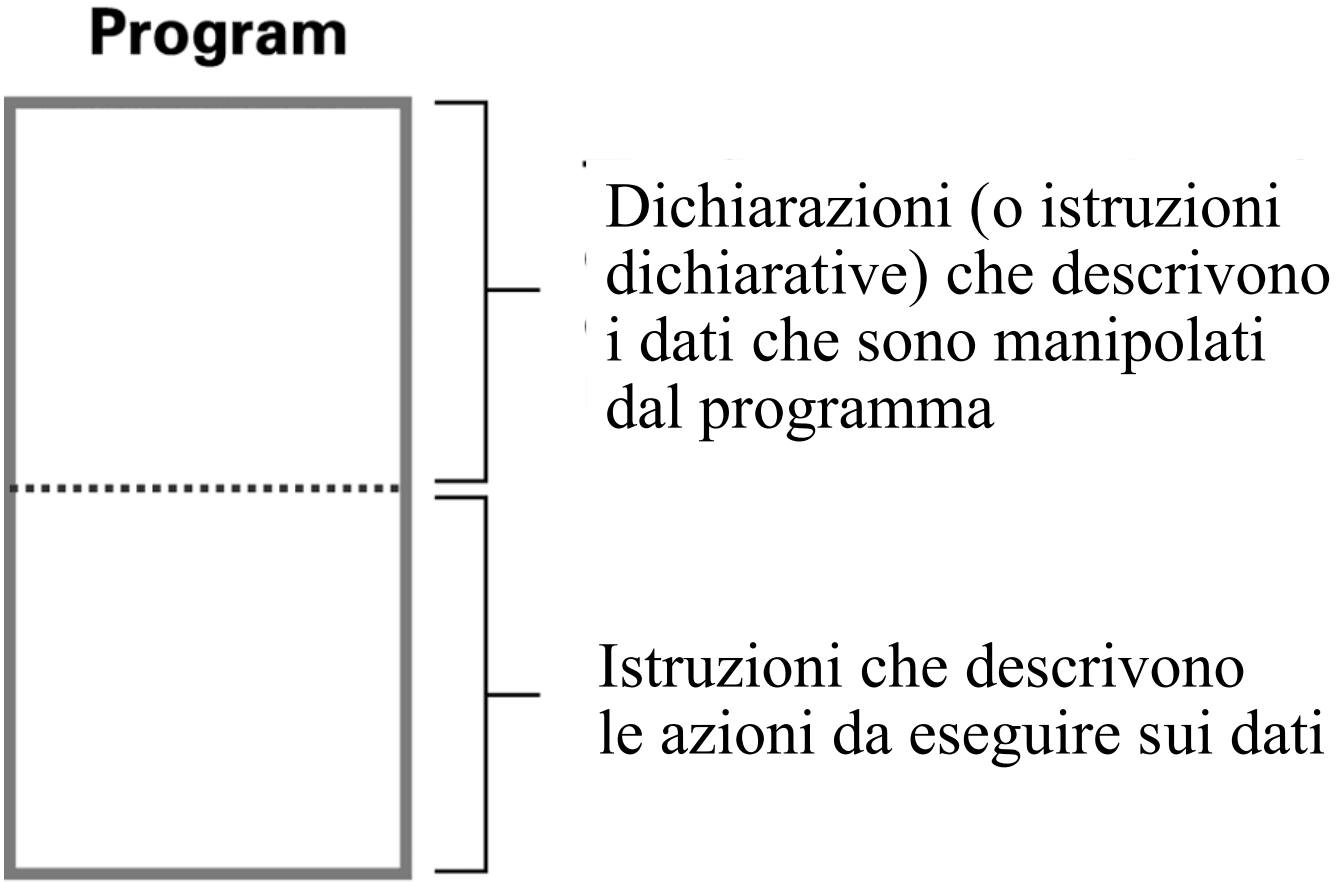
18 - Ancora sui linguaggi di programmazione

Sommario

- Principali componenti di un linguaggio di programmazione
  - Variabili e costanti
  - Strutture dati: array e record
  - Strutture di controllo
  - Procedure e funzioni: passaggio dei parametri
- L'implementazione dei linguaggi: la compilazione
  - L'analisi lessicale
  - Il parser
  - La generazione del codice

Principali componenti di un programma (imperativo)



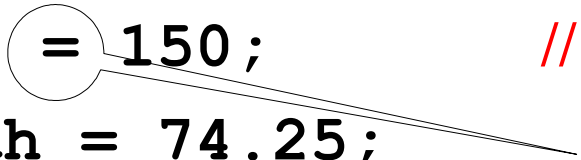
Variabili e costanti

- Le dichiarazioni introducono le **variabili**: celle di memoria caratterizzate da identificatore (il nome) e da un valore (assunto mediante *assegnamenti*)
- Ogni variabile ha un **tipo**, che determina i valori che può assumere (questo determina lo spazio necessario in memoria).
  Esempi di **tipi di dati**:
  - **booleani** (0,1),
  - **interi** (rappresentati in complemento a due)
  - **reali** (rappresentato in virgola mobile)
  - **caratteri** (rappresentati con ASCII (in C) o UNICODE (in Java))

- Si possono anche dichiarare delle **costanti:** hanno un nome e un valore come le variabili, ma il valore non può essere cambiato con un assegnamento.

Esempi di dichiarazioni di variabili e di assegnamenti (C, C++, Java...)

```
float Length, Width;  // dichiara due variabili di tipo float
int   Price, Total, Tax;
char Symbol = '*';  // dichiarazione con inizializzazione
Price = 150;          // assegnamento
Length = 74.25;
Symbol = '#';
Total = 125.76;   // NO: errore di tipo!
Width = 34;       // OK: un intero è anche un reale...
```



si noti il simbolo di assegnamento!!

