Programming 1

Chapter 6: Arrays

Arrays Hold Multiple Values

- **Array**: variable that can store multiple values of the same type
- Values are stored in adjacent memory locations
- Declared using `[ ]` operator:
  ```
  int tests[5];
  ```
Array - Memory Layout

- The definition:
  ```
  int tests[5];
  ```
  allocates the following memory:

![Diagram of memory allocation]

- first element
- second element
- third element
- fourth element
- fifth element

Array Terminology

- In the definition `int tests[5];`
  - `int` is the data type of the array elements
  - `tests` is the name of the array
  - `5`, in `[5]`, is the size declarator. It shows the number of elements in the array.
  - The size of an array is (number of elements) * (size of each element)
Array Terminology

◆ The size of an array is:
  ■ the total number of bytes allocated for it
  ■ (number of elements) * (number of bytes for each element)

◆ Examples:
  int tests[5] is an array of 20 bytes, assuming 4 bytes for an int
  long double measures[10] is an array of 80 bytes, assuming 8 bytes for a long double

Size Declarators

◆ Named constants are commonly used as size declarators.

    const int SIZE = 5;
    int tests[SIZE];

◆ This eases program maintenance when the size of the array needs to be changed.
Accessing Array Elements

- Each element in an array is assigned a unique *subscript*.
- Subscripts start at 0

![Subscripts diagram]

---

Accessing Array Elements

- The last element’s subscript is $n-1$ where $n$ is the number of elements in the array.

![Subscripts diagram]
Accessing Array Elements

◆ Array elements can be used as regular variables:
  ```
  tests[0] = 79;
  cout << tests[0];
  cin >> tests[1];
  tests[4] = tests[0] + tests[1];
  ```

◆ Arrays must be accessed via individual elements:
  ```
  cout << tests; // not legal
  ```

---

**Program 7-1**

1 // This program asks for the number of hours worked
2 // by six employees. It stores the values in an array.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     const int NUM_EMPLOYEES = 6;
9     int hours[NUM_EMPLOYEES];
10
11     // Get the hours worked by each employee.
12     cout << "Enter the hours worked by ";
13     << NUM_EMPLOYEES << " employees: ";
14     cin >> hours[0];
15     cin >> hours[1];
16     cin >> hours[2];
17     cin >> hours[3];
18     cin >> hours[4];
19     cin >> hours[5];
```

*(Program Continues)*
Here are the contents of the `hours` array, with the values entered by the user in the example output:

```cpp
// Display the values in the array.
cout << "The hours you entered are:";
cout << " " << hours[0];
cout << " " << hours[1];
cout << " " << hours[2];
cout << " " << hours[3];
cout << " " << hours[4];
cout << " " << hours[5] << endl;
return 0;
```

Program Output with Example Input Shown in Bold
Enter the hours worked by 6 employees: 20 12 40 30 30 15 [Enter]
The hours you entered are: 20 12 40 30 30 15

Accessing Array Contents

- Can access element with a constant or literal subscript:
  ```cpp
cout << tests[3] << endl;
  ```

- Can use integer expression as subscript:
  ```cpp
  int i = 3;
cout << tests[i] << endl;
  ```
Using a Loop to Step Through an Array

Example – The following code defines an array, `numbers`, and assigns 99 to each element:

```c
const int ARRAY_SIZE = 5;
int numbers[ARRAY_SIZE];

for (int count = 0; count < ARRAY_SIZE; count++)
    numbers[count] = 99;
```

A Closer Look At the Loop

The variable `count` starts at 0, which is the first valid subscript value.
The loop ends when the variable `count` reaches 5, which is the first invalid subscript value.
The variable `count` is incremented after each iteration.
Array Initialization

- Arrays can be initialized with an initialization list:

```c
const int SIZE = 5;
int tests[SIZE] = {79, 82, 91, 77, 84};
```

- The values are stored in the array in the order in which they appear in the list.
- The initialization list cannot exceed the array size.

---

Code From Program 7-6

```c
const int MONTHS = 12;

for (int count = 0; count < MONTHS; count++)
{
    cout << "Month " << (count + 1) << " has ";
    cout << days[count] << " days.\n";
}
```

Program Output

Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
Partial Array Initialization

◆ If array is initialized with fewer initial values than the size declarator, the remaining elements will be set to 0:

```c
int numbers[7] = {1, 2, 4, 8};
```

![Diagram of an array with initialized and uninitialized elements]

Implicit Array Sizing

◆ Can determine array size by the size of the initialization list:

```c
int quizzes[]={12,17,15,11};
```

◆ Must use either array size declarator or initialization list at array definition:

```c
int quizzes[4] = {12, 17, 15, 11};
```
Initializing With a String

◆ Character array can be initialized by enclosing string in " ":
  
  ```
  const int SIZE = 6;
  char fName[SIZE] = "Henry";
  ```

◆ Must leave room for \0 at end of array
◆ If initializing character-by-character, must add in \0 explicitly:
  
  ```
  char fName[SIZE] =
  { 'H', 'e', 'n', 'r', 'y', '\0'};
  ```

Processing Array Contents

◆ Array elements can be treated as ordinary variables of the same type as the array

◆ When using ++, -- operators, don’t confuse the element with the subscript:
  
  ```
  tests[i]++; // add 1 to tests[i]
  tests[i++]; // increment i, no
              // effect on tests
  ```
Array Assignment

To copy one array to another,
◆ Don’t try to assign one array to the other:
    
    ```
    newTests = tests;  // Won't work
    ```

◆ Instead, assign element-by-element:
    
    ```
    for (i = 0; i < ARRAY_SIZE; i++)
        newTests[i] = tests[i];
    ```


Printing the Contents of an Array

◆ You can display the contents of a character array by sending its name to cout:
    
    ```
    char fName[] = "Henry";
    cout << fName << endl;
    ```

    But, this ONLY works with character arrays!
Printing the Contents of an Array

- For other types of arrays, you must print element-by-element:

```c
for (i = 0; i < ARRAY_SIZE; i++)
    cout << tests[i] << endl;
```

Summing and Averaging Array Elements

- Use a simple loop to add together array elements:
  ```c
  int tnum;
  double average, sum = 0;
  for (tnum = 0; tnum < SIZE; tnum++)
      sum += tests[tnum];
  ```
- Once summed, can compute average:
  ```c
  average = sum / SIZE;
  ```
Finding the Highest Value in an Array

```c
int count;
int highest;
highest = numbers[0];
for (count = 1; count < SIZE; count++) {
    if (numbers[count] > highest)
        highest = numbers[count];
}
```

When this code is finished, the `highest` variable will contain the highest value in the `numbers` array.

Finding the Lowest Value in an Array

```c
int count;
int lowest;
lowest = numbers[0];
for (count = 1; count < SIZE; count++) {
    if (numbers[count] < lowest)
        lowest = numbers[count];
}
```

When this code is finished, the `lowest` variable will contain the lowest value in the `numbers` array.