Dalhousie University
CSCI 2132 — Software Development
Winter 2018
Assignment 5

Distributed Wednesday, February 28 2018.

Due 9:00PM, Wednesday, March 7 2018.

Instructions:

1. The difficulty rating of this assignment is *silver*. Please read the course web page for more information about assignment difficulty rating, late policy (no late assignments are accepted) and grace periods before you start.

2. Each question in this assignment requires you to create one or more regular files on bluenose. Use the exact names (case-sensitive) as specified by each question.

3. C programs will be marked for correctness, design/efficiency, and style/documentation. Please refer to the following web page for guidelines on style and documentation for short C programs (which applies to this assignment):

   `http://web.cs.dal.ca/~mhe/csci2132/assignments.htm`

   You will lose a lot of marks if you do not follow the guidelines on style and documentation.

4. Create a directory named `a5` that contains the following files (these are the files that assignment questions ask you to create): `a5q1.c` and `a5q2.c`. Submit this directory electronically using the command `submit`. The instructions of using `submit` can be found at:

   `http://web.cs.dal.ca/~mhe/csci2132/assignments.htm`

5. Do NOT submit hard copies of your work.

Questions:

1. [15 marks] Suppose that students enrolled in one course are required to take four tests, and each student's final grade for this course is the average of his/her grades of these four tests. This question asks you to write a program that can be used to compute the lowest final grade, highest final grade and the average final grade.

   General Requirements: Use `a5q1.c` as the name of your C source code file.

   We will use the following command on bluenose to compile your program:

```
gcc -std=c99 -o grades a5q1.c
```

Your program should NOT print anything to prompt for user input. As many numbers have to be read from stdin to test your programs, you are expected to use input redirection to read the input from a file. More precisely, you can run your program using the following command:

```
./grades < grades.1
```

In the above command, the file `grades.1` is a plain text file of the following format:

(a) The first line has one integer. If this integer is 1, 2, or 3, then you are asked to compute the lowest, highest, or average final grade, respectively.

(b) The second line also has one integer, and it is the number of students.

(c) Starting from the third line, each line stores the grades of one student. More precisely, it contains four integers (separated by spaces) in the range $[0, 100]$, which are grades this student obtained in the four required tests.

For example, consider the following input file:

```
1
3
90 80 77 79
65 78 81 88
92 91 95 93
```

It means that you are asked to find out the lowest final grade. There are three students. The first student got $90, 80, 77, 79$ as grades in his four tests, and so on.

Your program should then compute the lowest, highest or average final grade according to the first integer read, and print the result as a floating point number, keeping two digits after the decimal point. A trailing newline is also printed. For this input, the second student's final grade is $(65 + 78 + 81 + 88)/4 = 78$, which is the lowest among the three students. Thus the output is:

```
78.00
```

Error Handling: You can assume that user input contains integers only. However, your program should print an appropriate error message and terminate in each of the following cases:

(a) The first number of the input is not 1, 2, or 3 (i.e., an incorrect option is given).

(b) The second number is not positive.

(c) Each grade in the input is not an integer in the range $[0, 100]$.

If there is more than one problem with user input, your program just has to detect one of them.

Testing: To test your program automatically, we will use it as a UNIX filter.

For the example above, we will test it using the following command in the directory containing the executable program (assume that `grades.1` is the input file)

```
./grades < grades.1
```

To help you make sure that the output format of your program is exactly what this questions asks for, several files are provided in the following folder on bluenose to show how exactly your program will be automatically tested:

```
/users/faculty/prof2132/public/a5test/
```

In this folder, open the file `a5q1test` to see the commands used to test the program with three test cases. The output files, generated using output redirection, are also given.

To check whether the output of your program on any test case is correct, redirect your output into a file, and use the UNIX utility `diff` (learned in Lab 3) to compare your output file with the output file in the above folder, to see whether they are identical.

Since these output files are given, we will **apply a penalty to any program that does not strictly meet the requirement on output format**. This includes the case in which your output has extra spaces, or has a missing newline at the end.

We will use different test cases to test your program thoroughly. Therefore, construct a number of test cases to test your program thoroughly before submitting it.

Hint: You are not required to organize your program using functions yet for this assignment (it will be required for Assignment 6). However, if you do use functions, then, when handling errors, you may need to terminate the program when a function that is not `main` is being executed. Read page 203 of the text book to find out how to use the `exit` function to do this.

