

Programming Basics 7

Instructor Younghoon Kim



The background of the slide is a light blue gradient. In the top right corner, there is a close-up of a silver computer keyboard, showing keys like 'delete', 'enter', 'return', and 'shift'. Below the keyboard is a white computer mouse. In the bottom left corner, there is a silver pen. In the bottom right corner, there is a spiral-bound notebook with a light blue cover. A large, blue, rounded rectangular banner is positioned in the center of the slide, containing the text 'Array Basics' in white.

Array Basics

- **Get two numbers from a user**
 - `int a, b;`
- **Get 10 numbers from a user**
 - `int a, b, c, d, e, f, g, h, i, j;`
- **Get 100 numbers from a user**
 - `int a1, a2, a3, ..., a100;`
- **Get N numbers from a user ?**

- A group of contiguous memory locations
- All elements have the same type.

All elements of this array
share the array name, c

→ c[0]

c[1]

c[2]

c[3]

c[4]

c[5]

c[6]

c[7]

c[8]

c[9]

c[10]

c[11]

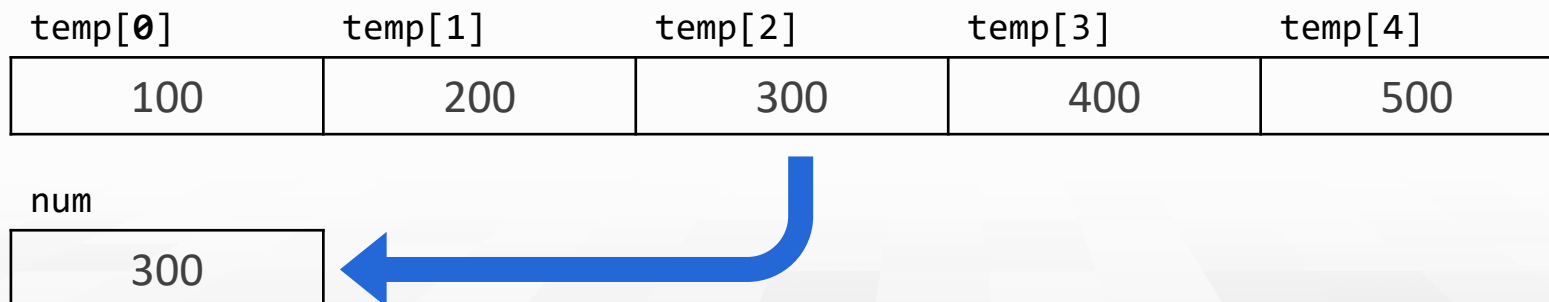
Position number of the
element within array c

-45
6
0
72
1543
-89
0
62
-3
1
6453
78

```
int temp[5] = {100, 200, 300, 400, 500};  
int num = temp[2];    // num = 300;
```

■ Above code

- defines an array with squared brackets ([]) specifying the number of elements.
- initializes the array with an initializer list within curly brackets ({}).
- refers third element using the array's name followed by the position number in square brackets ([]).



Array Declaration

■ How to declare an array variable

- Similar to regular variables

```
int num;           // defining an integer variable
int numbers[10];  // defining an integer array with 10 elements
```

- **data type** must be firstly specified.
- **variable name** will follow.
- **squared brackets** make the variable an array variable.
- The **number of elements** will be placed within brackets.
 - » can be omitted with initialization.

- ### ■ Specifying the type and the number of elements makes the computer reserve the appropriate amount of memory.

