SIES: BASIC C PROGRAMMING

L #12: C ARRAYS

Seung Beop Lee

School of International Engineering and Science

CHONBUK NATIONAL UNIVERSITY

Outline

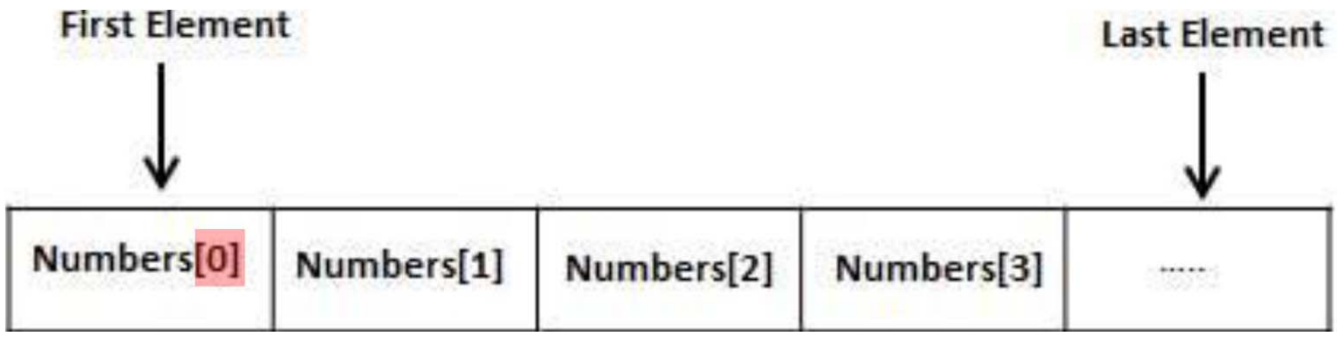
- Declaring Arrays
- Initializing Arrays
- Accessing Array Elements
- Arrays in Detail

Array

Array

```
/* one by one initialization */
double balance[5];
balance[0] = 10;
balance[1] = 20;
balance[2] = 30;
balance[3] = 40;
balance[4] = 50;
```

- What is the array in C?
  - C programming language provides a **data structure** called the **array**, which can store a **fixed-size** sequential collection of **elements** of the **same type**.
  - An **array** is used to store a collection of data, but it is often more useful to think of an array as a **collection of variables** of the **same data type**.
    - ➢ Instead of declaring individual variables, such as number0, number1, ..., and number99, you can declare **one array variable** such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.
  - Note that all arrays consist of **contiguous memory locations** and the **number of the elements in the array** starts from zero.
  - The **lowest address** corresponds to the **first element** and the **highest address** to the **last element**.



Declaring Arrays

- Syntax
  - To **declare** an array in C, a programmer specifies the **type of the elements** and the **number of elements** (i.e. arraySize) required by an array as follows:

```
type arrayName [ arraySize ];
```

  - This syntax is called a **single-dimensional** array.
  - Note that the **arraySize** must be an **integer constant** greater than zero and **type** can be **any valid C data type**.
    - ✓ There is an example to declare a **10**-element array called **balance** of type double:

```
double balance[10];
```

    - ✓ Now **balance** is **a variable array** which is sufficient to hold up-to 10 double numbers because the data type is double and the number of the elements is 10.

Initializing Arrays

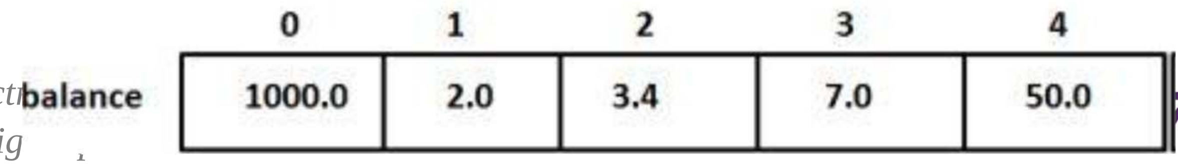
- Syntax
  - You can **initialize array** in C either **one by one** or using **a single statement** as

```c
/* one by one initialization */
double balance[5];
balance[0] = 10;
balance[1] = 20;
balance[2] = 30;
balance[3] = 40;
balance[4] = 50;

// a single statement initialization
double balance[5] = { 10, 20, 30, 40, 50 };
```

  - Note that the **number of values** between braces { } **can not be larger** than the **number of elements** that you declare for the array between square brackets [ ].

