Chapter 4

Arrays

Learning Objectives

- Introduction to Arrays
  - Declaring and referencing arrays
  - For-loops and arrays
  - Arrays in memory
- Arrays in Functions
  - Arrays as function arguments, return values
- Programming with Arrays
  - Partially Filled Arrays, searching, sorting
- Multidimensional Arrays

Introduction to Arrays

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

Declaring Arrays

- Declare the array → allocates memory
  int score[5];
  - Declares array of 5 integers named "score"
  - Similar to declaring five variables:
    int score[0], score[1], score[2], score[3], score[4]
- Individual parts called many things:
  - Indexed or subscripted variables
  - "Elements" of the array
  - Value in brackets called index or subscript
    - Numbered from 0 to size - 1

Accessing Arrays

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

Array Program Example: Program Using an Array (1 of 2)

Display 5.1 Program Using an Array

```
1  //Reads in five scores and shows how much each
2  //score differs from the highest score.
3  #include <iostream>
4  using namespace std;
5  int main( )
6  {
7      int i, score[5], max;
8      cout << "Enter 5 scores:\n";
9      cin >> score[0];
10     max = score[0];
11     for (i = 1; i < 5; i++)
12     {
13         cin >> score[i];
14         if (score[i] > max)
15             max = score[i];
16         //max is the largest of the values score[0],..., score[i].
17     }
```

Array Program Example: Program Using an Array (2 of 2)

```
18        cout << "The highest score is " << max << endl
19             << "The scores and their\n"
20             << "differences from the highest are:\n";
21        for (i = 0; i < 5; i++)
22            cout << score[i] << " off by "
23                 << (max - score[i]) << endl;
24        return 0;
25    }
```

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

for-loops with Arrays

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

```
for (idx = 0; idx<5; idx++)
{
      cout << score[idx] << "off by "
                  << max – score[idx] << endl;
}
```

  - Loop control variable (idx) counts from 0 – 4

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

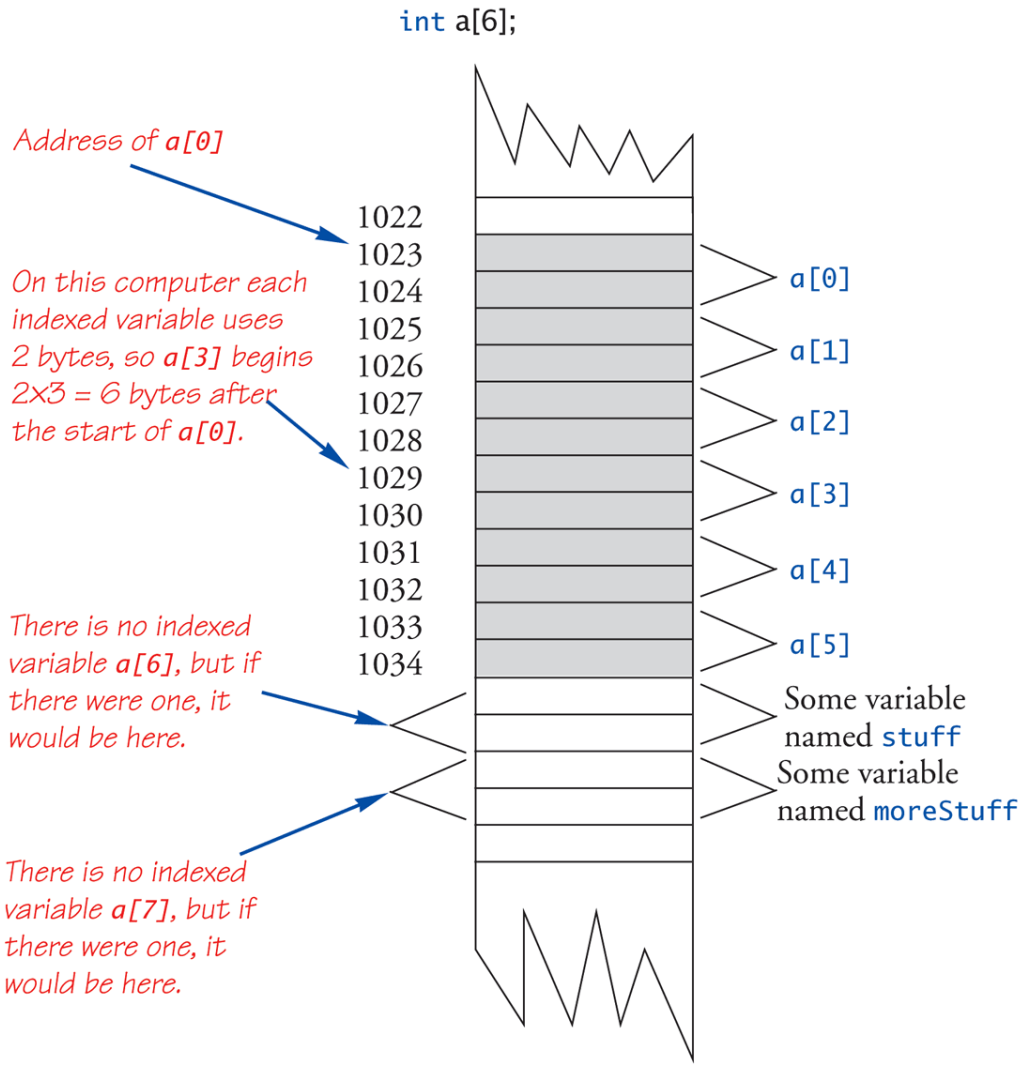
- Indexes range from 0 to (array_size – 1)
  - Example:
    double temperature[24]; // 24 is array size
    // Declares array of 24 double values called temperature
    - They are indexed as:
      temperature[0], temperature[1] … temperature[23]
  - Common mistake:
    temperature[24] = 5;
    - Index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