Initializing an Array in a Declaration

- Arrays are not automatically initialized.
- Uninitialized array elements contain garbage values.
- 'Initializer list' or 'Array initializer'
 - An array can be initialized with curly brackets({ }) following an equals sign (=).
 - Array initialization with curly brackets is permitted **only** in a declaration.
 - Not specified elements are initialized with zeroes.
 - The size of one-dimensional array is not necessarily specified.
 - » Only when the initialization is following

```
int n[5] = { 1, 2, 3, 4, 5 };

int n[5] = { 0 };           // unassigned elements
                           // have zeroes as values

int n[] = { 1, 2, 3, 4, 5 }; // unspecified size is set to
                           // the number of elements
```

- **String variables in C are with 'char array' data type.**
 - Use squared brackets after the variable name to specify the string length.
 - The size of the string **MUST** be greater than the actual string length.
- **In fact, "hello" is a character array with the size of six.**
 - The last element is a **null character (\0)**.

```
char str[6] = "hello";  
char str[] = "hello";  
char str[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
```

- Initialization with " " is also permitted **only** in a declaration.
- **char name[20] can have at most 19 alphabets.**

- Initialization with " " is permitted only in a declaration.
- A string of characters enclosed in " " is called a string literal
 - a.k.a *string constant* or *constant string*
 - String literals are stored in C as an array of chars, terminated by a null byte.
 - They are constant, so string literals cannot be modified.
- When an array is declared with a string literal,
 - the computer assigns a certain size of memory for the array,
 - and it reads the string literal and copy the characters into that memory.
 - It is automatically done only with array declarations, so initialization cannot be done after declaration.
- We will visit this slide again after learning pointers.

Initializing an Array with a Loop

- Arrays are not automatically initialized.
- Below code uses for statements to initialize the elements of a five-integer array `n` to zeros.

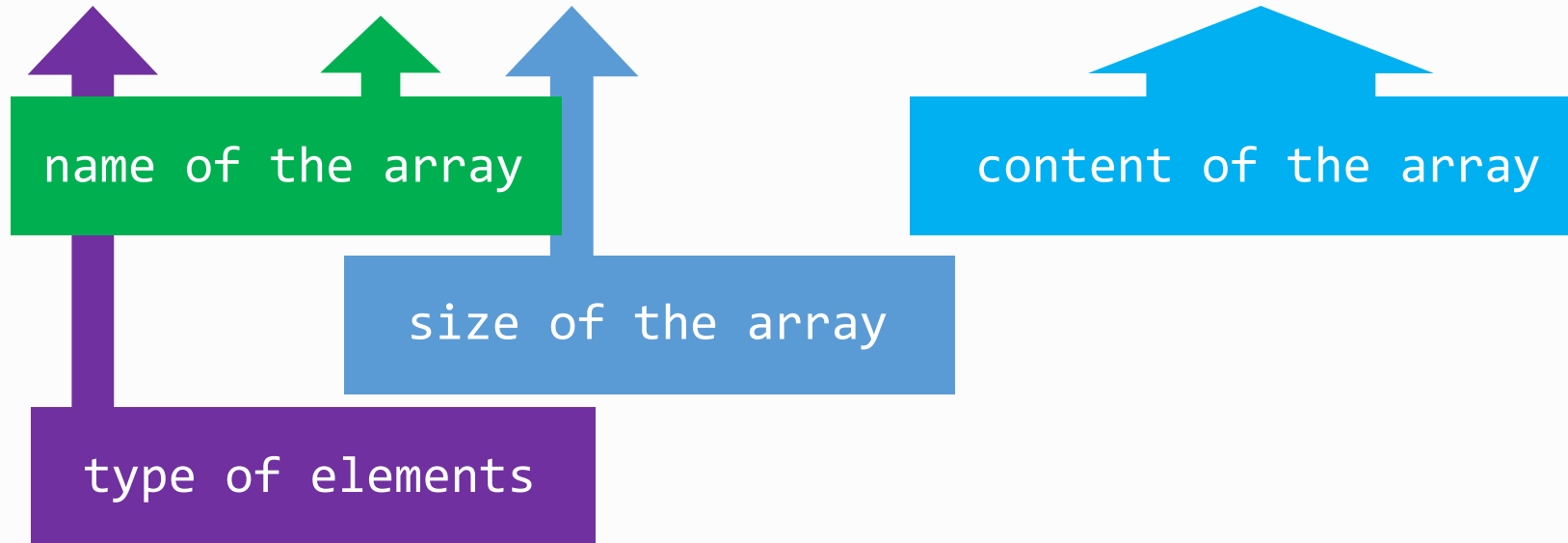
```
int n[5];  
for( size_t i=0; i<5; i++) {  
    n[i] = 0;  
}
```

- **size_t data type**
 - represents an unsigned integral type according to C standard.
 - is recommended for any variable representing an array's size or an array's indices.
 - is defined in header `<stddef.h>`, which is often included by other headers.