2. [15 marks] **Background:** A point (x_1, y_1) is said to *dominate* another point (x_2, y_2) in the plane if both $x_1 \geq x_2$ and $y_1 \geq y_2$ hold. For example, $(5, 2)$ dominates $(3, 1)$, but neither of the two points $(3, 2)$ or $(2, 5)$ dominates the other. A point p is a *maximal point* of a point set S if there does not exist another point in S that dominates p.

   The problem: In this question, you are asked to write a program based on sorting that reports all the maximal points in a given point set. Note that this problem is useful for the support of certain new features of SQL (structured query language) for database systems.

How do we solve this problem? One of the most efficient solutions makes use of an algorithm design paradigm called "reducing to known problems". Informally speaking, this paradigm means that when we encounter a new problem, we can try to solve it by making use of solutions to other problems that we already know how to solve.

The problem of finding all the maximal points can be solved by making use of solutions to the sorting problem. To see this, first sort all the points by x-coordinates in ascending order. Then, check all the points from right to left. Think of this question: What points should you report? I will state the answer in the next paragraph; I would however suggest you to try to figure out how by yourself first and then read the next paragraph to confirm whether your own solution is correct.

Here is the solution: sort all the points in ascending order by x-coordinate. Then check the points one by one from right to left, recording the largest y-coordinate seen so far. For each point, if its y-coordinate is larger than the largest y-coordinate of all the points to its right, report it as a maximal point. Note that the rightmost point is always a maximal point. Draw a figure yourself to see the correctness. To simplify your work, you can assume that x-coordinates are distinct.

You are asked to implement this solution. In your implementation, you can use any reasonable sorting algorithm including insertion sort and bubble sort (though you should know why they are not as efficient as better solutions), and you will not lose any marks if you do not implement the most efficient sorting algorithms. You are forbidden to use any existing C library functions that can be used to sort an array, such as `qsort`. You will lose a lot of marks if you do so.

No marks will be given if your solution is not based on sorting as specified above.

General Requirements: Use `a5q2.c` as the name of your C source code file.

We will use the following command on bluenose to compile your program:

```
gcc -std=c99 -o max a5q2.c
```

As many numbers have to be read from stdin to test your program, you are expected to use input redirection to read the input from a file. More precisely, you can run your program using the following command:

```
./max < max.in.0
```

In the above command, the file `max.in.0` is a plain text file that contains the coordinates of exactly **ten** points. This file has exactly ten lines, and each line contains two integers. The first integer is the x-coordinate of a point, and the second integer is its y-coordinate. Hence, in this question, you just have to handle point sets of size 10 and you can assume that coordinates are `int` values.

For example, the following file `max.in.0` contains coordinates of 10 points:

```
12 5
29 7
0 13
55 9
47 2
23 -5
-7 90
11 33
25 -24
21 7
```

Your program should then find the maximal points among these 10 points, and report the maximal points in decreasing order of x-coordinates. To report a point, print its x-coordinate and y-coordinate in a single line, use exactly one space between these two values (this is the only space character in this line), and terminate this line with a newline character. For example, the output of the above example is expected to be

```
55 9
11 33
-7 90
```

Error Handling: No error handling is required.

Testing: To test your program automatically, we will use it as a UNIX filter.

For the example above, we will test it using the following command in the directory containing the executable program:

```
./max < max.in.0
```

To help you make sure that the output format of your program is exactly what this question asks for, several files are provided in the following folder on bluenose to show how exactly your program will be automatically tested:

```
/users/faculty/prof2132/public/a5test/
```

In this folder, open the file `a5q2test` to see the commands used to test the program with two test cases. The output files, generated using output redirection, are also given.

To check whether the output of your program on any test case is correct, redirect your output into a file, and use the UNIX utility `diff` (learned in Lab 3) to compare your output file with the output files in the above folder, to see whether they are identical.

Since these output files are given, we will **apply a penalty to any program that does not strictly meet the requirement on output format**. This includes the case in which your output has extra spaces, or has a missing newline at the end. Note that **it is extremely important to use diff here, as the number of values in the output file is large**.

We will use different test cases to test your program thoroughly. Therefore, construct a number of test cases to test your program thoroughly before submitting it.

Note: Since after sorting, all that is required to compute maximal points is essentially a linear scan, this means that our solution is as efficient as the sorting algorithm used in our implementation. As sorting is very well studied, this means that computing maximal points in this way is very efficient. As you may already know, the number of comparisons required of the best sorting algorithms is proportional to only $n \lg n$.

This type of questions are typical technical interview questions for software developers (and your interviewer will not simply tell you the solution).