1 Regular Expressions

Describe the languages denoted by the following regular expressions:

1. $a(a|b)^*a$
2. $((\epsilon|a)b^*)^*$
3. $(a|b)^*a(a|b)(a|b)$
4. $a*b*a^*b*a^*$
5. $(aa|bb)^*((ab|ba)(aa|bb)^*(ab|ba)(aa|bb)^*)^*$

2 Lexical Analysis

Here's a function definition in C++:

```cpp
int foo(char a, long *b2) {
    vector<long> my_vector(a + 3, *b2 >> 2);
    return bar(my_vector);
}
```

A lexer would typically categorise this into integers (3 and 2), operators (+, *, and >>), identifiers (int, foo, char, a, long, b2, vector, my_vector, and bar), miscellaneous delimiters (, ), { }, <, >, ;, and ;), and keywords (return).

1. Write regular expressions that can match C++'s:
   (a) integers
   (b) identifiers
   (c) operators (only those above)
   (d) keywords (only those above)
   (e) delimiters (only those above)

2. Draw an NFA for a lexical analyzer that can recognize all the token categories present in the excerpt. Mark each accepting state with the label of the corresponding token, and its priority. Your analyzer must not recognize more syntactic elements of the language (e.g. other operators) than those necessary for scanning the code above.

3. Describe in detail (i.e. including all transitions from the initial state until a token or an error is emitted) how the following strings will be split into tokens by the lexical analyzer NFA you defined in part 2.
(a) return me;
(b) returnme;

4. Convert the NFA you drew in part 2 to a DFA.

5. Describe in detail (i.e. including all transitions from the initial state until a token or an error is emitted) how the following strings will be split into tokens by the lexical analyzer DFA you defined in part 4.
   (a) vector<vector<long> > a;
   (b) vector<vector<long>> a;