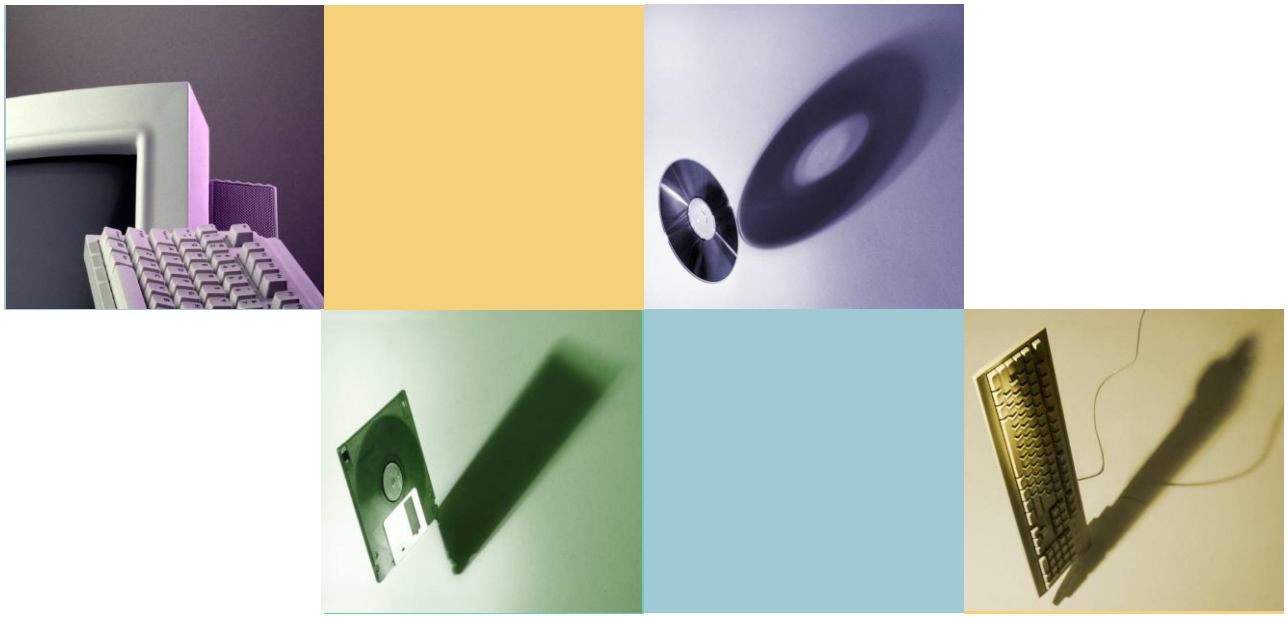
Object-Oriented Programming



Chuan-Kang Ting

Dept of Computer Science and Information Engineering
National Chung Cheng University

