Practice of Information Processing

（IMAC）

Third Lecture(part 3)：Array(1)

Makoto Hirota

Contents of the latter half

- Array
  - Concept
  - Index
  - Initialization
  - Strings and array
  - Multi-dimensional array

Array

- The way managing many variables of the same type by assigning one name and consecutive numbers (index)

Ex) When you want to prepare 100 int type variables,

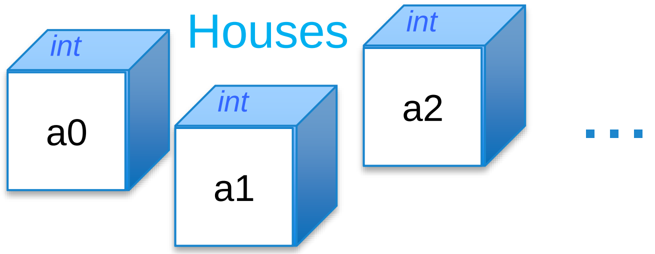
<Previous variable declaration>

```
int a0, a1, a2, …, a99;
a0=10;
a1=20;
```

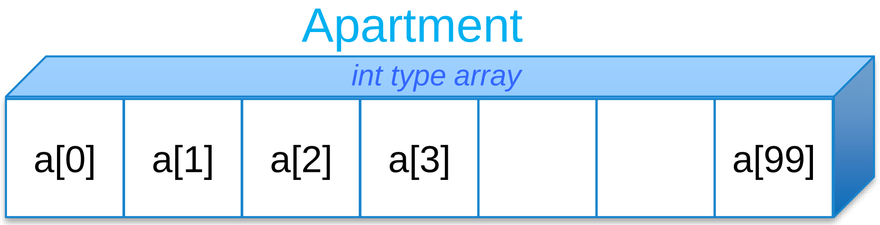
<Variable declaration using array>

```
int a[100];
a[0]=10;
a[1]=20;
```

Add [] (braces) after the variable name



Schematic of single variable



Schematic of array

A separate data storage location (memory) is prepared for each variable

A continuous memory space is allocated.

"Variable" has "Address" in memory space

Index of array

- Specify the number of elements in [] when declaring an array

ex) `int a[5];` Prepare an int type array `a` with 5 elements

`int b[100];` Prepare an int type array `b` with 100 elements

Actually prepared array

`a[0], a[1], a[2], a[3], a[4];` (5 elements)

`b[0], b[1], b[2], ··· , b[99];` (100 elements)

Since the subscript starts from 0, the subscript of the array with the number of elements N is 0 to N-1.

Variables a [5], b [100] do not exist
Be careful because it is a frequent mistake in the program!

Initialization of array

- In the array declaration, only the memory area is reserved, and it is unknown what is inside (indefinite).
- Like variables, it needs to be initialized
  - If you use it without initialization, unexpected values may be used.
  - Be careful when compiling without the -Wall option
- Two ways of initialization

reference:
int a [3] = {};
0 will be assigned into all variables

（1）Assign individually

```
int a[3];

a[0]=10;
a[1]=20;
a[2]=30;
```

`for` statement is very convenient when you input all zeros.

（2）Set when declaring

```
int a[3] = {10,20,30};
```

Note: The following assignments are not possible

✗

```
int a[3];
a = {10,20,30};
```
（compile error）

Ex： Definition and manipulation of Array array.c

- Compile and run the sample program array.c

```
#include <stdio.h>

#define N 3 /* Macro definition */

int main()
    /**** variable declaration****/
    int a[N];
    int i, j;

    /**** processing contents****/
    /* initialization of array */
    for (i = 0; i < N; i++){
        a[i] = 0;
    }

    /* array operation */
    a[0] = 10;
    a[1] = 20;
    a[2] = 30;

    /* array display */
     for (j = 0; j < N; j++){
        printf("a[%d] = %d¥n", j, a[j]);
    }

    return 0;
}
```

```
$ gcc -Wall -o array array.c
$ ./array
```

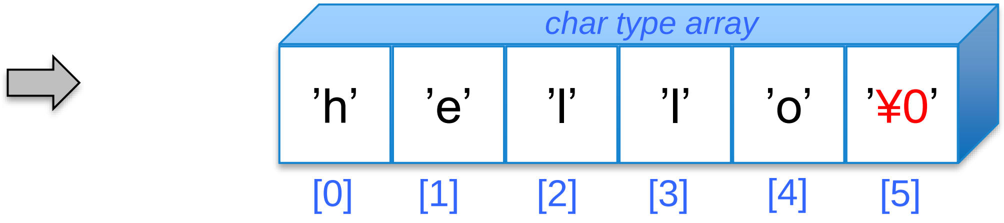
The `for` statement is often used to manipulate arrays. Understand the relationship between the number of elements and subscripts of the array and the variables `i` and `j` in the `for` statement.

