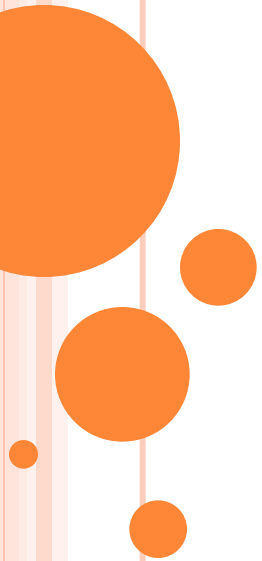
Chapter 6 Notes – **Arrays**

C How to Program, 6th Edition

Deitel & Deitel



SWE110 Lecture Note

Instructor: Zachi Baharav

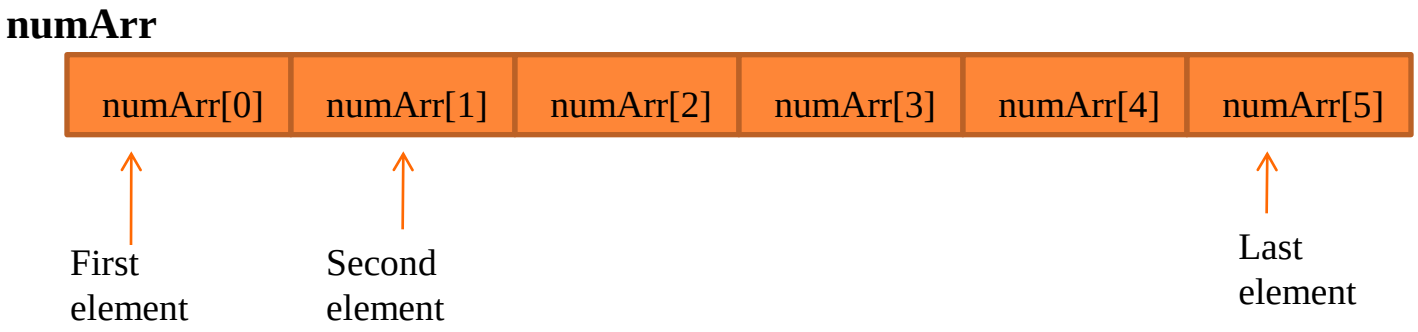
(Slides credit: Thien An Nguyen (An))

Objectives

- What is an array
- Array initialization, accessing an array, array manipulations, static and automatic arrays.
- In the lab: we will work on Assignment 4.
- Require reading: Chapter 6
- Tutoring: if you feel lost, send Joee a request for private tutor session(s). It’s free.

6.2 – Arrays

- Array is a data structure of multiple elements of the same data type.
- In memory, an array is a contiguous set of fixed size memory locations.
- In the next example, `numArr` is the name of an array of six elements.
- The elements of numArr can be accessed by indexes from 0 to 5.



6.3 – Defining Arrays

- To define an array, you need 3 things:
  - The array type
  - The array name.
  - The number of elements (the array size).

```
int numarr[5];     // declaration for numarr in the previous slide

char chararr[4];   // declare a characters array, chararr,
                   // that has 4 elements
```

Array Initializations

- There are two ways to initialize an array:
  - Declare an array and use an initialize list:

```
// declare and initialize numarr elements
int numarr[5] = {6, 6, 6, 6, 6};           // specified 5 elements

// declare and initialize chararr elements
char chararr[] = {'H', 'i', ' ', '!'};     // implied 4 elements
```

  - Go through each element of the array and assign it a value.

```
// the below initialization of numarr is equivalent to the above
int numarr[5];              // declare numarr, an array of 5 ints
int i;
for (i = 0; i < 5; i++){  // initialize numarr
  numarr[i] = 6;
}

// the below is an equivalent init
char chararr[4];            // declare chararr, an array of 4 chars
chararr[0] = 'H';           // init the first element
chararr[1] = 'i';
chararr[2] = ' ';
chararr[3] = '!';            // init the last element
```