Costanti e letterali (literals)

- Una ***costante*** è essenzialmente una variabile (ha un nome scelto dal programmatore e un valore) il cui valore non può essere modificato.
- Un ***letterale*** è una sequenza di caratteri che rappresenta uno specifico valore.
- Esempi di letterali: **`157`** (di tipo **`int`**), **`3.14`** (**`float`**), **`'x'`** (**`char`**), **`"pippo"`** (**`String`** o **`char[]`**)

```
const float PI = 3.1416; // Una dichiarazione di costante
```

Dichiara la costante **`PI`** di tipo **`float`** e le assegna il valore **`3.1416`**.

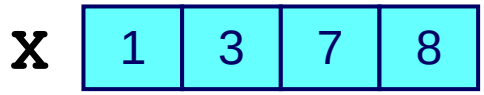
- E' buona prassi evitare di usare letterali nei programmi, introducendo opportune costanti.

Strutture dati: array e record

- Tutti i linguaggi di programmazione ad alto livello permettono di definire due tipi di aggregati di variabili (o *strutture dati*)
  - ***array*** *(array omogenei)*: sequenze di valori tutti dello stesso tipo
  - ***record*** *(array eterogenei)*: gruppi di variabili di tipo diverso
- Le strutture dati seguono criteri logici e non fisici.
- I ***compilatori*** traducono le istruzioni che manipolano strutture dati in istruzioni che operano su locazioni di memoria.

Array (omogenei)

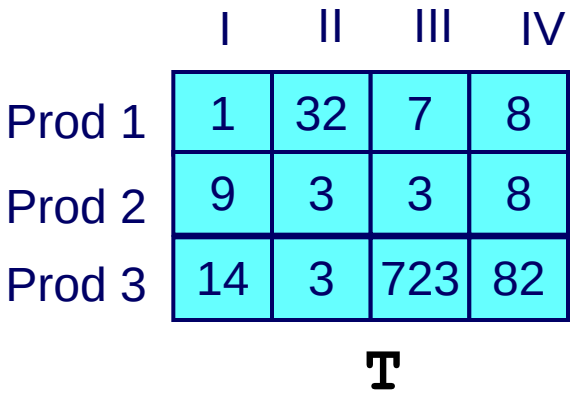
- Possiamo definire sequenze di variabili di una certa lunghezza



X: Array monodimensionale di variabili intere di dimensione 4

Array di 3 stringhe (sequenze di caratteri)

- Possiamo definire tabelle a due dimensioni (matrici di variabili), ad esempio per memorizzare le vendite di un prodotto in un trimestre dell’anno



T: Array bidimensionale di interi di dimensione 3x4

Dichiarazione di array

- Come si specifica la struttura di un array? E come si usano le singole variabili nella struttura?
  - La struttura si specifica indicando nella dichiarazione il tipo delle variabili e le dimensioni
  - es :

- `int X[4];`
- `int T[3][4];`

Nomi degli array

X

1	3	7	8

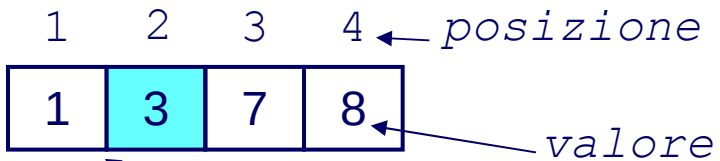
T

	I	II	III	IV
Prod 1	1	32	7	8
Prod 2	9	3	3	8
Prod 3	14	3	723	82

Accesso a variabili di un array

- Uso di una singola variabile di un array:
  - Si usa il nome dell'array, specificando (tra parentesi quadre) le coordinate della variabile desiderato
  - Ogni elemento di ogni dimensione è identificato da un indice compreso tra *1* a *N* (o tra *0* a *N-1*, dipende dal linguaggio)

Noi generalmente seguiremo la convenzione di partire da 1 (come FORTRAN, diverso da C, Java...)



`X[2]`

	I	II	III	IV
Prod 1	1	32	7	8
Prod 2	9	3	3	8
Prod 3	14	3	723	82

`T[2][1]`

Cosa fa il seguente comando?

```
if (X[2] > T[2][1])
then T[2][1] = X[2]
else X[2] =  T[2][1]
```

Esempio

- Modificare la procedura **RicLin** in pseudocodice, in modo che:
- abbia tre parametri: **`X`** di tipo **`char`**, **`S`** di tipo array di **`char`**, e **`N`** di tipo **`int`**, che rappresenta la lunghezza di **`S`**
- restituisca la posizione nell'array in cui si trova **`X`**, o **`-1`** se non c'è

Versione originale in pseudocodice:

```
procedure RicLin (X, S) // in verde i cambiamenti
  N ← lunghezza della sequenza S;
  k ←  1;    //indice del prossimo elemento di S
  trovato ← FALSE; // variabile booleana
  while (NOT trovato AND  k <= N) do
     if ( X = Sk)
     then  trovato <=  TRUE;
     else   k ← k + 1;
  return trovato;
```

