Compitino 2

2–3 dicembre 1999

Correzione del compitino del Turno 1

Esercizio 1

Si considerino le seguenti definizioni di tipo:

```
TYPE lettera = 'A'..'Z';
     cifra = '0'..'9';
     indice = 1..10;
     a = RECORD
           tab: ARRAY[lettera, indice] OF cifra;
           ins: SET OF cifra
         END;
```

Scrivere in Pascal una `FUNCTION` che riceva un parametro `x` di tipo `a` e restituisca, come risultato, il numero di elementi dell'array memorizzato nel campo `tab` di `x`, che contengono valori appartenenti all'insieme memorizzato nel campo `ins` di `x`.

La funzione richiesta deve ricevere un parametro di tipo `a` e restituire un numero intero, senza operare alcuna modifica sul parametro. Pertanto l'intestazione può essere:

```
FUNCTION f (x: a): integer;
```

In alternativa, visto che il tipo `a` è strutturato e occupa una certa quantità di memoria, al posto del passaggio per valore può risultare utile il passaggio per riferimento, che in questo caso permette di risparmiare tempo e spazio.

Il record `x` ricevuto come parametro ha due campi: un array bidimensionale i cui elementi sono di tipo `cifra` (campo `tab`) e un `SET OF cifra` (campo `ins`). La funzione deve esaminare gli elementi dell'array che si trova nel campo `tab`, e contare quanti di essi appartengono all'insieme memorizzato nel campo `ins`. Osserviamo che per selezionare un elemento dell'array occorrono due indici, rispettivamente di tipo `lettera` e di tipo `indice`. In particolare, possiamo dichiarare:

```
VAR
   i: lettera;
   j: indice;
```

Per selezionare l'elemento di posto `i, j` dell'array memorizzato nel campo `tab` del record `x`, è sufficiente scrivere `x.tab[i, j]`. Per selezionare invece l'insieme memorizzato nel campo `ins` di `x`, scriviamo `x.ins`.

La scansione dell'array avviene utilizziamo due cicli innestati. All'interno, mediante una variabile `cont`, preventivamente inizializzata a `0`, contiamo il numero di elementi dell'array che appartengono all'insieme `x.ins`. Dopo avere esaminato tutti gli elementi della matrice, restituiamo come risultato, scrivendo `f := cont`, il valore trovato:

```
FUNCTION f (x: a): integer;
   VAR
      i: lettera;
      j: indice;
      cont: integer;

BEGIN
   cont := 0;
   FOR i := 'A' TO 'Z' DO
      FOR j := 1 TO 10 DO
         IF x.tab[i, j] IN x.ins THEN cont := cont + 1;
   f := cont
END;
```

Esercizio 2

Per ognuna delle seguenti linee di codice individuare delle dichiarazioni di variabile ed eventualemente di tipo, nonché eventuali intestazioni di procedura o funzione, in modo che le istruzioni che vi appaiono risultino corrette dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile, spiegare il motivo.

- `a := -b[pred(a)]`
- `a := -b{pred(a)}`
- `a := -b(pred(a))`

Nel primo assegnamento, l'identificatore `b` è seguito da un'espressione racchiusa tra parentesi quadre. Pertanto `b` indica un array, il cui indice è dello stesso tipo di `a` (infatti, come selettore si usa il risultato di `pred(a)`). Alla variabile `a` viene assegnato il valore di un elemento dell'array, preceduto dal segno meno. Pertanto gli elementi dell'array e la variabile `a` devono essere di tipo `integer`. Possiamo dunque fornire le seguenti dichiarazioni:

```
VAR
   a: integer;
   b: ARRAY[1..100] OF integer;
```

Dopo il secondo assegnamento è contenuto un commento tra parentesi graffe. Pertanto l'assegnamento è `a := -b`. Essendoci l'operatore meno, le variabili possono essere `integer` o `real`. Ad esempio, possiamo dichiarare entrambe `a` e `b` di tipo `integer`.
Nel terzo assegnamento, il nome `b` è seguito da un'espressione tra parentesi tonde. Pertanto `b` è una `FUNCTION`. Con gli stessi ragionamenti fatti per il primo assegnamento possiamo concludere che `a` deve essere dichiarata di tipo `integer`, mentre per `b` può essere fornita la seguente intestazione

```
FUNCTION b (x: integer): integer;
```

Esercizio 3

Per ogni blocco di codice del seguente programma indicare i nomi delle variabili che possono essere utilizzate e delle procedure che possono essere richiamate, precisando, per ciascun nome, il blocco in cui è stata effettuata la dichiarazione.

```
PROGRAM p;
   VAR
      d, b: char;

   PROCEDURE c;
      VAR
         b: integer;
   BEGIN {c}
      ...
   END; {c}

   PROCEDURE a;
      VAR
         d, e: integer;

      PROCEDURE b;
         VAR
            c, f: integer;
      BEGIN {b}
         ...
      END; {b}

   BEGIN {a}
      ...
   END; {a}

BEGIN {p}
   ...
END. {p}
```

Iniziamo ad analizzare il blocco `p` (programma principale). In esso è possibile utilizzare tutti gli oggetti dichiarati localmente, cioè le variabili `d` e `b` e i sottoprogrammi `PROCEDURE c` e `PROCEDURE a`. Non vi sono altri oggetti utilizzabili (salvo quelli corrispondenti agli identificatori predefiniti che lasciamo sottointesi).

