# IM 1003: Computer Programming
# Arrays and C++ standard library

Ling-Chieh Kung

Department of Information Management
National Taiwan University



# Outline

- **Arrays**
  - **One-dimensional arrays**
  - Two-dimensional arrays
- C++ standard library



# Why arrays?

- Suppose we want to write a program to store 5 student's scores. We will need to declare 5 variables.
  - `int score1, score2, score3, score4, score5;`
- What if we have 500 students? May we declare 500 variables?
- Even if we have only 5 students, we are not able to write a loop for the input process.

```
for (int i = 0; i < 5; i++)
{
  cin >> score1; // and then... ?
}
```



# Why arrays?

- An array is a collection of variables with **the same type**.

```
int score[5];
```

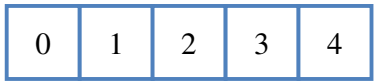
  - These variables are declared with **the same array name** (`score`).
  - They are distinguished by their **indices**.

```
cin >> score[2];
```

- An array is also a **type**: A nonbasic data type.
  - The concept of an array type will become clearer when we discuss pointers.

# Array declaration

- ***data type array name*[*number of elements*];**
- E.g., **int score[5];**
  - This is an integer array with 5 elements (the **array length** is 5).
- It looks like




  - Each square represents one element, which is a **variable**.
  - All the elements are of the same type (in this example, an integer).
  - The **index** starts at **0**! They are **score[0]**, **score[1]**, …, and **score[4]**.
- It occupies 4 bytes * 5 = 20 **continuous** bytes.



# An example

- Let's write a program for the user to input 5 scores:

```
int score[5]; // declaration
for (int i = 0; i < 5; i++)
{
  cin >> score[i]; // use indices!
}
```

- If we have 500 students:

```
int score[500]; // declaration
for (int i = 0; i < 500; i++)
{
  cin >> score[i];
}
```



# Array Initialization

- Arrays will not be initialized automatically.
  - See example "**04_01_arrayInit**".
- Various ways of initializing an array:
  - **int day[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};**
  - **int day[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};** (size of **day** will be 12)
  - **int day[12] = {31, 28, 31};** (nine 0s)
  - **int day[3] = {1, 2, 3, 4};** (**error**!)
- How to initialize all elements to 0?
  - **int score[500] = {0};** (500 0s)



# Array elements

- We access one element by its **index**.
  - **score[0]** is the **first** element of **score**.
  - **score[4]** is the **fifth** element of **score**.
- Each element can be used as a usual variable:
  - **a[0] = 100;** (**a[0]** becomes 100)
  - **cin >> a[1];** (store the input in **a[1]**)
  - **a[4] = a[3] + a[2];**

# An example

- Let's print out all elements in an array:

```
int score[5] = {78, 76, 84, 88, 73};
for(int i = 0; i < 5; i++)
{
  cout << score[i] << " ";
}
```

- Certainly it can also be

```
int score[5] = {78, 76, 84, 88, 73};
for(int i = 1; i <= 5; i++)
{
  cout << score[i-1] << " ";
} // avoid nontrivial indexing!
```



# The boundary of an array

- In C++, it is **allowed** for one to "go outside an array".

```
int array[5] = {1, 2, 3, 4, 8};
int max = 0;

for(int i = 0; i < 10; i++)
{
  if(array[i] > max)
    max = array[i];
}
```

  - No compilation error!
  - **May or may not** generate a run time error: The result is **unpredictable**.
  - See also example "**04_02_arrayBound**".
- We must be aware of the array boundary by ourselves.



# Finding the array length

- Sometimes we do not know the number of elements in an array.
- One way of finding the **array length** is to use **`sizeof`**.
  - **`sizeof`** is a unary operator.
  - It returns the operand's size in byte.
  - The operand can be a type or a variable.
  - **`sizeof(`*`type`*`)`** or **`sizeof(`*`variable`*`)`**.
- Suppose the array is named score, its length equals

```
sizeof(score) / sizeof(score[0]);
```

  - **`sizeof(score)`** is the total number of bytes allocated to the array.
  - **`sizeof(score[0])`** is the number of bytes allocated to the first element.



