Lezione 20
14–15 dicembre 1999

Argomenti trattati
- Sottoprogrammi iterativi e ricorsivi per il trattamento delle liste.

20.1 Inserimento in una lista ordinata

Nella lezione precedente, abbiamo scritto una `FUNCTION trova` per individuare, in una *lista ordinata*, il puntatore ad un nodo contenente un valore dato. Chiaramente, usando la procedura di inserimento all'inizio, costruita in precedenza, la lista ottenuta non sarà in generale ordinata. Per la costruzione di liste ordinate è necessario disporre di una procedura d'inserimento differente, che dato il puntatore ad una lista ordinata e un intero x, inserisca x nella posizione opportuna della lista ordinata, al fine di ottenere una lista ancora ordinata. Ad esempio, data la lista contenente `11 15 20 25`, inserendo `18` si dovrà ottenere la lista `11 15 18 20 25`.

La procedura avrà la seguente intestazione:

```
PROCEDURE InserOrd (VAR l: tipolista; x: integer);
```

Si noti che, dovendo effettuare modifiche, il puntatore alla lista viene passato per riferimento.

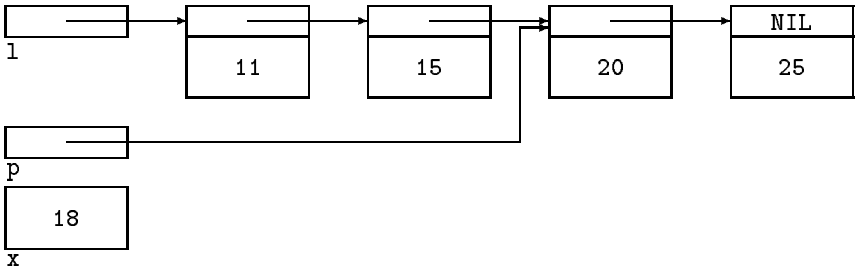
La procedura è composta da due parti fondamentali:

1. ricerca della posizione dove va effettuato l'inserimento;
2. creazione e inserimento del nuovo nodo.

La fase di ricerca è analoga a quella utilizzata nella `FUNCTION trova`. L'unica differenza è dovuta al fatto che questa volta il puntatore `l` è passato per riferimento: per evitare di perdere il puntatore iniziale alla lista, al posto di `l` utilizziamo per la scansione un puntatore ausiliario `p`:

```
finito := false;
p := l;
WHILE (p <> NIL) AND NOT finito DO
   IF p^.info >= x THEN
      {la posizione a cui effettuare l'inserimento e' stata determinata}
      finito := true
   ELSE
      p := p^.pros
```

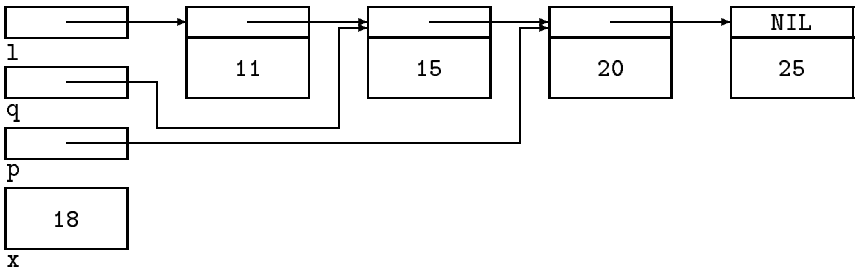
All'uscita da questo ciclo, la variabile `p` contiene un puntatore al nodo *prima* del quale effettuare l'inserimento (o `NIL` nel caso l'inserimento vada effettuato alla fine della lista).



Per effettuare l'inserimento è necessario modificare il campo `pros` del nodo che *precede* il nodo puntato da `p`. Per disporre di un puntatore a tale nodo, introduciamo un secondo puntatore ausiliario `q`, inizializzato a `NIL`, che ad ogni passo della fase di ricerca punterà al nodo che precede il nodo puntato da `p`. Introducendo `q`, la ricerca viene riscritta come:

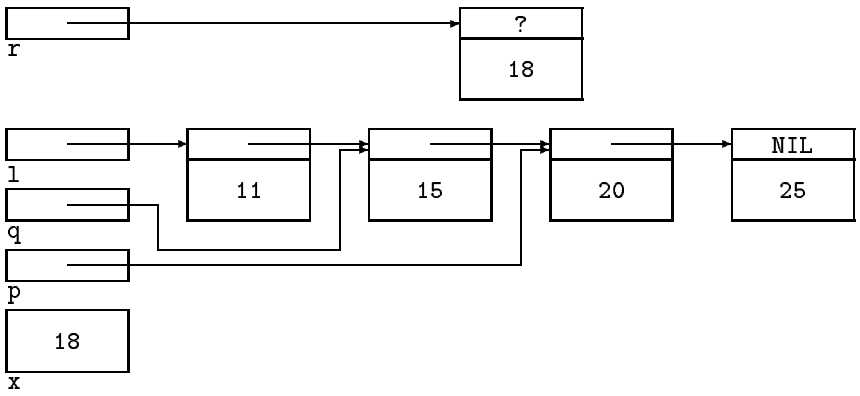
```
finito := false;
p := l;
q := NIL;
WHILE (p <> NIL) AND NOT finito DO
   IF p^.info >= x THEN
      {la posizione a cui effettuare l'inserimento e' stata determinata}
      finito := true
   ELSE
      BEGIN
         {spostamento in avanti dei puntatori}
         q := p;
         p := p^.pros
      END
```

In questo caso, dopo l'esecuzione del ciclo, si otterrà:



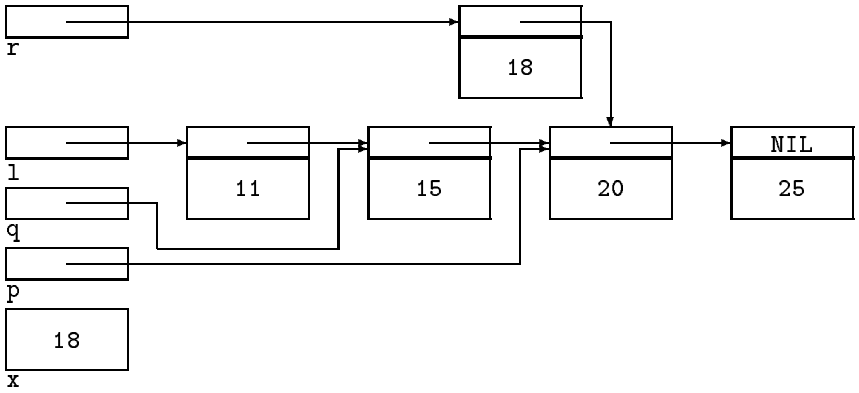
Terminata la fase di ricerca, occorre effettuare la creazione e l'inserimento del nodo. Per la creazione del nodo, utilizziamo un terzo puntatore ausiliario `r` e scriviamo:

```
new(r);
r^.info := x
```

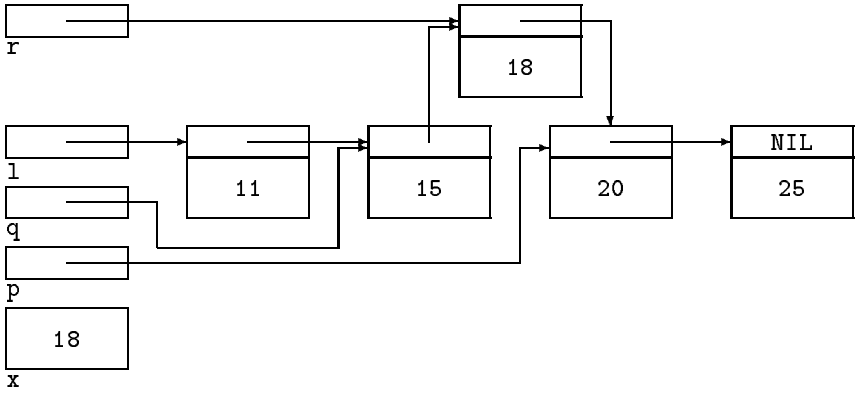


Per inserire il nuovo nodo *prima* del nodo puntato da `p` e *dopo* il nodo puntato da `q` occorre effettuare le seguenti operazioni:

1. agganciare la parte di lista puntata da `p` dopo il nuovo nodo (`r^.pros := p`);



2. agganciare il nuovo nodo dopo il nodo puntato da `q` (`q^.pros := r`)



   Se l'inserimento avviene immediatamente all'inizio della lista (dunque se `q = NIL`) occorre invece fare in modo che il puntatore iniziale della lista, cioè la variabile `l`, punti al nuovo nodo (`l := r`).

La seconda fase della procedura può dunque essere scritta come:

```
{creazione del nodo}
new(r);
r^.info := x;

{inserimento del nodo nella lista, tra i nodi puntati da q e p}
r^.pros := p;
IF q = NIL THEN {inserimento all'inizio lista}
   l := r
ELSE {inserimento all'interno della lista, dopo il nodo q^}
   q^.pros := r
```

La procedura completa è:

```
PROCEDURE InserOrd (VAR l: tipolista; x: integer);

{Inserisce il valore di x nella lista puntata da l ordinata in modo non decrescente}
{versione iterativa}

   VAR
      finito: boolean;
      p, q, r: tipolista;

BEGIN {InserOrd}
   {ricerca dalla posizione nella quale effettuare l'inserimento}
   {la ricerca viene effettuata in modo che, alla fine:}
   {p contenga il puntatore al nodo PRIMA del quale va effettuato l'inserimento}
   {q contenga il puntatore al nodo DOPO il quale va effettuato l'inserimento}

   finito := false;
   p := l;
   q := NIL;
   WHILE (p <> NIL) AND NOT finito DO
      IF p^.info >= x THEN
         {la posizione a cui effettuare l'inserimento e' stata determinata}
         finito := true
      ELSE
         BEGIN
            {spostamento in avanti dei puntatori}
            q := p;
            p := p^.pros
         END; {else}

   {creazione del nodo}
   new(r);
   r^.info := x;

   {inserimento del nodo nella lista, tra i nodi puntati da q e p}
   r^.pros := p;
   IF q = NIL THEN {inserimento all'inizio lista}
      l := r
   ELSE {inserimento all'interno della lista, dopo il nodo q^}
      q^.pros := r
END; {InserOrd}
```

20.2 Trattamento ricorsivo di liste

Osserviamo che una lista di elementi può essere definita in maniera ricorsiva come segue.

Una *lista* di elementi è:

- la lista vuota, oppure
- un elemento seguito da una lista.

Le definizioni dei tipi `tipolista` e `nodolista`, date in precedenza, e la realizzazione di liste tramite puntatori, riflettono la definizione ricorsiva di lista.

I sottoprogrammi che abbiamo sviluppato per il trattamento delle liste possono essere riscritti in maniera ricorsiva, basandosi sulla definizione ricorsiva di lista. Per costruire un sottoprogramma ricorsivo consideriamo sempre due casi fondamentali:

1. il caso di lista vuota, in cui il sottoprogramma può concludere il proprio lavoro senza chiamate ricorsive;
2. il caso di lista non vuota, in cui il sottoprogramma effettua delle elaborazioni sul primo nodo (che dipendono dalla funzionalità che deve essere svolta dal sottoprogramma), e poi si richiama ricorsivamente sulla lista che segue il primo elemento.

