1 Lexical Analysis

We break up compilers into multiple phases, each with the primary purpose of making the next easier to implement. The lexical analyzer is the first phase of a compiler. Its main purpose is to read input strings, and convert them into tokens to be sent to the parser for syntax analysis. You will be implementing a lexer in project 1.

1. Convert the following C++ text into its lexed representation:

```cpp
float limitedSquare(x) float x; {
    /* returns x-squared, but never more than 100 */
    return (x<=-10||x>=10)?100:x*x;
}
```

**Answer:**

```
KEYWORD("float") ID("limitedSquare") LPAREN ID("x") RPAREN KEYWORD("float") ID("x") SEMICOLON LCURLY KEYWORD("return") RPAREN ID("x") OP("<=") NUMBER("-10") OP("||") ID("x") OP(">=") NUMBER("10") LPAREN QUESTION NUMBER("100") COLON ID("x") OP("*" ) ID("x") SEMICOLON RCURLY
```

2 Regular Expression

A regular language is a kind of language that has some additional structure. Specifically, a regular language can be described by a regular expression.

1. What language is denoted by each of the following regular expressions? Try writing down a few simple strings and give a concise description of the language.

   (a) `[_a-zA-Z][_a-zA-Z0-9]*`
   **Answer:** Identifiers (in many languages)

   (b) `(\epsilon|a)b*` (Could you simplify this expression?)
   **Answer:** All possible sequences with characters a and b

   (c) `(0+1)+|1(0+)`
   **Answer:** Binary strings with different first and last bits.

2. Write a regular expression for the following languages:

   (a) Binary numbers.
   **Answer:** `0|1(0|1)*`
(b) Even binary numbers. (read from left to right – i.e., “1101” is 13).
   Answer: “[01]*0” or “1[01]*0|0”, depending on whether we allow leading zeros.

(c) All strings of lowercase letters in which the letters are in ascending lexicographic order.
   Answer: a*b*c*....y*z*

(d) Ada identifiers, which are strings of letters, digits, and underscores starting with a letter,
such that underscores do NOT appear next to other underscores or at the end of the identifier.
   Answer: [a-zA-Z][a-zA-Z0-9]*

3 Regular Expression with Backreference

Backreferences match the same text as previously matched by a capturing group. In the basic GNU
regular expression, a subexpression enclosed in parenthesis is a capturing group. For example, in
regular expression \( \w \)\1, subexpression \w is a capturing group, and the matched text is referred
as the first reference \1. Therefore, \( \w \)\1 could match any pair of repeated letters: “aa”, “bb”,
etc. As you might guess, \2 refers to the second matched group and so forth.

1. Write a regular expression for the following languages:
   (a) Suppose now you have a large CSV(Comma Separated Values) file, each row of which has
   the form firstname,lastname,email. You want to find out all uninteresting people who
   use boring email address of the form firstname+lastname@xxx.com and avoid making
   friends with them. Try to write a normal (without backreference) regular expression that
   describes this pattern, or state it’s impossible.
   Answer: Not possible

   (b) Now try to write expression for the mentioned pattern using backreference.
   Answer: (\w+), (\w+), \1+2@