# An example of finding the array length

- Let's print out all elements in an array:

```
int array[] = {1, 2, 3};
int length = sizeof(array) / sizeof(array[0]);
for(int i = 0; i < length; i++)
{
  cout << array[i] << " ";
}
```

  - As usual, we use the **loop counter** (**`i`**) for array indexing.

# An example of finding the maximum

- Let's try to find the **maximum** number is an array.

```
int array[] = {1, 2, 3, 4, 8};
int length = sizeof(array) / sizeof(array[0]);
int max = array[0];

for(int i = 0; i < length; i++)
{
  if(array[i] > max)
    max = array[i];
}
```



# Good programming style

- Declare a **constant** and then use it for:
  - The declaration of an array.
  - Any loop that traverse the whole array.

```
const int ARRAY_LEN = 10;
int array[ARRAY_LEN] = {0};
int sum = 0;

for (int i = 0; i < ARRAY_LEN; i++)
{
  cin >> array[i];
  sum += array[i];
}
```

- Why?



# Good programming style

- When using **`sizeof`** to count the length of, e.g., an integer array:
  - Use **`sizeof(a) / sizeof(a[0])`**.
  - Do not use **`sizeof(a) / sizeof(int)`**.
- Why?



# Some things you cannot (should not) do

- Suppose you have two arrays **`array1`** and **`array2`**.
  - Even if they have the same length and their elements have the same type, you **cannot** write **`array1 = array2`**. This results in a syntax error.
  - You also **cannot** compare two arrays with **`==`**, **`>`**, **`<`**, etc.
- Although allowed in Dev-C++, you should not declare an array with its length being a **nonconstant** variable.
  - This results in a syntax error in some compilers.
  - In ANSI C++, the array length must be **fixed**.
  - Arrays with dynamic sizes will be covered later.

```
int x = 0;
cin >> x;
int array[x];
```

- The index of an array variable should be an **integer**.
  - Some compiler allows a fractional index (casting is done automatically).

# Outline

- **Arrays**
  - One-dimensional arrays
  - **Two-dimensional arrays**
- C++ standard library



# Two-dimensional arrays

- While a one-dimensional array is like a **vector**, a two-dimensional array is like a **matrix** or **table**.
- Intuitively, a two-dimensional array is composed by **rows** and **columns**.
  - To declare a two-dimensional array, we should specify the numbers of rows and columns.

```
data type array name[rows][columns];
```

- As an example, let's declare an array with 3 rows and 7 columns.

```
double score[3][7];
```



# Two-dimensional arrays

- `double score[3][7];`

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | [0][0] | [0][1] | [0][2] | | | | |
| 1 | [1][0] | | | | [*x*][*y*] | | |
| 2 | [2][0] | | | | | | |

  - `score[0][0]` is the first element; `score[0][1]` is the second element.
  - `score[1][0]` is the **eighth** element.
  - What are the values of *x* and *y*?



# Initialization and array length

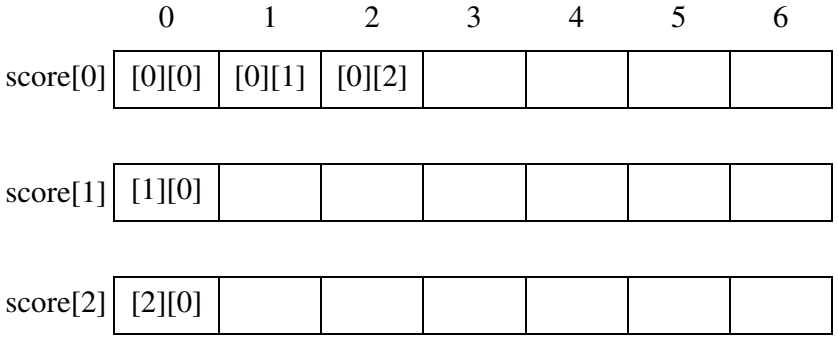
- We may initialize a two-dimensional array as follows:
  - `int score[2][3] = {{1, 2, 3}, {4, 5, 6}};`
  - `int score[2][3] = {1, 2, 3, 4, 5, 6};`
  - `int score[][3] = {1, 2, 3, 4, 5, 6};`
- Finding the numbers of rows and columns:

```
int score[2][3] = {{4, 5, 6}, {7, 8, 9}};
int arrayLength = sizeof(score) / sizeof(score[0][0]);
int rowCount = sizeof(score[0]) / sizeof(score[0][0]);
int columnCount = arrayLength / rowLength;
cout << rowCount << " " << columnCount;
```

  - What is `score[0]`? Isn't `score` a two-dimensional array?