  - If you **omit** the size of the array, an array <u>which is big enough to hold the initialization</u> is created.

```c
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```



Accessing Arrays Elements

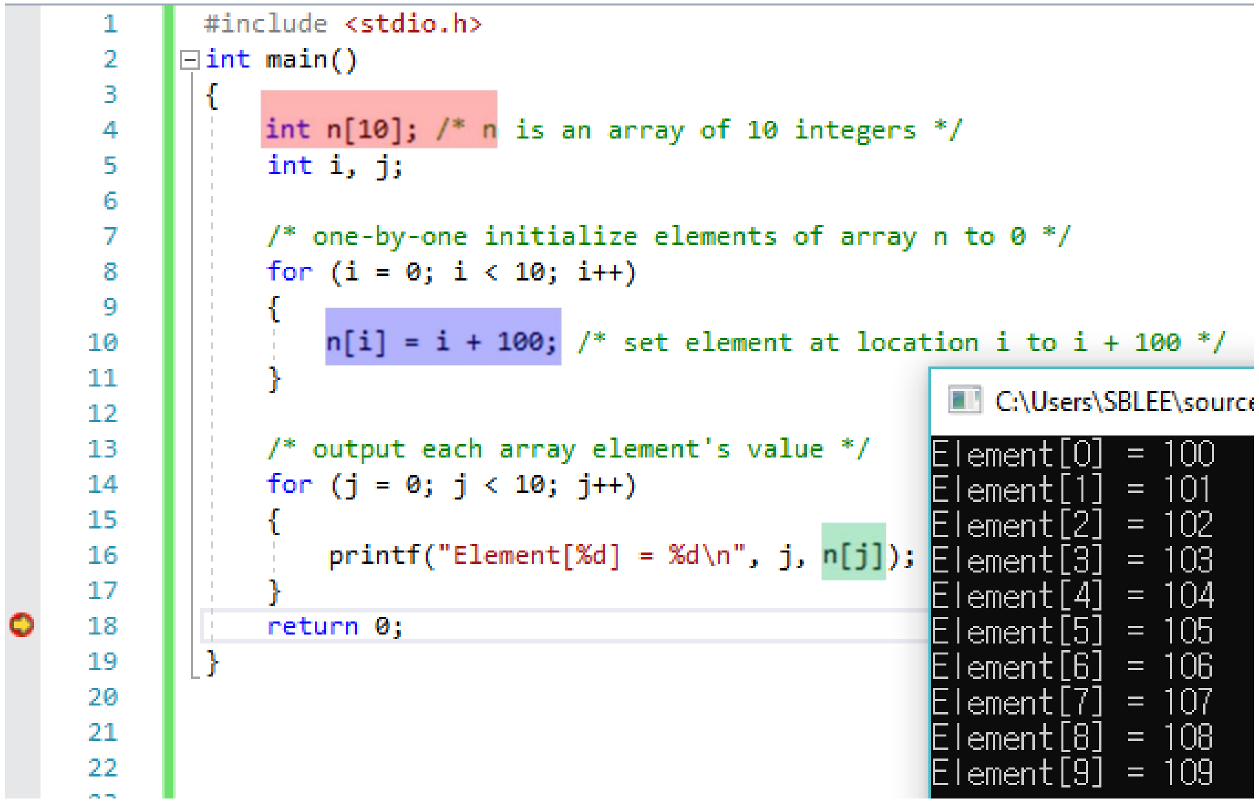
- Syntax
  - **An element** is accessed by **indexing** the array name. This is done by placing the **index of the element** within **square brackets** after the **name of the array**. For example:

```
double salary = balance[9];
```

  - The above statement will take the value of the *10th element* from the array *balance* and assign the value to the variable *salary*.

Accessing Arrays Elements

- Example
  - The following is an example which will use all the above mentioned three concepts viz. **declaration**, **assignment** and **accessing** **arrays**:



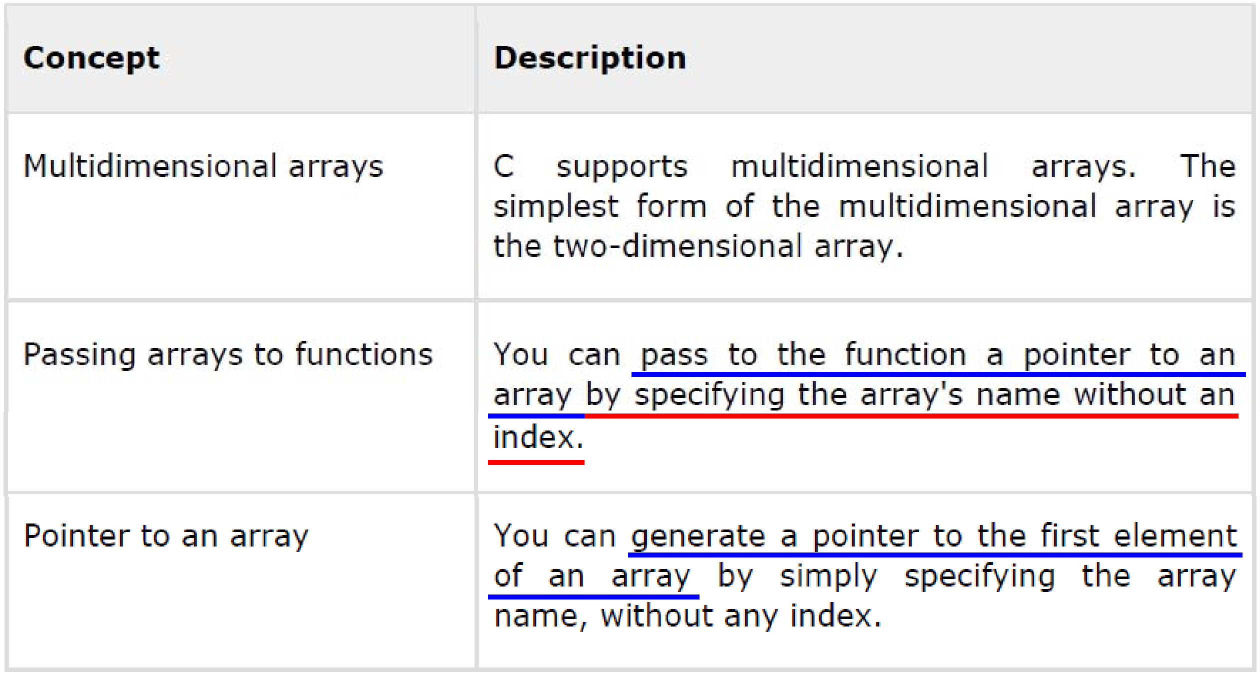
```c
#include <stdio.h>
int main()
{
    int n[10]; /* n is an array of 10 integers */
    int i, j;

    /* one-by-one initialize elements of array n to 0 */
    for (i = 0; i < 10; i++)
    {
        n[i] = i + 100; /* set element at location i to i + 100 */
    }

    /* output each array element's value */
    for (j = 0; j < 10; j++)
    {
        printf("Element[%d] = %d\n", j, n[j]);
    }
    return 0;
}
```

C:\Users\SBLEE\source

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Arrays in Detail

- Four important concepts related to array
  - Arrays are important to C and should need a lot more attention.
  - The following important **concepts** related to array should be clear to a C programmer:



Concept	**Description**
Multidimensional arrays	C supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
Passing arrays to functions	You can pass to the function a pointer to an array by specifying the array's name without an index.
Pointer to an array	You can generate a pointer to the first element of an array by simply specifying the array name, without any index.

Multidimensional Arrays

- Syntax
  - C programming language allows **multidimensional arrays**. Here is the general form of a multidimensional array declaration:

```
type name[size1][size2]...[sizeN];
```

  - For example, the following declaration creates a **three dimensional** 5 . 10 . 4 integer array, as follows:

```
int threedim[5][10][4];
```

Multidimensional Arrays

- Two-Dimensional Arrays
  - The simplest form of the **multidimensional** array is the **two-dimensional** array.
  - To declare a two-dimensional integer array of size x, y, you would write something as follows:

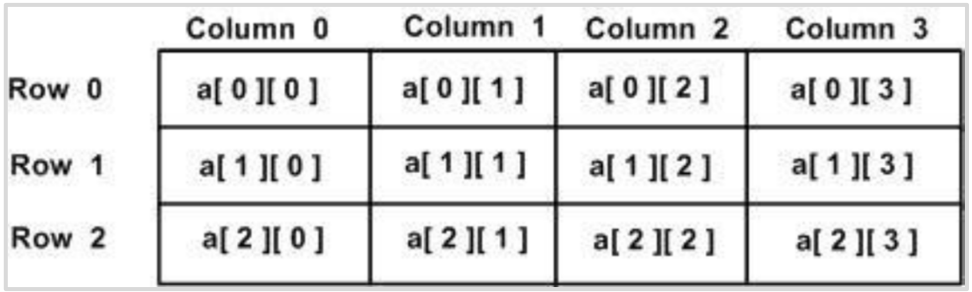
```
type arrayName [ x ][ y ];
```

where **type** can be <u>any valid C data type</u> and **arrayName** will be <u>a valid C identifier</u>.

A two-dimensional array can be think as a table which will have <u>x number of rows</u> and <u>y number of columns</u>.

  - A 2-dimentional array **a**, which contains three rows and four columns can be shown as below:

```
int a[3][4];
```

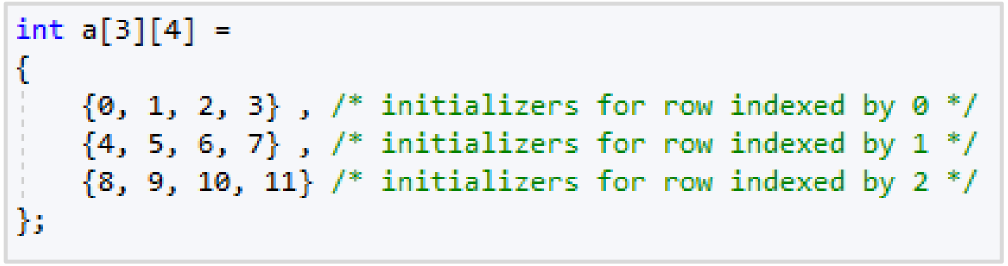


	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

  - Thus, **every element** in array **a** is identified by <u>an element name of the form a[i][j]</u>, where **a** is the **name of the array**, and **i and j** are the subscripts that <u>uniquely identify each element in **a**</u>.

Multidimensional Arrays

- Initializing Two-Dimensional Arrays
  - Multidimensional arrays may be initialized by specifying **bracketed values for each row**. Following is an array with 3 rows and each row has 4 columns.



```c
int a[3][4] =
{
    {0, 1, 2, 3} , /* initializers for row indexed by 0 */
    {4, 5, 6, 7} , /* initializers for row indexed by 1 */
    {8, 9, 10, 11} /* initializers for row indexed by 2 */
};
```

  - The nested braces { } , which indicate the intended row, are optional.
  - The role of the comma (,) is the division between the columns.

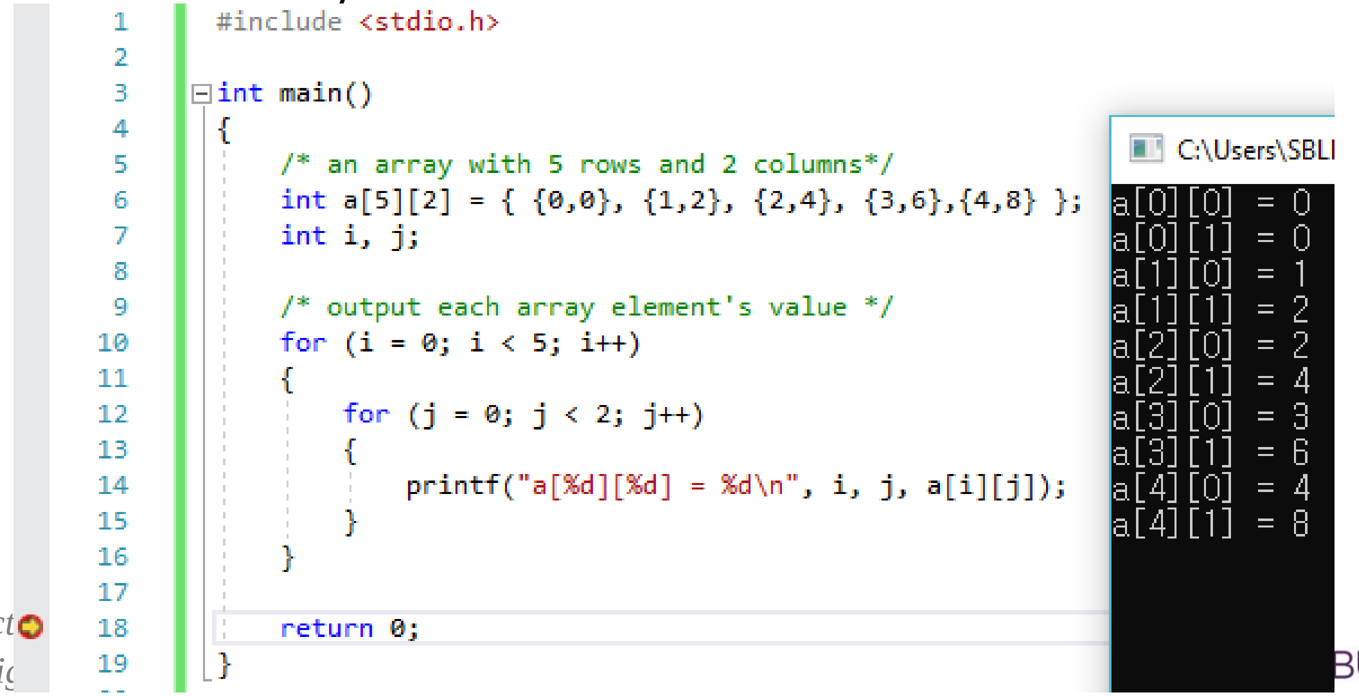
Multidimensional Arrays

▪ Accessing Two-Dimensional Array Elements

- An element in 2-dimensional array is accessed by using the subscripts, i.e., <u>row index and column index of the array</u>. For example:

```
int val = a[2][3];
```

- The variable *val* is assigned by the value of the **4th element** from the **3rd row** of the array **a**.
- Let us check below program where we have used nested loop to handle a two dimensional array:



```c
#include <stdio.h>

int main()
{
    /* an array with 5 rows and 2 columns*/
    int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8} };
    int i, j;

    /* output each array element's value */
    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 2; j++)
        {
            printf("a[%d][%d] = %d\n", i, j, a[i][j]);
        }
    }

    return 0;
}
```

```
a[0][0] = 0
a[0][1] = 0
a[1][0] = 1
a[1][1] = 2
a[2][0] = 2
a[2][1] = 4
a[3][0] = 3
a[3][1] = 6
a[4][0] = 4
a[4][1] = 8
```

Passing Arrays as Function Arguments

- Syntax
  - If you want to pass a **single-dimension array** as an **argument** in a function, you would have to declare **function formal parameter** in one of following three ways.
  - All three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received.
  - Similar way, you can pass a multi-dimensional array as formal parameters.

- Way-1
  - Formal parameters as a pointer as follows. You will study what is the pointer in the next chapter.

```
void myFunction(int *param)
{
.
.
.
}
```

Passing Arrays as Function Arguments

- Way-2
  - Formal parameters as a sized array as follows:

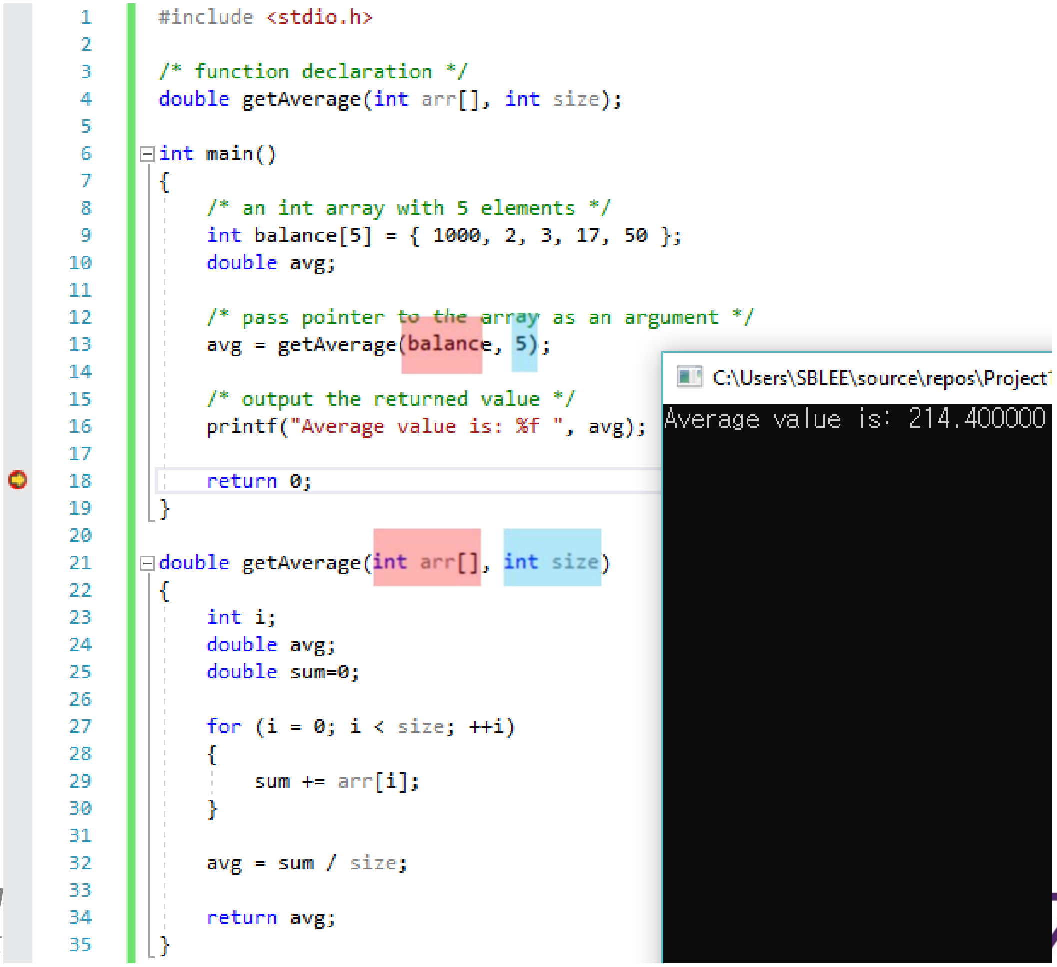
```
void myFunction(int param[10])
{
.
.
.
}
```

- Way-3
  - Formal parameters as an unsized array as follows:

