C++ for Science and Engineering COSC3000/6000

2018 Spring Semester

Part XI
Arrays

1 Static Arrays

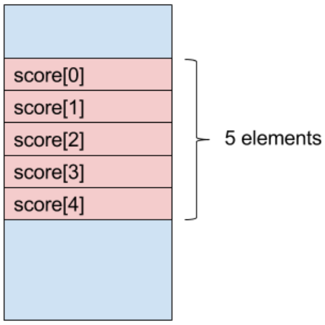
1.1 Declaring an Array

- An array, named score, containing five variables of type int can be declared as

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

- This is like declaring 5 variables of type int: score[0], score[1], . . . , score[4]

- The value in brackets is called
  - A subscript
  - An index
- To declare an array, use the syntax:
  Type_Name Array_Name[Declared_Size];
  - Type_Name can be any type
  - Declared_Size must be a constant



- Once declared, the array consists of the indexed variables:
  - **Array_Name[0]** to **Array_Name[Declared_Size -1]**

1.1.1 The Array Variables

- The variables making up the array are referred to as
  - Indexed variables
  - Subscripted variables
  - Elements of the array
- The number of indexed variables in an array is the declared size, or size, of the array
  - The largest index is one less than the size
  - The first index value is zero

1.1.2 Array Variable Types

- An array can have indexed variables of any type
- All indexed variables in an array are of the same type
  - This is the **base type** of the array
- An indexed variable can be used anywhere an ordinary variable of the base type is used

1.1.3 Using [ ] With Arrays

- In an array declaration, [ ]'s enclose the sizeof the array such as this array of 5 integers:

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

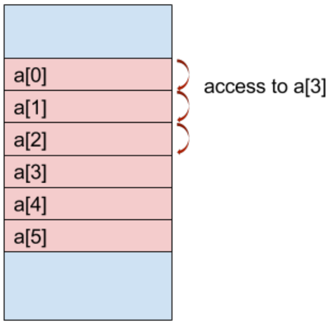
- When referring to one of the indexed variables,the [ ]'s enclose a number identifying one of the indexed variables
  - score[3] is one of the indexed variables
  - The value in the [ ]'s can be any expression that evaluates to one of **the integers 0 to (size -1)**

1.2 Indexed Variable Assignment

- To assign a value to an indexed variable, use the assignment operator:

```
                int n = 2;
                score[n + 1] = 99;
```

- In this example, variable score[3] is assigned 99



1.2.1 Loops and Arrays

- for-loops are commonly used to step through arrays
  - Example:

```
for (i = 0; i < 5; i++)  // for(i=(First index) ; i < (Last index) (size - 1) ; i++)
{
        cout << score[i] << " off by " << (max - score[i]) << endl;
}
```

could display the difference between each score and the maximum score stored in an array

1.2.2 Constants and Arrays

- Use constants to declare the size of an array
  - Using a constant allows your code to be easily altered for use on a smaller or larger set of data
    - Example:

```
                const int NUMBER_OF_STUDENTS = 50;
                int score[NUMBER_OF_STUDENTS];
                :
                for ( i = 0; i < NUMBER_OF_STUDENTS; i++)
                        cout << score[i] << " off by " << (max - score[i]) << endl;
```

- Only the value of the constant must be changed to make this code work for any number of students

1.2.3 Variables and Declarations

- Most compilers do not allow the use of a variable to declare the size of an array
  - Example:

```
cout << "Enter number of students: ";
cin >> number;
int score[number];
```

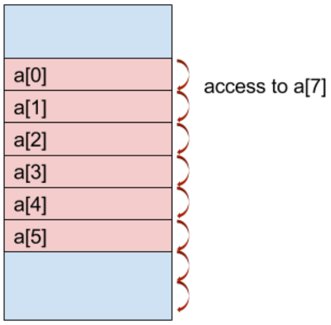
  - This code is **illegal** on many compilers (many older compilers do not allow.)
  - Recent GNU compiler on Mac allows to declare arrays of variable length. But this is not recommended, instead, use **dynamic array** or **std::vector** class.