Analizziamo ora il blocco `c`. In esso possono essere utilizzati gli identificatori definiti localmente, in questo caso la variabile `b`, piú quelli definiti nell'ambiente esterno (cioè in `p`) *prima* di `c` che non siano adombrati. Poiché l'identificatore `b` definito in `p` è adombrato dalla variabile `b` dichiarata in `c`, e la procedura `a` è definita dopo, gli unici identificatori ereditati da `p` sono `d` e `c`.

Consideriamo ora il blocco `a`. In esso possono essere utilizzati gli identificatori locali `d, e, b`. Gli identificatori `d` e `b` del programma principale sono adombrati, rispettivamente, dalla variabile `d` e dalla procedura `b` definite localmente. Possono essere richiamate le procedure `c` e `a` del programma principale.

Esaminiamo infine il blocco `b`. In esso possono essere utilizzati gli identificatori locali `c` e `f`. Le risorse ereditate dal blocco esterno sono gli identificatori `d,e, b` definiti nel blocco `a`, e l'identificatore `a` definito nel programma principale. L'identificatore `c` del programma principale, che era visibile in `a`, è qui adombrato dalla variabile `c`. Riassumendo, indichiamo gli identificatori utilizzabili in ciascun blocco di codice e, tra parentesi, il nome del blocco in cui sono stati dichiarati:

- variabili utilizzabili e procedure richiamabili nel codice di `p`: `d(p)`, `b(p)`, `c(p)`, `a(p)`;
- variabili utilizzabili e procedure richiamabili nel codice di `c`: `b(c)`, `d(p)`, `c(p)`;
- variabili utilizzabili e procedure richiamabili nel codice di `a`: `d(a)`, `e(a)`, `b(a)`, `c(p)`, `a(p)`;
- variabili utilizzabili e procedure richiamabili nel codice di `b`: `c(b)`, `f(b)`, `d(a)` `e(a)`, `b(a)`, `a(p)`.

Esercizio 4

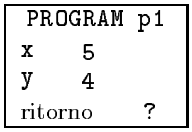
Per ognuno dei seguenti programmi scrivere l'output prodotto su input `5`.

```
PROGRAM p1 (input, output);

   VAR
      x, y: integer;

   PROCEDURE a;
      VAR y: integer;
   BEGIN {a}
      y := 3 * x;
      x := y - 1
   END; {a}

   PROCEDURE b;
      VAR x: integer;
   BEGIN {b}
      x := y - 2;
      a
   END; {b}

BEGIN {p1}
   readln(x);
   y := x - 1;
   b;
   writeln(x, y)
END. {p1}
```

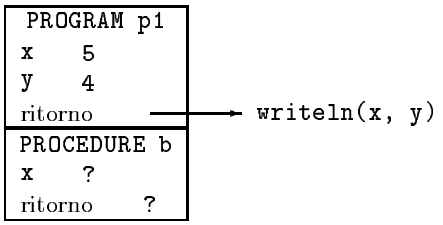
```
PROGRAM p2 (input, output);

   VAR
      x, y: integer;

   FUNCTION f (n: integer): integer;

   BEGIN {f}
      IF (n MOD 3 = 0) OR (n <= 1) THEN
         f := n
      ELSE IF n MOD 2 = 0 THEN
         f := 3 * f(n DIV 2)
      ELSE
         f := 2 * f(n - 1) + 2 * f(n - 2)
   END; {f}
```

```
BEGIN {p2}
   readln(x);
   y := f(x);
   writeln(y)
END. {p2}

PROGRAM p3 (input, output);

   VAR
      x, y: integer;

   PROCEDURE a (v: integer;
                VAR w: integer);

   BEGIN
      w := 3 * (v + 2);
      v := w - v
   END;

BEGIN {p3}
   readln(x);
   a(x, y);
   writeln(x, y)
END. {p3}
```

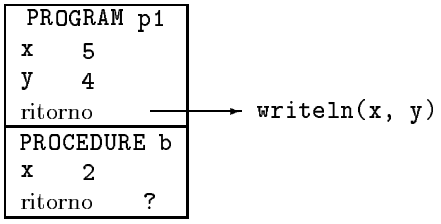
Simuliamo il comportamento del programma `p1` su ingresso `5` (poiché il programma non contiene funzioni, evitiamo di rappresentare nei record di attivazione lo spazio per il valore restituito dalle funzioni). All'inizio, nella memoria è contenuto solo il record d'attivazione del programma principale. Dopo l'esecuzione dell'istruzione `readln(x)` e dell'assegnamento `y := x - 1`, tale record contiene:



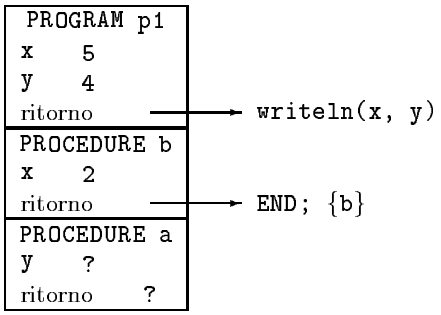
Nell'istruzione successiva viene chiamata la procedura `b` che, a sua volta, contiene una variabile locale di nome `x`. Dopo la chiamata di `b`, il contenuto dello stack sarà:



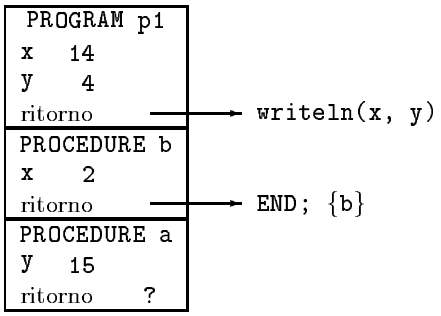
La prima istruzione della procedura `b` è l'assegnamento `x := y - 2`. La variabile `y` non è dichiarata localmente. In base alle regole di visibilità, la procedura `b` può accedere alla variabile `y` del programma principale, che contiene `4`. Pertanto il valore da assegnare a `x` è `2`. Essendo `x` dichiarata localmente, il risultato viene posto nella variabile `x` che si trova nel record di attivazione di `b`:



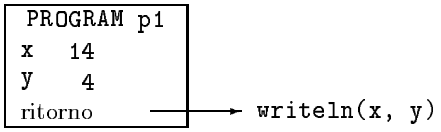
L'istruzione successiva è una chiamata della procedura `a`, in cui è dichiarata una variabile locale di nome `y`. Dopo tale chiamata il contenuto della memoria è:



La prima istruzione di `a` è l'assegnamento `y := 3 * x`. In base alle regole di visibilità, l'identificatore `x` che compare in tale assegnamento è quello dichiarato nel programma principale; l'identificatore `y` indica invece la variabile locale. Pertanto, l'effetto di tale istruzione è quello di assegnare il valore `15` alla variabile `y` che si trova nel record di attivazione di `a`. L'istruzione successiva è l'assegnamento `x := y - 1`. Anche in questo caso, in base alle regole di visibilità, `y` denota la variabile locale di `a`, mentre `x` denota una variabile dichiarata nel programma principale. Pertanto, questa istruzione produce un effetto sul record di attivazione del programma principale:



Terminata l'esecuzione del codice di `a`, se ne distrugge il record di attivazione, e si riprende l'esecuzione della procedura `b` dal punto di ritorno, corrispondente alla fine della procedura stessa. Pertanto, si effettua il rientro al programma principale. A questo punto la memoria contiene:



L'esecuzione riprende quindi dall'istruzione `writeln(x, y)` che scrive in output i valori `14 4`.

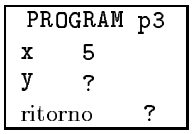
Per determinare l'output prodotto dal programma `p2` possiamo utilizzare lo stesso procedimento, disegnando man mano l'evoluzione dello stack. Tuttavia, siccome il programma è abbastanza semplice e la funzione ricorsiva `f` non modifica variabili globali, possiamo semplicemente osservare che l'output del programma è il valore prodotto dalla chiamata `f(5)`.

In base alle istruzioni contenute nella funzione, notiamo che `f(5)` è ottenibile come `2 * f(4) + 2 * f(3)`.

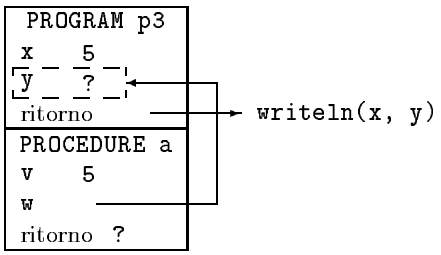
Di nuovo, esaminando il testo della funzione, osserviamo che `f(4)` è uguale a `3 * f(2)`, `f(2)` è uguale a `3 * f(1)`, e `f(1)` è uguale a `1`. Sostituendo otteniamo `f(2) = 3` e `f(4) = 9`.

Per completare il calcolo di `f(5)`, dobbiamo valutare `f(3)`. Dal testo della funzione è immediato vedere che `f(3)` è uguale a `3`. Sostituendo i valori ottenuti per `f(3)` e per `f(4)` nell'espressione data per `f(5)`, concludiamo che `f(5)`, e dunque l'output del programma, è `24`.

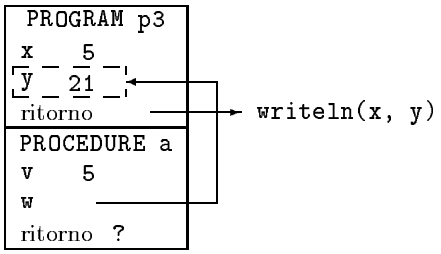
Esaminiamo infine il programma `p3`, sempre con input `5`. Dopo l'esecuzione dell'istruzione `readln(x)`, il contenuto della memoria è:



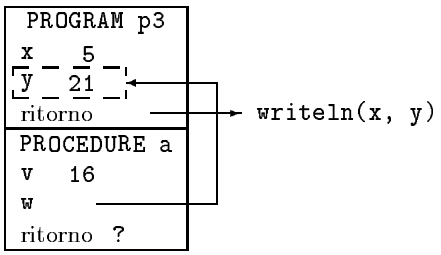
L'istruzione successiva è una chiamata della `PROCEDURE a` in cui al parametro formale `v` per valore è associato il valore della variabile `x`, mentre al parametro formale `w` per riferimento è associata la variabile `y` del programma principale. Dopo l'esecuzione di tale istruzione, il contenuto della memoria diviene:



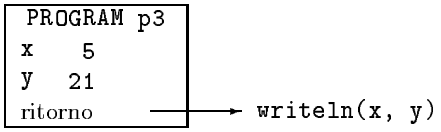
La prima istruzione della procedura assegna il valore `3 * (v + 2)`, cioè `21`, a `w`, che è un riferimento a `y`. Pertanto l'effetto di questa istruzione è quello di assegnare `21` alla variabile `y` del programma principale:



L'istruzione successiva assegna il risultato dell'espressione `w - v`, dunque `16`, alla variabile `v`:



Essendo terminato il codice di `a`, si rientra al punto di chiamata, dopo avere distrutto il record di attivazione di `a`:



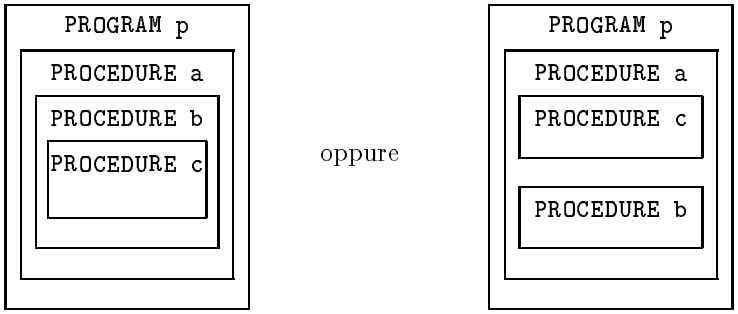
L'esecuzione riprende dall'istruzione `writeln(x, y)`, con la quale vengono stampati in output i contenuti delle due variabili `x` e `y`, cioè i numeri `5` e `21`.

Esercizio 5

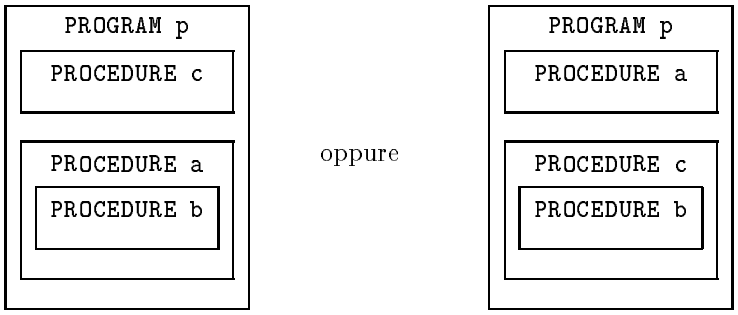
Indicare come potrebbe essere strutturato un programma `p` che contenga tre procedure `a`, `b` e `c`, e nel quale non sia utilizzata la direttiva `forward`, in modo tale che:

- la procedura `b` sia in grado di richiamare le altre due procedure, ma non sia richiamabile dal programma principale; il programma principale sia in grado di richiamare la procedura `a`.
- il programma principale possa richiamare la procedura `b` e la procedura `a`, la procedura `b` possa richiamare la procedura `c`, ma non la procedura `a`, la procedura `a` possa richiamare tutte le procedure.

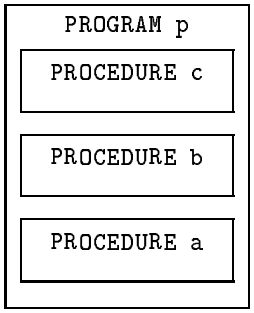
Nel primo caso, la procedura `a` deve essere dichiarata internamente al programma principale, per poter essere richiamata da esso. La procedura `b`, al contrario deve trovarsi internamente ad un'altra procedura (in modo che il programma principale non la possa richiamare) e in una posizione tale da potere richiamare le altre due procedure. Due possibili soluzioni consistono nel dichiarare `b` interna ad `a`, e `c` interna a `b`, oppure `b` e `c` interne ad `a`, con `c` dichiarata prima di `b`:



Sono possibili anche altre soluzioni: ad esempio, si potrebbero dichiarare le procedure `a` e `c` interne al programma principale, e `b` interna alla seconda delle due:



Nel secondo caso le procedure `a` e `b`, per potere essere richiamate dal programma principale, dovranno essere dichiarate immediatamente al suo interno. In particolare `b` deve trovarsi prima di `a` (altrimenti potrebbe richiamare `a`). Affinché sia `a` che `b` possano richiamare `c`, la procedura `c` verrà dichiarata, sempre a livello del programma principale, prima di entrambe:



Esercizio 6

Si consideri la funzione f sugli interi definita come:

$$f(x) = \begin{cases} 1 & \text{se } x \leq 1 \\ 2 * f(x-1) + f(x-2) & \text{altrimenti.} \end{cases}$$

Scrivere in Pascal una `FUNCTION f` ricorsiva per il calcolo di f.

