CSCI 123 Introduction to Programming Concepts in C++

Brad Rippe

Arrays
Overview

7.1 Introduction to Arrays
7.2 Arrays in Functions
7.3 Programming with Arrays
7.4 Multidimensional Arrays
Introduction to Arrays
Introduction to Arrays

• An array is used to process a collection of data of the same type
  – Examples: A list of names
  – A list of temperatures

• Why do we need arrays?
  – Imagine keeping track of 5 test scores, or 100, or 1000 in memory
    • How would you name all the variables?
    • How would you process each of the variables?
Declaring an Array

• An array, named score, containing five variables of type int can be declared as
  
  ```
  int intArray[5];
  ```

• This is like declaring 5 variables of type int:
  
  ```
  intArray[0], intArray[1], ... , intArray[4]
  ```

• The value in brackets is called
  
  – A subscript
  – An index
The Array Variables

• The variables making up the array are referred to as
  – Indexed variables
  – Subscripted variables
  – Elements of the array

• The number of indexed variables in an array is the declared size, or size, of the array
  – The largest index is one less than the size
  – The first index value is zero
Array Variable Types

• An array can have indexed variables of any type

• All indexed variables in an array are of the same type
  – This is the base type of the array

• An indexed variable can be used anywhere an ordinary variable of the base type is used
Using [ ] With Arrays

• In an array declaration, [ ]'s enclose the size of the array such as this array of 5 integers:

```java
int intArray[5];
```

• When referring to one of the indexed variables, the [ ]'s enclose a number identifying one of the indexed variables

  – `intArray[3]` is one of the indexed variables
  – The value in the [ ]'s can be any expression that evaluates to one of the integers 0 to (size -1)
Indexed Variable Assignment

• To assign a value to an indexed variable, use the assignment operator:

```c
int n = 2;
intArray[n + 1] = 99;
```

– In this example, variable `intArray[3]` is assigned 99
Loops And Arrays

- for-loops are commonly used to step through arrays

  - Example:

```cpp
for(int i = 0; i < 10; i++) {
    cout << "intsArray[" << i << "]=" << intsArray[i] << endl;
}
```

could display the difference between each score and the maximum score stored in an array

forArray.cpp
Initializing Element from cin

```cpp
char grades[5];

cout << "Please type in five grades (A,B,C,D, or F)\n";
// Arrays are indexed from 0...size-1
// This loop initializes all the values in the array
// by getting the value from the user
for(int i = 0; i < 5; i++) {
    cout << "Grade " << (i+1) << ": ";
    cin >> grades[i];
}
```
Constants and Arrays

- Use constants to declare the size of an array
  - Using a constant allows your code to be easily altered for use on a smaller or larger set of data

- Example:
  ```cpp
  const int NUMBER_OF_GRADES = 5;
  char grades[NUMBER_OF_GRADES];
  ...
  for ( i = 0; i < NUMBER_OF_GRADES; i++) {
    cout << "Grade " << (i+1) << ": ";
    cin >> grades[i];
  }
  ```

- Only the value of the constant must be changed to make this code work for any number of grades
Variables and Declarations

• Most compilers do not allow the use of a variable to declare the size of an array

Example: `cout << "Enter number of grades: "; cin >> size; int grades[size];`

– This code is illegal on many compilers
Array Declaration Syntax

• To declare an array, use the syntax:
  Type   arrayName[declaredSize];
  – Type can be any type (int, char, string, etc)
  – declaredSize can be a constant to make your program more versatile

• Once declared, the array consists of the indexed variables:
  arrayName[0] to arrayName[declaredSize -1]
Computer Memory

• Computer memory consists of numbered locations called bytes
  – A byte's number is its address

• A simple variable is stored in consecutive bytes
  – The number of bytes depends on the variable's type

• A variable's address is the address of its first byte
int intsArray[5];

for(int i = 0; i < 5; i++) {
    intsArray[i] = i*i;
}

for(int i = 0; i < 5; i++) {
    cout << "intsArray[" << i << "]=" << intsArray[i] << endl;
}
ints on this machine consist of 2 bytes of data. Therefore, the third intArray element begins at memory location 2004.
Arrays and Memory

• Declaring the array  `intsArray[5]`
  – Reserves memory for five variables of type int
  – The variables are stored one after another
  – The address of `intsArray[0]` is remembered
    • The addresses of the other indexed variables is not remembered
  – To determine the address of `intsArray[2]`
    • Start at `intsArray[0]`
    • Count past enough memory for two integers to find `intsArray[2]`
Array Index Out of Range

• A common error is using a nonexistent index
  – Index values for `int intsArray[5]` are the values 0 through 4
  – An index value not allowed by the array declaration is out of range
  – Using an out of range index value does not produce an error message!