6.4 – Array Examples

- The example declares, initializes and displays an array:

```
#include <stdio.h>

int main()
{
  int iarr[5];              // declare iarr, an array of 5 ints
  int idx;                  // declare a variable name idx

  // initialize the 1st three elements of iarr using loop control
  for (i = 0; i < 3; i++){// initialize iarr
    iarr[i] = i + 1;        // iarr: [1, 2, 3]
  }

  // initialize the 4th and 5th elements
  iarr[i] = 4;              // the 4th element value is 4
  iarr[4] = 5;              // the 5th element value is 5

  // use a loop to print out the array elements
  for (i = 0; i < 5; i++){
    printf("%d ", iarr[i]); // 1 2 3 4 5
  }
}
```

Quick Check1

- What is the output of the following code fragment?

```
#include <stdio.h>

int main()
{
  int arr[3] = {9, 6, 1}; // declare arr, an array of 3 ints
  int idx;                // declare a variable name idx

  for (idx = 0; idx < 3; idx++){
    printf("%d ", arr[idx]);// ?
  }

  // update the array
  arr[0] = 2;
  arr[2] = arr[0] + 3;

  // use a loop to print out the array elements
  for (idx = 0; idx < 3; idx++){
    printf("%d ", arr[idx]); // ?
  }
}
```

Quick Answer1

- What is the output of the following code fragment?

```
#include <stdio.h>

int main()
{
  int arr[3] = {9, 6, 1}; // declare arr, an array of 3 ints
  int idx;                // declare a variable name idx

  for (idx = 0; idx < 3; idx++){
    printf("%d ", arr[idx]);// 9 6 1
  }

  // update the array
  arr[0] = 2;             // update the 1st element
  arr[2] = arr[0] + 3;    // update the last element

  // use a loop to print out the array elements
  for (idx = 0; idx < 3; idx++){
    printf("%d ", arr[idx]); // 2 6 5
  }
}
```

Quick Check2

- What is the output of the following code fragment?

```
#include <stdio.h>

int main()
{
  char arr[3] = {'b', 'a', 'z'};  // declare arr, an array of 3 chars
  int idx;                        // declare a variable name idx

  for (idx = 0; idx < 3; idx++){
    printf("%c ", arr[idx]);      // ?
  }

  // update the array
  arr[1] = '1';
  arr[2] = 'a';

  // use a loop to print out the array elements
  for (idx = 0; idx < 3; idx++){
    printf("%c ", arr[idx]);      // ?
  }
}
```

Quick Answer2

- What is the output of the following code fragment?

```
#include <stdio.h>

int main()
{
  char arr[3] = {'b', 'a', 'z'};  // declare arr, an array of 3 chars
  int idx;                        // declare a variable name idx

  for (idx = 0; idx < 3; idx++){
    printf("%c ", arr[idx]);      // b a z
  }

  // update the array
  arr[1] = '1';                   // update the 2nd element
  arr[2] = 'a';                   // update the last element

  // use a loop to print out the array elements
  for (idx = 0; idx < 3; idx++){
    printf("%c ", arr[idx]);      // b 1 a
  }
}
```

Quick Check3

- Which of the following declarations is invalid?

```
int iarr[3];
char carr[];
int[] arr;
char []arr2;
```

- What happens if the array initializer's list is shorter than the array's size?

```
int iarray1[6] = {5, 5};

int iarray2[5] = {0};
```

- What happen if the array initializer's list is longer than the array's size?

```
int iarray1[1] = {5, 5};

int iarray2[] = {5, 5};
```

Quick Answer3

- Which of the following declarations is invalid?

```
int iarr[3];
char carr[];                // error! unkown size
int[] arr;                  // error! syntax error
char []arr2;                // error! Syntax error
```

- What happens if the array initializer's list is shorter than the array's size?