Ad esempio, la scrittura in `output` del contenuto di una lista può avvenire come segue:

- se la lista è vuota non occorre scrivere in `output` nulla;
- se la lista non è vuota si scrive il contenuto del primo elemento, seguito dal contenuto della lista che inizia dopo il primo elemento.

Utilizzando questo schema, possiamo scrivere la seguente procedura ricorsiva:

```
PROCEDURE ScriviLista (l: tipolista);

{Scrive in output il contenuto della lista puntata da l}
{versione ricorsiva}

BEGIN {ScriviLista}
   IF l <> NIL THEN
      BEGIN
         writeln(l^.info);
         ScriviLista(l^.pros)
      END {if}
END; {Scrivilista}
```

Volendo scrivere in `output` il contenuto di una lista partendo dall'ultimo elemento, è sufficiente osservare che, nel caso di lista non vuota, andrà scritta *prima* la parte di lista che inizia dopo il primo elemento, e *poi* il primo elemento. Dunque, è sufficiente scambiare la chiamata di `writeln` con la chiamata ricorsiva:

```
PROCEDURE ScriviAlContrario (l: tipolista);

{Scrive in output il contenuto della lista puntata da l}

BEGIN {ScriviAlContrario}
   IF l <> NIL THEN
      BEGIN
         ScriviAlContrario(l^.pros);
```

```
        writeln(l^.info)
      END {if}
END; {ScriviAlContrario}
```

Si noti che, mentre per la scrittura del contenuto di una lista, dal primo all'ultimo elemento, sia la soluzione iterativa che la soluzione ricorsiva sono estremamente semplici, per la scrittura al contrario, una soluzione iterativa risulterebbe estremamente complicata. Questo, dunque, è un esempio in cui l'uso della ricorsione è di fondamentale importanza per la scrittura di codice semplice.

Riscriviamo ora ricorsivamente la funzione per la ricerca di un elemento x in una lista (non ordinata). Analizziamo di nuovo la definizione ricorsiva.

- La lista vuota sicuramente non può contenere x, dunque in questo caso si restituisce immediatamente il puntatore `NIL`.
- Nel caso di lista non vuota, si confronta il valore x con il primo elemento: se il primo elemento contiene x, la ricerca termina restituendo il puntatore al primo elemento, altrimenti la ricerca prosegue, ricorsivamente, nella parte di lista che segue il primo elemento.

La `FUNCTION trova` può essere dunque riscritta in modo ricorsivo come:

```
FUNCTION trova (l: tipolista; x: integer): tipolista;

{Restituisce il puntatore al primo nodo della lista l contenente il }
{valore di x, se presente, oppure NIL}
{versione ricorsiva}

BEGIN {trova}
   IF l = NIL THEN
      trova := NIL
   ELSE IF l^.info = x THEN
      trova := l
   ELSE
      trova := trova(l^.pros, x)
END; {trova}
```

Nel caso la lista sia ordinata e si raggiunga un nodo contenente un valore superiore ad x, è inutile effettuare la chiamata ricorsiva, ma si può restituire immediatamente `NIL`. In tal caso, la `FUNCTION` diventa:

```
FUNCTION trova (l: tipolista; x: integer): tipolista;

{Restituisce il puntatore al primo nodo della lista l contenente il}
{valore di x, se presente, oppure NIL}
{Si suppone che la lista sia ordinata in maniera non descrescente}
{versione ricorsiva}

   VAR
      finito: boolean;

BEGIN {trova}
   IF l = NIL THEN
      trova := NIL
   ELSE IF l^.info = x THEN
```

```
      trova := l
   ELSE IF l^.info < x THEN
      trova := trova(l^.pros, x)
   ELSE
      trova := NIL
END; {trova}
```

Sfruttando la ricorsione, il codice della procedura di inserimento in una lista ordinata può essere notevolmente semplificato.

- Nel caso di lista vuota è sufficiente creare direttamente il nuovo elemento, come unico elemento della lista.
- Nel caso di lista non vuota, si confronta il nuovo elemento da inserire con il valore del primo elemento della lista. Se il nuovo elemento è minore del primo elemento, allora lo si inserisce prima del primo elemento, altrimenti lo si inserisce, mediante una chiamata ricorsiva, nella parte di lista che segue il primo elemento.

Il codice della procedura, scritta basandosi su questo schema, è:

```
PROCEDURE InserOrd (VAR l: tipolista; x: integer);

{Inserisce il valore di x nella lista puntata da l ordinata in modo non decrescente}
{versione ricorsiva}

   VAR
      p: tipolista;

BEGIN {InserOrd}
   IF l = NIL THEN
   {se la lista e' vuota, crea il primo elemento}
      BEGIN
         new(l);
         l^.info := x;
         l^.pros := NIL
      END
   {se la lista non e' vuota...}
   ELSE IF x < l^.info THEN
      {se il x e' minore del primo elemento, inserisce x all'inizio}
      BEGIN
         new(p);
         p^.info := x;
         p^.pros := l;
         l := p
      END
   ELSE
      {altrimenti inserisce x nella parte di lista che segue il primo elemento}
      InserOrd(l^.pros, x)
END; {InserOrd}
```

Per evitare la duplicazione della parte di codice relativa alla creazione del nuovo nodo e al suo collegamento alla lista, possiamo introdurre una variabile di tipo `boolean`, che indichi quando sia necessario fare l'inserimento, e spostare l'operazione di inserimento alla fine. Il codice può essere riscritto come:

```
PROCEDURE InserOrd (VAR l: tipolista; x: integer);

{Inserisce il valore di x nella lista puntata da l ordinata in modo non decrescente}
{versione ricorsiva}

   VAR
      p: tipolista;
      inserire: boolean; {indica quando effettuare l'inserimento}

BEGIN {InserOrd}
   inserire := false;
   IF l = NIL THEN
      inserire := true
   ELSE IF x < l^.info THEN
      inserire := true
   ELSE
      InserOrd(l^.pros, x);

   IF inserire THEN
      BEGIN
         new(p);
         p^.info := x;
         p^.pros := l;
         l := p
      END
END; {InserOrd}
```

20.3 Cancellazione di un elemento da una lista

Dopo avere studiato l'inserimento, la ricerca e la scansione di una lista, consideriamo ora il problema della *cancellazione* di un elemento.