  – Example:
    
    ```
    cout << intsArray[6] << endl;
    ```
Out of Range Problems

• If an array is declared as: `int intsArray[5]`;
and an integer is declared as: `int i = 7`;
• Executing the statement `intsArray[i]= 238`;
causes...
  • The computer to calculate the address of the illegal
    `intsArray[7]`
  • (This address could be where some other variable is stored)
  • The value 238 is stored at the address calculated for
    `intsArray[7]`
  • No warning is given!
char grades[5];

cout << "Please type in five grades (A,B,C,D, or F)\n";
for(int i = 0; i < 5; i++) {
    cout << "Grade " << (i+1) << ": ";
    cin >> grades[i];
}
chars on this machine consist of a byte of data. Therefore, the third grade begins at memory location 2002.
Initializing Arrays

• To initialize an array when it is declared
  – The values for the indexed variables are enclosed in braces and separated by commas

• Example: int children[3] = { 2, 12, 1 }; Is equivalent to:
  int children[3];
  children[0] = 2;
  children[1] = 12;
  children[2] = 1;
Default Values

• If too few values are listed in an initialization statement
  – The listed values are used to initialize the first of the indexed variables
  – The remaining indexed variables are initialized to a zero of the base type

Initializing

• Valid
  
  ```
  ```

• Invalid
  
  ```
  int intArray[3];
  intArray = { 3, 5, 11 };
  ```
Un-initialized Arrays

- If no values are listed in the array declaration, some compilers will initialize each variable to a zero of the base type
  - Visual Studio does not initialize each variable to a zero
  - When an array is created, its elements are assigned with arbitrary values.
  - DO NOT DEPEND ON THIS!
Section 7.1 Conclusion

- Can you
  - Describe the difference between a[4] and int a[5]?
  - Show the output of

    ```
    char symbol[3] = {'a', 'b', 'c'};
    for (int index = 0; index < 3; index++)
      cout << symbol[index];
    ```
7.2

Arrays in Functions
Arrays in Functions

• Indexed variables can be arguments to functions
  – Example: If a program contains these declarations:
    ```
    int i;
    int n;
    int a[10];
    void myFunction(int n);
    ```
  – Variables a[0] through a[9] are of type int, making these calls legal:
    ```
    myFunction( a[ 0 ] );
    myFunction( a[ 3 ] );
    myFunction( a[ i ] );
    ```
Arrays as Function Arguments

• A formal parameter can be for an entire array
  – Such a parameter is called an array parameter
    • It is not a call-by-value parameter
    • It is not a call-by-reference parameter
    • Array parameters behave much like call-by-reference parameters
Array Parameter Declaration

• An array parameter is indicated using empty brackets in the parameter list such as

```c
void getGrades(char grades[]);

void displayGrades(char grades[]);
```
Function Calls With Arrays

• If function `getGrades` is declared in this way:
  ```
  void getGrades(char grades[]);
  ```

• and array `grades` is declared this way:
  ```
  char grades[NUMBER_OF_GRADES];
  ```

• `getGrades` is called in this way:
  ```
  getGrades(grades);
  ```
Function Call Details

• A formal parameter is identified as an array parameter by the [ ]'s with no index expression

```java
void getGrades(char grades[]);
```

• An array argument does not use the [ ]'s

```java
getGrades(grades);
```
Array Formal Parameters

• An array formal parameter is a placeholder for the argument

  – When an array is an argument in a function call, an action performed on the array parameter is performed on the array argument

  – The values of the indexed variables can be changed by the function
Array Argument Details

