Grading methodology

1) Code decomposition and restriction

Working Hypothesis: we assume that the 2 tasks are coded (dictionary check / anagram search). However, in case some task is missing, remove **0.5 pt** per missing task.

The main project constraint is a maximum of 40 lines/function ; you can check with geany line numbers.

(a) If there is only a single main() function => **0 pt** in the cell.

(b) Otherwise, -**0.5 pt** for each too large function But no negative value in this column.

2) The **vector/algorithm restrictions** are stated p 9 of the specifications: *no use of **insert** and **erase** of **vector**. No use of qsort from <algorithm>. The use of **swap has been allowed** for performance reasons.* Please note there is no restriction on methods on **string**.

(c) remove **0.25 pt** for each restriction violation

But no negative value in this column .

→ in the violation column provide:

- ☐ the name of functions violating the size constraint of max 40 lines.
- ☐ The line numbers not respecting the vector/algorithm restrictions

→ in the Comments on code decomposition column, provide your own brief analysis of the decomposition and restriction violation if any ; don't hesitate to suggest good practices.

3) Style and conventions.

Checking the violations listed in the **Criteria List** given next page: **-0.5 pt** for each **violation criteria code.**

The column **violation_list** shows the violation criteria **code** followed by the **line number** it occurs. For instance **[L2]57** means that **line 57** is too long and is a **wrapping one**. Here is the link to the conventions. Keep the violation_list alphabetically sorted and separate each entry by a semicolumn. If the same type of violation occurs multiple times, we show **the line numbers at least for the number of times requested for the penalty to apply.** For instance **[L2]57,102,233** means that the issue with L2 occurs at least for lines 57, 102 and 233. In the comment on style and convention column provide at least OK or a brief evaluation. We may provide warnings on other aspects that are not graded this time.

CRITERIA LIST BASED ON **ONLY** THOSE **[CONVENTIONS]**

[L1] missing indentation for control statement (if, for, while…) or the brace style is not constant over the whole code written by the student.
=> Penalize if it occurs in at least TWO places.
=> PLEASE first check [L11] that indicates we accept 3 more indentation styles of *single controlled instruction*

[L11] missing indentation for the body of any function.

[L14] double indentation in a block of code (too much indentation)

```
for(i=0 ; i< MAX ; i++)
   {     printf("this is doubly indented\n");    }
```

[L2] there are at least TWO lines beyond the maximum size of 87 (= wrapping)

[L22] a long instruction or function declaration/definition/call organized on more than one line must *align with element of the previous line* that makes it readable, for example with the start of the parameter list for function calls, or with the start of the evaluated expression in an "if". You can be flexible about the alignment start.

```
if(nb_robot > 0 && nb_obstacle > 0)
    deplace_robot( tab_robot, nb_robot,
                   tab_obstacle, nb_obstacle);
```

[R2] there is a global variable ; remember that global **constexpr**, **enum** or **define** are not penalized; instead these are good practices.