Linguaggi di Programmazione
Corso C

Parte n.8
Analisi Lessicale

Nicola Fanizzi (*fanizzi@di.uniba.it*)

Dipartimento di Informatica
Università degli Studi di Bari

Analizzatore Lessicale

L'analisi lessicale è l'attività del compilatore tesa ad aggregare i caratteri di un programma sorgente per

riconoscere e classificare

le stringhe appartenenti al linguaggio di programmazione

Un analizzatore lessicale (o **Scanner**) svolge questa attività

spesso realizzato con un apposito sottoprogramma del compilatore

agisce su chiamata dell'*analizzatore sintattico*

legge un carattere alla volta finchè non
identifica un simbolo di una categoria sintattica del linguaggio di programmazione

Analizzatore Lessicale II

Categorie sintattiche:

- parole chiave
- identificatori
- numeri
- separatori ed operatori
- stringhe

Altre funzioni:

- eliminazione spazi superflui, commenti e caratteri di controllo
- produzione listati del sorgente con gli eventuali errori
- espansione delle macro

Progetto di uno Scanner

Categorie sintattiche del linguaggio da riconoscere:

parole chiave: `program`, `procedure`, `begin`, `end`, etc.

identificatori: `somma_parziale`, `h1`, etc. (lunghezza max: 15 caratteri)

separatori ed operatori a singolo carattere: `+ - ; :` etc.

operatori a due caratteri: `:=` `<=` etc.

numeri interi positivi: es. `223472`

stringhe: es. `'ciao, mondo!'`

commenti: `{questo e' un commento}`

Spazi e terminatori di linea servono a separare i vari simboli del linguaggio

Lessico

descritto mediante espressioni regolari:

KEYWORD = `program` | `procedure` | `begin` | `end` | . . .

L = A | . . . | Z | . . . a | . . . | z

D = 0| 1 | 2 | . . . | 9

ID = L(D|L)*

O1 = +| - | . . .

O2 = := | <> | <= | . . .

N = (D)+

CSET = *insieme dei caratteri stampabili compreso lo spazio*

CS = *come CSET escluso l'apice*

CC = *come CSET escluso* }

SS = 'CS*'

COMMENT = {(CC|*eoln*)*}

B = (' '|*eoln*)

SYMBOL = KEYWORD | ID | O1 | O2 | N | S

SOURCE = (SYMBOL | COMMENT | B)* *eof*

L'espressione regolare SOURCE descrive un programma sorgente

Errori Lessicali

codice	messaggio di errore
1	identificatore troppo lungo
2	numero intero troppo grande
3	fine linea inattesa
4	fine file inattesa
5	carattere non ammesso

Compiti

1. **aggregare** singoli caratteri del sorgente nei simboli del linguaggio di programmazione

2. **fornire** l'informazione strutturata sul loro contenuto

Strutture record:

```
union CODE_TYPE {
char NAME[16];
int IVAL;
char S_BUF[80];
}

struct TOKEN_TYPE {
int COL;
union CODE_TYPE CODE;
}
```