Esercizio

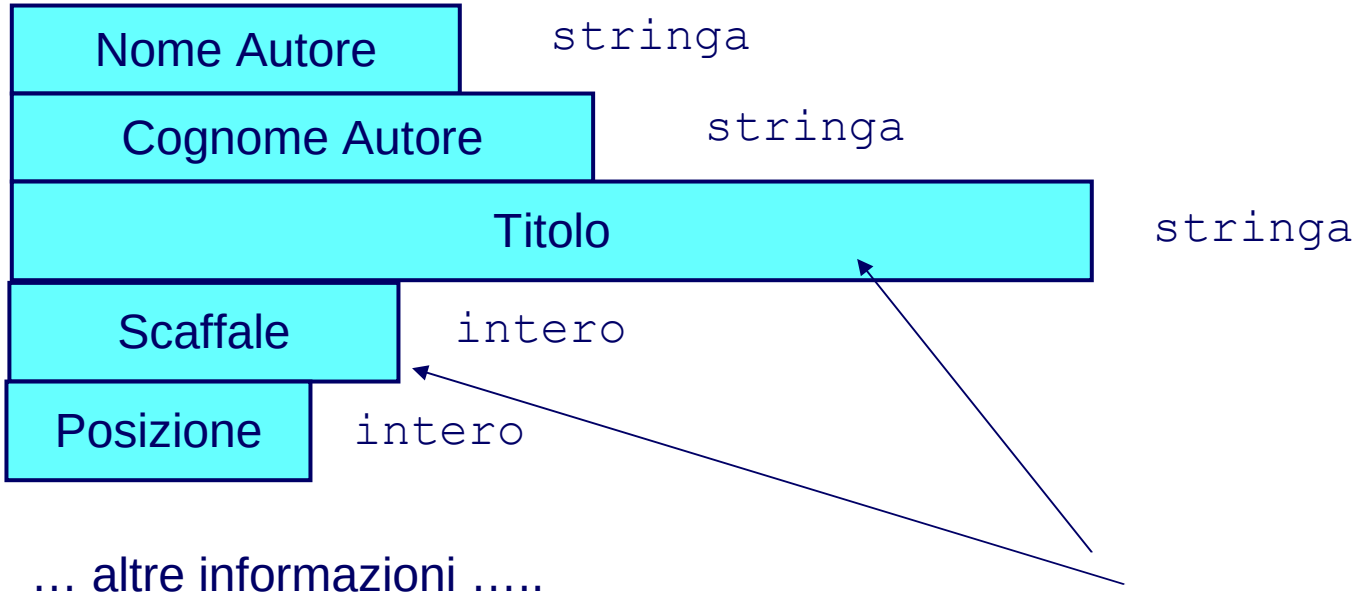
- Scrivere una procedura **palindromo** che ha come parametri un array di interi A e la sua lunghezza N, e restituisce **true** se A è palindromo, **false** altrimenti.

N.B. Una sequenza di numeri (o caratteri) è **palindroma** se, letta al rovescio, rimane identica. Es.:

A = 011110, A = “anna”, A= “ailatiditalia” sono palindrome

Record (o *array* eterogenei)

- Gli array sono sequenze di variabili dello stesso tipo
- I ***record*** sono aggregati di variabili di tipo *diverso* e permettono di definire nuovi tipi di dati
- A cosa possono servire…..
  - Per esempio, a rappresentare le schede della biblioteca:



Campi del record

Definizione di record in C

- Il tipo record *scheda* espresso in linguaggio C (**`struct`**)

```
struct scheda {
    char nome[100]; //stringa di al più
                    //100 caratteri
    char cognome[100];
    char titolo[300];
    int scaffale;
    int posizione;
};
```

- Questo **dichiara** un nuovo ***tipo di dati*** chiamato **`scheda`**, ma **non crea** nessun record.

Creazione e uso di record

```
// dichiaro una variabiledi tipo 'scheda'
    struct scheda nuovo_libro;

// assegno valori ai diversi campi
  nuovo_libro.nome = ”Jorge";
  nuovo_libro.cognome = ”Amado";
  nuovo_libro.titolo =
      ”Dona Flor e i suoi due mariti";
  nuovo_libro.scaffale = 8;
  nuovo_libro.posizione = 356;
```

Array di record

- Si possono definire array di record
  - questo può servire, ad esempio a definire l'insieme delle schede di una biblioteca:

```
struct scheda archivio[100000]
```

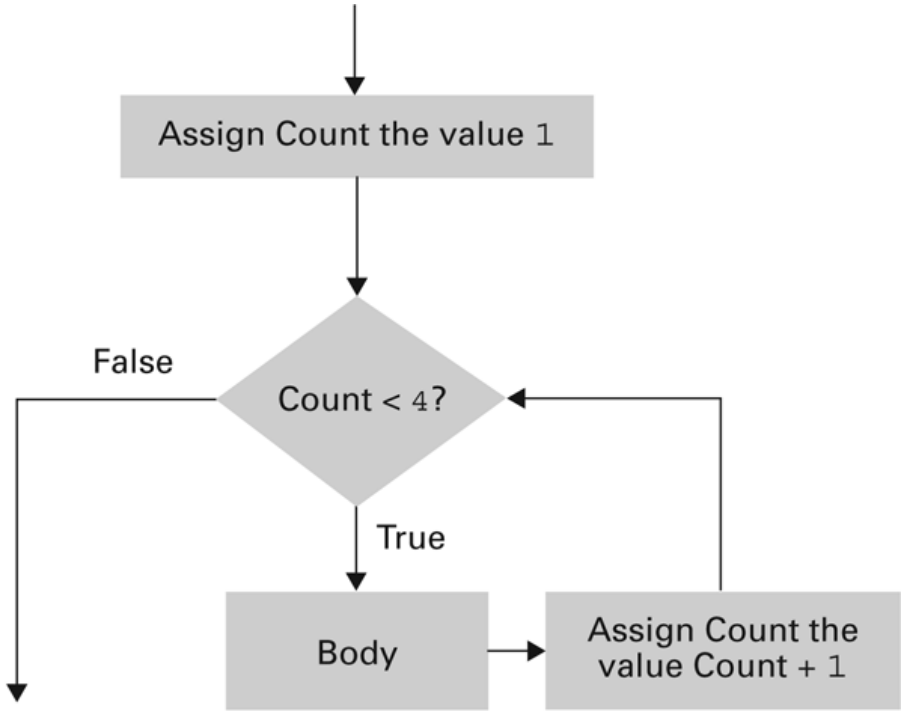
Importante: **`archivio[2]`** è il record con indice 2 dell'array **`archivio[100000]`**, mentre **`archivio[2].titolo`** è il campo titolo del libro che corrisponde al record **`archivio[2]`.**

- **Esercizio**: scrivere un algoritmo che restituisce il titolo del primo libro scritto da un autore di cognome "Amado" in un array di record **`archivio`** di lunghezza **100000**.

