## **3. Structure the program by using functions**

In this section, you will learn the techniques necessary for creating large-scale programs.

As an example, consider a calculator program that reads and calculates numbers from standard inputs.

Since it is impossible to write a large program all at once, it is necessary to divide the program.

Since a program is a reflection of an algorithm, it is necessary to first think carefully about the algorithm and then consider where to divide the algorithm so that it is easy to understand.

When the algorithm becomes complicated, it becomes impossible to write it only with the main() function.

If the algorithm description is properly divided, the readability of the program will be greatly improved.

In the C language, algorithm division is realized by functions.

By dividing the algorithm using functions and making it into a subroutine, it becomes possible to improve the efficiency of program development. This improves readability, hides information, and makes it easier to reuse program components.

Subroutines by functions is an indispensable technique for humans to write algorithms in programming languages.

By making a subroutine, when writing a program, people using the subroutine do not have to care about its inside.

This is called information hiding.

(In programming languages, "information hiding" is used in a good way to be complimented.)

In this chapter, we will take the "simple calculator" program as an example.

### **03.01. Designing a simple calculator program**

In a simple calculator program, only one addition, subtraction, multiplication, and division can be executed at a time.

When thinking about an algorithm, it is easier to avoid mistakes if you divide the work from a broad perspective.

[Process 1] Receive a triplet of a number, an operator, and a number in one line from the standard input

[Process 2] Check if the input is valid

[Process 3] Calculate according to the operator

[Process 4] Display the result

[Process 5] Repeat the process from 1 to 4 until there is a signal to end

Note:

(1) Numerical values ​​are real numbers (negative values ​​are possible including floating points)

(2) Operators are addition, subtraction, multiplication and division (Four kinds)

(3) When a user inputs, he/she separates the numerical value and the operator with one or more spaces, and type a line feed at the end of the input

(4) When calculating, determine whether or not the operation is possible

Processing example:

|  |  |
| --- | --- |
| Input | calculation result |
| 3.4 + 1.2 | 5.6 |
| -2.2 - -2 | -0.2 |
| -2 \* -3 | 6 |
| -0.3 / 0.17 | -1.7647058823 |
| 3 / 0 | Error |

In the program, we use

fgets() function

sscanf() function

strcmp() function

These are from the C language library.

**Exercise**

**03-01-ex1**: Display the man pages of fgets(), sscanf(), strcmp() on the terminal.

**03-01-ex2**: Show the header file(s) required for each of fgets(), sscanf(), and strcmp().

**03-01-ex3**: N. A.

### **03.02. Trial run of fgets()**

The fgets() function is a function that reads one line of input from the file indicated by the file descriptor (third argument) and copies it to the memory space indicated by the first argument.

For the third argument, use the file descriptor, which is usually the return value of fopen().

However, since the standard input is used here, a special descriptor stdin that represents standard input is specified.

(Not limited to this, there are quite a few cases in Unix where you can operate things other than files by treating them just like files.)

(For example, data streams and physical devices are also operated by treating them just like files.)

Fgets() copies one line of data to the memory space specified by the first argument, up to the number of bytes specified by the second argument.

Moreover, write '\0' at the end of the copied byte string and use it as the end of the character string.

In other words, if you give the number of bytes-1 of the memory space prepared for the first argument to the second argument, you can avoid buffer overrun.

In Linux, "one line to read" includes the newline character ('\n') itself.

Therefore, the number of characters that can be read in one line at a time is practically the number of bytes in the prepared buffer -1 ('\n') -1 ('\0').

03-02-fgets.c (code/03-02/03-02-fgets.c)

Read this program first, then compile and run it.

In this program, the size is matched by finding the second argument of fgets() with the first argument given by sizeof().

sizeof() returns the total number of bytes when applied to a fixed-length array name.

This program prepares only 5 bytes for the buffer (one line).

In other words, if you exceed 5-1-1 = 3 characters, say, type 4 or more characters at a time, the buffer size limit will be reached, and the loop of the while statement will go again to get the character string after that.

Also, the last byte of the string is always displayed as a line break. Remember, the newline character is also the character entered by the user! (by typing “Return” key)

If you want to end with something other than Ctrl+c while the calculator accepts input, press Ctrl+d.

(Ctrl+c means to press down the Ctrl key and press the c key.)

Ctrl+d allows you to enter the special character EOF (end of file) on the terminal.

(You can change the key assignment with the command stty. Man stty and see the section on Special characters. Execute stty -a for a list.)

**Exercise**

**03-02-ex1**: The gets() function has a fatal flaw. Explain the flaw.

**03-02-ex2**: NA

**03-02-ex3**: NA

**03-02-ex4**: NA

**03-02-ex5**: NA

**03-02-ex6**: NA

**03-02-ex7**: For 03-02-fgets.c, explain the reason why you can terminate the program with Ctrl-d in relation to the specifications of the functions used in the program.

### **03.03. Test run of sscanf()**

I think you have already learned scanf() for the standard input.

sscanf() is its relative and does the same for the specified character array, not for the standard input.

Since both scanf() and sscanf() can know exactly how many successfully read by the return value, error handling can be performed by that.

(In fact, you should handle the error, because the user might do anything unpredictable.)

Be careful, you need space characters between the arguments.

There is also a potential risk of buffer overruns with sscanf().

When reading a character string, specify a number less than or equal to the number of bytes in the prepared buffer at %s.

In the example below, operator[5] has only 5 bytes, so %4s is used to prevent buffer overruns.

03-03-sscanf.c (code/03-03/03-03-sscanf.c)

Difference from 03-02 (no bug) (code/03-03/03-03-sscanf.c)

**Exercise**

**03-03-ex1**: Explain how the end of a string is represented in C language.

**03-03-ex2**: Create a test program and check the actual integer value of '\0’.

**03-03-ex3**: Explain what a null pointer is.

**03-03-ex4**: Create a test program and check the actual integer value of NULL.

**03-03-ex5**: Describe how to pass a "phrase with whitespace characters" (e.g. "hello world") as one string variable with sscanf().

### **03.04. Simple Calculator At**

At this point, all you have to do is to calculate based on the operator.

Create a different user-defined function for each of these different operations.

Use the strcmp() function to check what the operator is specified by the user, and call the corresponding user-defined function.

At this time, a conditional branch is provided internally so that only division is not divided by 0.

When the call is completed, the operation result is displayed and the operation is completed.

With such a program, if you want to increase the number of operators in the future, you can immediately do that by extending the program in the same way as the previous operators.

(It's not a structurally beautiful program yet, but I think this is the limit now in terms of the balance with the amount to be learned.)

Actually, I added an operator called “abs” here.

This calculates the absolute value of the difference between the two numbers.

Try adding the operator “max” in the same way.

This is an operator that tells you the larger of the two numbers.

(You can also prepare “min” in the same way.)

03-04-SimpleCalculator.c (code/03-04/03-04-SimpleCalculator.c)

Difference from 03-03 (code/03-04/03-03\_03-04.html)

**Exercise**

**03-04-ex1**: Append the following functions to the above simple calculator program.

- Operator "max". "A max B" is used to find the larger of the numerical values ​​of A or B.

- Operator “min”. "A min B" is used to find the smaller of the numerical values ​​A and B.

- Implementations corresponding to these operators should be described as functions in the source file, as with the operator “abs”.