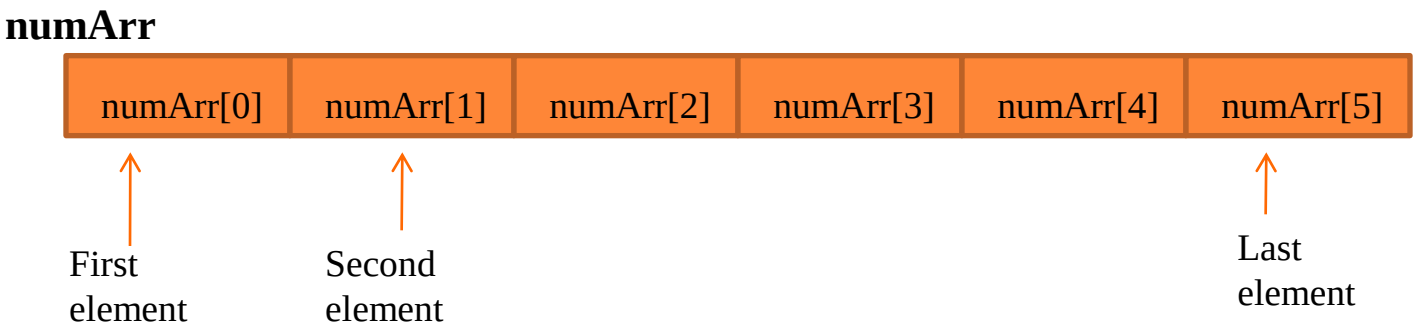
```
int iarray1[6] = {5, 5};  // If the initializer's list is shorter
                          // than the array size, the remaining
                          // elements are initialized to 0
int iarray2[5] = {0};     // 0, 0, 0, 0, 0, 0
```

- What happens if the array initializer's list is longer than the array's size?

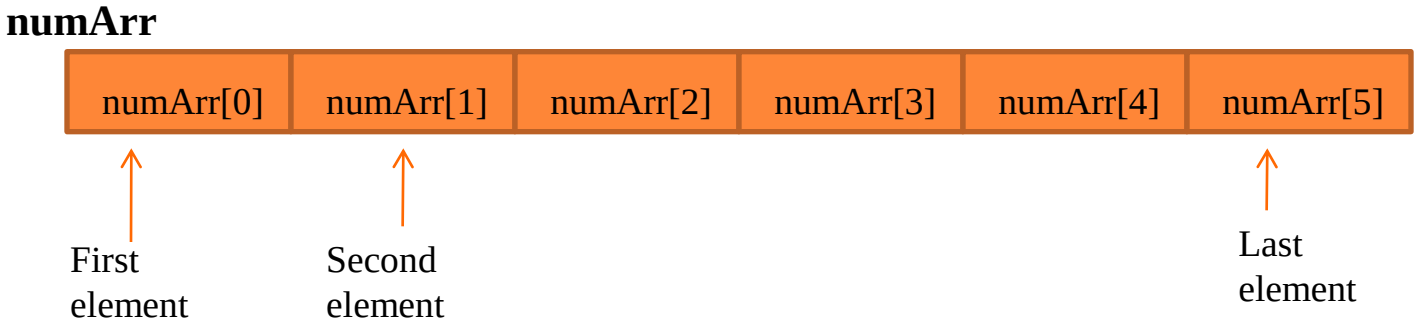
```
int iarray1[1] = {5, 5};          // error
int iarray2[] = {5, 5};           // ok. creates an array of 2 elements
```

What is Arrays for?

- Observe the figure one more time. What do you see?



What is Arrays for?



- Does it look like a collection of contiguous variables of the same data type?
- That's right. In our figure, instead of declaring 5 individual variables, we declared an array of 5 elements and use indexes to access each element.
- It is still ok to use 5 variables instead of an array. Would it still be ok to declare hundreds of variables in your program?
- Please don't. It's tedious, it's error prone, and it is complicated.

