POLITECNICO DI MILANO

3. Semantica dei Linguaggi di Programmazione

Informatica 3

Andrea Mocci, mocci@elet.polimi.it

Esercizio 1: EBNF

- Utilizzando il linguaggio descritto dalla seguente EBNF:

```
<program>    ::= { <statement>* }
<statement>  ::= <assignment> | <loop>
<assignment>::= <identifier> = <expr>;
<loop>  ::= while <expr> { <statement>+ }
<expr>  ::= <identifier> | <number> | ( <expr> ) |
   <expr> <op> <expr> | !<expr>
<op>  ::= == | <= | >= | != | && | ||
```

- dove <identifier> e <number> rappresentano rispettivamente gli identificatori C e i numeri interi
- scrivere una funzione in grado di predire se la radice di un numero è reale o immaginaria

Problema

- La soluzione immediata sarebbe molto semplice:

```
IF x>=0 THEN "REALE"
ELSE "IMMAGINARIO"
```

- Ma il costrutto if non appartiene ai costrutti permessi

Soluzione


```
aux=0;
while ((x>=0)&&(aux==0)){
 // in questo caso la radice e` reale
 aux=1;
}
while ((x<0)&&(aux==0)) {
 // in questo caso la radice e` immaginaria
 aux=1;
}
```

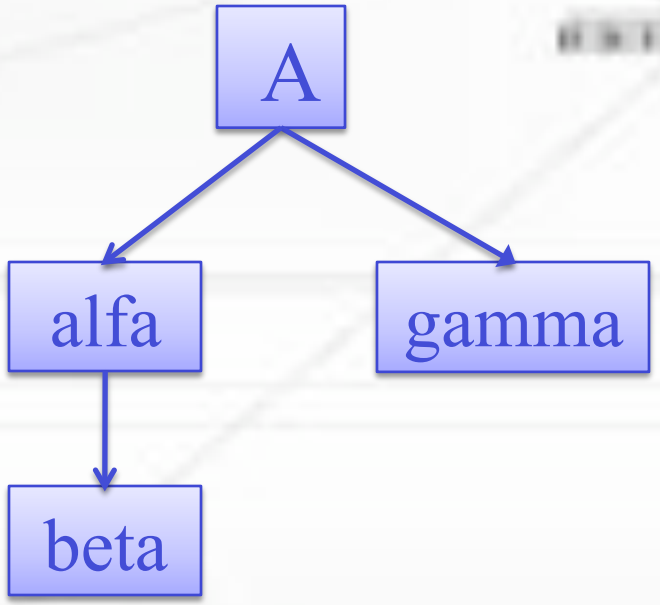
Esercizio 2: SIMPLESEM

```
1.program A{
2.  integer b,y;
3.  routine alfa() {
4.    integer a,x,w;
5.    routine beta () {
6.          integer z,g;
7.          a=z+x+b+w;
8.    };.....
9.    x=a+y; .....
10. };
11. routine gamma() {
12.  integer a,x,z,w; .....
13.  };
14.}
```

- Considerata la seguente catena di chiamate:
  - A→gamma→alfa→beta→gamma
- Mostrare lo stato della macchina astratta inclusi:
  - link statici
  - link dinamici
- Mostrare la rappresentazione delle variabili con <d,o> a seconda di scope statico o dinamico quando viene eseguita le istruzioni 7 e 9

