Linguaggi di Programmazione I – Lezione 1

Prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

4 marzo 2008

Panoramica della lezione

Introduzione al corso

Concetti generali

Data Object e legami

Blocchi di istruzioni

Legami di nome

Legami di locazione

Bibliografia

Introduzione al corso 3 / 43

Obbiettivi

- Capacità di riflessione sulle differenze tra i vari paradigmi di programmazione ed in particolare dei paradigmi *imperativo* e *ad oggetti*.
- Capacità media di progettare *ad oggetti*.
- Capacità media di programmare in Java.

Scheduling

- Parte prima (concetti generali) – 8 lezioni
  1. Paradigmi dei linguaggi di programmazione.
  2. Il modello imperativo.
  3. Il modello ad oggetti.
  4. Progettazione orientata ad oggetti e UML.
  5. Prima prova scritta intercorso.
- Parte seconda (Java) – 16 lezioni
  1. Studio dei costrutti fondamentali: identificatori, parole chiavi, tipi; espressioni e controllo di flusso; array; ereditarietà; overloading e overriding; polimorfismo.
  2. Seconda prova scritta intercorso.
  3. Studio dei costrutti più avanzati: qualificatori di classi, metodi, attributi; classi astratte; interfacce; classi interne; gestione degli errori: eccezioni; programmi *text-based*; threads; networking.
  4. Prova scritta finale.

Esame

L'esame consisterà in:

1. una prova scritta, strutturata, a discrezione della commissione, in forma di test chiusi a risposta multipla o in forma di test a risposta aperta o in forma mista;
2. una prova orale, consistente prevalentemente nella discussione della prova scritta e in eventuali integrazioni a discrezione delle commissione; gli studenti che alla prova scritta avranno riportato valutazioni insufficienti saranno sconsigliati dal sostenere la prova orale.

Sono previste due prove scritte intermedie, con la stessa strutturazione dell'esame finale, il cui superamento consentirà di optare, in tutti e soli gli appelli del corrente anno accademico, per sostenere un esame scritto ridotto, riguardante la sola parte finale del programma. In questo caso la valutazione finale terrà conto delle valutazioni delle prove intermedie.

Altre informazioni

- *Libri di testo*:
  - Dershem - Jipping. *Programming languages: structures and models*.
  - Wampler. *The essence on object oriented programming with Java and UML*.
  - Fowler. *UML distilled*.
  - Eckel. *Thinking in Java*.
- *Materiali*: http://sette.dnsalias.org
- *Orari delle lezioni*:
  Martedì: 11:00 - 13:00 Aula C6
  Giovedì: 11:00 - 13:00 Aula B5
- *Ricevimento studenti*: interno 2M14; orario da concordare per appuntamento: 081676814 oppure mailto://marcello.sette@gmail.com.

Concetti generali

Terminologia

- **Linguaggio di programmazione**: è un linguaggio che è usato per esprimere (mediante un programma) un processo con il quale un processore può risolvere un problema.
- **Processore**: è la macchina che eseguirà il processo descritto dal programma; non si deve intendere come un singolo oggetto, ma come una *architettura di elaborazione*.
- **Programma**: è l'espressione codificata di un processo.

Breve storia

1954	FORTRAN (FORmula TRANslation)
1960	COBOL (Common Business Oriented Language) ALGOL 60 (Algorithmic Oriented Language) PL/1 (Programming Language 1) Simula 67 ALGOL 68 PASCAL LISP (LISt Processing) APL BASIC
1970/80	PROLOG SMALLTALK C MODULA/2 ADA

Paradigmi

Imperativo: Esecuzione sequenziale basata su modifiche da apportare a memoria.

Logico: Programma come descrizione logica di un problema. Esecuzione analoga a processi di dimostrazione di teoremi.

Funzionale: Programma e ogni sua componente sono funzioni. Esecuzione come valutazione di funzioni.

