Lezione 5
12–13 ottobre 1999

Argomenti trattati
- Grammatiche, forma di Backus–Naur, carte sintattiche.
- La grammatica del linguaggio Pascal: introduzione.
- La struttura di un programma Pascal.
- La selezione: l'istruzione `IF`.
- Il concetto di tipo di dato.
- Tipi di dati semplici: i tipi enumerativi, il tipo `integer`.

5.1 Grammatiche

Per definire la sintassi di un linguaggio occorre specificarne la *grammatica*, cioè l'insieme delle regole per la costruzione delle *frasi* o *sentenze* del linguaggio.

Una grammatica G è definita dai seguenti quattro elementi:

1. un insieme finito T di *simboli terminali*, cioè dei simboli che costituiranno le sentenze del linguaggio;
2. un insieme finito N di *simboli non terminali*, o *metasimboli*, utilizzati nella costruzione delle sentenze del linguaggio;
3. un insieme finito P di *regole di produzione*; per i nostri scopi ci limiteremo a considerare grammatiche in cui le regole di produzione specificano come un metasimbolo possa essere sostituito da una sequenza di simboli terminali e metasimboli;
4. un *simbolo iniziale* S, appartenente all'insieme dei simboli non terminali, utilizzato come punto di partenza nella costruzione delle sentenze.

Il linguaggio generato dalla grammatica G è l'insieme di tutte le sequenze di simboli terminali ottenibili applicando le regole di produzione dell'insieme P, a partire dal simbolo iniziale S.

Esempio

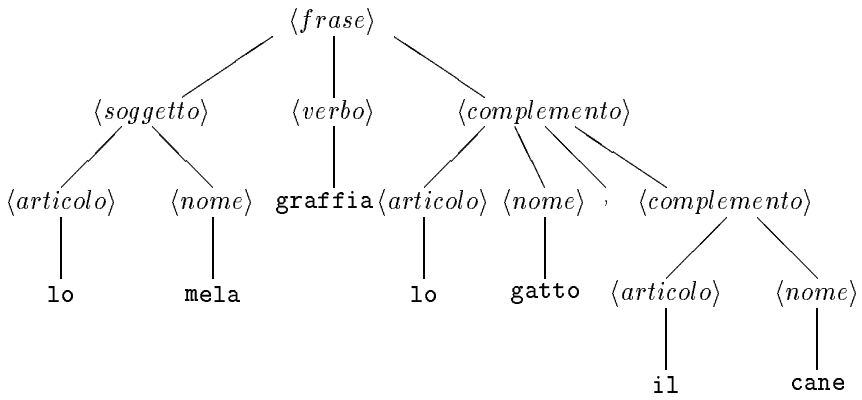
Si consideri la grammatica $G = (T, N, P, S)$ definita come segue:

- $T = \{$`il`, `lo`, `la`, `cane`, `mela`, `gatto`, `mangia`, `graffia`, ,$\}$;

- $N = \{\langle frase\rangle, \langle soggetto\rangle, \langle verbo\rangle, \langle complemento\rangle, \langle articolo\rangle, \langle nome\rangle\}$;
- P è l'insieme formato dalle seguenti regole espresse in *BNF* (forma di *Backus–Naur*):
  - $\langle frase\rangle$ ::= $\langle soggetto\rangle$ $\langle verbo\rangle$ $\langle complemento\rangle$
  - $\langle soggetto\rangle$ ::= $\langle articolo\rangle$ $\langle nome\rangle$
  - $\langle articolo\rangle$ ::= `il` | `la` | `lo`
  - $\langle nome\rangle$ ::= `cane` | `mela` | `gatto`
  - $\langle verbo\rangle$ ::= `mangia` | `graffia`
  - $\langle complemento\rangle$ ::= $\langle articolo\rangle$ $\langle nome\rangle$ | $\langle articolo\rangle$ $\langle nome\rangle$, $\langle complemento\rangle$

  La parte destra di ciascuna regola specifica come deve essere trasformato il metasimbolo indicato sulla sinistra. Il simbolo | indica un'alternativa. Ad esempio l'ultima regola va letta come:
  $\langle complemento\rangle$ è un $\langle articolo\rangle$ seguito da un $\langle nome\rangle$, oppure un $\langle articolo\rangle$ seguito da un $\langle nome\rangle$ seguito dal simbolo , seguito da un $\langle complemento\rangle$;
- $S = \langle frase\rangle$.

