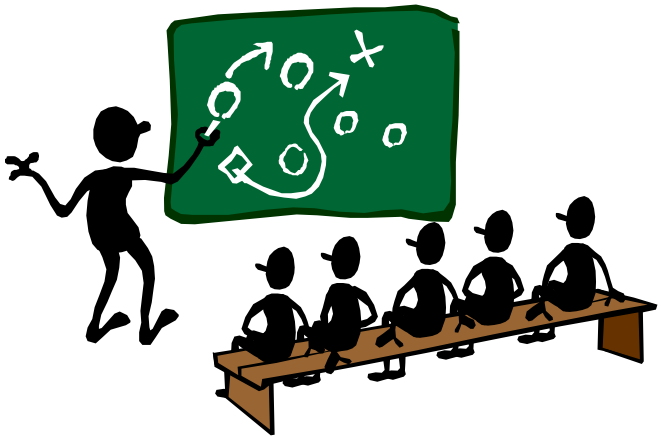
C++ Programming Language
Chapter 5 Arrays



Juinn-Dar Huang

Associate Professor

jdhuang@mail.nctu.edu.tw

February 2011

Learning Objectives

- Introduction to arrays
  - declaring/defining and referencing arrays
  - for-loops and arrays
  - arrays in memory
- Arrays in functions
  - arrays as function parameters and arguments
- Programming with arrays
  - searching and sorting in arrays
- Multidimensional arrays

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Introduction to Arrays

- Array
  - a collection of data of same type
- Array is an aggregate data type
  - int, float, double, char are simple data types
- Commonly used to store a set of related items
  - e.g., test scores, names, …
  - avoid declaring multiple simple variables
  - manipulate a set of data as one entity

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Defining Arrays

- Define an array ➔ type array_name[size];
  - e.g., int score[5];
    define an array of 5 variables of int named score
  - size must be an expression evaluating to integral CONSTANT
  - size must be known at compile time
    - statically allocated

```
int a_arr[10+20];          // ok
const int K = 100;
int b_arr[K];              // ok, b_arr[100];
int n = 100;
int c_arr[n];              // error, n is not a constant
```

- Define an array ➔ allocate a block of memory

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Accessing Arrays

- Access an element of array
  - e.g., cout << score[3];
  - value in brackets is called an index or a subscript
- Indices for an array of size n are ALWAYS from 0 to n – 1
  - e.g., int arr[100]; ➔ arr[0], arr[1], …, arr[98], arr[99]
  - each of arr[0], arr[1], …, arr[98], arr[99] is a variable of int
- Note two uses of brackets:
  - in declaration/definition, specifies **SIZE** of array
    - e.g., int arr[**100**];
  - anywhere else, specifies an **indexed** element
    - e.g. arr[**8**] = 59;
- Index need not be literal/constant;
  it can be any expression evaluating to an integral value

```
int score[5], n = 2;
score[n+1] = 99;      // score[3] = 99;
```

Array Example (1/2)

Display 5.1 Program Using an Array

```
1   //Reads in five scores and shows how much each
2   //score differs from the highest score.
3   #include <iostream>
4   using namespace std;
5   int main( )
6   {
7       int i, score[5], max;
8       cout << "Enter 5 scores:\n";
9       cin >> score[0];
10      max = score[0];
11      for (i = 1; i < 5; i++)
12      {
13          cin >> score[i];
14          if (score[i] > max)
15              max = score[i];
16          //max is the largest of the values score[0],..., score[i].
17      }
```

Array Example (2/2)

```
18        cout << "The highest score is " << max << endl
19             << "The scores and their\n"
20             << "differences from the highest are:\n";
21        for (i = 0; i < 5; i++)
22            cout << score[i] << " off by "
23                 << (max - score[i]) << endl;
24        return 0;
25    }
```

SAMPLE DIALOGUE

Enter 5 scores:
5 9 2 10 6
The highest score is 10
The scores and their
differences from the highest are:
5 off by 5
9 off by 1
2 off by 8
10 off by 0
6 off by 4

