Laboratory 10  Arrays

10.1 Objectives

• To be able to declare and process arrays

10.2 Prerequisites

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

10.3 Correspondence to Text

The material in this lab best corresponds to sections 5.3, 7.2, 9.1-9.3 in the text.

10.4 Introduction

Up until now, all of the variables that we have used have been stored in single 'boxes' unrelated to each other. This prevents us from doing many kinds of processing. For example, one common problem in Computer Science is sorting a list of numbers (e.g., to print out a list of all grades of all students by student ID). It is relatively easy to sort two numbers. If the variables x and y hold two numbers, the following code will print out a sorted list of x and y:

```cpp
if (x < y)
    cout << "The sorted list is " << x << y;
else
    cout << "The sorted list is " << y << x;
```

If we have 3 numbers, x, y, and z, we could do the following:

```cpp
if (y < x)
    swap(x, y);
if (z < x)
    cout << "The sorted list is " << z << x << y;
else if (z > y)
    cout << "The sorted list is " << x << y << z;
else
    cout << "The sorted list is " << x << z << y;
```

This code assumes that a function swap exists which swaps the values of the arguments (e.g., if `x = 4` and `y = 2`, after `swap(x,y)`, `x = 2` and `y = 4`).

To sort 4 numbers, x, y, z, and w, we could do the following:

```cpp
if (y < x)
    swap(x, y);
```
if (z < y)
{
    if (z < x)
    {
        swap(x, z);
        if (z < y)
            swap(y, z);
    } else
        swap(y, z);
}
if (w < z)
{
    if (w < x)
        cout << "The sorted list is " << w << x << y << z;
    else if (w < y)
        cout << "The sorted list is " << x << w << y << z;
    else
        cout << "The sorted list is " << x << y << w << z;
} else
    cout << "The sorted list is " << x << y << z << w;

And after that, life gets complicated! One problem is that there is really no way to make
this comparison process repetitive or to simplify it in any way. Another problem is that many
times programs that sort numbers (e.g., student numbers for reports each semester) may have a
different number of numbers each time they run the program (i.e., there may be 8000 students
in the Fall, but only 6000 in the Spring). What do we do with those 2000 extra variables that
aren’t being used? We can’t just input the number of students (6000 or 8000), and then tell the
program to ignore 2000 variables – how would we do that? Also, it’s going to be a pain declaring
8000 different variables to hold the student numbers.

There are other problems, too. The bottom line is that we need something else that will
allow us to hold chunks of similar data in one place, and that will allow us to do some loop
processing with these chunks of data. The mechanism for this ‘something else’ is called an array.

10.4.1 What is an array?

An array is a collection of a fixed number of values all of the same type. You might think of an
array as a collection of identical drawers in a dresser (that are very small because they can only
hold one item); each drawer can hold the same type of item and the drawers can be identified as
the first drawer, the second drawer, etc.

The identification of a drawer is called its index. Indices in C++ start with 0. So the first
‘drawer’ (or slot) in an array is indexed by 0. Because the index is 0, many people refer to this
as the ‘zeroth’ drawer. If there are 10 drawers, the indices will range from 0 to 9, with the last
drawer having an index of 9.

The item inside the drawer or slot is referred to as an element. For example, an array of ten numbers might look like the box below; the elements are in the boxes and the indices are beneath the boxes (this is a typical way to represent an array):

```
<table>
<thead>
<tr>
<th>3.2</th>
<th>-234.3</th>
<th>23.0</th>
<th>2434.234</th>
<th>3.2</th>
<th>-2.0</th>
<th>89.9</th>
<th>12.0</th>
<th>67.22</th>
<th>66.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
```

In the above array, there are ten elements that are of type float and ten indices (of type int). The element that corresponds to index 4 is 3.2; the index corresponding to the element 66.0 is 9.

10.5 Declaring arrays

To declare the above array, and name it sample, you would use the following:

```c
float sample[10];
```

This would declare an array named sample that has ten slots that will be filled with values of type float, and the indices for this array will range from 0 to 9.

Suppose that we wanted to do the count vowel processing that we did last lab, but we don’t want to discard every character after it is read (we may want to do other processing with the characters). We could declare an array of characters that would hold one line of data (one line is 80 characters):

```c
char line[80];
```

This would declare an array named line that would hold 80 characters in index 0 through index 79, all of type char.

- Answer questions Declaring Arrays in lab 10.

10.6 Referencing Array elements

Now that we know how to declare arrays, we need to know how to get the value of an element stored at a particular index. This is done by subscripts, written in square brackets. To refer to the element in the index 2 of the array sample, we would write:

```c
sample[2]
```

To refer to this element, we would say (in words) “sample sub two.” Just as referring to variables can be used for storing or for retrieving elements, array referencing can be used to store elements in index 2 or to get the value of the array element stored at index 2. For example,

```c
sample[2] = sample[3];
```
would store the value 2434.234 (sample[3]) in the slot indexed by 2 (sample[2], the slot that previously held 23.0).

