Chapter 5

Arrays
Arrays

• A collection of data of the same type
  – Avoids declaring multiple variables to hold related pieces of data
    • e.g. int sum1, sum2, sum3, sum4;
    • Instead use int sum[4];

• Used for lists of like entities
  – Can manipulate entire list as one entity
Declaring Arrays

- Must be declared before use
  - Just like variables
- Declaration only allocates storage
  - Int arrays not automatically initialized to zero
- Arrays do not know their size
  - Even though it is declared with a particular size, C++ does not keep a record of it.
  - Unlike Java, no size() or length() functions to get the size
Accessing Arrays

• Access using index/subscript
  – cout << score[3];

• Two uses of square brackets
  – In declaration, specifies size of array
  – Everywhere else, specifies a subscript

• Size and subscripts do not need to be literal
  – int score[MAX_SCORES];
  – score[n+1] = 99;
    • If n is 2, identical to: score[3]
Array Example

Display 5.1  Program Using an Array

```cpp
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 Example

```cpp
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
Array Pitfalls

- 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
  - `int temperature[24];`
  - `temperature[24] = 10;`
Defined Constants for Sizes

• Always use defined/named constant for array size
• If size changes, requires 1 change in program
• Example:
  – `const int NUMBER_OF_STUDENTS = 5;`
  – `int score[NUMBER_OF_STUDENTS];`
• Improves
  – Readability
  – Versatility
  – Maintainability
Arrays in Memory

• 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)
    • Allows you to access elements before & after array boundaries
An Array in Memory

Display 5.2  An Array in Memory

```c
int a[6];
```

Address of `a[0]`

On this computer each indexed variable uses 2 bytes, so `a[3]` begins $2 \times 3 = 6$ bytes after the start of `a[0]`.

There is no indexed variable `a[6]`, but if there were one, it would be here.

There is no indexed variable `a[7]`, but if there were one, it would be here.

Some variable named `stuff`
Some variable named `moreStuff`
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
Arrays in Functions

- As arguments to functions
  - Indexed variables
    - An individual "element" of an array can be function parameter
  - Entire arrays
    - All array elements can be passed as "one entity"

- As return value from function
  - Can be done \rightarrow chapter 10
Indexed Variables as Arguments

• Indexed variable handled same as simple variable of array base type

```c
void myFunction(double par1);

int i;
double n, a[10];

myFunction(i); // i is converted to double
myFunction(a[3]); // a[3] is double
myFunction(n); // n is double
```
Entire Arrays as Arguments

• Formal parameter can be entire array
  – Argument then passed in function call is array name
  – Called "array parameter"

• Send size of array as well
  – Typically done as second parameter
  – Simple int type formal parameter
Function with an Array Parameter

Display 5.3  Function with an Array Parameter

**SAMPLE DIALOGUE FUNCTION DECLARATION**

```c
void fillUp(int a[], int size);
// Precondition: size is the declared size of the array a.
// The user will type in size integers.
// Postcondition: The array a is filled with size integers
// from the keyboard.
```

**SAMPLE DIALOGUE FUNCTION DEFINITION**

```c
void fillUp(int a[], int size)
{
    cout << "Enter " << size << " numbers:\n";
    for (int i = 0; i < size; i++)
        cin >> a[i];
    cout << "The last array index used is " << (size - 1) << endl;
}
```
Passing Array to a Function

• To call the fillup() function
  - int score[5], numberOfScores = 5;
  - fillup(score, numberOfScores);
  - 1\textsuperscript{st} argument is entire array
  - 2\textsuperscript{nd} argument is integer value
  - Note no brackets in array argument!

• Think of array as 3 pieces
  - Address of first indexed variable
  - Array base type
  - Size of array

• Only address is passed, just like call-by-reference
Const Parameter Modifier

• Since address of array is passed to function
  – Function can modify array
  – Often desirable, sometimes not!

