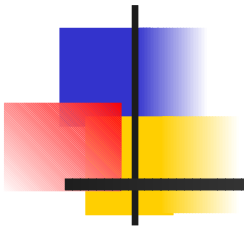
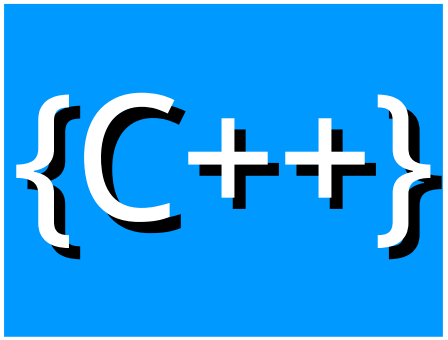


# Pointers

Week 8




Yang-Cheng Chang
Yuan-Ze University
yczhang@saturn.yzu.edu.tw

# Relationship between pointers and built-in arrays

- Pointers can be used to do any operation involving array subscripting.
- Assume the following declarations:

```
// create 5-element int array b; b is a const pointer
int b[ 5 ];
// create int pointer bPtr, which isn't a const
pointer
int *bPtr;
```

# Relationship between pointers and built-in arrays

- We can set `bPtr` to the address of the first element in the built-in array `b` with the statement

```
// assign address of built-in array b to bPtr
bPtr = b;
```

- This is equivalent to assigning the address of the first element as follows:

```
// also assigns address of built-in array b to bPtr
bPtr = &b[ 0 ];
```

# Relationship between pointers and built-in arrays

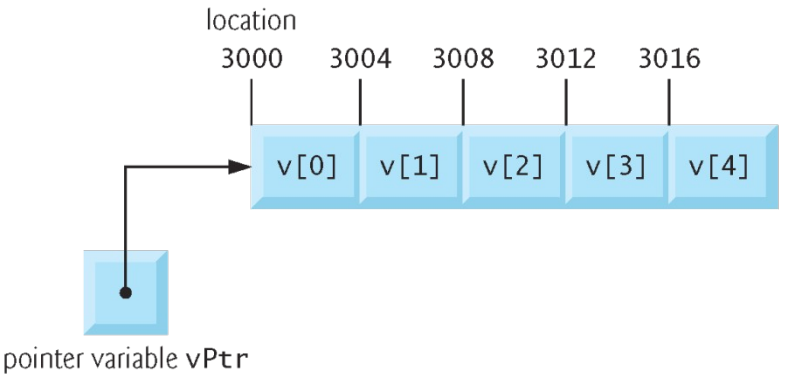



**Fig. 8.15** | Built-in array `v` and a pointer variable `int *vPtr` that points to `v`.

# Relationship between pointers and built-in arrays

***Pointer/Offset Notation***

- Built-in array element `b[ 3 ]` can alternatively be referenced with the pointer expression

```
*( bPtr + 3 )
```

- The `3` in the preceding expression is the offset to the pointer.
- This notation is referred to as pointer/offset notation.
  - The parentheses are necessary, because the precedence of `*` is higher than that of `+`.

# Relationship between pointers and built-in arrays

- Just as the built-in array element can be referenced with a pointer expression, the *address*

```
&b[ 3 ]
```

- can be written with the pointer expression

```
bPtr + 3
```

# Relationship between pointers and built-in arrays

***Pointer/Offset Notation with the Built-In Array's Name as the Pointer***

- The built-in array name can be treated as a pointer and used in pointer arithmetic.
- For example, the expression

```
*( b + 3 )
```

- also refers to the element `b[ 3 ]`.
- In general, all subscripted built-in array expressions can be written with a pointer and an offset.

# Relationship between pointers and built-in arrays

***Pointer/Subscript Notation***

- Pointers can be subscripted exactly as built-in arrays can.
- For example, the expression

```
bPtr[ 1 ]
```

- refers to `b[ 1 ]`; this expression uses pointer/subscript notation.

# Relationship between pointers and built-in arrays

***Demonstrating the Relationship Between Pointers and Built-In Arrays***

- Figure 8.17 uses the four notations discussed in this section for referring to built-in array elements—*array subscript notation, pointer/offset notation with the built-in array's name as a pointer, pointer subscript notation and pointer/offset notation with a pointer*—to accomplish the same task, namely displaying the four elements of the built-in array of `int`s named `b`.

