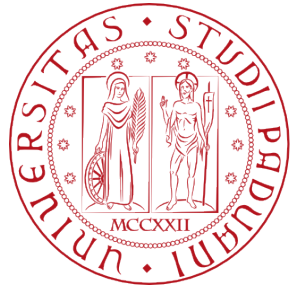
Automi e Linguaggi Formali

a.a. 2018/2019

LT in Informatica
29 Aprile 2019



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Laboratorio

- Il laboratorio può essere svolto in gruppi di massimo tre persone
- Permette di ottenere un bonus di massimo 2 punti
- Testo e consegna degli esercizi sul moodle del corso
- Uno solo dei componenti consegna gli esercizi, indicando i nomi dei componenti del gruppo
- Gli esercizi vanno consegnati entro le ore 23:55 di venerdì 7 Giugno 2019

Da TinyREXX a C++


```
pull n
do while n > 0
    say n
    n = n - 1
end
say n
```

Da TinyREXX a C++

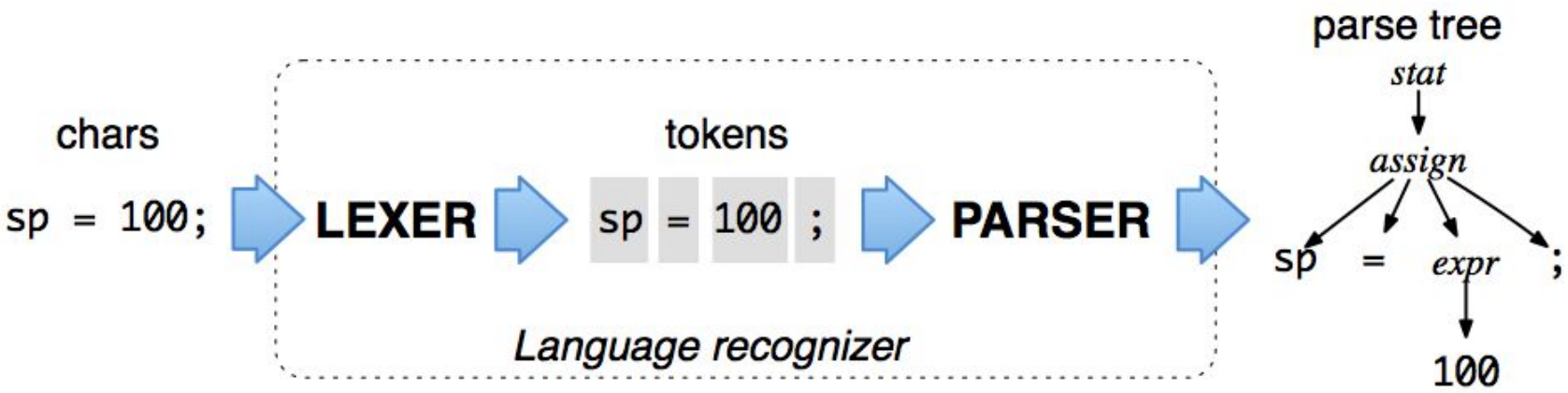

```
pull n
do while n > 0
    say n
    n = n - 1
end
say n
```


```
#include <iostream>

using namespace std;

int main() {
    int n = 0;
    cin >> n;
    while(n > 0) {
        cout << n << endl;
        n = n - 1;
    }
    cout << n << endl;
}
```

Analisi sintattica



- Il lexer prepara i token da passare al parser
- Il parser accetta uno stream dei token preparati dal lexer e cerca di riconoscere la struttura del testo
- La struttura del testo viene rappresentata con un albero sintattico

ANTLR v4

- Strumento automatico che permette di generare analizzatori sintattici di testi (parser)
- Sviluppato in Java permette di generare codice in C++, Java, C#, Python, Swift, Go, ...
- Supporta espressioni regolari, grammatiche EBNF e la generazione di alberi sintattici

Questo strumento può produrre in automatico il codice relativo a:

- Lexer: parte lessicale dove si definiscono i token e i separatori da ignorare
- Parser: parte grammaticale contenente le regole di produzione
- Listener: parte semantica che definisce le azioni da eseguire sul testo

Lexer

- Legge la stringa di caratteri del testo e raggruppa i caratteri in sequenze chiamate token
- Le regole usano espressioni regolari per definire i token
- ANTLR costruisce un DFA per riconoscere i token

Sintassi delle regole del Lexer

- Il nome della regola deve iniziare con una lettera maiuscola
- `'letterale'` carattere o sequenza di caratteri come `'while'` o `'='`
- `[char set]` uno tra i caratteri specificati. `x-y` identifica l'insieme di caratteri compresi tra `x` e `y`. Esempi: `[abc]` o `[a-zA-Z]`
- `.` un carattere qualsiasi
- operatori di composizione: `|` (unione), `*` (chiusura di Kleene), `+` (una o più ripetizioni)
- parentesi per raggruppare le operazioni

Parser

- Il parser crea una rappresentazione ad albero della lista di tokens (struttura grammaticale)
- Albero: insieme di nodi e archi diretti (da padre a figlio), tale che un nodo (radice) non ha padri, e tutti gli altri hanno esattamente un padre. Foglie: nodi senza figli.
- Albero sintattico: nodo interno = regola, figli = simboli terminali
- Mostra l'ordine in cui eseguire le istruzioni del programma
- Il parser usa: grammatiche libere da contesto per definire le regole
- ANTLR usa un algoritmo chiamato LL(*) per costruire l'albero sintattico

Regole del Parser

- Il nome della regola deve iniziare con una lettera minuscola
- `'letterale'` carattere o sequenza di caratteri
- `nonterminale` simbolo non terminale della grammatica
- `TOKEN` token riconosciuto dal Lexer
- Le alternative sono separate da `|` :

```
superClass
  : 'extends' ID
  | // parola vuota
  ;
```

- la regola si può scrivere su più righe
- `//` inizia un commento
- si possono avere alternative vuote
- `;` termina la regola

- le parentesi raggruppano sottoregole alternative:
  - `(x | y | x)` fa il match con una sola tra `x, y, z`