Instructions:

- Provide your answer in the box after each question. If you absolutely need extra space, use the backs of the pages, but try to avoid it. Keep your answers short and to the point.

- You are not allowed to use a cheat sheet.

- Read every question carefully before answering. In particular, do not waste time on a proof if none is asked for, and do not forget to provide one if it is required.

- Do not forget to write your banner number and name on the top of this page.

- This exam has 8 pages, including this title page. Notify me immediately if your copy has fewer than 8 pages.

- The total number of marks in this exam is 95.
1 Regular Languages and Finite Automata

Question 1.1 (Definition) 5 marks
A language is a set of strings over an alphabet. What are the conditions this set has to satisfy for the language to be regular? In other words, provide the set-theoretic definition of regular languages. Do not use their relationship to regular expressions and finite automata as a definition.

Question 1.2 (Finite automata) 15 marks
In class we discussed that a language is regular if and only if it can be recognized by a finite automaton.

(a) Provide the formal definition of a deterministic finite automaton (DFA).

(b) When does a DFA accept a string $\sigma$?

(c) What are the two differences between a non-deterministic finite automaton (NFA) and a DFA?
Question 1.3 (Pumping Lemma) 5 marks
State the Pumping Lemma.

Question 1.4 (Deciding whether a language is regular) 15 marks
For each of the following languages, state whether it is regular or not. If it is, prove your claim by providing a graphical representation of a (deterministic or non-deterministic) finite automaton that recognizes it. If it is not, prove your claim using the Pumping Lemma.

(a) The language $L$ of all binary strings with at least one 1 in each substring of length three. In other words, the string $10100111001$ belongs to this language, the string $101000$ does not.

(b) The language $L$ of all binary strings that are palindromes, that is, strings that are of the form $x_1x_2\ldots x_nx_{n-1}\ldots x_1$ or $x_1x_2\ldots x_{n-1}x_nx_{n-1}\ldots x_1$.

(c) The language $L$ of all binary strings whose second and second-last letters are 1s. For example, the strings $010011$ and $11$ belong to this language, the strings $1011$ and $0101101$ do not.
Question 2.1 (Grammars) 10 marks

(a) Define formally what a context-free grammar is.

(b) Define formally when a grammar is LL(k).
Question 2.2 (Recognizing LL(1) grammars)  
Is the following grammar LL(1)? (T is the start symbol.) Prove that your answer is correct.

\[
T \rightarrow A \ B \\
A \rightarrow P \ Q \\
A \rightarrow C \ B \\
P \rightarrow p \ P \\
P \rightarrow \varepsilon \\
Q \rightarrow q \ Q \\
Q \rightarrow \varepsilon \\
B \rightarrow b \ B \\
B \rightarrow e \\
C \rightarrow c \ C \\
C \rightarrow f
\]
Question 2.3 (Parsing LL(1) languages)  

Provide the pseudo-code of a recursive-descent parser that accepts all strings in the language defined by the grammar in Question 2.2 and rejects all strings not in this language.
Question 3.1 (Is it lexical, syntactic or semantic?) 10 marks

For each of the following conditions on a valid C program, state whether it is a lexical, syntactic or semantic constraint.

(a) An identifier is a non-empty sequence of letters, digits and underscores not starting with a letter and not equal to one of the keywords of the language.

(b) Statements need to be separated by semicolons.

(c) A variable has to be declared before it is used.

(d) Every ‘{’ has to be matched by a ‘}’ and vice versa.

(e) A string is a sequence of characters starting and ending with a double quote and not containing any unescaped double quotes.

(f) The dereferencing operator ‘*’ expects a pointer as its argument.

(g) The number of arguments provided to a function call matches the number of formal parameters in the function definition.
Question 3.2 (Attribute grammars) 10 marks

Provide a context-free grammar that defines the language of all valid arithmetic expressions using multiplication, division, addition, subtraction, and parenthesization. The only operands you need to recognize are numbers, and you can treat numbers as indivisible tokens that have already been recognized by a scanner. Your grammar does not necessarily have to be LL(1). (The second part of this question is substantially easier if it is not.)

Augment your grammar so that it becomes an attribute grammar that computes the value of every expression that conforms to the grammar. The calculation of these values should respect the standard precedence rules for addition, subtraction, multiplication, and division and should respect left-associativity of these operations.