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

for-loops with Arrays

- Natural counting loop
  - works well for walking through elements of an array
- Example:

```
for (idx = 0; idx < 5; ++idx) {
      cout << score[idx] << “off by ”
                 << max – score[idx] << endl;
}
```

  - loop control variable (idx) counts from 0 to 4

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Major Array Pitfalls

- Array index always starts from zero!
  - sometimes good, sometimes not good …
  - however, you have no choice ☹
- C++ will let you go beyond range

```
int arr[10]; a = -3, b = 15;
arr[a] = 23;        // no compilation error
arr[b] = 24;        // no compilation error
```

  - unpredictable results; usually a disaster
  - compiler will not detect these errors!
    - price for runtime range checking is simply too high
- Up to programmer to stay in range
  - it is **YOUR** responsibility!

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Example: Out-of-Range

- Indices range from 0 to (array_size – 1)
  - example:

```
double temp[24];        // array size of 24
// they are indexed as: temp[0], temp[1], …, temp[23]
```

  - common mistake:

```
temp [24] = 5;
// index 24 is out of range!
// no warning, possibly disastrous results
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Named Constant as Array Size (1/2)

- Use named constant for array size
- Example:

```
const int NUMBER_OF_STUDENTS = 5;
int score[NUMBER_OF_STUDENTS];
```

- Improves readability and maintainability

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Named Constant as Array Size (2/2)

- Use everywhere size of array is needed
  - in for-loop for traversal:

```
for (idx = 0; idx < NUMBER_OF_STUDENTS; ++idx) {
    // manipulate array
}
```

  - in calculations involving size:
    lastIndex = (NUMBER_OF_STUDENTS – 1);

  - when passing array to functions (later)

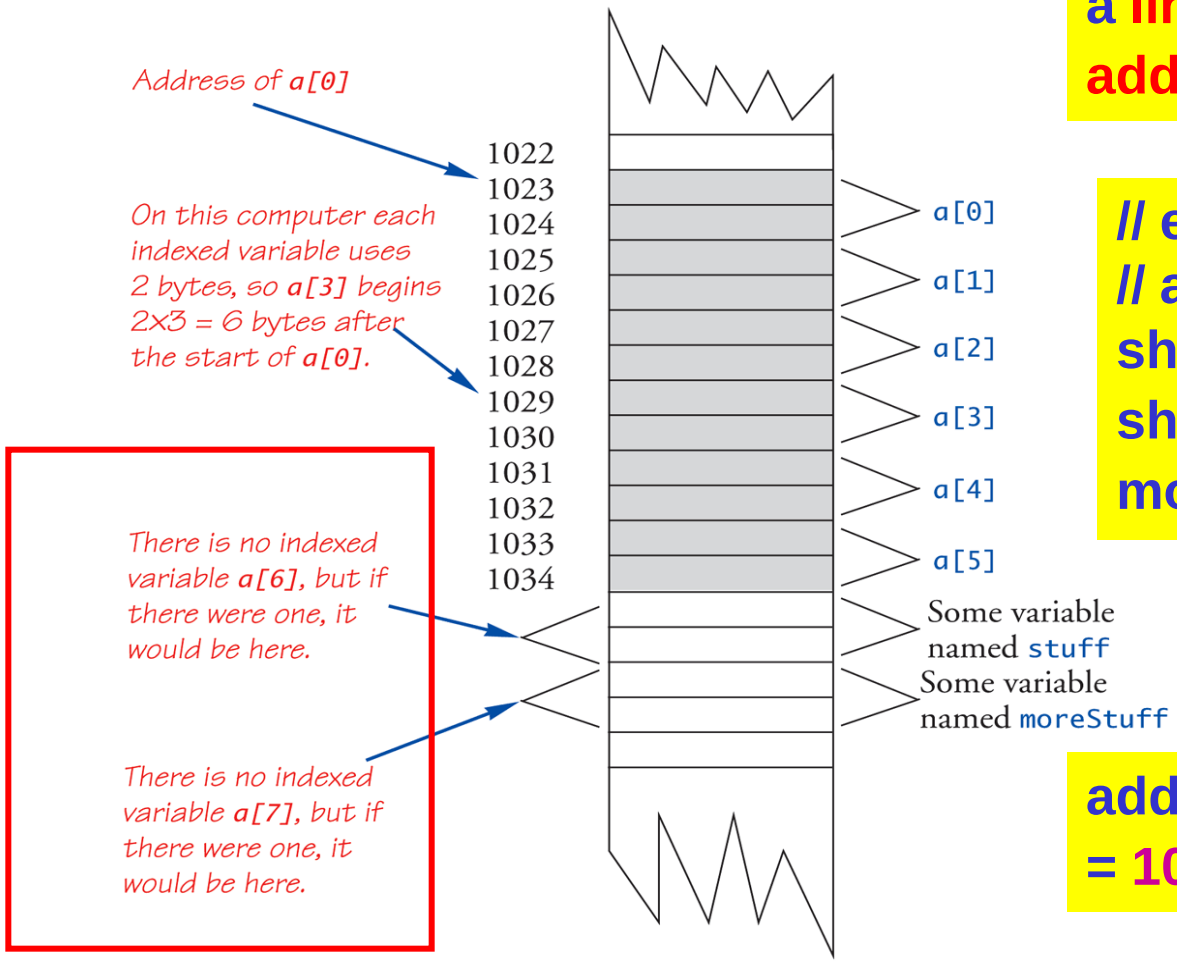
- If size changes ➔ requires only ONE change in program!

Arrays in Memory (1/2)

- Define a simple variable:
  - allocate memory space for that variable
  - variable name actually represents the address
- Define an array
  - allocate a block of memory for all elements of array
  - array name is actually a **pointer** to the **first element** of array
- An array is **ALWAYS** allocated contiguously
  - i.e., elements a[i] and a[i+1] are sequential in memory
  - enable fast and easy address calculations
  - simple addition from array beginning (index 0)

Arrays in Memory (2/2)

Display 5.2 **An Array in Memory**



Memory is regarded as a linear order of addressable bytes in C/C++

