UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

02 - Memory Management

INGEGNERIA INFORMATICA

Programmazione Avanzata

Anno di corso: 1

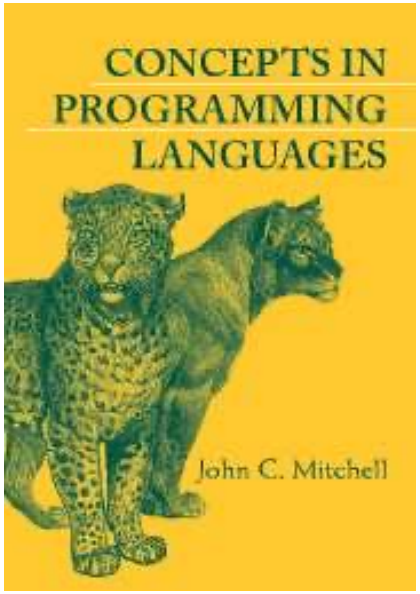
Anno accademico di offerta: 2023/2024

Crediti: 6

Prof. Claudio MENGHI

Dalmine

24 Settembre 2024



Capitolo 7 - Scope, Functions, and Storage Management

Introduzione

- Quando dichiariamo una variabile, il computer dove la memorizza?
- Quali sono le regole per accedere ad una variabile?
- Come vengono passate ai sottoprogrammi i dati?
- Principali feature:
  - Divisione di un programma in sottoprogrammi
    - Non come il BASIC
      - Non unica sequenza di istruzioni (con GOTO)
    - Non si sanno tutte le variabili prima dell'esecuzione e l'allocazione della memoria avviene dinamicamente
      - Non costringo al programmatore di dichiarare tutte le varibili fin dall'inizio
  - Uso della ricorsione

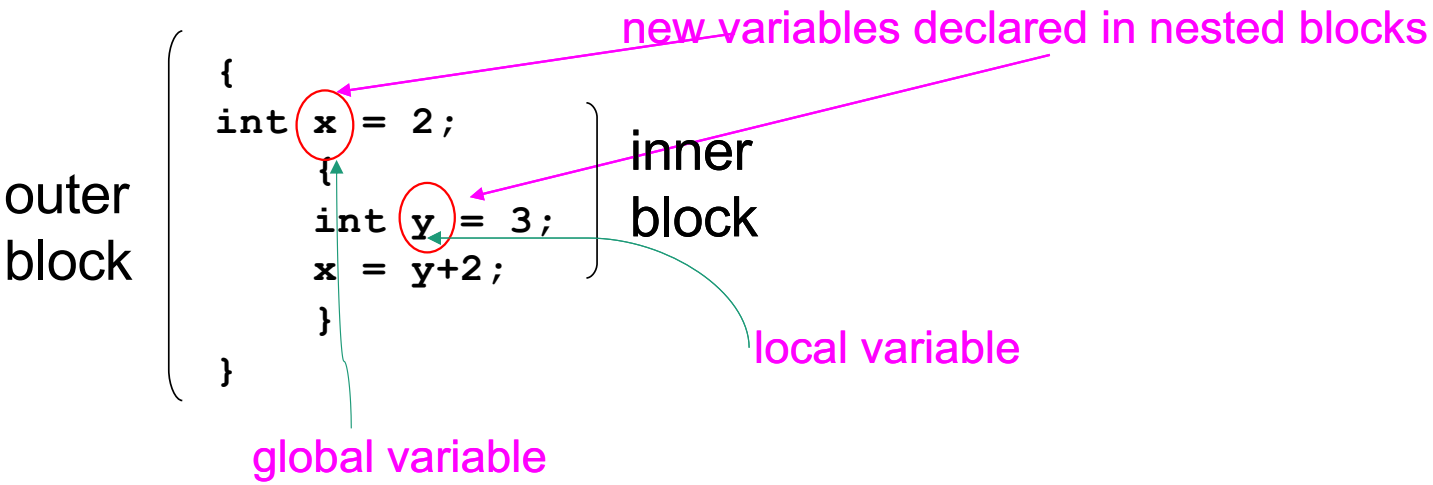
Scope, Functions, and Storage Management