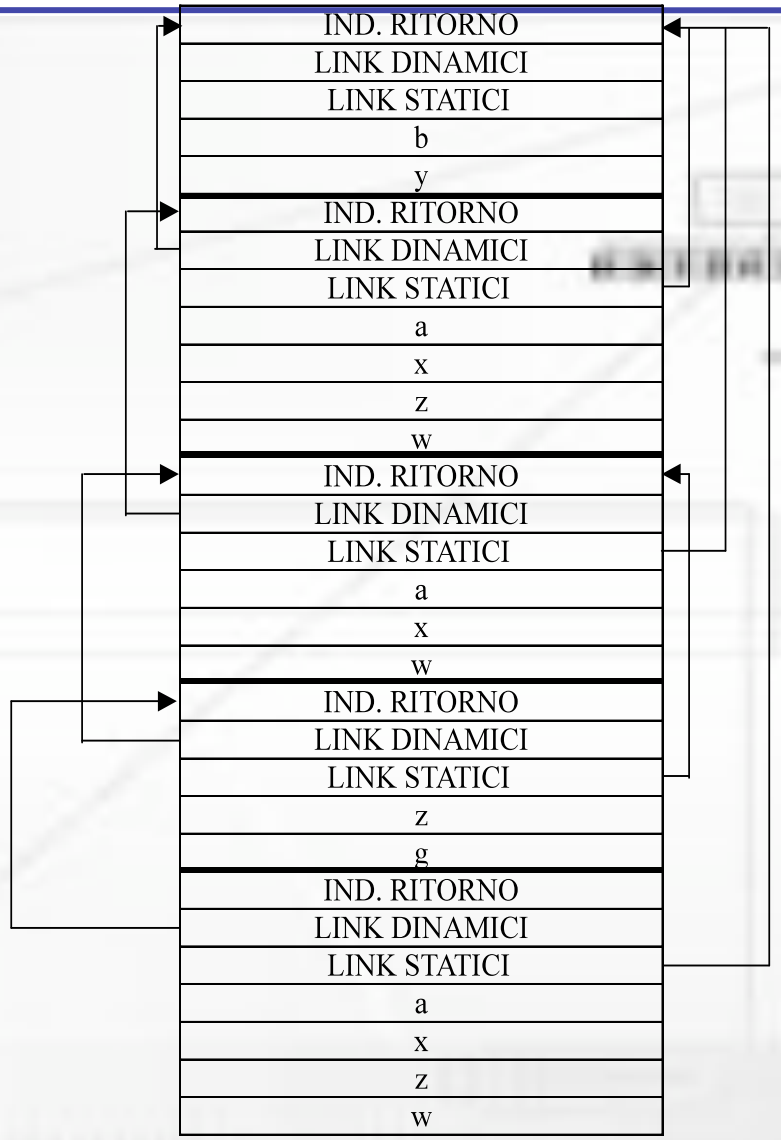
Soluzione: SNT (Static Nesting Tree)


```
1.program A{
2.  integer b,y;
3.  routine alfa() {
4.    integer a,x,w;
5.    routine beta () {
6.          integer z,g;
7.          a=z+x+b+w;
8.    };.....
9.    x=a+y; .....
10. };
11. routine gamma() {
12.  integer a,x,z,w; .....
13.  };
14.}
```