Strutture di controllo

- I linguaggi di programmazione ad alto livello hanno essenzialmente le stesse strutture di controllo che abbiamo visto per lo pseudo-codice:
  - **assegnamento**
  - **if-then-else**
  - **while-do** e variazioni, come **repeat-until** e **do-while**
- Le regole sintattiche sono più rigide, e dipendono dal linguaggio.
- Ci sono altri comandi che non abbiamo visto:
  - selezione tra più alternative (**case** o **switch**)
  - iterazione determinata (**for**, molto comune)

Strutture di controllo: significato e esempio di for




```
for (int Count = 1; Count < 4; Count++)
    body ;
```

incrementa la variabile `Count` di 1

```
float prezzi [10];
...   // riempio prezzi
float totale = 0;
for (int i = 1; i <= 10; i++){
  totale = totale + prezzi[i];
}
print totale;
```

- *Calcola la somma dei valori nell'array prezzi*
- *con **while**:*

```
int i = 1;
while (i <= 10){
  totale = totale + prezzi[i];
  i++;
}
```

Procedure e funzioni

- Anche le procedure nei linguaggi ad alto livello sono molto simili a quelle dello pseudo-codice. Chiamiamo **funzione** una procedura che restituisce un risultato.
- Distinguiamo tra **dichiarazione** e **invocazione** di procedura.
- Una **dichiarazione di procedura** ha:
  - un'**intestazione** che comprende il **nome**, l'eventuale **tipo del risultato** e la lista dei **parametri formali** (nomi di variabili preceduti dal tipo)
  - un corpo, contenente dichiarazioni di variabili locali e istruzioni.
- Una **invocazione** o **chiamata** di procedure comprende il nome della procedura e una lista di espressioni (i **parametri attuali**)

La procedura PrevediPopolazione in C

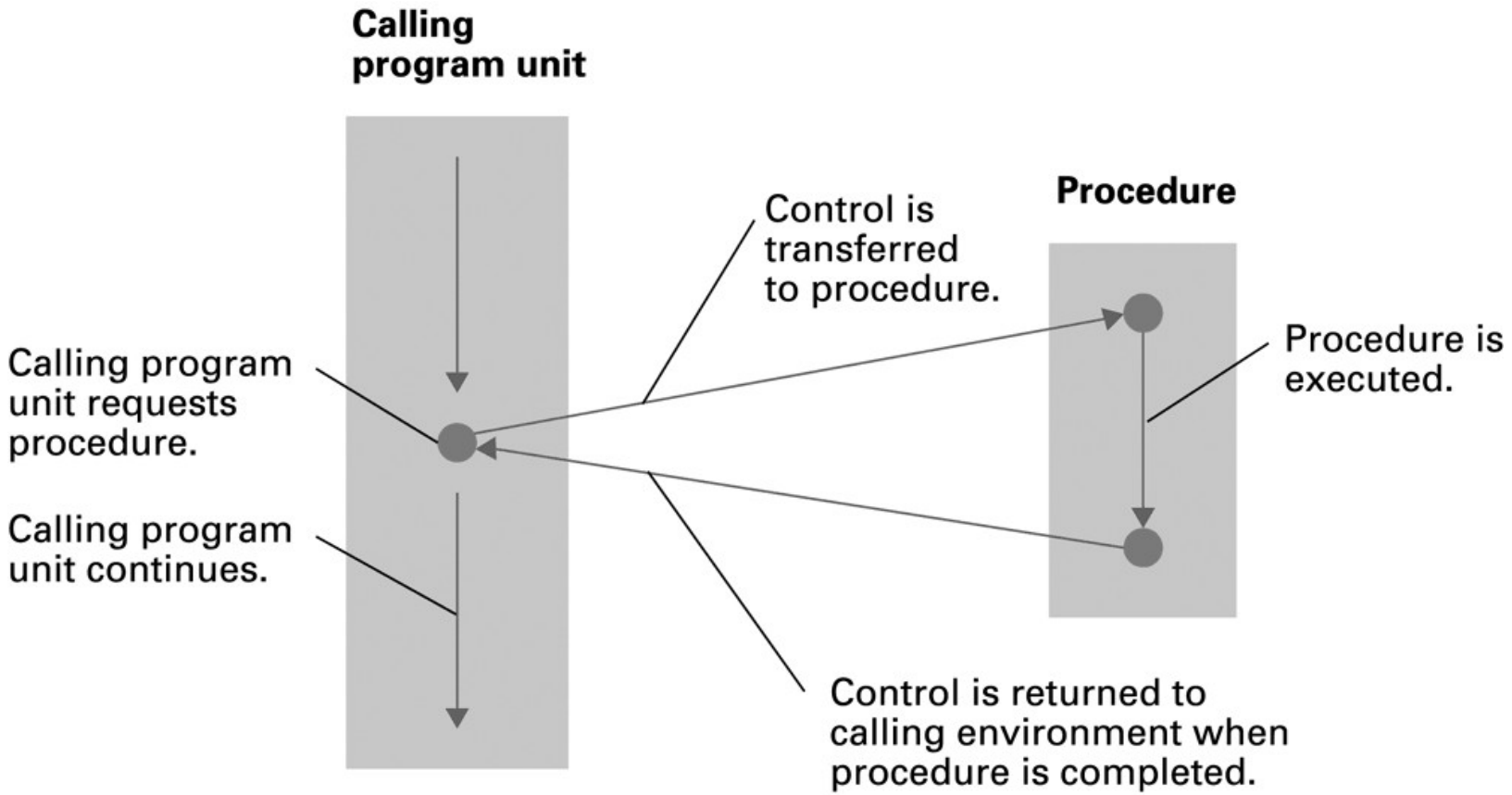
tipo del risultato

parametro formale

```
void PrevediPopolazione (float CrescitaAnnuale)
{int Anno;
 Popolazione[1] = 100.0;
 for (Anno = 1; Anno < 10; Anno++){
 Popolazione[Anno+1] = Popolazione[Anno] +
            Popolazione[Anno] * CrescitaAnnuale;
}
// Esempio di chiamata:
PrevediPopolazione(3.75);
// stampo il contenuto dell'array Popolazione
```

