

Analisi Lessicale con Lex

Corso di
Linguaggi di Programmazione (B)
CdS in Informatica

Analizzatore lessicale o scanner

- Il parser genera un albero sintattico le cui foglie sono i simboli terminali della grammatica, ovvero i token che lo scanner ha estratto dal programma sorgente e passato al parser.
- Lo scanner può interagire con il parser in due modi differenti:
 - Lavorare in un passo separato, producendo i token in una grossa tabella in memoria di massa;
 - Interagire direttamente con il parser che chiama lo scanner quando è necessario il prossimo token nell'analisi sintattica (preferibile).

4

Analizzatore lessicale o scanner

- Lo scanner, attraverso un esame carattere per carattere dell'ingresso, separa il programma sorgente in parti chiamate token che rappresentano i nomi delle variabili, operatori, label, ecc.
 - ☒ Esempio di token : <IDE, X1>
- Il tipo di token è rappresentato con un numero intero unico (esempio, variabile con il numero 1, costante 2, label 3).

2

Linguaggi regolari e scanner

- Per la costruzione (e riconoscimento) dei simboli base usati in un programma (quello che si indica generalmente come lessico), ad esempio gli identificatori, sono sufficienti le grammatiche lineari.
- I linguaggi regolari vengono riconosciuti in modo efficiente attraverso automi a stati finiti. Tuttavia, sono linguaggi abbastanza limitati e con le loro grammatiche non si possono descrivere anche semplici costrutti dei linguaggi di programmazione.

5

Analizzatore lessicale o scanner

- Il riconoscimento dei token è la prima fase eseguita durante la compilazione di un programma e prende il nome di analisi lessicale.
- Una volta terminata l'analisi lessicale sono stati individuati, ad esempio, le costanti, le variabili, etc. utilizzate nel programma e costruita quella che si chiama tabella dei simboli.

3

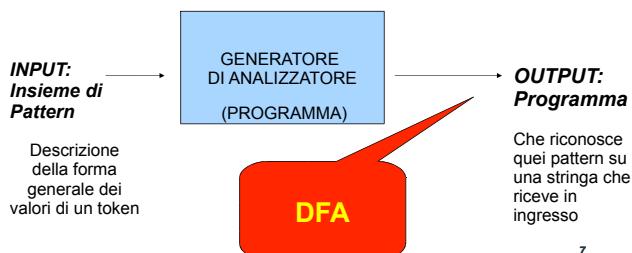
Linguaggi regolari e scanner

- Per la costruzione (e riconoscimento) dei simboli base usati in un programma (quello che si indica generalmente come lessico), ad esempio gli identificatori, sono sufficienti le grammatiche lineari.
- I linguaggi regolari vengono riconosciuti in modo efficiente attraverso automi a stati finiti. Tuttavia, sono linguaggi abbastanza limitati e con le loro grammatiche non si possono descrivere anche semplici costrutti dei linguaggi di programmazione.

6

GENERATORI DI ANALIZZATORI LESSICALI

BASATI SU TECNICHE VISTE NEI MODULI PRECEDENTI



I/O

- Lex accetta una specifica di alto livello per lo string matching e produce un programma che riconosce espressioni regolari

Il programma riconosce tali espressioni nello stream di input partizionandolo in stringhe corrispondenti

Al riconoscimento di una delle stringhe si può eseguire un corrispondente pezzo di codice fornito dall'utente

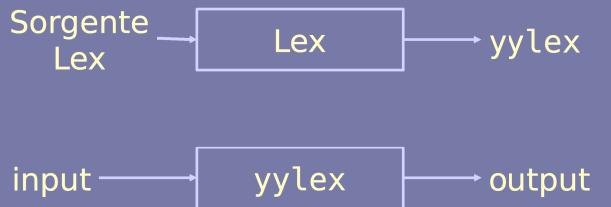
N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Introduzione

- Si veda M. E. Lesk e E. Schmidt Lex - *A Lexical Analyzer Generator*
- Lex è un tool per la generazione di analizzatori lessicali
 - ☒disponibile su Unix
 - ma non solo (vedi FLEX)
 - ☒genera riconoscitori scritti in linguaggio C

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Flusso



N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Finalità

- Lo scopo di Lex è quello di partizionare l'input in un insieme di stringhe ciascuna in corrispondenza di una ben precisa regola sintattica
- Lex è pensato per la costruzione di analizzatori lessicali da interfacciare automaticamente con parser generati da Yacc

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Struttura di un Sorgente Lex

- Un programma Lex è suddiviso in tre sezioni: definizioni, regole e routine ausiliarie; separate dal simbolo %%

```
<definizioni>
%%
<regole>
%%
<routine_ausiliarie>
```

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Sezioni

- La sezione `<regole>` è la più importante ed è l'unica obbligatoria
- In questa sezione sono specificate le regole di matching e le azioni da intraprendere per ogni stringa riconosciuta

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Sezione Regole

- La sezione `<regole>` è una tabella a due colonne:
`<regola_di_matching> <azione>`
 - ☒`<regola_di_matching>` serve ad identificare stringhe sul flusso di input; sono espresse tramite espressioni regolari
 - ☒`<azione>` frammento più o meno complesso di codice C
 - Es. `integer printf("trovato INT");`