Using Character Arrays to Store and Manipulate Strings

- A string constant, written as "Hello" is an array of characters with a null character at the end of the string.
- In memory, it looks:

```
-------------------------
| H | e | l | l | o | \0 |
-------------------------
```

- The length of the above string is 6: 5 characters + 1 terminator.

Using Character Arrays to Store and Manipulate Strings

- Declaring and initialization variables for strings:

```
char arr1[6] = "Hello";   // explicitly declare arr1 with 6 chars
char arr2[] = "Hello";    // implicitly declare arr2 with 6 chars
char arr3[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; // another way
char arr4[] = {'H', 'e', 'l', 'l', 'o', '\0'};  // yet, another way

char arr5[];              // no way. It has to have a size
```

- Displaying strings:

```
printf("%s", arr1);       // formating string of characters - Hello
printf("%s", arr2);       // Hello
```

Using Character Arrays to Store and Manipulate Strings

- Since the null terminator character '\0' is used to indicate the end of a string, the following strings might give you an expected result:

```
int main(void)
{
  char arr1[] = {'H', 'e', 'l', 'l', '\0', 'o'};       // 6 elements
  char arr2[7] = {'H', 'e', 'l', '\0', ' ', ' ', ' '}; // 7 elements

  printf ("%s", arr1);    // Hell
  printf ("%s", arr2);    // Hel
}
```

- In memory:

```
--------------------     -------------------------
|H |e |l |l |\0 |o |     |H |e |l |l |\0|  |  |  |
--------------------     -------------------------
```

Using Character Arrays to Store and Manipulate Strings

- Initialize a string with user input:

```
int main(void)
{
  char arr[10];              // declare arr, an array length 20
  printf("Enter a string: ");
  scanf ("%s", arr);         // initialize arr with user input

  printf ("Output: %s", arr); // print user input to standard output
}
```

Sample Output:

```
Enter a string: Hello
Output: Hello
```

Using Character Arrays to Store and Manipulate Strings

- With the same program, suppose the user enters a string that is longer than 10 characters:

```
int main(void)
{
  char arr[10];              // declare arr, an array length 20
  printf("Enter a string: ");
  scanf ("%s", arr);         // initialize arr with user input

  printf ("Output: %s", arr); // print user input to standard output
}
```

Sample Output:

```
Enter a string: Hello World!
Output: Hello World!
```

- Note that the output looks like expected. However, the array was corrupted.

Using Character Arrays to Store and Manipulate Strings

- Traversing arrays:

```
#include <stdio.h>
#define LEN 10

int main(void)
{
  char arr[LEN];           // declare arr, an array length 20
  int length = 0,          // store the true length of the string
      idx;                 // use as index of the array

  printf("Enter a string: ");
  scanf ("%s", arr);       // initialize arr with user input

  while (arr[idx] != '\0'){ // loop until a '\0'
    length++;              // increment your counter
  }

  printf ("Length of your string %d:", length); // print input's length
}
```

Static , and Automatic Local Arrays

- In chapter 5, we walked through the example of a variable declared static inside a function (**`testFun`** in the next slide).
- In main, every time function testFun is called, count is incremented.
- In this section, we will look at example of a static local array as well as an automatic local array.

```c
#include <stdio.h>

int testFun()
{
  static int count = 0;   // declare and initialize a static variable
  count++;                // count is initialized at compile time

  return count;
}

int main()
{
  printf("%d ",testFun());          // 1
  printf("%d ",testFun());          // 2

  return 0;
}
```

Automatic Local Arrays

```
#include <stdio.h>

void testStaticArray()
{
  int arr[] = {9, 5, 3};            // declare and init. a local array

  int idx;
  for (idx = 0; idx < 3; idx++){    // modify the array elements
    arr[idx] = arr[idx] + 1;        // modify all of the array elements
    printf ("%d ", arr[idx]);
  }
}

int main()
{
  testStaticArray();       // 10 6 4
  testStaticArray();       // 10 6 4

  return 0;
}
```

