

WES-CS GROUP MEETING #8

Exercise 1: Code Tracing

The code segment shown below was supposed to set the *characters* array to hold the characters

```
a * b * c * d * e * f * g * h * i * j *
```

However, the code does not work correctly.

```
char [] characters = new char [20];
for (int x = 0; x < characters.length; x++) {
    characters[x] = '*';
}

char a = 'a';
for (int x = 0; x < characters.length; x++) {
    while (a != 'k') {
        if (x%2==0) characters[x] = a;
        a = (char)(a + 1);
    }
}
```

First, trace the code to find out what the final values in the *characters* array actually are.

Then, fix the code so that it works as intended.

Exercise 2: Arrays

For this exercise you will use the blue and pink strips of paper. The blue strips have code fragments, each of which does something to an array called `arr` (in each case, assume that `arr` is a non-empty, initialized array of `int` values). The pink strips have English descriptions of what the code fragments do. For each blue strip, find a matching pink strip (there will be some pink strips left over).

Exercise 3: Nested Loops

The six pictures shown below were printed by a simple Java program.

```
*
**
***
****
*****
*****
*****
*****

*
***
*****
*****
*****
*****
*****
*****

*****
*****
*****
*****
*****
*****
*****
*****

*****
*
*
*
*
*
*
*****

*****
**
* *
* * *
* * *
* * *
* * *
*****

*****
**
* *
* * *
* * *
* * *
* * *
*****
```

The first picture was printed by the program shown below.

```
public class Loops {
    public static void printDesign1( int n ) {
        // print n rows of increasing length
        for (int row=0; row<n; row++) {
            for (int col=0; col<row+1; col++) {
                System.out.print('*');
            }
            System.out.println();
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int n = 7;
        printDesign1( n );
    }
}
```

Trace the code to understand how it works. Then choose one or more of the other pictures, and add new methods that print the pictures you've chosen. (Make sure that your methods work for any value of n greater than 0.)

Now design your own picture (some pattern of stars in an n -by- n box) and ask your neighbor to code it.

Exercise 4: A Program with a Menu

For this exercise, you will write a program that displays the favorite color, or the birthday or the number of siblings of someone in your group.

Here's an example of what will happen when you run the program.

- First, the program will display:

```
Whose information would you like to access?  
Enter 1, 2, 3, or 4 to choose a person, or -1 to quit:  
1) Bob  
2) Jill  
3) Mike  
4) Ashley  
Enter your choice now:
```

- After you enter 1, 2, 3, or 4, the program will display:

```
What would you like to know about <name of person>?  
Please select one of the following:  
A) Favorite color  
B) Birthday  
C) Number of siblings  
Enter your choice now:
```

- After you enter A, B, or C, the program will display the desired information; for example, if you chose '2' for Jill and 'A' for favorite color, the program will print Jill's favorite color.

- The program should continue to ask you to choose a person and then a piece of information, until you type -1.

Part (a): Start by defining a *Person* class to store one person's name, favorite color, birthday, and number of siblings (you might consider several different ways to represent the birthday).

Write a constructor that has four arguments (the four pieces of information about the person). Also write accessor methods that return the name, favorite color, birthday, and number of siblings.

Write a main method to test your *Person* class. Create an array of *Person* objects, one for each person in your group, then print all of the information for each person.

Part (b): Now change the main method so that after creating the *Person* array it prints the menus shown above (that allow the user to select one person and one piece of information) and the requested information. Make use of switch statements that switch on the value typed in to decide which element of the array to use, and which of its methods to call.

Part (c): Finally, add a loop so that as long as you don't type a Q the program keeps asking you for your choices.

```
for (int k=0; k < arr.length/2; k++) {
    int tmp = arr[k];
    int index = arr.length-(k+1);
    arr[k] = arr[index];
    arr[index] = tmp;
}
```

```
for (int k=0; k < arr.length-1; k++) {
    if (arr[k] > arr[k+1]) return false;
}
return true;
```

```
for (int k=0; k < arr.length-1; k++) {
    for (int j=k+1; j < arr.length; j++) {
        if (arr[k] == arr[j]) return false;
    }
}
return true;
```

```
for (int k=0; k < arr.length-1; k++) {
    for (int j=k; j < arr.length; j++) {
        if (arr[k] == arr[j]) return false;
    }
}
return true;
```

```
for (int k=1; k < arr.length-1; k++) {
    if (arr[0] == arr[k]) return false;
}
return true;
```

```
int tmp = 0;
for (int k=0; k < arr.length; k++) {
    tmp += arr[k];
}
return (tmp > 0);
```

```
for (int k=0; k < arr.length; k++) {
    if (arr[k] >= N) return false;
}
return true;
```

```
int tmp = arr[0];
for (int k=1; k < arr.length; k++) {
    if (arr[k] > tmp) tmp = arr[k];
}
return tmp;
```

```
int tmp = 0;
for (int k=0; k < arr.length; k++) {
    tmp += arr[k];
}
return (double)tmp/arr.length;
```

```
int tmp = 0;
for (int k=1; k < arr.length; k++) {
    if (arr[k] == arr[0]) tmp++;
}
return tmp;
```

The code reverses the order of the values in the array.

The code returns true if the values in the array are in sorted order, from low to high.

The code returns true if the array contains no duplicate values (i.e., no value is stored in more than one place in the array).

The code always returns false.

The code returns true if the value in `arr[0]` doesn't occur in any other place in the array.

The code returns true if the sum of the values in the array is positive.

The code returns true if all values in the array are less than N.

The code returns the largest value in the array.

The code returns the average of the values in the array.

The code returns the number of times the value in `arr[0]` occurs in other places in the array, too.

The code returns true if the values in the array are in sorted order, from high to low.

The code always returns true.

The code returns true if all values in the array are positive.

The code returns true if the number of values in the array is less than N.

The code returns the smallest value in the array.

The code checks whether the values in the array are the same forwards and backwards.