Laboratory 12  More Arrays

12.1 Objectives

- To understand reference parameters thoroughly
- To be able to process arrays using loops
- To be able to perform common functions of an array
- To be able to process only part of an array

12.2 Prerequisites

You should thoroughly understand for loops in order to be able to do this lab.

12.3 Correspondence to Text

The material in this lab best corresponds to sections 9.3-9.5 in the text.

12.4 Introduction

In this lab, you will get more practice on arrays, see some common functions (e.g., finding max or min) that are performed on arrays, and get some extra practice with reference parameters.

12.5 Review of Reference Parameters

You have already learned the difference between value and reference parameters, but this lab requires that you thoroughly understand them, so we’ll review a little.

Reference parameters access the variable in the calling function; they are actually pointers back to the value of the actual parameter in the calling function. Whenever the value of a reference parameter is changed, the value of the variable (that is the actual parameter in the calling function) is changed as well. Value parameters copy the value of an actual parameter into the parameter in the function. Whenever the value of a value parameter is changed, the value of the variable (that is the actual parameter in the calling function) is not affected.

- Answer questions Reference Parameters in lab 12.

12.5.1 Syntactic Difference

For simple variables (i.e., variables that are not array variables), in order to pass a variable by reference, you use the & symbol; in order to pass a variable by value, you use nothing.

Thus a prototype for a function swap that has two char parameters, both passed by reference is: void swap(char&, char&);
The function header for this function is: `void swap(char &x, char &y)`

When the parameters (x and y in this case) are used in the calling and called function, both reference and value parameters are treated in exactly the same manner. The only thing different about reference and value parameters syntactically is the way that the parameters are passed. There is no way to tell when looking at a parameter’s use in an executable statement whether it is a variable, a value parameter, or a reference parameter.

Array parameters are passed by reference even though you do not use the & in the parameter list. This avoids copying the array into each function that the parameter is passed to.

### 12.5.2 When to use which

The following table describes some situations when you would want so use value and reference parameters.

<table>
<thead>
<tr>
<th>Function Purpose</th>
<th>Parameters</th>
<th>Returns</th>
<th>Pass by</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Max of 2 Numbers</td>
<td>2 Numbers</td>
<td>Max of 2</td>
<td>value</td>
<td>Numbers are not modified</td>
</tr>
<tr>
<td>Find Max of Numbers in Array</td>
<td>1 Array</td>
<td>Max nbr</td>
<td>value</td>
<td>Always pass by value if possible</td>
</tr>
<tr>
<td>Place Max of 2 nbrs in 2nd Nbr</td>
<td>2 Numbers</td>
<td>None</td>
<td>reference</td>
<td>Arrays always passed by reference</td>
</tr>
</tbody>
</table>

- After you understand the table, answer **When to Use Which** in lab 12 questions.

### 12.6 Typical Array Problems

This section looks at some typical array processing. Make sure that you fully understand all of these loops and what they do. These examples also demonstrate how to pass entire arrays and how to pass elements of an array. Entire arrays are always passed by reference, but individual elements are passed by value or by reference depending on their purpose in the function.

To pass an entire array, use the variable name for the array with no indices. To pass individual elements of the array, use the name of the array with the appropriate index. For example, assume that we have an array `sample`. Also assume that there is a function, `sortels` which requires the first element of an array as its first parameter, and the entire array as its second. To call `sortels` with the first element of `sample` and the entire array `sample` would be: `sortels(sample[0], sample)`
12.6.1 Finding the minimum element

- Copy the program min12.cpp. This program prints the minimum element in sample. It is similar to lab11.cpp in the way that the array is read and written, but these processes have been moved to functions. As a result, the const declaration that declares the size of the array has been moved outside the scope of main so that it is global and can be accessed by all functions.

One other detail to notice is that the array is passed as a const parameter when it will not be modified by the function. All arrays are passed by reference rather than by value, but if the array is not going to be modified in the function, it is a good idea to use a const before the argument. To do this, write const before the formal parameter in both the function prototype, and in the function definition (the function header). This has been done for writearray because this function does not modify the array; readarray on the other hand does modify the values in the array, so it is not passed as a const parameter.

- Add the required read statements, and compile and test it.

- Copy min12.cpp to max.cpp, and modify it so that it finds the maximum element. Again, do not change any output statements, except anywhere where the old program has the word “minimum”, your program should have the word “maximum” (and you may want to change the names of the variables that are output).

