CS 315
Project 2

Assigned: Nov. 7, 2018
Due: Nov. 15, 2018 23:59

Parser for a Robot Programming Language

The second project builds on your language design of the first project. This project involves building a parser for your design using the yacc tool. Please refer to the description of Project 1 for the requirements for your programming language design. There are some minor changes, please carefully read the instructions below.

Part A - Revised and Augmented Language Design (25 points)

The requirements for the language are the same as Project 1 except for a few minor extensions. You can use as much of your previous design work as you can. However, if you have not already done so, you should incorporate the following elements into your design for this second project:

- main program statements (beginning of the execution)
- declarations (variables, constants etc.)
- assignment statement
- Selection statements
- Loop Statements
- Input and output statements
- function definition and function call statements

Please note that we are not expecting a single correct answer. This is a design problem. As long as your language is consistent, unambigous and it makes sense with respect to the specifications given above, it is fine.

Part B - Implementing the Parser (65 points)

For the second part of the project, you are required to implement a parser using the yacc tool. The parser reads the source code of a program written in your programming language from the input file. If the source code represents a valid program in your programming language, the parser should print out a message indicating the acceptance of the input (e.g. "Input program is valid"). Otherwise, the parser should print out an error message indicating the line number of the source code that contains the error (e.g. "Syntax error on line **!" where ** will be the line number of the source program at which the error was detected).

You should use the lexical analyzer that was developed in the first part of the project, but you may have to modify it (for example, to count line numbers). Also, the lexical analyzer will return tokens, instead of printing messages.

VERY IMPORTANT NOTE:

- Your yacc and lex files must compile in the `dijkstra.cs.bilkent.edu.tr` machine; otherwise, you will receive 0 from Part B.
- You should strive to eliminate ALL conflicts and ambiguities in your language, modifying your grammar if necessary. You will need to provide unquestionably convincing arguments for any conflicts that are left in your final submission.

Part C - Example Program (10 points)

Finally, you will have to submit a test program written in your language that exercises all of the core features we required in the project. You can reuse the test program from your first project, but note that you have to make sure to revise it according to the changes you made in your grammar.

VERY IMPORTANT NOTE: If you do not submit a test program, we will have no way of evaluating your parser, hence you will receive a 0 from Part C!

Logistics

- You will be working with the same group you worked for Project 1.
- There are two parts that you will hand in before the due date of the project.
    1. A project report (in PDF format) including the following components:
        - Title page with your **group name and ID** as well as **names, IDs and sections** for all of the project group members.
        - The complete BNF description of your language (based on the terminal symbols returned by your lex implementation)
        - General description of the structure of your language and those nonterminals that you think are important. Try to make the life of the grading assistant as easy as possible by making sure that somebody reading your report can understand and parse through a program written in your language. Make sure to note all rules adopted by your language (i.e. precedence rules and other ways in which ambiguities were resolved).
        - Descriptions of how each nontrivial token used in your grammar.
        - A thorough explanation of every conflict left unresolved in your final submission. Ideally, you should strive to eliminate all conflicts with no warnings or conflict errors given by yacc on your specification file.
    2. Your lex and yacc description files, together with the example programs described above, written in your language. Specifically, do the following:
        - Create a folder named **CS315f18_groupXX** where XX will be your group number.
        - Copy into this directory the following files:
            - The project report (in PDF format).
            - **CS315f18_groupXX.lex** : Your lex specification file.
            - **CS315f18_groupXX.yacc** : Your yacc specification file.
            - **CS315f18_groupXX.test** : Your example program.
            - a **Makefile** that produces your complete parser with an executable called **parser** in the `dijkstra.cs.bilkent.edu.tr` machine. Check that when you type **make** in the same directory the desired executable is generated.
            - Before you proceed with the next step, you should delete all other files in this directory using the Unix **rm** command. BE CAREFUL, do not remove your lex and yacc files. MAKE FREQUENT BACKUPS.
        - Compress this folder into a single file using tools such as zip or rar.

Submission

Please email the zip (or rar) file you created to Gizem Çaylak before **23:59 on due date**. Late submissions will be accepted, with 20 points (out of 100) deduction for each extra day..

If your submission does not adhere to the above guidelines, points will be deducted.
Make sure you have correct file naming.
Your parser must compile and run on `dijkstra.cs.bilkent.edu.tr`. The evaluation of your parser will be done only on this machine.