Indicare il valore di $f(4)$. Utilizzando la ricorsione e una selezione, è possibile scrivere la seguente `FUNCTION` che calcola il valore di f:

```
FUNCTION f (x: integer): integer;
BEGIN
   IF x <= 1 THEN
      f := 1
   ELSE f := 2 * f(x - 1) + f(x - 2)
END;
```

Dalla definizione si ha: $f(4) = 2 * f(3) + f(2)$, $f(3) = 2 * f(2) + f(1)$, $f(2) = 2 * f(1) + f(0)$, $f(0) = f(1) = 1$. Sostituendo via via nelle espressioni precedenti, si ottiene: $f(2) = 3$, $f(3) = 7$ e, infine, $f(4) = 17$.

Correzione del compitino del Turno serale

Esercizio 1

Si considerino le seguenti definizioni di tipo:

```
TYPE lettera = 'A'..'Z';
     cifra = '0'..'9';
     indice = 1..10;
     r = RECORD
            tab: ARRAY[lettera, indice] OF SET OF cifra;
            el: cifra
         END;
```

Scrivere in Pascal una `FUNCTION` che riceva un parametro `x` di tipo `r` e restituisca, come risultato, il numero di elementi dell'array memorizzato nel campo `tab` di `x`, ai quali appartenga il valore memorizzato nel campo `el` di `x`.

La funzione richiesta deve ricevere un parametro di tipo `r` e restituire un numero intero, senza operare alcuna modifica sul parametro. Pertanto l'intestazione può essere:

```
FUNCTION f (x: r): integer;
```

In alternativa, visto che il tipo `r` è strutturato e occupa una certa quantità di memoria, al posto del passaggio per valore può risultare utile il passaggio per riferimento, che in questo caso permette di risparmiare tempo e spazio.

Il record `x` ricevuto come parametro ha due campi: un array bidimensionale i cui elementi sono di tipo `SET OF cifra` (campo `tab`) e un valore di tipo `cifra` (campo `ins`). La funzione deve esaminare gli elementi dell'array che si trova nel campo `tab`: ciascuno di essi è un insieme di valori di tipo `cifra`; la funzione deve contare quanti di questi insiemi contengono il valore memorizzato nel campo `el`. Osserviamo che per selezionare un elemento dell'array occorrono due indici, rispettivamente di tipo `lettera` e di tipo `indice`. In particolare, possiamo dichiarare:

```
VAR
   i: lettera;
   j: indice;
```

Per selezionare l'elemento di posto `i, j` dell'array memorizzato nel campo `tab` del record `x`, è sufficiente scrivere `x.tab[i, j]`. Per selezionare invece il valore memorizzato nel campo `el` di `x`, scriviamo `x.el`.

La scansione dell'array avviene utilizziamo due cicli innestati. All'interno, mediante una variabile `cont`, preventivamente inizializzata a `0`, contiamo il numero di elementi dell'array che ai quali appartengono il valore `x.el`. Dopo avere esaminato tutti gli elementi della matrice, restituiamo come risultato, scrivendo `f := cont`, il valore trovato:

```
FUNCTION f (x: r): integer;
   VAR
      i: lettera;
      j: indice;
      cont: integer;

BEGIN
   cont := 0;
   FOR i := 'A' TO 'Z' DO
      FOR j := 1 TO 10 DO
         IF x.el IN x.tab[i, j] THEN cont := cont + 1;
   f := cont
END;
```

Esercizio 2

Per ognuna delle seguenti linee di codice individuare delle dichiarazioni di variabile ed eventualemente di tipo, nonché eventuali intestazioni di procedura o funzione, in modo che le istruzioni che vi appaiono risultino corrette dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile, spiegare il motivo.

- `a := b[chr(a)]`
- `a := b{chr(a)}`
- `a := b(chr(a))`

Nel primo assegnamento, l'identificatore `b` è seguito da un'espressione racchiusa tra parentesi quadre. Pertanto `b` indica un array, il cui indice è un `char` (infatti, come selettore si usa il risultato della funzione `chr`). Alla variabile `a` viene assegnato il valore di un elemento dell'array; inoltre alla stessa variabile viene applicata la funzione `chr`. Pertanto gli elementi dell'array e la variabile `a` devono essere di tipo `integer` (o subrange). Possiamo dunque fornire le seguenti dichiarazioni:

```
VAR
   a: integer;
   b: ARRAY[char] OF integer;
```

Dopo il secondo assegnamento è contenuto un commento tra parentesi graffe. Pertanto l'assegnamento è `a := b`. Affinché l'assegnamento sia possibile, le variabili `a` e `b` devono essere dello stesso tipo (anche strutturato, con eccezione dei file) o di tipi compatibili. Ad esempio, possiamo dichiarare entrambe `a` e `b` di tipo `integer`.
Nel terzo assegnamento, il nome `b` è seguito da un'espressione tra parentesi tonde. Pertanto `b` è una `FUNCTION`. Con gli stessi ragionamenti fatti per il primo assegnamento possiamo concludere che `a` deve essere dichiarata di tipo `integer`, mentre per `b` può essere fornita la seguente intestazione

```
FUNCTION b (x: char): integer;
```