# Relationship between pointers and built-in arrays

```
1  // Fig. 8.17: fig08_17.cpp
2  // Using subscripting and pointer notations with built-in arrays.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int b[] = { 10, 20, 30, 40 }; // create 4-element built-in array b
9     int *bPtr = b; // set bPtr to point to built-in array b
10
11    // output built-in array b using array subscript notation
12    cout << "Array b displayed with:\n\nArray subscript notation\n";
13
14    for ( size_t i = 0; i < 4; ++i )
15       cout << "b[" << i << "] = " << b[ i ] << '\n';
16
17    // output built-in array b using array name and pointer/offset notation
18    cout << "\nPointer/offset notation where "
19       << "the pointer is the array name\n";
20
21    for ( size_t offset1 = 0; offset1 < 4; ++offset1 )
22       cout << "*(b + " << offset1 << ") = " << *( b + offset1 ) << '\n';
23
```

**Fig. 8.17** | Using subscripting and pointer notations with built-in arrays. (Part 1 of 4.)

# Relationship between pointers and built-in arrays

```
24      // output built-in array b using bPtr and array subscript notation
25      cout << "\nPointer subscript notation\n";
26
27      for ( size_t j = 0; j < 4; ++j )
28         cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';
29
30      cout << "\nPointer/offset notation\n";
31
32      // output built-in array b using bPtr and pointer/offset notation
33      for ( size_t offset2 = 0; offset2 < 4; ++offset2 )
34         cout << "*(bPtr + " << offset2 << ") = "
35            << *( bPtr + offset2 ) << '\n';
36   } // end main
```

**Fig. 8.17** | Using subscripting and pointer notations with built-in arrays. (Part 2 of 4.)

# Relationship between pointers and built-in arrays

```
Array b displayed with:

Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
```

**Fig. 8.17** | Using subscripting and pointer notations with built-in arrays. (Part 3 of 4.)

# Relationship between pointers and built-in arrays

```
Pointer/offset notation where the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

**Fig. 8.17** | Using subscripting and pointer notations with built-in arrays. (Part 4 of 4.)

# Assignment 8

- Find the median of the students' scores
  - —Use rand() to generate a set of scores(0~100), and save them to a array(studentScores)
  - —Write a function(findMedianScore) to find the median in a array
    - Use Pointers to access the array in a function
    - Use Loop to implement bubble sort

```
float findMedianScore(int *ptrScore, int amounts);
int studentScores[10] = {0} // use rand() to generate values
int size = sizeof(studentsScores)/sizeof(studentsScores[0])
float median = findMedianScore(studentScores,size)
```

# Median

1, 3, 3, **6**, 7, 8, 9

Median = **6**

1, 2, 3, **4**, **5**, 6, 8, 9

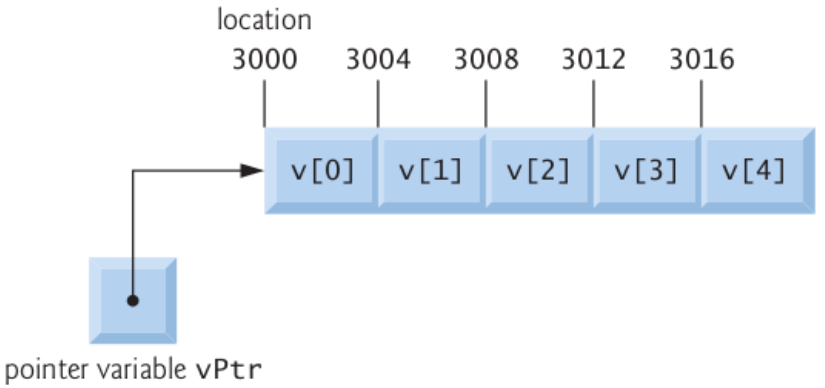
Median = (4 + 5) ÷ 2

= **4.5**

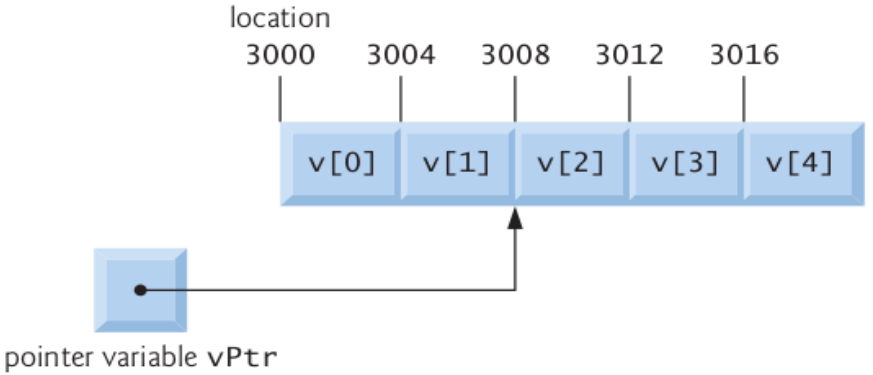
From wikipedia

# Using Pointers to access the array elements

```
int *vPtr = v;
int *vPtr = &v[ 0 ];
```




```
vPtr += 2;
```



# How to determine the number of elements in an array?

- To determine the size of your array in bytes, you can use the sizeof operator:

```
int a[4];
int n = sizeof(a);
```

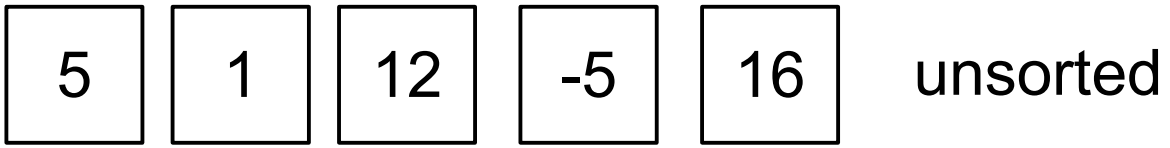
- int are 4 bytes long, so n is 16
- So the preferred divisor is sizeof(a[0]), the size of the zeroeth element of the array.