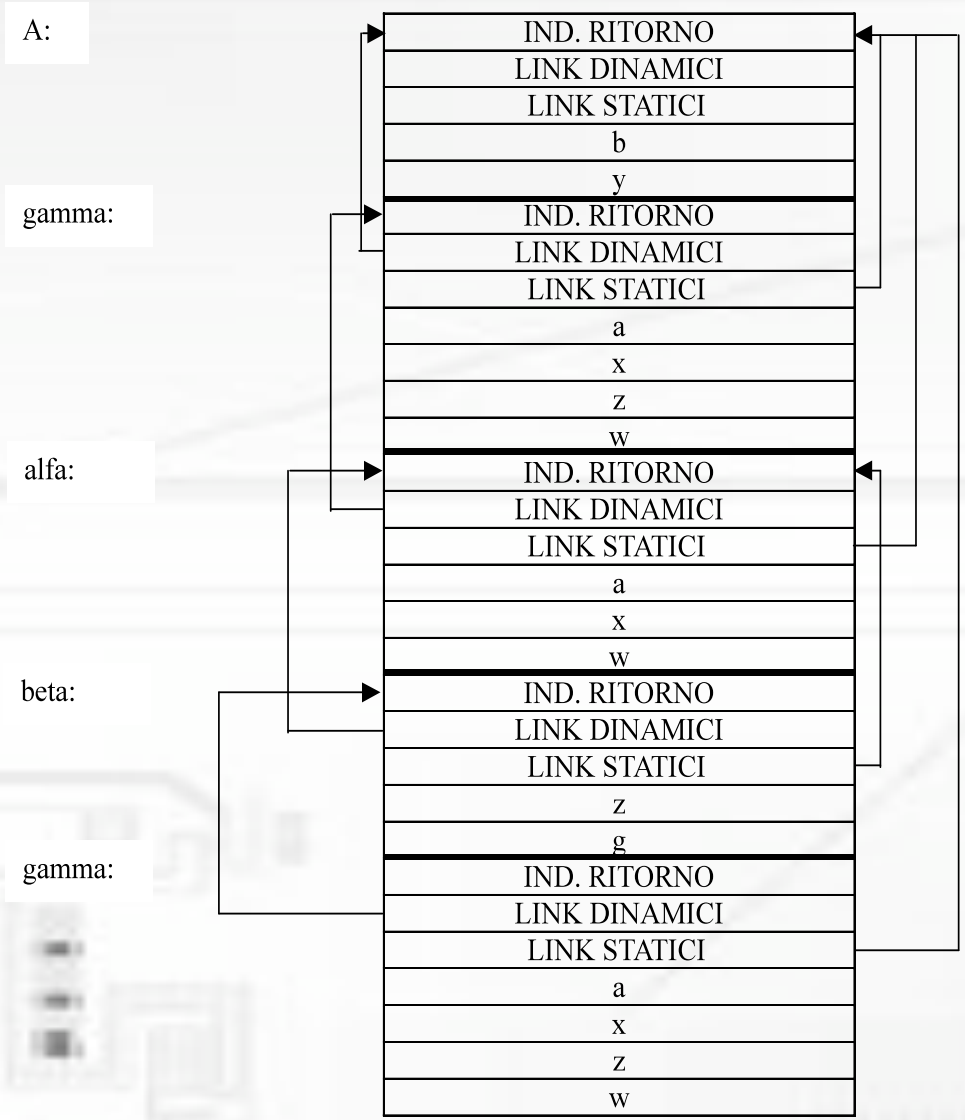


Soluzione: stato dello stack


```
1.program A{
2.  integer b,y;
3.  routine alfa() {
4.    integer a,x,w;
5.    routine beta () {
6.          integer z,g;
7.          a=z+x+b+w;
8.    };.....
9.    x=a+y; .....
10. };
11. routine gamma() {
12.  integer a,x,z,w; .....
13.  };
14.}
```



Soluzione: regole di scope statico



- Linea 7: a=z+x+b+w;