La seguente figura mostra la costruzione della sentenza `lo mela graffia lo gatto, il cane` a partire da $\langle frase\rangle$ applicando le regole di produzione sopra indicate.

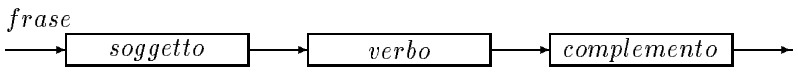


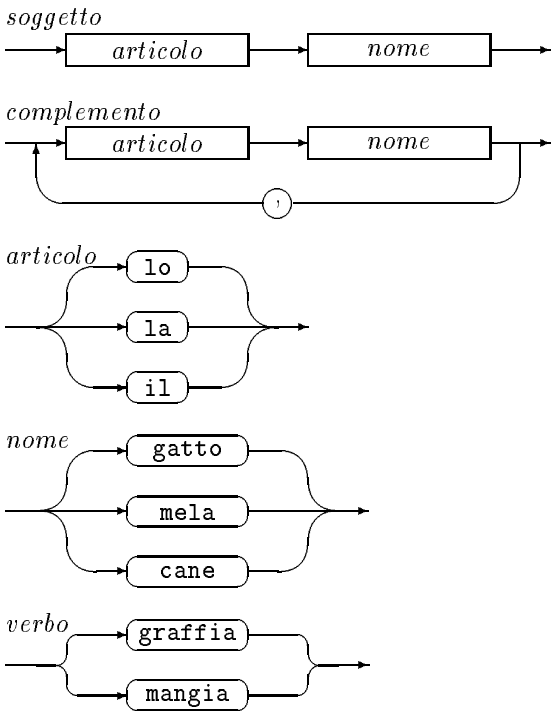
È possibile seguire anche il procedimento inverso, cioè partire da una sequenza di simboli terminali e verificare se essa è una sentenza del linguaggio, risalendo al simbolo iniziale, mediante le regole di produzione.

Un compilatore conosce la grammatica del linguaggio, e applica questo procedimento per verificare la correttezza sintattica dei programmi.

Carte sintattiche

In alternativa alla forma BNF, le regole di produzione possono essere espresse mediante particolari diagrammi detti *carte sintattiche*. In particolare, si specifica una carta sintattica per ciascun simbolo non terminale della grammatica. In una carta sintattica, i rettangoli indicano simboli non terminali (che andranno espansi con le carte sintattiche corrispondenti), mentre gli ovali indicano simboli terminali, che quindi non devono essere ulteriormente espansi. Ogni biforcazione indica un'alternativa. A titolo d'esempio si riportano le carte sintattiche corrispondenti alle produzioni della grammatica introdotta nell'esempio precedente.





5.2 La grammatica del linguaggio Pascal: elementi base

Presentiamo ora alcuni elementi alla base della grammatica del linguaggi Pascal.

Alfabeto: contiene tutte le lettere dell'alfabeto inglese, tutte le cifre, e gli usuali simboli che si trovano sulla tastiera, come $+$, $-$, $*$, /, (, ecc.

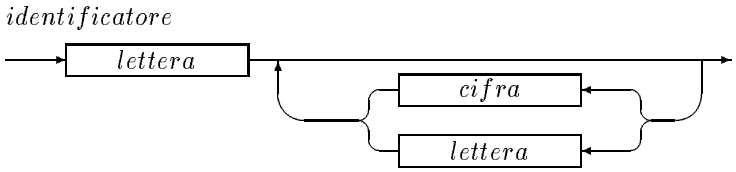
Simboli speciali: sono simboli o sequenze di simboli che assumono un significato particolare all'interno dei programmi. I simboli speciali del Pascal sono:

```
+    -    *    /    =
<    >    <=   >=   <>
.    ,    :    ;    :=    ..
(    )    [    ]    ^
```