Orientato ad oggetti: Programma costituito da oggetti che scambiano messaggi.

Parallelo: Programmi che descrivono entità distribuite che sono eseguite contemporaneamente ed in modo asincrono.

Macchina astratta

È quella architettura di elaborazione il cui linguaggio di programmazione:

- ha la stessa potenza del linguaggio F della macchina fisica;
- è costituito da un sottoinsieme minimale di costrutti di F.

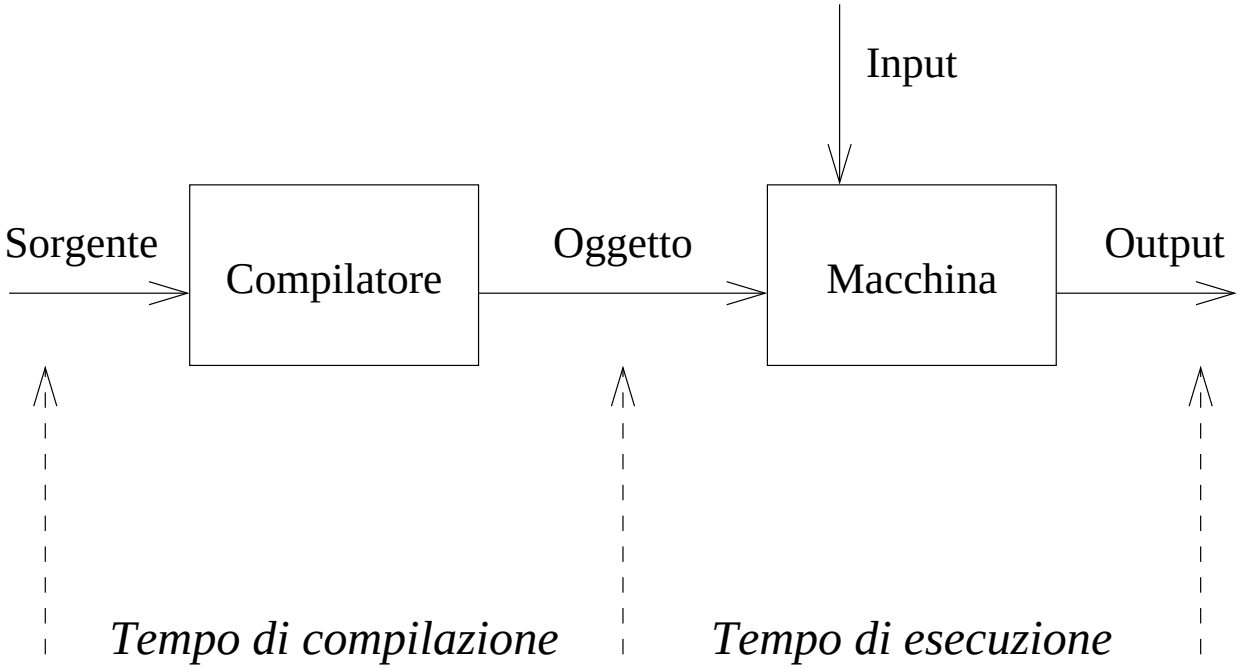
Proprietà:

1. essa è la rappresentazione comune a più macchine fisiche;
2. i suoi programmi sono in genere più "prolissi" degli equivalenti di ogni macchina fisica rappresentata.