  a=<1,3>

  z=<0,3>

  x=<1,4>

  b=<2,3>

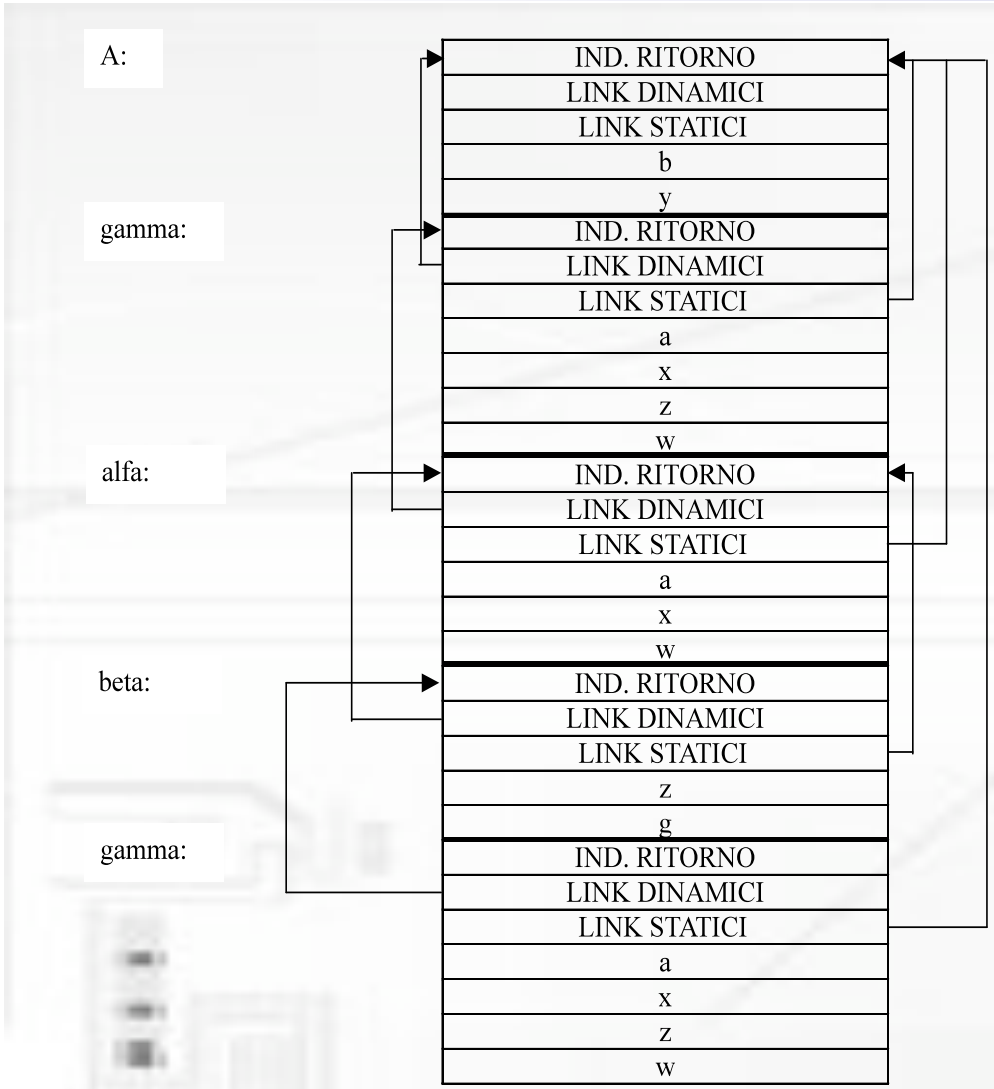
  w=<1,5>
- Linea 9: x=a+y;

  x=<0,4>

  a=<0,3>

  y=<1,4>

Soluzione: regole di scope dinamico



- Linea 7: a=z+x+b+w;

  a=<1,3>

  z=<0,3>

  x=<1,4>

  b=<3,3>

  w=<1,5>

- Linea 9: x=a+y;

  x=<0,4>

  a=<0,3>

  y=<2,4>

Esercizio 3: passaggio dei parametri


```
1. program esempio;
2. var x:integer
3.  procedure p(y:integer)
4.  begin
5.        x:=5;
6.        y:=y+x;
7.  end;
8. begin
9.  x:=10;
10.  p(x);
11. write(x);
12. end
```

- Analizzare il valore stampato a seconda della tipologia di passaggio di parametri considerata

Soluzione: call by reference


```
1.  program esempio;
2.  var x:integer
3.   procedure p(y:integer)
4.   begin
5.         x:=5;
6.         y:=y+x;
7.   end;
8.  begin
9.   x:=10;
10.  p(x);
11.  write(x);
12. end
```

- I parametri formali e i parametri attuali dividono la stessa area di memoria

# Riga	x	y
9	10	
3	10	10
5	5	5
6	10	10
11	10	

Soluzione: call by value


```
1.  program esempio;
2.  var x:integer
3.   procedure p(y:integer)
4.   begin
5.         x:=5;
6.         y:=y+x;
7.   end;
8.  begin
9.   x:=10;
10.  p(x);
11.  write(x);
12.  end
```