Static Local Arrays

```
#include <stdio.h>

void testStaticArray()
{
  staic int arr[] = {9, 5, 3};     // declare and initialize a static
                                   // local array

  int idx;
  for (idx = 0; idx < 3; idx++){   // modify the array elements
    arr[idx] = arr[idx] + 1;       // modify all of the array elements
    printf ("%d ", arr[idx]);
  }
}

int main()
{
  testStaticArray();       // 10 6 4
  testStaticArray();       // 11 7 5

  return 0;
}
```

6.5 – Passing Arrays to Functions

- Array names are the addresses of the first element in the array, which is the address of the array.

```
int main()
{
  int iarray[] = {1, 2, 3, 4, 5}; // init with an initilizer list

  printf("%p", iarray);           // prints the address of iarray
  printf("%p", &iarray[0]);       // prints the address of iarray
  printf("%p", &iarray);          // prints the address of iarray

  return 0;
}
```

Quick Check1

- What is an array name evaluate to?

- Given an array name, would you be able to tell its size?

  For example:

```
// iarray is an array
iarray[0] = 6;   // update the ? of the array
iarray[10] = 99; // update the ? of the array
```

Quick Answer1

- What does an array name evaluate to?

  It evaluates to the address of the first element of the array, which is the address of the array.

- Given an array name, would you be able to tell its size?

  For example:

```
// iarray is an array
iarray[0] = 6;    // update the ? of the array
iarray[10] = 99; // update the ? of the array
```

  No. I can only tell there are at least 11 elements in the above array.

Quick Check2

- Which of the following is invalid?

```
int iarr[0];      // declare an integer array
char carr[-1];    // declare a character array
double darr[];    // declare a double array
```

- What is the first index of an array? How do I access it?

- What is the last index of an array? How do I access it?

Quick Answer2

- Which of the following is invalid?

```
int iarr[0];      // Invalid declaration. The size must be > 0
char carr[-1];    // Invalid declaration. The size must be > 0
double darr[];    // Invalid declaration. Must give a size.
```

- What is the first index of an array? How do I access it?

  First index is 0. Access the 1st element: `iarr[0]`

- What is the last index of an array? How do I access it?

  Last index is `the array's size - 1`.

  To access the last element: `iarr[size - 1]`

6.5 – Passing Arrays to Functions

- There are 3 ways for declaring a function input parameter type array. Each way has 3 parts:
  - Like declaring an unsized array

```
void functionx(int arrayname[], int size)
{
   ...              // code in the function
}
```

  - Like declaring a sized array

```
void functionx(int arrayname[6], int size)
{
   ...              // code in the function
}
```

  - Like declaring a pointer (see next chapter)

```
void functionx(int *arrayname, int size)
{
   ...              // code in the function
}
```

6.5 – Passing Arrays to Functions

- Example1 - passing an array into a function to print its contents:

```
void printArray(int arr[], int len) // passing array to printArray
{
  for (int i = 0; i < len; i++){
    printf("%d ", arr[i]);          // -11 2 43 55
  }
}

int main()
{
  int iarr[] = {-11, 2, 43, 55};   // declare iarr with an init. list

  printArray(iarr, 4);             // call printArray to print the array

  return 0;                        // success return code
}
```

6.5 – Passing Arrays to Functions

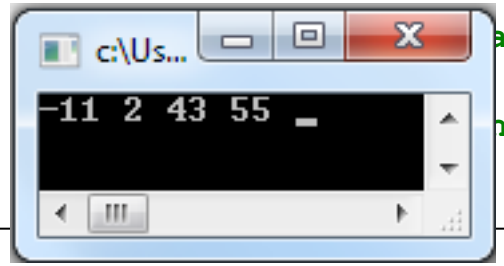
- Example1 - passing an array into a function to print its contents:

```
void printArray(int arr[], int len) // passing array to printArray
{
  for (int i = 0; i < len; i++){
    printf("%d ", arr[i]);          // -11 2 43 55
  }
}

int main()
{
  int iarr[] = {-11, 2, 43, 55};   // declare iarr with an init. list

  printArray(iarr, 4);                                  ay to print the array

  return 0;                                             n code
}
```



