6.1 Array Basics

An array allows you to store a group of items of the same data type together in memory

- Why? Instead of creating multiple similar variables such as employee1, employee2, employee3 and so on...
- It’s more efficient to create just one variable
  * string employees[50];
  * double salesAmounts[7];
- The number in the [ ] is the size of the array
6.1 Array Basics

• 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.

6.1 Array Basics

• The storage locations in an array are elements
• Each element of the array has a unique number called a subscript that identifies it – the subscript always starts at 0
6.1 Array Basics

Assigning values can be done individually using a subscript...

```java
numbers[0] = 20;
numbers[1] = 30;
numbers[2] = 40;
numbers[3] = 50;
numbers[4] = 60;
```

But, it is much more efficient to use a loop to step through the array
6.1 Array Basics

• Inputting and Outputting Array Contents
  – You can read values from the keyboard and store them in an array element just as you can a regular variable.
  – You can also output the contents of an array element.
6.1 Array Basics

• Arrays can be initialized with an initialization list:

    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.

• When you provide an initialization list, the size declarator can be left out.

    int numbersArray[] = { 10, 20, 30, 40, 50 };
6.1 Array Basics

• Using a loop to step through an array

```c
1 // Create an array to hold three integers.
2 const int SIZE = 3;
3 int myValues[SIZE];
4
5 // Assign 99 to each array element.
6 for (int index = 0; index < SIZE; index++)
7 {
8     myValues[index] = 99;
9 }
```

Program 6.2 (ArrayLoop.cpp)

```c
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     const int SIZE = 3; // Constant for array size
7     int hours[SIZE]; // Array to hold hours
8     int index;
9
10     // Get the hours for each employee.
11     for (int index = 0; index < SIZE; index++)
12     {
13         cout << "Enter the hours worked by employee " << index + 1 << " : " << endl;
14         cin >> hours[index];
15     }
16
17     // Display the values in the array.
18     cout << "The hours you entered are:" << endl;
19     for (int index = 0; index < SIZE; index++)
20     {
21         cout << hours[index] << endl;
22     }
23     return 0;
24 }
```

Program Output with Input Shown in Bold

Enter the hours worked by employee 1: 40 [Enter]
Enter the hours worked by employee 2: 20 [Enter]
Enter the hours worked by employee 3: 15 [Enter]
The hours you entered are:
40
20
15
6.1 Array Basics

• No Array Bounds Checking in C++

  – When you use a value as an array subscript, C++ does not check it to make sure it is a valid subscript.

  – In other words, you can use subscripts that are beyond the bounds of the array.

```c
// Create an array
const int SIZE = 5;
int numbers[SIZE];

// ERROR! The following statement uses an invalid subscript!
numbers[5] = 99;
```
6.1 Array Basics

- **Watch for Off-By-One Errors**
  - An off-by-one error happens when you use array subscripts that are off by one.
  - This can happen when you start subscripts at 1 rather than 0:

```c
// This code has an off-by-one error.
const int SIZE = 100;
int numbers[SIZE];
for (int index = 1; index <= SIZE; index++)
{
    numbers[index] = 0;
}
```

6.2 Sequentially Searching an Array

A sequential search algorithm is a simple technique for finding an item in a string or numeric array:

- Uses a loop to sequentially step through an array
- Compares each element with the value being searched for
- Stops when the value is found or the end of the array is hit

```c
found = false;
index = 0;
while (found == false && index < SIZE)
{
    if (array[index] == searchValue)
    {
        found = true;
    }
    else
    {
        index = index + 1;
    }
}
6.3 Processing the Contents of an Array

Copying an array can be done using loops

```c
const int SIZE = 5;
int firstArray[SIZE] = { 100, 200, 300, 400, 500 };
int secondArray[SIZE];

for (int index = 0; index < SIZE; index++)
{
    secondArray[index] = firstArray[index];
}
```
6.3 Processing the Contents of an Array

- To compare two arrays, you must compare element-by-element:

```cpp
const int SIZE = 5;
int firstArray[SIZE] = { 5, 10, 15, 20, 25 };  
int secondArray[SIZE] = { 5, 10, 15, 20, 25 };  
bool arraysEqual = true;  // Flag variable
int count = 0;  // Loop counter variable
// Compare the two arrays. NOT if( firstArray == secondArray )
while (arraysEqual && count < SIZE)
{
    if (firstArray[count] != secondArray[count])
        arraysEqual = false;
    count++;
}
if (arraysEqual)
    cout << "The arrays are equal.\n";
else
    cout << "The arrays are not equal.\n";
```
6.3 Processing the Contents of an Array

- Totaling the Values in an Array
  - Initialize an accumulator variable to 0
  - Use a loop to step through the array, adding each element to the accumulator variable

Program 6-7 (TotalArray.cpp)

```cpp
#include <iostream>
using namespace std;

int main()
{
    int SIZE = 5;
    int numbers[SIZE] = { 2, 4, 6, 8, 10 };

    int total = 0;

    for (int index = 0; index < SIZE; index++)
    {
        total += numbers[index];
    }

    cout << "The total is " << total << endl;
    return 0;
}
```

Program Output

The total is 30
6.3 Processing the Contents of an Array

- Averaging the Values in an Array
  - First get the total of the array elements
  - Then, divide the total by the number of elements

```cpp
const int SIZE = 4;
const int SIZE = 4;

double total = 0.0;

total += scores[index];

calculate the average.
average = total / SIZE;

// Display the average.
cout << “The average is ” << average << “
```
6.3 Processing the Contents of an Array

Finding the highest & lowest values in an array

- **The highest**
  - Create a variable to hold the highest value
  - Assign the value at element 0 to the highest
  - Use a loop to step through the rest of the elements
  - Each iteration, a comparison is made to the highest variable
  - If the element is greater than the highest value, that value is then assigned to the highest variable

- **The lowest**
  - Same process, but checks if the element is less than the lowest value

```
#include <iostream>
using namespace std;

int main()
{
    // Declare a constant for the array size.
    const int SIZE = 5;

    // Declare an array.
    int numbers[SIZE] = { 0, 1, 12, 6, 2 };

    // Declare a variable to hold the highest value, and initialize it with the first value in the array.
    int highest = numbers[0];

    // Step through the rest of the array, beginning at element 1. When a value greater than highest is found,
    // assign that value to highest.
    for (int index = 1; index < SIZE; index++)
    {
        if (numbers[index] > highest)
            highest = numbers[index];
    }

    // Display the highest value.
    cout << "The highest value is " << highest << endl;
    return 0;
}
```

**Program Output**

```
The highest value is 12
```
6.3 Processing the Contents of an Array

• Passing an Array as an Argument to a Function
  – Typically requires that you pass two arguments:
    • The array itself
    • An integer that specifies the number of elements in the array

```cpp
void showArray(int array[], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << array[i] << " ";
    }
}
```
Program 6.11  (ArrayArgument.cpp)

1 #include <iostream>
2 using namespace std;
3
4 // Function prototype
5 int getTotal(int[], int);
6
7 int main()
8 {
9     // A constant for the array size
10     const int SIZE = 5;
11
12     // An array initialized with values
13     int numbers[SIZE] = { 2, 4, 6, 8, 10 }; 
14
15     // A variable to hold the sum of the elements
16     int sum;
17
18     // Get the sum of the elements.
19     sum = getTotal(numbers, SIZE);
20
21     // Display the sum of the array elements.
22     cout << "The sum of the array elements is "
23         << sum << endl;
24
25     return 0;
26 }
27

Continued…

28 // The getTotal function accepts an integer array, and the
29 // array's size as arguments. It returns the total of the
30 // array elements.
31 int getTotal(int arr[], int size)
32 {
33     // Accumulator, initialized to 0
34     int total = 0;
35
36     // Calculate the total of the array elements.
37     for (int index = 0; index < size; index++)
38     {
39         total = total + arr[index];
40     }
41
42     // Return the total.
43     return total;
44 }

Program Output:
The sum of the array elements is 30
6.3 Processing the Contents of an Array

- Partially Filled Arrays
  - Sometimes an array is only partially filled
  - To avoid processing the unfilled elements, you must have an accompanying integer variable that holds the number of items stored in the array.
    - When the array is empty, 0 is stored in this variable
    - The variable is incremented each time an item is added to the array
    - The variable's value is used as the array's size when stepping through the array.
Program 6.12 (PartiallyFilledArray.cpp)

