Lezione 22
21–22 dicembre 1999

Argomenti trattati
- Sottoprogrammi iterativi e ricorsivi per il trattamento degli alberi binari.
- Record con varianti.

22.1 Inserimento iterativo in un albero di ricerca

Riscriviamo ora la procedura di inserimento di un nuovo valore in un albero di ricerca, senza utilizzare la ricorsione. L'intestazione della procedura è

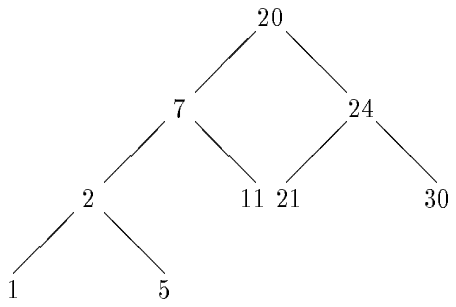
```
PROCEDURE inserisci (VAR a: tipoalbero; x: integer);
```

dove `tipoalbero` è definito come segue:

```
TYPE
   tipoalbero = ^nodoalbero;
   nodoalbero = RECORD
         info: integer;
         sx, dx: tipoalbero
      END;
```

La procedura ha una struttura simile a quella di inserimento in una lista ordinata: dopo avere identificato la posizione alla quale effettuare l'inserimento, si crea un nuovo nodo e lo si inserisce nella struttura.

Mentre nel caso delle liste, la ricerca della posizione procedeva linearmente, seguendo i puntatori lungo la lista, nel caso degli alberi di ricerca ci si muoverà di volta in volta seguendo il puntatore sinistro o destro in base al risultato del confronto tra la chiave da inserire e il valore contenuto nel nodo considerato, fino a raggiungere una posizione libera in cui effettuare l'inserimento. Ad esempio, si supponga di volere inserire il valore 8 nel seguente albero di ricerca:



La procedura inizia esaminando il valore contenuto nella radice. Poiché il valore da inserire è minore del valore della radice, ci si sposta sul figlio sinistro, cioè il nodo che contiene 7. Al successivo confronto, essendo 8 maggiore di 7, ci si sposta sul figlio destro, che contiene 11. Dunque, essendo 8 minore di 11, ci si sposta di nuovo a sinistra, dove, non essendovi alcun nodo, si effettua l'inserimento.

Il ciclo di ricerca può essere schematizzato come segue:

```
considera l'intero albero, partendo dalla sua radice
WHILE l'albero considerato non e' vuoto DO
   IF il valore da inserire e' minore del valore della radice THEN
      considera il sottoalbero sinistro
   ELSE
      considera il sottoalbero destro
```

Il ciclo può essere riscritto facendo uso dei puntatori come:

```
p := a;
WHILE p <> NIL DO
   IF x < p^.info THEN
      p := p^.sx
   ELSE
      p := p^.dx
```

Questo ciclo da solo non è di alcuna utilità. Infatti, dopo avere percorso il cammino sul quale effettuare l'inserimento, all'uscita dal ciclo `p` conterrà `NIL`. Per effettuare correttamente l'inserimento è necessario accedere al nodo *dopo* il quale effettuare l'inserimento e sapere in *quale direzione* (sinistra o destra) effettuare l'inserimento. Ad esempio, per inserire 8 nell'albero precedente, occorrerà un puntatore al nodo contenente 11 (di cui andrà modificato il campo `sx`) e sapere che l'inserimento va effettuato a sinistra.

A questo scopo, introduciamo un puntatore ausiliario `q`, inizializzato a `NIL`, che ad ogni iterazione del ciclo contiene il valore che aveva `p` nell'iterazione precedente. Inoltre, utilizziamo una variabile di nome `direzione`, che può assumere valori `sinistra` e `destra`, per memorizzare da che parte è stato l'ultimo spostamento. La fase di ricerca può essere dunque riscritta come:

```
p := a;
q := NIL;
WHILE p <> NIL DO
   BEGIN
      q := p;
      IF x < p^.info THEN
         BEGIN
            direzione := sinistra;
            p := p^.sx
         END {then}
      ELSE
         BEGIN
            direzione := destra;
            p := p^.dx
         END{else}
   END {while}
```