- Scope: permette l’associazione di aree di programma ad aree di memoria
- Function calls: richiedono una nuova area di memoria dove salvare parametri e variabili
- Tail recursions (o tail calls)

Scope, Functions, and Storage Management

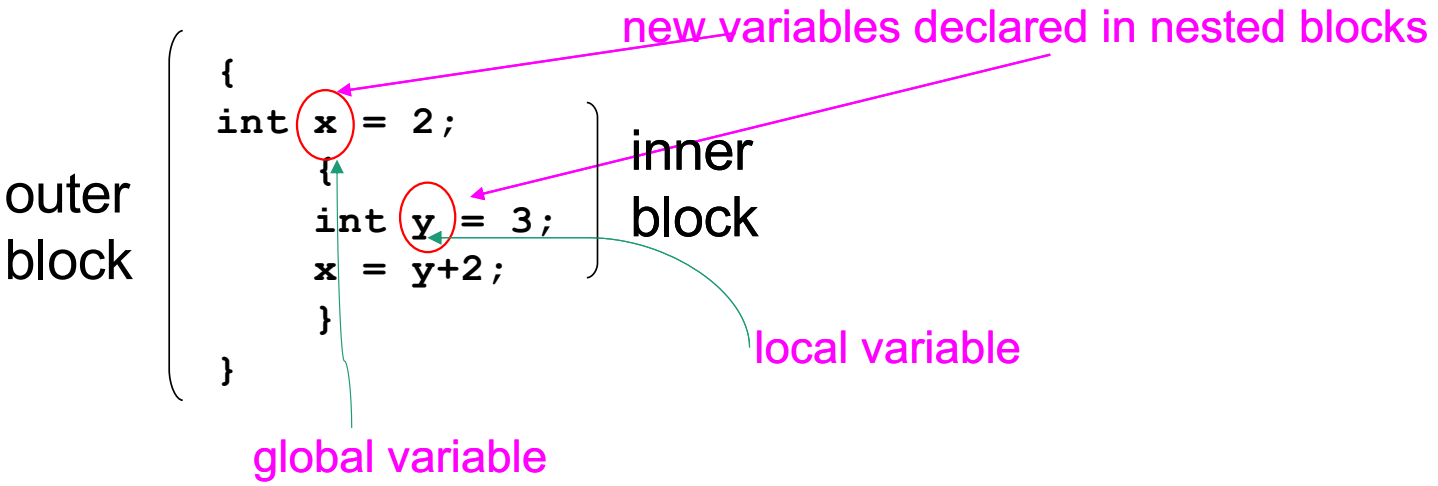
- Block-structured languages and stack storage
- In-line Blocks
  - activation records
  - storage for local, global variables
- First-order functions
  - parameter passing
  - tail recursion and iteration
- NO - Higher-order functions
  - deviations from stack discipline
  - language expressiveness => implementation complexity

Block-Structured Languages



Block-Structured Languages

- Storage management
  - Enter block: allocate space for variables
  - Exits block: some or all space may be deallocated
- Nested blocks, local variables



Examples

- Blocks in common languages
  - C/c++/Java { … }
  - Algol begin … end
  - ML let … in … end
- Two forms of blocks
  - In-line blocks
    - Blocks for control structure like if, for and so on.. similar to block inline
  - Blocks associated with functions or procedures

- Topic: block-based memory management, access to *local variables*, *parameters*, *global vars*
- It allows **recursive functions**

Alcune note

Alcuni linguaggi (come Fortran) allocavano in modo fisso le variabili

- Svantaggi …
  - Ricorsione?

