Espressioni regolari in UNIX

Violetta Lonati [a]

Dipartimento Scienze dell'Informazione

Università degli Studi di Milano

Espressioni regolari

- Σ alfabeto (insieme finito)

Espressioni regolari

Σ alfabeto (insieme finito)

1. $\emptyset, \varepsilon, \sigma(\sigma \in \Sigma)$ *sono espressioni regolari*

Espressioni regolari

Σ alfabeto (insieme finito)

1. $\emptyset, \varepsilon, \sigma(\sigma \in \Sigma)$ *sono espressioni regolari*
2. Se p e q sono espressioni regolari, allora anche $(p + q), (p \cdot q), (p*)$ *sono espressioni regolari*.

Espressioni regolari

Σ alfabeto (insieme finito)

1. $\emptyset, \varepsilon, \sigma(\sigma \in \Sigma)$ *sono espressioni regolari*
2. Se p e q sono espressioni regolari, allora anche $(p + q), (p \cdot q), (p*)$ *sono espressioni regolari*.

NOTA: le parentesi possono essere omesse, con la convenzione che valga quest'ordine di precedenza: $*$ $\cdot$ $+$

Linguaggi denotati da ER

ER $p \quad \longrightarrow \quad L_p \subseteq \Sigma^*$

Linguaggi denotati da ER

ER $p \quad \longrightarrow \quad L_p \subseteq \Sigma^*$

1.

Linguaggi denotati da ER

ER $p \longrightarrow L_p \subseteq \Sigma^*$

1. $\emptyset$
   ε
   σ

Linguaggi denotati da ER

	ER p	$\longrightarrow$	$L_p \subseteq \Sigma^*$
	$\emptyset$		$L_\emptyset = \emptyset$
1.	ε		$L_\varepsilon = \{\varepsilon\}$
	σ		$L_\sigma = \{\sigma\}$

Linguaggi denotati da ER

	ER p	$\longrightarrow$	$L_p \subseteq \Sigma^*$
	$\emptyset$		$L_\emptyset = \emptyset$
1.	ε		$L_\varepsilon = \{\varepsilon\}$
	σ		$L_\sigma = \{\sigma\}$
2.			

Linguaggi denotati da ER

ER p	$\longrightarrow$	$L_p \subseteq \Sigma^*$
	$\emptyset$	$L_\emptyset = \emptyset$
1.	ε	$L_\varepsilon = \{\varepsilon\}$
	σ	$L_\sigma = \{\sigma\}$
	$(p + q)$	
2.	$(p \cdot q)$	
	(p^*)	

Linguaggi denotati da ER

	ER p	$\longrightarrow$	$L_p \subseteq \Sigma^*$
	$\emptyset$		$L_\emptyset = \emptyset$
1.	ε		$L_\varepsilon = \{\varepsilon\}$
	σ		$L_\sigma = \{\sigma\}$
	$(p + q)$		$L_{(p+q)} = L_p \cup L_q$
2.	$(p \cdot q)$		$L_{(p \cdot q)} = L_p \cdot L_q$
	(p^*)		$L_{(p^*)} = (L_p)^*$

Linguaggi denotati da ER

	ER p	$\longrightarrow$	$L_p \subseteq \Sigma^*$
	$\emptyset$		$L_\emptyset = \emptyset$
1.	ε		$L_\varepsilon = \{\varepsilon\}$
	σ		$L_\sigma = \{\sigma\}$
	$(p+q)$		$L_{(p+q)} = L_p \cup L_q$
2.	$(p \cdot q)$		$L_{(p \cdot q)} = L_p \cdot L_q$
	(p^*)		$L_{(p^*)} = (L_p)^*$

