Arrays - I

Dr. Usman Habib

Introduction

- Arrays
  - Structures of related data items
  - Static entity - same size throughout program
- A few types
  - C-like, pointer-based arrays
  - C++, arrays as objects

Arrays

- Array
  - Consecutive group of memory locations
  - Same name and type
- To refer to an element, specify
  - Array name and position number
- Format: *arrayname[position number]*
  - First element at position 0
  - **`n`** element array **`c`**:

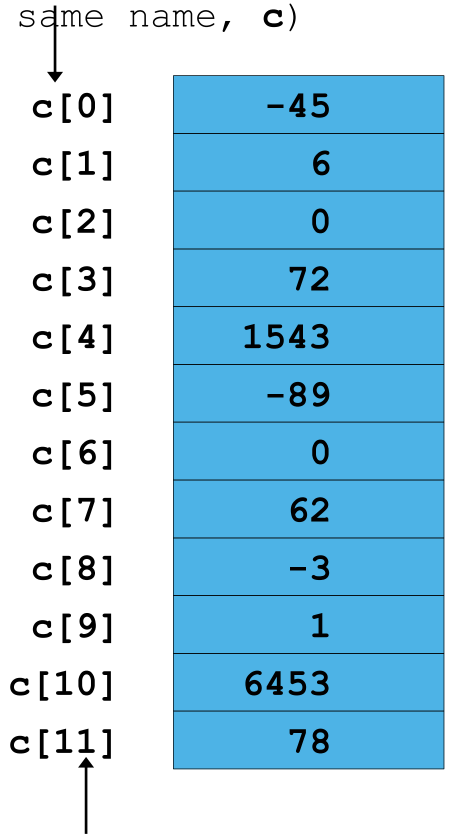
    ```
    c[ 0 ], c[ 1 ]…c[ n - 1 ]
    ```
- Array elements are like normal variables

  ```
  c[ 0 ] =  3;
  cout << c[ 0 ];
  ```
- Performing operations in subscript. If **`x = 3`**,

  ```
  c[ 5 – 2 ] == c[ 3 ] == c[ x ]
  ```

Arrays

Name of array (Note that all elements of this array have the same name, **c**)



c[0]	**-45**
c[1]	**6**
c[2]	**0**
c[3]	**72**
c[4]	**1543**
c[5]	**-89**
c[6]	**0**
c[7]	**62**
c[8]	**-3**
c[9]	**1**
c[10]	**6453**
c[11]	**78**

Position number of the element within array **c**

Declaring Arrays

- Declaring arrays - specify:
  - Name
  - Type of array
  - Number of elements
  - Examples

```
int c[ 10 ];
float hi[ 3284 ];
```

- Declaring multiple arrays of same type
  - Similar format as other variables
  - Example

```
int b[ 100 ], x[ 27 ];
```

Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  - If not enough initializers, rightmost elements become 0
  - If too many initializers, a syntax error is generated

```
int n[ 5 ] = { 0 }
```

  - Sets all the elements to **`0`**
- If size omitted, the initializers determine it

```
int n[] = { 1, 2, 3, 4, 5 };
```

  - 5 initializers, therefore **`n`** is a 5 element array

```cpp
// Fig. 4.4: fig04_04.cpp
// Initializing an array with a declaration
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

int main()
{
   int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

   cout << "Element" << setw( 13 ) << "Value" << endl;

   for ( int i = 0; i < 10; i++ )
      cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;

   return 0;
}
```

Notice how they array is declared and elements referenced.

```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
```

```cpp
// Fig. 4.7: fig04_07.cpp
// A const object must be initialized

int main()
{
   const int x;  // Error: x must be initialized

   x = 7;        // Err

   return 0;
}
```

Notice that **`const`** variables must be initialized because they cannot be modified later.

```
Fig04_07.cpp:
Error E2304 Fig04_07.cpp 6: Constant variable 'x' must be
   initialized in function main()
Error E2024 Fig04_07.cpp 8: Cannot modify a const object in
   function main()
*** 2 errors in Compile ***
```

Passing Arrays to Functions

- Specify the name without any brackets
  - To pass array **`myArray`** declared as

    ```
    int myArray[ 24 ];
    ```

    to function **`myFunction`**, a function call would resemble

    ```
    myFunction( myArray, 24 );
    ```

  - Array size is usually passed to function
- Arrays passed call-by-reference
  - Value of name of array is address of the first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Individual array elements passed by call-by-value
  - pass subscripted name (i.e., **`myArray[ 3 ]`**) to function

Passing Arrays to Functions

- Function prototype:

```
void modifyArray( int b[], int arraySize );
```

  - Parameter names optional in prototype
    - **`int b[]`** could be simply **`int []`**
    - **`int arraysize`** could be simply **`int`**

```cpp
// Fig. 4.14: fig04_14.cpp
// Passing arrays and individual array elements to functions
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

void modifyArray( int [], int );  // appears strange
void modifyElement( int );

int main()
{
   const int arraySize = 5;
   int i, a[ arraySize ] = { 0, 1, 2, 3, 4 };

   cout << "Effects of passing entire array call-by-reference:"
        << "\n\nThe values of the original array are:\n";

   for ( i = 0; i < arraySize; i++ )
      cout << setw( 3 ) << a[ i ];

   cout << endl;

   // array a passed call-by-reference
   modifyArray( a, arraySize );

   cout << "The values of the modified array are:\n";
```

Functions can modify entire arrays. Individual array elements are not modified by default.

No parameter names in function prototype.

```
The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8
```

```cpp

   for ( i = 0; i < arraySize; i++ )
      cout << setw( 3 ) << a[ i ];

   cout << "\n\n\n"
        << "Effects of passing array element call-by-value:"
        << "\n\nThe value of a[3] is " << a[ 3 ] << '\n';

   modifyElement( a[ 3 ] );

   cout << "The value of a[3] is " << a[ 3 ] << endl;

   return 0;
}

// In function modifyArray, "b" points to the original
// array "a" in memory.
void modifyArray( int b[], int sizeOfArray )
{
   for ( int j = 0; j < sizeOfArray; j++ )
      b[ j ] *= 2;
}

// In function modifyElement, "e" is a loca
// array element a[ 3 ] passed from main.
void modifyElement( int e )
{
   cout << "Value in modifyElement is "
        << ( e *= 2 ) << endl;
}
```

Parameter names required in function definition

```
Effects of passing array element call-by-
value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

```
Effects of passing entire array call-by-reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element call-by-value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

References

Dietal and Dietal : How to Program C++

3rd Edition