

# COMP 281 Detailed Marking Guidelines

This document is an effort to make the marking for COMP281 more transparent for the students.

Please note that this document contains guidelines that we have applied in order to be able to deliver a fair and consistent grading, but that they are *guidelines*: there could be reasons to deviate from them. Also the list of items here is *not exhaustive!* There could be many other ways in which you could lose points.

That is, when handing in your work: please think about whether you wrote a good program that is efficient, nicely commented etc. and whether you do a good job of explaining your approach in the report.

## Functionality and Correctness

The single most important criterion for this category is: *Does the code correctly implement a solution to the problem stated in the assignment.*

It is hard to predict exactly the type of mistakes that will be made, so it is difficult to give an exact list of penalties that apply. General guidelines that we are following are as follows:

- functionality mostly correct, but not dealing appropriately with 1 special case (e.g., only works for positive integers): subtract 3
- many special cases not treated correctly: subtract 4–8 depending on severity.
- code does not compile due to syntax errors: subtract 5–10 depending on the number/severity of the errors. (This is assuming that the rest of the code would be good if these errors were fixed)
- code does not work *at all*, or it is not clear what the code is trying to accomplish: 0 points for functionality.

In addition, when taking a 'short cut' to just get OJ to accept without implementing the problem as stated, we subtract most of the marks (e.g., -8) based on perceived badness of the approach. [An example of this is not implementing *arbitrary* precision division, but just using a float or double to do the division.]

In addition, points are subtracted here for not correctly submitting the assignment. E.g.:

- Program was not submitted to OJ: -5
- Program does not mention runID: -2
- Code handed in as .txt file: -1

As a general comment: the penalties for functionality can be quite severe, and this is a design choice. I think it is fair that students who did get all the assignment implemented correctly should see a higher mark than students who did not, and penalizing incorrect solutions in this fashion aims to accomplish that. Also note that there are multiple problems in each assignment, so you can compensate with other problems.

# Programming style, use of comments, indentation and identifiers

In this section, we judge both the style of the code (comments, etc.) as well as the efficiency of the code. So there are several aspects to be considered here.

## Cap for non-functional programs

First, there is a cap in the number of points you can get for a program that does not work at all. This is done to avoid people from earning 8 points for code that, even though neatly commented and indented, does not work at all.

To drive this point home, we want to avoid that people who submit:

```
int main(void)
{
    //Now print 'Hello World'!
    printf("Hello world!");
    return(0);
}
```

(for, e.g., a problem asking to implement matrix multiplication) get a passing grade.

Exactly where this cap is placed depends on how bad the functionality would be. I gave the demonstrators a guideline to cap the marks you can get at 4, but I hope it is clear that the exact cap will depend on what is submitted: if you submit 'hello world' you will be capped at 0.

## Indentation

Indentation is an important part of readable code. Failing to properly indent will lead to subtractions:

- No indentation at all (-3).
- Severe inconsistencies in indentation (-2).
- Indentation of code and comments not completely consistent (-1).

## Comments

Comments are also an important part of readable code. Failing to properly comment will lead to subtractions:

- No comments at all (-3).
- Insufficient comments (-2).
- There are a couple of statements that could use a comment (-1).

## Constants

Variables or numbers that are constant should be defined as constants. This is just good programming practice.

- If you failed to use a constant where appropriate: -1.
- If this also lead to replicated occurrences of the same number: -2.

## Naming Conventions etc.

While naming conventions for C are not set in stone, we would like you to be at least consistent. If you are not, you will be penalized about 3 marks (depending on severity).

If naming is consistent, you should be fine except for a few conventions which are adhered to in general:

- Defined (i.e., using `#define`) constants should be all upper case: -1
- Variables should start with a lower case first letter: -1
- Avoid names that differ only in case, like *foo* and *Foo*: -1
- Avoid names that look like each other (on many terminals and printers, 'l', '1' and 'I' look quite similar): -1
- Name your constants with underscores ```_``, e.g. MAX_LENGTH, so they can be read easier: -1`

Btw., for pointers to good C-style, have a look at:

<https://www.doc.ic.ac.uk/lab/cplus/cstyle.html>

## Efficiency / Modularity / Generality

Since we want to award top marks only to students who deliver the very best programs, we also subtract points for programs that are not as efficient, modular or general as they could be. Note that C programming is about writing very efficient programs, so this is an important aspect of the learning outcomes of the module.

In this category, it is absolutely not possible to perfectly capture all possible inefficiencies etc. So the following are necessarily crude guidelines:

- Not using functions when that would have significantly improved the readability and modularity of the code, without making the program much less efficient: -3
- Using non-generic code while generic code (e.g., `void *`) could be written without loss in performance: -3.
- Well motivated global variables are okay, as long as they are not manipulated in too complex manners. If the use is cryptic or not well motivated: -3.
- Storing large objects on the stack or passing them by value: subtract 2–4 marks depending on severity.
- Memory leak: not calling `free` (-2 per occurrence up to max. of -6)
- Inclusion of non-used header file: -2.

## Report

The report is the place where you can show that you really understand what is going on. The guidelines are as follows. For each problem, you can score:

- 0, if the description is not clear
- 1, if the description is somewhat clear, but lacks in some aspects
- 2, if the description is clear, to-the-point, and shows insight that you understand the important aspects of the problem.

Also, it is important to write descent reports that look good, so we also penalize:

- Bad use of English language: many grammar mistakes and/or typos: -3
- Ugly looking report: -1
- No name: -1

(These are total subtractions; not per problem).