Arrays in Memory

- Recall simple variables:
  - Allocated memory in an "address"
- Array declarations allocate memory for entire array
- Sequentially-allocated
  - Means addresses allocated "back-to-back"
  - Allows indexing calculations
    - Simple "addition" from array beginning (index 0)

An Array in Memory

Display 5.2 An Array in Memory



Initializing Arrays

- As simple variables can be initialized at declaration:
  ```
  int price = 0;        // 0 is initial value
  ```
- Arrays can as well:
  ```
  int children[3] = {2, 12, 1};
  ```
  - Equivalent to following:
    ```
    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

Indexed Variables as Arguments

- Formal parameters as a pointer

```
void myFunction(int *param) {
  ……..
}
```

- Formal parameters as a sized array

```
void myFunction(int param[10]) {
  ……..
}
```

- Formal parameters as an unsized array

```
void myFunction(int param[]) {
  …….
}
```

Array as Argument: How?

- What's really passed?
- Think of array as 3 "pieces"
  - Address of first indexed variable (e.g., arrName[0])
  - Array base type
  - Size of array
- Only address of first element is passed!
  - Just the beginning address of array
  - Very similar to "call-by-reference"

Partially-filled Arrays

- Difficult to know exact array size needed
- Must declare to be the 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

Multidimensional Arrays

- **Initialization (row-major)**

```
int array[3][4] =
{
{ 1, 2, 3, 4 }, // row 0
{ 6, 7, 8, 9 }, // row 1
{ 11, 12, 13, 14 } // row 2
};
int array[3][5] =
{
{ 1, 2 },
{ 6, 7, 8 },
{ 11, 12, 13, 14 }
};
```

```
int array[][5] =  // int array[3][]
{
{ 1, 2, 3, 4, 5 },
{ 6, 7, 8, 9, 10 },
{ 11, 12, 13, 14, 15 }
};

int array[3][5] = { 0 };

int array[][] =
{
{ 1, 2, 3, 4 },
{ 5, 6, 7, 8 }
};
```