Vogliamo costruire una procedura `cancella` che riceva come parametro il puntatore ad una lista contenente numeri interi e un valore intero `x`, ed elimini dalla lista la prima occorrenza di `x`, se presente. Ad esempio, data la lista contente i valori `8 1 11 3 11`, cancellando `11` il contenuto della lista diventerà `8 1 3 11`.

La procedura avrà la seguente intestazione:

```
PROCEDURE cancella (VAR l: tipolista; x: integer);
```

in cui, dovendo modificare la lista, il puntatore `l` è passato per riferimento.

Sviluppiamo prima di tutto la procedura in maniera iterativa. La struttura della procedura è simile a quella di inserimento in una lista ordinata:

1. ricerca della posizione dove effettuare la cancellazione;
2. rimozione dell'elemento (da effettuare solo nel caso l'elemento sia stato trovato).

Ad alto livello lo schema della procedura sarà dunque:

```
cerca l'elemento da cancellare e determinane la posizione
IF l'elemento e' stato trovato THEN eliminalo
```

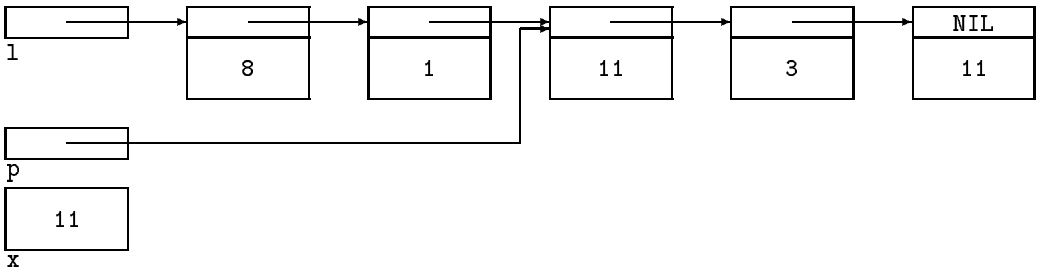
La ricerca può essere effettuata utilizzando un puntantore ausiliario `p` e scrivendo:

```
trovato := false;
p := l;
WHILE (p <> NIL) AND NOT trovato DO
   IF p^.info = x THEN
      {la posizione a cui effettuare la cancellazione e' stata determinata}
      trovato := true
   ELSE
         p := p^.pros
```

Al termine dell'esecuzione del ciclo si possono verificare le seguenti situazioni:

- la variabile `trovato` contiene `false`, il puntatore `p` contiene `NIL`, e dunque l'elemento non è stato trovato; in questo caso la procedura non deve fare altre operazioni;
- la variabile `trovato` contiene `true` e il puntatore `p` punta al nodo che deve essere eliminato dalla lista.

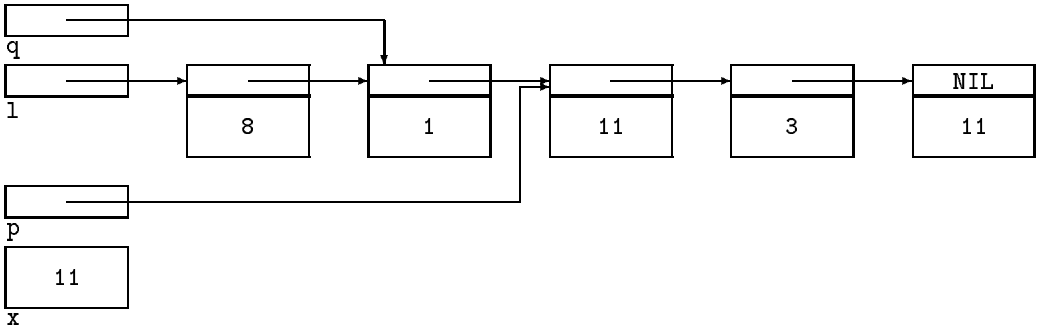
Analizziamo in dettaglio il secondo caso:



Per sganciare il nodo puntato da `p` dalla lista, è necessario collegare il nodo che precede quello puntato da `p` al nodo che segue quello puntato da `p`. Per poter modificare il nodo che precede quello puntato da `p` è necessario disporre di un puntatore ad esso. Come abbiamo già fatto nel caso di inserimento in una lista ordinata, riscriviamo la parte di ricerca introducendo un secondo puntatore ausiliario `q`, inizializzato a `NIL`, che ad ogni passo della fase di ricerca punti al nodo che precede il nodo puntato da `p`:

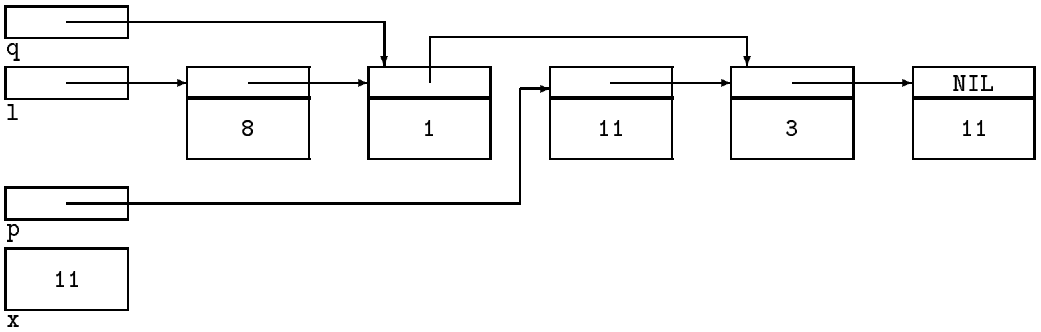
```
trovato := false;
p := l;
q := NIL;
WHILE (p <> NIL) AND NOT trovato DO
   IF p^.info = x THEN
      {la posizione a cui effettuare la cancellazione e' stata determinata}
      trovato := true
   ELSE
      BEGIN
         {spostamento in avanti dei puntatori}
         q := p;
         p := p^.pros
      END {else}
```

