Interpreter for pseudocode used in IB Diploma Programme

Igor Krzywda

June 2020

0.1 Criterion A

0.1.1 Current situation

Having spent almost a full-year attending IB computer science classes, I observed an issue. We are required to know pseudocode, yet in order to explain basic concepts we use Java, which is good for exercise, but not so for sole explaining. The only form in which we use pseudocode is to learn the semantics, which combined with a very modest web editor is a time consuming, inefficient and frustrating process.

The main issues of existing web-based solution are:

1. The tool is described by the author as a "quick and dirty hack", which makes it very modest and buggy at times and in effect quite a nuissance in longer sessions.
2. The tool is based on search-replace commands, so there are no lexical, syntax and semantic analyzers making it impossible to find errors made by the user.
3. There is no accessible output in target code, preventing users from deeper analysis.
4. The program is web based, which makes it unstable just by relying on school Internet and requiring a tab in a web browser.
5. Saving projects is clunky.

0.1.2 The clients

The target clients for this software are students and teachers that want to learn and teach computer science concepts using pseudocode in practice by building simple scripts. The program is not aiming to replace the primary language used in lessons, but rather as a tool that will enable to write prototype algorithms fast and with understanding before jumping back into the language of choice. There is also a subset of clients who are tinkerers and want to look "under the hood" and learn the basics of how language translators work by fixing the code or even adding their own content on GitHub, which could make an opportunity for gaining experience and even CAS activities.

0.1.3 The solution / plan

The program will be a pseudocode translator enabling the clients to code in pseudocode in their favourite text editor. The program will be used via command-line in the terminal, just like any other interpretator/compiler. The code will be written in C[++ if I don't manage], which will ensure compatibility with majority of machines. Program will be accessible to anyone in the form of source code on GitHub.

0.1.4 Existing alternatives

So far we have used a web-based pseudocode interpretator written in JavaScript. The script is very minimalistic and quite buggy. I intend on building an open-source alternative that will enable any form of troubleshooting in form of making changes to the source code. Because my program will be written in C, the CPU and memory usage will be very low, which will give a better experience given that the client will neither be reliant on Internet connection nor on RAM-hungry web browsers. The existing editor: http://ibcomp.fis.edu/pseudocode/pcode.html [to be edited]

0.1.5 Criteria for success

1. Program successfully runs all pseudocode aspects according to IB's requirements described in https://ib.compscihub.net/wp-content/uploads/2015/04/IB-Pseudocode-rules.pdf
2. With the use of a flag, the program returns resulting code in C for deeper analysis.
3. Program returns errors concerning syntax like no loop termination.
4. Program returns basic logical errors like index out of bounds.
5. Program successfully compiles on Linux and Windows machines.
6. Program is accessible under MIT licence.