```
// example
// a short int takes 2 bytes
short a[6]; // start @ 1023
short stuff; // start @ 1035
moreStuff; // start @ 1037
```

address of a[4]
= 1023 + 4 * 2 = 1031

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Initializing Arrays (1/2)

- A local array is uninitialized by default

```
int a[5];    // a[0] = ?, a[1] = ? …, a[4] = ?
```

- An array can be initialized

```
int score[3] = { 2, 12, 1};
```

  - which is equivalent to following:

```
int score[3];
score[0] = 2;
score[1] = 12;
score[2] = 1;
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Initializing Arrays (2/2)

- If fewer values than size
  - fills from the beginning
  - fills the remaining elements with 0 of array base type
  - e.g., int a[6] = {10, 11, 12, 13}; // a[4] = 0, a[5] = 0
- If more values than size
  - compilation error
  - e.g., double b[3] = { 6.0, 6.5, 7.0, 8.0}; // error
- If size is unspecified
  - size is automatically determined based on number of initialization values
  - example:
    int b[] = {5, 12, 11}; // equivalent ➔ int b[3] = {5, 12, 11};
    int c[]; // error, unknown size

Arrays in Functions

- As argument to function
  - indexed variable
    - an individual element of an array can be a function argument
  - entire array
    - all elements can be passed as one entity
- As return value from function
  - can be done ➔ discuss in Chapter 10

Indexed Variables as Arguments

- Indexed variable handled same as simple variable of base type
  - example:

```
void myFunc(int par1);

void f() {
   int i = 10, a[5] = {1, 2, 3, 4, 5};
   myFunc(i);              // ok, i is an int
   myFunc(a[3]);           // ok, a[3] is an int, too
   myFunc(a[i – 8]);       // ok, a[2] is an int, too
   myFunc(a[i – 4]);       // no compilation error,

}                          // but a[6] does not exist}
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Arrays as Parameters/Arguments

- In function declaration and definition

  void f1(char arr[ ]); // just use empty brackets

  void f2(char arr[10]); // **still ok**, compiler simply ignores what's inside [ ]

- In function call
  - use array name as actual argument