Esercizio 3

Per ogni blocco di codice del seguente programma indicare i nomi delle variabili che possono essere utilizzate e delle procedure che possono essere richiamate, precisando, per ciascun nome, il blocco in cui è stata effettuata la dichiarazione.

```
PROGRAM p;
   VAR
      d, b: char;

   PROCEDURE c;
      VAR
         b: integer;
   BEGIN {c}
      ...
   END; {c}

   PROCEDURE a;
      VAR
         c, d: integer;

      PROCEDURE b;
         VAR
            a, c: integer;
      BEGIN {b}
         ...
      END; {b}

   BEGIN {a}
      ...
   END; {a}

BEGIN {p}
   ...
END. {p}
```

Iniziamo ad analizzare il blocco `p` (programma principale). In esso è possibile utilizzare tutti gli oggetti dichiarati localmente, cioè le variabili `d` e `b` e i sottoprogrammi `PROCEDURE c` e `PROCEDURE`

`a`. Non vi sono altri oggetti utilizzabili (salvo quelli corrispondenti agli identificatori predefiniti che lasciamo sottointesi).

Analizziamo ora il blocco `c`. In esso possono essere utilizzati gli identificatori definiti localmente, in questo caso la variabile `b`, piú quelli definiti nell'ambiente esterno (cioè in `p`) *prima* di `c` che non siano adombrati. Poiché l'identificatore `b` definito in `p` è adombrato dalla variabile `b` dichiarata in `c`, e la procedura `a` è definita dopo, gli unici identificatori ereditati da `p` sono `d` e `c`.

Consideriamo ora il blocco `a`. In esso possono essere utilizzati gli identificatori locali `c, d, b`. Gli identificatori `d`, `b` e `c` del programma principale sono adombrati, rispettivamente, dalla variabile `d`, dalla procedura `b` e dalla variabile `c` definite localmente. Puó essere richiamata la procedura `a` del programma principale.

Esaminiamo infine il blocco `b`. In esso possono essere utilizzati gli identificatori locali `a` e `c`. Le risorse ereditate dal blocco esterno sono gli identificatori `d, b` definiti nel blocco `a`. L'identificatore `c` dichiarato in `a`, è qui adombrato dalla variabile `c`. Riassumendo, indichiamo gli identificatori utilizzabili in ciascun blocco di codice e, tra parentesi, il nome del blocco in cui sono stati dichiarati:

- variabili utilizzabili e procedure richiamabili nel codice di `p`: `d(p)`, `b(p)`, `c(p)`, `a(p)`;
- variabili utilizzabili e procedure richiamabili nel codice di `c`: `b(c)`, `d(p)`, `c(p)`;
- variabili utilizzabili e procedure richiamabili nel codice di `a`: `c(a)`, `d(a)`, `b(a)`, `a(p)`;
- variabili utilizzabili e procedure richiamabili nel codice di `b`: `a(b)`, `c(b)`, `d(a)`, `b(a)`.

Esercizio 4

Per ognuno dei seguenti programmi scrivere l'output prodotto su input `5`.

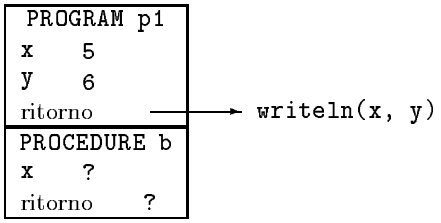
```
PROGRAM p1 (input, output);

   VAR
      x, y: integer;

   PROCEDURE a;
      VAR y: integer;
   BEGIN {a}
      y := 5 * x;
      x := y - 2
   END; {a}

   PROCEDURE b;
      VAR x: integer;
   BEGIN {b}
      x := y - 2;
      a
   END; {b}

BEGIN {p1}
   readln(x);
   y := x + 1;
   b;
   writeln(x, y)
END. {p1}
```

```
PROGRAM p2 (input, output);

   VAR
      x, y: integer;

   FUNCTION f (n: integer): integer;

   BEGIN {f}
      IF (n MOD 3 = 0) OR (n <= 1) THEN
         f := n
      ELSE IF n MOD 2 = 0 THEN
         f := 4 * f(n DIV 2)
      ELSE
         f := 2 * f(n - 1) + 3 * f(n - 2)
   END; {f}

BEGIN {p2}
   readln(x);
   y := f(x);
   writeln(y)
END. {p2}

PROGRAM p3 (input, output);

   VAR
      x, y: integer;

   PROCEDURE a (v: integer;
                  VAR w: integer);

   BEGIN
      w := 5 * (v + 1);
      v := w - v
   END;

BEGIN {p3}
   readln(x);
   a(x, y);
   writeln(x, y)
END. {p3}
```

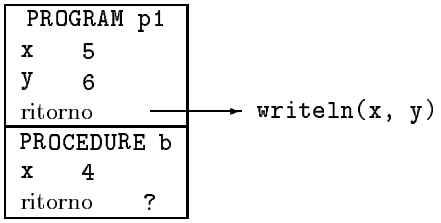
Simuliamo il comportamento del programma `p1` su ingresso `5` (poiché il programma non contiene funzioni, evitiamo di rappresentare nei record di attivazione lo spazio per il valore restituito dalle funzioni). All'inizio, nella memoria è contenuto solo il record d'attivazione del programma principale. Dopo l'esecuzione dell'istruzione `readln(x)` e dell'assegnamento `y := x + 1`, tale record contiene:

PROGRAM p1	
x	5
y	6
ritorno	?

