Chapter 6: Arrays

CSE 2010 Week 9

Background

- So far in this course, we have learned how to declare variables of different data types (int, double, string, char, bool), we have even learned how to create our own data types (classes).
- When we declare different variables and objects, even if they represent the same kind of value, they are stored and accessed separately.
    - Example:

```
double salary1 = 65000.00;
double salary2 = 45000.00;
double salary3 = 38000.00;
```

- The above 3 variables are all the same data type and represent salaries, but they are stored in separate places in memory and have no relation to one another.
- Using arrays, we can conveniently store and manage collections of values in easy-to-use containers.

Arrays

- Definition:
  - Static arrays are data structures (aka containers) that store an indexed sequence of elements of the same data type.
  - The size of an array is fixed and determined at compilation time (hence the term *static* arrays).
    - We will learn how to dynamically allocate arrays with pointers later.
- “Indexed sequence” means that each element in an array has an index # associated with it (like strings).
  - The first element of an array always has index 0, next element is index 1, next is index 2..and so on.
  - The last element of an array will always have index (size of array -1).
- Let’s look at different ways of defining arrays and what that looks like in memory...

Array Definition: Example 1

- Syntax:

  ```
  datatype arrayName[ ] = {element1 , element2 , …, elementn};
  ```

  - Will create an array of *n* elements with a size of *n*.

- Example:

  ```
  int x[] = {6,10,17};
  ```

- How you can imagine it is stored in memory:

Element Value	6	10	17
Index #	0	1	2

Array Definition: Example 2

- Syntax:
  `datatype arrayName[n];`
  - Will create an “empty” array of size *n*
- Example:
  `double y[7];`
- How you can imagine it is stored in memory:

Value	?	?	?	?	?	?	?
Index #	0	1	2	3	4	5	6

- Note: Space for 6 elements will be allocated, but that does not mean that those spaces are empty or initialized to zero. There may be old values stored in memory.

Array Definition: Example 3

- Syntax:
  ```
  datatype arrayName[n]{0};
  ```
  - Will create an array of size n with all values set to 0
- Example:
  ```
  double z[7]{0};
  ```
- How you can imagine it is stored in memory:

Value	0	0	0	0	0	0	0
Index #	0	1	2	3	4	5	6

Array Definition: Example 4

- Syntax:

  ```
  datatype arrayName[k] = {element 1 , element 2 , …, element n};
  ```

  - Will create an array of size *k*, initialized with *n* elements.

- Example:

  ```
  double salaries[10]= {65000.00,45000.00,38000.00};
  ```

- How you can imagine it is stored in memory:

Element Value	65000.00	45000.00	38000.00							
Index #	0	1	2	3	4	5	6	7	8	9

Array Definition: Example 5

- Strings are special data types, so they act a little different. Any value not initialized will be set to an empty string.
- Example:
```
string names[6] = {“Bob”,”Sally”,”Jack”};
```
- How you can imagine it is stored in memory:

Value	Bob	Sally	Jack	\0	\0	\0	\0
Index #	0	1	2	3	4	5	6

Accessing Array Elements

- Once an array has been defined, how do we access its elements?
- Remember that each element has an index # associated with it. In an array with a size of *n*, the first element will always have index #0, and the last element will always have index # *n-1*.
- Using the [ ] operator (index operator), we can access the elements.

Syntax:

```
arrayName[i]
```

- Will access array element w/index *i*. You can then use the value as you normally would use any value in an expression.
  - A cout statement
  - A function call
  - An arithmetic expression
  - Another assignment statement

Displaying Array Elements with a for loop

- When we have an initialized array, we know the size of the array, and know the range of the indices.
- For example, if we have an array of size n, we know the indices will range from 0 -> n-1
- Using this range, we can easily print everything inside the array with a for loop.

```
//define an array with an initialization list
int arr1[] = {2,4,6,8,10};
int array_size = 5;
//print the arrays
cout << "Elements in arr1:";
//use a for loop with the index range
for(int i = 0; i < array_size;i++){
        cout << arr1[i] << " ";
}
cout << "\n";
```

Output: Elements in arr1:2 4 6 8 10

Displaying Array Elements with a range based for loop

- We can also use a range based for loop.

```
//define an array with an initialization list
int arr1[] = {2,4,6,8,10};
//print the arrays
cout << "Elements in arr1:";
//use a range based for loop
//i will be set equal to each element
for(int i:arr1){
        cout << i << " ";
}
cout << "\n";
```

Output: Elements in arr1:2 4 6 8 10

Changing Array Elements

- Using the [ ] operator, we can also have an array element on the left side of an assignment statement, updating its value.

Example:

```
int x[] = {6,10,17}; //Elements in x = 6, 10, 17

x[2] = 15; // Assign element at index 2 the value of 15

//Elements in x = 6, 15, 17
```

Example

```
/*
 * Filename: RandomArray.cpp
 * Program that fills an array with random numbers and then prints them
 */
#include <iostream>
#include <cstdlib>//needed for srand() and rand()
#include <ctime>//needed for time()

using namespace std;

int main()
{
        //define an array with all zeroes
        int x[5]{0};
        //print the array elements
        cout << "Elements in x: ";
        //use a range based for loop to print elements
        //i will be each array element
        for(int i:x)
                cout << i << " ";
        cout << "\n";

        //initialize the random # generator
        srand(time(NULL));
        //fill the array with random numbers
        for(int i = 0; i < 5; i++){
                int random = rand() % 15;
                cout << "random #: " << random << "\n";
                //set array element i equal to random #
                x[i] = random;
        }
        //print new array elements
        cout << "Elements in x: ";
        for(int i:x)
                cout << i << " ";
        cout << "\n";

        return 0;

}
```

Output:

```
Elements in x:0 0 0 0 0
random #: 5
random #: 3
random #: 12
random #: 4
random #: 3
Elements in x: 5 3 12 4 3
```

Note on range based for loops for arrays:

- Do not use range based for loops if you don't want to print everything in the array.
- Example: An array of size 10 that only has 2 elements in it.

Storing user input into an array

- We can store a user's input into an array using a loop.
- The size of an array is determined at compilation time, meaning that its size cannot change as the program is running, so when you write your program that is filling arrays with values you have to keep track of two things:
  1. The max size of the array
  2. The actual number of elements in the array

Let's look at an example of this (userArray.cpp on Canvas)

Arrays as Function Parameters

- Just like other types, we can have an array as a function parameter and pass them as arguments, but they are special.
- Arrays are always passed by reference, even without the & operator in the parameter name.
- This means that any modification to an array parameter within a function WILL affect the array's value in memory.
- Example:

```
void fillArray(int a[], int size);
```

A function whose first parameter is an array of integers, and second parameter is the size of that array. Changes made to the array in the function will affect the array in memory.

- If you pass an array into a function and don't want to make changes to it, use the const keyword.

```
void printArray(const int a[], int size);
```

A function whose first parameter is a constant array of integers, and second parameter is the size of that array. We cannot make changes to the array since it is a constant.

Let’s look at an example that brings it all together (arrays.cpp on Canvas)