Chapter 5

Arrays



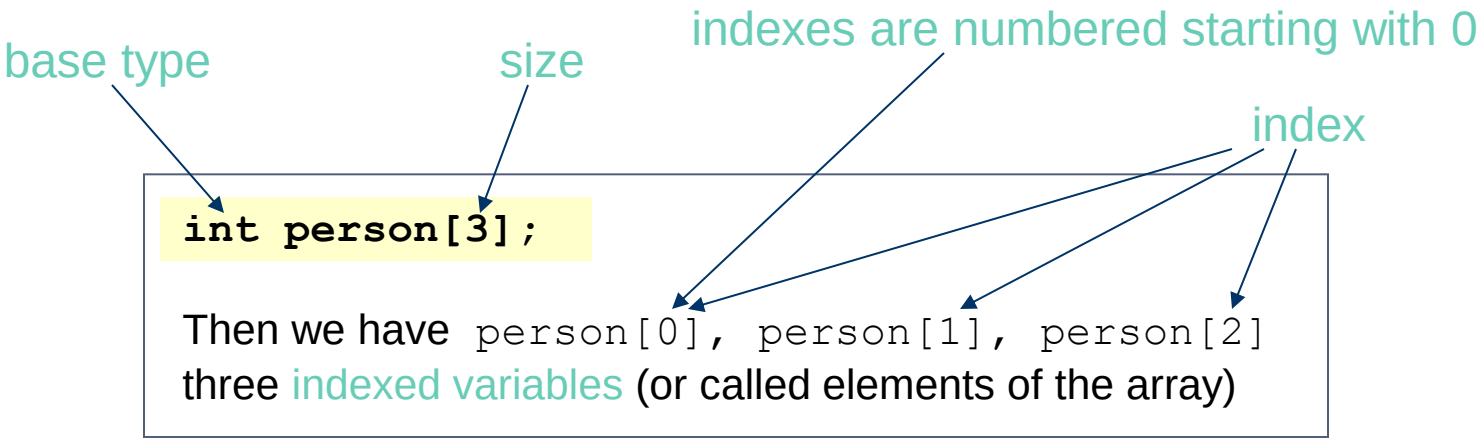
Outlines

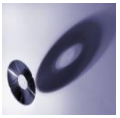
- **Introduction**
- **Arrays in Functions**
- **Programming with Arrays**
- **Multidimensional Arrays**



Introduction (1)

- **What is an array?**
  - An array is a collection of data of the same type
- **Terminology**



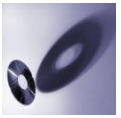

Introduction (2)

- **Usage**
  - A powerful storage mechanism
  - Use a defined constant for the size of an array
    - Improves readability, versatility, and maintainability

```
const int NUM_STUDENTS = 50;

int i, score[NUM_STUDENTS], max;
```

    - However, you CANNOT use a variable for the array size

```
int numStudents = 50;

int score[numStudents];        //Error
```

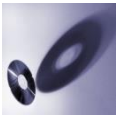




Introduction (3)

- **Arrays in Memory**
  - When declaring an array, e.g. `int a[6]`, the computer reserves memory to hold 6 variables of type `int`
  - The computer then remembers the address of `a[0]` **except** the address of any other indexed variables
  - e.g. To get the address of `a[3]`, the computer
    - Starts with the address of `a[0]`
    - Adds the needed bytes