$$L_1 \cdot L_2 = \{x = u \cdot v \text{ con } u \in L_1, v \in L_2\}$$

Linguaggi denotati da ER

	ER p	$\longrightarrow$	$L_p \subseteq \Sigma^*$
	$\emptyset$		$L_\emptyset = \emptyset$
1.	ε		$L_\varepsilon = \{\varepsilon\}$
	σ		$L_\sigma = \{\sigma\}$
	$(p + q)$		$L_{(p+q)} = L_p \cup L_q$
2.	$(p \cdot q)$		$L_{(p \cdot q)} = L_p \cdot L_q$
	(p^*)		$L_{(p^*)} = (L_p)^*$

$$L^* = \cup_{n \geq 0} (L_p)^n$$

Espressioni regolari in UNIX

Le espressioni regolari sono usate in UNIX come formalismo per specificare linguaggi sull'alfabeto dei caratteri.

Espressioni regolari in UNIX

Le espressioni regolari sono usate in UNIX come formalismo per specificare linguaggi sull'alfabeto dei caratteri.

`grep, sed, lex, vi, awk, ad, ex, pg`

Espressioni regolari in UNIX

Le espressioni regolari sono usate in UNIX come formalismo per specificare linguaggi sull'alfabeto dei caratteri.

`grep, sed, lex, vi, awk, ad, ex, pg`

sono alcuni comandi che effettuano elaborazioni di testi e possono essere applicati a insiemi di stringhe specificate attraverso sottoclassi di ER, oppure attraverso estensioni di ER.

Esempi di applicazioni

- Ricerca di parole, pattern matching, find and replace

Esempi di applicazioni

- Ricerca di parole, pattern matching, find and replace
- Progettazione di compilatori per linguaggi di programmazione

Esempi di applicazioni

- Ricerca di parole, pattern matching, find and replace
- Progettazione di compilatori per linguaggi di programmazione
- Linguaggi di programmazione specifici per la elaborazione di stringhe (es: PERL)

Sintassi delle ER in UNIX

- La sintassi delle espressioni regolari usate nei comandi UNIX non é identica a quella definita precedentemente. Per i dettagli, si veda `man regex`.

Sintassi delle ER in UNIX

- La sintassi delle espressioni regolari usate nei comandi UNIX non é identica a quella definita precedentemente. Per i dettagli, si veda `man regex`.
- Due tipi di ER: *base/semplici* e *estese*.

Sintassi delle ER in UNIX

- La sintassi delle espressioni regolari usate nei comandi UNIX non é identica a quella definita precedentemente. Per i dettagli, si veda `man regex`.
- Due tipi di ER: *base/semplici* e *estese*.

NOTA: possibile confusione-incompatibilita' tra espressioni regolari *base/semplici* e *estese*.

Espressioni regolari semplici su Σ

- Classe su Σ:

$$(x_1 + x_2 + \ldots + x_n)\,, \qquad x_k \in \Sigma$$

Espressioni regolari semplici su Σ

- Classe su Σ:

$$(x_1 + x_2 + \ldots + x_n)\ , \qquad x_k \in \Sigma$$

Sia $\Sigma = \{a, b, c, d\}$

Espressioni regolari semplici su Σ

- Classe su Σ:

$$(x_1 + x_2 + \ldots + x_n)\ , \qquad x_k \in \Sigma$$

Sia $\Sigma = \{a, b, c, d\}$

- Esempi di classi su Σ: $\quad (a) = a$

Espressioni regolari semplici su Σ

- Classe su Σ:

$$(x_1 + x_2 + \ldots + x_n)\ , \qquad x_k \in \Sigma$$

Sia $\Sigma = \{a, b, c, d\}$

- Esempi di classi su Σ: $(b + d)$

Espressioni regolari semplici su Σ

- Classe su Σ:

$$(x_1 + x_2 + \ldots + x_n)\ , \qquad x_k \in \Sigma$$

Sia $\Sigma = \{a, b, c, d\}$

- Esempi di classi su Σ: $(a + b + d + c)$

Espressioni regolari semplici su Σ

- Classe su Σ:

$$(x_1 + x_2 + \ldots + x_n)\ , \qquad x_k \in \Sigma$$

Sia $\Sigma = \{a, b, c, d\}$

- Esempi di classi su Σ: $(a + b + d + c)$
- Non sono classi su Σ: $a \cdot (b + c)$