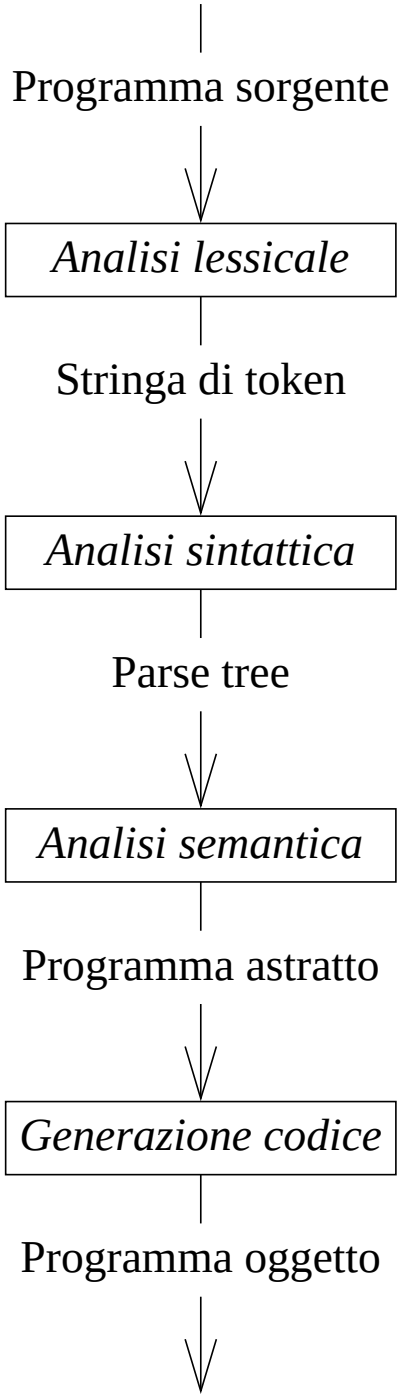
Traduttori

- **Interpreti**: traducono ed eseguono un costrutto alla volta.
  PRO: debug - fase di sviluppo.
- **Compilatori**: prima traducono l'intero programma; poi la traduzione può essere eseguita (anche più volte).
  PRO: velocità di esecuzione finale - fase di rilascio.

Compilazione ed esecuzione



Compilatore



Proprietà dei linguaggi

- Semplicità – (concisione) VS (leggibilità)
  - Semantica: minimo numero di concetti e strutture.
  - Sintattica: unica rappresentabilità di ogni concetto.
- Astrazione – (rappresentare solo attributi essenziali)
  - Dati: nascondere i dettagli di oggetti.
  - Procedure: facilitare la modularità del progetto.
- Espressività – (facilità di rappresentazione di oggetti) VS (semplicità)
- Ortogonalità – (meno eccezioni alle regole del linguaggio)
- Portabilità

Criteri di scelta del linguaggio

- Disponibilità dei traduttori
- Maggiore conoscenza da parte del programmatore
- Esistenza di standard di portabilità
- Sintassi aderente al problema
- Semantica aderente alla architettura fisica
- Comodità dell'ambiente di programmazione

Modello imperativo

- Simula le azioni dell'elaboratore a livello di linguaggio macchina.
- Ogni unità di esecuzione consiste di 4 passi:

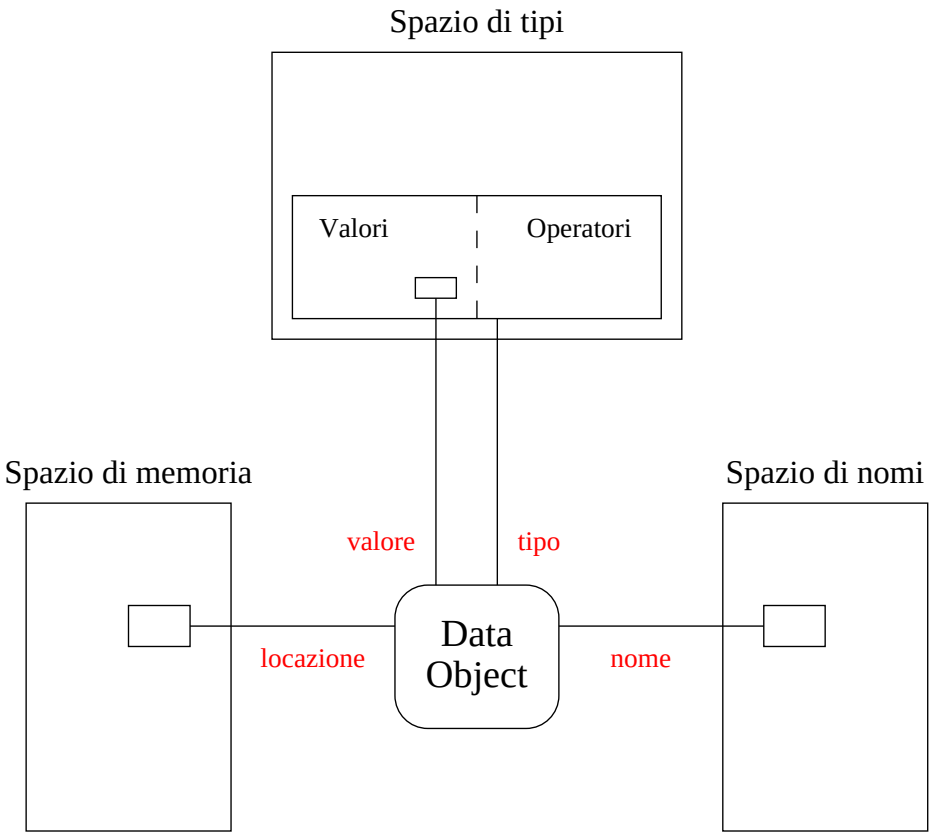
  1. ottenere indirizzi delle locazioni di operandi e risultato;
  2. ottenere dati di operandi da locazioni di operandi;
  3. valutare risultato;
  4. memorizzare risultato in locazione risultato.

- Si caratterizza per l'uso dei nomi come astrazione di indirizzi di locazioni di memoria.

Data Object

- Un *data object* è la quadrupla (L, N, V, T), ove:
  - L: locazione.
  - N: nome.
  - V: valore.
  - T: tipo.
- Un *legame* è la determinazione di una delle componenti.

Legami



Modifiche di legami

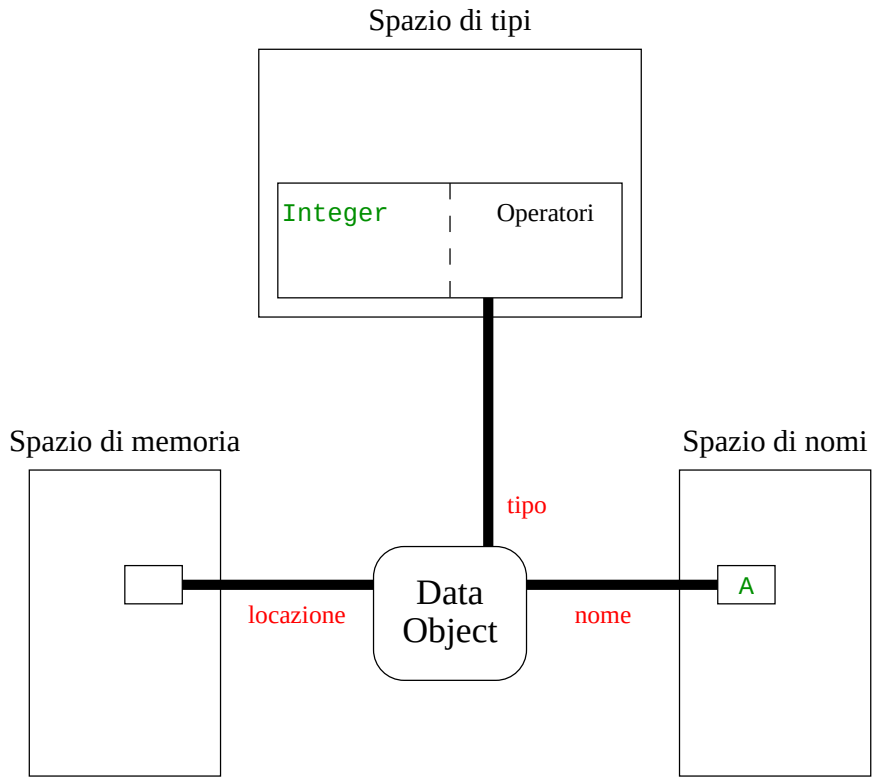
Variazioni di legami possono avvenire:

1. Durante la compilazione (compile time).
2. Durante il caricamento in memoria (load time).
3. Durante l'esecuzione (run time).