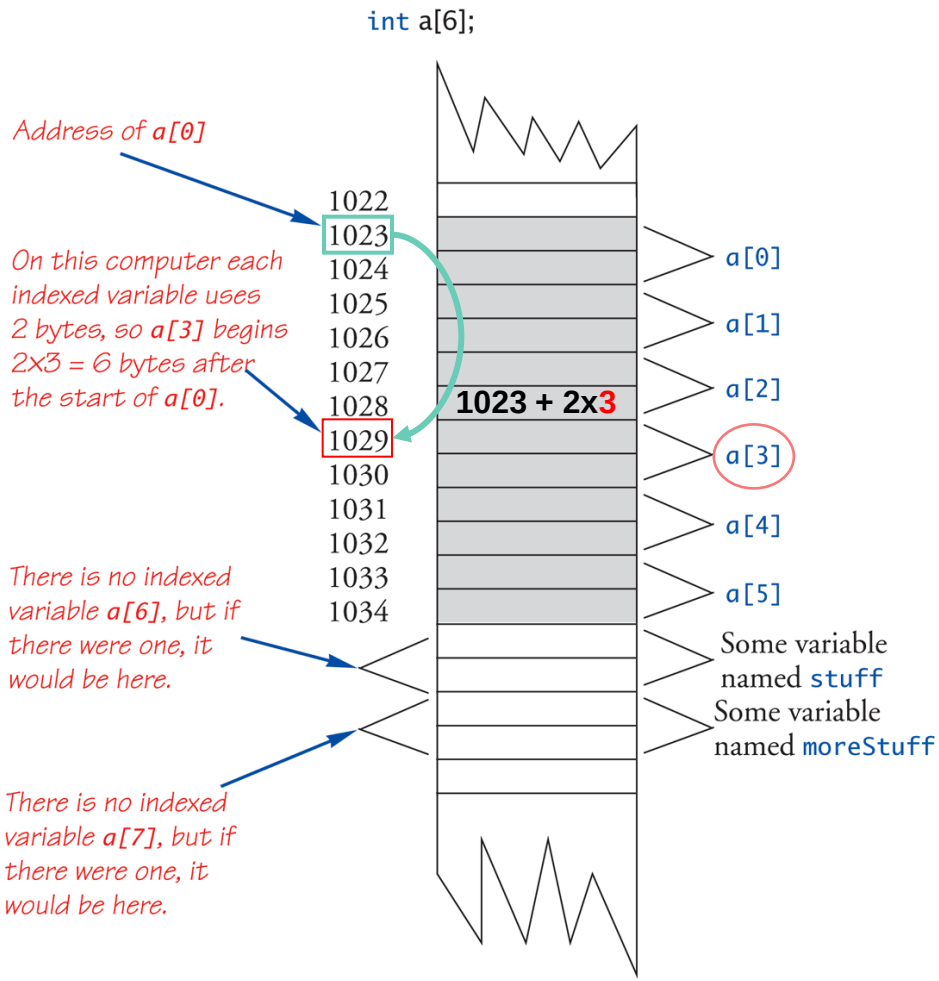
      = Bytes of type `int` * index

      = 2 * 3
    - The result is the address of `a[3]`

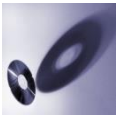

Arrays in Memory (cont'd)

Display 5.2 An Array in Memory



Major Pitfall

- **Array indexes always start with zero!**
- **C++ will "let" you go beyond range**


Initializing Arrays

- **Initialization**

```
int a[3] = {5, 16, 8};
```

is equivalent to

```
int a[] = {5, 16, 8};
```

- **Pitfall**
  - Although array indexed variables may sometimes be automatically initialized to zero, you **cannot and should not** count on it

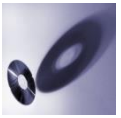


Arrays in Functions (1)

- **Indexed Variables**
  - Call-by-value argument
  - Call-by-reference argument

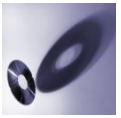
```
testFunction(a[3]);

int i=3;
testFunction(a[i]);
```

View an index variable as a "**variable**"


Arrays in Functions (2)

- **Entire Arrays as Function Arguments**
  - Array parameter
    - Expressed by a square bracket with no index inside, e.g., `a[]`
    - **Neither** a call-by-value parameter **nor** a call-by-reference parameter
  - Array argument
    - Expressed without any square brackets or index, e.g., `a`


Example

Display 5.3 **Function with an Array Parameter**

SAMPLE DIALOGUEFUNCTION DECLARATION

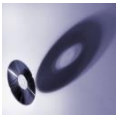
```
void fillUp(int a[], int size);
//Precondition: size is the declared size of the array a.
//The user will type in size integers.
//Postcondition: The array a is filled with size integers
//from the keyboard.
```

SAMPLE DIALOGUEFUNCTION 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;
}
```

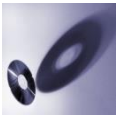
Sample Function Call

```
int score[5], numScores = 5;
fillUp(score, numScores);
```


Arrays in Functions (3)

- **Array Parameter**
  - Behaves very much like a *call-by-reference* parameter
  - If the formal parameter in the function body is changed, then the array argument will be changed


Quiz

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.
```

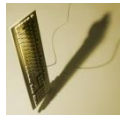
SAMPLE DIALOGUE FUNCTION DEFINITION

```
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;
}
```

Sample Function Call

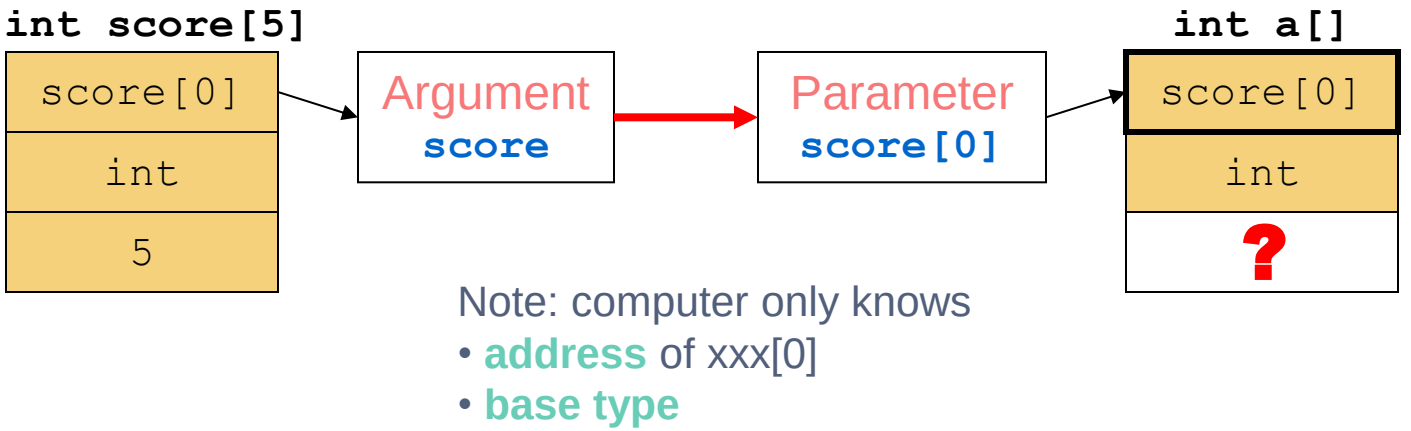
```
int score[5], numScores = 5;
fillUp(score, numScores);
```



