

# CSC 110: Fundamentals of Programming I

## Assignment #5: Arrays

### Due date

Sunday, March 6th, 2015 at 11:55 pm via submission to connex.

### How to hand in your work

Submit the file **SearchDNA.java** through the Assignment #5 link on connex.

### Official Assignment Specification

Your methods must adhere to the [specification document](#).

An important task in bioinformatics is the identification of DNA and RNA sequences. In this assignment we will be looking at *nucleic acid sequences*. These *sequences* contain up to four different *bases* denoted by letters: *A* for *adenine*, *C* for *cytosine*, *G* for *guanine*, and *T* for *thymine*. *Sequence* strings are compared in order to determine whether *nucleic acid sequences* match each other, or are related through *mutations*. Real sequence data as used by biochemists and in bioinformatics research consist of very long strings of *A*, *C*, *G* and *T*. Determining relatedness can require the use of very complex algorithms, beyond the scope of this assignment.

The *sequences* in this assignment will all contain between 2 and 4 of the possible *bases* (*A*, *C*, *G*, and *T*). Your task is to search through a collection of sequence data and count how many times a specific *sequence* occurs. (For example, if the collection contains the following sequences: {*ACTG*, *GATC*, *ACT*, *GTC*, *AC*, *GATC*, *GA*} and we search for the specific sequence *GATC* we would report that it was found 2 times.

One of the difficulties in this assignment will be dealing with *mutated sequences*. A *mutation* can occur due to insertions of additional *bases* within a *sequence*. For the purpose of this assignment, a *mutated sequence* contains at least two of the same *bases* occurring in a row (so in the sequence *GAAATC* the *A* has *mutated*, and in the sequence *CCGGAT* both the *C* and *G* have *mutated*). Another task in this assignment is to detect how many of the *sequences* in the collection are *mutated*. The final task will be to search through the collection of sequence data for a specific *sequence*, but you must treat original and *mutated sequences* the same (For example, if the collection contains {*TGC*, *AC*, *TTGC*, *TACG*, *TGGCC*, *AGTC*} and we search for the specific sequence *TGC* we would report that it was found 3 times (because *TTGC* and *TGGCC* are mutated forms of *TGC*).

## Recommended steps to follow in order

1. Similar Assignment 4, the [specification document](#) outlines all of the required methods for this assignment. Appendix A illustrates some examples of how to call and test the methods in your SearchDNA.java program.
2. The `printArray` method is a nice place to start. Focus on passing in an array of Strings as a parameter and using a loop to visit each element in the array. Remember that array indexes start at 0 (similar to Strings)!
3. After finishing `printArray`, we recommend working on the `findLongest` or the `findFrequency` methods, as they are quite similar. Within both methods, a loop should be created to visit each element in the array. In the `findLongest` method, you must keep track of which String in the array contains the most characters, whereas in the `findFrequency` method, you must keep track of how many times a specific String is found in the array. Make sure you finish and test them both before moving to the next step.
4. The methods involving mutations are a little more difficult. In this assignment, a mutation occurs when two or more characters in a String are repeated in a row. Think about how you might be able to detect a mutation in a String. Once you come up with a strategy, test it with a number of Strings to see if it works!

## Marking

Your mark will be based on the following criteria:

- Your code *must compile and run*. Some examples of how to test your methods, along with expected output, are outlined in **Appendix A**.
- Your code must conform to all the requirements mentioned in the [specification document](#).
- The main method must show all of your testing code. Each of the required methods should be tested.
- Your code must follow the guidelines outlined in `Style_Guidelines.pdf`, found through the Lectures & Stuff link in the Lab Resources folder on connex. You may notice that the specification document provides some very nice comments you are welcome to borrow.

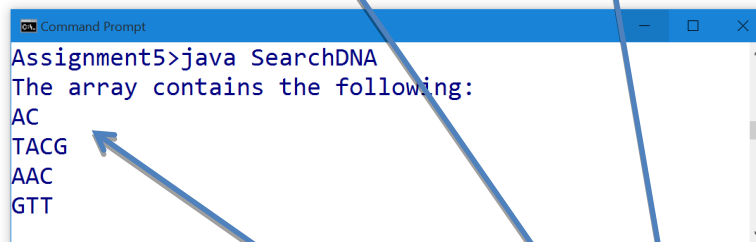
## Appendix A – Testing your code

As you work through a solution, it is recommended that you save, compile and test your code after every line or two of code that you write. This can be something as easy as printing out the value of a variable, or calling a method to print out the value returned. It is important to do this to confirm a component of your code works correctly, so you can be confident using that component throughout your code later.