In questo caso, dopo l'esecuzione del codice, la situazione sarà quella rappresentata nella seguente figura:



A questo punto, per scollegare l'elemento dalla lista, è sufficiente fare in modo che il nodo puntato da `q` sia seguito dalla parte di lista successiva al nodo da eliminare. A tale scopo basta scrivere l'assegnamento:

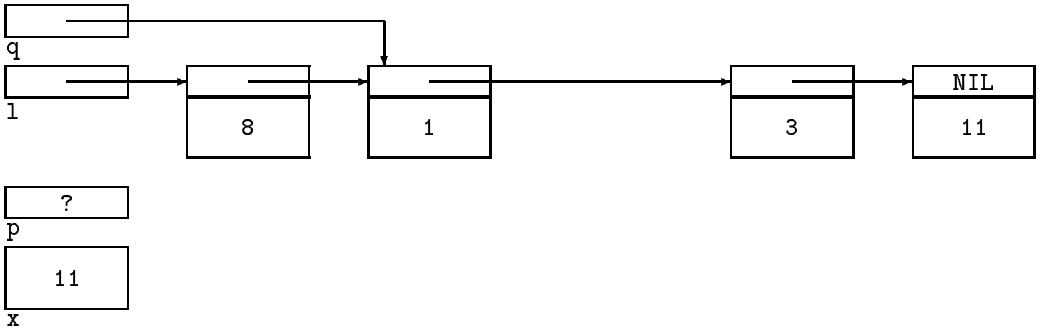
```
q^.pros := p^.pros
```



Infine, per rendere disponibile l'area di memoria occupata dal nodo che è stato rimosso dalla lista, si utilizza la procedura `dispose`

```
dispose(p)
```

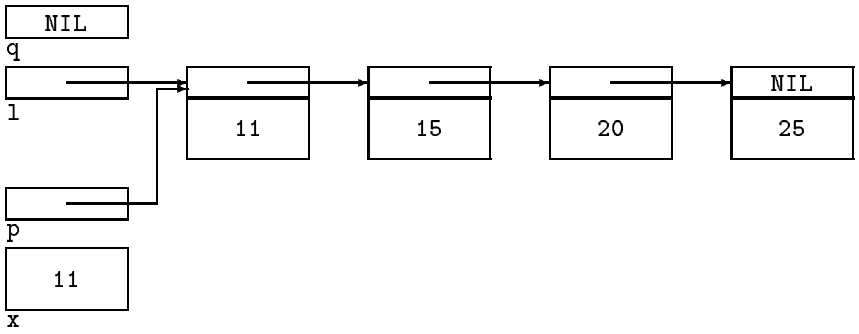
che, oltre a rilasciare l'area di memoria, rende indefinito il valore del puntatore `p`:



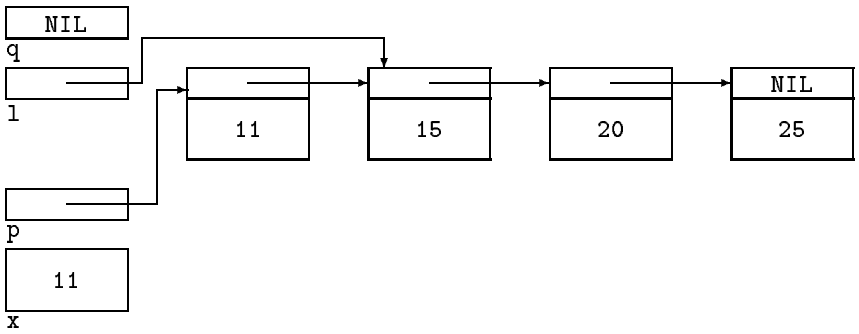
Analizziamo ora due casi particolari.

- Se il nodo da eliminare è l'ultimo della lista, cioè se `p^.pros` contiene `NIL`, l'effetto dell'assegnamento `q^.pros := p^.pros` è di assegnare `NIL` a `q^.pros`, facendo dunque terminare la lista sul nodo puntato da `q`.

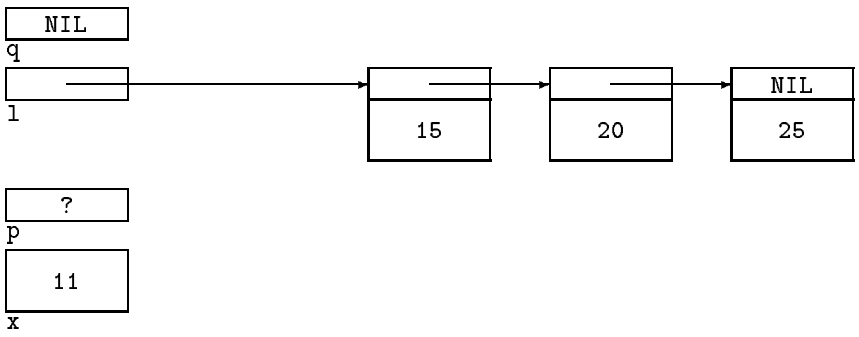
- Se il nodo da eliminare è il primo della lista, dopo la ricerca `q` punta a `NIL` ed entrambi i puntatori `p` e `l` puntano al primo elemento della lista. Ad esempio, volendo cancellare `11` dalla lista contente `11 15 20 25`, dopo la ricerca i puntatori saranno organizzati come nella seguente figura:



In questo caso, per sganciare il nodo dalla lista è sufficiente far puntare `l` al secondo nodo della lista, mediante l'assegnamento `l := l^.pros`:



Mediante `dispose(p)` si può quindi rilasciare l'area di memoria utilizzata per il nodo che è stato rimosso:



Riassumendo, la fase di cancellazione viene effettuata se la lista contiene un nodo con il valore cercato, cioè se, all'uscita dal ciclo, la variabile `trovato` contiene valore `true`. In questo caso si possono verificare due situazioni differenti:

- il nodo da eliminare è il primo: si modifica direttamente il puntatore `l`;
- il nodo da eliminare è successivo al primo: si modifica il campo `pros` del nodo che precede quello da eliminare.