Parole riservate (o parole chiave): sono parole che hanno un significato fissato nel linguaggio. Non possono essere utilizzate diversamente, e non possono essere ridefinite. Le parole riservate del Pascal sono:

```
AND          END          NIL          SET
ARRAY        FILE         NOT          THEN
BEGIN        FOR          OF           TO
CASE         FUNCTION     OR           TYPE
CONST        GOTO         PACKED       UNTIL
DIV          IF           PROCEDURE    VAR
DO           IN           PROGRAM      WHILE
DOWNTO       LABEL        RECORD       WITH
ELSE         MOD          REPEAT
```

Gli *identificatori* sono nomi utilizzati nei programmi per indicare constanti, variabili, tipi, ecc. Un identificatore in Pascal è una sequenza di lettere e cifre che inizia con una lettera:



In Pascal esistono alcuni identificatori predefiniti, come ad esempio `input` e `true`, che, a differenza delle parole chiave, possono essere ridefiniti.

Molti testi (ad esempio, Bishop, Grogono, Jensen & Wirth) riportano la grammatica completa del linguaggio Pascal, espressa mediante carte sintattiche.

5.3 Primi esempi di programmi Pascal

Dopo avere definito gli elementi base della grammatica del Pascal, esaminiamo la struttura di un semplice programma:

```
PROGRAM s (input,output);
TYPE
   addendo = -1000 .. 1000;
VAR
   x,y,z: addendo;

BEGIN
   readln(x,y);
   z := x + y;
   writeln(z)
END.
```

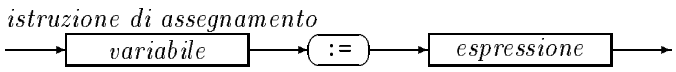
Un programma Pascal inizia sempre con la parola riservata `PROGRAM` e si conclude con un punto. La prima riga del programmma è costituita dall'*intestazione*, che contiene il nome del programma e un elenco dei canali utilizzati dal programma per comunicare con l'esterno, in questo caso `input` e `output`. Le parole `input` e `output` sono due identificatori predefiniti e rappresentano, rispettivamente, il canale d'ingresso standard, associato di solito alla tastiera, e il canale d'uscita standard, associato solitamente al video.

Il testo del programma è poi costituito da due parti fondamentali, una riguardante i *dati* su cui opera il programma, l'altra riguardante le *azioni* svolte dal programma.

La parte relativa ai *dati* contiene in questo caso la definizione del tipo `addendo`, un *subrange* del tipo `integer`, e la dichiarazione delle variabili `x,y,z` di tipo `addendo`. Il significato di queste dichiarazioni, che verrà chiarito in generale piú avanti, è che tutte le variabili dichiarate di tipo `addendo`, cioè le tre variabili `x,y,z`, posso assumere come valori numeri interi, nell'intervallo da `-1000` a `1000`.

La parte riguardante le *azioni* che il programma deve svolgere è racchiusa tra le parole riservate `BEGIN` e `END`. Azioni successive (cioè in sequenza) sono separate dal carattere punto e virgola. Nella prima e nella terza istruzione sono utilizzati gli identificatori predefiniti `readln` e `writeln`, delle procedure standard di lettura e scrittura. La seconda istruzione è un assegnamento. In Pascal, l'operatore d'assegnamento è indicato con il simbolo speciale `:=`.

La carta sintattica dell'istruzione di assegnamento è:



dove, per il momento, con *variabile* intendiamo l'identificatore di una variabile (dichiarata nella sezione `VAR` del programma).

Per eseguire il programma, la macchina Pascal organizza prima di tutto la propria memoria, sulla base della sezione del programma corrispondente ai dati. In questo caso, in base alle dichiarazioni contenute nella sezione `VAR`, vengono predisposti tre contenitori di nomi `x,y,z`, ciascuno dei quali deve essere in grado di contenere un elemento di tipo `addendo`, cioè un intero dell'intervallo da `-1000` a `1000`. Il valore che si trova inizialmente in questi contenitori non è noto.