Espressioni regolari semplici su Σ

- Classe su Σ:

$$(x_1 + x_2 + \ldots + x_n)\ , \qquad x_k \in \Sigma$$

Sia $\Sigma = \{a, b, c, d\}$

- Esempi di classi su Σ: $(a + b + d + c)$
- Non sono classi su Σ: $a^* + c$

Espressioni regolari semplici su Σ

- Espressione regolare semplice su Σ:

$$(c_1 \cdot c_2 \cdots c_n)$$

dove c_k è una classe oppure una classe seguita da $*$

Espressioni regolari semplici su Σ

- Espressione regolare semplice su Σ:

$$(c_1 \cdot c_2 \cdots c_n)$$

dove c_k è una classe oppure una classe seguita da $*$

  - Esempi di ER semplici su $\{a, b, c, d\}$:
    $(a + b + c + d)(a + b)^*$

Espressioni regolari semplici su Σ

- Espressione regolare semplice su Σ:

$$(c_1 \cdot c_2 \cdots c_n)$$

dove c_k è una classe oppure una classe seguita da $*$

  - Esempi di ER semplici su $\{a, b, c, d\}$:
    $(a)(a + c)^*(a + b)$

Espressioni regolari semplici su Σ

- Espressione regolare semplice su Σ:

$$(c_1 \cdot c_2 \cdots c_n)$$

dove c_k è una classe oppure una classe seguita da $*$

  - Esempi di ER semplici su $\{a, b, c, d\}$:
    $(a)(a + c)^*(a + b)$
  - Non sono ER semplici su $\{a, b, c, d\}$:
    $(a + b + c)^* + ab$

Espressioni regolari semplici su Σ

- Espressione regolare semplice su Σ:

  $$(c_1 \cdot c_2 \cdots c_n)$$

  dove c_k è una classe oppure una classe seguita da $*$

  - Esempi di ER semplici su $\{a, b, c, d\}$:
    $(a)(a + c)^*(a + b)$
  - Non sono ER semplici su $\{a, b, c, d\}$:
    $a^* + b^*$

Espressioni regolari semplici in UNIX

Le Espressioni regolari semplici in UNIX sono espressioni regolari semplici sull'alfabeto ASCII, con due avvertimenti:

- I caratteri speciali ∧ $. \ - [ ] * + ? { } | ( ) ” / % < > vanno codificati con parole opportune.

Espressioni regolari semplici in UNIX

Le Espressioni regolari semplici in UNIX sono espressioni regolari semplici sull'alfabeto ASCII, con due avvertimenti:

- I caratteri speciali ∧ $. \ - [ ] * + ? { } | ( ) ” / % < > vanno codificati con parole opportune.
- Si usano dei trucchi per specificare certi sottoinsiemi in forma compatta, sfruttando l'ordinamento dei caratteri ASCII: A < B < ... < Z ; a < b < ... < z ; 0 < 1 < ... < 9 .

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX: parola del tipo x-y con x,y $\in \Sigma$.

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX: parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  $[x_1 x_2 \ldots x_n]$ oppure $[\wedge x_1 x_2 \ldots x_n]$
  dove $x_i \in \Sigma \cup I(\Sigma)$.

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  [$x_1 x_2 \ldots x_n$] oppure [$\wedge x_1 x_2 \ldots x_n$]
  dove $x_i \in \Sigma \cup I(\Sigma)$.

  Esempi:
  $\mathcal{C} =$ [2] $\longrightarrow L = \{2\}$

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  $[x_1 x_2 \ldots x_n]$ oppure $[\wedge x_1 x_2 \ldots x_n]$
  dove $x_i \in \Sigma \cup I(\Sigma)$.