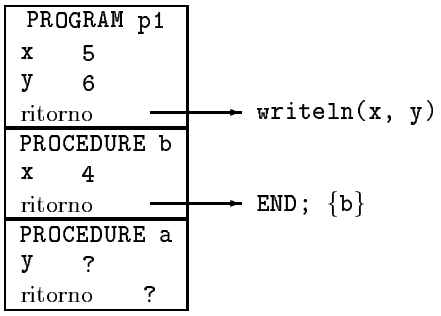
Nell'istruzione successiva viene chiamata la procedura `b` che, a sua volta, contiene una variabile locale di nome `x`. Dopo la chiamata di `b`, il contenuto dello stack sarà:



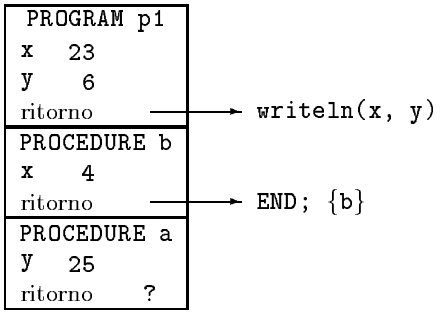
La prima istruzione della procedura `b` è l'assegnamento `x := y - 2`. La variabile `y` non è dichiarata localmente. In base alle regole di visibilità, la procedura `b` può accedere alla variabile `y` del programma principale, che contiene `6`. Pertanto il valore da assegnare a `x` è `4`. Essendo `x` dichiarata localmente, il risultato viene posto nella variabile `x` che si trova nel record di attivazione di `b`:



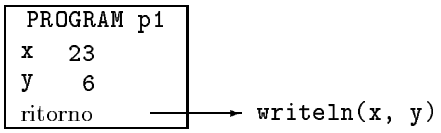
L'istruzione successiva è una chiamata della procedura `a`, in cui è dichiarata una variabile locale di nome `y`. Dopo tale chiamata il contenuto della memoria è:



La prima istruzione di `a` è l'assegnamento `y := 5 * x`. In base alle regole di visibilità, l'identificatore `x` che compare in tale assegnamento è quello dichiarato nel programma principale; l'identificatore `y` indica invece la variabile locale. Pertanto, l'effetto di tale istruzione è quello di assegnare il valore `25` alla variabile `y` che si trova nel record di attivazione di `a`. L'istruzione successiva è l'assegnamento `x := y - 2`. Anche in questo caso, in base alle regole di visibilità, `y` denota la variabile locale di `a`, mentre `x` denota una variabile dichiarata nel programma principale. Pertanto, questa istruzione produce un effetto sul record di attivazione del programma principale:



Terminata l'esecuzione del codice di `a`, se ne distrugge il record di attivazione, e si riprende l'esecuzione della procedura `b` dal punto di ritorno, corrispondente alla fine della procedura stessa. Pertanto, si effettua il rientro al programma principale. A questo punto la memoria contiene:



L'esecuzione riprende quindi dall'istruzione `writeln(x, y)` che scrive in output i valori `23 6`.

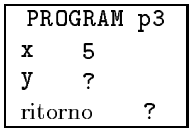
Per determinare l'output prodotto dal programma `p2` possiamo utilizzare lo stesso procedimento, disegnando man mano l'evoluzione dello stack. Tuttavia, siccome il programma è abbastanza semplice e la funzione ricorsiva `f` non modifica variabili globali, possiamo semplicemente osservare che l'output del programma è il valore prodotto dalla chiamata `f(5)`.

In base alle istruzioni contenute nella funzione, notiamo che `f(5)` è ottenibile come `2 * f(4) + 3 * f(3)`.

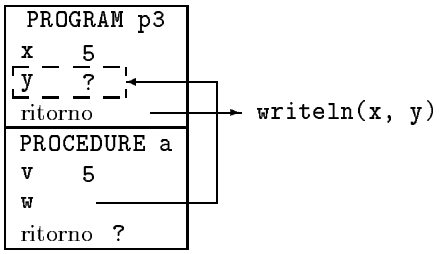
Di nuovo, esaminando il testo della funzione, osserviamo che `f(4)` è uguale a `4 * f(2)`, `f(2)` è uguale a `4 * f(1)`, e `f(1)` è uguale a `1`. Sostituendo otteniamo `f(2) = 4` e `f(4) = 16`.

Per completare il calcolo di `f(5)`, dobbiamo valutare `f(3)`. Dal testo della funzione è immediato vedere che `f(3)` è uguale a `3`. Sostituendo i valori ottenuti per `f(3)` e per `f(4)` nell'espressione data per `f(5)`, concludiamo che `f(5)`, e dunque l'output del programma, è `41`.