Una volta che è stato predisposto lo spazio per i dati, può avere inizio l'esecuzione. Per eseguire la prima istruzione, `readln(x,y)`, la macchina si pone in attesa di due numeri da `input`. Quando li riceve, li pone nei contenitori `x` e `y`, eliminando ciò che c'era prima.

L'istruzione successiva è l'assegnamento `z := x + y`. In questo caso, in base al contenuto delle variabili `x` e `y`, la macchina calcola il valore dell'espressione scritta alla destra dell'operatore di assegnamento, e pone il risultato nel contenitore il cui nome è scritto a sinistra, cioè `z`, eliminando ciò che c'era prima.

Infine, con l'istruzione `writeln(z)`, la macchina copia il contenuto di `z` in `output`.

Il seguente programma legge da `input` due numeri interi e scrive in `output` il valore del minimo tra i due. Per determinare il minimo tra i due numeri viene utilizzata un'istruzione di selezione. Abbiamo riportato tra parentesi graffe alcuni commenti che aiutano a capire il significato delle istruzioni presenti nel programma.

```
PROGRAM min (input, output);

{scrive in output il valore del minimo tra due interi letti da input}

VAR x, y, m: integer;

BEGIN
   {leggi i numeri}
   readln(x, y);

   {determina il minimo}
   IF x < y THEN m := x
            ELSE m := y;

   {scrivi il risultato}
   writeln('Il minimo e'' :',m)
END.
```

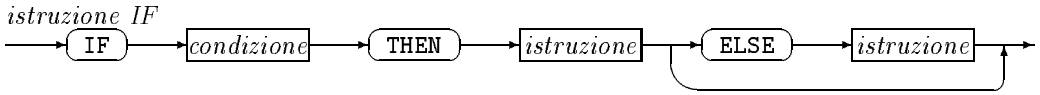
Nell'ultima istruzione del programma viene chiamata la procedura predefinita `writeln` con due parametri. Il primo parametro è la stringa di caratteri `'Il minimo e'' :'` che in output apparirà come

```
Il minimo e' :
```

L'apice iniziale e l'apice finale servono per delimitare la stringa. Il doppio apice interno, dopo la lettera `e`, serve per indicare un singolo apice all'interno della stringa. Il secondo parametro di `writeln` è la variabile `m`. In questo modo, a destra della stringa di caratteri verrà visualizzato il contenuto della variabile `m`.

5.4 La selezione

Nel programma precedente abbiamo utilizzato il costrutto fondamentale per la selezione, la cui carta sintattica è:



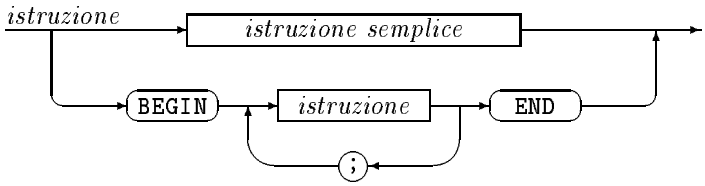
In altre parole, un costrutto `IF` ha la forma:

```
IF <condizione> THEN <istruzione>
                ELSE <istruzione>
```

in cui la parte `ELSE <istruzione>` può essere omessa.

Si noti l'assenza di una parola chiave (equivalente a `FINESE`), per indicare la fine del costrutto. Inoltre, ciascuna alternativa è formata da un'unica istruzione.

Tuttavia, un'istruzione Pascal è un'*istruzione semplice* (come le istruzioni di assegnamento, le istruzioni `IF`, e altre istruzioni che introdurremo in seguito), o un'*istruzione composta*, cioè un'istruzione ottenuta raggruppando piú istruzioni, separate tra di loro da punti e virgola, tra le parole chiave `BEGIN` ed `END`, che fungono così da parentesi. Formalmente, la carta sintattica delle istruzioni Pascal è dunque:



Esempio

Dati in ingresso due numeri, il dividendo e divisore, vogliamo calcolare quoziente e resto.

