CSC 204 Lab 11: Intro to Arrays

Goals

After doing this lab, you should be able to:

- declare and trace a one-dimensional array of simple data types
- work with arrays of objects
- use the `for` statement to read in, process, and print out an array
- complete simple methods involving array processing

Lab Preparation

Begin reading the material in Chapter 7.

Materials Needed

Be sure you have the following on hand during the lab.

- This sheet and your course notebook.
- The files for Lab 11 downloaded from `http://theochem.mercer.edu/csc204`.

Working with One-Dimensional Arrays

1. Let's consider the Java program `BegArray` that you copied over and is shown on your reference sheet. Notice how it declares an array of size 5 using `new`. It will initialize each of these five slots to zero automatically for you. Draw yourself a picture of array `a` below. You can draw it either horizontally or vertically.

Now, let's trace through this program and determine how it will initialize the slots in your array above. Notice how we can call the private instance variable `length` of an array. This variable contains an array's declared size.

2. Compile and test your program to see if your array is indeed correct. If it is, edit your program and modify the boolean expression in the first for loop to read `k < a.length + 1`. Recompile and execute your program. Give the first line of this error message below.

Going out of bounds on an array is a very common run-time error in a program. You want to be very careful to not go past the defined size of your array!

3. Let's build our own one-dimensional array now and initialize it with random integers. Create a program named `RandomArray`, and declare an array `random` of 50 integers. Now, let's set up a

`for` loop that will assign a random integer between 9 and 50 inclusively. So, we'll need to use a `Random` variable inside our loop. Think about the expression that you'll use. Hint : You want to get an integer in the range [0 , 41), and then add on a displacement of 9. Remember to cast your result to an integer.

Once your array has been initialized, let's print it out nicely spread over five lines. For example,

```
36 29 46 22 33 33 41 13 17 40
 13 47 16 50 9 49 32 17 47 38
28 27 29 36 15 44 31 44 34 27
14 12 23 35 18 44 48 26 32 26
31 22 48 31 26 20 45 27 11 15
```

Fancier One Dimensional Arrays

4. Let's trace through the program `Parallel` now that is shown on your reference sheet. Here we have four different parallel arrays. These are arrays with the same size that store different types of data. Remember an array is homogenous data structure which means we can't mix our data types within it. But, we can easily declare multiple arrays of the same size whose different elements can be accessed "in parallel."

Draw and initialize your four parallel arrays below. Notice how `new` is not needed when you give your array a set of initial values. Also, check out how we can even have an array of `Fraction` objects! What output to you expect for this program? Check your answer once you're done.

5. We're now ready to try out `CrazyArray.java`. This program uses indirect referencing of arrays. That is, it uses an array value as an array index. You'll want to draw yourself a picture of the array `a` in this one, and trace through this program very slowly. Use arrows to help yourself.

Compile and execute `CrazyArray.java` and check your answers found during the trace.

Completing Methods with Array Parameters

1. Edit file `MyArray.java`. Notice how the main method is already complete. You will want to complete the bodies of four methods involving arrays.

2. First off, let's write a void method named `fillUp` which can be used to fill up an array of six items with integers that are input from the keyboard. Notice how this function takes an array `a` as a parameter. You just need to add a for loop here that iterates six times and each time prompts the user to insert an integer. Your code should then get their integer (as a string) and convert it into an integer to be added to the array. A sample input is printed on the back of this sheet.

3. Our next method call will be to a void method named `printOut` which is used to print out all of the contents of our array `a` on the same line. It should be set up very similar to method `fillUp`. Include an output prompt as shown on the back.

4. Your third method will be a value-returning method named `sumUp` which takes as input our array `a` and returns to the calling function the sum of the items in this array. Once again, you will need to use a `for` loop. Don't forget to flush your local sum variable to zero prior to this loop. Also, since this method is "value-returning" we need to include a return statement at the end of it that returns the integer value representing the sum of our array. For example, `return sum;` .

5. Your fourth and final method is a value-returning method named `posCount` which takes as input our array `a` and returns to the calling function the number of integers in the array which are positive.

You will need to declare a local count variable and remember to set it to zero prior to your loop. Within your loop, use a single-alternative `if` statement to check and see if an array item is positive. Once again, you'll also need a `return` statement.

6. Compile and test your program using the sample input on the back as well as some sample input of your own.

Sample Input

```
Enter value 1:  -5
Enter value 2:  9
Enter value 3:  20
Enter value 4:  30
Enter value 5:  40
Enter value 6:  50
```

Sample Output

```
*** PrintOut of Array ***
-5 9 20 30 40 50
************************

Array Sum :  145

Num of Positive Values :  5
```

REFERENCE SHEET

```
import java.util.*;
public class BegArray
{
        public static void main(String[] args)
        {
                int[] a = new int[5];

                for (int k = 0;  k < a.length;  k++)
                        a[k] = k * k;

                a[ 2 + 4 / 3 ] = -1;

                for (int k = 0;  k < a.length;  k++)
                        System.out.println(a[k]);
         }
}
```

```
import java.util.*;
public class Parallel
{
  public static void main(String[] args)
  {
        char[] a = {'C', '$', 'd', 'Z'};
        boolean[] b = {false, false, true, false};
        String[] c = {"baseball", "basketball", "golf", "tennis"};
        Fraction[] d ={new Fraction(1,2), new Fraction(2,3),
                       new Fraction(-2,5), new Fraction(12,25)};
        for (int k = 3;  k >= 0;  k--)
        {
                System.out.print(a[k] + " " + b[k] + " " + d[k] + " ");
                if (c[k].compareTo("basketball") < 0)
                        System.out.println("Before basketball");
                else if (c[k].compareTo("basketball") > 0)
                        System.out.println("After basketball");
                else
                        System.out.println("Hoops Time!");
        }
  }
}
import java.util.*;
public class CrazyArray
{
        public static void main(String[] args)
        {
                int[] a = {3, 72, 0, 41, -23, 64};
                int i = 2;
                a[ a[i] ] = a[ a[i] ] - 1;
                int j = a[i];
                System.out.println( a[ 2 + a[j] ] );
```

```
        }
    }
```