Algorithms Correctness and Efficiency Coursework Submission Guide

Release 1.0

Algorithms Correctness and Efficiency Teaching Team

Nov 21, 2023

CONTENTS

0.1 Changelog

> **Warning:** The content of this file may change in future, please always refer to the latest version on Moodle.

2023-10-23 1.0
Original version

CHAPTER

ONE

SOURCE CODE SUBMISSION MANUAL

1.1 Introduction

1.1.1 Keywords

The keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this guide are to be interpreted as follow:

MUST
: This word, or the terms **REQUIRED**, **SHOULD** or **SHALL**, mean that the instruction is an absolute requirement of this guide.

MUST NOT
: This phrase, or the phrases **SHOULD NOT** or **SHALL NOT**, mean that the instruction is an absolute prohibition of this guide.

MAY
: This word, or the phrases **RECOMMENDED** or **OPTIONAL**, mean that the instruction is truly optional.

1.1.2 Definitions

`standard input`
: This term, or the terms `System.in` or `stdin`, mean that the stream from which input to the program is taken. Typically this is the keyboard, but it can be specified that input is to come from a serial port or a disk file, for example.

`standard output`
: This term, or the terms `System.out` or `stdout`, mean that the stream to which output from the program is sent. Typically this is a display, but it can be redirected to a serial port or a file.

`standard error`
: This term, or the terms `System.err` or `stderr`, mean that the stream to which error output from the program is sent. Typically this is a display, but it can be redirected to a serial port or a file.

`<empty line>`
: This term, or the terms `\n`, `\r\n`, `<LF>` or `<CR><LF>`, mean the character `\n` after the last non-whitespace character in the previous content.

  For example, the following representations are all equivalent:

<empty line>	\n
first second third last <empty line>	first\nsecond\nthird\nlast\n

1.2 Submission

You must submit Java source code files, one for each question, that containing all your code for the corresponding question. Source code files must not require any other files outside of the standard Java packages which are always available.

The source file must compile and execute without warnings or errors using the command

- Compile:

```
javac -encoding UTF-8 -sourcepath . <SOURCE_FILE_NAME>
```

- Execute:

```
java -Dfile.encoding=UTF-8 -XX:+UseSerialGC -Xss64m -Xms1920m -Xmx1920m <COMPILED_
↪CLASS_NAME>
```

This command will be run on our Linux server CSLinux. If it does not compile, for any reason, then you will lose all the marks for testing (common reasons in the past have been submitting a file with the wrong filename, or developing your solution on your personal computer without having tested it on our Linux server). If the file compiles but has warnings then you will lose some marks for not correcting the warnings.

The completed source code file should be uploaded to the coursework submission link on the COMP2048 Moodle page. You may submit as many times as you wish and the last submission will be used for marking. However, if you submit after the deadline, your last submission time will be considered for the Late Submission penalty. Do not wait until the last moment to submit the coursework. You need to plan ahead to allow time for foreseeable things outside of your control.

1.2.1 Sample Files and Command-line Usages

- Compile:

```
javac -encoding UTF-8 -sourcepath . <SOURCE_FILE_NAME>
```

- Execute:

```
java -Dfile.encoding=UTF-8 -XX:+UseSerialGC -Xss64m -Xms1920m -Xmx1920m <COMPILED_
↪CLASS_NAME> < 1.in
```

Then check the output is exactly the same as the content in the corresponding `.out` files.

Note:

- < is used for redirect the input from your keyboard to a given file.

- `1.in` is a file containing the input in the same folder with the program, and it may be replaced by other file names in testing and marking. You can replace it with other `.in` files such as `2.in` as long as it exists.

1.2.2 Pre-Submission Checklist

Before submitting your code, you **MAY** do the following checks to help you pass the automated judging system:

- The file name and the main class shares the same case-sensitive names.
- No usage of `package` statement.
- No usage of third-party packages that are not included in the JDK (import packages only begin with `java.`).
- Only the main class is marked as `public`.
- Using a single scanner/reader for inputs (`System.in` appears no more than once).
- No code that produces unnecessary empty lines (i.e., `System.out.println("");`).
- Compiling and running your latest code on CSLinux.

During your manually testing on your code, ensure the following that helps you handle all `<empty line>`, `\n`, `\r\n`, `<LF>`, `<CR><LF>` issues:

- No empty lines in visual:
  - Pressing `ENTER` exactly once at the last line of input then the output is shown.
  - The output is exactly the next line to the last line of input.

1.3 Technical Notes

This part contains important technical information and it is important that you read and understand all the information below.