Per il calcolo del quoziente, privo di decimali, e del resto della divisione tra valori di tipo `integer`, il Pascal dispone degli operatori `DIV` e `MOD`.

```
PROGRAM divisioni (input,output);

VAR dividendo, divisore, quoziente, resto: integer;

BEGIN {divisioni}

   {prima fase: lettura dei dati}
   write('Inserisci il dividendo e il divisore ');
   readln(dividendo,divisore);

    {seconda fase: verifica dei dati, eventuale calcolo e scrittura risultati}
    IF divisore = 0
       THEN
          {divisione impossibile}
          writeln('Errore: non e'' possibile dividere per 0')
       ELSE
          BEGIN
             {calcolo del quoziente e del resto}
             quoziente := dividendo DIV divisore;
             resto := dividendo MOD divisore;

             {scrittura dei risultati}
```

```
            writeln('Il resto quoziente della divisione e''',quoziente);
            writeln('Il resto della divisione e''',resto);
          END {else}

END. {divisioni}
```

5.5 Tipi di dati

Nella memoria della macchina le informazioni sono rappresentate come sequenze di bit. In questo modo, informazioni di natura differente vengono rappresentate in modo omogeneo, facilitando il trattamento dei dati da parte del processore. D'altra parte, per il programmatore è piú comodo immaginare i dati così come sono, senza doversi occupare di come essi vengano rappresentati effettivamente dalla macchina. I linguaggi ad alto livello come il Pascal permettono di astrarre rispetto alla rappresentazione dei dati utilizzata dalla macchina, fornendo la possibilità di definire svariati *tipi* di dati.

Piú precisamente, il *tipo* di una variabile è un attributo che specifica la classe di valori che quella variabile può assumere. Ogni tipo, oltre che da un insieme di valori, è caratterizzato da un insieme di possibili *operazioni*. Ad esempio, l'operazione di divisione intera, indicata con l'operatore `DIV`, è un'operazione che opera su due valori di tipo `integer` e restituisce un valore di tipo `integer`; tale operazione non è applicabile ai numeri reali.

In Pascal le variabili devono essere dichiarate all'inizio del programma o dei sottoprogrammi. Dichiarando una variabile se ne specificano il *nome* e il *tipo*. Questi due attributi non possono essere modificati durante l'esecuzione. Il terzo attributo di una variabile, cioè il *valore*, può invece mutare nel corso dell'esecuzione.

In Pascal i tipi di dati sono suddivisi in tre gruppi fondamentali:

Tipi semplici: corrispondono a valori elementari (come ad esempio un numero intero o un giorno della settimana).

Tipi strutturati: corrispondono a valori che a loro volta sono composti da altri valori (ad esempio una data).

Tipi puntatori: permettono la creazione dinamica di nuove variabili durante l'esecuzione.

Nella prima parte del corso tratteremo esclusivamente i tipi semplici. I tipi strutturati e i tipi puntatori verranno esaminati, rispettivamente, nella seconda e nella terza parte del corso.

5.6 I tipi semplici

I tipi semplici corrispondono a valori elementari, cioè a valori non ulteriormente decomponibili. In Pascal esistono tipi semplici definibili dal programmatore e tipi semplici predefiniti. In particolare, i tipi semplici sono a loro volta suddivisi nella famiglia dei *tipi scalari* e nel tipo `real`. Esamineremo prima di tutto la famiglia dei tipi scalari, suddivisa a sua volta in *tipi enumerativi*, tipo `integer`, tipo `char`, tipo `boolean` e tipi *subrange*.

5.7 Tipi enumerativi

Un tipo enumerativo viene definito elencando tutti i valori (o meglio, gli identificatori dei valori) che compongono il tipo. Ad esempio, la dichiarazione

```
TYPE
   giorno = (lun, mar, mer, gio, ven, sab, dom);
   colore = (rosso, giallo, verde, blu, bianco, nero);
```

definisce due tipi, di nomi `giorno` e `colore`, insieme agli identificatori delle costanti dei due tipi `lun, mar, ...`, e `rosso, giallo, ...`. Tra gli identificatori di ciascun tipo viene definito implicitamente un ordinamento, per cui risulta, ad esempio, `mar < ven` e `nero >= verde`. Sono disponibili due funzioni `pred` e `succ` che restituiscono, rispettivamente, il predecessore e il successore di un elemento di un tipo enumerativo. Ad esempio `pred(gio)` è `mer`. La funzione `pred` non è definita per il primo elemento dell'enumerazione, mentre la funzione `succ` non è definita per l'ultimo.

