Arrays

Unit 5
Gaddis: 7.1-4,6
CS 1428
Fall 2017
Jill Seaman

Array Data Type

- **Array**: a variable that contains multiple values of the same type.
- Values are stored consecutively in memory.
- An array variable definition statement in C++:
  ```cpp
  int numbers[5];
  ```
- This creates an array called `numbers` which contains 5 integer values (ints).

Array - Memory Layout

- The definition: `int numbers[5];`
- allocates the following memory:
  (values are stored consecutively in memory)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>second</td>
<td>third</td>
<td>fourth</td>
</tr>
<tr>
<td>element</td>
<td>element</td>
<td>element</td>
<td>element</td>
</tr>
</tbody>
</table>
  
Array Terminology

- Given the following array definition:
  ```cpp
  int numbers[5];
  ```
- `numbers` is the **name** of the array
- `int` is the data type of the array `elements`
- `5` is the **size declarator**: the number of elements (values) in the array.
Size Declarator

- The size declarator must be an integer and a constant.
  - it must be greater than 0
  - IT CANNOT BE A VARIABLE!
- It can be a literal or a named constant.

```c
const int SIZE = 40;
double grades[SIZE];
```
- Named constants ease program maintenance when the size of the array must be changed.

*Unless you are using a special compiler (mine is not special).*

7.2 Accessing Array Elements

- Each element of the array has a unique subscript (or index) that indicates its position in the array.
- The subscripts are 0-based
  - the first element has subscript 0
  - the second element has subscript 1
  - ...
  - the last element has subscript (size -1)

```
const int SIZE = 40;
double grades[SIZE];
```

Accessing Array Elements

- Syntax to access one element:
  ```c
  numbers[2] //the third element of numbers array
  ```
- Called “numbers at 2” or “numbers sub 2”

Array subscripts

- The subscript is ALWAYS an integer
  - regardless of the type of the array elements.
- the subscript can be ANY integer expression
  - literal: 2
  - variable: i
  - expression: (i+2)/2

```
numbers[2]
numbers[i]
numbers[(i+2)/2]
```
Array subscripts

• Given the following array definition:
  
  double tests[10];

  the expression tests[i] may be used exactly like any variable of type double.

  tests[0] = 79;
  cout << tests[0];
  cin >> tests[1];
  tests[4] = tests[0] + tests[1];

Using array elements:

```cpp
double values[3];  //array definition
values[0] = 22.3;  //assignment to array element
values[1] = 11.1;
cout << “Enter a number: “;
cin >> values[2];
double sum = values[0] + values[1] + values[2];
double avg = sum/3.0;
cout << “Values at zero: “ << values[0] << endl;
int i=2;
if (values[i] > 32.0)
    cout << “Above freezing” << endl;
```

7.4 Array initialization

• You can initialize arrays when they are defined.

```cpp
const int NUM_SCORES = 3;
float scores[NUM_SCORES] = {86.5, 92.1, 77.5};
```

• Values are assigned in order:
  
  scores[0] = 86.5
  scores[1] = 92.1
  scores[2] = 77.5

• NOTE: uninitialized arrays have GARBAGE values stored in them (not necessarily 0).

Partial Array Initialization

• When you initialize, you don’t need to specify a value for every position.

  ```cpp
  float scores[7] = {86.5, 92.1, 77.5};
  ```

• In this case, the first 3 elements are initialized to the specified values.

• The uninitialized values WILL be set to 0!!!!

  ```plaintext
  86.5 92.1 77.5 0 0 0 0
  ```
Implicit array sizing

- When you initialize, you don’t need to specify the size declarator.

```c
float scores[] = {86.5, 92.1, 77.5};
```

- In this case, the compiler determines the size of the array from the number of elements listed.

```
86.5  92.1  77.5
```

Operations over arrays

- Most array operations must be done one element at a time.
- Input the 7 programming assignment grades for a student in CS1428:

```c
const int NUM_SCORES = 7;
int scores[NUM_SCORES];
cout << "Enter the " << NUM_SCORES << " programming assignment scores: " << endl;
cin >> scores[0];
cin >> scores[1];
cin >> scores[2];
cin >> scores[3];
cin >> scores[4];
cin >> scores[5];
cin >> scores[6];
```

- Is there a better way?

7.6 Processing Array Contents

- Generally there are NO operations that you can perform over an entire array.
- Some operations may appear to work (no errors) but you don’t get the desired results.

```c
int numbers1[] = {1, 2, 3};
int numbers2[] = {4, 5, 6};
cin >> numbers1;       //input, won’t work
cout << numbers1 << endl;  //output, won’t work
numbers1 = numbers2;       //assignment, won’t work
if (numbers1 == numbers2)    //comparison, won’t work
    ...
numbers3 = numbers1 + numbers2; //addition, won’t work
```

Array input using a loop

- We can use a for loop to input into the array
- The subscript/index can be a variable

```c
const int NUM_SCORES = 7;
int scores[NUM_SCORES];
cout << "Enter the " << NUM_SCORES << " programming assignment scores: " << endl;
for (int i=0; i < NUM_SCORES; i++) {
    cin >> scores[i];
}
```

- This code is equivalent to the code on the previous slide.
Array output using a loop

• We can use a for loop to **output** the elements of the array

```cpp
const int NUM_SCORES = 7;
int scores[NUM_SCORES];
cout << "Enter the " << NUM_SCORES << " programming assignment scores: " << endl;
for (int i=0; i < NUM_SCORES; i++) {
cin >> scores[i];
}
cout << "You entered these values: ";
for (int i=0; i < NUM_SCORES; i++) {
cout << scores[i] << " ";
}
cout << endl;
```

Finding the maximum value in an array

• We can use a for loop to **find the max** value:
• Note: keep track of the maximum value encountered so far (the **running maximum**)

```cpp
const int NUM_SCORES = 7;
int scores[NUM_SCORES];
cout << "Enter the " << NUM_SCORES << " programming assignment scores: " << endl;
for (int i=0; i < NUM_SCORES; i++) {
cin >> scores[i];
}
int maximum = scores[0]; //init max to first elem
for (int i=1; i < NUM_SCORES; i++) { //start i at 1
    if (scores[i] > maximum)
        maximum = scores[i]; //save the new maximum
} // no else needed
```

Summing values in an array

• We can use a for loop to **sum** the elements of the array (the **running total**)

```cpp
const int NUM_SCORES = 7;
int scores[NUM_SCORES];
cout << "Enter the " << NUM_SCORES << " programming assignment scores: " << endl;
for (int i=0; i < NUM_SCORES; i++) {
cin >> scores[i];
}
int total = 0; //initialize accumulator
for (int i=0; i < NUM_SCORES; i++) {
total = total + scores[i];
}
```

Array assignment

• To **copy/assign** one array to another, you must assign element by element.

```cpp
const int SIZE = 4;
int values1[SIZE] = {100, 200, 300, 400};
int values2[SIZE];
// values2 = values1; WRONG, won’t work correctly
for (int i = 0; i < SIZE; i++) {
    values2[i] = values1[i];
}
values2 = {3,6,9,27}; //only works for initialization
```

• Note: this also does not work:
7.3 C++: No bounds checking

- C++ does not check it to make sure an array subscript is valid (between 0 and size-1)
- If you use a subscript that is outside the bounds of the array you **may not** get a warning or error.
- You may unintentionally change memory allocated to other variables.

```c
const int SIZE = 3;
int values[SIZE];
for (int i=0; i < 5; i++) {
    values[i] = 100;
}
```

This code defines a three-element array and then writes five values to it (changing the memory after the array).