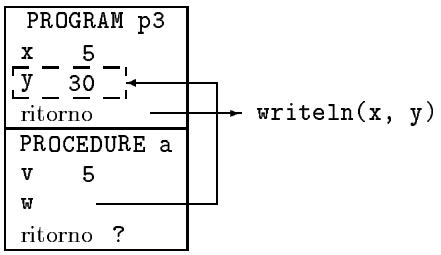
Esaminiamo infine il programma `p3`, sempre con input `5`. Dopo l'esecuzione dell'istruzione `readln(x)`, il contenuto della memoria è:



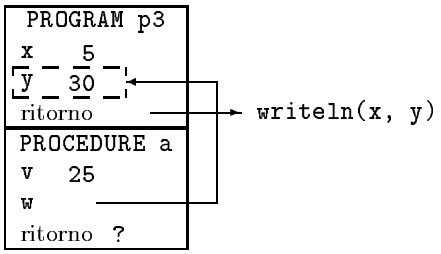
L'istruzione successiva è una chiamata della `PROCEDURE a` in cui al parametro formale `v` per valore è associato il valore della variabile `x`, mentre al parametro formale `w` per riferimento è associata la variabile `y` del programma principale. Dopo l'esecuzione di tale istruzione, il contenuto della memoria diviene:



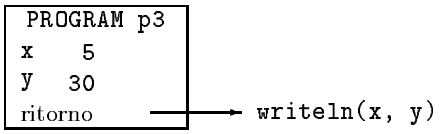
La prima istruzione della procedura assegna il valore `5 * (v + 1)`, cioè `30`, a `w`, che è un riferimento a `y`. Pertanto l'effetto di questa istruzione è quello di assegnare `30` alla variabile `y` del programma principale:



L'istruzione successiva assegna il risultato dell'espressione `w - v`, dunque `25`, alla variabile `v`:



Essendo terminato il codice di `a`, si rientra al punto di chiamata, dopo avere distrutto il record di attivazione di `a`:



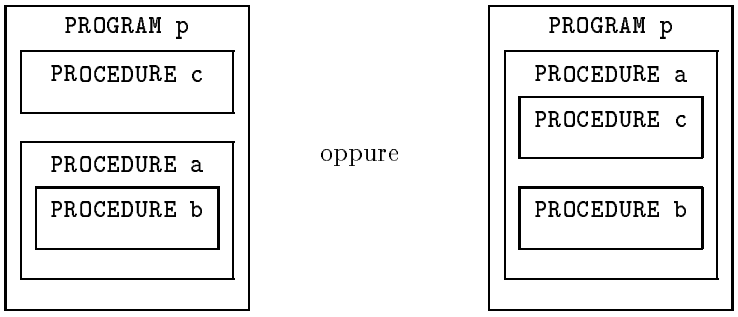
L'esecuzione riprende dall'istruzione `writeln(x, y)`, con la quale vengono stampati in output i contenuti delle due variabili `x` e `y`, cioè i numeri `5` e `30`.

Esercizio 5

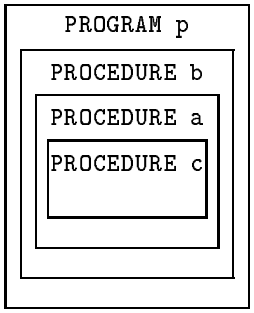
Indicare come potrebbe essere strutturato un programma `p` che contenga tre procedure `a`, `b` e `c`, e nel quale non sia utilizzata la direttiva `forward`, in modo tale che:

- la procedura `b` sia in grado di richiamare le altre due procedure, ma non sia richiamabile dal programma principale; il programma principale sia in grado di richiamare la procedura `a`; la procedura `c` *non possa* richiamare la procedura `b`.
- il programma principale possa richiamare *solo* la procedura `b`, le procedure `a` e `c` si possano richiamare a vicenda, la procedura `c` *non possa* essere richiamata dalla procedura `b`.

Nel primo caso, la procedura `a` deve essere dichiarata internamente al programma principale, per poter essere richiamata da esso. La procedura `b`, al contrario deve trovarsi internamente ad un'altra procedura (in modo che il programma principale non la possa richiamare) e in una posizione tale da potere richiamare le altre due procedure. La procedura `c` dovrà inoltre essere posizionata in modo da non potere richiamare `b`. Due possibili soluzioni sono rappresentate nella seguente figura:



Nel secondo caso il programma principale può richiamare solo `b`; pertanto, essa dovrà essere l'unica procedura dichiarata immediatamente al suo interno. Affinché `b` non possa richiamare `c`, la procedura `c` andrà posta all'interno di `a`:



Esercizio 6

Si consideri la funzione f sugli interi definita come:

$$f(x) = \begin{cases} 2 & \text{se } x \leq 1 \\ 3 * f(x-1) + f(x-2) & \text{altrimenti.} \end{cases}$$

Scrivere in Pascal una `FUNCTION f` ricorsiva per il calcolo di f.

Indicare il valore di $f(4)$. Utilizzando la ricorsione e una selezione, è possibile scrivere la seguente `FUNCTION` che calcola il valore di f:

```
FUNCTION f (x: integer): integer;
BEGIN
   IF x <= 1 THEN
      f := 2
   ELSE f := 3 * f(x - 1) + f(x - 2)
END;
```

Dalla definizione si ha: $f(4) = 3 * f(3) + f(2)$, $f(3) = 3 * f(2) + f(1)$, $f(2) = 3 * f(1) + f(0)$, $f(0) = f(1) = 2$. Sostituendo via via nelle espressioni precedenti, si ottiene: $f(2) = 8$, $f(3) = 26$ e, infine, $f(4) = 86$.