README

This is the readme for a compiler for the COS341 main project, created by Justin Hudson (u21543152) and Nicholas Harvey (u22522116). It accepts input as RecSPL code, and outputs a token list, a syntax tree, a symbol table, and intermediate code.

OVERVIEW

This compiler is distributed as a JAR file (specifically Main.jar in the same folder as this README.) As such, please ensure that your device has Java installed and up-to-date.

Source code is processed from a file named input.txt in the same folder as Main.jar. An example is included with this distribution; it is suggested to run it first before testing any more advanced cases so that the program's functionality can be displayed to an extent. Example output is also included in the 'resources' folder, as detailed below, which may be of use in the unfortunate eventuality that the program is incompatible with one's device.

The source code can be found at https://github.com/u21543152/SPLCompiler (note that the README on the Github is not up to date, and this document is.)

RUNNING THE PROGRAM

The program is run using the run.bat script within the same folder as this README, or alternatively from a terminal using 'java -jar Main.jar'. Upon running, it will read the input file input.txt and process it, generating several output files in the 'resources' folder that is in the same directory as this README. These files are described below.

OUTPUT FILES

output.xml
This file contains the RecSPL tokens extracted from the source code.

syntax_tree.xml
This file contains the parsed syntax tree in XML format.

syntax_tree.dot
This file is a representation of the syntax tree in DOT format, which can be visualised online using Graphviz at https://dreampuf.github.io/GraphvizOnline/ .

error_log.txt
This file is what the error- and out-streams are directed into, and may be useful for providing insight on where things went wrong.

intermediate_code.txt
This file contains the intermediate code between RecSPL and BASIC.

symbol_table.txt
This file contains a printout of the symbol table, and is also the means of conducting type checking. Any parsed variable that is later assigned to a different type than its own will be given a type of ERROR, and a warning will be displayed. Variables with the type of 'flex' may be ignored, as they are used exclusively for function parameters. A warning will also be displayed for any variable assigned based on user input; the type of the user input must match the variable's type or else a type error will occur.

The second table below the symbol table proper is the unique identity table, used for the code generation. While not strictly necessary for type checking, it may nonetheless be of interest.

REUSING THE COMPILER

For best results, 'reset' the compiler by deleting all output files after using it to process a given input.txt. You can also accomplish this by simply deleting the 'resources' folder - a new one will be generated if necessary when you run the program again.

A suggested alteration to make to the example input.txt for the sake of verifying functionality is to adjust line 21, which reads 'V_var1 = add (V_param2 , V_param3) ;', to instead be 'V_var1 = add (V_b, V_param3) ;'. It can be observed that the symbol table will indicate a type error, as detailed below, due to attempting to use a text string in an addition operation.