```
void f() {
    char table[1000];
    f1(table);          // ok
    f2(table);          // still ok
}
```

- Need another parameter for array size if required

  void f3(char arr[ ], int size);

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Function with Array Parameter (1/2)

Display 5.3 Function with an Array Parameter

SAMPLE DIALOGUE

FUNCTION DECLARATION

```
void fillUp(int a[], int size);
//Precondition: size is the declared size of the array a.
//The user will type in size integers.
//Postcondition: The array a is filled with size integers
//from the keyboard.
```

SAMPLE DIALOGUE

FUNCTION DEFINITION

```
void fillUp(int a[], int size)
{
    cout << "Enter " << size << " numbers:\n";
    for (int i = 0; i < size; i++)
        cin >> a[i];
    cout << "The last array index used is " << (size - 1) << endl;
}
```

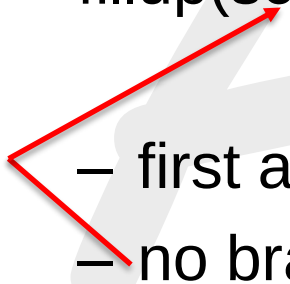
Note those empty brackets

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Function with Array Parameter (2/2)

- Inside function definition

```
const int NUM_SCORES = 5;
int score[NUM_SCORES];
fillup(score, NUM_SCORES);
```



  - first argument is array name
  - no brackets in array argument
  - second argument is integer value for array size

Array Name as Argument: How?

- What is really passed?

```
void func(int a[]);
void f() {
    int arr[10];
    func(arr);
}
```

- Just the beginning address of array is passed
  - that is also the address of the first element
  - any element can then be **addressed** (see Page 13 again!)
- Actually, pass-by-pointer-value is implicitly used
  - array name is actually a **pointer** to the **first element (P. 12)**
- Size of array is not passed!

Array Name as Argument

- It may seem strange
  - no brackets [ ] in array argument
  - must pass size separately if required
- One nice property:
  - can use same function to process arrays of variant sizes
  - example:

```
int score1[5], score2[10];
fillUp(score1, 5);
fillUp(score2, 10);
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

const Parameter Modifier

- Recall: array argument actually passes the address of the first element
  - pass-by-pointer-value
- Function can then modify array!
  - sometimes desirable, sometimes not!
- Want to protect array from modification?
  - use const modifier!
  - tell compiler modifications to array are **NOT** allowed!

```
void func(const int table[]);
// contents of table cannot be modified in func
```

Functions that Return an Array

- Functions cannot return arrays same way as simple types are returned
- Require use of pointers
- Discuss later in Chapter 10

Programming with Arrays

- Searching a specific element in an array
- Sorting an entire array

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Searching an Array (1/2)

Display 5.6 Searching an Array

```
32          cout << "Search again?(y/n followed by Return): ";
33          cin >> ans;
34      } while ((ans != 'n') && (ans != 'N'));
35      cout << "End of program.\n";
36      return 0;
37  }

38  void fillArray(int a[], int size, int& numberUsed)
39  <The rest of the definition of fillArray is given in Display 5.5>
40  int search(const int a[], int numberUsed, int target)
41  {
42      int index = 0;
43      bool found = false;
44      while ((!found) && (index < numberUsed))
45      if (target == a[index])
46          found = true;
47      else
48          index++;
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Searching an Array (2/2)

