CS 4031 Compiler Construction

Assignment 4

Section BCS-6A and BCS-8A

NOTE: Don't resort to cheating or any form of plagiarism. Submit the assignment on time otherwise there will be -5 penalty for every hour.

To Submit: A zipped folder named "group #.zip" (# is your group number). It should contain all files present in the provided template (your code) along with a file named "symbol_table.txt" containing your symbol table (generated by your parser) and "Grammar.txt" file containing your grammar. Only 1 student in a group has to submit the assignment.

Penalties:

- -5 marks for not following submission instructions
- -5 for messy coding
- -5 for non-runnable (submitted) code
- Late penalty is -5 every hour (this will be followed strictly from now onwards)
- Grammar.txt should contain your grammar else (-10)

Instructions:

- Program takes a file name as command line argument and outputs symbol_table.txt
- Unlike DFA, Parser is executed only once and whole analysis is done
- The start rule is called manually, rest is done by the parser
- Lexer (DFA Implementation) is executed first and tokens are made
- These tokens are used by parser one by one to do syntax analysis, semantic analysis and many more. We will be focusing solely on syntax analysis for this assignment
- Parser outputs "syntax error" and exits if there is some syntax error
- Helper functions named peek (in lexer) and expect (in parser) are already implemented in template. These two functions are sufficient to implement parser.
- User doesn't need to know if a lexer existed or not. What user do is just initializes a parser instance and the parser (containing a lexer instance) calls lexer constructor and populates its token array. Then Parser may

call lexer methods again and again and do what it is supposed to. Keep that in mind while coding.

- There is some commented code in the end of “parser.cpp” use it as reference if you are using the template
- If you define some new function add its definition in respective header as well. Remember good code is called good because it follows good programming practices.

Recursive Descent Parsing:

There are many kinds of parsing, we will just be focusing on recursive descent parsing. It is also called bottom-up parsing. It has some flaws which needs to be addressed in grammar (to be implemented). e.g., it doesn’t work with left recursive grammar. So, you need to remove that left recursion before you do anything. Operator precedence comes second. Remember operators with high precedence is present in a deeper level in parse tree than operators with lower precedence. Precedence of %, * and / is greater than + and -. In this parsing, rules of your grammar are functions. Write code according to your grammar.

Bonus Work: (given when project is complete)

Everyone (if he/she is using the template or not) is required to download the template.zip archive from A2 and see provided code samples. There are 4 test cases provided ranging from sample_code0.mc to sample_code3.mc. There are descriptive comments on top of every file. Read them, if you are able to do what is being asked, you will be given bonus marks.

Assignment Details

Implement grammar for my_compiler language keeping the sample codes in mind. If you want to do bonus work, you should be able to run that specific sample code without errors. If you are able to run source_code0.mc you will be given full credit. Rest samples are for bonus work. Code (to be executed) can be changed during evaluation. Template users can just extend their template to support parser. **Read above guidelines carefully**, **otherwise you may lose marks.**

TA email: l180968@lhr.nu.edu.pk