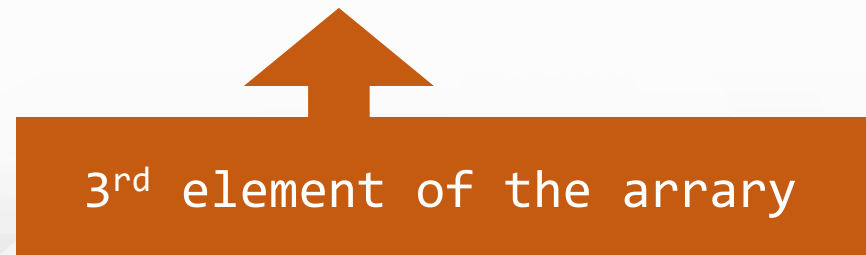
- **Referring to a particular location or element**
 - The position number within square brackets is called an **index**.
 - We must specify the array's name and the index.
- **The first element in every array is the zeroth element.**
 - Zero-based numbering or zero indexed arrays
- **An index must be an integer or an integer expression.**
 - Example:

```
a=5, b=6;  
c[11] = 2;      // an integer index  
c[a+b] = 2;     // an integer expression index, same result  
b = c[1];
```

```
int array_temp[5] = {100, 200, 300, 400, 500};
```



```
int num = array_temp[2];
```



- **Two arrays with initializer lists are used in the example.**
 - Without the initializer, we can see garbage values.
 - What happens when the initializer has more elements than the size?
 - We will check different ways of initializing a string variable.

```
#include <stdio.h>
int main() {
    int numbers[10];
    //int numbers[10] = {0, };
    for(int i=0; i<10; i++)
        printf("%d ", numbers[i]);
    printf("\n");
    char hello[6] = "hello";
    printf("%s\n", hello);
    return 0;
}
```

- **An integer array is initialized with a loop.**
 - We can use int data type instead of size_t. What's the difference?

```
#include <stdio.h>
int main() {
    int n[5];
    for( size_t i=0; i<5; i++) {
        n[i] = 0;
    }
    for( size_t i=0; i<5; i++) {
        printf("%d ", n[i]);
    }
    printf("\n");
    return 0;
}
```

■ An example of array definition

```
int b[100], x[27];
```

- Above definition reserves 100 elements for integer array b and 27 elements for integer array x.
 - » Indices lie in the ranges 0–99 and 0–26, respectively.

■ An example of array indexing

- Printing the sum of the values contained in the first three elements of array c

```
printf("%d", c[0] + c[1] + c[2]);
```

■ Symbolic constant

- The `#define` preprocessor directive is used for defining symbolic constants.
- A symbolic constant is an identifier that's replaced with replacement text by the C preprocessor before the program is compiled.
- All occurrences of the symbolic constant are replaced with the replacement text following the constant.

```
(Somewhere in the code)
#define NUM 10           // defining symbolic constant before using it
```

```
...
```

```
(Somewhere below in the code)
printf("%d\n", NUM);      // OK. NUM is replaced with text '10'
printf("NUM\n");         // 'NUM' will be printed instead of '10'
printf("%s\n", NUM);     // ERROR. '%s' is for strings not for integers
```


Array Example with Symbolic Constant

- **Specifying an Array's Size with a Symbolic Constant**

- Useful when the target array size is fixed during the execution.

```
#define N 10

int a[N], i;
for (i = 0; i < N; i++)
    a[i] = 1;                // initializes a

for (i = 0; i < N; i++)
    scanf("%d", &a[i]);      // reads data into a

for (i = 0; i < N; i++)
    sum += a[i];             // sums the elements of a
```

- **for loops are a well-match to array operations.**

- It naturally supplies indexes for the array.