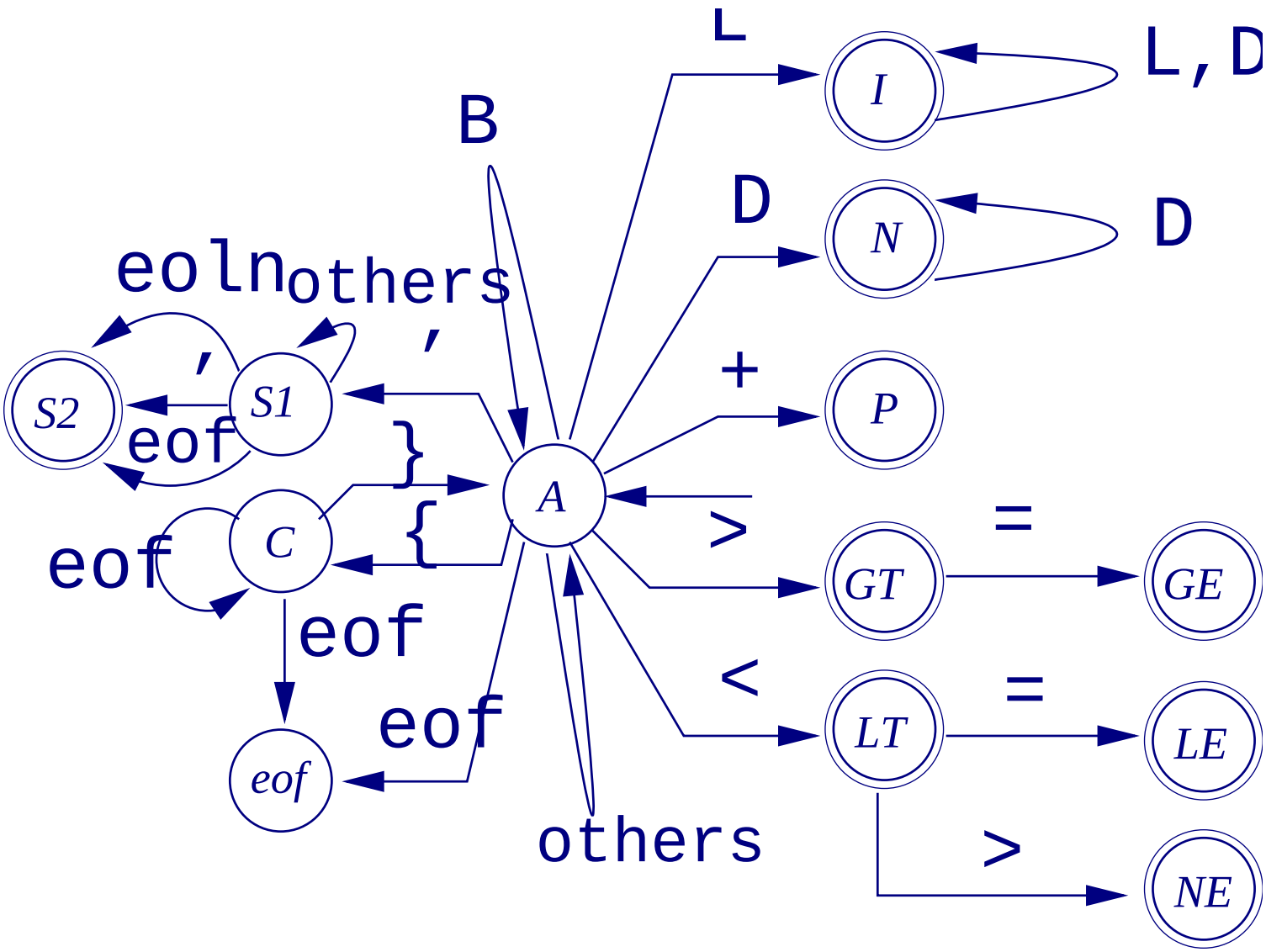
Codici

costante mnemonica	costante numerica	significato
I_C	1	identificatore
N_C	2	intero
S_C	3	stringa
EOF_C	4	fine file
...	...	...
PROGRAM_C	11	keyword PROGRAM
PROCEDURE_C	12	keyword PROCEDURE
BEGIN_C	13	keyword BEGIN
END_C	14	keyword END
...	...	...
PLUS_C	51	operatore +
MINUS_C	52	operatore -
...	...	...
ASSIGN_C	71	operatore :=
GE_C	72	operatore >=
...	...	...

Funzione scanner

```
void SCAN(TOKEN_TYPE * sym)
```

Automa


L
L,D
I
B
D
N
D
eoln
others
'
'
S2
S1
eof
+
P
}
A
{
>
=
C
eof
GT
GE
eof
<
=
eof
LT
LE
eof
others
>
NE

Osservazioni

- simboli terminali: caratteri stampabili, terminatori di linea e di file
- sugli archi un carattere o una espressione regolare
- l'etichetta *others* identifica tutti i caratteri non compresi sugli altri archi
  i caratteri ivi compresi non possono iniziare un simbolo; va fornita una segnalazione d'errore
- stato finale = simbolo riconosciuto; blank e fine linea vengono saltati
- stato EOF = raggiunto il carattere di fine file
- stato S1 = inizio stringa (con un apice);
  se prima del secondo apice si trova un fine-linea o fine-file allora si ha una condizione di errore 3 o 4
- un fine-file non può trovarsi all'interno di un commento
  nel caso, va segnalato come errore 4
- non viene trattata in dettaglio la distinzione tra identificatore e parola chiave