```
49        if (found)
50            return index;
51        else
52            return -1;
53    }
```

SAMPLE DIALOGUE

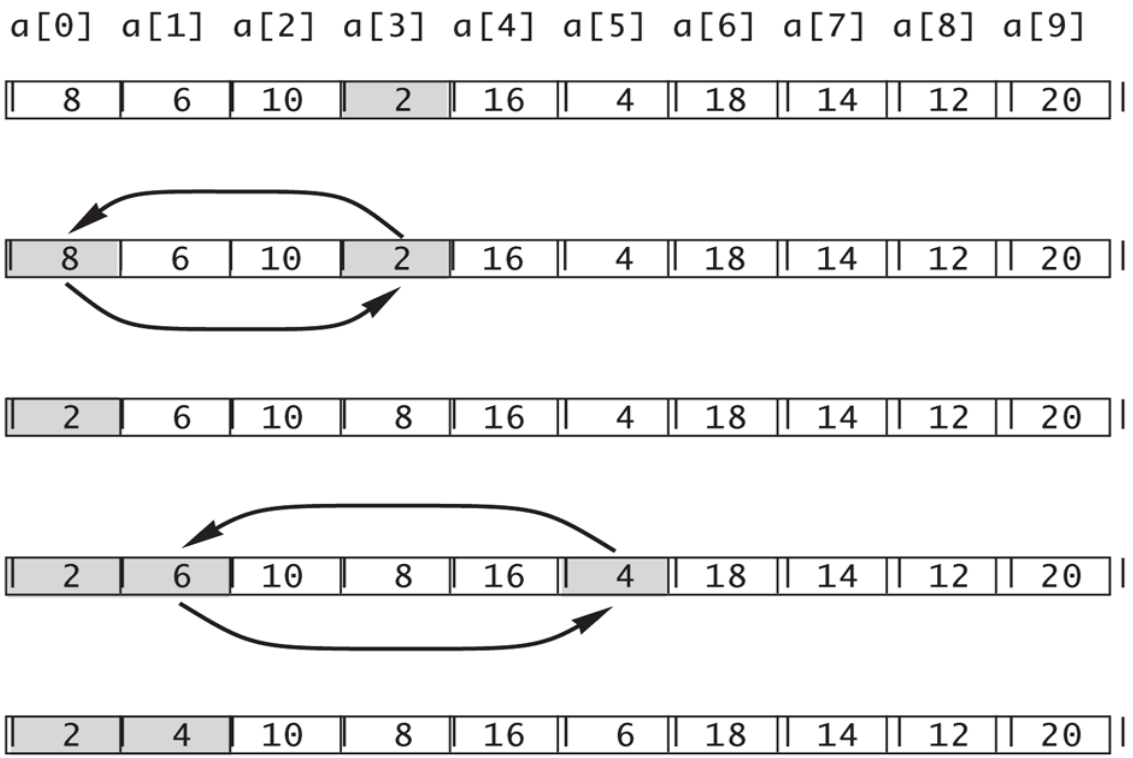
```
Enter up to 20 nonnegative whole numbers.
Mark the end of the list with a negative number.
10 20 30 40 50 60 70 80 -1
Enter a number to search for: 10
10 is stored in array position 0
(Remember: The first position is 0.)
Search again?(y/n followed by Return): y
Enter a number to search for: 40
40 is stored in array position 3
(Remember: The first position is 0.)
Search again?(y/n followed by Return): y
Enter a number to search for: 42
42 is not on the list.
Search again?(y/n followed by Return): n
End of program.
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Sorting an Array: Selection Short (1/5)

- Selection Sort Algorithm

Display 5.7 Selection Sort



Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Sorting an Array: Selection Short (2/5)

Display 5.8 Sorting an Array

```
1   //Tests the procedure sort.
2   #include <iostream>
3   using namespace std;

4   void fillArray(int a[], int size, int& numberUsed);
5   //Precondition: size is the declared size of the array a.
6   //Postcondition: numberUsed is the number of values stored in a.
7   //a[0] through a[numberUsed - 1] have been filled with
8   //nonnegative integers read from the keyboard.
9   void sort(int a[], int numberUsed);
10  //Precondition: numberUsed <= declared size of the array a.
```

(continued)

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Sorting an Array: Selection Short (3/5)

