CS320
Lab 2
Lex and Yacc

Srinibas Swain (srinibas@iiitg.ac.in)

We hope you enjoy the lab and, more generally, the unit!

We will illustrate the use of these programs with a language CHAIN based on certain expressions involving strings.

CHAIN

The language **CHAIN** consists of expressions of the following type. An expression consists of a number of terms, with # between each pair of consecutive terms, where each term is either a string of lower-case letters or an application of the Reverse function to such a string. Examples of such expressions include

```
mala # y # Reverse(mala)
block # drive # cut # pull # hook # sweep # Reverse(sweep)
Reverse(side) # Reverse(direction) # Reverse(gear)
```

For lexical analysis, we wish to treat every lower-case alphabetical string as a lexeme for the token STRING, and the word `Reverse` as a lexeme for the token `REVERSE`. We focus mainly on the Rules section, in the middle of the file. It consists of a series of statements of the form

*p*attern { action }

where the pattern is a regular expression and the action consists of instructions, written in C, specifying what to do with text that matches the pattern. In your file, each pattern represents a set of possible lexemes which you wish to identify. These are:

- a string of lower-case letters;
  - This is taken to be an instance of the token STRING (i.e., a lexeme for that token).
- the specific string `Reverse`;
  - Such a string is taken to be an instance of the token REVERSE.
- certain specific characters: #, (,);
- white space, being any sequence of spaces and tabs;
- the newline character.

Note that all matching is case-sensitive.

Our action is, in most cases, to print a message saying what token and lexeme have been found. For white space, we take no action at all. A character that cannot be matched by any pattern yields an error message.

If you run lex on the file chain.l, then lex generates the C program lex.yy.c. This is the source code for the lexical analyser. You compile it using a C compiler such as cc.

```
$ flex chain.l
$ cc lex.yy.c
```

When you run the program, it will initially wait for you to input a line of text to analyse. Do so, pressing Return at the end of the line. Then the lexical analyser will print, to standard output, messages showing how it has analysed your input. The printing of these messages is done by the `printf` statements from the file `chain.l`. Note how it skips over white space, and only reports on the lexemes and tokens.

```
$ ./a.out
mala
# y #Reverse( mala)
Token: STRING; Lexeme: mala
Token and Lexeme: #
Token: STRING; Lexeme: y
Token and Lexeme: #
Token: REVERSE; Lexeme: Reverse
Token and Lexeme: (
Token: STRING; Lexeme: mala
Token and Lexeme: )
Token and Lexeme: <newline>
```