• What does the computer know about an array?
  – The base type
  – The address of the first indexed variable
  – The number of indexed variables

• What does a function know about an array argument?
  – The base type
  – The address of the first indexed variable
Array Parameter Considerations

• Because a function does not know the size of an array argument...
  – The programmer should include a formal parameter that specifies the size of the array
  – The function can process arrays of various sizes
• Function `getGrades` from function `ArrayParam` can be used to initialize the values in the array an array of any size:

```c
char grades[NUMBER_OF_GRADES];
getGrades(grades);
char moreGrades[100];
getGrades(moreGrades);
```
Array parameters allow a function to change the values stored in the array argument.

If a function should not change the values of the array argument, use the modifier const.

An array parameter modified with const is a constant array parameter.

Example:

```c
void displayGrades(const char grades[]);
```
Using const With Arrays

• If `const` is used to modify an array parameter:
  
  – `const` is used in both the function declaration and definition to modify the array parameter

  – The compiler will issue an error if you write code that changes the values stored in the array parameter
Function Calls and const

• If a function with a constant array parameter calls another function using the const array parameter as an argument...

  – The called function must use a constant array parameter as a placeholder for the array

  – The compiler will issue an error if a function is called that does not have a const array parameter to accept the array argument
const Parameters Example

displayGrades(grades);
...
void displayGrades(const char grades[]) {
    for(int i = 0; i < NUMBER_OF_GRADES; i++) {
        cout << "Grade " << (i+1) << " = "
            << static_cast<char>(toupper(grades[i])) << endl;
    }
    cout << "The average grade is " << averageGrades(grades) << endl;
}

- averageGrades has no constant array parameter
- This code generates an error message because averageGrades could change the array parameter
Constants as Arguments

• The book uses:

• When function `displayGrades` or `averageGrades` is called, `NUMBER_OF_GRADES` could be used as an argument
  – Can't `NUMBER_OF_GRADES` be used directly without making it an argument?
    • Using `NUMBER_OF_GRADES` as an argument makes it clear that `displayGrades` and `averageGrades` requires the array's declared size
    • This makes `displayGrades` and `averageGrades` easier to be used in other programs
Returning An Array

• Recall that functions can return a value of type int, double, char, ..., or a class type

• Functions **cannot** return arrays

• We learn later how to return a pointer to an array
Section 7.2 Conclusion

• Can you

  – Write a function definition for a function called oneMore, which has a formal parameter for an array of integers and increases the value of each array element by one. Are other formal parameters needed?
7.3

Programming with Arrays
Programming With Arrays

• The size needed for an array is changeable
  – Often varies from one run of a program to another
  – Is often not known when the program is written

• A common solution to the size problem
  – Declare the array size to be the largest that could be needed
  – Decide how to deal with partially filled arrays
Partially Filled Arrays

• When using arrays that are partially filled
  – Functions dealing with the array may not need to know the declared size of the array, only how many elements are stored in the array
  – A parameter, size, may be sufficient to ensure that referenced index values are legal

  displayInts (const int aArray, int size);
Searching Arrays

• A sequential search is one way to search an array for a given value
  – Look at each element from first to last to see if the target value is equal to any of the array elements
  – The index of the target value can be returned to indicate where the value was found in the array
  – A value of -1 can be returned if the value was not found
The search Function

• The search function in `sequentialSearch.cpp`
  – Uses a for loop to compare array elements to the target value
  – Looks at each element in `theArray` to see if it matches the value of `find`
  – If the value of `find` is in the array, the index of `find` is returned
  – If `find` is not in `theArray`, -1 is returned

```cpp
int searchArray(const int theArray[], int currentSize, int find)
```
Program Example: Sorting an Array

• Sorting a list of values is very common task
  – Create an alphabetical listing
  – Create a list of values in ascending order
  – Create a list of values in descending order

• Many sorting algorithms exist
  – Some are very efficient
  – Some are easier to understand
Program Example: The Selection Sort Algorithm

• When the sort is complete, the elements of the array are ordered such that

\[ a[0] < a[1] < \ldots < a[\text{numberUsed} - 1] \]

– This leads to an outline of an algorithm:
  
