Project 2 Report

Group

T05_G03

Group Members and Contributions

- Bruno Miguel Lopes da Rocha Fernandes (up202108871) - 50%
- Vasco Moutinho de Oliveira (up202108881) - 50%

Project Description

This project was divided in two parts. The goal of the first part was to implement a low-level machine with a stack-based architecture. The second part consisted in implementing a small imperative language with arithmetic and boolean expressions, variables, conditionals and loops.

Implementation

Part 1

The main data types used were `Stack`, `Inst`, `Code`, `StackValue` and `State`. We created a function for each instruction. Then we created a function `exec` that takes a `Code` and a `State` and returns a tuple with `(Code, Stack, State)`. This function executes the first instruction in the `Code` and returns the new `State`. We also created a function `run` that calls `exec` until the `Code` is empty.

Part 2

In this part we created the data types `Aexp`, `Bexp`, `Stm`, `Tokeb` and `Program`. Firstly, we created the function `lexer` that takes a string and returns a list of Tokens. Then we created the `compiler` function that takes a list of Statements and returns a list of Instructions. We also created `compA` and `compB` that take an arithmetic expression and a boolean expression, respectively, and return a list of Instructions. Then we created the `parse` function that takes the list of Tokens and returns a list of Statements. This function calls `parseProgram` that will parse the whole program. For this, we created different functions to parse either arithmetic expressions, boolean expressions or statements.

Final Remarks

With this project we learned how to implement a low-level machine with a stack-based architecture and how to implement a small imperative language with arithmetic and boolean expressions, variables, conditionals and loops, all this using Haskell.