- You program **MAY** have multiple classes if you wish, but **only** in one java file for each question. And **only** the class with your `main` method **SHOULD** be marked as `public`.
- Your program **MUST** read its input from `standard input`.
- Your program **SHOULD** send its output to `standard output`. Your program may also send output to `standard error`, but **only** output sent to `standard output` will be considered during judging.
- If your program exits with a non-zero exit code, it will be judged as a run-error.
- Program submitted will be run inside a `sandbox`.
  - The sandbox will allocate **2GB** of memory for your program. Your entire program, including its runtime environment, must execute within this memory limit. For Java, the runtime environment includes the interpreter (JVM).
- We suggest that you do not use package statements (that is, we suggest that your solution reside in the default package).

1.4 Possible results

A submission can have the following results (not all of these may be available depending on configuration of the system):

CORRECT
: The submission passed all tests: you solved this problem!

COMPILER-ERROR
: There was an error when compiling your program. Note that when compilation takes more than 30 seconds, it is aborted and this counts as a compilation error.

TIMELIMIT
: Your program took longer than the maximum allowed time for this problem. Therefore it has been aborted. This might indicate that your program hangs in a loop or that your solution is not efficient enough.

RUN-ERROR
: There was an error during the execution of your program. This can have a lot of different causes like division by zero, incorrectly addressing memory (e.g. by indexing arrays out of bounds), trying to use more memory than the limit, reading or writing to files, etc. Also check that your program exits with exit code 0!

NO-OUTPUT
: Your program did not generate any output. Check that you write to standard out.

OUTPUT-LIMIT
: Your program generated more output than the allowed limit. The solution is considered incorrect.

WRONG-ANSWER
: The output of your program was incorrect. This can happen simply because your solution is not correct, but remember that your output must comply exactly with the specifications of the judges. See *testing* below for more details.

The judges may have prepared multiple test files for each problem.

1.5 Judging Process

The judging system is fully automated. Judging is done in the following way:

1.5.1 Submitting solutions

You should submit all related files to Moodle according to the `Coursework Issue Sheet`.

1.5.2 Compilation

Your program will be compiled on **CS-Linux**. All submitted source files will be passed to the compiler which generates a single program to run. For Java and Kotlin the given main class will be checked; for Python we do a syntax check using the `py_compile` module.

1.5.3 Testing

After your program has compiled successfully it will be executed and its output compared to the output of the judges. Before comparing the output, the exit status of your program is checked: if your program exits with a non-zero exit code, the result will be a run-error even if the output of the program is correct! There are some restrictions during execution. If your program violates these it will also be aborted with a run-error, see *the section on restrictions*.

When comparing program output, it has to exactly match to output of the judges, except that some extra whitespace may be ignored. So take care that you follow the output specifications. In case of problem statements which do not have unique output (e.g. with floating point answers), the system may use a modified comparison function. This will be documented in the problem description.

1.5.4 Restrictions

Submissions are run in a sandbox to prevent abuse, keep the judging system stable and give everyone clear and equal environments. There are some restrictions to which all submissions are subjected:

compile time
: Compilation of your program may take no longer than 30 seconds. After that, compilation will be aborted and the result will be a compile error. In practice this should never give rise to problems. Should this happen to a normal program, please inform the judges right away.

source size
: The total amount of source code in a single submission may not exceed 256 kilobytes, otherwise your submission will be rejected.

memory
: The judges will specify how much memory you have available during execution of your program. This may vary per problem. It is the total amount of memory (including program code, statically and dynamically defined variables, stack, Java/Python VM,)! If your program tries to use more memory, it will most likely abort, resulting in a run error.

creating new files
: Do not create new files. The sandbox will not allow this and the file open function will return a failure. Using the file without handling this error can result in a runtime error depending on the submission language.

number of processes
: You are not supposed to explicitly create multiple processes (threads). This is to no avail anyway, because your program has exactly 1 processor core fully at its disposal.

  People who have never programmed with multiple processes (or have never heard of threads) do not have to worry: a normal program runs in one process.

internet access
: Your programs are not allowed to access the Internet. Any attempts to access the Internet from your programs will result in a run-error.

1.6 Code examples

Below are a few examples on how to read input and write output for a problem.

The examples are solutions for the following problem: the first line of the input contains the number of testcases. Then each testcase consists of a line containing a name (a single word) of at most 99 characters. For each testcase output the string `Hello <name>!` on a separate line.

Sample input and output for this problem:

Input	Output
3 world Jan SantaClaus	Hello world! Hello Jan! Hello SantaClaus!

Note: The number 3 on the first line indicates that 3 testcases follow.

What follows is a number of possible solutions to this problem for different programming languages.

Listing 1: *A solution in Java*

```java
// Note: do not use any 'package' statements

import java.util.*;

class Main {
        public static void main(String[] args) {
                Scanner scanner = new Scanner(System.in);
                int nTests = scanner.nextInt();

                for (int i = 0; i < nTests; i++) {
                        String name = scanner.next();
                        System.out.println("Hello " + name + "!");
                }
        }
}
```