Computational Thinking and Programming – A.Y. 2018/2019

Written examination – 13/12/2019

Given name: _____

Family name: _____

Matriculation number: _____

University e-mail: _____

Group name: _____

Is it your first try? Yes | No

The examination is organised in three different sections:

- Section 1: basic questions [max. score: 8]. It contains four simple questions about the topics of the whole course. Each question requires a short answer. Each question answered correctly will give you 2 points.
- Section 2: understanding [max. score 4]. It contains an algorithm in Python, and you have explain what it does and to report the particular results of some of its executions according to specific input values.
- Section 3: development [max. score 4] It describes a particular computational problem to solve, and you are asked to write an algorithm in Python for addressing it.

You have 1 hour and 30 minutes for completing the examination. By the final deadline, you should deliver only the original text (i.e. this document) with the definitive answers to the various exercises that must to be written with a pen – pencils are not permitted. You can keep all the draft papers that you may use during the examination for your convenience – blank sheets will be provided to you on request.

Section 1: basic questions

1 – Which of the following algorithms is not based on recursion:

- Merge sort
- Linear search
- Line wrap
- Quick sort
- Insertion sort

2 – Consider the following function in Python:

```
def f(s1, s2, n):
    if len(s1) > n and len(s2) > n:
        return s1[n] == s2[n]
    else:
        return len(s1) - len(s2)
```

Write down the value that is returned by the function above when called as follows: `f("donald", "duck", 4)`

3 – Write down a small function in Python that takes in input a non-empty string `s` and a positive integer number `n` and returns a new string composed by the last character of the input string repeated `n` times.

4 – Introduce the *halting problem* and sketch out its resolution.

Section 2: understanding

Consider the following function written in Python:

```
def f(mat_string):
    c = 0
    lst = list()

    for chr in mat_string:
        n = int(chr)
        lst.append(n)
        if n / 2 != n // 2:
            c = c + 1

    return g(lst, c)


def g(mat_list, c):
    if c <= 0 or len(mat_list) == 0:
        return 0
    else:
        v = 0
        result = list()

        for i in mat_list:
            if v > 0:
                result.append(i)
            if i > 0 and v == 0:
                v = i

        return v + g(result, c - 1)
```

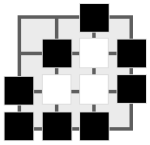
Consider the variable `my_mat_string` containing the **string** of your matriculation number as written in the first page. What is the value returned by calling the function `f` as shown as follows:

```
f(my_mat_string)
```

Section 3: development

AtariGo is a simplified version of Go. Its rules are pretty simple. Two teams, Black and White, take turns placing a stone (game piece) of their own colour on a vacant point (intersection) of the grid on the board. Once placed, stones do not move. A vacant point adjacent to a stone is called a liberty for that stone. Connected stones formed a group and share their liberties. A stone or group with no liberties is captured. Black plays first. The first team to capture anything wins.

Suppose you want to develop a software that is able to play AtariGo on a 4x4 board, like the one shown in the figure that is already populated with some stones. And suppose we use tuples for defining every position in the board (shown on the right side of the figure).



```
(0, 0) (1, 0) (2, 0) (3, 0)
(0, 1) (1, 1) (2, 1) (3, 1)
(0, 2) (1, 2) (2, 2) (3, 2)
(0, 3) (1, 3) (2, 3) (3, 3)
```

One of the functions to implement returns the set of positions of the board that are not occupied by any stone and that does not result in the stone being immediately captured if placed. Supposing White has to play in the board in the figure, such a function would return the set containing the tuples `(0, 0)`, `(1, 0)` and `(0, 1)` – but not `(3, 3)` since if White places a stone there it will be immediately captured.

Write a function in Python – `def get_good_white_moves(white, black)` – that takes in input the set of tuples identifying the stones placed in the 4x4 board in the previous turns by the two players (`white` and `black`, respectively) and that returns the set of all the tuples identifying possible places where White can put its stone in the current turn, according to the rules mentioned above.