dichiarazione di variabile locale

Il flusso di controllo dell'invocazione di procedura



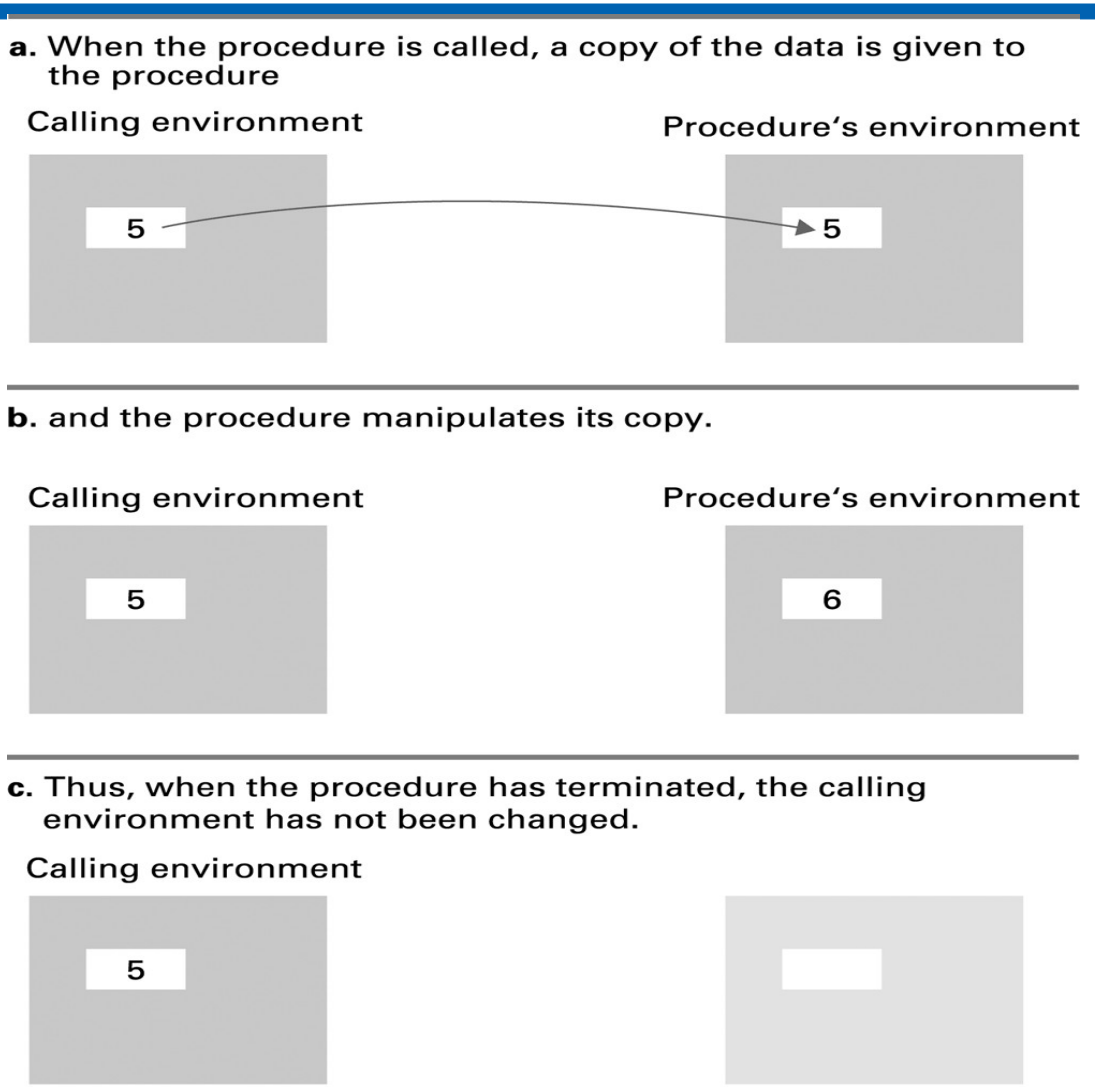
Il passaggio dei parametri

- Quando si invoca una procedura, i parametri formali prendono il valore dei parametri attuali. Ci sono molti modi per farlo:
  - passaggio per **valore**: il valore del parametro attuale viene copiato nel parametro formale
  - passaggio per **riferimento**: il parametro formale prende un riferimento al parametro attuale, senza farne una copia.
  - molti altri modi...
- I diversi tipi di passaggio di parametri hanno effetti diversi

Passaggio di parametri per valore

```
procedure Demo(int Formale)
    Formale = Formale + 1;
```