- Test the program with testlab10 (if you have deleted testlab10, you can copy it again).

12.6.2 Reversing elements in an array

- Copy program reverse.cpp. This program reverses the elements in an array.

  Look at function reversearray. The array is reversed by swapping element 1 with the last element, element 2 with the next to last element, etc. The loop only goes half way through the array. Why is this?

  Notice how the array elements are passed to the function swap. Because the value of the array elements are modified, the array elements are passed by reference. But there is nothing different about passing array elements than passing simple variables. The swap function would be identical whether array elements or simple variables were passed.

  There are several tricky parts about the read function as well. Look at readarray. This is the first time we’ve used an array where we may use less than the entire array. The largest word that this program can handle is 10, but we might enter a word with less than ten characters (e.g., “word” or “dad”). We would like the program to function correctly, but if we reversed “word” in an array of ten characters and used all of them the reversed array would be:

  \[
  \begin{array}{cccccccccc}
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
  \end{array}
  \]

  instead of what we’d like:
Therefore, when reading in values where we might not need to use the entire array, we often return a length value that is used in subsequent functions. In this case, we determine the length of the array by looking for a return character. We need to use the function `cin.get` because `cin` skips over white space, and the return character is considered white space. The word is read, and then the length of the word is passed to the other functions in the program.

Finally, the last if statement increments i by one if we went up to the maximum number of characters. This is because when we're doing the comparison in the for statement, we're actually reading one character too many and then throwing it away for any word that is less than the maximum number of characters. That is, if we read “word\n” (5 characters) what we want in the array is “word” (4 characters) so the length that we return is the length before we read \n. But if we read 10 characters before the \n, then the first ten characters will be treated as the word, and we don't want to throw any of those away because we haven't read an extra \n. In order to not throw away an extra character if we read the maximum number of characters, we increment i (to arraysize + 1) so that when i is decremented in the return statement (throwing away the \n), it will decrement i back to where it should be (to arraysize). If you do not understand this subtlety fully, it will not hinder your ability to do the rest of the lab; I explained it only for those who are interested in the details of this function.

- Compile and test the program using the string “word” and using the string “dad”

Because the string “dad” is the same backwards and forwards, you will not see any difference when the array is reversed. “dad” is called a palindrome because it is the same backwards and forwards. Let's write a program to determine if a word is a palindrome. There are many ways to do this, but one way is to copy an array to another array, reverse the second array, and then compare the two. If the forwards and reversed arrays are equal, the word is a palindrome; otherwise it isn’t.

- Copy program pal.cpp.

- This program is similar to reverse.cpp, but an extra function has been added: ispalindrome. ispalindrome has three functions, two of which you need to write: copyarray which should copy the first array to the second, and equalarrays which should compare the individual elements of the two arrays. The function ispalindrome is written, and should not be modified; just write the two functions declared in ispalindrome that are not already written.

Both of these process only length number of elements; they should not examine the entire array.

Write the two functions copyarray and equalarrays. Do not add any couts to the program. copyarray should simply assign all of the elements from 0 to length to the second array; equalarrays should check the elements from 0 to length, and return 1 if the arrays are equal; 0 if they are not.

- Save, compile, and test your program.
12.7 Synthesis

Write a program to calculate your grade average for your labs, and print the highest and lowest grade. Use an array to hold your 14 lab grades; the array elements should be of type float. Print out your highest, lowest, and average grades. I have written the cout statements for you as part of a main function. Because we want to be able to keep a running average, it should be possible to enter less than 14 grades (e.g., this week, you would enter 12 because there will be 12 labs that you have completed).

- Copy the program labgrades.cpp.
- Write the functions for this program (I have written readgrades for you). Do not add any cout statements or modify those that exist.
  To run the program, you will enter all 14 lab grades or as many lab grades as you have followed by a -1 indicating that there are no more lab scores.
- Save and compile the program.
- Copy the file labgrades, and use this file to test your program (redirect your input).
- Run the program. Notice the -1 indicating that there are no more lab grades to read. Test the program with this file (redirecting the input).
  This is what your output should be:

  This program calculates an average, lowest, and highest for up to 14 grades.
Enter a list of your grades (14 or fewer): Your average grade is 85.9
  Your lowest grade is 67 and your highest grade is 100