Soluzione del tema d'esame del 14 aprile 2000

Esercizio 1

Dopo avere dichiarato il tipo *lista di caratteri*, scrivere in Pascal una `PROCEDURE` che riceva come parametro il puntatore a una lista di caratteri e rimuova da tale lista il primo nodo contenente una lettera maiuscola. Ad esempio, se la lista contiene inizialmente i caratteri `'2' 'p' 'A' '&' '1'`, dopo l'esecuzione della `PROCEDURE` la lista dovrà contenere `'2' 'p' '&' '1'`.

Il tipo *lista di caratteri* può essere realizzato in Pascal utilizzando il seguente `tipolista`:

```
TYPE
   tipolista = ^nodolista;
   nodolista = RECORD
         info: char;
         pros: tipolista
      END;
```

La procedura richiesta deve ricevere come parametro il puntatore ad una lista e deve modificare la struttura della lista stessa, eliminandone un nodo. Dunque il parametro dovrà essere passato per riferimento. Possiamo pertanto definire la seguente intestazione per la procedura:

```
PROCEDURE eliminamaiuscola (VAR l: tipolista);
```

La procedura può utilizzare sia un procedimento iterativo che un procedimento ricorsivo. Sviluppiamo prima di tutto una soluzione ricorsiva. A tale scopo osserviamo che se la lista è vuota non deve essere fatta alcuna modifica, mentre se la lista è formata da un elemento, seguito da una lista, si può utilizzare il seguente schema:

```
IF il primo nodo contiene una maiuscola
   THEN cancellalo
   ELSE elabora ricorsivamente la lista che segue il primo nodo
```

Il codice completo della procedura, costruita in base allo schema precedente, è:

```
PROCEDURE eliminamaiuscola (VAR l: tipolista);

{elimina la prima lettera maiuscola presente nella lista passata come parametro}

BEGIN
   IF l <> NIL THEN
      IF l^.info IN ['A'..'Z']
         THEN l := l^.pros
         ELSE eliminamaiuscola(l^.pros)
END;
```

Al fine di rendere disponibile l'area occupata dal nodo eliminato dalla lista, è opportuno utilizzare la procedura `dispose`. A tale scopo, prima di modificare il puntatore `l` al nodo da cancellare, lo si memorizza in un puntatore `p`, al quale si applica poi `dispose`:

```
PROCEDURE eliminamaiuscola (VAR l: tipolista);

{elimina la prima lettera maiuscola presente nella lista passata come parametro}

   VAR
      p: tipolista;

BEGIN
   IF l <> NIL THEN
      IF l^.info IN ['A'..'Z'] THEN
         BEGIN
            p := l;
            l := l^.pros;
            dispose(p)
         END
      ELSE
         eliminamaiuscola(l^.pros)
END;
```

Si osservi che con una semplice modifica al ramo `THEN` è possibile ottenere una procedura per la cancellazione di *tutte* le lettere maiuscole presenti in una lista.

Sviluppiamo ora una soluzione iterativa. In una prima fase, utilizzando un puntatore ausiliario `p`, scandiamo la lista alla ricerca del nodo da eliminare. Se questo viene trovato, effettuiamo la cancellazione:

```
PROCEDURE eliminamaiuscola (VAR l: tipolista);

   VAR
      p: tipolista;
      trovato: boolean;

BEGIN
   trovato := false;
   p := l;
   WHILE (p <> NIL) AND NOT trovato DO
      IF p^.info IN ['A'..'Z']
         THEN trovato := true
         ELSE p := p^.pros;
```

```
   IF trovato THEN elimina dalla lista il nodo puntato da p
END;
```

