Criterion E

Word count: 323

1 Success criteria

As success criteria I set were expressed as features, they were shown in the film, below is the original list, where green shows that criterion was met and red that it has not.

- executing IB pseudocode consistent with developed standard based on documentation and exam booklets made by IB:
  - methods
  - recursion[1]
  - containers and ADT's
    - fixed-sized, multidimensional arrays
    - stacks
    - queues
  - while and "from to" loops
  - if else statements
  - arithmetics with operator precedence
- throwing lexical errors (undefined characters)
- throwing syntax errors
- throwing run-time errors
- flag to toggle printing abstract syntax tree
- flag to toggle logging and printing call stack
- cheat-sheet available with a flag

Additional goals:

- whole syntax defined in BNF (Backaus-Naur form)

[1] recursion does not work in all cases and needs fixing

2 Client feedback

I gave my friends link to repository with the project [2] for them to compile and test the program. This was an issue, as no one had C++ environment to compile the project, so not everyone tried it. However from those who compiled it the feedback was rather positive[3]. However the biggest test of the program was during a lesson where I shared my screen and we were solving pseudocode tasks. The program completely changed the dynamic of the lesson, as then we could not only find errors quickly, but also we were tinkering with different solutions to problems. After the lesson, I asked our teacher about his opinion and the feedback was positive[4], but the issue was that the program has to be compiled from source, so not everyone could use it.

3 Area for imporvement

3.1 Not building one global AST for a program

The biggest inefficiency of the system is the fact that the AST is built as a whole and only then is it fed into the interpreter. This makes for wonky error detection, as the error methods are scattered across lexer, parser, activation record and interpreter. A much more effective approach would be to feed every tree built from a nonterminal and evaluate it on the spot. If the tree is not a method declaration, the tree is disposed of, in other case, it would be saved to a map of methods, as it is done now. Errors could be reported through returned values to a central error function. Additionally, the memory use would be cut down by a significant amount, as only the AST's of methods would be stored, making for more reliable and extensible system.

3.2 GUI

Because this program is targeted at students, GUI would not add to already steep learning curve of programming for beginners.

[2] `https://github.com/ikrzywda/ibpci`
[3] See Appendix C for full thoughts on the project
[4] See Appendix C