# Progetto di Linguaggi e Compilatori – Parte 1
# A.A. 2019-20

# Studente Filippo Callegari

**Attenzione: una volta letto il presente testo è *obbligatorio* consegnare alla scadenza quanto elaborato, indipendentemente dal fatto che lo si consideri adeguato o meno.**

Entro il termine stabilito si *deve* inviare via email una relazione in formato PDF dal nome

"`ProgettoLC parte1 Filippo Callegari Relazione`".

La relazione può contenere immagini passate a scanner di grafici o figure fatte a mano.

Si richiede:

**Per tutti gli esercizi** una descrizione dettagliata di tutte le tecniche **non**-standard impiegate (le tecniche standard imparate a lezione non vanno descritte).

**Per tutti gli esercizi** una descrizione delle assunzioni fatte riguardo alla specifica, sia relativamente a scelte non previste espressamente dalla specifica stessa, che a scelte in contrasto a quanto previsto (con relative motivazioni).

Non è assolutamente utile perdere tempo per includere nella relazione il testo dei vari esercizi o un suo riassunto o una qualsiasi rielaborazione, incluso descrizioni del problema da risolvere. Ci si deve concentrare solo sulla descrizione della soluzione e delle eventuali variazioni rispetto a quanto richiesto.

## Esercizio 1

Data la grammatica

$F \rightarrow$ `atom` | $F$ `and` $F$ | $F$ `or` $F$ | `(` $F$ `)` | `true` | `next` $F$ | `always` $F$

1. Definire un'equivalente grammatica LL(1).
2. Costruirne il parser top-down, capace di gestire, con opportune azioni, l'error recovery (sia riempendo opportunamente tutta la tabella, che prevedendo cosa fare in caso di mancato match con il lookahead).
3. Mostrare l'esecuzione di suddetto parser sull'input

   ```
   next atom next atom and or and next next atom
   ```

   (utilizzando i passi di error recovery definiti).

Per rendere più rapida la correzione, nelle tabelle si mettano le righe dei non terminali e le colonne dei token in ordine alfabetico.

## Esercizio 2

Data la grammatica

$E \rightarrow L$ `=` $R$ | $R$

$L \rightarrow$ `*` $R$ | `id` | $L$ `[` $E$ `]`

$R \rightarrow$ `num` | $L$ | $R$ `+` $R$ | $L$ `pp` | `pp` $L$

1. Costruire i parsers SLR e LALR (di questa grammatica!). Qualora ci fossero conflitti nelle rispettive tabelle, si riportino tutti i casi. Inserire inoltre le opportune azioni di error recovery (riempendo tutta la tabella).
2. Mostrare l'esecuzione di suddetti parser sull'input

   ```
   id[id+ id+ ] = num id pp + num
   ```

   (utilizzando i passi di error recovery definiti). Qualora nelle rispettive tabelle ci fossero conflitti shift/reduce, si utilizzi lo shift; mentre in caso di reduce/reduce si utilizzi il reduce con regola testualmente precedente.

Per rendere più rapida la correzione, nelle tabelle si mettano le colonne dei token in ordine alfabetico (e le righe degli stati in ordine numerico).