```
void myFunction(int param[])
{
.
.
.
}
```

Passing Arrays as Function Arguments

- Example
  - Consider the following function, which will take an array as an argument along with another argument and based on the passed arguments, it will return average of the numbers passed through the array as follows:



```c
#include <stdio.h>

/* function declaration */
double getAverage(int arr[], int size);

int main()
{
    /* an int array with 5 elements */
    int balance[5] = { 1000, 2, 3, 17, 50 };
    double avg;

    /* pass pointer to the array as an argument */
    avg = getAverage(balance, 5);

    /* output the returned value */
    printf("Average value is: %f ", avg);

    return 0;
}

double getAverage(int arr[], int size)
{
    int i;
    double avg;
    double sum=0;

    for (i = 0; i < size; ++i)
    {
        sum += arr[i];
    }

    avg = sum / size;

    return avg;
}
```

C:\Users\SBLEE\source\repos\Project

```
Average value is: 214.400000
```

Pointer to an Array

- Syntax
  - It is most likely that you would **not understand** this chapter until you are through the chapter related to Pointers in C.

```
double balance[50];
```

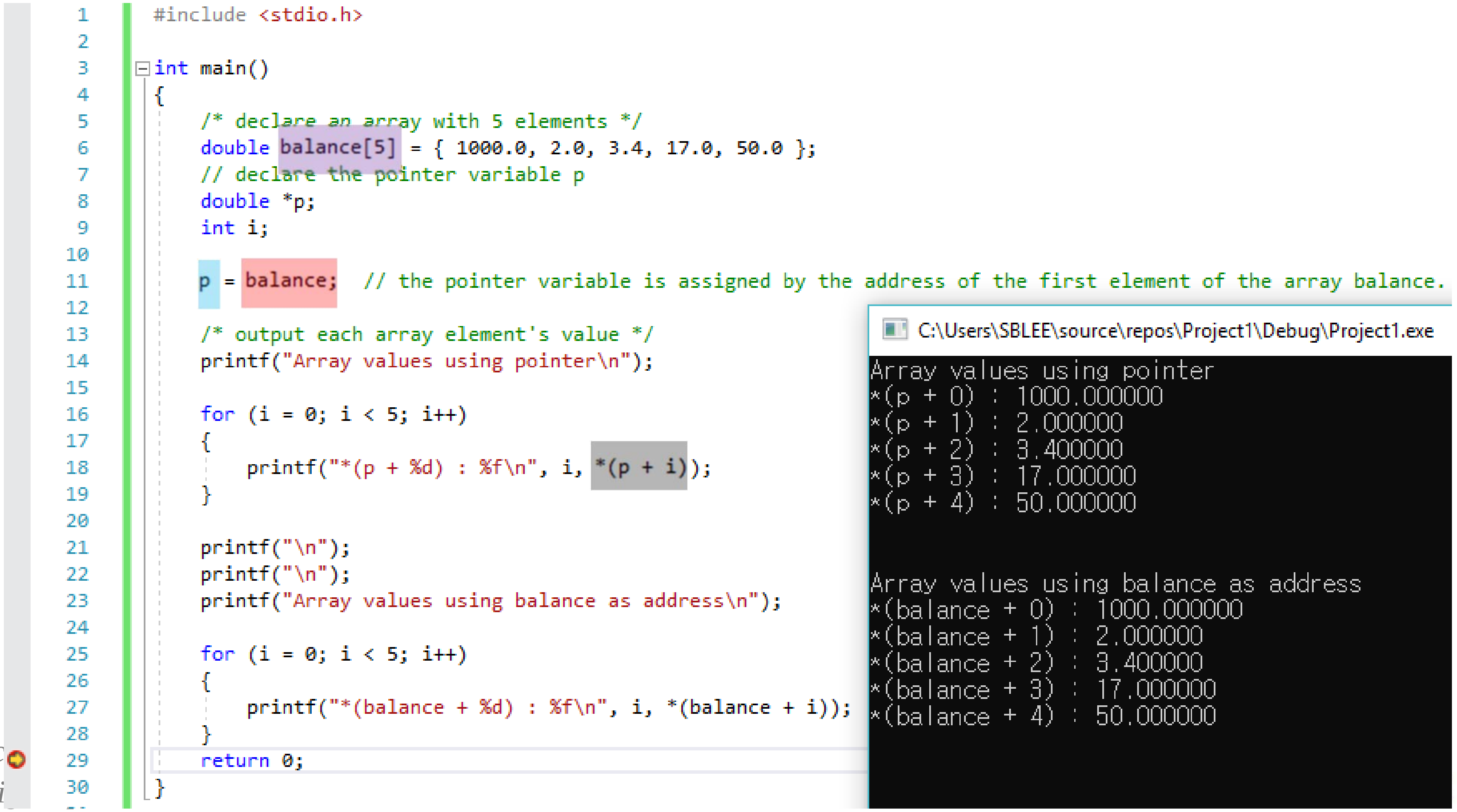
  - An **array name** is a **constant pointer** to the **first element of the array**.
  - So, **the array's name** *balance* is **a pointer** to **&balance[0]**, which is the **address (&)** of the **first element of the array balance (balance[0])**.
  - Thus, the following program fragment assigns ***p*** the **address of the first element of balance**:

```
double *p;
double balance[10];

p = balance;
```

  - **double *p;** means the **declaration** of the **pointer variable** (p) which saves the **address** of the variable or the array.