Display 5.8 Sorting an Array

```
11  //The array elements a[0] through a[numberUsed - 1] have values.
12  //Postcondition: The values of a[0] through a[numberUsed - 1] have
13  //been rearranged so that a[0] <= a[1] <= ... <= a[numberUsed - 1].

14  void swapValues(int& v1, int& v2);
15  //Interchanges the values of v1 and v2.

16  int indexOfSmallest(const int a[], int startIndex, int numberUsed);
17  //Precondition: 0 <= startIndex < numberUsed. Reference array elements
18  //have values. Returns the index i such that a[i] is the smallest of the
19  //values a[startIndex], a[startIndex + 1], ..., a[numberUsed - 1].

20  int main( )
21  {
22      cout << "This program sorts numbers from lowest to highest.\n";

23      int sampleArray[10], numberUsed;
24      fillArray(sampleArray, 10, numberUsed);
25      sort(sampleArray, numberUsed);

26      cout << "In sorted order the numbers are:\n";
27      for (int index = 0; index < numberUsed; index++)
28          cout << sampleArray[index] << " ";
29      cout << endl;

30      return 0;
31  }
```

Sorting an Array: Selection Short (4/5)

```
32  void fillArray(int a[], int size, int& numberUsed)
33            <The rest of the definition of fillArray is given in Display 5.5.>

34  void sort(int a[], int numberUsed)
35  {
36      int indexOfNextSmallest;
37      for (int index = 0; index < numberUsed - 1; index++)
38      {//Place the correct value in a[index]:
39          indexOfNextSmallest =
40                        indexOfSmallest(a, index, numberUsed);
41          swapValues(a[index], a[indexOfNextSmallest]);
42          //a[0] <= a[1] <=...<= a[index] are the smallest of the original array
43          //elements. The rest of the elements are in the remaining positions.
44      }
45  }

46  void swapValues(int& v1, int& v2)
47  {
48      int temp;
49      temp = v1;
50      v1 = v2;
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Sorting an Array: Selection Short (5/5)

Display 5.8 Sorting an Array

```
51        v2 = temp;
52    }
53
54    int indexOfSmallest(const int a[], int startIndex, int numberUsed)
55    {
56        int min = a[startIndex],
57            indexOfMin = startIndex;
58        for (int index = startIndex + 1; index < numberUsed; index++)
59            if (a[index] < min)
60            {
61                min = a[index];
62                indexOfMin = index;
63                //min is the smallest of a[startIndex] through a[index]
64            }
65        return indexOfMin;
66    }
```

More sorting algorithms in Data Structure !

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Multidimensional Arrays

- Arrays with more than one index

  e.g., char page[30][100];

  – two indices: an array of arrays

  – visualize as:
  page[0][0], page[0][1], …, page[0][99]
  page[1][0], page[1][1], …, page[1][99]
  …
  page[29][0], page[29][1], …, page[29][99]

  – total 30*100 = 3000 elements in this 2D array

- C++ allows any number of indices

Multidimensional Arrays in Functions (1/2)

- In function declaration and definition
  - MUST specify sizes for ALL dimensions except for the first one

    void f1(char arr[ ][6][7][8]); // a 4-dimentional array

    void f2(char arr[9][6][7][8]); // **still ok**, compiler ignores what's inside [ ]
- In function call
  - use array name as actual argument; same way as for 1D array

