C-coding style rules for Programming Fundamentals

After the deadline of each lab session, the teaching assistants (TAs) will assess your last accepted solution (for each problem) on programming style. This means that they will not assess whether the program produces the right output (because Themis did that already), but they assess based on the following style guide. Note that there are always a few students that use their own style rules in their private programming projects, and they complain that they do not like the rules imposed on them. Well, the simple rule is that you use these guide rules during this course: they were designed with good software engineering principles in mind. We leave it up to you to decide whether you adopt them also in your private programming projects or not.

Use common sense. Remember that this style guide is only a guide. Your primary concern should be making sure that others can read and understand the text of your program. If you think an additional comment or particular organization will get your ideas across more effectively, do it.

The TAs will subtract 0.5 points for each violation of the style guide rules. Of course, if you violated a rule multiple times, then you will only be punished once. Weekly, a maximum of two grade points can be subtracted from the grade that was given by Themis. If no violation of the style rules is detected, then the grade that was given by Themis is the final grade.

1 General Program Format

As can be seen in the slides of week 1, each program should follow the following template:

```
/* file   : skeleton.c */
/* author : Joe Sixpack (j.sixpack@student.rug.nl) */
/* date   : Mon Sep 4 2023 */
/* version: 1.0 */

/* Description:
 * Here should be a description of
 * what this program is supposed to do.
 */

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  /* here your own code */
  return 0;
}
```

2 One statement per line

We only allow one statement per line, so lines like `x=0; y=1;` are not accepted and should read:

```
  x=0;
  y=1;
```

3 Comment lines

It should be clear from the comment lines in your code what the logic of the program is. Not including comments in the program is surely not accepted. On the other hand, it is impossible to tell exactly how much comment lines are needed.

4 Identifiers

Choose meaningful names for all variables, parameters, and functions. Local variables that are used as running variables in loops can have 'simple' names like `i`, `j`, `k`, ...

All identifiers are in *camelCase*. That is, they start with a lower-case letter, followed by letters and digits. Each word starts with an upper-case letter (e.g. `numberOfPersons`). We do not use any other characters (like underscores)!

The name of a `void`-function is in imperative form (like `void printValue(int value)`), while a function that returns a value should have a name that gives a description of its results (like `int isPrime(int number)` returns an `int` that should be interpreted as a Boolean value).

Macros and constants that are defined via the C-preprocessor `#define`-directive are entirely written in upper case letters.

5 Indentation

Identation is critical for the readability of a program. No identation is absolutely unacceptable.

The open brace (`{`) goes at the end of the line before the start of the corresponding code block, and the closing brace (`}`) goes on its own line (except in `if-else` statements, see below), indented to match the beginning of the line containing the corresponding open brace, and may include (but not required) a comment that indicates which opening brace it matches.

We indent one level (suggested is 2 or 3 spaces, do not use TABs) for each level of curly braces (`{}`). Curly braces are required in an `if-else` statement, even if a code block contains only a single statement. In case of an `else`, the keyword `else` and its opening brace (`{`) is placed on the same line as the closing brace (`}`) of the `if`-part.

Example of the proper indentation of a small program fragment:

```
int gcd(int a, int b) {
  /* this is a naive implementtaion of Eclid's gcd algorithm */
  if (a < b) {
    /* swap a and b */
    int h = a;
    a = b;
    b = h;
  }
  /* here we know that a >= b */
  if (b == 0) {
    return a;  /* because gcd(a,0) = a */
  } else {
    while (a > b) {
      a = a - b;
      if (a < b) {
        /* swap a and b */
        int h = a;
        a = b;
        b = h;
      }
    } /* end while loop */
    return a;
  }
}  /* end function gcd */
```