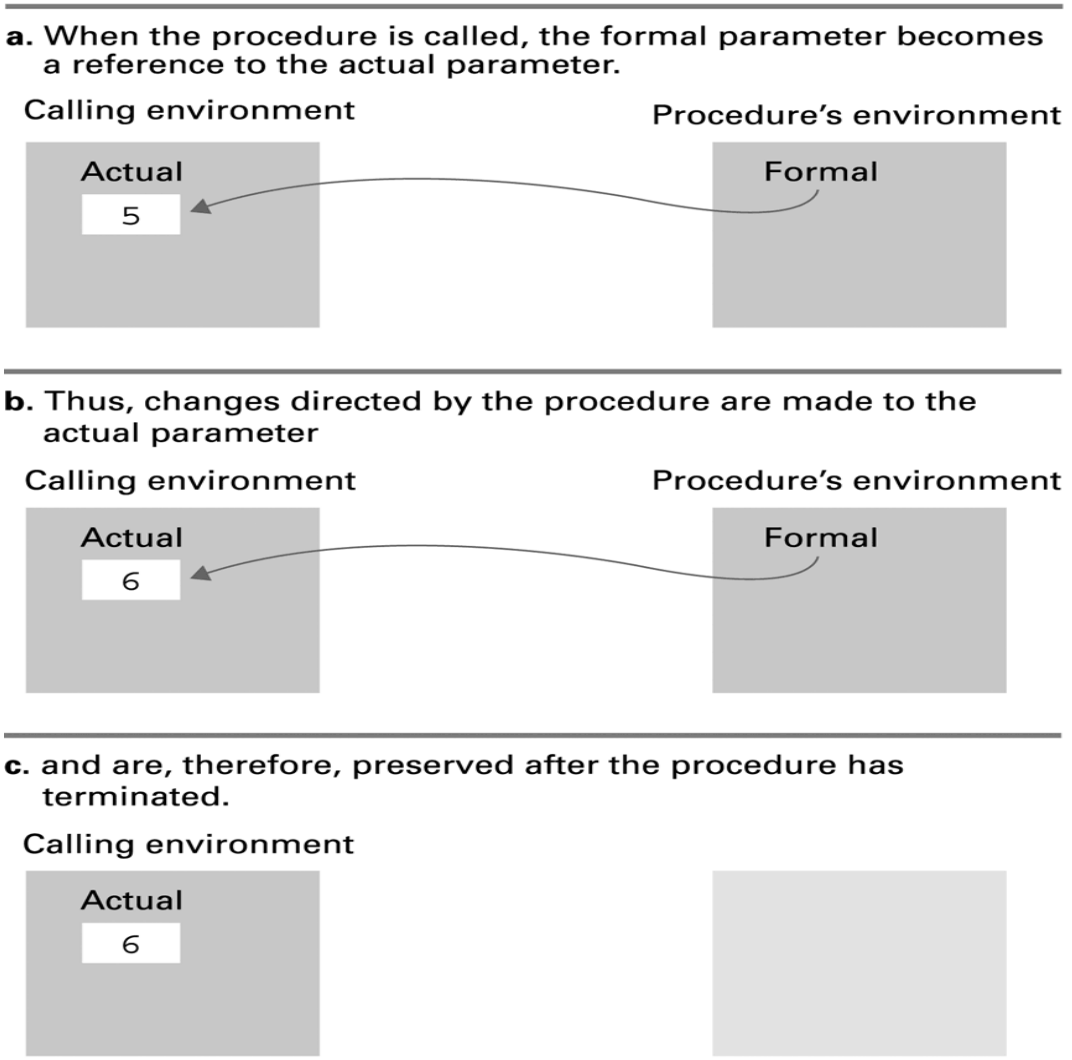
```
...
int Attuale;
Attuale = 5;
Demo(Attuale);
print Attuale
// stampa 5
```



Passaggio di parametri per riferimento

```
procedure Demo(int Formale)
    Formale = Formale + 1;
```

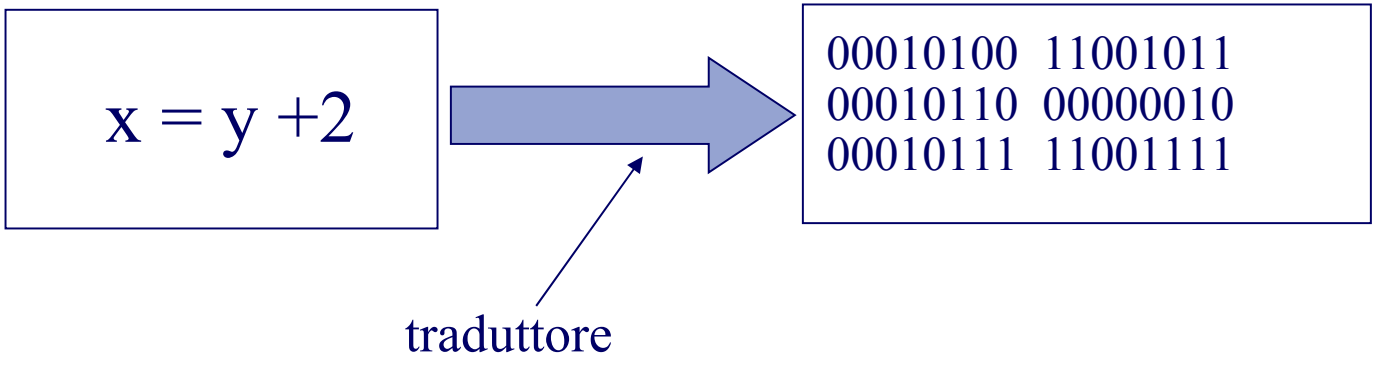
```
...
int Attuale;
Attuale = 5;
Demo(Attuale);
print Attuale
// stampa 6
```

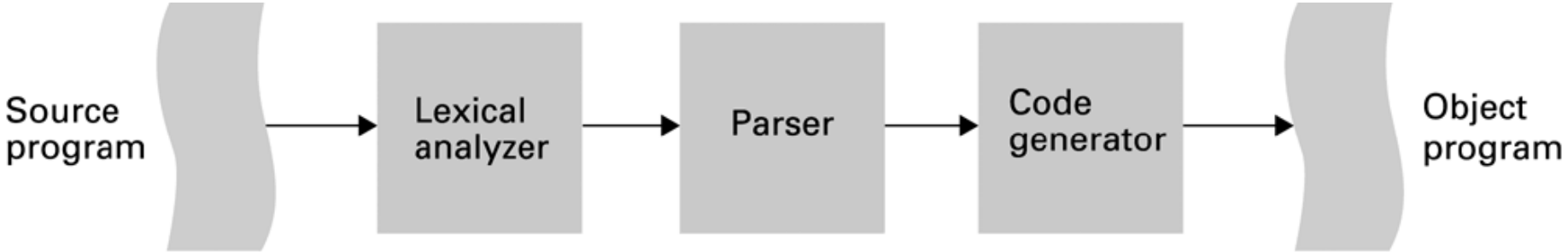


L'implementazione di un linguaggio: il processo di traduzione

codice sorgente (scritto in qualche linguaggio di alto livello)

codice oggetto (in linguaggio macchina, eseguibile





Due tipi di traduttori

- **Interpreti**
  - traducono ed eseguono direttamente ciascuna istruzione del programma sorgente, istruzione per istruzione, ottenendo il risultato finale

- **Compilatori**
  - accettano in input l'intero programma e producono in output la rappresentazione dell'intero programma in linguaggio macchina

L'analisi lessicale

La prima fase della compilazione

- elimina dal programma tutti i commenti
- individua nel programma i ***token***
- passa la sequenza di token ottenuta al ***parser***
- **Esempio:** dal programma...