Implementazione

```
/* variabili globali */
char ch; int ch_col;
/* costanti */
const int A_S=0, C_S=1, I_S=2, N_S=3,
S1_S=4, GT_S=6, LT_S=7
void scan(TOKEN_TYPE *sym){
int s=A_S; int exit_flag=0, read_flag=1;
do
switch(s) {
   case A_S:
      switch(ch) {
         case '{':  s=C_S; break;
         case ' ' :
         case EOLN: break;
         case EOF: exit_flag=1; break;
         case 'A':
         ...
         case 'Z': s=I_S; break;
         case '0':
         ...
         case '9':  s=N_S; break;
         case '\'':  s=N_S; break;
         case '+':  exit_flag=1; break;
         case '>':  s=GT_S; break;
         case '<':  s=LT_S; break;
         default:  ;
      } /* switch interno A_S*/
   case C_S:
      switch(ch) {
         case '}':  s=A_S; break;
         case EOF: exit_flag=1; break;
         default:  ;
```

```
   } /* switch interno C_S*/
case I_S:
   switch(ch) {
      case '0':
      ...
      case '9':
      case 'A':
      ...
      case 'Z': break;
      default:  read_flag=0; exit_flag=1;
   } /* switch interno I_S*/
case N_S:
   switch(ch) {
      case '0':
      ...
      case '9':  break;
      default:  read_flag=0; exit_flag=1;
   } /* switch interno I_S*/
case S1_S:
   switch(ch) {
      case '\'':
      case EOLN: exit_flag=1; break;
      case EOF:
         read_flag=0; exit_flag=1; break;
      default:  ;
   } /* switch interno S1_S*/
case GT_S:
   switch(ch) {
      case '=':  exit_flag=1; break;
      default:  read_flag=0; exit_flag=1; break;
   } /* switch interno GT_S*/
case LT_S:
   switch(ch) {
```

```
          case '=':
          case '>':  exit_flag=1; break;
          default:  read_flag=0; exit_flag=1; break;
       } /* switch interno LT_S*/
    default:  /* caso others */
} /* switch esterno s*/

if (read_flag) readch(ch,ch_col);
while(!exit_flag)
} /* funzione scan */
```

- la procedura

```
readch(char *ch,int *ch_col)
```

  fornisce in `ch` il prossimo carattere e
  in `ch_col` la sua posizione nella linea corrente

- occorre anche una procedura di inizializzazione

```
init_scan(token_type *sym)
```

  apre il file sorgente,
  inizializza la tabella delle parole chiave e
  legge il primo carattere chiamando `readch()`

Azioni

sono pezzi di codice da eseguire secondo l'evoluzione del riconoscimento da parte dell'automa

iniziale associato allo stato di partenza

es. $A \longrightarrow I$ il primo carattere dell'identificatore viene copiato (contenuto in `ch`) in un array `sym_name[0]`

associate ad arco codice eseguito quando si transita sull'arco associato

es. $I \longrightarrow I$ la lettera o la cifra in `ch` viene aggiunta alla stringa `sym_name`; se si è raggiunto il limite si segnala l'errore

finali associate ad uno stato finale con archi uscenti

se c'e' stato errore lo si segnala altrimenti si fa il controllo con la tabella delle parole-chiave quindi la categoria sintattica corrispondente sarà identificatore oppure quella della keyword trovata

Implementazione II

```
void scan(TOKEN_TYPE *sym){
int s=A_S; int exit_flag=0 read_flag=1;
do
switch(s) {
   case A_S:
         case 'A':
         ...
         case 'Z':
            sym->col=ch_col;
            i=1; sym->name[0]=ch;
            s=I_S;
            break;
      } /* switch interno A_S*/
...
   case I_S:
      switch(ch) {
         case '0':
         ...
         case '9':
         case 'A':
         ...
         case 'Z':
            if (i<I_LENGTH) {i++; sym->name[i-1]=ch;}
            else error_flag=1;
         default:
            if (error_flag) error(1,sym->col)
            for( ; i<I_LENGTH ; i++) sym->name[i-1]=' ';
            if (is_keyword(sym->name,key))
               sym->code=key;
            else sym->code=I_C;
      } /* switch interno I_S*/
```