  Esempi:
  $\mathcal{C} =$ [25A] $\longrightarrow L =$ {2,5,A}

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  [$x_1 x_2 \ldots x_n$] oppure [$\wedge x_1 x_2 \ldots x_n$]
  dove $x_i \in \Sigma \cup I(\Sigma)$.

  Esempi:
  $\mathcal{C}$ = [A-D] $\longrightarrow L = \{$A,B,C,D$\}$

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  $[x_1 x_2 \ldots x_n]$ oppure $[\wedge x_1 x_2 \ldots x_n]$
  dove $x_i \in \Sigma \cup I(\Sigma)$.

  Esempi:
  $\mathcal{C} =$ [17-92a-c] $\longrightarrow L =$ {1,7,8,9,2,a,b,c}

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  [$x_1 x_2 \ldots x_n$] oppure [$\wedge x_1 x_2 \ldots x_n$]
  dove $x_i \in \Sigma \cup I(\Sigma)$.

  Esempi:
  $\mathcal{C}$ = [$\wedge$A-D] $\longrightarrow$ $L = \{$A,B,C,D$\}^c$

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  $[x_1 x_2 \ldots x_n]$ oppure $[\wedge x_1 x_2 \ldots x_n]$
  dove $x_i \in \Sigma \cup I(\Sigma)$.
- RE semplice in `UNIX`: concatenazione di classi o classi seguite da *

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  [$x_1 x_2 \ldots x_n$] oppure [$\wedge x_1 x_2 \ldots x_n$]
  dove $x_i \in \Sigma \cup I(\Sigma)$.
- RE semplice in `UNIX`: concatenazione di classi o classi seguite da *
  Esempio: [25A]*[a-d]

Intervalli e classi di caratteri

- Intervallo di caratteri in UNIX:
  parola del tipo x-y con x,y $\in \Sigma$.
- $I(\Sigma) = \Sigma$-$\Sigma = \{$intervalli su $\Sigma\}$.
- Classe in UNIX:
  $[x_1 x_2 \ldots x_n]$ oppure $[\wedge x_1 x_2 \ldots x_n]$
  dove $x_i \in \Sigma \cup I(\Sigma)$.
- RE semplice in `UNIX`: concatenazione di classi o classi seguite da *
  Esempio: [25A]*[a-d]
  Esempio: [2]*[a-d]=2*[a-d]

Espressioni regolari estese

Le ER estese si ottengono iterativamente a partire da quelle semplici:

Espressioni regolari estese

Le ER estese si ottengono iterativamente a partire da quelle semplici:

1. Le ER semplici sono ER estese

Espressioni regolari estese

Le ER estese si ottengono iterativamente a partire da quelle semplici:

1. Le ER semplici sono ER estese
2. Se p, q sono ER estese, allora sono ER estese anche
$$(pq), (p|q), (p)*, (p)+, (p)?$$

Espressioni regolari estese

I simboli possono essere interpretati come operazioni su linguaggi:

ER estesa $r \quad \longrightarrow \quad$ Linguaggio L_r

Espressioni regolari estese

I simboli possono essere interpretati come operazioni su linguaggi:

ER estesa r	$\longrightarrow$	Linguaggio L_r
(pq)		$L_{(pq)}=L_p \cdot L_q$

Espressioni regolari estese

I simboli possono essere interpretati come operazioni su linguaggi:

ER estesa r	$\longrightarrow$	Linguaggio L_r
(pq)		$L_{(pq)=}L_p \cdot L_q$
$(p\|q)$		$L_{(p\|q)} = L_p \cup L_q$

Espressioni regolari estese

I simboli possono essere interpretati come operazioni su linguaggi:

ER estesa r	$\longrightarrow$	Linguaggio L_r
(pq)		$L_{(pq)=}L_p \cdot L_q$
$(p\|q)$		$L_{(p\|q)} = L_p \cup L_q$
$(p)*$		$L_{(p)*} = (L_p)^*$