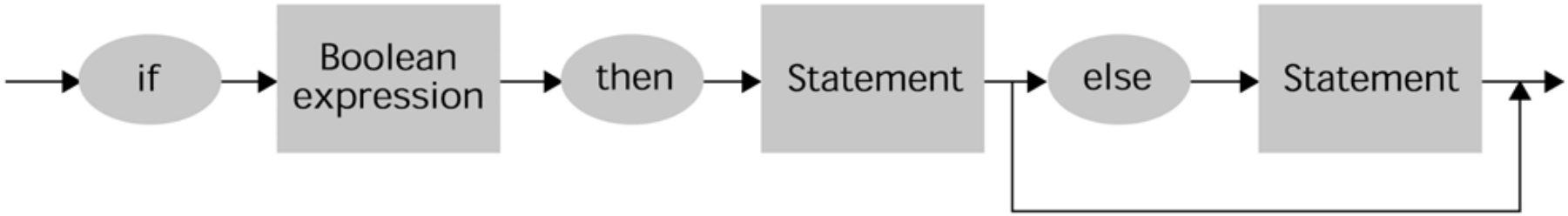
```
// la seguente istruzione calcola il valore di una
    certa espressione aritmetica
x := anno * 2 + base * (x * 3)
```

alla sequenza di token...

```
<id>x assign <id>anno mult <int>2 plus
<id>base mult lpar <id>x mult <int>3 rpar
```

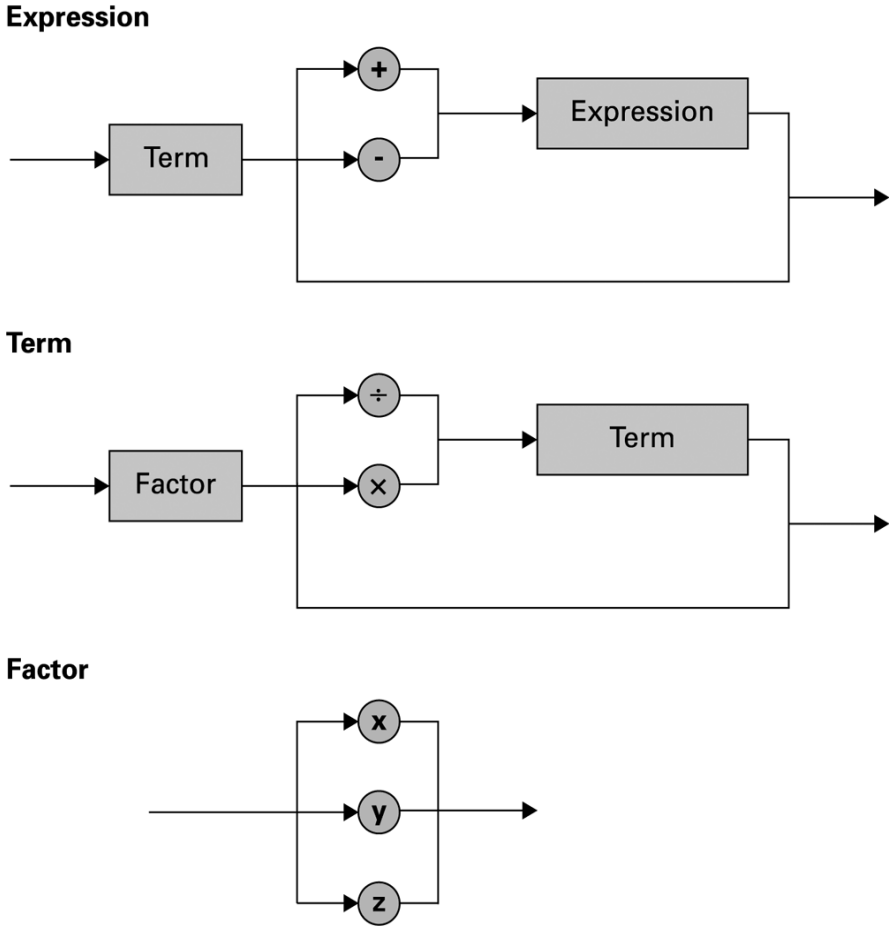
Il parser (analizzatore sintattico)

- Il parser controlla che la sequenza di token appartenga al linguaggio di programmazione, e costruisce l'**albero sintattico** del programma, usando le regole del linguaggio.
- Esempio di regola per il costrutto **if-then-else** descritta come **diagramma sintattico**.