- The #define part at the beginning of a sentence is called a macro definition, and a specific character string (macro) in the program can be replaced with another character string.
- If you write `"#define N 3",` the preprocessor (procedure before compilation) will replace `"N"` in the program with `"3"`. It is custom to write macro names in uppercase.
- The advantages of using macros are as follows.
    - You can easily change the constants related to the entire program (In this example, the number of elements in the array can be easily changed)
    - Readability is improved because programs can be written with constant names instead of numbers (In this example, using `N` instead of `3` makes it easier to understand the number of elements)

Strings and arrays

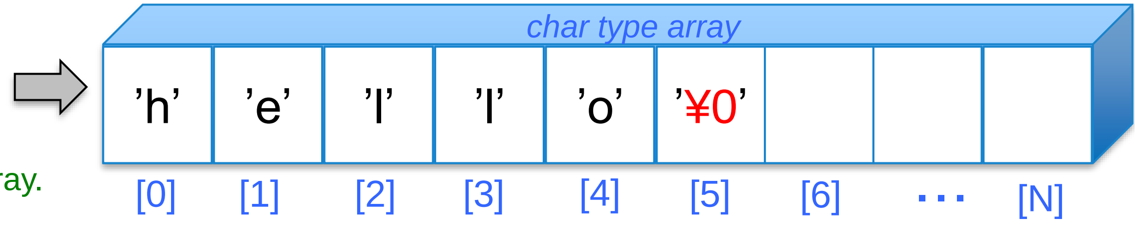
- In C language, strings are represented as char type arrays
- At the end of the string, the special character ‘¥0’ (NUL character) indicating the end of the string is inserted.

```
char str[] = “hello”;
```



* When declaring an array, the number of elements in [] can be omitted. At this time, the memory is secured according to the number of elements given at the time of initialization.

```
char str[N] = “hello”;
```



* When a character string is assigned to a large array. (A character string can be assigned in such an assignment statement only at the time of declaration.)

In order to handle character strings with different lengths in an array with a fixed number of elements in this way, processing to detect NUL characters is required.

Ex) Extracts and displays character strings character by character

```
while (str[i] != '¥0') {
      printf("%c¥n", str[i]);
      i++;
}
```

results

```
h
e
l
l
o
```

Ex： Array as character strings :string.c

■ Compile and run the sample program string.c

```
#include <stdio.h>

#define LEN 100 /* Macro definition */

int main()
    /**** variable declaration****/
    char str[LEN];
    int i;
    int n=0;

    /**** processing contents****/
    /* initialization of array */
    for (i = 0; i < LEN; i++){
        str[i] = 0;
    }

    /* input of strings */
    scanf("%s",str);  ← & is not required for array

    /* display strings as array*/
    /* repeat while str[n] is not ¥0 */
    while (str[n] != '¥0'){
        printf("str[%d] = %c¥n", n, str[n]);
        n++;
        if(n >= LEN){
            break;
        }
    }

    return 0;
}
```

```
$ gcc -Wall -o strings strings.c
$ ./strings
hello ↲ (Enter)
```

When reading a character string from the keyboard, give the array a sufficient length (LEN = 100).

Since it is not necessary to display 100 array elements, the end of the character string is determined by the '¥0'(NUL character) that is automatically inserted at the end of the input character string.

＜question＞What happen when inputting “hello world”

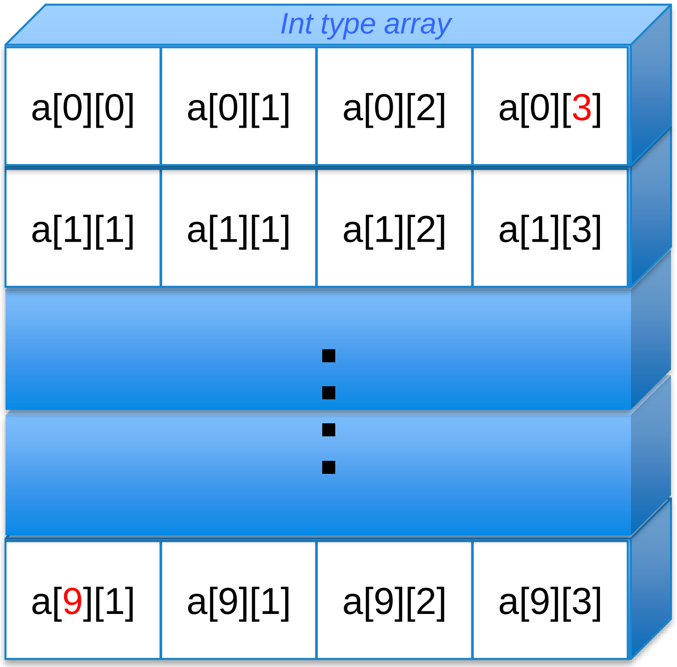
```
$ ./strings
hello world ↲ (Enter)
```

