Formal Languages and Compilers
(A.Y. 2015/2016)
2h30m

September 6^{th}, 2016

First name: **Last name:**

Matriculation n.: **e-mail:**

Lexical Analysis

Exercise 1 – 6pt

In the definition of a regular expression real languages generally includes the not operator (in addition to the traditional symbols defining regular expressions). The operator permits to identify the set of strings that do not match the regular expression pattern, given an alphabet Σ. So, given a regular expression r on Σ, the strings matching $\neg r$ on Σ are those not matching r. If useful the operator can be included in the definition of the regular expressions for the following languages:

- The language $\mathcal{L}$ including strings over the alphabet $\Sigma = \{a, b, \ldots, z, A, B, \ldots, Z\}$ which contains only admitted passwords. Rules to be followed entering an admitted password are:
  - it should contain at least one caps letter
  - it should include at least 8 characters

  (**Suggestion:** may be it can be useful to approach the problem considering at first the properties of regular languages in relation to intersection, complement, and union.)
- The language $\mathcal{L}$ on the alphabet $\Sigma = \{a, b, c, -, >\}$ representing all the possible triples of strings on $\{a, b, c\}$ such that each element in a triple ends with a different symbol.
  (e.g. $< a, abb, c >$, $< abb, cc, ba >$, $< ccb, ba, abc > \in \mathcal{L}$, $< a, b, b >$, $< ab, ac >$, $< aa, c, abc > \notin \mathcal{L}$)

Syntax Analysis

Exercise 2 – 13pts

Let's $\mathcal{G}$ the grammar defined by the following productions:

$$S \longrightarrow aSA \mid a \quad A \longrightarrow cASA \mid b \tag{1}$$

1. Discuss the applicability of parsers LR(0) and LR(1)
2. Show the steps of parser LR(1) in the acceptance or rejection of the string *aacbab*

Semantic Analysis

Exercise 3 – 14pts

Consider the following excerpt from a grammar for a complex programming language:

$$\begin{array}{rcl} S & \rightarrow & \textbf{threeway}\ (\textbf{id}) : \\ & & \quad val_1\ :\ S_1 \\ & & \quad val_2\ :\ S_2 \\ & & \quad val_3\ :\ S_3 \end{array}$$

The command permits to define a three way conditional statement. In particular in case the value of **id** is equal to val_1 then S_1 is executed, while if it is equal to val_2 the S_2 statement is executed, and finally in case it is equal to val_2 then S_3 is executed. In case no match is found the next line to be executed is the one following the command. No check related to the type of the **id** and of the value has to be perfomed.

- Provide an L-attributed SDD for the command that permits to translate it in a three-address code program that behaves as expected[1]
- Show the parse tree and derive the three address code program for the code snippet below. In doing this refer to the translation schemes for expressions and commands which have been introduced during the course. It is not necessary to check the type of the expressions, and it can be assumed that variables have been declared somewhere before reaching the statement.

```
. . .
threeway tre:
  1: i = i + 1
  2: i = i + 2
  3: i = i + 3
. . .
```

[1]For your convenience I recall that the function *top.get(id.lexeme)* permits to retrieve the address of the specified id, while the function *gen(. . .)* is used to generate three-address code in the right format for the different istructions, and finally *new Temp()* and *new Label()* permit to generate a new temporary address and a new label, respectively.