CS 315
Project Part 1

Assigned: Feb. 18, 2022
Due: Feb. 27, 2022, 23:59

A Programming Language for Sets and its Lexical Analyzer

This semester's projects are about the design of a new language for finite sets. This newly designed language will be similar to imperative languages with the main difference of only working with variables and expressions of mainly the set type.

Part A - Language Design (40 points)

First, you will **give a name** to your language and design its syntax. Note that the best way to hand in your design is its grammar in BNF form, followed by a description of each of your language components. The following is a list of features required in your language:

- Representing sets and set elements (syntax of a set and set element)
- variable identifiers
- operations for creating and deleting a set
- set operators (e.g., union, intersection, ...)
- set relations (e.g., subset, superset, ...)
- assignment operator
- statements to input/output sets, from/to keyboard/console or file.
- conditional statements (if, if-else, ...)
- loops
- primitive functions (e.g, cardinality, ...)
- function definitions and function calls
- comments.

All of these features must be built-in your language. Do not assume importing from a library.

You are encouraged to use your imagination to extend the list given above.

For the sake of simplicity, you can assume that set elements can not be sets.

You will have a chance to do minor revisions on your syntax design for Project 2, to be assigned later. Language designs are almost never exactly right in the first iteration. Just try your best to make it as readable/writable/usable/reliable as you can and keep your eyes open for what does and what does not work :)

Part B - Lexical Analysis (30 points)

In the second part of this project, you will design and implement a lexical analyzer for your language, using the lex tool available on Unix style systems. Your scanner should read its input stream and output a sequence of tokens corresponding to the lexemes defined in your language. Since at this stage you will not be able to connect the output to a parser, your scanner will print the names of the tokens on the screen. For instance, if we were designing a C like syntax, for the input

```
if ( answer == 2 ) { ...
```

the lexical analyzer should produce an output, similar to the following:

```
IF LP IDENTIFIER EQUAL_OP NUMBER RP LBRACE ...
```

Part C - Example Programs (30 points)

Finally, you will prepare test programs of your choice that exercise all of the language constructs in your language, including the ones that you may have defined in addition to the required list given above. Your test programs should include at least the ones given below:

a. a program that reads a set from a file, whose name is given,
   displays the elements on the console, or a special message if the set is empty.
b. a program that initializes a variable to a set of elements of your choice,
   initializes the second variable to a set of elements of your choice,
   initializes the third variable to the union of the sets in the other two variables,
   displays the elements of the third variable,
   saves the contents of the third variable to a file,
   deletes all of these variables.
c. a program that defines a fuunction that thats a set and element as its argument,
   removes that element from the set and returns the remaining elements,
   reads a set from a file,
   removes an element of your choice from the set by calling the function defined,
   displays the resulting set.

Make sure your lex implementation correctly identifies all the tokens. The TA will test your lexical analyzer with these example programs along with other programs written in your language.

Do not panic! You are not required to write an interpreter or compiler for this language. Just write a few programs in the language you designed and make sure that the lexical analyzer produces the right sequence of tokens. Be creative, have some fun.

Groups

The project can be implemented in groups of two or three students. The members of the groups will be the same for both parts of the project, this and the next.

Submission

- There are several parts that you will hand in.
    0. A project report including the following components:
        - Name, ID and section for all of the project group members.
        - The name of your language.
        - The complete BNF description of your language.
        - One paragraph explanation for each language construct (i.e. nonterminals) detailing their intended usage and meaning, as well as **all of the associated conventions.**
        - Descriptions of how nontrivial tokens (comments, identifiers, literals, reserved words, etc) are defined in your language. For all of these, explain what your motivations and constraints were and how they relate to various language criteria such as readability, writability, reliability, etc.
        - Evaluate your language in terms of
            . Readability
            a. Writability
            b. Reliability
    1. The lex description file.
    2. Example programs, written in your language.
- Make sure your lexical analyzer compiles and runs on `dijkstra.cs.bilkent.edu.tr`. The TA will test your project on the dijkstra machine, and any project that does not compile and/or run on this machine will get 0 on Part-B.
- Please upload **all of the above items** to Moodle before the due date. PDF format is preferred for the project reports. *Late submissions will be accepted, with 20 points (out of 100) deduction for each extra day..*

Resources

- Running lex and yacc on Linux systems (accessible in Bilkent Campus)
- The Lex & Yacc Page (dinosaur.compilertools.net)
- Discrete Mathematics - Sets

Good Luck! - Have fun.

CS 315
Project Part 2

Assigned: Mar. 7, 2022
Due: Mar. 16, 2022 23:59

Parser for a Programming Language for Sets

The second project builds on your language design of the first project. This project involves building a parser for your language 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 read carefully the instructions below.

Part A - Revised and Augmented Language Design (20 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 the second part of the project:

- Specification of the beginning of the execution,
- Declarations (variables, constants etc.),
- Operations for creating and deleting a set,
- Assignment statement,
- Expressions (involving set operations, relations, boolean operations, their combination),
- Precedence, associativity of the operators,
- Conditional statements,
- Loop Statements,
- Statements to input/output sets, from/to keyboard/console or file.
- function definition and function call statements.

- Comments.

Please note that there is no single correct answer. This is a design project. As long as your language is consistent, unambiguous and it makes sense with respect to the specifications given above, it is fine. However, it is expected to be readable, writable and reliable, as much as possible.

Part B - Implementing the Parser (60 points)

For the second 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 an 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 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` specification 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 test your parser on the programs that you submitted in the first part of the project. Also show that your parser finds syntax errors by introducing small error in these programs.

Make sure you test your parser for edge cases, such as empty sets, complex expressions, and void return values of functions.

Part D - Teamwork (10 points)

You will be working with the same group you worked for Project 1. Since this is a team project, each member is expected to put about the same amount of work into the project. However, sometimes this is not the case. The remaining 10 points of your grade will come from the peer assessment. Each member will evaluate him/herself and the other members of the team. You will receive an email message about how you will submit your evaluations, later. If you do not submit your evaluations, your teamwork grade will be 0. The teamwork grade of a team member will be computed by taking into account the comments written and the grades received. Keep in mind that the most important part of your evaluations is the comments section. Needless to say, do NOT collaborate in preparing and submitting your peer assessments.

Logistics

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 you note all rules adopted by your language (i.e. precedence rules and other ways in which ambiguities were resolved).
    - Description of how each nontrivial token is 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 **CS315s22_teamXX** where XX will be your group number.
    - Copy the following files into this directory:
        - The project report (in PDF format).
        - **CS315s22_teamXX.lex** : Your lex specification file.
        - **CS315s22_teamXX.yacc** : Your yacc specification file.
        - **CS315s22_teamXX.txt** : Your example program, in text format.
        - 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 upload the zip (or rar) file you created to Moodle (CS 315 (All Sections) Programming Languages) before the 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.