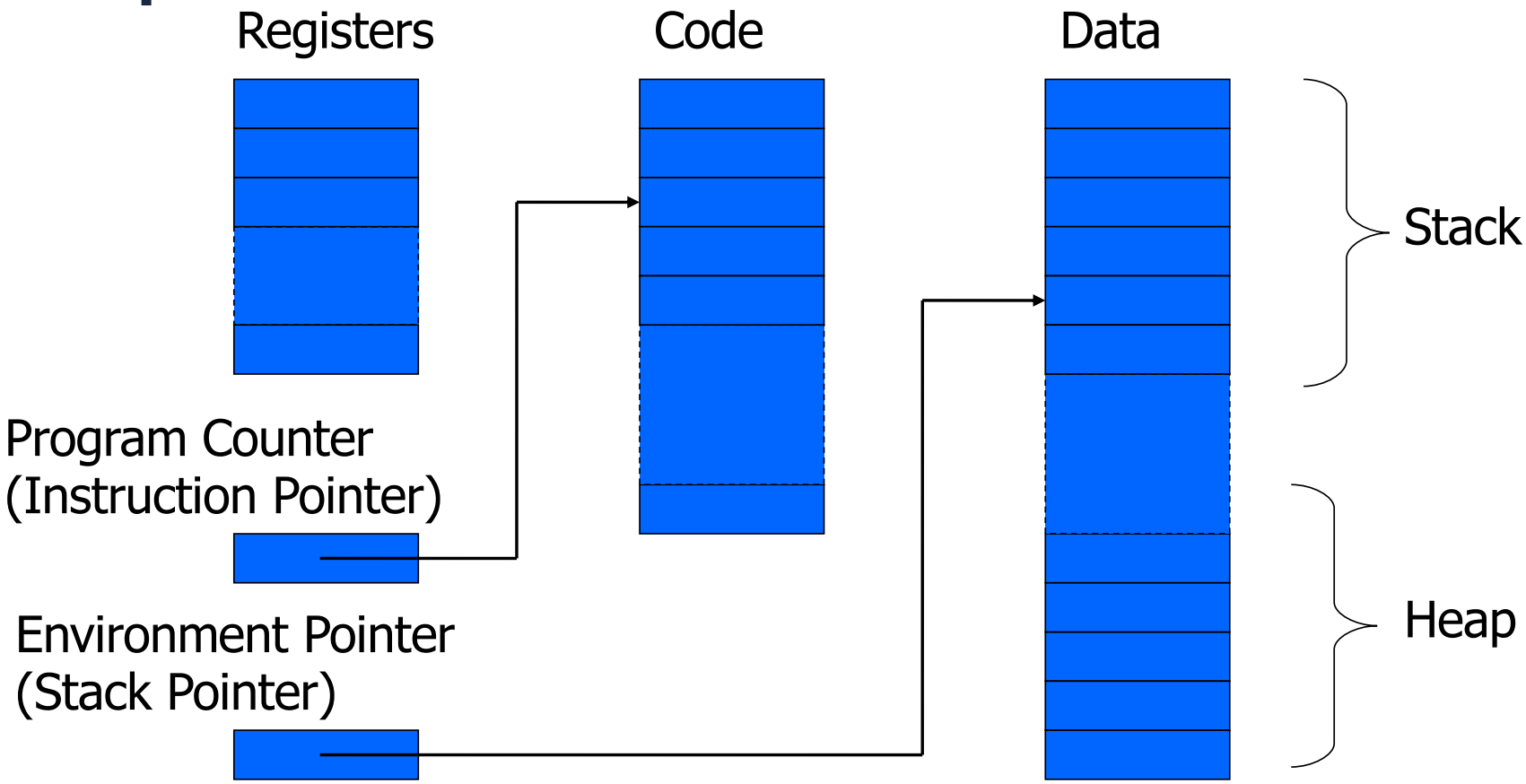
Block-structured languages:

- New variables may be declared at various points in a program
- Each declaration is visible within a block
- When a program begins executing the instructions contained in a block, the memory is allocated
- When a program exits, the memory is freed
- An identifier that is not declared in the current block is considered global to the block

Alcune note

c e c++ non consentono la dichiarazione di funzioni locali all'interno di blocchi innestati.

Simplified Machine Model



Interested in Memory Mgmt Only

- Registers, Code segment, Program counter
  - Ignore registers
  - Details of instruction set will not matter
- Data Segment
  - Stack contains data related to block entry/exit
  - Heap contains data of varying lifetime
  - **Environment pointer (Stack Pointer)** points to current stack position
    - Block entry: add new activation record (**Stack Record**) to stack
    - Block exit: remove most recent activation record

In-line Blocks

- Sono blocchi che non sono il corpo di una funzione o di una procedura

```
{ int x=0;
   int y=x+1;
        {  int z=(x+y)*(x-y);
         }
}
```