Per eliminare dalla lista il nodo puntato da `p`, è necessario disporre di un puntatore `q` al nodo *che precede* il nodo da eliminare (per maggiori dettagli si veda la Lezione 20). Pertanto, nella prima fase, introduciamo questo puntatore, che viene spostato in modo da trovarsi sempre sul nodo che precede il nodo puntato da `p`. Per effettuare la cancellazione, distinguiamo il caso in cui il nodo da cancellare sia il primo della lista dal caso il cui il nodo sia uno dei successivi. Nel primo caso, per effettuare la cancellazione si modifica direttamente il puntatore iniziale `l`, negli altri casi, si modifica il campo `pros` del nodo che precede il nodo da cancellare:

```
PROCEDURE eliminamaiuscola (VAR l: tipolista);

{elimina la prima lettera maiuscola presente nella lista passata come parametro}

   VAR
      p, q: tipolista;
      trovato: boolean;

BEGIN
   trovato := false;
   p := l;
   q := NIL;
   WHILE (p <> NIL) AND NOT trovato DO
      IF p^.info IN ['A'..'Z'] THEN
         trovato := true
      ELSE
         BEGIN
            q := p;
            p := p^.pros
         END;
   IF trovato THEN
      BEGIN
         IF q = NIL THEN
            l := l^.pros
         ELSE
            q^.pros := p^.pros;
         dispose(p)
      END
END;
```

Esercizio 2

Siano `x` e `y` due variabili di tipo `real`. Si esprimano in linguaggio Pascal le condizioni indicate tra le parole `IF` e `THEN`:

- `IF` `x` e `y` hanno lo stesso segno `THEN` ...
- `IF` il quadrato di `x` è maggiore di `y` `THEN` ...
- `IF` la parte intera di `x` è pari `THEN` ...
- `IF` il valore contenuto in `x` è intero `THEN` ...

Un semplice modo per verificare che `x` ed `y` abbiano lo stesso segno è quello di controllare che il loro prodotto sia positivo. Pertanto la prima condizione può essere espressa come `x * y > 0`.

La seconda condizione può essere scritta come `x * x > y`.

Per la terza condizione è necessario ricavare la parte intera di `x`, mediante la funzione `trunc` (che restituisce un valore di tipo `integer`), dividerla per due, e controllare che il resto sia nullo. Pertanto la condizione si può scrivere come `trunc(x) MOD 2 = 0`.

Infine, per la quarta condizione, si può controllare che la parte intera di `x` coincida con `x`: `trunc(x) = x`.

Esercizio 3

Scrivere l'output prodotto da ciascuno dei seguenti programmi.

```
PROGRAM p1 (output);
TYPE punt = ^integer;
VAR r: punt;
   PROCEDURE s (q: punt; VAR p: punt);
   VAR x: integer;
   BEGIN
      x := q^;
      new(q);
      q^ := 2 * x;
      p^ := q^ + p^
   END;
BEGIN
   new(r);
   r^ := 4;
   s(r, r);
   writeln(r^)
END.
```

```
PROGRAM p2 (output);
TYPE punt = ^integer;
VAR r: punt;
   PROCEDURE s (VAR q: punt; VAR p: punt);
   VAR x: integer;
   BEGIN
      x := q^;
      new(q);
      q^ := 2 * x;
      p^ := q^ + p^
   END;
BEGIN
   new(r);
   r^ := 4;
   s(r, r);
   writeln(r^)
END.
```

```
PROGRAM p3 (output);
TYPE punt = ^integer;
VAR r: punt;
   PROCEDURE s (VAR q: punt; p: punt);
   VAR x: integer;
   BEGIN
      x := q^;
      new(q);
      q^ := 2 * x;
      p^ := q^ + p^
   END;
BEGIN
   new(r);
   r^ := 4;
   s(r, r);
   writeln(r^)
END.
```

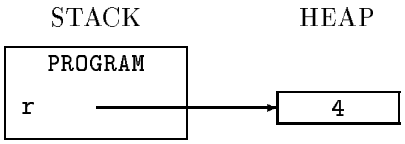
```
PROGRAM p4 (output);
TYPE punt = ^integer;
VAR r: punt;
   PROCEDURE s (q: punt; p: punt);
   VAR x: integer;
   BEGIN
      x := q^;
      new(q);
      q^ := 2 * x;
      p^ := q^ + p^
   END;
BEGIN
   new(r);
   r^ := 4;
   s(r, r);
   writeln(r^)
END.
```

I quattro programmi si differenziano esclusivamente per le modalità utilizzate nel passaggio dei parametri alla `PROCEDURE s`. Per determinare l'output prodotto, descriveremo il contenuto dello stack e dello heap durante l'esecuzione dei programmi.