È possibile definire un nuovo tipo per *subrange*, cioè restringendo l'insieme dei valori di un tipo già definito. Ad esempio, la definizione

```
TYPE
   giornolavorativo = lun..ven;
```

definisce un tipo corrispondente alla restrizione del tipo `giorno` ai valori compresi nell'intervallo da `lun` a `ven`.

5.8 Il tipo `integer`

Il tipo `integer` è un tipo scalare predefinito del Pascal, le cui costanti sono i numeri interi che la macchina è in grado di rappresentare. Il massimo intero rappresentabile è dato dal valore della costante predefinita `maxint` (quando gli interi sono rappresentati utilizzando 2 byte, di solito i valori rappresentabili vanno da -32768 a 32767).

Sono disponibili i seguenti operatori binari tra interi, che forniscono un risultato intero: `+`, `-`, `*`, `DIV`, `MOD`. Essi denotano rispettivamente le operazioni di somma, sottrazione, moltiplicazione, divisione intera, modulo. Inoltre sono disponibili gli usuali operatori unari `+` e `-`, per indicare numeri positivi o negativi. Le costanti predefinite di tipo intero vengono indicate con sequenze di cifre, eventualmente precedute dal segno, come `-48` o `765`.

Nota. Lo Standard ISO 7185 del Pascal prevede che l'operazione `x MOD y`, con `y`>0, fornisca come risultato la *classe di resto modulo* `y` contenente `x`. In particolare, il resto della divisione di `x` per `y` è dato dalla formula `x - (x DIV y) * y`. Quando il resto è negativo, il risultato di `x MOD y` è uguale al resto piú `y`, altrimenti coincide con il resto. Ad esempio, `5 MOD 3` è uguale a 2, mentre `-5 MOD 3` è uguale a 1. Lo Standard prevede inoltre che l'operazione `x MOD y` provochi un errore d'esecuzione quando il valore di `y` è minore o uguale a zero.

Esercizi

1. Individuate, senza ricorrere all'esecuzione sulla macchina, ma simulando con carta e penna le azioni eseguite dalla macchina Pascal durante l'esecuzione, le funzioni svolte dai seguenti programmi:

   ```
   PROGRAM a (input,output);
   VAR x,y: integer;
   BEGIN
      readln(x,y);
      x := y;
      y := x;
      writeln(x,y)
   END.
   ```

```
PROGRAM b (input,output);
VAR x,y,z: integer;
BEGIN
   readln(x,y);
   z := x;
   x := y;
   y := z;
   writeln(x,y)
END.
```

```
PROGRAM c (input,output);
VAR x,y: integer;
BEGIN
   readln(x,y);
   x := x + y;
   y := x - y;
   x := x - y;
   writeln(x,y)
END.
```

2. Abbiamo visto che un'istruzione IF può avere la forma

```
IF <condizione> THEN <istruzione>
                ELSE <istruzione>
```

oppure la forma

```
IF <condizione> THEN <istruzione>
```

Le parti indicate con `<istruzione>` potrebbero essere a loro volta istruzioni IF. In questo modo è possibile innestare istruzioni IF all'interno di altre. Si consideri il programma

```
PROGRAM p (input,output);
VAR x,y: integer;
BEGIN
   readln(x,y);
   IF x>0 THEN IF y>0 THEN writeln('eseguo THEN') ELSE writeln('eseguo ELSE');
   writeln('Fine programma')
END.
```

Individuate una coppia di valori di input, che permetta di capire a quale dei due IF si riferisce la clausula ELSE. Fate compilare ed eseguire il programma su tale coppia. Come deve essere modificato il programma affinché la clausula ELSE venga associata all'IF che ne è privo?