  - ***p** means the **value saved at the pointer variable** (p) **or** the **value saved at the address of pointer variable** (p).
  - Therefore, ***(balance + 0)** is a legitimate way of **accessing the data at balance[0]**.

Pointer to an Array

- Example
  - Once you store the address of first element of the array *balance* in p, you can access array elements using *p, *(p+1), *(p+2) and so on. Below is the example to show all the concepts discussed above:



```c
#include <stdio.h>

int main()
{
    /* declare an array with 5 elements */
    double balance[5] = { 1000.0, 2.0, 3.4, 17.0, 50.0 };
    // declare the pointer variable p
    double *p;
    int i;

    p = balance;  // the pointer variable is assigned by the address of the first element of the array balance.

    /* output each array element's value */
    printf("Array values using pointer\n");

    for (i = 0; i < 5; i++)
    {
        printf("*(p + %d) : %f\n", i, *(p + i));
    }

    printf("\n");
    printf("\n");
    printf("Array values using balance as address\n");

    for (i = 0; i < 5; i++)
    {
        printf("*(balance + %d) : %f\n", i, *(balance + i));
    }
    return 0;
}
```

C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe

```
Array values using pointer
*(p + 0) : 1000.000000
*(p + 1) : 2.000000
*(p + 2) : 3.400000
*(p + 3) : 17.000000
*(p + 4) : 50.000000


Array values using balance as address
*(balance + 0) : 1000.000000
*(balance + 1) : 2.000000
*(balance + 2) : 3.400000
*(balance + 3) : 17.000000
*(balance + 4) : 50.000000
```

Summary

Summary

- ✓ We considered the **Declaring Arrays, Initializing Arrays, Accessing Array Elements, and Arrays in Detail**.

Thank You