Prima di tutto, osserviamo che dopo l'esecuzione delle istruzioni:

```
new(r);
r^ := 4
```

il contenuto della memoria sarà:

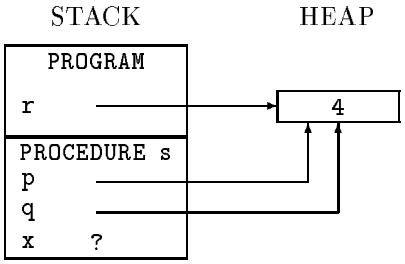


A questo punto si ha la chiamata della `PROCEDURE s`. Pertanto, come si è visto nella Lezione 15, occorre:

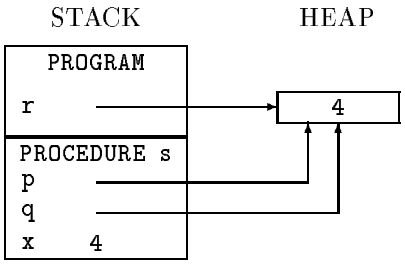
1. memorizzare nel record di attivazione del programma principale il punto di rientro (che, per semplicità, essendoci un'unica chiamata, non verrà indicato nelle figure successive),
2. creare sullo stack il record di attivazione della procedura, nel quale verranno memorizzati i parametri `q` e `p`, e la variabile locale `x`,
3. effettuare il passaggio dei parametri.

In base alle modalità di passaggio dei parametri utilizzate, si avranno differenti situazioni.

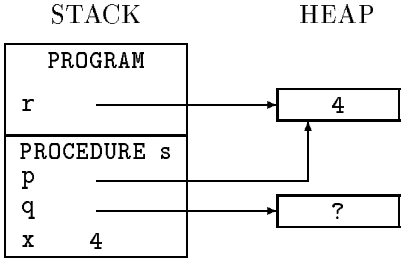
Consideriamo, per primo, il caso in cui *entrambi i parametri siano passati per valore* (`PROGRAM p4`). In `q` e `p` vengono copiati i rispettivi parametri attuali, cioè il puntatore `r`. Pertanto `q` e `p` punteranno alla stessa area di memoria a cui punta `r`:



L'assegnamento successivo ad `x` ha il seguente effetto:

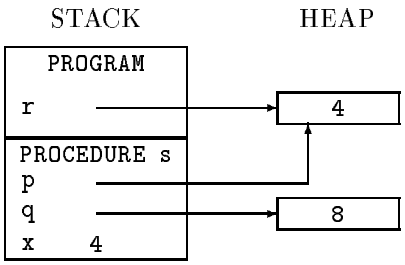


L'istruzione seguente è una chiamata alla procedura `new`, con parametro `q`. Pertanto, verrà creata nello heap una nuova variabile dinamica (di tipo `integer`) a cui verrà fatta puntare la variabile `q`.

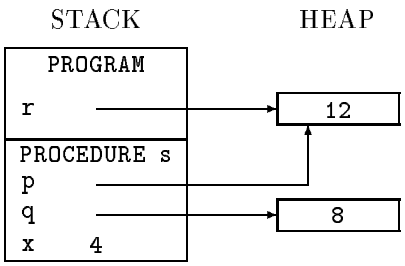


La procedura si conclude con due istruzioni di assegnamento:

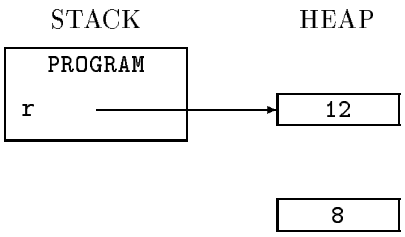
```
q^ := 2 * x
```




```
p^ := q^ + p^
```

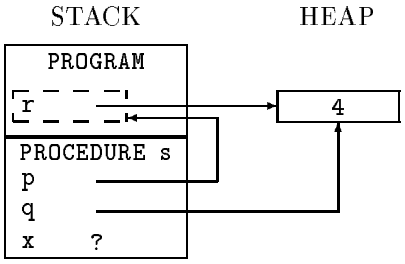


Al rientro dalla procedura, il record di attivazione di `s` viene distrutto. Pertanto il contenuto della memoria risulta:



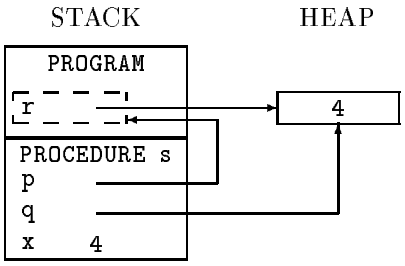
Dunque, l'output prodotto dall'istruzione `writeln` del programma principale è `12`.

