Indian Institute of Technology Tirupati

CS3191 Compiler Design Laboratory

Objectives

- To understand the developmental aspects of compiler.
- To learn the tools, namely Lex and Yacc used in the development of compilers.

Lab Outcomes

- Ability to write regular expressions for different language constructs.
- Ability to develop simple tools for scanning and parsing.
- Ability to develop a miniature compiler.

General Guidelines

Any kind of copying, sharing code with others, and malpractices attract high penalties to the extent of referring to the **Institute Level Disciplinary committee**.

Submission Guidelines

All Exercises should be submitted in the following format.

- All files and folders should be lowercase letters
- Create a folder with name yourrollnumber_week_X (say for week 1 and roll no CS17B001, the directory name should be cs17b001_week_1) and create subfolders, namely 1, 2,.., n for each problem given for the first lab, in this single folder
- Prepare a separate lex and make files for each of the exercise problems
- Makefile should generate final executable file named **scanner**
- The input must be given through a file and the file name should be taken through command line arguments
- Copy the lex and make files into their respective sub folders
- Create a Readme file in the main folder
- Dont keep any unrealed or executable files
- Finally tar and compress the yourrollnumber_week_X directory as yourrollnumber_week_X.tar.gz and upload the same to the course page at Moodle before the due date.

Evaluation Guidelines

- Output, Logic (wherever applicable), Naming convention, code readability, comments etc., Adherence to the instructions

Lab #2 Due Date: Aug 21, 2019, 6pm

Objective: Development of a simple hand made Lexical analyzer (DFA Simulator)

Recall our discussion during the lab lecture in last week. Lexical analyzer is a realization of a deterministic finite automaton (DFA) constructed to identify pattern(s). Hence in this Lab, you need to implement a simple but generic lexical analyzer, named as **slex** in one of your preferred languages from C, C++, or Java. Essentially, **slex** should implement the logic to read the state transition table of a DFA and tokenize the given input string. The steps for implementing **slex** are the following.

1. Read the number of states, the number of final states, and the number of input symbols - as command line arguments.
2. Then read the state transition table of a given DFA in **table.txt** file.
3. For each string given in input file **strings.txt**, check if string is acceptable by the DFA. Print **yes** if acceptable, otherwise print **no**.

For example, the following is the state transition table of a DFA of some language.

Table 1: State transition table of a DFA

	a	**b**	**c**
0	1	3	4
1	0	1	3
2	1	3	2
3	2	4	1
4	0	1	4

For example, the command **./slex 5 2 3** with **5 2 3** as command line arguments, indicates a DFA with 5 states of which 2 are final states, and the alphabet has 3 symbols. The other specifications of the DFA should be given in *table.txt* file in the following order.

- First line should specify the states of the DFA where each state should be separated by space.
- Second line should specify the final states of the DFA where each state should be separated by space.
- Third line should specify the symbols of the input alphabet.
  Assume that the input symbols are all valid ASCII characters except $ and ^ which represent epsilon and null set, respectively.
- Next, the transition table of DFA should be given in a tabular form.

For example, the DFA shown in Table 1 should be specified in table.txt as shown below.

```
0 1 2 3 4          // num of states 5, each state is separated by space
2 4                // 2 and 4 are the two final states
a b c              // symbols in alphabet
1 3 4
0 1 3
1 3 2
```

2 4 1

0 1 4

Submission instructions: The name of the executable is **slex** and other instructions are same as those of previous exercises.