**2. Exemple adecvate si semnificative care sa ilustreze necesitatea unui asemenea mecanism la nivelul limbajelor de programare**

Example 1

Imagine I tell you "Cook dinner for me!" with nothing else. Can you complete my request? No, because I didn't tell you what I wanted you to cook. In other words, I did not provide enough information for the task to be completed.

Command line arguments can be seen as the information I did not provide in the example mentioned earlier. They are a way of accessing information outside the environment of the program which is vital for the completion of the program.

Example 2

Imagine I tell you “Play football”. You will probably play football whoever you see it fit. Now imagine I say “Play football with Michael” or Play football as a striker”. You will probably change the way you play football to satisfy my condition.

Command line arguments can be seen in this example as those details provided on how to play football. They are the specifics of how the program must do its job or with what resources it has to work. Nonetheless, they may not be necessary, as seen from the example when the first command “Play football” can be fulfilled without any other instructions. However, command line arguments allow the program to be flexible.

Small comparison

Command line arguments can be seen as the input of a program to be executed in the same way functions in programming languages have parameters for the work they are about to do (with the assumption that input is necessary for the program/function to do its job). So, any examples on the necessity of this mechanism in programming languages can be mirrored with any examples on why we need parameterization for functions in programming languages.

**3. O analiza critica si comparativa a modului in care o astfel de functionalitate este implementata la nivelul diferitelor limbaje de programare**

*Python*

In contrast to C which offers access to the command-line arguments through 2 arguments in the main function, Python offers a module, namely the Python sys module, which provides access to any command-line arguments via the sys.argv variable. The sys.argv variable is of type ‘list’ where all elements are of type ‘string’ and ,similar to C though, the first element is the name of the program. The length of this list can be found through the built-in function ‘len’, which can be applied to any iterable data structure in python.

Program for printing eventual command line arguments:

import sys

for i in range(len(sys.argv)):

print(‘arg[‘ + str(i)+’] = ‘ + sys.argv[i])

*Java*

Similarly to C, Java offers access to the command-line arguments through an argument in the main function (called ‘args’). However, closely to Python, this argument is a list (more precisely an array) of strings. Nevertheless, one notable distinction from C, Python and Turbo Pascal is that the first value in this list is no longer the name of the program, but the first argument passed.

Program for printing eventual command line arguments:

public class CommandLine {

public static void main(String args[]) {

for (int i = 0; i < args.length; i++)

System.out.println("args[" + i + "]: " + args[i]);

}

}

*Haskell*

Considering the fact that Haskell is a functional programming language, it is only normal that the mechanism by which it offers access to command-line arguments is similar to Turbo Pascal, namely through a function (getArgs). getArgs has a return type of ‘IO [String]’ and is an I/O action. However, this function is defined in another module (like in Python) which has to be imported (System.Environment) and, resembling Java, this function returns only the arguments without the name of the program. Still, the name of the program can be retrieved through another function call, namely getProgName, whose return type is IO String and is also an I/O action.

Program for printing eventual command line arguments:

import System.Environment

import Data.List

main = do

args <- getArgs -- IO [String]

progName <- getProgName -- IO String

putStrLn "The arguments are:"

mapM putStrLn args

putStrLn "The program name is:"

putStrLn progName

**4. Propuneri de extindere/imbunatatire**

One possible extension of the current mechanism for obtaining command line arguments would be the ability to specify the data type of the arguments. This extension would have the following characteristics:

* The data types available would be a list of primitive data types, namely *boolean, integer, float, char, string.* The choice to only allow a predefined set of data types is derived from the following 2 reasons. Firstly, a mechanism for allowing the user to define a datatype at the command line level seems unnecessary since not all programming have the ability to process a user-defined datatype, and those that do, already provide this mechanism. Secondly, most arguments at the level of command line interface denote already a primitive data type specified earlier.
* The specification of data type for an argument would be done in the following way: the data type of the argument, followed by a colon and then the actual argument, ex: “string:doc.txt”, “float:10.5”.
* The arguments would still be separated by a blank space. Ex “program1 int:10 string:out.txt”
* The arguments would still be given as a list of strings inside the program. Even though with this mechanism we would know the data type of each argument, we still can not get over the problem of message sharing between the operating system and the programming language. However, these strings could be split around the colon and know the data types of the arguments immediately, in contrast to what is happening now where each string has to be interpreted and its data type guessed based on its position in the list of arguments.

The main advantage of this mechanism would seem to be the diminishment of the number of errors which happen at run-time due to wrong casting of command-line arguments. This would happen since now we would know to what data type we should do the cast, thus giving the responsibility of providing the correct type to the user.