Structured Programming
Using C++

Lecture 8: Introduction to Arrays

Dr. Amal Khalifa

Lecture Contents:

- Introduction to Arrays
  - What is an array
  - Declaring and referencing arrays
  - Arrays in memory
  - For-loops and arrays

- Arrays in Functions
  - Arrays as function arguments, return values
  - Arrays in recursive functions (next time!!)
Introduction to Arrays

- Array definition:
  - A collection of data of same type

- First "aggregate" data type
  - Means "grouping"
  - int, float, double, char are simple data types

- Used for lists of like items
  - Test scores, temperatures, names, etc.
  - Avoids declaring multiple simple variables
  - Can manipulate "list" as one entity

What is an array??

- Individual parts called many things:
  - Indexed or subscripted variables
  - "Elements" of the array
  - Same data type
  - called index or subscript
    - Numbered from 0 to size - 1
Declaring Arrays

- Declare the array:
  - allocates memory
  - Example:
    ```cpp
    int score[5];
    ```
  - Declares array of 5 integers named "score"
  - Similar to declaring five variables:
    ```cpp
    int score[0], score[1], score[2], score[3], score[4]
    ```

Accessing Arrays

- Access using index/subscript
  ```cpp
  cout << score[3];
  ```
- Size, subscript need not be literal
  ```cpp
  int score[MAX_SCORES];
  score[n+1] = 99;
  ```
  - If n is 2, identical to: score[3]
- Note two uses of brackets:
  - In declaration, specifies SIZE of array
  - Anywhere else, specifies a subscript
Arrays in Memory

- Array declarations allocate memory for entire array
- Sequentially-allocated
  - addresses allocated "back-to-back"
  - Allows indexing calculations
    - Simple "addition" from array beginning (index 0)

Initializing Arrays

- simple variables initialization??
- Arrays can as well:
  int children[3] = {2, 12, 1};
  - Equivalent to:
    int children[3];
    children[0] = 2;
    children[1] = 12;
    children[2] = 1;
Auto-Initializing Arrays

- If fewer values than size supplied:
  - Fills from beginning
  - Fills "rest" with zero of array base type

- If array-size is left out
  - Declares array with size required based on number of initialization values
  - Example:
    ```
    int b[] = {5, 12, 11};
    ```
    - Allocates array b to size 3

Array Usage

- Powerful storage mechanism

- Can issue command like:
  - "Do this to i\textsuperscript{th} indexed variable" where i is computed by program
  - "Display all elements of array score"
  - "Fill elements of array score from user input"
  - "Find highest value in array score"
  - "Find lowest value in array score"
Use an array to get the scores of 5 students and show how much each score differs from the highest score.

**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

**Display 5.1  Program Using an Array**

```cpp
#include <iostream>
using namespace std;

int main()
{
  int i, score[5], max;
  cout << "Enter 5 scores:\n";
  cin >> score[0];
  max = score[0];
  for (i = 1; i < 5; i++)
  {
    cin >> score[i];
    if (score[i] > max)
      max = score[i];
    //max is the largest of the values score[0],..., score[i].
  }
  cout << "The highest score is " << max << endl
       << "The scores and their:\n";
  for (i = 0; i < 5; i++)
    cout << score[i] << " off by ";
    (max - score[i]) << endl;
  return 0;
}
```

Dr. Amal Khalifa, 2012
for-loops with Arrays

- **Natural counting loop**
  - Naturally works well "counting thru" elements of an array

- **Example:**
  ```c++
  for (idx = 0; idx<5; idx++)
  {
      cout << score[idx] << "off by " << max - score[idx] << endl;
  }
  ```
  - Loop control variable (idx) counts from 0 – 5

---

Major Array Pitfall

- Array indexes always start with zero!
- Zero is "first" number to computer scientists
- C++ will "let" you go beyond range
  - Unpredictable results
  - Compiler will not detect these errors!
- Up to programmer to "stay in range"
Major Array Pitfall Example

- Indexes range from 0 to (array_size – 1)
  - Example:
    ```c
    double temperature[24]; // size is 24
    ```
  - They are indexed as:
    ```c
    temperature[0], temperature[1] … temperature[23]
    ```
  - Common mistake:
    ```c
    temperature[24] = 5;
    ```
  - Index 24 is "out of range"!
  - No warning, possibly disastrous results

Defined Constant as Array Size

- Always use defined/named constant for array size
  ```c
  const int NUMBER_OF_STUDENTS = 5;
  int score[NUMBER_OF_STUDENTS];
  ```
  - Improves readability
  - Improves maintainability
    - If size changes → requires only ONE change in program!
  - Use everywhere size of array is needed
    - In for-loop for traversal
    - When passing array to functions (later)
Arrays in Functions

- **As arguments to functions**
  - Indexed variables
    - An individual "element" of an array can be function parameter
  - Entire arrays
    - All array elements can be passed as "one entity"

- **As return value from function**
  - Can be done → chapter 10

---

Example: Payroll

Use an array to store the salaries of 10 employees. Use a function to independently compute their new salaries after a 20% raise. Write a test program that allows the user to enter the salaries then print out the new salaries.
Indexed Variables as Arguments