In entrambi casi occorre poi rilasciare l'area di memoria che conteneva il nodo eliminato:

```
IF trovato THEN
   BEGIN
      IF q = NIL THEN
         {se il nodo da cancellare e' il primo}
         l := l^.pros
      ELSE
         q^.pros := p^.pros;
      dispose(p)
   END
```

Il testo completo della procedura è dunque:

```
PROCEDURE cancella (VAR l: tipolista; x: integer);

{Cancella il primo nodo contenente x, se presente, dalla lista puntata da l}
{versione iterativa}

   VAR
      p, q: tipolista;
      trovato: boolean;

BEGIN {cancella}
   {ricerca della posizione nella quale effettuare la cancellazione}
   {la ricerca viene effettuata in modo che, alla fine:}
   {p punti al nodo SUCCESSIVO a quello da cancellare}
   {q punti al nodo da cancellare, o contenga NIL se va cancellata la radice}
   {trovato contenga false se il nodo non e' presente}

   trovato := false;
   p := l;
   q := NIL;
   WHILE (p <> NIL) AND NOT trovato DO
      IF p^.info = x THEN
         {la posizione a cui effettuare la cancellazione e' stata determinata}
         trovato := true
      ELSE
         BEGIN
            {spostamento in avanti dei puntatori}
            q := p;
            p := p^.pros
         END; {else}

   {elimina dalla lista l'elemento puntato da p}
   IF trovato THEN
      BEGIN
         IF q = NIL THEN
            {se il nodo da cancellare e' il primo}
            l := l^.pros
         ELSE
            q^.pros := p^.pros;
         dispose(p)
```

```
      END
END; {cancella}
```

Sviluppiamo ora la procedura di cancellazione in maniera *ricorsiva*.

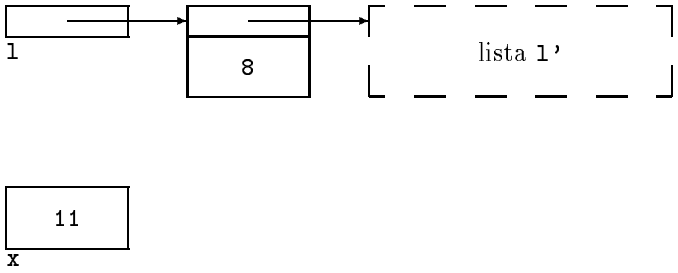
Nel caso di *lista vuota* non occorre effettuare alcuna operazione; nel caso di *lista non vuota*, cioè formata da un nodo seguito da una lista, si possono verificare le seguenti situazioni:

- il nodo contiene il valore da cancellare: in questo caso è sufficiente rimuovere il nodo dalla lista;
- il nodo non contiene il valore da cancellare: in questo caso si procede ricorsivamente sulla parte di lista che segue il primo nodo.

In altre parole, l'operazione `cancella x dalla lista l` può essere espressa come:

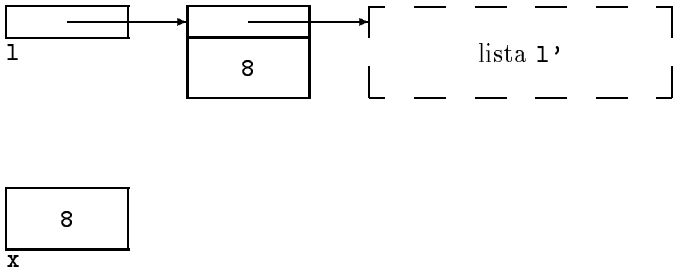
```
IF l non e' vuota THEN
   IF il primo nodo di l contiene x
      THEN elimina da l il primo nodo
      ELSE cancella x dalla lista l' che segue il primo elemento
```

La situazione in cui il contenuto del primo della lista è diverso dal valore da cancellare, è rappresentata nella seguente figura:

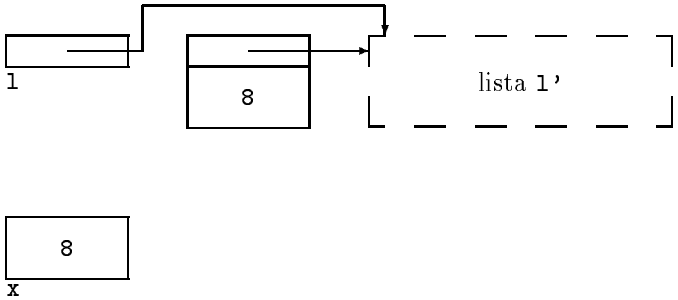


In questo caso è sufficiente effettuare ricorsivamente la cancellazione nella lista `l'`, cioè la lista puntata da `l^.pros`.