Analizziamo ora la situazione in cui *`q` è passato per valore e `p` per riferimento* (`PROGRAM p1`). In questo caso, durante l'esecuzione della procedura, tutte le operazioni effettuate su `p`, verranno in realtà effettuate su `r`. Il contenuto della memoria dopo la chiamata è rappresentato nella figura seguente, in cui la freccia da `p` al riquadro tratteggiato contenente `r`, indica che `p` è un riferimento ad `r`:

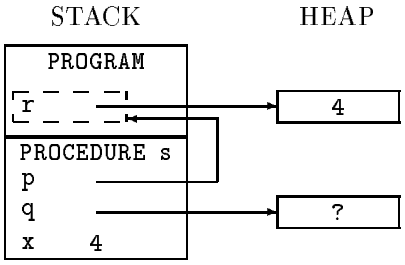


Gli effetti delle istruzioni successive, sono rappresentati qui di seguito:

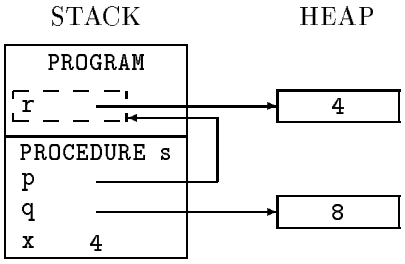
```
x := q^
```



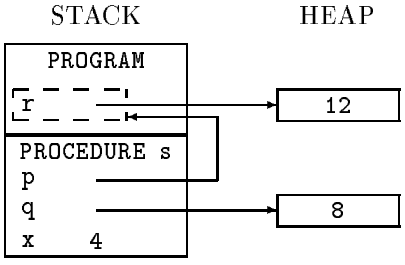

```
new(q)
```



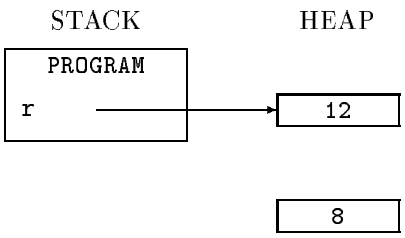

```
q^ := 2 * x
```




```
p^ := q^ + p^
```

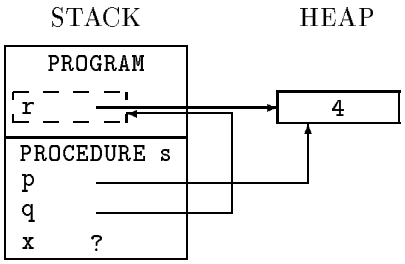


Poiché `p` in realtà è `r`, l'ultima istruzione modifica l'area puntata da `r`. Al rientro dalla procedura, il contenuto della memoria è:

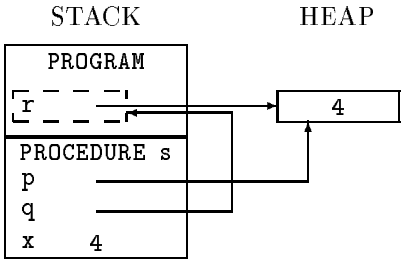


L'output prodotto è dunque `12`.

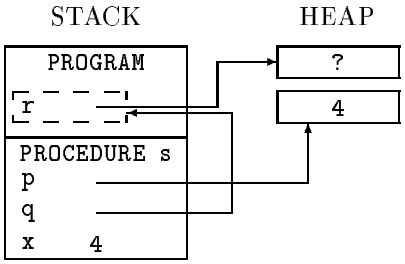
Consideriamo ora il caso il cui *il parametro* `q` *sia passato per riferimento e il parametro* `p` *per valore* (`PROGRAM p3`). In questo caso, `p` conterrà una copia del valore di `r` al momento della chiamata, punterà cioè all'area a cui, al momento della chiamata, punta `r`. Il parametro `q` sarà invece un riferimento a `r`: ogni operazione effettuata utilizzando `q`, verrà in realtà effettuata utilizzando `r`.