6.5 – Passing Arrays to Functions

- Example2: when the array size is included in the array subscript, the compiler will check to make sure it's > 0. Then ignores it.

```
void updateArray(float arr[3], int len) // passing array to updateArray
{
  for (int i = 0; i < len; i++){
    arr[i] = arr[i] + 4;              // add 4 to all elements
    printf("%f ", arr[i]);            // -7.0 6.0 47.0 59.0
  }
}

int main()
{
  float farr[] = {-11, 2, 43, 55}; // declare farr with an init. list
  updateArray(farr, 4);            // call updateArray to print the array
  printf("\n");
  for (int i = 0; i < 4; i++){
    printf("%f ", farr[i]);         // -7.0 6.0 47.0 59.0
  }

  return 0;                         // success return code
}
```

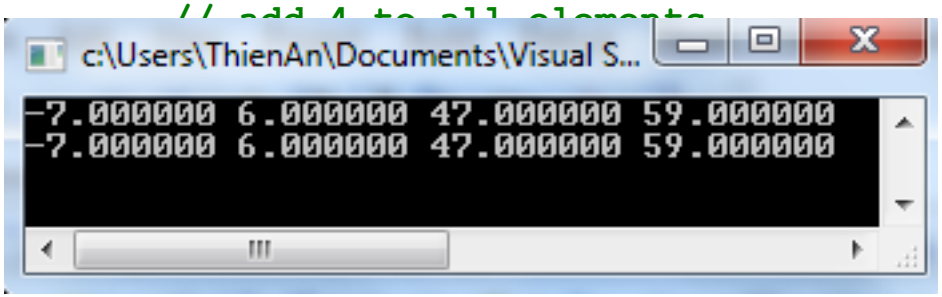
6.5 – Passing Arrays to Functions

- Example2: when the array size is included in the array subscription, the compiler will checks to make sure it's > 0. Then ignore it.

```
void updateArray(float arr[3], int len) // passing array to updateArray
{
  for (int i = 0; i < len; i++){
    arr[i] = arr[i] + 4;                // add 4 to all elements
    printf("%f ", arr[i]);
  }
}

int main()
{
  float farr[] = {-11, 2, 43, 55}; // declare farr with an init. list
  updateArray(farr, 4);            // call updateArray to print the array
  printf("\n");
  for (int i = 0; i < 4; i++){
    printf("%f ", farr[i]);        // -15.0 6.0 47.0 59.0
  }

  return 0;                        // success return code
}
```



6.5 – Define a Constant

- When working with small programs that contain only couple lines of code, you might not see the importance of replacing those fixed array length with a constant.
- Imagine in the industrial environment where a product is the program with thousand lines of code. Hard coding the array's length initially and changing the length later is consider bad programming practice because it is almost impossible to change the hard coded array length without introducing runtime errors.

```
#define LEN <the constant>

#define LEN 4
```

6.5 – Bad Practice Example

- Example3: In the example below, originally, iarr was declared as an array of 4 elements. It was changed to 5 later. However, the caller forgot to change its calling input parameter.

```
void printArray(int arr[], int len) // passing array to updateArray
{
  for (int i = 0; i < len; i++){
    printf("%d ", arr[i]);          //  -11 2 43. It should be -11 2 43 0
  }
}

int main()
{
  int iarr[34] = {-11, 2, 43};      // declare farr with an init. list
  printArray(iarr, 3);              // call printArray to print the array

  return 0;                         // success return code
}
```