  ```
  for (int index = 0; index < numberUsed; index++)
    place the index\text{th} smallest element in a[index]
  ```
Program Example:
Sort Algorithm Development

• One array is sufficient to do our sorting
  – Search for the smallest value in the array
  – Place this value in a[0], and place the value that was in a[0] in the location where the smallest was found
  – Starting at a[1], find the smallest remaining value swap it with the value currently in a[1]
  – Starting at a[2], continue the process until the array is sorted
Selection Sort

\[
\begin{array}{cccccccccc}
\text{a[0]} & \text{a[1]} & \text{a[2]} & \text{a[3]} & \text{a[4]} & \text{a[5]} & \text{a[6]} & \text{a[7]} & \text{a[8]} & \text{a[9]} \\
8 & 6 & 10 & 2 & 16 & 4 & 18 & 14 & 12 & 20
\end{array}
\]

1. Select the minimum element from the array and put it in the first position.

\[
\begin{array}{cccccccccc}
\text{2} & 6 & 10 & 8 & 16 & 4 & 18 & 14 & 12 & 20
\end{array}
\]

2. Repeat the process for the remaining elements.

\[
\begin{array}{cccccccccc}
\text{2} & 6 & 10 & 8 & 16 & 4 & 18 & 14 & 12 & 20
\end{array}
\]

3. Continue until the array is sorted.

\[
\begin{array}{cccccccccc}
\text{2} & \text{4} & 10 & 8 & 16 & 6 & 18 & 14 & 12 & 20
\end{array}
\]
int main() {
    const int arraySize = 10;
    int a[arraySize] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 }
    selectionSort(a,arraySize); // sort the array
    cout << "\nData items in ascending order\n"

    for(int j = 0; j < arraySize;j++) {
        cout << setw( 4 ) << a[j];
    }

    cout << endl;
    return 0;
}
void selectionSort(int theArray[], int size) {
    int smallest; // index of smallest element
    // loop over size - 1 elements
    for(int i = 0; i < size-1; i++) {
        smallest = i; // first index of remaining array
        // loop to find index of smallest element in the array
        for(int index = i+1; index < size; index++) {
            if(theArray[index] < theArray[smallest])
                smallest = index;
        }
        swap(theArray[i], theArray[smallest]);
    }
}

void swap(int& element1, int& element2) {
    int temp = element1;
    element1 = element2;
    element2 = temp;
}
Section 7.3 Conclusion

• Can you

  – Write a program that will read up to 10 letters into an array and write the letters back to the screen in the reverse order?

  abcd should be output as dcba

  Use a period as a sentinel value to mark the end of input
7.4

Multidimensional Arrays
Multi-Dimensional Arrays

- C++ allows arrays with multiple index values
  - char page [30] [100];
    declares an array of characters named page
  - page has two index values:
    - The first ranges from 0 to 29
    - The second ranges from 0 to 99
  - Each index is enclosed in its own brackets
  - Page can be visualized as an array of
    30 rows and 100 columns
Index Values of page

• The indexed variables for array page are
  page[0][0], page[0][1], ..., page[0][99]
  page[1][0], page[1][1], ..., page[1][99]
  ...
  page[29][0], page[29][1], ..., page[29][99]

• page is actually an array of size 30
  – page's base type is an array of 100 characters
Multidimensional Array Parameters

• Recall that the size of an array is not needed when declaring a formal parameter:
  void displayGrades(const char a[], int size);

• The base type of a multi-dimensional array must be completely specified in the parameter declaration

  void displayPage(const char page[][100],
                   int pgSize);
Program Example: Grading Program

• Grade records for a class can be stored in a two-dimensional array
  – For a class with 7 students and 5 quizzes the array could be declared as

    int grade[7][5];

    • The first array index refers to the number of a student
    • The second array index refers to a quiz number

• Since student and quiz numbers start with one, we subtract one to obtain the correct index

multiArray.cpp
& multiArrayWFunction.cpp
Section 7.5 Conclusion

• Can you

  – Write code that will fill the array `a` (declared below) with numbers typed at the keyboard? The numbers will be input five per line, on four lines.

    ```c
    int a[4][5];
    ```