### Testing the `printArray` method:

One way to do this is to create an array of Strings in the main method and then call the **printArray** method from main. An example is shown below:

```
public static void main(String[] args) {  
    String[] testArray = {"AC", "TACG", "AAC", "GTT"};  
    printArray(testArray);  
}
```



The screenshot shows a Command Prompt window titled "Command Prompt". The command entered is "Assignment5>java SearchDNA". The output displayed is "The array contains the following:" followed by four lines: "AC", "TACG", "AAC", and "GTT".

#### What to look for:

- Are all of the words printed?
- Does the method accept an array of Strings as a parameter?
- Is each word on a new line?

### Testing the *findLongest* method:

One way to do this is to create an array of Strings in the main method and then call the **longestWord** method from main, assign the value returned by the method to a String variable, and print out the word.

```
public static void main(String[] args) {  
    String[] testArray = {"AC", "TACG", "AAC", "GTT"};  
    String longest = findLongest(testArray);  
    System.out.println("The longest String is "+longest);  
  
    String[] test2 = {"TG", "TGA", "ACT", "GA", "CCT"};  
    longest = findLongest(test2);  
    System.out.println("The longest String is "+longest);  
}
```

```
Command Prompt  
Assignments\Assignment5>java SearchDNA  
The longest String is TACG  
The longest String is TGA
```

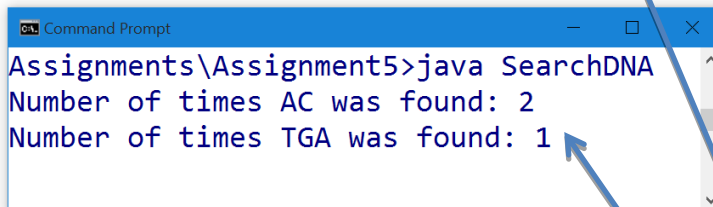
#### What to look for:

- Does the method accept an array of Strings as a parameter?
- Is the longest word in the array returned?
- If there is a tie, is the word that comes first in the array returned?

### Testing the *findFrequency* method:

One way to do this is to create a String and an array of Strings in the main method. Then call the **findFrequency** method from main passing in the String and the array as parameters, assign the value returned by the method to an int variable, and print out the value of the integer. (Matches are underlined in red)

```
public static void main(String[] args) {  
    String[] testArray = {"AC", "TAC", "GAC", "AC", "GTAC"};  
    String toFind = "AC";  
    int numTimes = findFrequency(toFind, testArray);  
    System.out.println("Number of times "+toFind+" was found: "+numTimes);  
  
    String[] test2 = {"ACTG", "TGA", "TTGA", "TGGGGA", "TGAC"};  
    toFind = "TGA";  
    numTimes = findFrequency(toFind, test2);  
    System.out.println("Number of times "+toFind+" was found: "+numTimes);  
}
```



```
Assignments\Assignment5>java SearchDNA  
Number of times AC was found: 2  
Number of times TGA was found: 1
```

#### What to look for:

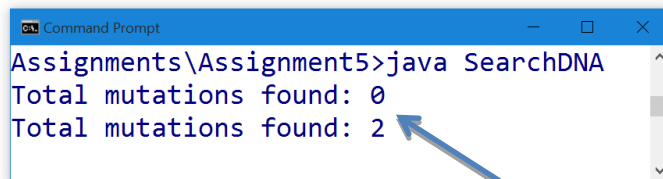
- Does the method accept 2 parameters (A String and an array of Strings)?
- Is the number of times the String occurs in Array returned?

### Testing the `countTotalMutations` method:

One way to do this is to create a String and an array of Strings in the main method. Then call the **countTotalMutations** method from main passing in the String array as a parameter, and then assign the value returned by the method to an int variable, and print out the value of the integer.

Remember that in this assignment, a *mutation* occurs when any *base* is repeated twice in a row. So, the sequences *TAG* and *ACTA* are not mutations, but the sequences *TGGGGAA* and *AATTCCGG* are. In the example below, Strings including one or more mutations are underlined in red.

```
public static void main(String[] args) {  
    String[] testArray = {"AC", "TAC", "GAC", "AC", "GTAC"};  
    int numTimes = countTotalMutations(testArray);  
    System.out.println("Total mutations found: "+numTimes);  
  
    String[] test2 = {"ACTG", "TGA", "TTGA", "TGGGGAA", "TGAC"};  
    numTimes = countTotalMutations(test2);  
    System.out.println("Total mutations found: "+numTimes);  
}
```



```
Assignments\Assignment5>java SearchDNA  
Total mutations found: 0  
Total mutations found: 2
```

#### What to look for:

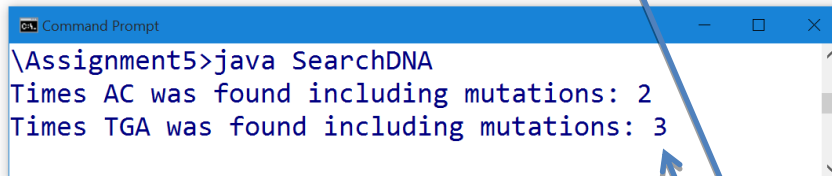
- Does the method accept 1 parameter (An array of Strings)?
- Is the correct value representing the number of Strings containing a mutation in the array returned?