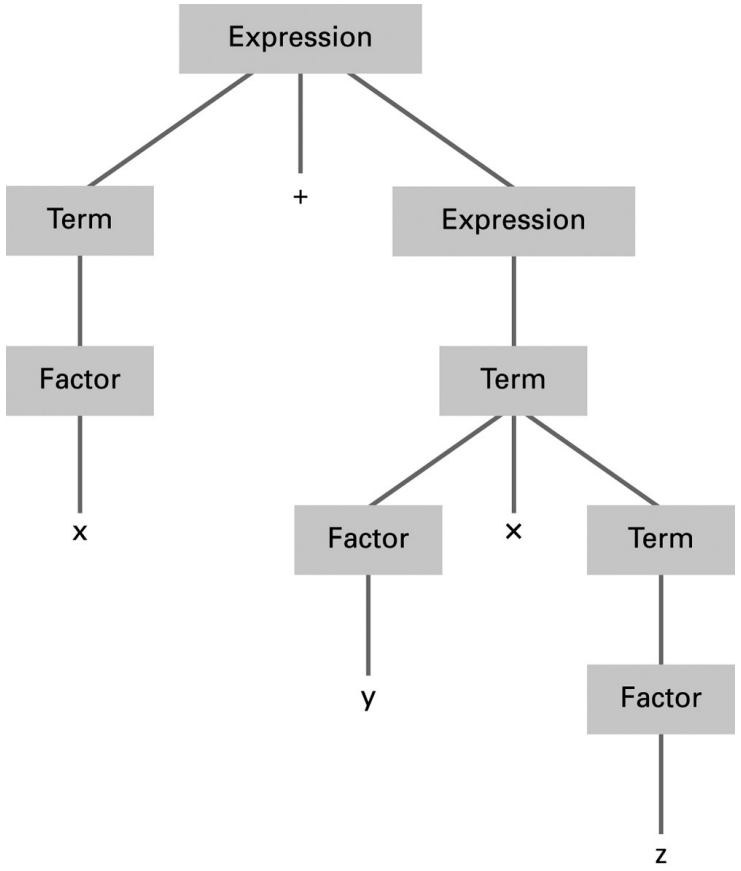


Esempio di costruzione di albero sintattico

Diagrammi sintattici per espressioni

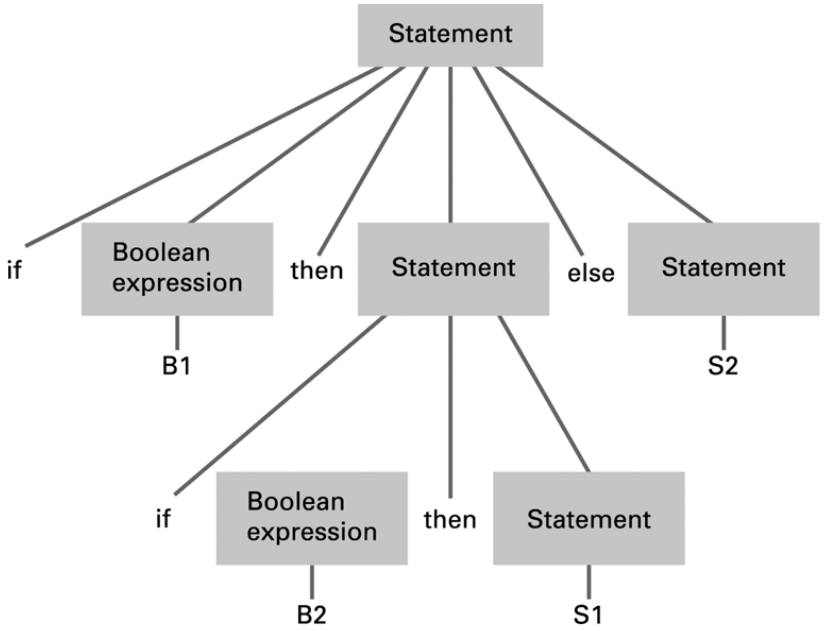


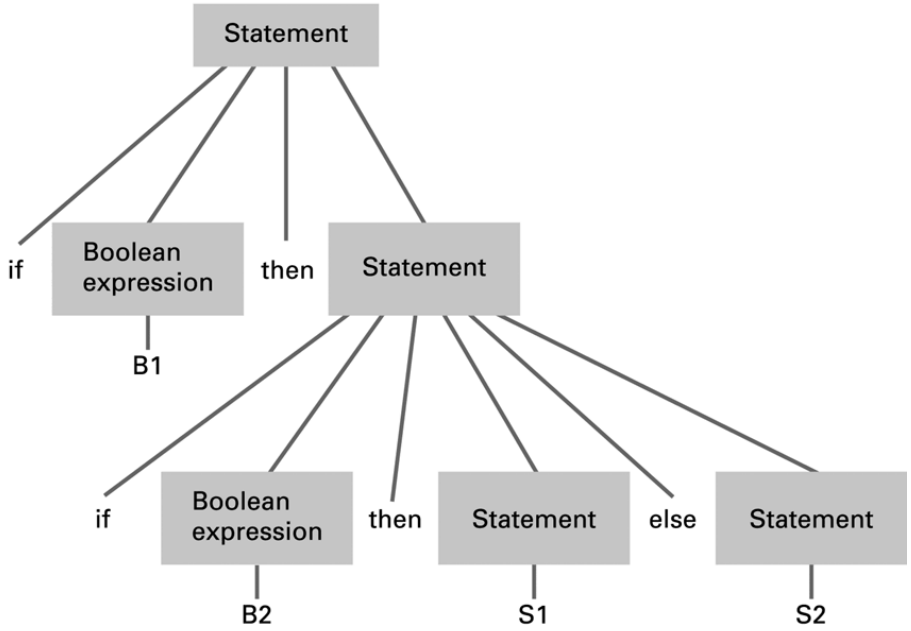
Albero sintattico di x+y*z



Ambiguità:
due diversi alberi di derivazione per l'istruzione

```
if B1
then if B2
     then S1
else S2
```





Controllo dei tipi

- Il **parser** genera una **tabella dei simboli** a partire dalle dichiarazioni del programma, associando a ogni variabile il suo tipo
- Controlla anche che i tipi siano usati in modo corretto negli assegnamenti, nelle invocazioni di procedure, nelle espressioni

Generazione del codice e ottimizzazione

- Dall'albero sintattico, con l'aiuto della tabella dei simboli, viene generato il codice oggetto (es: in linguaggio macchina).
- Il codice oggetto viene **ottimizzato** per garantire migliore efficienza.
- Esempio: compilando

```
x = y + z;
w = x + z;
```

si può sfruttare il fatto che **`x`** e **`y`** sono già in un registro.