1.3 Arrays and Memory

- Declaring the array int a[6]
  - Reserves memory for six variables of type int
  - The variables are stored one after another
  - The address of a[0] is remembered
    - ∗ The addresses of the other indexed variables is not remembered
  - To determine the address of a[3]
    - ∗ Start at a[0]
    - ∗ Count enough memory for three integers to find a[3]

1.3.1 Array Index Out of Range

- A common error is using a nonexistent index .
  - Index values for int a[6] are the values 0 through 5.
  - An index value not allowed by the array declaration is out of range.
  - Using an out of range index value does not produce an error message!
- If an array is declared as: **int a[6];** and an integer is declared as: **int i = 7;**
- Executing the statement **a[i] = 238;** causes. . .
  - The computer to calculate the address of the illegal a[7] (This address could be where some other variable is stored)
  - The value 238 is stored at the address calculated for a[7]
  - No warning is given!



1.4 Initializing Arrays

- To initialize an array when it is declared
  - The values for the indexed variables are enclosed in braces and separated by commas
- Example:

```
int children[3] = { 2, 12, 1 };
```

- Is equivalent to:

```
int children[3];
children[0] = 2;
children[1] = 12;
children[2] = 1;
```

- If too few values are listed in an initialization statement
  - The listed values are used to initialize the first of the indexed variables
  - The remaining indexed variables are initialized to a zero of the base type
  - Example:
    ```
                    int a[10] = {5, 5};
    ```
  - initializes a[0] and a[1] to 5 and a[2] through a[9] to 0
- If no values are listed in the array declaration, some compilers will initialize each variable to a zero of the base type
  - DO NOT DEPEND ON THIS!

1.5 Arrays in Functions

- Indexed variables can be arguments to functions
  - Example: If a program contains these declarations:
    ```
    int i, n, a[10];
    void my_function(int n);
    ```
  - Variables a[0] through a[9] are of type int, making these calls legal:
    ```
    my_function( a[ 0 ] );
    my_function( a[ 3 ] );
    my_function( a[ i ] );
    ```

1.5.1 Arrays as Function Arguments

- A formal parameter can be for an entire array
  - Such a parameter is called an array parameter
    - It is not a call-by-value parameter
    - It is not a call-by-reference parameter
    - Array parameters behave much like **call-by-reference** parameters

1.5.2 Array Parameter Declaration

- An array parameter is indicated using empty brackets in the parameter list such as
  ```
  void fill_up(int a[ ], int size);
  ```

1.5.3 Function Calls With Arrays

- If function **fill_up** is declared above
  ```
  const int number_of_scores = 5;
  int score[number_of_scores];
  fill_up(score, number_of_scores);
  ```

1.5.4 Array Formal Parameters

- An array formal parameter is a placeholder for the argument
  - When an array is an argument in a function call, an action performed on the array parameter is performed on the array argument
  - The values of the indexed variables can be changed by the function

- What does the computer know about an array?
  - The base type
  - The address of the first indexed variable
  - The number of indexed variables
- What does a function know about an array argument?
  - The base type
  - The address of the first indexed variable
- Because a function **does not know the size** of an array argument...
  - The programmer should include a formal parameter that specifies the size of the array
  - The function can process arrays of various sizes

1.5.5 const Modifier

- Array parameters allow a function to change the values stored in the array argument
- If a function should not change the values of the array argument, use the modifier **const**
- An array parameter modified with **const** is a constant array parameter
  - Example:
    ```
    void show_the_world(const int a[ ], int size);
    ```
- If **const** is used to modify an array parameter:
  - **const** is used in both the function declaration and definition to modify the array parameter
  - The compiler will issue an error if you write code that changes the values stored in the array parameter
- If a function with a constant array parameter calls another function using the **const** array parameter as an argument...
  - The called function must use a constant array parameter as a placeholder for the array
  - The compiler will issue an error if a function is called that does not have a **const** array parameter to accept the array argument
    ```
    double compute_average(int a[ ], int size);
    void show_difference(const int a[ ], int size) {
            double average = compute_average(a, size);
    }
    ```
  - **compute_average** has no constant array parameter
  - This code generates an **error** message because **compute_average** could change the array parameter

1.5.6 Returning An Array

- Recall that functions can return a value of type int, double, char, . . . , or a class type
- Functions cannot return arrays (*)
- We learn later how to return a pointer to an array