```
int a[4];
int n = sizeof(a) / sizeof(a[0]);
```

- a[0] is int, so n is 4

# Bubble Sort

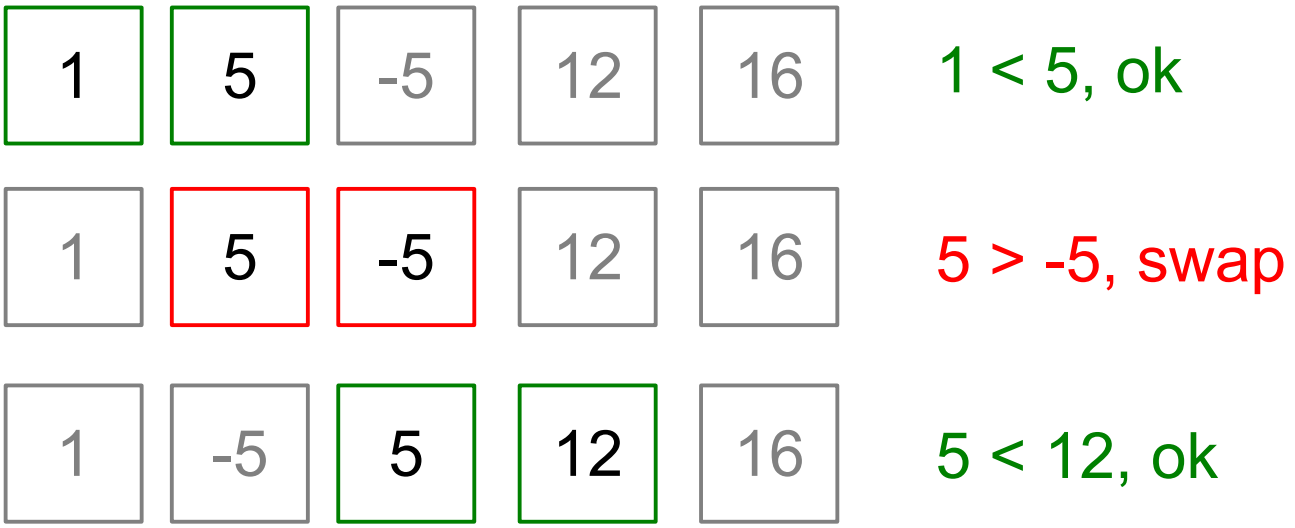
1. Compare each pair of adjacent elements from the beginning of an array and, if they are in reversed order, swap them.
2. If at least one swap has been done, repeat step 1.



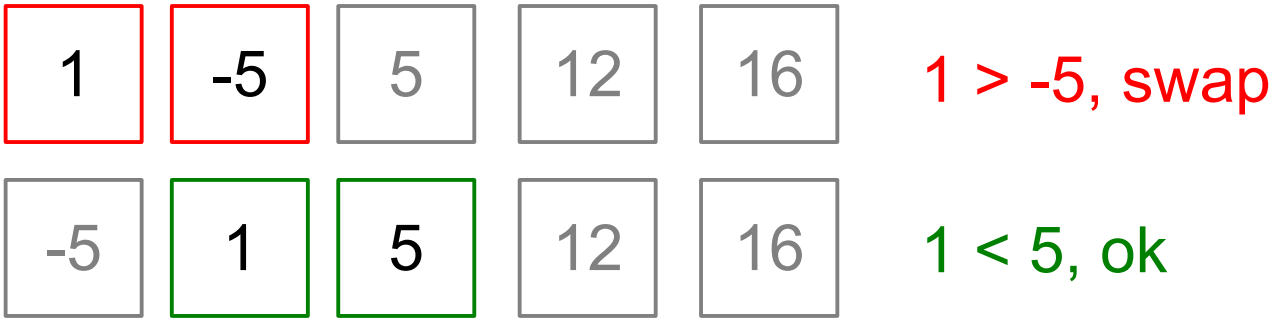
# Bubble Sort


5 1 12 -5 16
5 > 1, swap
1 5 12 -5 16
5 < 12, ok
1 5 12 -5 16
12 > -5, swap
1 5 -5 12 16
12 < 16, ok

# Bubble Sort

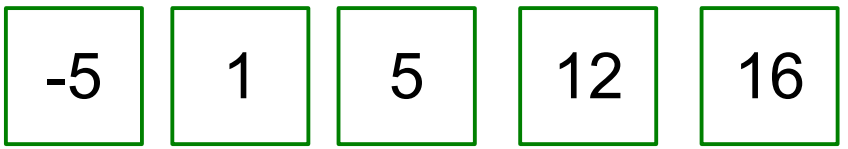


# Bubble Sort



# Bubble Sort


-1 < 5, ok




sorted