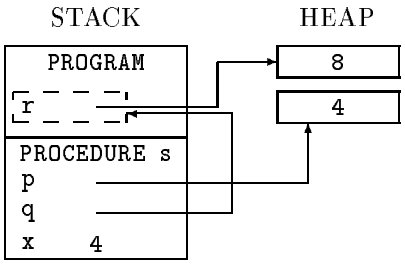

```
x := q^
```



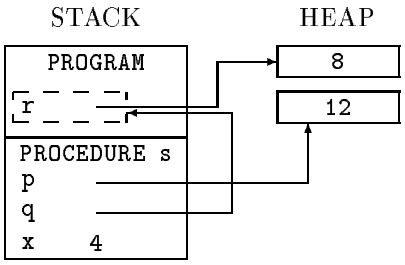
Poiché `q` è un riferimento a `r`, l'esecuzione dell'istruzione `new(q)` avrà come effetto quello di creare una variabile dinamica di tipo `integer` nello heap, e di far puntare `r` a tale variabile. Il puntatore `p` non subisce alcuna modifica:



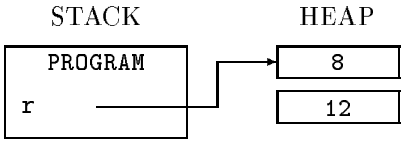

```
q^ := 2 * x
```



`p^ := q^ + p^`

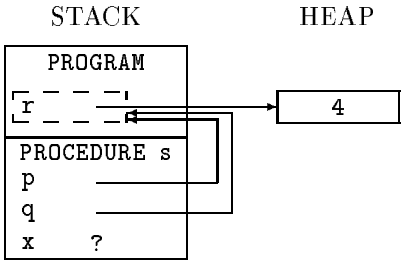


Al rientro, la memoria contiene:

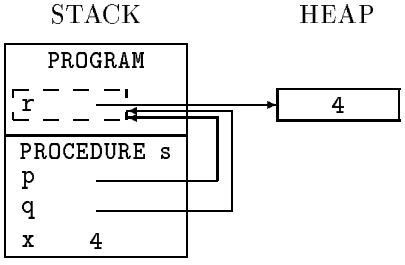


L'output prodotto è il valore contenuto nell'area puntata da `r`, cioè `8`.

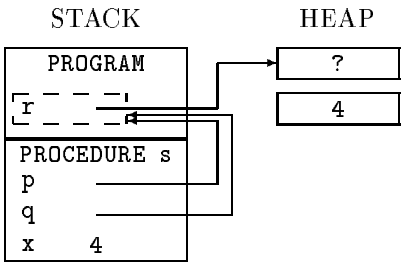
Consideriamo, infine, il caso il cui *entrambi i parametri siano passati per riferimento* (`PROGRAM p2`). Poiché il parametro attuale utilizzato nella chiamata è sempre `r`, ogni operazione contenente `p` o `q` riguarderà in realtà `r`:



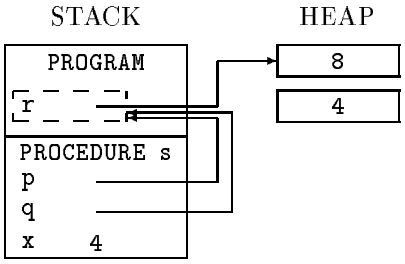
`x := q^`



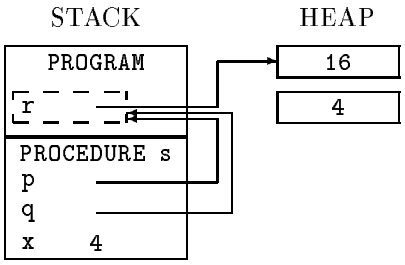
Come nel caso precedente, l'istruzione `new(q)` modifica il puntatore `r`. Poiché anche `p` denota la stessa variabile `r`, il comportamento delle istruzioni successive risulterà in questo caso differente:



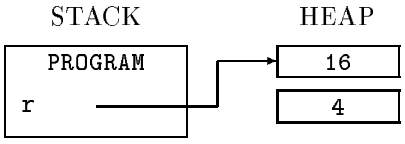

```
q^ := 2 * x
```




```
p^ := q^ + p^
```



Il contenuto della memoria al rientro è:



Dunque, l'output prodotto è `16`.