1. (5 points) Write a regular expression for the language \(L = \{0^n1^m \mid (n + m) \text{ is even}\}\).

2. (20 points) Let the alphabet \(\Sigma = \{0, 1\}\).
   
   (a) (5 points) Write a regular expression for the language of all strings over \(\Sigma\) that contain the contiguous substring 11.

   (b) (5 points) Write a regular expression for the language of all strings over \(\Sigma\) that do not contain the contiguous substring 11.
(c) (5 points) Give a non-deterministic finite automaton (NFA) for the language of all strings over $\Sigma$ that contain the contiguous substring 11.

(d) (5 points) Give a non-deterministic finite automaton (NFA) for the language of all strings over $\Sigma$ that don’t contain the contiguous substring 11.
3. (10 points)

(a) (5 points) Give a non-deterministic finite automaton (NFA) for the language \( L = (010 \mid 01)^* \). The NFA must contain at most 3 states. (Hint: draw an NFA with 4 states, then optimize).

(b) (5 points) Give a deterministic finite automaton for the language \( L \).
4. (30 points)

Consider the following grammar:

\[
S \rightarrow L = R \\
L \rightarrow *R \mid id \\
R \rightarrow L
\]

You can think of L and R as standing for l-value and r-value, respectively. * is the dereference operator or indirection operator in C-like languages.

A shift-reduce parser can perform the following sequence of actions to accept the string “*id = id”.

\[
\text{shift} \rightarrow \text{shift} \rightarrow \text{reduce} \rightarrow \text{reduce} \rightarrow \text{reduce} \rightarrow \text{shift} \rightarrow \text{shift} \rightarrow \text{reduce} \rightarrow \text{reduce} \rightarrow \text{reduce} \rightarrow \text{accept}
\]

(a) (10 points) Give a sequence of actions that a shift-reduce parser can take to accept the string “id = id”.

(b) (10 points) Give a sequence of actions that a shift-reduce parser can take to accept the string “*id = *id”.

(c) (10 points) Is the grammar ambiguous? Why or why not?
5. (15 points)

Consider the following grammar:

\[
S \rightarrow \text{if } E \text{ then } S \text{ else } S | \text{begin } S \text{ L } \text{print } E | \varepsilon \\
L \rightarrow \text{end} | ; S L \\
E \rightarrow \text{num} = \text{num}
\]

The goal is to write a recursive-descent parser for the grammar. You are given the following \text{L()} and \text{E()} functions. Your job is to write the \text{S()} function on the next page.

\[
\text{L()} \\
\quad \text{if (token = end) } \
\quad \quad \text{match(end);} \\
\quad \text{else if (token = ;) } \
\quad \quad \text{match(;) } \text{S(); L();} \\
\quad \text{else } \
\quad \quad \text{throw SyntaxError;} \\
\]

\[
\text{E()} \\
\quad \text{if (token = num) } \
\quad \quad \text{match(num); match(=); match(num);} \\
\quad \text{else } \
\quad \quad \text{throw SyntaxError;} \\
\]

S() { 

if (token = if) {
    match(if); E(); match(then); S();
} else if (token = begin) {
    match(begin); S(); L();
} else if (token = print) {
    Match(print); E();
} else {
    /* empty string */
}

}
6. (20 points)
The following is a code snippet of legal-01.dcf:

```java
class Program {
    int A[100];
    int length;

    void main() {
        int temp;

        length = 100;

        callout("srandom", 17);

        for i = 0, length {
            temp = callout("random");
            A[i] = temp;
        }

        /* <HERE> */
    }
}
```

What should the symbol tables look like at <HERE>, considering the semantics of the Decaf language? Complete the symbol tables on the next page in the similar way to the symbol tables presented at Lecture 5. (Hint: note that the Decaf language is different from the language presented at Lecture 5).