Possiamo notare che:

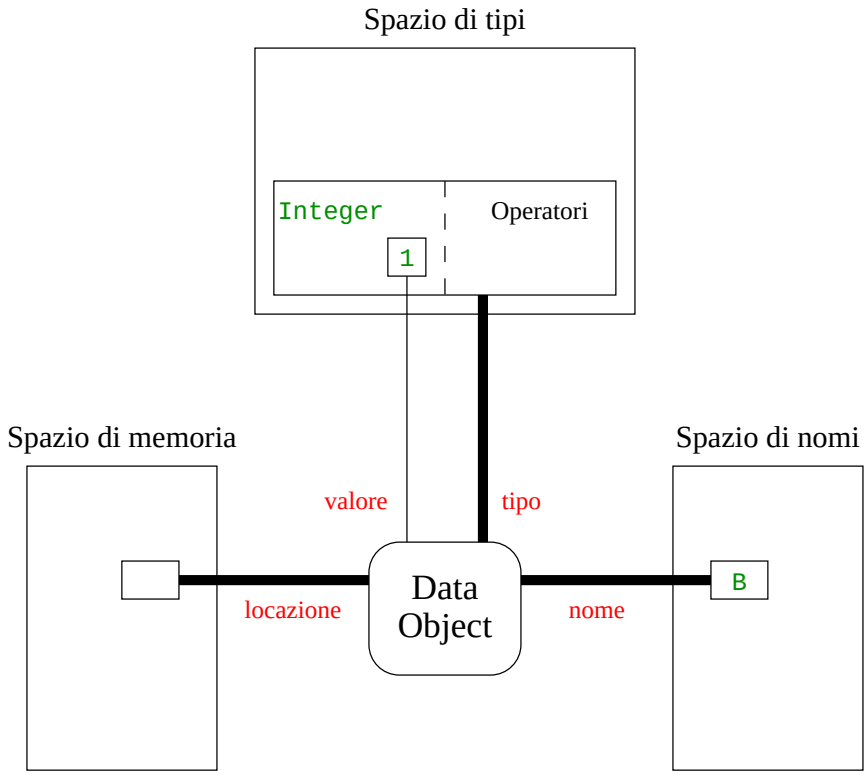
- il *location binding* avviene durante il caricamento in memoria;
- il *name binding* avviene durante la compilazione, nell'istante in cui il compilatore incontra una dichiarazione;
- il *type binding* avviene durante la compilazione, nell'istante in cui il compilatore incontra una dichiarazione di tipo; un tipo è definito dal sottospazio di valori (e dai relativi operatori) che un *data object* può assumere.