- Indexed variable handled same as simple variable of array base type
  ```c
  void myFunction(double par1);
  int i;
  double n, a[10];
  ```

- Can we make these function calls??:
  ```c
  myFunction(i);
  myFunction(a[3]);
  myFunction(n);
  ```

Subtlety of Indexing

- Consider:
  ```c
  myFunction(a[i]);
  ```
  - Value of i is determined first
    - It determines which indexed variable is sent
  ```c
  myFunction(a[i*5]);
  ```
  - Perfectly legal, from compiler's view
  - Programmer responsible for staying "in-bounds" of array
Partially-filled Arrays

- Difficult to know exact array size needed
- Must declare to be largest possible size
  - Must then keep "track" of valid data in array
  - Additional "tracking" variable needed
    - int numberUsed;
    - Tracks current number of elements in array

Array as Argument: How?

- What's really passed?
  - "Contiguous" portion of memory
  - Very similar to "pass-by-reference"
  - Only address of 1st element is passed to functions
    - Just the beginning address of array

- Think of array as 3 "pieces"
  - Address of first indexed variable (arrName[0])
  - Array base type
  - Size of array
Array Parameters

- May seem strange
  - No brackets in array argument
  - Must send size separately

- One nice property:
  - Can use SAME function to fill any size array!
  - Exemplifies "re-use" properties of functions
  - Example:
    ```
    int score[5], time[10];
    fillUp(score, 5);
    fillUp(time, 10);
    ```

Example

Use an array to get the scores of 5 students and write a program to show how much each score differs from the highest score.

- Function for input
- Function to compute the highest score
- Function for display

**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
Example: Golf scores

Display 5.5  Partially Filled Array

1 //Shows the difference between each of a list of golf scores and their average.
2 #include <iostream>
3 using namespace std;
4 const int MAX_NUMBER_SCORES = 10;
5
6 void fillArray(int a[], int size, int& numberUsed);
7 //Precondition: size is the declared size of the array a.
8 //Postcondition: numberUsed is the number of values stored in a.
9 //Nonnegative integers read from the keyboard.
10 double computeAverage(const int a[], int numberUsed);
11 //Precondition: a[0] through a[numberUsed-1] have values; numberUsed > 0.
12 //Returns the average of numbers a[0] through a[numberUsed-1].
13 void showDifference(const int a[], int numberUsed);
14 //Precondition: The first numberUsed indexed variables of a have values.
15 //Postcondition: Gives screen output showing how much each of the first
16 //numberUsed elements of the array a differs from their average.

(continued)

Example: C++ Arrays Example:

Display 5.5  Partially Filled Array

17 int main( )
18 {
19     int score[MAX_NUMBER_SCORES], numberUsed;
20     cout << "This program reads golf scores and shows\n";
21     cout << "how much each differs from the average.\n";
22     cout << "Enter golf scores:\n";
23     fillArray(score, MAX_NUMBER_SCORES, numberUsed);
24     showDifference(score, numberUsed);
25     return 0;
26 }

5-26
Arrays Example:

```cpp
void fillArray(int a[], int size, int& numberUsed)
{
    cout << "Enter up to " << size << " nonnegative whole numbers.\n"
    << "Mark the end of the list with a negative number.\n";
    int next, index = 0;
    cin >> next;
    while ((next >= 0) && (index < size))
    {
        a[index] = next;
        index++;
        cin >> next;
    }
    numberUsed = index;
}
```

Arrays Example:

```cpp
double computeAverage(const int a[], int numberUsed)
{
    double total = 0;
    for (int index = 0; index < numberUsed; index++)
        total = total + a[index];
    if (numberUsed > 0)
    {
        return (total/numberUsed);
    }
    else
    {
        cout << "ERROR: number of elements is 0 in computeAverage.\n"
        << "computeAverage returns 0.\n";
        return 0;
    }
}
```
Arrays Example:

Display 5.5  Partially Filled Array

```cpp
57  void showDifference(const int a[], int numberUsed)
58  {
59      double average = computeAverage(a, numberUsed);
60      cout << "Average of the " << numberUsed
61      << " scores = " << average << endl
62      << "The scores are:\n";
63      for (int index = 0; index < numberUsed; index++)
64          cout << a[index] << " differs from average by "
65          << (a[index] - average) << endl;
66  }
```

**SAMPLE DIALOGUE**

This program reads golf scores and shows how much each differs from the average. Enter golf scores:
Enter up to 10 nonnegative whole numbers. Mark the end of the list with a negative number.
69 74 68 -1
Average of the 3 scores = 70.3333
The scores are:
69 differs from average by -1.33333
74 differs from average by 3.66667
68 differs from average by -2.33333

The const Parameter Modifier

- Recall: array parameter actually passes address of 1st element
  - Similar to pass-by-reference
- Function can then modify array!
  - Often desirable, sometimes not!
- Protect array contents from modification
  - Use "const" modifier before array parameter
    - Called "constant array parameter"
    - Tells compiler to "not allow" modifications
Functions that Return an Array

- Functions cannot return arrays same way simple types are returned
- Requires use of a "pointer"
- Will be discussed in chapter 10…

That’s all for today !!
Thanks…..