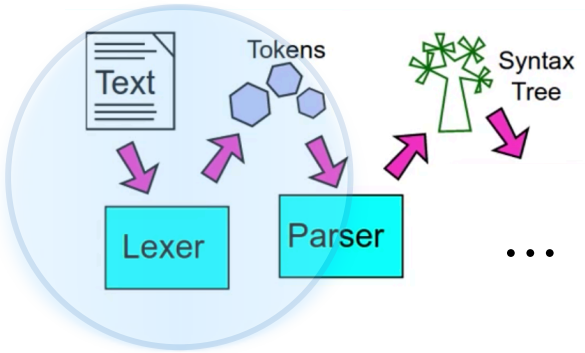
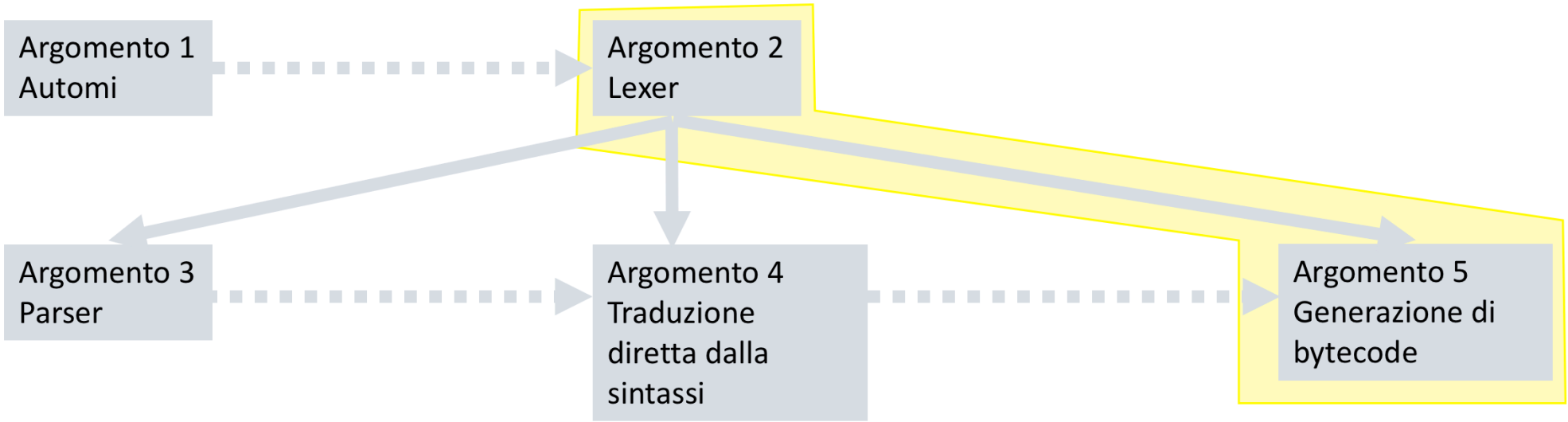


2. Analisi lessicale

Implementazione in Java di un lexer per un semplice linguaggio di programmazione

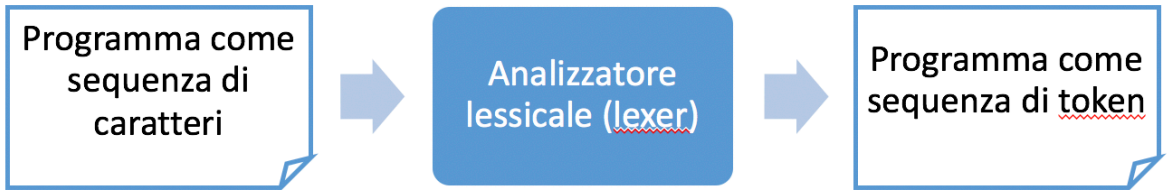
Progetto di laboratorio LFT LAB

- Il progetto di laboratorio consiste in una serie di esercitazioni assistite mirate allo sviluppo di un semplice traduttore.
- Dove siamo:
  - Primo step: analisi lessicale