Consider reason

Reference: Sample program string2.c corresponding to the space

Multidimensional array

- Array with two or more subscripts
  - Effective when dealing with coordinates and spreadsheets
- A two-dimensional array
  - Example: `int a [10] [4];`



Declaration of array

```
int array[M][N];
```

Similar to a one-dimensional array,
the subscripts of the array are the value up
to M-1, N-1 for the number of elements M, N

Initialization of multidimensional array

- （1）Set with declaration

```
int a[4][2] = {{1,2}, {3,4}, {5,6}, {7,8}};
```

Reference: it is sufficient to initialize with all 0s
int a [4] [2] = {{}}; Or
int a [4] [2] = {};

- （2）Assign individually
  - A two-dimensional array is initialized with double loops (nested `for` statements).

```
int a[M][N];
int i,j;
for (i=0;i<M;i++){
    for (j=0;j<N;j++){
        a[i][j] = 0;
    }
}
```

Suppose M and N are defined by macros

Ex) Display of 2D array score.c

- Compile and run the sample program score.c

```
/*
score.c
*/

#include <stdio.h>
/* Macro definition */
#define ID_NUM 10
#define SUB_NUM 3

int main()
    /**** variable declaration****/
    int score[ID_NUM][SUB_NUM]={{80,94,80},{70,71,60},{65,83,73},{75,80,65},
        {78,88,83},{92,100,98},{82,88,93},{79,85,80},{88,95,90},{66,70,72}};
    int i;

    /**** processing contents****/
    /* display scores*/
    printf("ID  SUB1  SUB2  SUB3¥n");
    for (i = 0; i < ID_NUM; i++){
        printf("[%d]  %3d  %3d  %3d¥n", i, score[i][0],score[i][1],
               score[i][2]);
    }
    return 0;
}
```

Understand this source as it will be used in the interim report, next week.

<Practice>
Redirect the output result on the console and save it as text.

Redirect

- You can easily save the output of the program to a file by using the method "Redirect (>)" on the console terminal.
  - Redirection is a method of changing the standard output destination of a program to a file.
  - The standard output is where printf etc. are displayed. In the terminal, the console terminal is the standard output destination.

```
$ ./score > score.txt
$ cat score.txt  ← cat is command to output the contents
```

The output contents of the execution result are saved in a file called score.txt. Please open it in an editor

Exercise 3-4: Input to a one-dimensional array array_input.c

- Create a program array_input.c that gets 5 integers into array `a` from the keyboard, and that outputs the values of each array element and the sum..

＜example＞

```
$ ./array_input
10 ↵ (Enter key)
20 ↵
30 ↵
40 ↵
50 ↵
Your inputs are:
a[0]=10
a[1]=20
a[2]=30
a[3]=40
a[4]=50
Total sum is 150
```

Exercise 3-5: Turn over a string reverse.c

- Create a program reverse.c that converts the character string input from the keyboard in reverse order from the end and displays it.

<example>

```
$ ./reverse
hello ↵（Enter key）
olleh
```

If you can afford it, refer to string2.c so that you can invert the string with spaces.
“Hello World” → “dlroW olleH”

Exercise 3-6: Matrix calculation for multidimensional arrays

- Create a program matrix_multiple.c that finds the product C of the 4-by-3 matrix A and the 3-by-4 matrix B.
- C = A × B

$$A=\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

$$B=\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

The matrix values have already been entered in the sample source matrix_multiple.c, so please use it.

points:
1. Be aware of which is the row and which is the column when coding
   ```
   int A[4][3]
   ```
   row column
2. Matrix subscripts start at 0

＜example＞

```
$ ./matrix_multiple
C[0][0] = 14
C[0][1] = 20
C[0][2] = 26
C[0][3] = 32
C[1][0] = 35
C[1][1] = 50
C[1][2] = 65
C[1][3] = 80
C[2][0] = 14
C[2][1] = 20
C[2][2] = 26
C[2][3] = 32
C[3][0] = 35
C[3][1] = 50
C[3][2] = 65
C[3][3] = 80
```

```
$ ./matrix_multiple
C =
   | 14  20  26  32|
   | 35  50  65  80|
   | 14  20  26  32|
   | 35  50  65  80|
```

Summary

- Review of Previous Lectures
- Selection and iterative processes
  - Selection process： if， switch statements
  - Iterative process： for， while statements
- Array
  - Concept
  - Index
  - Initialization of array
  - Strings and array
  - Multidimensional array

Next

- Example answer of exercise
- Exercise of array
- Midterm report

- See you next week!