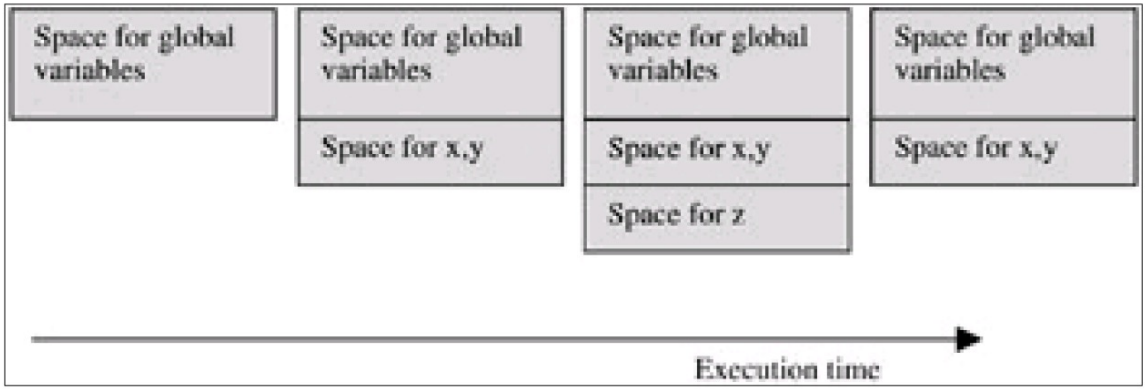
In-line Blocks

- Activation record
  - Data structure stored on run-time stack
  - Contains space for local variables (if any)
- Example

```
{ int x=0;
   int y=x+1;
        {  int z=(x+y)*(x-y);
         }
}
```

Push record with space for x, y
Set values of x, y
    Push record for inner block
    Set value of z
    Pop record for inner block
Pop record for outer block

In-line Blocks




```
{ int x=0;
   int y=x+1;
        {  int z=(x+y)*(x-y);
         }
}
```

Intermediate results on the stack

May need space for variables and intermediate results like (x+y), (x-y)

Example:

int z = (x+y) *(x-y)

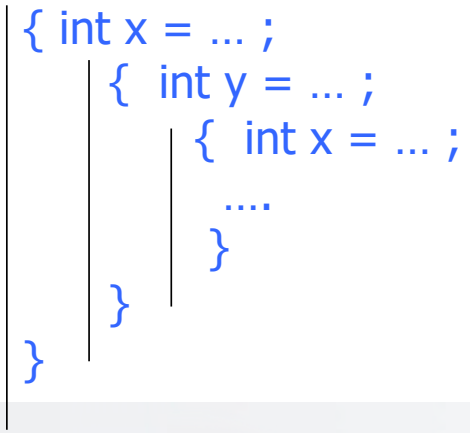
push x+y

push x-y

a1= pop a2 =pop push a1*a2

Some basic concepts

- Scope
  - Region of program text where declaration is visible
- Lifetime
  - Period of time when location is allocated to program



- Inner declaration of x hides outer one.
- Called "hole in scope"
- Lifetime of outer x includes time when inner block is executed
- Lifetime ≠ scope
- Lines indicate "contour model" of scope.