Espressioni regolari estese

I simboli possono essere interpretati come operazioni su linguaggi:

ER estesa r	$\longrightarrow$	Linguaggio L_r
(pq)		$L_{(pq)=}L_p \cdot L_q$
$(p\|q)$		$L_{(p\|q)} = L_p \cup L_q$
$(p)*$		$L_{(p)*} = (L_p)^*$
$(p)+$		$L_{(p)+} = (L_p)^+ = (L_p)^* \setminus \{\varepsilon\}$

Espressioni regolari estese

I simboli possono essere interpretati come operazioni su linguaggi:

ER estesa r	$\longrightarrow$	Linguaggio L_r
(pq)		$L_{(pq)=}L_p \cdot L_q$
$(p\|q)$		$L_{(p\|q)} = L_p \cup L_q$
$(p)*$		$L_{(p)*} = (L_p)^*$
$(p)+$		$L_{(p)+} = (L_p)^+ = (L_p)^* \setminus \{\varepsilon\}$
$(p)?$		$L_{(p)?} = \{\varepsilon\} \cup L_p$

Esempi di ER estese

Esempi di ER estese

- ([1p-r]3)*|[f9]?
  denota il linguaggio
  $\{13,p3,q3,r3\}^* \cup \{f,9,\varepsilon\}$

Esempi di ER estese

- ([1p-r]3)*|[f9]?
  denota il linguaggio
  $\{13,p3,q3,r3\}^* \cup \{f,9,\varepsilon\}$
- [3-7Z]*|[d-g]?[13]m?
  denota il linguaggio
  $\{3,4,5,6,7,Z\}^* \cup \{d,e,f,g,\varepsilon\} \cdot \{1,3\} \cdot \{m,\varepsilon\}$

Esempi di ER estese

- ([1p-r]3)*|[f9]?
  denota il linguaggio
  $\{13,p3,q3,r3\}^* \cup \{f,9,\varepsilon\}$
- [3-7Z]*|[d-g]?[13]m?
  denota il linguaggio
  $\{3,4,5,6,7,Z\}^* \cup \{d,e,f,g,\varepsilon\} \cdot \{1,3\} \cdot \{m,\varepsilon\}$
- ([3-7Z]*|[d-g]?)[13]m?
  denota il linguaggio
  $(\{3,4,5,6,7,Z\}^* \cup \{d,e,f,g,\varepsilon\}) \cdot \{1,3\} \cdot \{m,\varepsilon\}$

Comando `grep`

`grep` ricerca nei file d'ingresso indicati le righe che contengono un match al modello specificato come ER semplice o estesa e restituisce le righe in cui è trovata la corrispondenza.

Comando `grep`

`grep` ricerca nei file d'ingresso indicati le righe che contengono un match al modello specificato come ER semplice o estesa e restituisce le righe in cui è trovata la corrispondenza.

Sintassi:

```
grep [opz] modello | -f
file_con_modello file_di_input
```

Comando `grep`

`grep` ricerca nei file d'ingresso indicati le righe che contengono un match al modello specificato come ER semplice o estesa e restituisce le righe in cui è trovata la corrispondenza.

Alcune opzioni:

- `-num` mostra `num` righe di contesto
- `-v` stampa le righe che non matchano
- ...

Comando `sed`

`sed` manipola un flusso di input, linea per linea, creando specifici cambiamenti.

Comando `sed`

`sed` manipola un flusso di input, linea per linea, creando specifici cambiamenti.

Sintassi:

```
sed [opz] com_di_editing [file]
```

Comando `sed`

`sed` manipola un flusso di input, linea per linea, creando specifici cambiamenti.