- I parametri attuali sono copiati in quelli formali

# Riga	x	y
9	10	
3	10	10
5	5	10
6	5	15
11	5	

Soluzione: call by result


```
1.  program esempio;
2.  var x:integer
3.   procedure p(y:integer)
4.   begin
5.         x:=5;
6.         y:=y+x;
7.   end;
8.  begin
9.   x:=10;
10.  p(x);
11.  write(x);
12.  end
```

- Il parametro formale viene copiato in quello attuale al termine della procedura

# Riga	x	y
9	10	
3	10	0
5	5	0
6	5	5
11	5	

Soluzione: call by value-result


```
1.  program esempio;
2.  var x:integer
3.   procedure p(y:integer)
4.   begin
5.         x:=5;
6.         y:=y+x;
7.   end;
8.  begin
9.   x:=10;
10.  p(x);
11.  write(x);
12. end
```

- Il parametro attuale viene copiato in quello formale alla chiamata; quello formale viene copiato in quello attuale al termine della procedura

# Riga	x	y
9	10	
3	10	10
5	5	10
6	5	15
11	15	

Soluzione: call by name


```
1.  program esempio;
2.  var x:integer
3.   procedure p(y:integer)
4.   begin
5.         x:=5;
6.         y:=y+x;
7.   end;
8.  begin
9.   x:=10;
10.  p(x);
11.  write(x);
12. end
```

- Nel testo della procedura i par. formali sostituiscono quelli attuali

# Riga	x	y
9	10	
3	10	10
5	5	5
6	10	10
11	10	

Esercizio 5: SIMPLESEM


```
program pippo
var a,b,c:integer
     procedure p(procedure x)
     var a,b,d:integer;
       procedure q
       var b,e:integer;
       ..............
       end q;
     if c=0 then
       begin
             c:=c+1;
             x(q);
       end
     else x();
     end p;
 ..............
c:=0;
p(p);
end;
```

- Disegnare lo stato della macchina astratta, tracciando link statici e dinamici, fino a quando q viene chiamata per la prima volta.

Soluzione: catena di chiamate

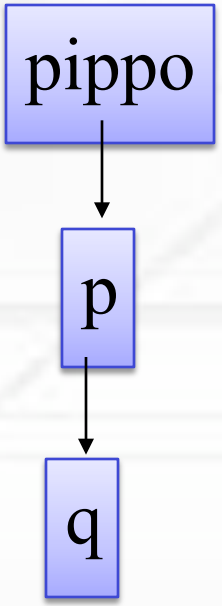

```
program pippo
var a,b,c:integer
    procedure p(procedure x)
    var a,b,d:integer;
      procedure q
      var b,e:integer;
      ..............
      end q;
    if c=0 then
      begin
            c:=c+1;
            x(q);
      end
    else x();
    end p;
 ..............
c:=0;
p(p);
end;
```

- Pippo pone c=0 e chiama p(p)
- In p(p) si entra nel ramo then dell'if ponendo quindi c=1 e chiamando p(q)
- In p(q) si entra nel ramo else dell'if quindi si chiama q
- Quindi la sequenza delle chiamate e':

