Homework #1: CMPT-379

Anoop Sarkar – `anoop@cs.sfu.ca`

- Only submit answers for questions marked with †.
- Checkout your group svn repository:
  `svn co https://punch.cs.sfu.ca/svn/CMPT379-1137-g-YOUR-GROUP-NAME`
- Copy the files for this homework (and then add and commit the files using svn):
  `scp -r fraser.sfu.ca:/home/anoop/cmpt379/hw1 CMPT379-1137-g-YOUR-GROUP-NAME/.`
- Put your solution programs in the `hw1/answer` directory. Use the `makefile` provided. There are strict filename requirements. Read the file `readme.txt` in the `hw1` directory for details.
- Create a file called `HANDLE` in your `hw1` directory which contains your group handle (no spaces).
- The `hw1/testcases` directory contains useful test cases; you will need to consult `readme.txt` for the mapping between the homework questions and test cases and instructions on how to run the auto check program.
- Always use `exit(0);` as the last line of your `main` function. Use `exit(1);` to indicate failure.
- Reading for this homework includes Chp 3 of the Dragon book. If needed, refer to:

  `http://tldp.org/HOWTO/Lex-YACC-HOWTO.html`

(1) † Provide a lex program to replace all single-line and multi-line comments from C or C++ programs with an equivalent amount of whitespace. Single-line comments begin with `//` and continue to the end of the line, and multi-line comments begin with `/*` and end with `*/` and have no intervening `*/`. Note that `/*/` is not a valid comment.

(2) † `simpletok.lex` is a lex program for a simple lexical analyzer. Modify the pattern definition of the token IDENTIFIER so that it has to start with a letter (a-z or A-Z) and followed by a (possibly empty) sequence of letters or numbers.

(3) † The general architecture of a lexical analyzer for a programming language is to specify tokens in terms of patterns. For instance, we can define a set of tokens (`T_A, T_B, T_C`) associated with pattern specified as regular expressions.

`T_A`	a
`T_B`	abb
`T_C`	a^*b^+

The lexical analysis engine should always pick the longest match possible, and in case of two patterns matching a prefix of the input of equal length, we break the tie by picking the pattern that was listed first in the specification (e.g. the token `T_B` is preferred over `T_C` for the input string *abb*, and for the same input string, token `T_A` followed by `T_C` would be incorrect).
Provide a lexical analyzer using lex for the tokens shown above.

(4) † The **`Decaf`** language specification is available in the Homeworks tab of the course web page. Using the **`Decaf`** language definition as your guide, provide a lex program that is a lexical analyzer for the **`Decaf`** language.

a. Note that the token names and lexeme values should be identical to the sample output provided to you. You can also refer to the full list of token names in the file `decaf-token-names.txt`.

b. You must include a special whitespace and comment token. The whitespace token should have a lexeme value that includes all the whitespace characters. The whitespace and comment lexemes should convert the newline character into the literal string `\n` so that the line number and character number of each token can be recovered from the lexical analyzer output.

c. Provide appropriate error reporting with the line number and location in the line where the error was detected.

(5) Lex can match a *right context* while matching a regexp by using the notation r_1/r_2 where the regexp r_1 will match only if the right context matches the regexp r_2. Lex can also match the *left context* with the use of *states*. `leftcontext-inp.lex` is a lex program that matches keyword called `inputfile` as the left context so that a string following this keyword is treated differently from one that does not. Modify the lex program to include a new left context for the keyword `outputfile` so that for the input:

```
inputfile "fileA"
outputfile "fileB"
```

Your program produces the output:

```
"fileA" is the input file.
"fileB" is the output file.
```

(6) Lex can be forced to perform backtracking regexp matching using the REJECT command as shown in the `reject.lex` program. On input `aaa` the lex program `reject.lex` will produce the output:

```
pattern a+: 6 -- pattern a*b?: 6
```

This is because there are $\frac{n(n+1)}{2}$ substrings for a string of length n. Predict the number of pattern matches for the following input, and check by running the lex program.

```
aaa
aa
ab
```

(7) Provide a lex program that reports the frequency of each pair of words in a text file. Each word is a sequence of non-whitespace (space and tab) characters separated by one or more whitespace characters. The lex program `simplebigram.lex` contains a hint.