Exercise: Reversing an array

DEMO

- Fill in the empty box to print elements in a reverse order.

```
#define N 10
int a[N], i;
for (i = 0; i < N; i++)
    a[i] = 1;           // initializes a

for (i = 0; i < N; i++)
    scanf("%d", &a[i]); // reads data into a

for (????;????;????)
    printf("%d ", a[i]);
printf("\n");
```

Exercise: Reversing an array (Solution)

DEMO

- Fill in the empty box to print elements in a reverse order.

```
#define N 10
int a[N], i;
for (i = 0; i < N; i++)
    a[i] = 1;           // initializes a

for (i = 0; i < N; i++)
    scanf("%d", &a[i]); // reads data into a

for (i = N-1; i>=0; i--)
    printf("%d ", a[i]);
printf("\n");
```

- Arrays are data structures consisting of related data items of the same type. Later, we discuss C's notion of struct (structure)—a data structure consisting of related data items of possibly different types. Arrays and structures are “static” entities in that they remain the same size throughout program execution.
- An array is a group of contiguous memory locations that all have the same type. To refer to a particular location or element in the array, we specify the array's name and the position number of the particular element in the array.
- Any element in an array be referred to by giving the array's name followed by the position number of the particular element in square brackets ([]). The first element in every array is the zeroth element. An array name, like other identifiers, can contain only letters, digits and underscores and cannot begin with a digit. The position number within square brackets is called an index or subscript. An index must be an integer or an integer expression.

Programming Basics 7

Instructor Younghoon Kim



The background of the slide is a light blue and white composition. In the top right corner, a portion of a silver computer keyboard is visible, showing keys like 'delete', 'enter', 'return', and 'shift'. Below the keyboard is a white computer mouse. In the bottom left corner, a silver pen is shown. In the bottom right corner, the spiral binding of a notebook is visible. A large blue rounded rectangle is centered on the slide, containing the title text.

Example Program with an Array

- **Problem statement**

- Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (1 means awful and 10 means excellent). Place the 40 responses in an integer array and summarize the results of the poll.

- **We wish to summarize the number of responses of each type (i.e., 1 through 10).**

■ Source code

```
#include <stdio.h>
#define RESPONSES_SIZE 40
#define FREQUENCY_SIZE 11

int main()
{
    int frequency[FREQUENCY_SIZE] = {0, };

    int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10};

    for(size_t answer = 0; answer < RESPONSES_SIZE; answer++) {
        frequency[responses[answer]]++;
    }

    printf("%s%17s\n", "Rating", "Frequency");

    for(size_t rating = 1; rating < FREQUENCY_SIZE; rating++) {
        printf("%6lu%17d\n", rating, frequency[rating]);
    }

    return 0;
}
```



```
#include <stdio.h>
#define RESPONSES_SIZE 40
#define FREQUENCY_SIZE 11

int main()
{
    int frequency[FREQUENCY_SIZE] = {0, };

    int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
                                     1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
                                     5, 6, 7, 5, 6, 4, 8, 6, 8, 10};
```

- The array `responses` is a 40-element array of the students' responses.
- We use an 11-element array `frequency` to count the number of occurrences of each response.
 - We ignore `frequency[0]` because it's logical to have response 1 increment `frequency[1]` rather than `frequency[0]`.
 - This allows us to use each response directly as the index in the frequency array.

```
for(size_t answer = 0; answer < RESPONSES_SIZE; answer++) {  
    frequency[responses[answer]]++;  
}
```

- The for loop takes the responses one at a time from the array responses and increments one of the 10 counters (frequency[1] to frequency[10]) in the frequency array.
- The key statement in the loop is the bold line which increments the appropriate frequency counter depending on the value of responses[answer].

```
int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, ... , 10};
```