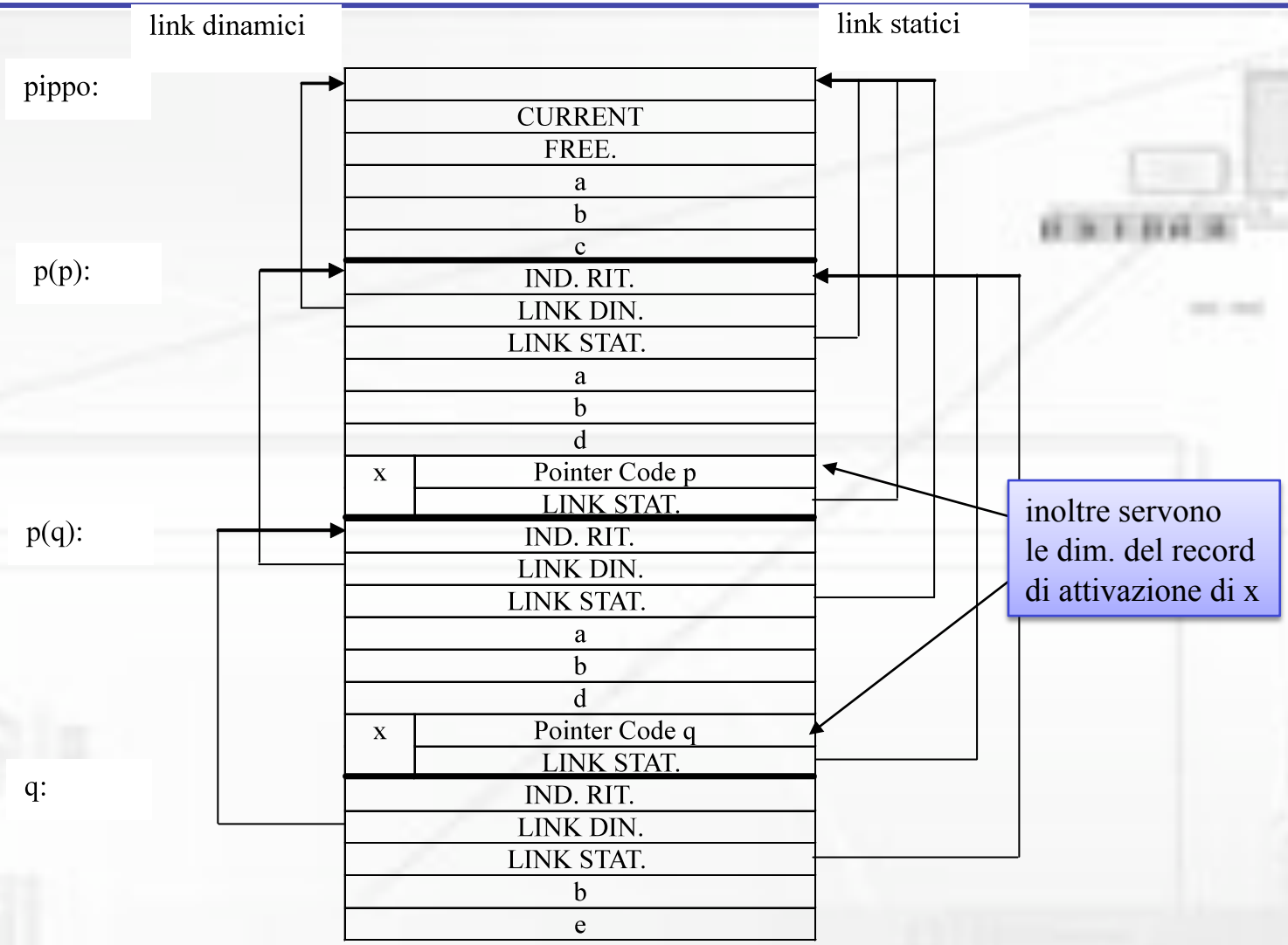
  pippo → p(p) → p(q) → q

Soluzione: SNT


```
program pippo
var a,b,c:integer
    procedure p(procedure x)
    var a,b,d:integer;
     procedure q
     var b,e:integer;
     ..............
     end q;
    if c=0 then
     begin
          c:=c+1;
          x(q);
     end
    else x();
    end p;
 ..............
c:=0;
p(p);
end;
```



Soluzione: stato dello stack



Esercizio 6: SIMPLESEM


```
program pippo
var l,m,n:real;
    procedure q(procedure w)
      procedure p
      var r,s:real;
      ......................
      end
    if n>0 then w()
    else begin
      n:=n+1.77;
      w(p);
    end
.........................
n:= -0.5;
q(q);
end
```

- Disegnare lo stato della macchina astratta, tracciando link statici e dinamici, fino alla chiamata di w effettuata nel ramo then della procedura q
- Si scriva inoltre come viene tradotto l'accesso alla variabile n in termini della coppia (distanza, offset) [scope statico].

