Programming and Algorithms Coursework 1

Release 1.0

Programming and Algorithms Teaching Team

Nov 16, 2023

CONTENTS

0.1 Changelog

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

2023-10-28 1.0
Release candidate.

CHAPTER

ONE

COURSEWORK SPECIFICATION

1.1 Introduction

This is the first marked coursework for Programming and Algorithms (COMP1038). It is worth **30% of the module mark**. It requires you to write a program to solve the given problem(s). The deadline for this coursework is **23:55 on Wednesday 22nd of November 2023**.

Read the entire document before beginning the coursework.

If you have any questions about this coursework, please ask in the Q&A forum on Moodle, after a lecture, in a lab, or during the advertised office hours. Do not post your program or parts of your program to Moodle as you are not allowed to share your coursework programs with other students. If any questions require this coursework to be clarified then this document will be updated and everyone will be notified via Moodle.

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 term `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 term `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 term `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 a single C source code file containing all your code for this exercise. This file must be called `calculator.c` and must not require any other files outside of the standard C headers which are always available. The first line of the file should be a comment which contains your student ID number, username, and full name, of the form:

```
// 6512345 zy12345 Joe Blogs
```

The file must compile without warnings or errors using the command

```
gcc -std=c99 -lm -Wall calculator.c -o calculator
```

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 1 Submission link on the COMP1038 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. After the deadline has passed, if you have already submitted your exercise then you will not be able to submit again. If you have not already submitted then you will be allowed to submit **once**.

Late submissions: COMP1038 late submission policy is different from the standard university policy. Late submissions will lose 2 percentage points **per hour**, rounded up to the next whole hour. This is to better represent the large benefit a small amount of extra time can give at the end of a programming exercise. No late submissions will be accepted more than 50 hours after the exercise deadline. If you have extenuating circumstances you should file them before the deadline.

1.3 Plagiarism

You should complete this coursework on your own. Anyone suspected of plagiarism will be investigated and punished in accordance with the university policy on plagiarism (see your student handbook and the University Quality Manual). This may include a mark of zero for this coursework.

You should write the source code required for this assignment yourself. If you use code from other sources (books, web pages, etc), you should use comments to acknowledge this (and marks will be heavily adjusted down accordingly). The only exception to this is the dynamic data-structures (linked lists and others) developed during the lectures and tutorials; you may use these, with or without modification, without penalty as long as you add a comment in your program saying you have taken them from the lectures or tutorials and saying how you have modified it (or not modified it). If you do not acknowledge their source in a comment then it will be regarded as potential plagiarism.

You must not copy or share source code with other students. You must not work together on your solution. You can informally talk about higher-level ideas but not to a level of detail that would allow you all to create the same source code.

Remember, it is quite easy for experienced lecturers to spot plagiarism in source code. We also have automated tools that can help us identify shared code, even with modifications designed to hide copying. If you are having problems you should ask questions rather then plagiarize. If you are not able to complete the exercise then you should still submit your incomplete program as that will still get you some of the marks for the parts you have done (but make sure your incomplete solution compiles and partially runs!).

1.4 Marking

The marking scheme will be as follows:

- **Tests (26 marks)**: Your program should correctly implement the task requirements. A number of tests will be run against your program with different input data designed to test if this is the case for each individual requirement. The tests themselves are secret but general examples of the tests might be:
  - Does the program work with the example I/O in the question?
  - Does the program work with typical valid input?
  - Does the program correctly deal with invalid input?
  - Does the program output match the required format?
  - Does your code run correctly without any run-time error?

As noted in the submission section, if your program does not compile then you will lose all testing marks. As well as if you submit a different type of file apart from a single C source code file containing all your code for this exercise and the file name is different from `calculator.c` then you will also lose all testing marks. We usually use an automatic marking system to test/mark your coursework submissions. So you should strictly follow the output format specified in this task description while implementing your program. Otherwise, your program will fail to test and you will lose marks. If your code compiles with warnings, each warning message causes 10% mark deduction.

- **Appropriate use of language features (2 marks)**: Your program should use the appropriate C language features in your solution. You can use any language features or techniques that you have seen in the course, or you have learned on your own, as long as they are appropriate to your solution. Examples of this might be:
  - Have you broken your program down into separate functions?
  - Are all your function arguments being used?
  - If your functions return values, are they being used?
  - Are you using loops to avoid repeating many lines of code?
  - Are your if/switch statements making a difference, or is the conditions always true or false making the statement pointless?
- **Source code formatting (2 marks)**: Your program should be correctly formatted and easy to understand by a competent C programmer. This includes, but is not limited to, indentation, bracketing, variable/function naming, and use of comments.

1.5 Question: calculator - 'Arithmetic calculator'

1.5.1 Problem Description

Your task is to implement an arithmetic calculator program in C programming language.

The program takes one mathematical expression in a line, then output the result to the expression, otherwise output `Invalid input` and finish.

Your program should be able to handle arithmetic operators `+`, `-`, `*`, `/`, `%` (modulo), and `(`, `)`.

1.5.2 Input

The input is one mathematical expression in a line of no more than 100 characters. A valid numerical value satisfies the following conditions:

- Its length must be no more than 12 digits including the decimal point.
- It is in one of the forms `123`, `-12.3`, `12.3000`, `123.` and `.123`, but **NOT** in the forms `1e23`, `00123`, `+12.3`, `- 12 .3`, and `00.123`.

A valid arithmetic expression contains arithmetic operators, numerical values, and white-spaces. It may contain extra spaces in between an operator and a value and you should simply ignore them. Your program should output `Invalid input` if the input contains more than 100 chars. You **MUST NOT** assume the expression is always legal and **SHOULD** validate the expression yourself.

Some examples of valid arithmetic expression are:

```
(1 + 2)* 3.14
-10.56785446/ 15.34
0.5* ((1.5 -1)+   2)
```

1.5.3 Output

Output the result to the input expression.

The answer should have an absolute or relative error of at most 10^{-6}, if the expression is invalid or cannot be calculated, output case-sensitive string `Invalid input`.

1.5.4 Sample Input/Output

Input	output
`1+a`	`Invalid input`

Input	output
`(1+2+3))`	`Invalid input`

Input	output
`3.14%2 -1`	`Invalid input`

Input	output
`3.141592685746* 2`	`Invalid input`

Input	output
`1+2 * 3`	`7.000000`

1.5.5 Sample Files and Command-line Usages

To compile the C code:

```
gcc -std=c99 -lm -Wall calculator.c -o calculator
```

To test the program with the sample files:

```
./calculator < 1.in
```

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

To detect memory leak:

```
valgrind --leak-check=full ./calculator < 1.in
```

Note:

- < is used for redirect the input from your keyboard to a given file.
- `1.in` is a file containing the input mathematical expression 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.

CHAPTER

TWO

SOURCE CODE SUBMISSION MANUAL

2.1 Technical Notes

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

- You program can **only** have one C file for each question.
- 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.

2.2 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.

2.3 Judging Process

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

2.3.1 Submitting solutions

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

2.3.2 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.

2.3.3 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.

2.3.4 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.

2.4 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 C*

```c
#include <stdio.h>

int main() {
        int i, ntests;
        char name[100];

        scanf("%d\n", &ntests);

        for (i = 0; i < ntests; i++) {
                scanf("%s\n", name);
                printf("Hello %s!\n", name);
        }
}
```