Episode 13 (Key Points in C Language)

Tanuki, there are many computer languages, which can be broadly classified into procedural languages and object-oriented languages. Among the procedural languages, C is the most important, in my opinion, because it is a language similar to an assembly language and an object-oriented language. Furthermore, the various Linux operating systems are programmed in C.
When studying C, it is a standard practice to start with the basics, such as variables, arrays, and includes that use existing programs. Here, I'll focus on pointer variables and structs, where C learners fall behind.

The key points of the C language are, without a doubt, pointer variables and structures!

Pointer variables are prefixed with an asterisk (*)!

int *a;

It's an integer definition of a pointer variable. But like a normal numeric variable.

a = 56; (Not this one.)

error.

a = Memory Address; (This is OK.)

However, only one address that actually exists in memory can be assigned. An arbitrary address is not allowed.

int *a;

It is possible to assign an address directly to a pointer variable by doing the following

a = (int*)0x0000ff11 (Hexadecimal 8-digit)

However, if the directly assigned address is in the middle of a program, there is a risk that it will run out of control or the OS itself will be damaged, so this is not usually done.

Basically, the address is assigned via a variable as follows.

int b = 20;

a = &b;

which is the same as a = &b;.Using the image dia gram on the right,The value of &b is [00000100]. and the value of a will also be the first address[0000 0100].

Imaged figure of memory

address	Variable name b
(0x)00000100	00000000
00000101	00000000
00000110	00000000
00000111	00010100

Integer 4 bytes

(Decimal 20)

Variable b can be used as a variable, and the address of the storage location of the value of variable b is stored in the pointer variable a, so it can be used as needed.

Kitsune, understood!

If variable b specified as an int (integer) type allocates a 4-byte storage area in memory, and the first address of that storage area is [00000100] in hexadecimal (0x), then that address is assigned to the pointer variable a, right?

It's just that the value 20 (00010100 in binary) is not stored, but the location (address) is stored. OK, OK, I get it.

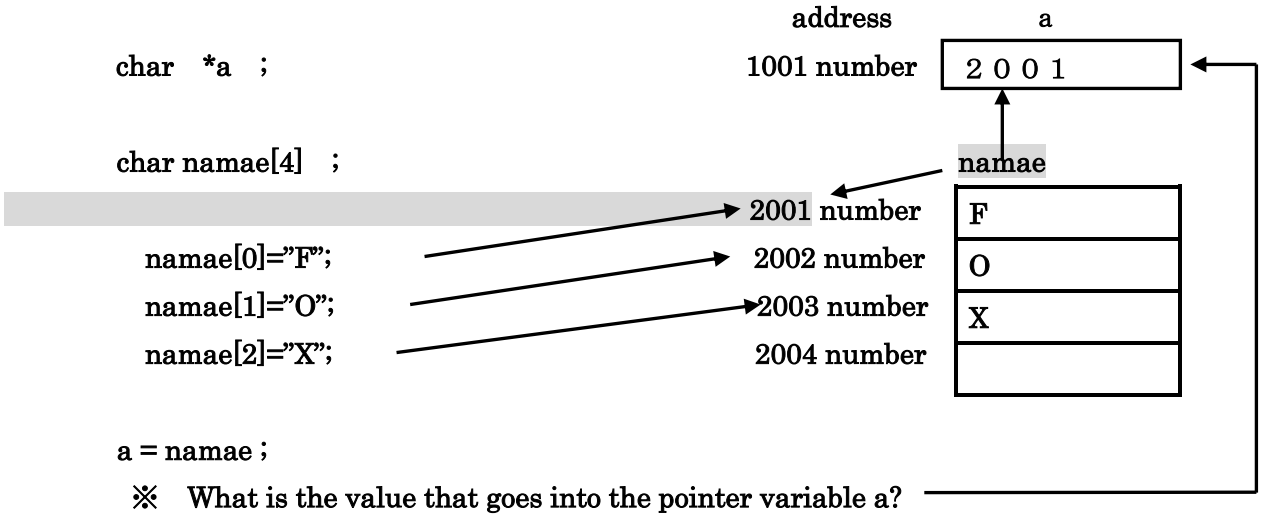
Now that we've talked about pointer variables of type integer (int), we should talk about pointer variables of type character (char).
The char in the character type declaration comes from the English word character. We must also talk about arrays.
Arrays handled in C are also pointer variables.
The pointer variable char *a corresponds to the variable char b, which corresponds to 1 byte of variable b. Since int b is 4 bytes, char *a cannot be used as a pointer variable for int b. This is an An error will result.
The types must be aligned so that a pointer variable of character type corresponds to a variable of character type, and a pointer variable of integer type corresponds to a variable of integer type.