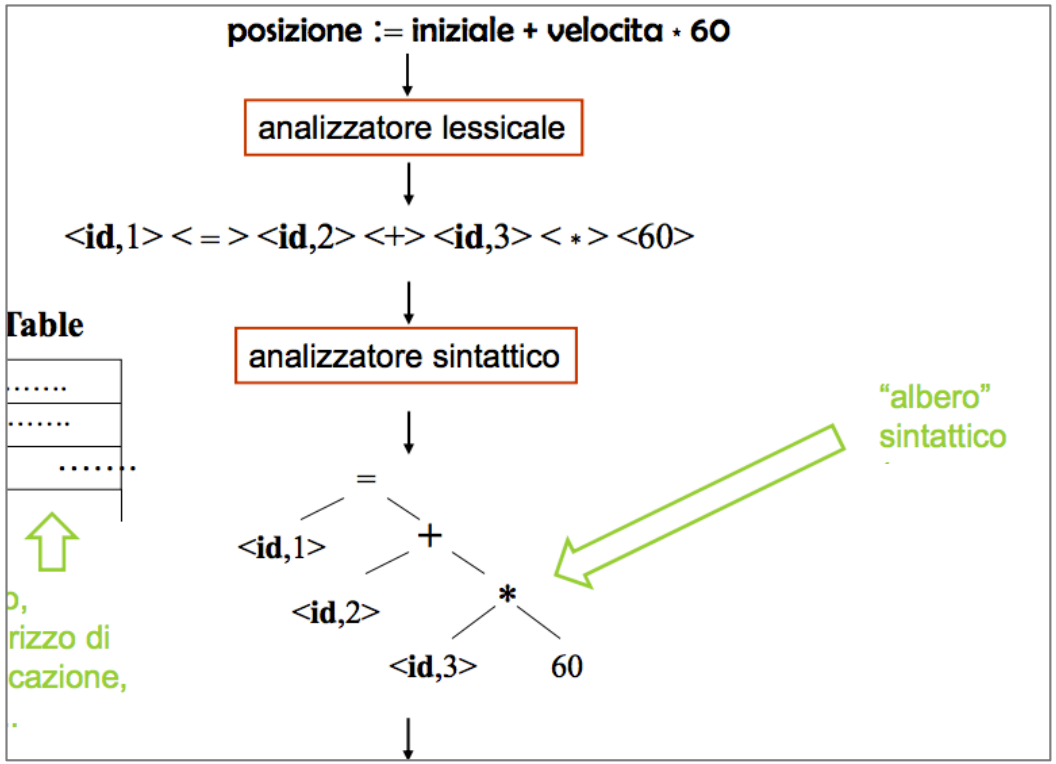
Analizzatore lessicale

- Cosa fa:
  - **Input:**
    - Legge un testo scritto in un certo linguaggio, nel nostro caso una **stringa di caratteri di un programma sorgente** scritto in un qualche linguaggio di programmazione dato
  - …e raggruppa i caratteri in sequenze chiamate **lessemi**
    - **Lessemi. Intuizione:** sequenze complete e dotate di senso per il linguaggio in oggetto; in altri termini: sequenze di caratteri ("parole") del programma che rispettano il possibile pattern di un token per quel linguaggio di programmazione.
  - **Output:** restituisce una corrispondente **sequenza di token**, dove un token corrisponde ad un'unità lessicale per quel linguaggio (elemento atomico), nel caso di un linguaggio di programmazione ad esempio:
    - un numero
    - un identificatore
    - un operatore relazionale
    - una parola chiave (while, read, print…)
    - …



Analizzatore lessicale

- Dove si posiziona in una *pipeline di traduzione*?
- Nelle sezioni successive, l'analizzatore lessicale implementato fornirà l'input a programmi di analisi sintattica e di traduzione (step successivi all'analisi sintattica).



(...continua...)

Analisi lessicale (teoria)

- *Token*: coppia <nome token, valore attributo>
  nome token: simbolo astratto che rappresenta un'unita` lessicale (una parola chiave, un identicatore, ecc.)
- *Pattern*: descrizione della forma che i lessemi di un'unita` lessicale possono avere.
  Esempio: se il token e` una parola chiave, il pattern e` la sequenza di caratteri che formano la parola chiave. Per gli identicatori, il pattern descrive stringhe di caratteri, che sono tutte identificatori.
- *Lessema*: sequenza di caratteri del programma sorgente che rispetta il pattern di un token

Nome token READ; pattern: carattere r, seguito da e, seguito da a e da d; lessema: read

Nome token ID; pattern: una lettera seguita da lettere e cifre; esempi di lessemi: **posizione, i**

Nome token: NUM; pattern: un numero;
Esempi di lessemi: **0, 45,879**

Il nostro linguaggio: i token

- I token del linguaggio sono descritti nel modo illustrato in **Tabella 1**.