Nel caso in cui il nodo da cancellare sia il primo della lista, si ha la seguente situazione:



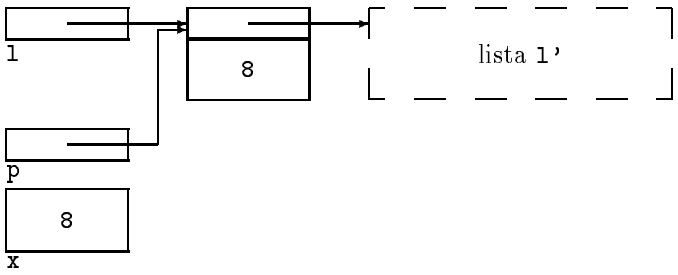
Per eliminare il primo nodo, è sufficiente far puntare `l` alla lista `l'`, cioè copiare in `l` il puntatore `l^.pros`:



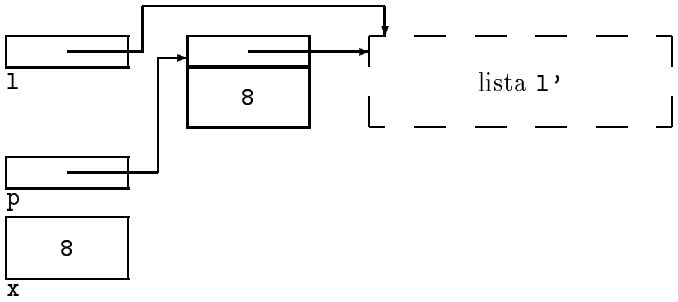
È immediato riscrivere le precedenti istruzioni in Pascal, ottenendo la seguente procedura ricorsiva:

```
PROCEDURE cancella (VAR l: tipolista; x: integer);

{Cancella il primo nodo contenente x, se presente, dalla lista puntata da l}
{versione ricorsiva}

BEGIN {cancella}
   IF l <> NIL THEN
      IF l^.info = x THEN
         l := l^.pros
      ELSE
         cancella(l^.pros, x)
END; {cancella}
```

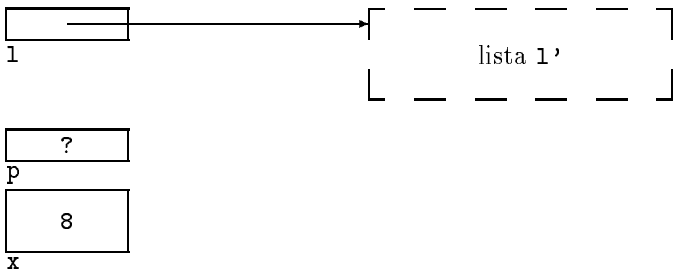
La procedura appena scritta elimina un elemento dalla lista, ma *non rilascia* l'area di memoria che era utilizzata dal nodo. Per effettuare questa operazione, introduciamo un puntatore ausiliario `p`. Prima di spostare il puntatore `l`, per eliminare il nodo dalla lista, si fa puntare `p` al nodo da eliminare, assegnando a `p` il valore di `l`:



Subito dopo, si sposta il puntatore `l` sulla lista `l'` che segue il primo nodo:



Infine, mediante `dispose(p)`, si rilascia l'area di memoria puntata da `p`:



La procedura può dunque essere riscritta come:

```
PROCEDURE cancella (VAR l: tipolista; x: integer);

{Cancella il primo nodo contenente x, se presente, dalla lista puntata da l}
{versione ricorsiva}

   VAR
      p: tipolista;

BEGIN {cancella}
   IF l <> NIL THEN
      IF l^.info = x THEN
         BEGIN
            p := l;
            l := l^.pros;
            dispose(p)
         END
      ELSE
         cancella(l^.pros, x)
END; {cancella}
```

Esercizi

Nei programmi presentati negli esercizi seguenti si fa riferimento alle seguenti dichiarazioni di tipo:

```
TYPE
   tipolista = ^nodolista;       {puntatore a un nodo della lista}
   nodolista = RECORD            {nodo della lista}
                  info: integer; {informazione contenuta nel nodo}
                  pros: tipolista{puntatore all'elemento successivo della lista}
               END;
```

1. Simulare "manualmente" l'esecuzione dei sottoprogrammi presentati, evidenziando l'evoluzione dello stack e dello heap.

2. Scrivere (in versione ricorsiva e iterativa) una procedura che, ricevendo come parametro il puntatore al primo elemento di una lista di interi, trasformi la lista raddoppiando il valore contenuto in ciascun nodo. Se ad esempio la lista contiene `25 8 10 15`, dopo l'esecuzione della procedura dovrà contenere `50 16 20 30`.

3. Scrivere, in versione iterativa e in versione ricorsiva, una procedura che riceva come parametro il puntatore al primo elemento di una lista di interi e un intero x, ed elimini dalla lista il primo nodo contenente x, se presente.

4. La `PROCEDURE InserOrd`, nelle tre versioni presentate, inserisce un elemento anche se già presente, creando un doppione. Se ad esempio la lista contiene `5 8 10 25`, e si vuole inserire `8`, dopo l'esecuzione della procedura la lista conterrà `5 8 8 10 25`. Scrivere, in versione iterativa e ricorsiva, una nuova procedura per l'inserimento di un elemento in una lista ordinata che, nel caso l'elemento sia già presente, non effettui l'inserimento.

5. Scrivere un programma che legga da `input` una sequenza di numeri interi, terminata da `0`, e la scriva in uscita ordinata in maniera crescente. Si utilizzino le liste.

6. Scrivere, in versione iterativa e ricorsiva, una `FUNCTION` che ricevendo come parametro il puntatore al primo elemento di una lista di interi, restituisca la somma dei valori contenuti nella lista.

7. Scrivere, in versione iterativa e ricorsiva, una `FUNCTION` che ricevendo come parametro il puntatore al primo elemento di una lista di interi, restituisca la media dei valori contenuti nella lista.

8. Costruire (in versione ricorsiva e iterativa) una procedura che data una lista di caratteri, visualizzi la stringa formata dai caratteri contenuti nella lista, presi in ordine dal primo all'ultimo.

9. Costruire una procedura ricorsiva che data una lista di caratteri visualizzi la stringa formata dai caratteri contenuti nella lista, presi in ordine dall'ultimo al primo.