- When answer is 2, responses[answer] is 6, so ++frequency[responses[answer]]; is interpreted as ++frequency[6]; which increments array element six, and so on.
- Regardless of the number of responses processed in the survey, only an 11-element array is required (ignoring element zero) to summarize the results.

```
printf("%s%17s\n", "Rating", "Frequency");

for(size_t rating = 1; rating < FREQUENCY_SIZE; rating++) {
    printf("%6lu%17d\n", rating, frequency[rating]);
}
```

- Above part is for summarizing the survey results.
- Check that the for loop uses the loop controlling variable directly as the index in the frequency array.
 - FREQUENCY_SIZE is 11, so we can visit all elements in the frequency array. (1 – 10)
- Follow-up question: How can we make the program to print the average, min and max rating?

The background of the slide is a light blue gradient. In the top right corner, there is a partial view of a silver computer keyboard with white keys. Below the keyboard is a white computer mouse. In the bottom left corner, there is a silver pen. In the bottom right corner, there is a spiral-bound notebook with a light blue cover. A large, blue, rounded rectangular banner is positioned in the center of the slide, containing the text "Min/Max" in white.

Min/Max

Q: What is the minimum value of an array?

- For a given data, the max or min value is commonly asked.
- How can we get the max or min value?
 - We need to visit all elements.
 - We need a temporal variable for storing current max/min value.

90	55	100	95	45	70	80	80	100	60
----	----	-----	----	----	----	----	----	-----	----

current_min_value	<i>(Updated while visiting)</i>
-------------------	---------------------------------

1 Declare a variable

2 Initialize it with a upper/lower bound of data

- It is OK to initialize the min/max variable as the first value of the array.

```
...  
int min_score = 100; // Upper bound is 100 for scores  
for (int i = 0; i < 10; i++) // Assuming 10 students  
    if (min_score > scores[i])  
        min_score = scores[i];  
...
```

- CHECKPOINT: Why upper bound for min and lower bound for max?

```
printf("%s%17s\n", "Rating", "Frequency");

for(size_t rating = 1; rating < FREQUENCY_SIZE; rating++) {
    printf("%6lu%17d\n", rating, frequency[rating]);
}
```

- Above part is for summarizing the survey results.
- Check that the for loop uses the loop controlling variable directly as the index in the frequency array.
 - FREQUENCY_SIZE is 11, so we can visit all elements in the frequency array. (1 – 10)
- **Follow-up question: How can we make the program to print the average, min and max rating?**

Exercise: Min, Max, Avg for Survey Results

DEMO

- Modify the program to print min, max and average values of the survey.

```
#include <stdio.h>
#define RESPONSES_SIZE 40
#define FREQUENCY_SIZE 11

int main()
{
    int frequency[FREQUENCY_SIZE] = {0, };

    int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10};

    for(size_t answer = 0; answer < RESPONSES_SIZE; answer++) {
        frequency[responses[answer]]++;
    }

    printf("%s%17s\n", "Rating", "Frequency");

    for(size_t rating = 1; rating < FREQUENCY_SIZE; rating++) {
        printf("%6lu%17d\n", rating, frequency[rating]);
    }

    return 0;
}
```

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

Min: 1, Max: 10, Avg: 6.025

Exercise: Min, Max, Avg for Survey Results (Solution)

DEMO

- Modify the program to print min, max and average values.

```
...
int min = 10, max = 1, sum = 0;

for(size_t answer = 0; answer < RESPONSES_SIZE; answer++) {
    frequency[responses[answer]]++;
    if(responses[answer] > max) {
        max = responses[answer];
    }
    if(responses[answer] < min) {
        min = responses[answer];
    }
    sum += responses[answer];
}

printf("%s%17s\n", "Rating", "Frequency");
for(size_t rating = 1; rating < FREQUENCY_SIZE; rating++) {
    printf("%6lu%17d\n", rating, frequency[rating]);
}
printf("\nMin: %d, Max: %d, Avg: %.3lf\n", min, max, (double)sum/RESPONSES_SIZE);
return 0;
}
```

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3
Min: 1, Max: 10, Avg: 6.025	