```
void f() {
    char table1[5][6][7][8];
    char table2[5][5][7][8];
    f1(table1);          // ok
    f2(table1);          // still ok
    f1(table2);          // compilation error, array size not matched!
}
```

??? WHY ???

??? WHY ???

Multidimensional Arrays in Functions (2/2)

- Need another parameter for size of the first dimension if required

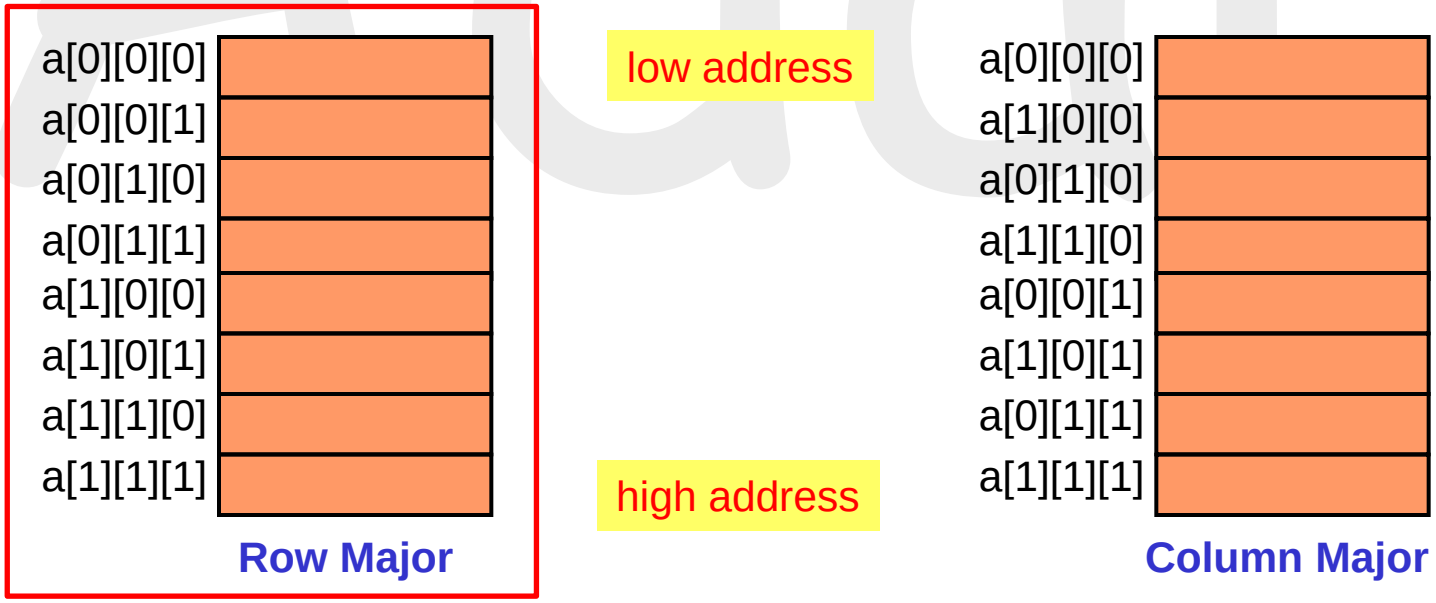
```
void f3(char arr[][6][7][8], int first_size);
```

- Example:

```
void DisplayPage(const char p[][100], int sizeDimension1)
{
        for (int index1=0; index1<sizeDimension1; index1++)
        {
                for (int index2=0; index2 < 100; index2++)
                        cout << p[index1][index2];
                cout << endl;
        }
}
```

Address Calculation for XD Arrays (1/2)

- How to put a multidimensional array into linearly addressed memory?
  - **row major (used in C/C++, …)**
  - column major (used in age-old FORTRAN, …)
- An array, a[2][2][2], has 8 elements
  - a[0][0][0], a[0][0][1], a[0][1][0], a[0][1][1], a[1][0][0], a[1][0][1], a[1][1][0], a[1][1][1]



Address Calculation for XD Arrays (2/2)

- For an array $a[u_1][u_2]\ldots[u_n]$ starting at the address A, what is the address of $a[i_1][i_2]\ldots[i_n]$?

  int a[6][7][8]; // assume starting from address 1000, sizeof(int) = 4

  a[1][2][3] = 10; // what is the address of a[1][2][3]?

  address = 1000 + ((1 * 7 * 8) + (2 * 8) + 3) * 4 = 1300

$$\begin{aligned} address = A &+ i_1u_2u_3\ldots u_n \\ &+ i_2u_3u_4\ldots u_n \\ &\ldots \\ &+ i_{n-1}u_n \\ &+ i_n \end{aligned}$$

  - during address calculation, sizes of all dimensions are required except for the first one! (Yep, that's why!)

Summary (1/2)

- Array is a collection of data of same type
- Indexed variables of array used just like any other simple variables
- for-loop: a natural way to traverse arrays
- Programmer is responsible for staying in bounds of array
- Array initializations
- Array in functions (declaration/definition and call)
- Array parameter is weird
  - actually, an implicit call-by-pointer-value

Summary (2/2)

- Constant array parameters
  - prevent modification of array contents by functions
- Array elements stored sequentially in memory
  - contiguous portion of memory
  - only address of the first element is passed to functions
- Multidimensional array
  - an array of arrays
  - address calculation for an individual element

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw