## **1. Basic programming review-"Hello ESYS" again-**

The purpose of this training is to familiarize you with the C language.

I think that the minimum C language programming has become possible in the classes so far.

By the way, when studying a foreign language, what you need is practice, not talent.

C language is also a "language". If you practice over time, you will definitely be able to write.

When studying a foreign language, it is necessary to have a teacher who can speak a proper foreign language and write sentences.

C language is also a "language".

As long as it is a language, it has not only grammar but also manners and customs.

Even in Japanese, if you just talk with your friends, you will naturally learn them if you interact with your friends without any effort.

However, some training is required to speak proper Japanese that is not misleading to anyone.

The level of Japanese spoken by the NHK announcer and the level of Japanese when you are talking with your friends are different even if they are the same Japanese.

The same is true for C.

Since you are in a position to become a programmer (in Japanese, a position equivalent to an announcer or a newspaper reporter), let's aim not only to just write, but to write a correct and compact program.

### **01.01. Annoying Hello ESYS**

First, let's review C language programming.

After reviewing the lessons you have taken so far, it is warming up.

The following sample is a "bad example". If you have written this way, please change your mind from now on.

(It's not wrong, but it's a misleading way of writing.)

01-01-HelloESYS-NotSoGood.c (code/01-01/01-01-HelloESYS-NotSoGood.c)

What's wrong?

Before looking at the next program, I hope you can point out four things.

Please download and compile as it is. Do you remember how to compile it from the command line in the terminal?

Please modify the program yourself. Please recompile it.

**Exercise**

**01-01-ex1**: What does "terminal" mean in the 5th floor computer system Ubuntu linux (this class) ?

**01-01-ex2**: What do you call this terminal-like interface with three letters of the alphabet? What is its official name?

**01-01-ex3**: Why is this text-only window called "terminal"?

**01-01-ex4**: A similar expression is the term console. Look up the etymology of the word and think about why it is named as a console.

### **01.02 Elegant Hello ESYS**

What was wrong with writing the destination program?

Let's take a look at the following improved program.

01-02-HelloESYS-Better.c (01-02/01-02-HelloESYS-Better.c)

the difference from 01-01 (01-02/01-01\_01-02.html)

Did you find out what was wrong with the program?

(1) There is no "#include <stdio.h>"

(2) The int type is not specified at the main() function

(3) The argument of the main() function is not described

(4) main() does not return a value (int type) at the end of the function

**Compile method**

First, about the compilation method on the command line, from now on, do like:

|  |
| --- |
| $ gcc -Wall -o target\_name Source\_C\_file  $ gcc -Wall -o 01-2-HelloESYS-Better 01-2-HelloESYS-Better.c |

You should have learned -o option. This is the option to have the executable file name created by something other than "a.out".

How about -Wall?

Unix commands are basically described in man pages.

In a terminal window, type man gcc.

|  |
| --- |
| $ gcc man |

(Especially for commands close to the OS, there are plenty of explanations.)

(The difference between the man page and the www page somewhere on the internet is that the former is, in principle, "a document written by the creator or someone close to it." The other is just another person reading and rewriting it. For example, man has a section called AUTHOR near the end.)

When you read the man's description, -Wall says "Warning." You can see it as an abbreviation for “Warning all."

In other words, it points out the lines that are grammatically acceptable but strange.

(It's like a Japanese language teacher who points out strange Japanese expressions.)

You should write a wonderful (grammatically impeccable) program that doesn't give any warnings even with -Wall. You should learn the habit of writing so.

(In the long run, this is the key to making the program source last longer. I can assure you from my experience.)

**Rethinking the function using "printf()" as an example**

This program calls printf(), which is provided as a part of the standard library by the C language.

When using a function that you have not created, such as printf(), you must always follow the method specified by the person who created it.

Functions in programming libraries such as printf() are explained in Chapter 3 of man.

Let's see how printf() is shown on a man page.

(You will revisit what the library is in Section 4.2)

When you read the man page, you are instructed to write "#include <stdio.h>" before you start using printf().

Therefore, you have to put "#include <stdio.h>" at the beginning of the file.