6.5 – Passing Arrays to Functions

- Example4: Define a constant instead of hard code the array length

```
#define LEN 4

void printArray(int arr[], int len) // passing array to updateArray
{
  for (int i = 0; i < len; i++){
    printf("%d ", arr[i]);          //  -11 2 43 0 <= correct array size
  }
}

int main()
{
  int iarr[LEN] = {-11, 2, 43};    // declare iarr with an init. list
  printArray(iarr, LEN);           // call printArray to print the array

  return 0;                        // success return code
}
```

6.5 – const array

- There are cases where you want to protect the content of the array from accidental modification, the qualifier const is used to flag an error whenever it sees an attempt to modify the array.

```
#define LEN 3

void printArray(const int arr[], int len)
{
  for (int i = 0; i < len; i++){
    printf("%d ", arr[i]);          //  -11 2 43
  }
}

int main()
{
  int iarr[LEN] = {-11, 2, 43};     // declare farr with an init. list
  printArray(iarr, LEN);            // call printArray to print the array

  return 0;                         // success return code
}
```

6.6 – Sorting Array with Bubble Sort

- Let's look at Figure 6.15 on textbook page 217.

6.8 – Searching Array

- The simplest way to search an array for a key is to go through the array elements and compare the elements with the key.
- If an array element matches the key, there is a hit.
- If the loop exits and the key is not found, the key does not exist in the array.

```
#define LEN 5

int findKey(int key, const int arr[], int len)
{
  int i = 0;                          // use to traverse the array
  for (i = 0; i < len; i++){
    if (arr[i] == key){
      return i;                       // return the position, indicate a hit
    }
  }

   return -1;                         // indicate key not found
}

int main()
{
  int arr[LEN] = {5, 3, 10, 99, 1}; // declare arr with an init. List
  int result;                         // store the result

  result = findKey(1, arr, LEN);   // call findKey to search for 1
  if (result >= 0){                   // the position of the key is not neg.
     printf("Key found!");
  } else{
     printf ("Key Not found!");
  }

  return 0;                           // success return code
}
```

6.9 – Multiple Subscripted Array

- Two-dimentional array example:

```
#define LEN 2
#define SIZE 3

int main()
{
  // declare a 2 dimensions array with initial values
  int arr[LEN][SIZE] = {{2, 1, 0}, {1, 3, 4}};
  int i, j;                          // use as indexes

  // prints the elements in the two dimensional array
  for (int i = 0; i < LEN; i++){
    for (int j = 0; j < SIZE; j++){
      printf("%d ", arr[i][j]);     // 2 1 0 1 3 4
    }
  }

  return 0;                          // success return code
}
```

6.9 – Multiple Subscripted Array

- Another way to initialize a two-dimentional array example:

```
#define LEN 2
#define SIZE 3

int main()
{
  // declare a 2 dimensions array with initial values
  int arr[LEN][SIZE] = {2, 1, 0, 1, 3, 4};
  int i, j;                          // use as indexes

  // prints the elements in the two dimensional array
  for (int i = 0; i < LEN; i++){
    for (int j = 0; j < SIZE; j++){
      printf("%d ", arr[i][j]);     // 2 1 0 1 3 4
    }
  }

  return 0;                          // success return code
}
```

6.9 – Passing 2-dimensional Array

- Passing a 2-dimensional array into a function:

```
#define LEN 2
#define SIZE 3

void printArray(const int arr[][SIZE], int len)
{
  int i, j;                          // use as index
  for (i = 0; i < len; i++){
    for (j = 0; j < SIZE; j++){
      printf("%d ", arr[i][j]);      // 2 1 0 1 3 4
    }
  }
}

int main()
{
  // declare a 2 dimensions array with initial values
  int arr[LEN][SIZE] = {{2, 1, 0}, {1, 3, 4}};
  printArray(arr, LEN);              // call printArray to print the array

  return 0;                          // success return code
}
```