PART 2. REPORT

1. **A common application of the TAD PILA is to evaluate postfix expressions. Describe in detail what TADs should be defined / modified in order to adapt the stack of integers (P2_E1) so that we have one that allows us to evaluate postfix expressions.**

To adapt our stack ADT to evaluate postfix expressions we would need to follow the next steps:

1. While reading the expression from left to right, push the element in the stack if it is an operand.
2. Pop the two operands from the stack, if the element is an operator and then evaluate it.
3. Push back the result of the evaluation and repeat it until the end of the expression.

To do so we would need to implement a new function to evaluate the expression, which makes use of the push and pop functions using the criteria defined in the steps.

2. **Explain the design decisions and alternatives that have been considered during the practice. In particular, explain in detail the implementation and design decisions of the exercises P2_E2, P2_E3 y P2_E4.**

For a start, in the **P2_E2** our target was to use a stack that contained integers, and do the mean of all of those integers.

To achieve this goal, we needed to implement a function that made the mean. This function consisted on emptying the stack while reading the numbers and doing the mean. At the same time, we passed the numbers into an auxiliary stack. When the original stack was empty, we just needed to pass the numbers back from the auxiliary stack to the original one and return the mean.

About the **P2_E3**, the aim was to modify the Stack ADT so that it could store any type of data in a dynamic way, which will allow to have several stacks while storing different elements.

The Stack ADT was modified to have an array of void items, instead of the previous elestack ADT. The Stack was also modified by adding to the structure the functions needed to destroy, copy and print any type of data.

The primitives had to be modified so that their arguments were type void (to store any type of data).

To check the correct functionality of the new Stack ADT, two main.c were done. One operates with nodes, and another one with integers.

Finally, the purpose of **P2_E4** was to check if a path between two given nodes existed.

To achieve it, we needed to implement two new functions, one to check if there was a possible path between the nodes, and another one to print it.

The function to check the possible path goes over all the connexions of the given node changing a label from white to black, this nodes to its connexions and so on. If the one we want to find a path to has its label as black, it means we have found a path.

For printing it, we use recursion (calling a function from itself), where we print each time the antecessor id of each node from the path.

3. **FINAL CONCLUSIONS: Include, at the end of the report, some brief conclusions about the practice, indicating the benefits that the practice has brought, what aspects seen in the theory classes have been reinforced, which sections of the practice have been more easily solved and which have been have been the most complicated (or have not been finally resolved), what new aspects of programming have been learned, what difficulties have been encountered, etc.**

This practice has brought up to us challenges which would have been impossible to manage without the theory classes contents, such as the management of a stack or even recursion, as seen in the last part of the practice.

We have considerably improved the ADT's implementation ability, since we were asked to develop programs with different input formats along the different weeks (from integers to nodes or a generic input format).

The greatest issues we have found were mainly from trying to avoid memory leaking, due to te amount of pointers that we implemented clearly wrong. However, we managed to solve most of it.

From our experience, the most interesting aspect we have learnt would be the depth search algorithm we developed for the last part of the practice. This algorithm closed the gap between theoretical and practical uses for programming.