Lab 7: Arrays

So far we have only dealt with simple variables that can hold only a single value, e.g. \( x = 7 \). Arrays allow us to store a list of values and to easily access members of that list.

Like variables (and functions), arrays have to be declared before they can be used. The declaration of an array defines both the name of the array and how many values it can store. (Like variables, arrays must be declared before they are used so that when the program is executing the computer knows how much memory to set aside for storing the values and what type the values are to be treated like.)

The declaration of an array to hold 100 integer values would be:

```c
int LongArray[100];
```

where the name of the array could be any valid variable name. One tricky feature of C/C++ arrays is that the values are numbered from 0 to one less than the array size. So, the 50th element of the array is `LongArray[49]`. For example you would print the 50th element with the command:

```c
cout << LongArray[49];
```

or you would set the value with the statement:

```c
LongArray[49] = 7;
```

You can also use an integer variable in the place of the value in the above commands. For example the statement:

```c
cout << LongArray[j];
```

prints the \( j \)th element of the array; where \( j \) should be an integer variable that was defined and given a value earlier in the program.

When you ask for the \( j \)th element of an array the computer actually begins at the start of the array and then takes \( j \) steps through memory to find the element you asked for. (The size of the steps depends on the type of the array, an array of doubles (or floats) requires larger steps than an array of integers, but we don’t need to worry about that now because it happens automatically.) However, C and C++ don’t do any bounds checking with arrays! If you define an array with 50 elements and then ask for element number 67 the computer will start at the beginning of the array and take 67
steps and return the value it finds—even though there’s only 50 elements in
the array. The value you get back will not be what you expected.

You can also ‘step’ beyond the memory allocated (set aside) for your
array (or program). For example, if you ask for element 1,000,000 from an
array with only 10 elements you may exceed the memory that the operating
system has set aside for your program to use. In this case the operating
system steps in, because you’re effectively trying to access the memory of a
different program, and halts your program—printing a cryptic error message
in the process.

Exercise 7.1

This is a simple exercise to introduce the use of arrays and some of the
common errors that occur when programming with them.

• Write a simple program that allows the user to enter 10 integers into
an array, then prints the array and calculates, stores and prints the
sum and average of the values in the array. Print this program to be
turned in, along with sample output.

• Add a line to the end of the program that prints the 11th element of
the array. What gets printed when you run the program? Try running
the program a few times. Is there any consistency in what gets printed
for the 11th element?

• Add a line at the end of the program to print the 10,000th element of
the array. What happens now? What is the exact error message you
get?

• Turn in your original program and the results of the last two experi-
ments.