### Testing the *findFreqWithMutations* method:

One way to do this is to create a String and an array of Strings in the main method. Then call the **findFreqWithMutations** method from main passing in the String and the array as parameters, assign the value returned by the method to an int variable, and print out the value of the integer.

Remember that in this assignment, a *mutation* occurs when any *base* is repeated twice in a row. So, removing the mutations from *TGGGGAA* would result in *TGA*. The sequences found to be matches when searching through the array including mutations are underlined in red.

```
public static void main(String[] args) {  
    String[] testArray = {"AC", "TAC", "GAC", "AC", "GTAC"};  
    String toFind = "AC";  
    int numTimes = findFreqWithMutations(toFind, testArray);  
    System.out.println("Times "+toFind+" was found including mutations: "+numTimes);  
  
    String[] test2 = {"ACTG", "TGA", "TTGA", "TGGGGAA", "TGAC"};  
    toFind = "TGA";  
    numTimes = findFreqWithMutations(toFind, test2);  
    System.out.println("Times "+toFind+" was found including mutations: "+numTimes);  
}
```



```
Command Prompt  
\\Assignment5>java SearchDNA  
Times AC was found including mutations: 2  
Times TGA was found including mutations: 3
```

#### What to look for:

- Does the method accept 2 parameters (A String and an array of Strings)?
- Is the right value returned?

### Extra challenge - Testing with real data from a file

On the connex page, under Resources > Assignment Resources > Assignment5, there are text files containing a collection of *sequences* that you can further test your program with. Also in the folder is a java program, FileToArrayExample.java. It contains code to read data from a file and place it into an Array of strings. Feel free to copy the method `arraySetup` found in FileToArrayExample.java and use it in your own program to further test your methods.

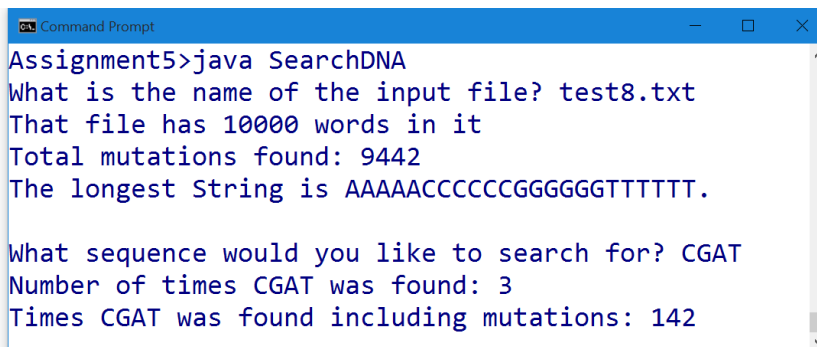
The `arraySetup` method can be used to fill the array with sequence data using any of the text files, which have been created to be used in this assignment. All of the text files contain different sets of *sequences* with varying amounts of mutations:

- test1.txt – 5 sequences, without mutations
- test2.txt – 25 sequences, without mutations
- test3.txt – 20 sequences, few mutations
- test4.txt – 50 sequences, few mutations
- test5.txt – 20 sequences, many mutations
- test6.txt – 100 sequences, many mutations
- test7.txt – 1000 sequences, many mutations
- test8.txt – 10000 sequences, many mutations

Below is an example of how we might use an input file to read in a collection of sequences, ask a user to search for a specific sequence, and output the results.

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    String[] dataArray = arraySetup();
    System.out.println("That file has "+dataArray.length+" words in it");
    int totalMuts = countTotalMutations(dataArray);
    String longest = findLongest(dataArray);
    System.out.println("Total mutations found: "+ totalMuts);
    System.out.println("The longest String is "+ longest + ".\n");

    System.out.print("What sequence would you like to search for? ");
    String seq = console.next();
    int numTimes = findFrequency(seq, dataArray);
    System.out.println("Number of times "+seq+" was found: " +numTimes);
    numTimes = findFreqWithMutations(seq, dataArray);
    System.out.println("Times "+seq+" was found including mutations: "+numTimes);
}
```



```
Assignment5>java SearchDNA
What is the name of the input file? test8.txt
That file has 10000 words in it
Total mutations found: 9442
The longest String is AAAAACCCCCGGGGGGTTTTTT.

What sequence would you like to search for? CGAT
Number of times CGAT was found: 3
Times CGAT was found including mutations: 142
```

**Even bigger challenge:** Print out which base sequence was mutated the MOST number of times (and how many times it was mutated).