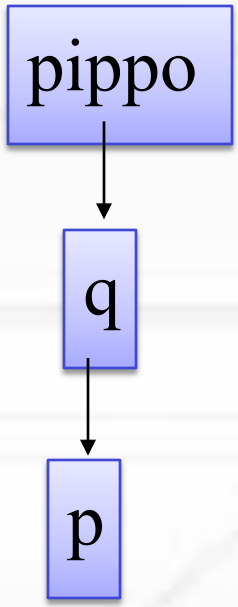
Soluzione: catena di chiamate


```
program pippo
var l,m,n:real;
    procedure q(procedure w)
      procedure p
      var r,s:real;
      ......................
      end
    if n>0 then w()
    else begin
      n:=n+1.77;
      w(p);
    end
.........................
n:= -0.5;
q(q);
end
```

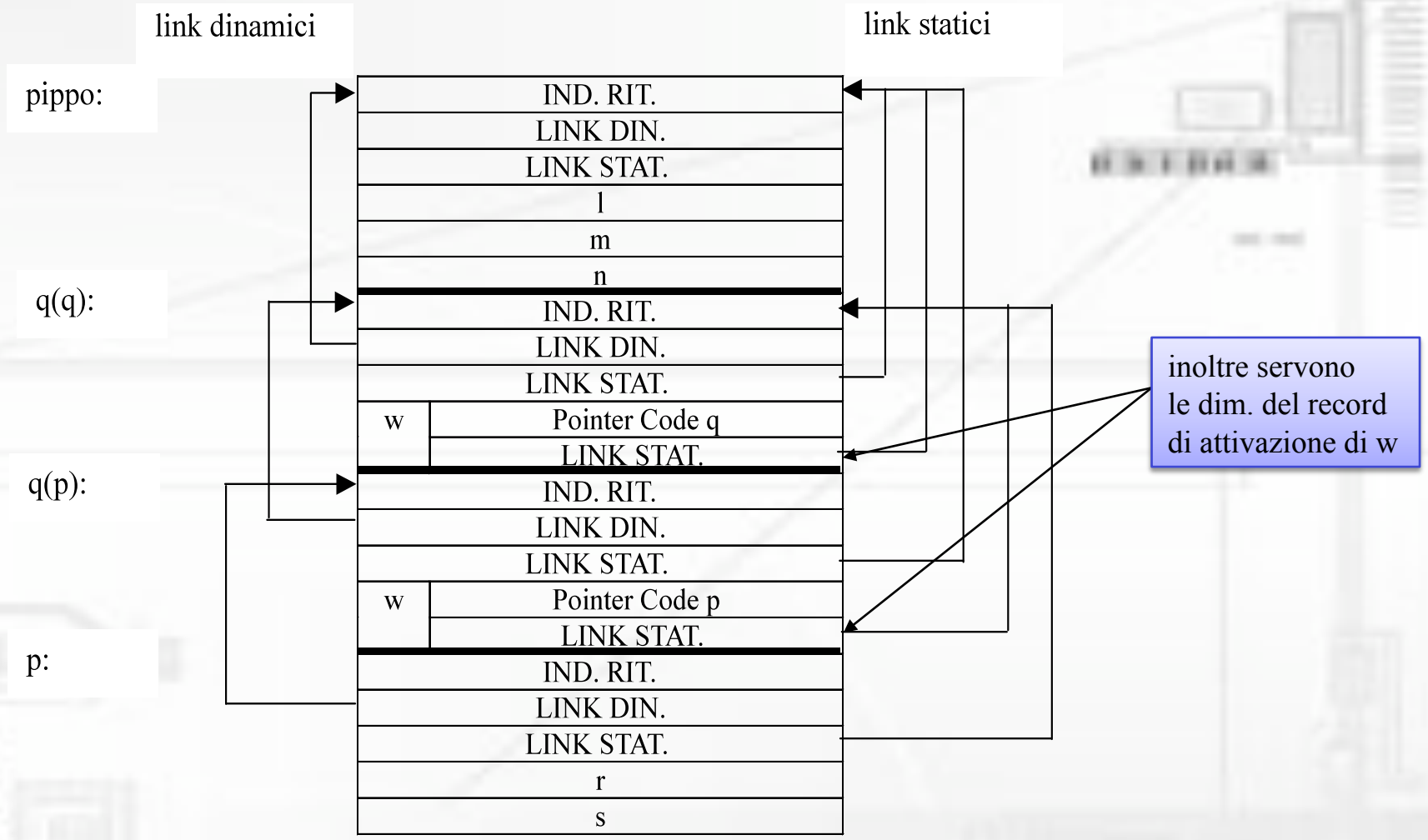
- Pippo pone n=-0.5 e chiama q(q)
- In q(q) si entra nel ramo else dell'if ponendo quindi n=-0.5+1.77=1.27 e chiamando q(p)
- In q(p) si entra nel ramo then dell'if quindi viene chiamata p
- Quindi la sequenza delle chiamate e':