• Protect array contents from modification
  – Use const modifier before array parameter
    • Called "constant array parameter"
    • Tells compiler to "not allow" modifications
Partially Filled Arrays

• Difficult to know exact array size needed
• Must declare to be 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
Partially Filled Arrays

Display 5.5  Partially Filled Array

1  //Shows the difference between each of a list of golf scores and their average.
2  #include <iostream>
3  using namespace std;
4  const int MAX_NUMBER_SCORES = 10;

5  void fillArray(int a[], int size, int& numberUsed);
6  //Precondition: size is the declared size of the array a.
7  //Postcondition: numberUsed is the number of values stored in a.
8  //a[0] through a[numberUsed–1] have been filled with
9  //nonnegative integers read from the keyboard.

10 double computeAverage(const int a[], int numberUsed);
11  //Precondition: a[0] through a[numberUsed–1] have values; numberUsed > 0.
12  //Returns the average of numbers a[0] through a[numberUsed–1].

13  void showDifference(const int a[], int numberUsed);
14  //Precondition: The first numberUsed indexed variables of a have values.
15  //Postcondition: Gives screen output showing how much each of the first
16  //numberUsed elements of the array a differs from their average.

(continued)
Partially Filled Arrays

Display 5.5  Partially Filled Array

```c++
int main( )
{
    int score[MAX_NUMBER_SCORES], numberUsed;

    cout << "This program reads golf scores and shows\n";
    cout << "how much each differs from the average.\n";

    cout << "Enter golf scores:\n";
    fillArray(score, MAX_NUMBER_SCORES, numberUsed);
    showDifference(score, numberUsed);

    return 0;
}
```
Partially Filled Arrays

```cpp
27  void fillArray(int a[], int size, int& numberUsed) {
28      cout << "Enter up to " << size << " nonnegative whole numbers.\n"
29         << "Mark the end of the list with a negative number.\n";
30      int next, index = 0;
31      cin >> next;
32      while ((next >= 0) && (index < size))
33      {
34          a[index] = next;
35          index++;
36          cin >> next;
37      }
39      numberUsed = index;
40  }
```
Partially Filled Arrays

```cpp
double computeAverage(const int a[], int numberUsed)
{
    double total = 0;
    for (int index = 0; index < numberUsed; index++)
        total = total + a[index];
    if (numberUsed > 0)
    {
        return (total/numberUsed);
    }
    else
    {
        cout << "ERROR: number of elements is 0 in computeAverage.\n"
             << "computeAverage returns 0.\n";
        return 0;
    }
}
```
Partially Filled Arrays

Display 5.5 Partially Filled Array

```c
57    void showDifference(const int a[], int numberUsed)
58    {
59        double average = computeAverage(a, numberUsed);
60        cout << "Average of the " << numberUsed
61            << " scores = " << average << endl
62            << "The scores are:\n";
63        for (int index = 0; index < numberUsed; index++)
64            cout << a[index] << " differs from average by "
65                << (a[index] - average) << endl;
66    }
```

**SAMPLE DIALOGUE**

This program reads golf scores and shows how much each differs from the average.

Enter golf scores:
Enter up to 10 nonnegative whole numbers.
Mark the end of the list with a negative number.

```
69  74  68  -1
```

Average of the 3 scores = 70.3333
The scores are:
69 differs from average by -1.33333
74 differs from average by 3.66667
68 differs from average by -2.33333
```
Multidimensional Arrays

- Arrays with more than one index
  - int matrix[30][100];
  - Two indexes: An "array of arrays"
  - Visualize as:
    matrix[0][0], matrix[0][1], ..., matrix[0][99]
    matrix[1][0], matrix[1][1], ..., matrix[1][99]
    ...
    matrix[29][0], matrix[29][1], ..., matrix[29][99]

- C++ allows any number of indexes
  - Typically no more than two
Multidimensional Array Parameters

- Similar to one-dimensional array
  - 1\textsuperscript{st} dimension size not given
    - Provided as second parameter
  - 2\textsuperscript{nd} dimension size IS given

```cpp
void DisplayPage(const char p[100][100], int sizeDimension1) {
    for (int index1=0; index1<sizeDimension1; index1++)
        for (int index2=0; index2 < 100; index2++)
            cout << p[index1][index2] << endl;
}
```