```
#include <iostream>
using namespace std;

int main()
{
  // Declare a constant for the array size.
  const int SIZE = 100;

  // Declare an array.
  int values[SIZE];

  // Declare a variable to hold the number of items
  // that are actually stored in the array.
  int count = 0;

  // Declare a variable to hold the user's input.
  int number;

  // Prompt the user to enter a number. If the user enters
  // -1 we will stop accepting input.
  cout << "Enter a number or -1 to quit: ";
  cin >> number;

  // If the input is not -1 and the array is not
  // full, process the input.
  while (number != -1 && count < SIZE)
  {
    // Store the input in the array.
    values[count] = number;

    // Increment count.
    count++;

    // Prompt the user for the next number.
    cout << "Enter a number or -1 to quit: ";
    cin >> number;
  }

  // Display the values stored in the array.
  cout << "Here are the numbers you entered: " << endl;
  for (int index = 0; index < count; index++)
  {
    cout << values[index] << endl;
  }
  return 0;
}
```

Continued…
6.4 Parallel Arrays

- By using the same subscript, you can establish a relationship between data stored in two or more arrays
6.4 Parallel Arrays

- From Program 6-14:

```cpp
for (int index = 0; index < NUM_EMPLOYEES; index++)
{
    cout << "Hours worked by employee 
        " index+1 << " 
        
        " cin >> hours[index];
    cout << "Hourly pay rate for employee 
        " index+1 << " 
        
        " cin >> payRate[index];
}
```

6.5 Two-Dimensional Arrays

A two-dimensional array is like several identical arrays put together

- Suppose a teacher has six students who take five tests
6.5 Two-Dimensional Arrays

• Use two size declarators in definition:

```c
const int ROWS = 4, COLS = 3;
int testScores[ROWS][COLS];
```

• First declarator is number of rows; second is number of columns
6.5 Two-Dimensional Arrays

- Two-dimensional arrays are initialized row-by-row:

```c
const int ROWS = 2, COLS = 2;
int numbers[ROWS][COLS] = { {84, 78},
                           {92, 97} };
```

- Can omit the inner { }, however they help to visually organize the initialization values

**Program Output**

Enter a number: 1 [Enter]
Enter a number: 2 [Enter]
Enter a number: 3 [Enter]
Enter a number: 4 [Enter]
Enter a number: 5 [Enter]
Enter a number: 6 [Enter]
Enter a number: 7 [Enter]
Enter a number: 8 [Enter]
Enter a number: 9 [Enter]
Enter a number: 10 [Enter]
Enter a number: 11 [Enter]
Enter a number: 12 [Enter]
Here are the values you entered:

```
1
2
3
4
5
6
7
8
9
10
11
12
```
6.5 Two-Dimensional Arrays

- Passing Two-Dimensional Arrays to Functions
  - Use array name as argument in function call:
    ```
    showArray(exams, SIZE);
    ```
  - Use empty `[ ]` for row, size declarator for column in prototype, header:
    ```
    const int COLS = 2;
    // Prototype
    void getExams(int [][COLS], int);
    ```
    ```
    // Header
    void getExams(int exams[][COLS], int rows)
    ```

```c++
void showArray(const int arr[][COLS], int rows)
{
    for (int x = 0; x < rows; x++)
    {
        for (int y = 0; y < COLS; y++)
        {
            cout << arr[x][y] << "\t";
        }
        cout << endl;
    }
}
```
6.5 Two-Dimensional Arrays

- Then, pass the array and its size when calling the function:

  \[
  \text{showArray}(\text{myArray}, \text{SIZE});
  \]

6.6 Arrays of Three or More Dimensions

- Arrays can also be three or more dimensions.

  \[
  \text{double data[3][3][4];}
  \]