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Sezioni

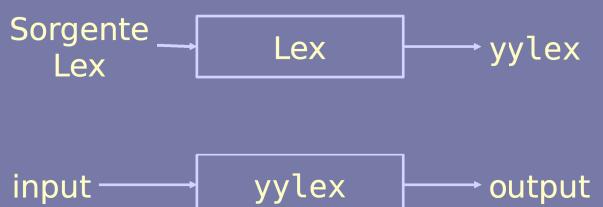
- Per default le stringhe non riconosciute vengono ricopiate sull'output
- ☒ Il più piccolo programma Lex ammesso è:

`%%`

che genera un programma che prende l'input e lo ricopia in output

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Flusso



N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Sezioni

- Nella sezione `<definizioni>` è possibile specificare delle abbreviazioni per i pattern da riconoscere
- Nella sezione `<routine_ausiliarie>` si definiscono funzioni C utilizzate dal riconoscitore generato da Lex
 - `<definizioni>` e `<routine_ausiliarie>` si limitano a fornire strumenti utilizzati nella sezione `<regole>`

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Esempio: sorgente Lex

- Mini-traduttore italiano-inglese (esempio di Walter Cazzola @ Unimi.it)

```
%%  
cane printf("dog");  
pesce printf("fish");  
pesce cane printf("shark");  
io printf("I");  
zio printf("uncle");
```

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Esempio (cont.): input per yylex

- Se al generatore ottenuto forniamo come input:
`il mio cane ed il mio pesce furono mangiati da un pescecane`
- Si avrà come output
`il mI dog ed il mI fish furono mangiati da un shark`

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Espressioni Regolari

- Le espressioni regolari definiscono i pattern con cui le stringhe in input vengono confrontate
- Esse contengono
 - ☒Caratteri di testo
che corrispondono ai normali caratteri nelle stringhe in fase di comparazione
 - ☒Caratteri operatore
che specificano ripetizioni, scelte, ed altre caratteristiche

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Osservazioni

- Lex non distingue dove iniziano e finiscono le parole
 - ☒file di input = sequenza di caratteri scanditi alla ricerca dei pattern definiti
 - ☒Lex non distingue a priori tra porzioni di parola e parole intere ($M^{“IO”} \rightarrow “I”$)

attenti a come si scrivono le regole
è facile ottenere sostituzioni non
desiderate

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Costanti Letterali

- Qualsiasi sequenza di lettere e cifre
 - ☒Ad esempio:
`pippo, z23, 2007`

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Osservazioni

- Lex usa più di un carattere di lookahead per riconoscere un linguaggio
- Lex cerca sempre il match più lungo possibile

Se così non fosse
la parola PESCECANE dell'esempio
sarebbe stata sostituita dalla stringa FISHDOG
e non dalla stringa SHARK

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Operatori

- I seguenti caratteri speciali servono ad esprimere espressioni regolari complesse
 - ☒ preceduto da \ o racchiuso tra virgolette
 - Per riconoscere ritorno a capo, tabulazioni e backspace si usano, resp., i caratteri speciali \n, \t, \b
- Per includere uno di questi caratteri in un pattern (escape)

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Escape

- L'operatore virgolette "..." indica che qualunque cosa sia contenuta tra le virgolette debba essere considerato come caratteri di testo
- Pertanto `xyz"++` corrisponde all'occorrenza della stringa `xyz++`

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Caratteri Arbitrari

- Nell'alfabeto ASCII esistono caratteri non stampabili, per riconoscerli tutti (tranne `return`) si utilizza l'operatore .
- Per riconoscerne solo alcuni si può ricorrere alla loro codifica ottale
 - ☒ ad esempio `\40-\176` corrisponde ai caratteri che ricadono nell'intervallo tra il 32 (`space`) e il 216 (~)

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Classi di Caratteri

- Usando [] si può esprimere un insieme di caratteri
 - ☒ All'interno di [] la maggior parte degli altri operatori perdono il loro significato speciale.
 - ☒ Solo \, - e ^ mantengono il significato particolare
 - \ serve ad effettuare l'escape degli operatori all'interno della classe
 - permette di descrivere in modo sintetico un intervallo di caratteri ASCII
 - ^ permette di definire un insieme di caratteri per negazione

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Espressioni Opzionali

- L'operatore ? indica un elemento opzionale di un'espressione
- Ad esempio il pattern `ab?c` permette di riconoscere sia `abc` che `ac`
 - ☒ La parte opzionale è rappresentata dal carattere o dal gruppo che precede ?