Esempio 1



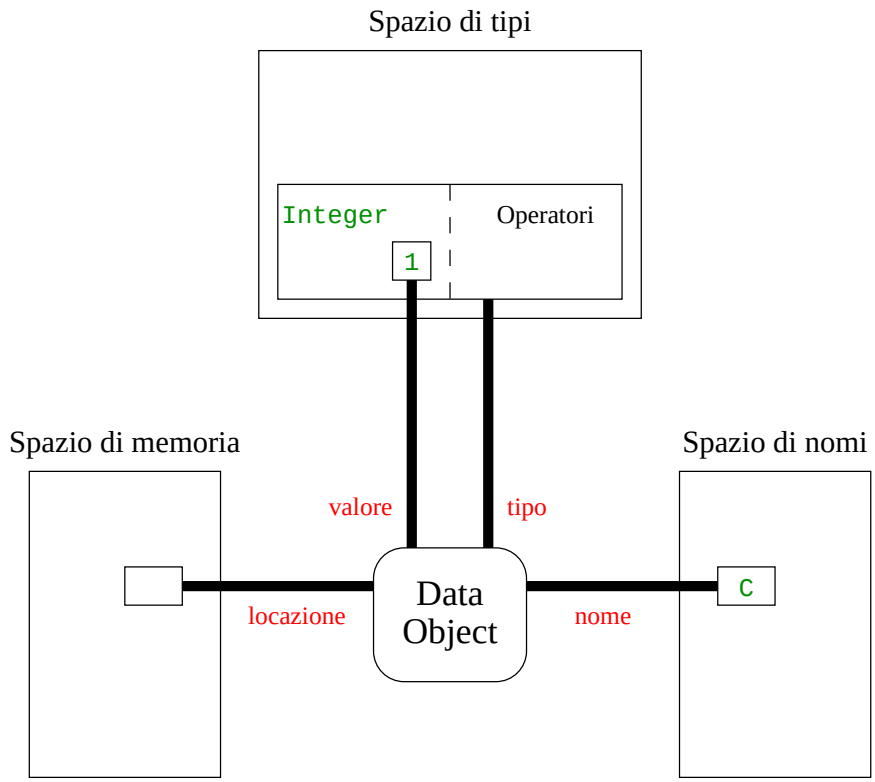

```
A: integer;
```

Esempio 2




```
B: integer:= 1;
```

Esempio 3

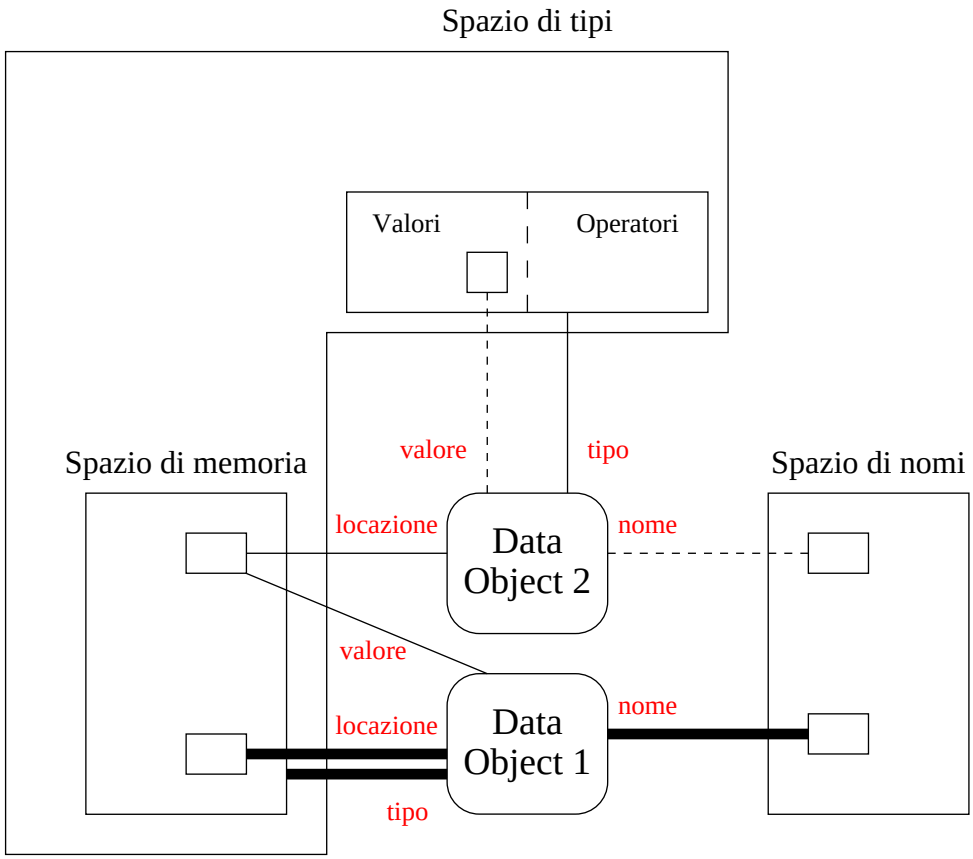



```
C: constant integer:= 1;
```


Tipi

- Legame solitamente instaurato e fisso durante la compilazione, tranne per linguaggi che realizzano *dynamic binding*. In essi il legame di tipo dipende dal valore del dato.
- **Type checking** è il meccanismo di controllo di consistenza della coppia valore-tipo.
  - Può avvenire: a) durante la compilazione, b) durante l'esecuzione, c) per nulla.
  - Un linguaggio è *fortemente tipizzato* se il controllo di tipo avviene il più possibile durante la compilazione e, negli altri casi, durante l'esecuzione.
  - Un linguaggio è *dinamicamente tipizzato* se il legame e di conseguenza anche il controllo di consistenza avvengono durante l'esecuzione.
  - Un linguaggio è *staticamente tipizzato* se il legame avviene durante la compilazione; in questo caso il controllo di consistenza può avvenire in entrambe le fasi.

Il puntatore



LP1 – Lezione 1 27 / 43

Blocchi di istruzioni 28 / 43

Necessità

Istruzioni raggruppate in blocchi per meglio definire:

- ambito delle strutture di controllo;
- ambito di una procedura;
- unità di compilazione separata;
- ambito dei legami di nome.

LP1 – Lezione 1 29 / 43

Definizioni

Blocchi che definiscono l'ambito di validità di un nome contengono due parti:

- una sezione di dichiarazione del nome;
- una sezione che comprende gli enunciati sui quali ha validità il legame.

Definiamo un piccolo pseudo-linguaggio per rappresentare un blocco:

```
...
BLOCK A;
  DECLARE I;
BEGIN A
...               {I DEL BLOCCO A}
END A;
...
```


Ereditarietà di legami