Categorie di token

Possibili pattern dei token

Nomi dei token, espressi come costanti numeriche

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Assegnamento	`assign`	259
To	`to`	260
Conditional	`conditional`	261
Option	`option`	262
Do	`do`	263
Else	`else`	264
While	`while`	265
Begin	`begin`	266
End	`end`	267
Print	`print`	268
Read	`read`	269
Disgiunzione	`||`	270
Congiunzione	`&&`	271
Negazione	!	33
Parentesi tonda sinistra	(	40
Parentesi tonda destra	)	41
Parentesi quadra sinistra	[	91
Parentesi quadra destra	]	93
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Punto e virgola	;	59
Virgola	,	44
EOF	Fine dell'input	-1

Tabella 1: Descrizione dei token del linguaggio

Esempi

- L'analizzatore lessicale deve ignorare tutti i caratteri riconosciuti come "spazi" (incluse le tabulazioni e i ritorni a capo), ma deve segnalare la presenza di caratteri illeciti, quali ad esempio # o @

L'output dell'analizzatore lessicale dovrà avere la forma $\langle token_0 \rangle \langle token_1 \rangle \cdots \langle token_n \rangle$. Ad esempio:

- per l'input `assign 300 to d;` l'output sarà $\langle 259, \text{assign} \rangle$ $\langle 256, 300 \rangle$ $\langle 260, \text{to} \rangle$ $\langle 257, \text{d} \rangle$ $\langle 59 \rangle$ $\langle -1 \rangle$;
- per l'input `print(*{d t})` l'output sarà $\langle 268, \text{print} \rangle$ $\langle 40 \rangle$ $\langle 42 \rangle$ $\langle 123 \rangle$ $\langle 257, \text{d} \rangle$ $\langle 257, \text{t} \rangle$ $\langle 125 \rangle$ $\langle 41 \rangle$ $\langle -1 \rangle$;
- per l'input `conditional option (> x y) assign 0 to x else print(y)` l'output sarà $\langle 261, \text{conditional} \rangle$ $\langle 262, \text{option} \rangle$ $\langle 40 \rangle$ $\langle 258, > \rangle$ $\langle 257, \text{x} \rangle$ $\langle 257, \text{y} \rangle$ $\langle 41 \rangle$ $\langle 259, \text{assign} \rangle$ $\langle 256, 0 \rangle$ $\langle 260, \text{to} \rangle$ $\langle 257, \text{x} \rangle$ $\langle 264, \text{else} \rangle$ $\langle 268, \text{print} \rangle$ $\langle 40 \rangle$ $\langle 257, \text{y} \rangle$ $\langle 41 \rangle$ $\langle -1 \rangle$;
- per l'input `while (dog<=printread) assign dog+1 to dog` l'output sarà $\langle 265, \text{while} \rangle$ $\langle 40 \rangle$ $\langle 257, \text{dog} \rangle$ $\langle 258, <= \rangle$ $\langle 257, \text{printread} \rangle$ $\langle 41 \rangle$ $\langle 259, \text{assign} \rangle$ $\langle 257, \text{dog} \rangle$ $\langle 43 \rangle$ $\langle 256, 1 \rangle$ $\langle 260, \text{to} \rangle$ $\langle 257, \text{dog} \rangle$ $\langle -1 \rangle$.

Token con e senza attributi

- In generale, i token della Tabella 1 hanno un attributo: ad esempio, l'attributo del token <256, 300 > è il numero 300, mentre l'attributo del token <259, assign> è la stringa "assign".
- Alcuni token della Tabella 1 sono senza attributo:
  Esempi:
  - il segno di moltiplicazione * è rappresentato dal token <42>
  - la parentesi tonda destra) è rappresentata dal token <41> .

