Assignment I - Lexical Analyzer

Compilers II (CS3423)

Deadline: 27th August 2023

1 Objective

To build a lexer that analyzes an input program P and outputs two entities: (1) the corresponding C-variant of P and (2) each token in P in a new line. The learning outcomes of this assignment: (1) Writing regular definitions, (2) Program translation, (3) Understanding Lex tool.

The details of the input program format are provided in Section 2. The deliverables of this assignment are mentioned in Section 3, and the grading pattern is mentioned in Section 5.

2 Sample Program

Consider a sample program in Figure 1. Input to your lexer is a program written in a C-like language with the following difference in language features.

- In this language, each statement will have a unique id starting with `pp` followed by a number. Here `pp` refers to 'program point', and the number is always a positive integer. `pp` is a reserved keyword and cannot be used as a variable name.
- The square braces imply the start and the end of the scope, and also they can be used to show the predicate during conditional branching (e.g., `in case that [x==4]`).
- `integer_2`, `character_1`, and `string` are pre-defined data types followed by their corresponding variable names (the numbers at the end of these datatypes represent their sizes).
- Variable names can contain digits and special characters, but the special characters can only appear at the beginning. Allowed special characters in variable names are: @, #, *, -, +, /, \, : and _.
- Operators +, -, *, / has the same meaning as in C (addition, subtraction, multiplication & division), but _ is an operator for square root. For example, at line `pp4`, `x_2` indicates the operation $\sqrt{x}$. Note: `x2/1x` is not an identifier and is an expression signifying the division operation. Hence, `x2/1x` cannot be a valid variable name, but `/x21x` is a valid name. Similarly, the `x_2` cannot be a valid variable name, but `_x2` is valid.
- Some reserved keywords which cannot be used as variable names. The list of the reserved keywords and their C equivalence are: `in case that` (is same as C-lang `if`), `otherwise` (same as `else`), and `jump to` (same as `goto`). Similarly, keywords `gteq`, `gt`, `lteq`, `lt`, `and`, and `or` represent >=, >, <=, &&, and ||, respectively. However, adding alphanumeric values post or before a reserved keyword makes a valid variable name. For example, `gt` is a reserved

```
pp1: null foo()
pp2: [
pp3:     integer_2 x1, x2, 1x;
pp2:     character_1 c1, 1c, _c;
pp3:     string s = "Hello World\n";
pp4:     x1 = x2/1x + x_2;
pp5:     in case that x1 gt 2 and 1x gteq 3
pp6:         do 1x=2;
pp7:     otherwise
pp8:         jump to pp4;
pp9: ]
```

Figure 1: An example format of the input programming language

keyword, but `gt1` or `1gt` is a valid variable name. Similarly, at line `pp1`, `null` is a keyword that refers to the return type of the function, `foo`. Likewise, `do` is also a keyword.

3 Assignment Submission Guidelines

3.1 Files to be Submitted

You should submit the following items:

- `overview.pdf`: Your overview document for the assignment. This file should contain
  - your name
  - your college ID
  - end-to-end compilation steps of `lex.yy.c`
  - clear instructions on how to run your analyzer on an input program
  - all known issues you have with your implementation
  - brief explanation on why you have chosen to use the corresponding regular definitions
- `Lex Source Program` and `lex.yy.c`: You should include everything required to compile and run the project. Please ensure that the directory structure of your source files is maintained within the archive so that your code can be compiled upon extraction. Your build process must **not** download anything from the internet.
- C program files generated by the lexical analyzer corresponding to input test cases. For example, `C_1.txt` is the output of the input test case `1.txt` (`1` is the test case number).
- `Sequence of tokens`: You should also provide the sequence of tokens generated by your lexical analyzer on the public test cases in a file `seq_tokens_i.txt` (`i` is the test case number) and submit the same. The file format of `seq_token_i.txt` is shown in Figure 2. The file should contain first your name and college ID followed by the name of the input files and the corresponding output from the lexical analyzer after running it on the input files. The output data will contain multiple lines, where each line first describes the nature token and then followed by a colon and the token itself. Violating the format may lead to the exclusion of the same during evaluation.

```
Name: John Doe
ID: CSXXXX11004
id: io
id: main
integer: 10
character: x
... \\more lines (not shown for brevity)
```

Figure 2: An example format of `seq_tokens_i.txt` file

3.2 Assignment Folder Format

- The project directory must be named XXXXXXXX (your ID).
- The project directory must contain only a subdirectory named PY, where Y is the homework number.
- The directory PY should contain only source files and a folder named TPY. No binary files should be present in the directory.
- The directory TPY should contain an overview file and a file called `seq_tokens.txt`. We may not evaluate an assignment if these files are not present.
- Tar the XXXXXXXX directory along with the contents and then gzip it to **XXXXXXXX.PY.tar.gz**. Submit the XXXXXXXX.PY.tar.gz file.

4 Test Case Submission Guidelines

`Public Test Cases`: You should provide exactly three test cases with programs that follow the basic construct of the program defined in Figure 1. You may add extra features to the program, but you cannot delete or modify any existing features. Your lexer will be analyzed on your public test cases first, and then on private test cases.

- The test cases directory must be named XXXXXXXX (your ID).
- The test case directory must contain only a subdirectory named TY, where Y is the homework number.
- The directory TY should contain only the contributed test cases in different `.txt` files named as `1.txt` and so on.
- Tar the XXXXXXXX directory along with the contents and then gzip it to XXXXXXXX.TY.tar.gz. Submit the XXXXXXXX.TY.tar.gz file.

5 Grading

Solutions will be graded on implementation efforts, correctness, and proper documentation. A **correct** program compiles without errors or warnings and behaves according to the requirements given. Your lexical analyzer will be tested against multiple private test cases (along with the public test cases) to verify its correctness; failure to do this may result in a lower grade on your assignment. Documentation includes the overview document and comments in the code.

The total 100 marks are distributed for each section in the following manner:

- `Contribution of public test cases:` 5 marks
- `Correctness on test cases:` 70 marks (5 each: 2.5 for token gen and 2.5 for C-program gen)
- `Proper Documentation:` 25 marks

6 Late Submission Policy

- We expect that all the students will meet the given deadline.
- A delayed submission will automatically incur a 20% penalty for each day of delay.
- A student may make as many submissions (before the given deadline or delayed submission) as he/she wants. But we will use only the last submitted one for evaluation.

7 Plagiarism and GPT Policy

- Plagiarism in any form will not be tolerated.
- We will not differentiate between the source (giver) and the sink (taker). We will consider both parties to have plagiarised.
- Any student suspected of plagiarism for an assignment will get a zero on that assignment (may lead to FR grade also) and will be reported to the institute disciplinary committee.
- Copying directly from the GPT tool will not be tolerated. You must not rephrase your answers using the GPT tool also.