Variables may be used in subscripting as well. If i is an int with the value 3,

```cpp
sample[i] = sample[i+2];
```

would store the value -2.0 (i + 2 = 5, sample[5] = -2.0) in the slot indexed by 3 (where 2434.234 used to be). The array would then look like this (including the earlier change assigning sample[3] to sample[2]):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>2434.234</th>
<th></th>
<th>2434.234</th>
<th>3.2</th>
<th>4.3</th>
<th>2.0</th>
<th>3.2</th>
<th>5.0</th>
<th>6.0</th>
<th>7.0</th>
<th>8.0</th>
<th>9.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`sample[3*i] = sample[10]` would result in an error since `sample[10]` is outside the range of the array (`sample[3*i]` is fine; if i=3, `sample[9]` would be the slot referenced). The C++ compiler will not give you an error, but you will get unpredictable results if you try to access values outside the range of the array.

- Do questions Referencing Arrays in lab 10.

10.7 For loops

For loops are commonly used with array processing. for statements are a variant of the while statement. They have an initialization, a condition, and an increment all separated by semicolons and all in parentheses. The initialization of the loop is done first, then the condition. If the condition evaluates to true, the loop is entered; otherwise, the loop is skipped. If the loop is entered the statements in the loop are performed, and then the increment statement. After the increment, control passes back to the comparison. If the comparison evaluates to true, the loop is entered again; otherwise, control passes to the first statement past the for statement.

The following two loops are equivalent; both print out the numbers 0-9:

```cpp
for (i=0; i<10; i++)
   cout << i;
while (i < 10)
{   
   cout << i;
   i++;
}
```

10.7.1 Initializing an array

One big advantage of arrays is that you can do something to each element within a loop. For example, you might want to initialize all of the elements in sample to 0. The following loop will do that:

```cpp
i = 0;
for (i=0; i<10; i++)
   cout << i;
while (i < 10)
{   
   cout << i;
   i++;
}
```
for (i=0; i<10; i++)
    sample[i] = 0.0;

This loop initializes i to be 0 (the first part of the parenthesized for expression). It then performs the i < 10 comparison (the second part of the parenthesized for expression). i (which is 0 at this point) is less than 10, so the for loop body is entered. The body assigns 0.0 to sample[i] (which is at this point sample[0]). Thus the array looks like this after the first time through the loop:

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

The loop completes, and goes up to the top of the for, and increments i by 1 (the third part of the parenthesized for expression: i++). i is now 1. Next, the i < 10 comparison is performed again: (the second part of the parenthesized for expression). Since i (now 1) is less than 10, the loop is entered again. sample[i] again gets the value 0, but this time sample[i] is sample[1]. The array now looks like:

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

The loop continues until all elements in the array from index 0 to index 9 have been assigned the value 0.0.

- Copy the program `lab10.cpp`. This program should read in ten numbers into the array `sample` and write them out.
- Add the for and cout statements to write out the numbers (you can look at the cin statement as an example).
- Save and compile the program.
- Copy the file `testlab10`.
- Test `lab10.cpp` redirecting the input from `testlab10`.

### 10.8 Casting

And now for something completely different ... 

Data in C++ is typed, but that type may be changed. There are many reasons for doing this; I just wanted to introduce the notion to you in this lab. As you know, characters are stored in ASCII format on most machines. That means that an 'A' inside the machine is stored as a 65 (or, actually, the binary equivalent: 1000001). It is possible to treat this 'A' as an integer and not as a character (that is, look at the 65 and not at the 'A' interpretation). To do this, you can coerce the character into being an integer.

We have actually done this before. Whenever we add two numbers and one is of type float and one is of type int, the int is coerced into being a float type so that the two can be added. This is called coercion.
The following code illustrates this:

```cpp
int x;
float y, z;

x = 3;
y = 23.345;
z = x + y;
```

Here, `x` is being implicitly coerced into being a `float`. It is possible to do this explicitly as well. Below are the three ways to do this:

```cpp
// Method 1
int x;
float y, z;

x = 3;
y = 23.345;
z = (float)x + y;

// Method 2
int x;
float y, z;

x = 3;
y = 23.345;
z = float(x) + y;

// Method 3
int x;
float y, z;

x = 3;
y = 23.345;
z = static_cast<float> (x) + y;
```

All three of these do the exact same operation; there are just different ways of doing it. Explicit casting is preferred to implicit coercion because it is easier to see what is happening to a person reading the program. In Standard C++, the last method is preferred over the other two because it explains the type of cast being performed.

- Answer questions Coercion in lab 10.
10.9 Synthesis

- Write a program that reads in 10 real numbers (floats), multiplies each by 10, and outputs the results. Name the program `add10.cpp`.
- Use an array to store the 10 numbers as you read them in.
- Compile your program.
- Test your program using the file `testlab10`. Redirecting the input from `testlab10`, the output should look like:

```
Enter 10 numbers> The array elements multiplied by 10 are:
  123
 -342
  0
  2345
  122
  456784
 -234
 -33580
  19864
  3780
```

If you have extra time, you might try adding up all of the numbers in the array and output the total sum also. For example, if you redirect the input from `testlab10` again, the output would be:

```
Enter 10 numbers> The array elements multiplied by 10 are:
  123
 -342
  0
  2345
  122
  456784
 -234
 -33580
  19864
  3780
Total Sum: 448952
```