pippo → q(q) → q(p) → p

Soluzione: SNT



Soluzione: stato dello stack



Soluzione: accesso alla variabile n

- Accesso alla variabile n da pippo: n=<0,5>
- Accesso alla variabile n per la prima chiamata di q: n=<1,5>
- Accesso alla variabile n per la seconda chiamata di q: n=<1,5> (valgono regole di scope statico)

Esercizio 7: SIMPLESEM

- Si dica, giustificando brevemente la risposta, se la seguente affermazione è vera o falsa:

- “Se un programma C non fa uso né di blocchi né di sottoprogrammi annidati (ossia se consiste solo del main e di una lista di sottoprogrammi dichiarati tutti allo stesso livello del main) la semantica determinata dalla catena statica equivale alla semantica della catena dinamica.”

Soluzione

- L'affermazione e' falsa
- Infatti, se si adotta la regola della catena statica le uniche variabili non locali usabili dai vari sottoprogrammi sono quelle globali; si invece si adotta la regola della catena dinamica un generico sottoprogramma P "cercherà" una variabile non locale x nel record di attivazione del sottoprogramma Q che lo ha chiamato e via via più in basso nella pila.

Esercizio

- Si consideri il seguente frammento di programma scritto in un linguaggio ipotetico che consente di dichiarare
  - variabili globali
  - funzioni all'interno di funzioni
- e che adotta regole di visibilità (scope) statiche:

Esercizio


```
// variabili globali
integer x, y, z;
// prototipo della funzione
void fun3 ();
void fun1 (integer m)
{// NB: il parametro m è passato
    per valore
  integer x, n;
  void fun2 ( ) {
    integer z;
    x = z + m + y; (*)
  };
  ...
  m = x + n + z; (**)
  ...
};
```

```
void fun3 () {
  integer x, m, z, w;
  ...
  fun1(z);
  ...
};
...
void main {
...
fun3();
...
};
```

Esercizio

- calcolare l'attributo (distanza, offset) per le variabili che appaiono nell'istruzione (*);
- calcolare l'attributo (distanza, offset) per le variabili che appaiono nell'istruzione (**);
- schizzare lo stato della memoria (mostrando i links statici e dinamici la disposizione delle variabili) nel caso in cui main chiami fun3, che chiama fun1, che chiama fun2, che chiama fun3, che chiama fun1.