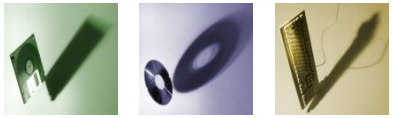


Arrays in Functions (4)

- **Scenario**
  - An array has three parts
    - The address of the first indexed variable, e.g. `score[0]`
    - The base type of the array, e.g. `int`
    - The size of the array, e.g. `5`
  - When function calls



Note: computer only knows
- **address** of xxx[0]
- **base type**

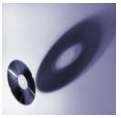


Arrays in Functions (5)

- **The `const` modifier**
  - For array parameter, function can change the array (as a weak form of call-by-reference way)
  - To disallow the array to be changed → Insert the modifier `const` before the array parameter

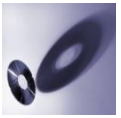
```
void calculateScore(const int a[], int size);
```

    - Recall: `const` can be used with *any* kind of parameter, but is normally useful for
      - array parameter
      - call-by-reference parameters for classes


Arrays in Function (6)

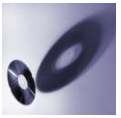
- **Returns an array?**
  - Functions cannot return arrays same way that simple types are returned
  - Done with a pointer to the array
  - We will discuss it later on


Programming with Arrays

- **You should have a lot of experience in it**
  - Searching
  - Sorting
- **Partially filled arrays (PFA)**
  - Keep track of how many elements are stored
  - Limit the size of array to use
  - Why bother?

```
void inputScore(int a[], int size, int& numberUsed);

void calculateScore(int a[], int numberUsed);
```


Multidimensional Arrays

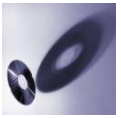
- **Basics**
  - An example

```
char page[30][100];
```

    - Visualize as:

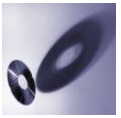
```
page[0][0], page[0][1],   …, page[0][99]
page[1][0], page[1][1],   …, page[1][99]
…
page[29][0], page[29][1], …, page[29][99]
```

  - Two-dimensional arrays: Array of array
  - C++ allows any number of indexes


Multidimensional Arrays (cont'd)

- **Array parameters**
  - One-Dimension
    - Declaration: `void showScores(int a[], int size);`
    - Invocation: `showScores(score, 5);  //int score[5]`
  - Two-Dimension
    - Declaration: `void showPages(int a[][100], int size);`
    - Invocation: `showPages(page, 30);  //int page[30][100]`
  - A multidimensional array is an **array of arrays**
    - Therefore, `page[30][100]` can be viewed as a one-D array of **size 30** whose **base type** is an one-D array of integer of size 100
    - The second dimension is part of the description of the base type

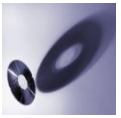




Summary (1)

- **Array**
  - A collection of data of same type
  - index always starts with zero
  - Background scenario when using arrays

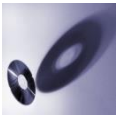
	address `a[0]`	base type `int`	size `5`
Referring to a[i]	v	v	
Array parameter	v	(defined by function)	

  - Be careful about the potential error of “out of range” – could be disastrous!


Summary (2)

- **Array in Functions**
  - Index variables
    - *Either* call-by-value *or* call-by-reference
  - Entire array
    - *Neither* call-by-value *nor* call-by-reference
    - Array parameter (as if weak form of call-by-reference)
    - The scenario of passing an array argument to a function
    - Function can change the array of the argument
      → To avoid, add the modifier **`const`**
  - Partially filled arrays


Summary (3)

- **Multidimensional Array**
  - An array of arrays
  - Multidimensional array parameters
    - The second dimension is part of the description of the base type