C program prompting user to input two integer numbers, then compute and print the results of addition, subtraction, multiplication, division, and remainder.
3. Write a C program to compute the perimeter and area of a rectangle with a height provided by user.
4. Write a C program to convert specified days into years, weeks and days. (Note: ignore leap year).
5. Write a C program to convert the temperature from Celsius to Fahrenheit. (Hint: $1^{\circ}\text{C} = 33.8^{\circ}\text{F}$).
6. Write a C program to return an absolute value of a number.
7. Write a C program to check whether a year is a leap year or not.
8. Write a C program to find maximum between two numbers.
9. Write a C program to find maximum between three numbers.
10. Write a C program to check whether a number is even or odd.
11. Write a C program to input a character and check whether it is alphanumeric or not.

12. Write a C program to input angles of a triangle and check whether triangle is valid or not.
13. Write a C program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to following:

Percentage > 90%: Grade A

Percentage > 80%: Grade B

Percentage > 70%: Grade C

Percentage > 60%: Grade D

Percentage > 40%: Grade E

Percentage < 40%: Grade F

5 Reference

- [1] Brian W. Kernighan & Dennis Ritchie (1988). *C Programming Language, 2nd Edition*. Prentice Hall.
- [2] Paul Deitel & Harvey Deitel (2008). *C: How to Program, 7th Edition*. Prentice Hall.
- [3] *C Programming Tutorial* (2014). Tutorials Point.
- [4] *C Programming* (2013). Wikibooks.
- [5] [https://en.wikipedia.org/wiki/C_\(programming_language\)](https://en.wikipedia.org/wiki/C_(programming_language))