Variable Declaration in C

```
char   *a   ;

char namae[4]   ;

namae[0]="F";
namae[1]="O";
namae[2]="X";

a = namae ;
```

※ What is the value that goes into the pointer variable a?

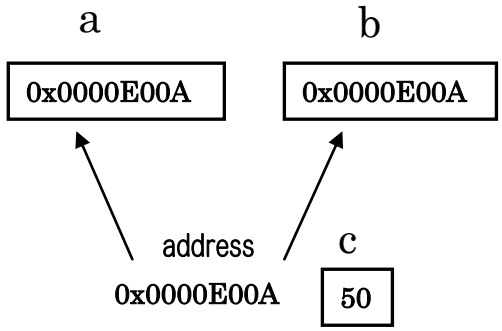
Imaged figure of memory



The program on the left declares a pointer variable of character type and an array of character type. Next, "F" is assigned to subscript 0, "O" to subscript 1, and "X" to subscript 2. The image on the right shows how the three characters are stored in memory. The memory area is addressed from 2001 to 2004 for the sake of convenience. Is the pointer variable *a created in a separate area from the array and its address set to 1001? So the array name namae is a pointer variable that stores the first address of the array (at 2001)? So the value of namae is assigned to a, so the value of a is [2001]. That makes sense!

When you use pointer variables, you get weird things like the following. Tanuki, you know what I mean.

```
int   *a, *b;
int   c;

c = 50;
a = &c;
b = a;
```



If I run this program, the values of *a and *b will be 50!

I know, I know.
You assign the address of the storage area that stores the value 50 in &c to the pointer variable a, and then you pass the address of variable c from a to the pointer variable b. In this way, the value 50 of c can be shared through the address. Also, you can retrieve the address of the variable's storage area by appending & to an ordinary variable. Strange, but I learned something.
I was just thinking that this address passing would make it possible to pass data between COBOL and FORTRAN, which are two different languages. Of course, it would be possible between COBOL and C, too.

Tanuki, you, my friend, have evolved!
Tanuki is right, you can pass data between different languages using address passing. In fact, complex calculations are done in C, and the results are stored in COBOL.

Now, let's talk about the second point, structures.
A structure is a package of data.
Inside the package, you can pack various variables, pointer variables, arrays, etc., and the package can be handled in a single life.

structure

Shape Declaration (Shape Layout)

```
struct      student {
(Structure) Declaration    Structure name (type)
    char name[20];     member variable
        ;character type    Array (20 pieces, 20 bytes)
    int kokugo;
        ;integer type
    int sansu;                                  member variable
        ;integer type


    }
```

student

name[0]
name[1]
name[2]
name[18]
name[19]
kokugo
sansu

How about this form: ・・・・?
No number (address) or anything is given.
There is only a form.

The characteristic of a structure is that only a framework for data entry is created.
The data is to be entered and become an entity (instance) in the form of name: Yamada, Japanese: 70, arithmetic: 80, name: Suzuki, Japanese: 60, arithmetic: 90, and so on, constrained by that framework.
Similar to classes in the Java language, structures can only define various variables. In contrast, a class can define data and a process (program) to be executed.

Now, let me show you an example of structure materialization (instantiation)

Embodiment example 01

struct student seito1 = { "FOX",70,60} ;

(Structure student) type variable(seito1)

An address is assigned only after materialization.

seito1 (pointer variable)

2001 number | 1001 | address

Data Acquisition

seito1.name

1001 number ; name FOX

seito1.kokugo

1001 number ;kokugo 70

seito1.sugaku

1001 number ;sugaku 60

address	
1001 number	F
1002 number	O
:	X
:	
	:
	blank space
	70
	60

Similarly, materialization in the case of student 2 (seito2)

struct student seito2 = {"Raccoon",90,80} ;

(Structure student) type variable(seito2)

An address is assigned only after materialization.

seito2

2002 number | 1501 | address

Data Acquisition

seito2.name

1501 number;name Raccoon

seito2.kokugo

1501 number;kokugo 90

seito2.sugaku

1501 number;sugaku 80

address	
1501 number	R
1502 number	a
:	c
:	c
	:
	blank space
	90
	80

I see what you mean, it is convenient to be able to input data for one person at a time. It would also prevent some input errors.
Computers are being improved to make them more convenient for people to use, so it is not surprising that they came up with the idea of a structure.

As I said before, structures and classes are different, but with structures and function pointers, you can create pseudo-classes. A class encapsulates (data + processing). A structure is only member variables (i.e., data). Since a function is a mass of processing, you can understand that you can create a pseudo-class by adding a function to a structure. Explaining pseudo-classes makes C look difficult, so I'll stop. What I want you to understand is that C, a procedural language, is connected to C++ and Java, object-oriented languages.

Next, in Episode 14, I will talk about the key points of the Java language, including classes.

Translated at DeepL