# Embedded one-dimensional arrays

- Two-dimensional arrays are not actually rows and columns.
- A two-dimensional array is actually **several** one-dimensional arrays.






# Embedded one-dimensional arrays

- So for a two dimensional array **`score`**:
  - **`score[0]`** is the ____th one-dimensional array.
  - **`score[0][j]`** is the ____th element of the ____th one-dimensional array.
  - **`score[i]`** is the ____th one-dimensional array.
- All these one-dimensional arrays must be of **the same length**.
  - Two-dimensional arrays with various row lengths can be constructed with the concept of pointers.
- Which description is better?
  - There is an array having three rows and seven columns.
  - There is an array having three rows, each having seven elements.
- This concept will become clearer after we introduce pointers.



# An example

- Let's write a program to do matrix addition.

```
int a[2][3] = {{1, 2, 3}, {1, 2, 3}};
int b[2][3] = {{4, 5, 6}, {7, 8, 9}};
int c[2][3] = {0};

for(int i = 0; i < 2; i++)
{
  for(int j = 0; j < 3; j++)
    c[i][j] = a[i][j] + b[i][j];
}
```

  - Two-dimensional arrays are typically processed with two levels of nested loops.



# Multi-dimensional arrays

- We may have arrays with even higher dimensions.
  - **`char threeDim[3][4][5];`**
  - **`Int eightDim[3][4][5][6][1][7][4][8];`**
- Difficult to imagine and use.

# Outline

- Arrays
  - One-dimensional arrays
  - Two-dimensional arrays
- **C++ standard library**



# C++ standard library

- In C++, many useful tools have been prepared in the **C++ standard library**.
  - A library is a collection of functions, variables, etc.
- To use them, we need to include proper header files.
- One example is the **`cout`** object and the **`<<`** operator.
  - They are provided in the **`iostream`** library.
  - So are **`cin`** and **`>>`**.



# Useful standard library

- Libraries we are going to introduce today include:
  - **`<iostream>`**
  - **`<iomanip>`**
  - **`<cmath>`**
  - **`<cctype>`**
- There are still many useful standard libraries.
- You may learn how to use them by yourself.



# `<iostream>`: Input/output stream

- **`cout`** and **`<<`**.
- **`cin`** and **`>>`**.
- **`endl`**:
  - A "new line" object.
  - Can be used to indicate a new line.

```
cout << "Hello " << endl << "World!" << endl;
cout << "Hello " << "\n" << "World!" << "\n";
cout << "Hello " endl "World!" endl; // syntax error!
cout << "Hello endl World! endl"; // logic error!
```

  - Equivalent to **`"\n"`** and **`cout.flush()`**: Clean the output buffer.

# <iomanip>: I/O manipulation

- Functions and objects defined in **<iomanip>** allow us to format our inputs and outputs.
  - **setw(int w)**: enforce the output to occupy at least **w** characters of width.
  - **setprecision(int p)**: set the total number of digits displayed to **p**.
  - **fixed**: make **setprecision()** to apply on digits after the decimal point.
- To use them, insert an output manipulator into an output stream:

```
cout << an output manipulator << the output stream;
```

- See example "**04_03_iomanip**".



# Output manipulators

- Some other output manipulators:
  - `cout << hex << 16;` // output "10"
  - `cout << uppercase << "aaBb";` // output "AABB"
  - `cout << boolalpha << true;` // output "true"



# <cmath> and <cctype>

- Many useful mathematical functions are defined in **<cmath>**.
  - C mathematics.
  - **pow()**, **sin()**, **cos()**, **abs()**, etc.
- Many useful functions for characters are defined in **<cctype>**.
  - C character type.
  - **tolower(char)** and **isalpha(char)**.
- Just try those functions and then you will learn how to use them.



# C and C++ libraries

- Many C++ libraries originated from C.
- For these libraries originated form C, they are denoted with **a leading "c"**:
  - **<cmath>**, **<cctype>**, **<cstring>**, etc.
- For those libraries defined specifically for C++, there is no such a leading "c".
  - **<iostream>**, **<iomanip>**, **<string>**, etc.