**User Manual**

Please follow the steps of each part after unzipping the submitted folder “Finale Deliverable”

**Part 1:**

1. Open the folder “Part 1 revisited”.
   1. The documentations of the AL and the PL are provided in the two pdf documents.
   2. Since we modified the language, the text file “What has changed!” mentions all the changes we performed.
   3. To execute the assembler and the interpreter, open the folder “Codes” then:
2. As you can see, there are four text files. For now, ignore the one named “AL Code.txt”. The three other text files are the ones we used to test the assembler and the interpreter.
3. Open the C codes.
4. We start by running the assembler. As you can see, the input file is set by default to the “AL Simple Arithmetics.txt” file, if you want to run the assembler with another file, you can copy its name from the comment provided right under the preprocessor directives.
5. Once running the Assembler with any of the three chosen files and receiving a success message, please go back to the folder in which the codes are stored, and you will find that a text file named “ML Code.txt” was generated. This is the input file to the interpreter.
6. Now we can run the interpreter with will display to the scree the output of the ML program.
   1. The folder “Assembler and Interpreter Expected Outputs” provides expected outputs of the assembler and the interpreter for each input file.

**Part 2:**

In this part there are no steps to follow, we just submitted what has been fixed/modified in the grammar.

**Part 3:**

This part is not very straight forward, we really tried to think of a way to simplify the steps, and this is what we could get:

1. Open the folder “Part 3 revisited”.
   1. The documentation of the lexer is provided in the pdf document.
   2. Since we made some modifications to the syntax of the symbol table, the text file “What has changed!” mentions all the changes we performed.
   3. To achieve a successful execution of the lexer, please follow these steps:
2. Open the folder “Input”. For now, only focus on the text file “inp.txt” and ignore the other files.
3. Copy the text file “inp.txt” to the home directory of your computer:  
   (for Windows OS: C:\Users\{current\_user\_name})   
   (for Mac OS: /Users/{current\_user\_name})
4. Go back to the original folder (i.e., “Part3 revisited”)
5. Open the folder “Lexical Analyzer Classes”.
6. Open your Java IDE (we used NetBeans).
7. Create a new project (the name doesn’t matter).
8. Copy the java classes in the folder “Lexical Analyzer Classes” to the default packages of the java project you just created.
9. Run the class Lexer.java.
10. Now if you go to the home directory again, you will find that three text files were generated. These correspond to the token stream, symbol table and literal table of the input source code.

If you want to test the lexer with other input files, under the folder “Input files”, we provided four other input files, please follow the same steps with the desired file, but make sure to change its name to “inp.txt” once copying it to the home directory.

**Part 4:**

1. Open the folder “Part 4 and 5”.
   1. The documentation of the parser is provided in the pdf document.
   2. To Execute the parser, please open the folder “Codes”, and follow these steps:
2. In this folder you can see that we have the input files ready (the token streams and the symbol tables of different programs). These files are the ones that were generated by the lexer when we executed it with each of the four source codes provided in part 3 (other than inp.txt).
3. Open the three C programs provided in the folder.
4. We start by executing the Acceptor:
5. As you can see in the 2nd #define macro, the input file to the acceptor is set by default to the “Simple Token Stream.txt” file, if you want to run it with another file, you can copy its name from the comment provided right under the preprocessor directives.
6. Once executing the acceptor with any of the four input files, it will generate a text file named “Parse Tree” after accepting the program. The generated file can be found in the same folder as the acceptor code.
7. We can now execute the static semantics:
8. Just run the “Static Semantics.c” program which will output a message confirming the success of the static semantics phase.
9. You can run the static semantics on the parse tree of any of the four programs.
10. We can now execute the generator:
11. The generator only works on the simple program, therefore, please make sure to run it only on the parse tree generated by the acceptor when running it on the simple token stream.
12. The generator will generate a text file named AL Code.txt; this exact same code is the one we encountered in Part 1.
13. You can now go back to part one and run the assembler on the text file “AL Code.txt”, you only need to change the value of the #define macro to “AL Code.txt”.
14. Then, once running the assembler, you can run the interpreter right away, and this is supposed to give the output: 10 0 25 1 (because it does the addition, subtraction, multiplication, and division of the numbers 5 5).