Essenzialmente due tipi di ereditarietà:

In ambito statico o lessicale.
Blocchi annidati ereditano i legami dai blocchi più esterni (*legami non locali*) e, di solito, possono aggiungere legami locali o sovrapporne di nuovi.

In ambito dinamico
Concetto qui esaminato solo in relazione ai blocchi annidati, ma che assume il proprio senso maggiore quando vi sono procedure chiamanti e chiamate. In questo caso i legami possono anche essere ereditati a run time dalla procedura chiamante.
Esamineremo tutti i dettagli nella prossima lezione.

Ereditarietà statica

```
PROGRAM P;
  DECLARE X;
BEGIN P
  ...               {X da P}
  BLOCK A;
    DECLARE Y;
  BEGIN A
    ...             {X da P, Y da A}
    BLOCK B;
      DECLARE Z;
    BEGIN B
      ...           {X da P, Y da A, Z da B}
    END B;
    ...             {X da P, Y da A}
  END A;
  ...               {X da P}
  BLOCK C;
    DECLARE Z;
  START C
    ...             {X da P, Z da C}
  END C;
  ...               {X da P}
END P;
```


Mascheramento

```
PROGRAM P;
  DECLARE X,Y;
BEGIN P
  ...               {X e Y da P}
  BLOCK A;
    DECLARE X,Z;
  BEGIN A
    ...             {X e Z da A, Y da P}
  END A;
  ...               {X e Y da P}
END P;
```

Allocazione statica di memoria

Si dice *allocazione statica di memoria* (compile-time) quando le variabili conservano il loro valore ogni volta che si rientra in un blocco (il legame di locazione è fissato e costante). Se il linguaggio prevede ciò, allora:

```
PROGRAM P;
  DECLARE I;
BEGIN P
  FOR I:=1 TO 10 DO
    BLOCK A;
      DECLARE J;
    BEGIN A
      IF I=1 THEN
        J:=1;              {I da P, J da A}
      ELSE
        J:=J*I;
      END IF
    END A;
END P;
```