Sintassi:

```
sed [opz] com_di_editing [file]
```

Ecco un comando di editing che effettua operazioni di ricerca e sostituzione:

```
s/ER/stringa_di_rimpiazzo/flag
```

Comando `sed`

```
s/ER/stringa_di_rimpiazzo/g
```

Comando `sed`

```
s/ER/stringa_di_rimpiazzo/g
```

I caratteri della stringa di rimpiazzo non hanno significati speciali, tranne:

- & che viene sostituito con l'ER individuata
- \n che viene sostituito l'n-esima stringa corrispondente all'ER chiusa tra una coppia di \(e \)

Analizzatore lessicale

- Testo in linguaggio naturale
  $\longrightarrow$ gruppi logicamente omogenei di parole.

Analizzatore lessicale

- Testo in linguaggio naturale
  $\longrightarrow$ gruppi logicamente omogenei di parole.
- Codice in linguaggio di programmazione
  $\longrightarrow$ tokens.

Analizzatore lessicale

- Testo in linguaggio naturale
  $\longrightarrow$ gruppi logicamente omogenei di parole.
- Codice in linguaggio di programmazione
  $\longrightarrow$ tokens.
- Esempi di tokens sono le variabili, gli operatori, i commenti...

Analizzatore lessicale

- Testo in linguaggio naturale
  $\longrightarrow$ gruppi logicamente omogenei di parole.
- Codice in linguaggio di programmazione
  $\longrightarrow$ tokens.
- Esempi di tokens sono le variabili, gli operatori, i commenti...
- L'analisi lessicale è la prima fase del processo di compilazione.

Analizzatore lessicale

- Ai token corrispondono linguaggi molto semplici, denotabili con ER;

Analizzatore lessicale

- Ai token corrispondono linguaggi molto semplici, denotabili con ER;
- Ogni token è riconosciuto da un automa a stati finiti (teorema di Kleene);

Analizzatore lessicale

- Ai token corrispondono linguaggi molto semplici, denotabili con ER;
- Ogni token è riconosciuto da un automa a stati finiti (teorema di Kleene);
- Gli automi individuano le sottostringhe del testo appartenenti ai tokens;

Analizzatore lessicale

- Ai token corrispondono linguaggi molto semplici, denotabili con ER;
- Ogni token è riconosciuto da un automa a stati finiti (teorema di Kleene);
- Gli automi individuano le sottostringhe del testo appartenenti ai tokens;
- Si attiva un'azione corrispondente al token, che sostituisce la sottostringa con una nuova;

Analizzatore lessicale

- Ai token corrispondono linguaggi molto semplici, denotabili con ER;
- Ogni token è riconosciuto da un automa a stati finiti (teorema di Kleene);
- Gli automi individuano le sottostringhe del testo appartenenti ai tokens;
- Si attiva un'azione corrispondente al token, che sostituisce la sottostringa con una nuova;
- L'output è una rappresentazione interna del testo iniziale, utile per la compilazione.

Esempio con `lex`

- `lex` è un comando che permette di costruire *analizzatori lessicali*.

Esempio con `lex`

- `lex` è un comando che permette di costruire *analizzatori lessicali*.
- L'analizzatore va specificato attraverso un file costituito da tre parti: definizioni, regole, codice (opzionale); le regole sono della forma "pattern (ER) azione".

Esempio con `lex`

- `lex` è un comando che permette di costruire *analizzatori lessicali*.
- L'analizzatore va specificato attraverso un file costituito da tre parti: definizioni, regole, codice (opzionale); le regole sono della forma "pattern (ER) azione".
- L'output è un programma in C, chiamato `lex.yy.c`.

Esempio con `lex`

- `lex` è un comando che permette di costruire *analizzatori lessicali*.
- L'analizzatore va specificato attraverso un file costituito da tre parti: definizioni, regole, codice (opzionale); le regole sono della forma "pattern (ER) azione".
- L'output è un programma in C, chiamato `lex.yy.c`.
- Il file `lex.yy.c` va compilato, per ottenere l'eseguibile dell'analizzatore lessicale.