Pattern di identificatori e numeri

- Gli identificatori corrispondono all'espressione regolare (tag 257)

$$[a - zA - Z][a - zA - Z0 - 9]*$$

- i numeri corrispondono all'espressione regolare (tag 256)

$$0 \mid [1 - 9][0 - 9]*$$

Analisi lessicali e errori

- Attenzione: l'analizzatore lessicale non è preposto al riconoscimento della struttura sintattica dei comandi del linguaggio. Pertanto, restituirà senza batter ciglio sequenze di token anche per frammenti di programma che vi suonano "errati", ad esempio:

```
5+;)
```

```
(34+26( - (2+15-( 27
```

```
else 5 == print < end
```

- Altri errori invece, come simboli non previsti o sequenze che non corrispondono a nessun pattern per nessun token previsto devono essere rilevati, ad esempio:
  - nel caso dell'input `17&5` ?
  - oppure dell'input `|||` ?

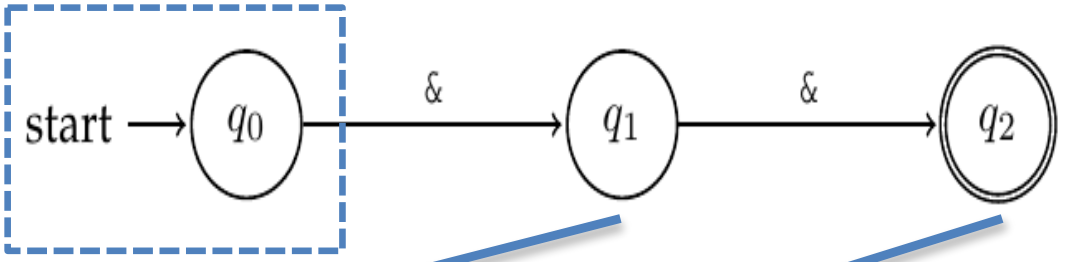
Esercizio 2.1

- Si scriva in Java un analizzatore lessicale che
  - legga da file un input e
  - stampi la sequenza di token corrispondente.
- Per questo esercizio, si possono utilizzare senza modifica le classi Tag, Token e Word (download da Moodle).
- Invece devono essere completate
  - la classe NumberTok
  - la classe Lexer

  a partire dalle implementazioni di partenza scaricabili su Moodle

NumberTok

- Idea: ispirandosi alla classe Word estendere Listing 5 per definire una classe NumberTok e rappresentare i token che corrispondono ai numeri
  - Caso simile a Word (token con attributo) ma …
    - Cos'e' il lessema nel caso dei numeri?
    - Ha senso trattarlo come una stringa di caratteri?




```
case '&':
    readch(br);
    if (peek == '&') {
        peek = ' ';
        return Word.and;
    } else {
        System.err.println("Erroneous character"
                + " after & : "  + peek );
        return null;
    }
```

Più complicati casi come > o < dove per capire cosa tokenizzare occorre andare a guardare il simbolo successivo

Input retraction:
Libro di testo 3.4

Identificatori e parole chiave

```
if (Character.isLetter(peek)) {

    // ... gestire il caso degli identificatori e delle parole
chiave //
```

Restituire uno degli oggetti specificati nella classe Word

Listing 4: Classe `Word`

```
public class Word extends Token {
    public String lexeme = "";
    public Word(int tag, String s) { super(tag); lexeme=s; }
    public String toString() { return "<" + tag + ", " + lexeme + ">"; }
    public static final Word
        assign = new Word(Tag.ASSIGN, "assign"),
        to = new Word(Tag.TO, "to"),
        conditional = new Word(Tag.COND, "conditional"),
        option = new Word(Tag.OPTION, "option"),
        dotok = new Word(Tag.DO, "do"),
        elsetok = new Word(Tag.ELSE, "else"),
        whiletok = new Word(Tag.WHILE, "while"),
        begin = new Word(Tag.BEGIN, "begin"),
        end = new Word(Tag.END, "end"),
        print = new Word(Tag.PRINT, "print"),
        read = new Word(Tag.READ, "read"),
        or = new Word(Tag.OR, "||"),
        and = new Word(Tag.AND, "&&"),
        lt = new Word(Tag.RELOP, "<"),
        gt = new Word(Tag.RELOP, ">"),
        eq = new Word(Tag.RELOP, "=="),
        le = new Word(Tag.RELOP, "<="),
        ne = new Word(Tag.RELOP, "<>"),
        ge = new Word(Tag.RELOP, ">=");
}
```

Identificatori e parole chiave

```
if (Character.isLetter(peek)) {

    // ... gestire il caso degli identificatori e delle parole
chiave //
```

Occorre istanziare e restituire
un nuovo oggetto Word che ha
id come tag e la stringa che corrisponde
al lessema come secondo parametro