# COMP1406 – Winter 2019

Problem 2 has been updated. Please see the change (in red).

Submit a single file called `assignment1.zip` to the submission server
`http://134.117.31.149:9091/`

Your zip file must contain a directory called `comp1406a1` and all of your `.java` files must be in this directory. Do not include your `.class` files.

There are 50 marks possible marks.

The objective of this assignment is to get you up to speed with basic Java syntax (control flow, branching, variables, methods, etc) and using arrays. This assignment will be graded for correctness only. The submission server will compile your code and run test cases to asses of your methods are correct or not. If one of your Java files does not compile, you will receive zero marks for that question. If your file does compile, then your grade will be determined by the test cases used.

We are not testing your code for efficiency. However, it is expected that your code will terminate in a reasonable amount of time. If your code takes longer than (roughly) five seconds to run with our test cases on the server, it will automatically terminate and you will receive a mark of zero for that question.

You may submit as many times as you wish before the submission deadline. We will use your *best* submission for your grade.

You are NOT allowed to use any other Java classes for this assignment. For example, you are not allowed to use `Arrays`, `ArrayList`, etc.
Using any other classes will result in a grade of zero.

## 1 Sequence Finder [10 marks]

In the provided `Find.java` file, complete the `locateSequence` method. For a given target sequence (non-empty array of integers), the method searches the input array (of integers) to find an occurrence of the target sequence if it is present. If the sequence is present, the method returns the array index position of where it starts in the array. If the sequence is not present, the method returns -1.

Consider the following array:

| 8 | 6 | 7 | 5 | 3 | 0 | 9 |
|---|---|---|---|---|---|---|

If we look for the sequence `7,5,3` in this array, the method should return `2`. If we look for the sequence `9`, it should return `6`, and if we look for the sequence `3,0,9,1`, it should return `-1`.

If the sequence appears in the array more than once, the returned position should correspond to the *last* occurrence of the sequence.

## 2 Sequence Finder (Again) [10 marks]

Create a class called `FindAgain` with the static method

```
public static int[] locateAllSequenceLocations(int[] target, int[] array)
```

The method returns an array of integers with size at least 1. The first value in the output array is number of occurrences of the target in the array. This number may be zero or greater. If the number of occurrences is $n$, then the next $n$ values in the output array are starting positions of the $n$ occurrence (in the order that they appear from left to right; that is, the positions are ordered from smallest to largest).

Your class must begin with the line `package comp1406a1;`.

## 3 Grades [15 marks]

In the provided `Grades.java` files, complete the `finalExamGradeNeeded` method. This method takes several input parameters: a target letter grade (`String`), a list of assignment grades (`double[]`), a list of tutorial grades (`double[]`) and a list of midterm grades (`double[]`). The method returns the minimum grade needed on the final exam to achieve the target letter grade based on the other input grades.

You must use the grading scheme as specified in the course outline. Be careful to read and understand the condition for passing the course. If the needed final exam grade is more than 100.0, then the method should return `-1.0`. If the target grade will be achieved (or surpassed) regardless of the final exam mark, the method should return `0.0`.

A table showing the relationship between letter grades and numerical grades is shown below. Note that a square bracket indicates that the end value is included in the range and the parenthesis indicates that it is not included. So, $[80, 85)$ is all values $x$ satisfying $80 \leq x < 85$.

You can expect the input values passed to the method to always satisfy the following:

- The letter grade will only be one of the 13 values shown in the table.
- there will be nine (9) assignment grades provided in the order: a1, a2, …, a6, study a1, study a2, study a3.
- there will be ten (10) tutorial grades provided.
- there will be three (3) midterm grades provided. A missed midterm will be recorded as zero.

| Letter | Grade Range | Grade Point |
|---|---|---|
| A+ | [90, 100] | 12 |
| A | [85, 90) | 11 |
| A- | [80, 85) | 10 |
| B+ | [77, 80) | 9 |
| B | [73, 77) | 8 |
| B- | [70, 73) | 7 |
| C+ | [67, 70) | 6 |
| C | [63, 67) | 5 |
| C- | [60, 63) | 4 |
| D+ | [57, 60) | 3 |
| D | [53, 57) | 2 |
| D- | [50, 53) | 1 |
| F | [0, 50) | 0 |

Note that comparing strings with `==` is problematic and may not return the result that we intuitively expect. Be sure to use the `.equals` method instead. For example, use `"A+".equals(grade)` instead of `"A+"==grade`. You might also consider the `switch` control flow construct instead of a chain of `if/else if/.../else`.

## 4 Sudoku Checker [15 marks]

In the provided file `SudokuChecker.java`, complete the following methods:

```
public static boolean checkRow(int row, byte[][] grid)
public static boolean checkColumn(int column, byte[][] grid)
public static boolean checkSubregion(int region, byte[][] grid)
```

These methods are used in the provided `check` method that takes a 2-dimensional ($9\times9$) array of numbers and determines if it represents a valid (solved) sudoku grid of numbers. A valid sudoku grid has the following properties:

- each row contains all of the digits from 1 to 9,
- each column contains all of the digits from 1 to 9,
- each of the nine 2-dimensional $3\times3$ subgregions contain all of the digits 1 to 9.

For example, the following is a valid sudoku grid (with $n = 3$). The nine subregions are shown shaded.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

The rows are labelled 0 to 8 (from top to bottom), the columns are labelled 0 to 8 (from left to right), and the subregions are labelled 0 to 8 as follows:

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Note: The `check` method will not be tested. We will be testing the three other methods with the auto-testing.

Note: Do NOT simply test your code with the `check` method. You need to individually test each of your three methods.

## Submission Recap

A complete assignment will consist of a single file (`assignment1.zip`) with the following four files included: `Find.java`, `FindAgain.java`, `Grades.java`, and `SudokuChecker.java`.

Be sure to use the default compression program on your compute to create the zip file. Do NOT use 7zip, winrar, etc. The server may reject your submission if you do not use the default compression (archive) program.