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Esempi

- `[jkl]` corrisponde a j o k oppure l
- `[0-9]` corrisponde a una qualsiasi cifra
- `[0-9+-]` corrisponde a qualsiasi cifra ed in più ai caratteri + e -
- `[^A-Za-z]` riconosce qualsiasi carattere che non sia una lettera

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Espressioni Condizionali

- L'operatore | permette di esprimere un'alternativa all'interno del pattern
- Ad esempio il pattern `ab|cd` permette di riconoscere ab oppure cd

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Gruppi

- L'operatore () permette di annidare le espressioni regolari fornendo un mezzo per costruire pattern molto complessi.
- Ad esempio il pattern $(ab|cd)?e$ permette di riconoscere abe, cde oppure semplicemente e

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Esempi

- $a\{1,4}$ tutte le stringhe: a, aa, aaa, aaaa, aaaaa
- $[a-z]^+$ qualsiasi stringa (non vuota) di minuscole
- $(ab|cd+)?(ef)^*$ stringhe tipo abefef, efefef, cdef, o cddd ma non abc, abcd, o abcdef

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Iterazioni

- Permettono di esprimere pattern complessi che presentano parti ripetute
- Operatori: *, +, {}
 - ☒* applicato ad un pattern, corrisponde ad una stringa in cui quel pattern occorre zero o più volte consecutivamente
 - ☒+ applicato ad un pattern, corrisponde ad una stringa in cui quel pattern occorre una o più volte consecutivamente
 - ☒{} specifica un minimo ed un massimo di occorrenze del pattern a cui è applicato

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Esempi (cont.)

- $[+-]?[1-9][0-9]^*$ qualsiasi numero con e senza segno
- $[A-Za-z][A-Za-z0-9]^*$ tutti gli identificatori

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Espansione

- L'operatore {} specifica anche (oltre alle iterazioni se contengono numeri) o espansioni di definizioni (se racchiudono un nome).
 - ☒Per esempio per {digit} si cerca una stringa predefinita denominata digit e la inserisce in quel punto dell'espressione

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Sensibilità al Contesto

- Operatori per riconoscere pattern all'interno d'un contesto (ristretto)
 - ☒^ come primo carattere di un'espressione, permette di riconoscere un pattern solo quando esso si trovi all'inizio di una riga
 - Dopo newline o all'inizio del flusso di input
 - Non è in conflitto con l'altro uso perché quest'ultimo è limitato all'interno di []

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Sensibilità al Contesto (2)

- Operatori per riconoscere pattern all'interno d'un contesto (ristretto)
 - ☒\$ come ultimo carattere di un'espressione, permette di riconoscere un pattern solo quando esso si trovi alla fine di una riga
 - subito dopo deve esserci newline
 - ☒/ fa riconoscere una stringa solo se seguita da un'altra stringa
 - Ad esempio il pattern ab/cd fa riconoscere ab solo se seguita da cd
 - Quindi gh\$ equivale a gh/\n

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Esempio (cont.)

- Se forniamo come input:
il mio cane ed il mio pesce furono mangiati da un pescecane. La mia gatta mi e' stata regalata da mio zio.
- Si avrà come output
my dog ed my fish furono mangiati da un shark. My cat mi e' stata regalata da my uncle.
- **Domanda:** Cosa succederebbe se nell'input apparisse la parola ozio?

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Ordine dei Pattern

- L'ordine in cui sono specificati i pattern è importante

☒Se si intende specificare

un'azione per a e aa ed
un'altra azione per le stringhe aⁿ, n > 1,2
si devono inserire due pattern

a{1,2} ed a+ esattamente in quest'ordine

altrimenti a+ farebbe riconoscere anche le
stringhe a e aa vanificando gli sforzi

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Azioni

- Per default, se non si esprime alcuna azione, i parser generati da Lex copiano in output tutto ciò che ricevono in input
- Tramite le azioni si può modificare questo comportamento

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Esempio

- Mini-traduttore 2.0

```
%%  
ca(ne|gna)           printf("dog");  
pesce                printf("fish");  
pescecane             printf("shark");  
gatt(o|a)              printf("cat");  
[ \t]+io[ \t]+          printf("dog");  
zio                  printf("uncle");  
mi(o|a)                printf("my");  
((I|i)1)|((L|l)a))[ \t]+mi(o|a)      printf("my");
```

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Azioni

- I casi particolari sono rappresentati dalle azioni predefinite:
 - ☒ECHO permette di mandare in output senza alterare quanto si è riconosciuto
 - ☒REJECT cerca un'alternativa al match corrente
- Il simbolo | dice di associare al pattern l'azione del pattern successivo

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Variabili Predefinite

- Lex mette a disposizione due variabili globali e alcune funzioni per agevolare la manipolazione di quanto riconosciuto
- **yytext** (`unsigned char []`) contiene la stringa appena riconosciuta
☒ Quindi ECHO è un'abbreviazione per
`printf("%s", yytext);`
- **yylengt** contiene il numero di caratteri riconosciuti

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica

Routine Predefinite

- Lex offre routine per rielaborare parte dell'input riconosciuto:
 - ☒ **yymore()**
istruisce il riconoscitore a conservare in yytext la stringa riconosciuta ed ad appendervi quanto sarà riconosciuto nella passata successiva
 - ☒ **yyless()**
istruisce il riconoscitore a reinserire nell'input gli ultimi caratteri riconosciuti

N. Fanizzi * Linguaggi di Programmazione (B) * CdS Informatica