Allocazione dinamica di memoria

- Si dice *allocazione dinamica di memoria* (run-time) quando il legame di locazione (e anche di nome) è creato all'inizio dell'esecuzione di un blocco e viene rilasciato a fine esecuzione.
- Essa è realizzata attraverso il *record di attivazione* di un blocco.
  - Un record di attivazione contiene tutte le informazioni sull'esecuzione del blocco necessarie per riprendere l'esecuzione dopo che essa è stata sospesa.
  - Può contenere informazioni complesse (come si vedrà in seguito), ma per realizzare un legame dinamico di locazione in blocchi annidati è sufficiente che contenga le locazioni dei dati locali più un puntatore al record di attivazione del blocco immediatamente più esterno.

Stack di attivazione

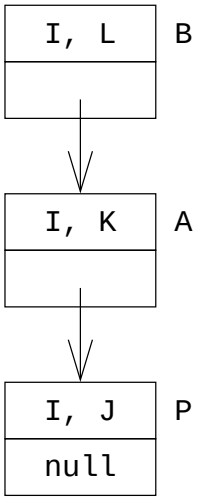
In ogni momento dell'esecuzione lo *stack di attivazione* contiene i record "attivi":

1. il top dello stack contiene sempre il record del blocco correntemente in esecuzione;
2. ogni volta che si entra in un blocco, il record di attivazione del blocco viene posto sullo stack (push);
3. ogni volta che si esce da un blocco, viene eliminato il record al top dello stack (pop).

Esempio di allocazione dinamica

All'ingresso in B

Contenuto dello stack di attivazione




```
PROGRAM P;
  DECLARE I,J;
BEGIN P
  BLOCK A;
    DECLARE I,K;
  BEGIN A
    BLOCK B;
      DECLARE I,L;
    BEGIN B
      ...          {I e L da B, K da A, J da P}
    END B;
    ...            {I e K da A, J da P}
  END A;
  BLOCK C;
    DECLARE I,N;
  BEGIN C
    ...            {I e N da C, J da P}
  END C;
...                {I e J da P}
END P;
```


Bibliografia

Bibliografia

- Dershem, Jipping. *Programming Languages: Structures and Models*. Cap. 1 e 2; cap. 3, par. 3.1.1 - 3.1.5, 3.1.12, 3.2.

Esercizio1

Si considerino i seguenti blocchi annidati. Per ciascun blocco, determinare i legami di ogni variabile. Evidenziare le variabili locali e globali.

```
PROGRAM P;
  BLOCK B1;
    DECLARE A,B,C;
  BEGIN B1
    BLOCK B2;
      DECLARE C,D;
    BEGIN B2
      BLOCK B3;
        DECLARE B,D,F;
      BEGIN
        ...
      END B3;
      ...
    END B2;
    BLOCK B4;
      DECLARE B,C,D;
    BEGIN B4
      ...
    END B4;
    ...
  END B1;
END P;
```


Esercizio 2

Tracciare il contenuto dello stack di attivazione durante l'esecuzione del seguente pseudo-programma.

```
PROGRAM P;
  DECLARE X,Y;
BEGIN P
  BLOCK A;
    DECLARE X,Y,Z;
  BEGIN A
    BLOCK B;
      DECLARE Y;
    BEGIN B
      BLOCK C;
        DECLARE X,Y;
      BEGIN C
        ...
      END C;
      BLOCK D;
        DECLARE Z;
      BEGIN D
        ...
      END D;
      ...
    END B;
  END A;
  BLOCK E;
    DECLARE Z;
  BEGIN E
    BLOCK F;
      DECLARE X;
    BEGIN F
      ...
    END F;
    ...
  END E;
  ...
END P;
```