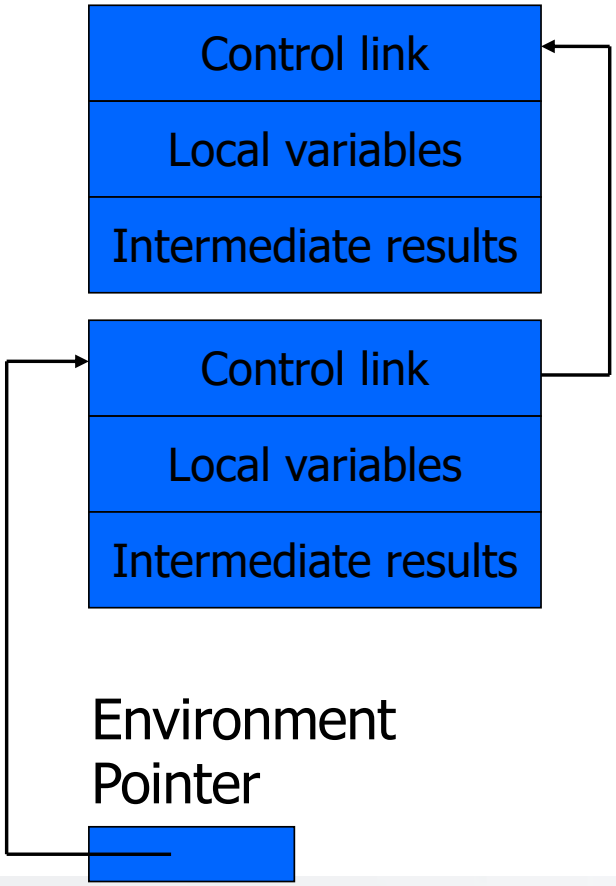
Control Link/Link register

- Environmant Pointer (EP) punta alla cima del record di attivazione corrente
- Record di attivazione ha dimensione variabile

- Come faccio a ripristinare EP quando faccio il pop del record di attivazione che non serve più?

- Uso il control link:
  - Puntatore alla cima del record di attivazione precedente
  - Viene salvato quando creo il record di attivazione
  - Viene ripristinato quando faccio il pop

Activation record for in-line block

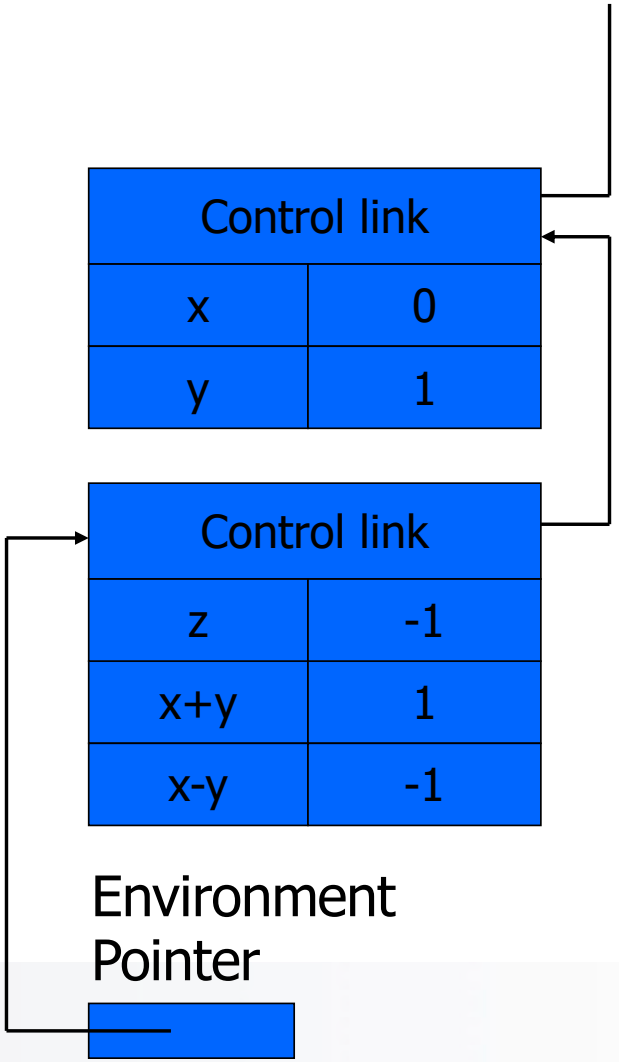


- Control link
  - pointer to previous record on stack
- Push record on stack:
  - Set new control link to point to old env ptr
  - Set env ptr to new record
- Pop record off stack
  - Follow control link of current record to reset environment pointer

Example

```
{ int x=0;
   int y=x+1;
        {
              int z=(x+y)*(x-y);
        }
}
```

Push record with space for x, y (set control link = old env pointer, set env pointer)
Set values of x, y
    Push record for inner block
    Set value of z
    Pop record for inner block (set env pointer to control link)
Pop record for outer block



Scoping rules

- Global and local variables
  - x, y are local to outer block
  - z is local to inner bock
  - x, y are global to inner block

```
{ int x=0;
  int y=x+1;
      {  int z=(x+y)*(x-y);
       }
}
```

- Static scope
  - global refers to declaration in closest enclosing block
- Dynamic scope
  - global refers to most recent activation record

These are same until we consider function calls.

UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

Domande?