10. Costruire (in versione ricorsiva e iterativa) una funzione che ricevendo come parametro un puntatore a una lista di interi, restituisca un puntatore al nodo contenente il valore minimo (o `NIL` nel caso di lista vuota).

11. Le procedure di cancellazione che abbiamo presentato, eliminano da una lista la prima occorrenza dell'elemento specificato. Se ad esempio la lista contiene `8 4 7 5 4 8 2` e si chiede di cancellare `4`, la lista risultante conterrà `8 7 5 4 8 2`. Scrivere una procedura iterativa e una procedura ricorsiva per la cancellazione da una lista di *tutte* le occorrenze dell'elemento specificato.

12. Riscrivere la procedura di cancellazione di un elemento da una lista, sia in forma iterativa che in forma ricorsiva, nell'ipotesi che la lista sia ordinata in maniera non decrescente. Scrivere sia una versione per la cancellazione della prima occorrenza dell'elemento, che una versione per la cancellazione di tutte le occorrenze dell'elemento.

13. Per ognuna delle seguenti procedure, individuare l'output prodotto, nel caso venga passato come parametro il puntatore a una lista contenente gli interi `4 10 2 5`, evidenziando eventuali errori o possibili errori.

    - ```
      PROCEDURE p1 (l: tipolista);
      BEGIN
         WHILE l <> NIL DO
         BEGIN
            writeln(l^.info);
            l := l^.pros
         END
      END;
      ```
    - ```
      PROCEDURE p2 (l: tipolista);
      BEGIN
         WHILE l <> NIL DO
         BEGIN
            l := l^.pros;
            writeln(l^.info)
         END
      END;
      ```
    - ```
      PROCEDURE p3 (l: tipolista);
      BEGIN
         IF l <> NIL THEN
         BEGIN
      ```
      ```

```
      writeln(l^.info);
      p3(l^.pros)
   END
END;
```

- ```
  PROCEDURE p4 (l: tipolista);
  BEGIN
     IF l <> NIL THEN
     BEGIN
       p4(l^.pros);
       writeln(l^.info)
     END
  END;
  ```

- ```
  PROCEDURE p5 (l: tipolista);
  BEGIN
     IF l <> NIL THEN
     BEGIN
       writeln(l^.info);
       p5(l^.pros);
       writeln(l^.info)
     END
  END;
  ```

- ```
  PROCEDURE p6 (l: tipolista);
  BEGIN
     IF l <> NIL THEN l^.pros := l;
     WHILE l <> NIL DO
     BEGIN
        writeln(l^.info);
        l := l^.pros
     END
  END;
  ```

14. Ognuna delle seguenti procedure riceve come parametro il puntatore ad una lista di interi. Per ogni procedura, individuare il contenuto della lista dopo l'esecuzione, supponendo che la lista contenga inizialmente gli interi `4 10 2 5`. Descrivere poi brevemente la funzionalità svolta dalla procedura.

    - ```
      PROCEDURE p1 (l: tipolista);
      VAR s: integer;
      BEGIN
         s := 0;
         WHILE l <> NIL DO
         BEGIN
            s := s + l^.info;
            l^.info := s - l^.info;
            l := l^.pros
         END
      END;
      ```

    - ```
      PROCEDURE p2 (l: tipolista);

         PROCEDURE a (l: tipolista; s: integer);
         BEGIN
      ```
      ```

```
      IF l <> NIL THEN
      BEGIN
         a(l^.pros, s + l^.info);
         l^.info := s
      END
   END;

BEGIN
   a (l,0)
END;
```

- ```
  PROCEDURE p3 (VAR l: tipolista);
  VAR s: integer;
  BEGIN
     s := 0;
     WHILE l <> NIL DO
     BEGIN
        s := s + l^.info;
        l^.info := s - l^.info;
        l := l^.pros
     END
  END;
  ```
- ```
  PROCEDURE p4 (VAR l: tipolista);

     PROCEDURE a (VAR l: tipolista; s: integer);
     BEGIN
        IF l <> NIL THEN
        BEGIN
           a (l^.pros, s + l^.info);
           l^.info:= s
        END
     END;

  BEGIN
     a (l,0)
  END;
  ```
- ```
  PROCEDURE p5 (l: tipolista);

     PROCEDURE a (l: tipolista; s: integer);
     BEGIN
        IF l <> NIL THEN
        BEGIN
           l^.info := s;
           a(l^.pros, s + l^.info)
        END
     END;

  BEGIN
     a(l,0)
  END;
  ```

15. Per ognuno dei seguenti frammenti di codice individuare delle dichiarazioni di variabile ed
  ```

eventualmente di tipo in modo che le istruzioni che vi appaiono risultino corrette. Se ciò non fosse possibile spiegare il motivo.

- ```
  r := s;
  writeln(chr(s^));
  ```
- ```
  q^.val := q <> NIL;
  q^.pros := q;
  ```
- ```
  p[a]^ := a+1;
  ```

16. Per ognuno dei tre frammenti di codice seguenti individuare delle dichiarazioni di variabile ed eventualmente di tipo 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.

- ```
  p^.b := NIL;
  p^.c := x[p^.a];
  ```
- ```
  r := s;
  s^:= succ(r^);
  ```
- ```
  y[q^] := q;
  q^ := chr(ord(q = NIL));
  ```

17. Per ognuno dei seguenti frammenti di codice individuare delle dichiarazioni di variabile ed eventualmente di tipo in modo che le istruzioni che vi appaiono risultino corrette. Se ciò non è possibile spiegare il motivo.

- ```
  r.a := r.b;
  r.b := r.b + 1;
  r.c := chr(r.a);
  ```
- ```
  p^.val := x[p <> NIL] > y;
  ```
- ```
  q[p^.a[i]] := i;
  ```
  ```