(If you don't use printf(), you don't need to write "#include <stdio.h>")

**Exercise**

**01-02-ex1**: You will need stdio.h when using the printf() function.

- Indicate the path name (location) of stdio.h that you will actually use.

- Show the evidence that the path is correct.

- Explain what stdio.h specifies in it, using printf() as an example.

- Is stdio.h expanded at compile time or link time?

**01-02-ex2:** Suppose that a float variable k contains the number -15.2. We want to use the printf() function to show it as “-15.20%”.

- Take the variable f as an argument and show how to write the format of printf().

- Show the relevant part of the printf() man page where the format appears.

(Extract the relevant part(s) and add a simple comment.)

**01-02-ex3**: Assuming that the float variable k contains the value -15.2. We want to use the printf() function to show it like -15.20% in right-justified 10 characters space(four spaces before the minus sign). Also, suppose you want to use the printf() function to show it in -2.34% in the right-justified 10 characters space when k contains -2.336 (five spaces before the minus sign).

- Take the variable f as an argument and show the format of printf() that can make these.

- Show the relevant part of the printf() man page where the format explanation appears.

(Extract the relevant part(s) and add a simple comment.)

### **01.03 Very polite Hello ESYS**

It will be a little further development level, but if you describe it more carefully, it is as follows: It is better to write in.

01-03-HelloESYS-Full.c (code/01-02/01-02-HelloESYS-Better.c)

Difference from01-02 (code/01-02/01-01\_01-02.html)

In a functional language such as C language, in principle, all functions have a return value.

In many cases, the return value has important implications.

In particular, it often returns a value that gives an indication of whether the function was able to perform the work as planned.

The only exception is void type functions. In this case, the return value is untyped (= no return value).

If you decide to define your own function in the future, use void type functions only when there is no way to fail or when you can't help returning a value.

printf() is a function with a return value. The return value of any function that returns value has meaning. Do not ignore it.

(Refer to the return value on the man page of printf().)

(If you know what value is returned and you can ignore it. Sometimes it is OK if you are an experienced programmer.)

By the way, when you are not so sure of the calculation order of the operators, use parentheses effectively.

**Exercise**

**01-03-ex1**: About the return value of printf().

- Explain the contents of the return value.

- Explain what kind of program you should write to know that printf() did not work properly.

- Try to explain for yourself why there are few examples of C language programs in the world that use the return value of printf() (even if you ask Google teacher, they rarely appear).

**01-03-ex2**: Some C language operators have two meanings even though they are represented by the same symbol.

- What is the symbol?

- Explain both meanings.

**01-03-ex3**: Some C language operators repeat the same symbol twice.

- Show as much as you know such operators.

- Explain the meaning of each operator when not repeating and when repeating.

### **01.04 Convenient macros to use**

I heard that you have already learned about the C language preprocessor.

Of course, you can define and use it, but there are also useful macros that are originally defined in the C compiler specifications.

01-04-HelloESYS-Macros.c (code/01-04/01-04-HelloESYS-Macros.c)

**Exercise**

**01-04-ex1**: In C language, various mathematically useful numerical values ​​are defined as macros.

- Where can we find it?

- Clarify the definition of five or more mathematical macros by showing the source by file name and lines.

**01-04-ex2**: In this exercise, we will compile a C language program using gcc.

- Indicate the version of gcc to use. Also, explain how to know the version.

(Hint: gcc has a function that tells you the version.)

- Show the C language specifications that the gcc version understands.

(There are various differences in C language, just as Japanese has standard language, Kansai dialect, and Ibaraki dialect. Unlike humans, gcc is inflexible, so gcc can interpret the C language of some “dialect” only when it is told it is written in some specific dialect.)

(Before starting the compilation work, you should specify in gcc which dialect you want to understand. Usually, it is done implicitly.)

**01-04-ex3**: What about the latest version of C language now?

- Show a reference (URL is acceptable) that defines the latest C language specifications, and briefly describe its features. References should be "official". Find out who developed the language specification.

**01-04-ex4**: See the gcc man page to see how many "dialects" gcc understands on this man page.

- Choose two of them and show a brief explanation.