In uscita dal ciclo, `q` contiene un puntatore al nodo *dopo* il quale va effettuato l'inserimento, `direzione` indica da che parte effettuare l'inserimento, con l'unica eccezione del caso di albero inizialmente vuoto in cui `q` contiene `NIL` e l'inserimento va effettuato direttamente creando un nodo a cui far puntare `a`. Tenendo conto di queste condizioni la fase successiva sarà:

```
new(p);
p^.info := x;
p^.sx := NIL;
p^.dx := NIL;

IF a = NIL THEN {se l'albero e' vuoto}
   a := p
ELSE IF direzione = sinistra THEN
   q^.sx := p
ELSE
   q^.dx := p
```

Segue il listato completo della procedura:

```
PROCEDURE inserisci (VAR a: tipoalbero; x: integer);

{inserisce il valore di x nell'albero di ricerca puntato da a}
{versione iterativa}

   VAR
      p, q: tipoalbero;
      direzione: (sinistra, destra);

BEGIN {inserisci}
   {ricerca della posizione alla quale effettuare l'inserimento}
   {dopo la ricerca q conterra' il puntatore al nodo DOPO il quale inserire}
   {direzione indichera' da che parte effettuare l'inserimento}

   p := a;
   q := NIL;
   WHILE p <> NIL DO
      BEGIN
         q := p;
         IF x < p^.info THEN
            BEGIN
               direzione := sinistra;
               p := p^.sx
            END {then}
         ELSE
            BEGIN
               direzione := destra;
               p := p^.dx
            END{else}
      END; {while}

   {creazione del nuovo nodo}
   new(p);
   p^.info := x;
   p^.sx := NIL;
   p^.dx := NIL;

   {collegamento del nodo all'albero}
   IF a = NIL THEN {se l'albero e' vuoto}
```

```
      a := p
   ELSE IF direzione = sinistra THEN
      q^.sx := p
   ELSE
      q^.dx := p
END; {inserisci}
```

22.2 Record con varianti

Supponiamo di volere rappresentare un'espressione aritmetica mediante un albero binario in cui i nodi interni rappresentano una delle quattro operazioni e le foglie dei numeri interi. Supponiamo di avere definito un tipo `operatori`, corrispondente alle quattro operazioni disponibili:

```
TYPE
   operatori = (piu, meno, per, diviso)
```

In un nodo dell'albero, oltre a rappresentare i puntatori al sottoalbero sinistro e al sottoalbero destro, dobbiamo memorizzare in alternativa:

- un valore di tipo `operatori`, oppure
- un valore di tipo `integer`.

La definizione di `RECORD` in Pascal permette di specificare, oltre a una parte fissa, anche una *parte variante*, che preveda differenti alternative. In questo specifico caso possiamo introdurre le seguenti definizioni:

```
TYPE
   operatori = (piu, meno, per, diviso);
   generenodi = (oper, val);
   tipoexpr = ^nodoexpr;
   nodoexpr = RECORD
         sx, dx: tipoexpr;
         CASE nodo : generenodi OF
            oper: (
                  segno: operatori
            );
            val: (
                  valore: integer
            )
      END; {record}
```

Una variabile di tipo `nodoexpr` è un record che può avere una delle seguenti due strutture:

1. campi `sx`, `dx`, `nodo`, `segno`, quando il campo `nodo` ha valore `oper`;
2. campi `sx`, `dx`, `nodo`, `valore`, quando il campo `nodo` ha valore `val`.

I campi `sx`, `dx` e `nodo`, comuni ad entrambe le alternative, formano la *parte fissa*; gli altri campi formano le *varianti* del record. Il campo `nodo` viene anche detto *etichetta*, perché in base al suo valore si riconosce quale variante del record è utilizzata. Si noti che un unico `END` chiude sia la definizione di `RECORD` che la parte variante.

Se `x` è una variabile di tipo `nodoexpr`, possiamo scrivere gli assegnamenti

```
x.sx := NIL;
x.nodo := val;
x.valore := 10
```

Se `x.nodo` contiene `val`, il record è organizzato in base alla seconda struttura che abbiamo indicato. Dunque il campo `segno` non esiste ed è scorretto utilizzare `x.segno` (non è detto che, in questo caso, l'esecuzione del programma venga interrotta, ma i risultati sono imprevedibili).

Per modificare la variante utilizzata, è sufficiente cambiare valore al campo `nodo`:

```
x.nodo := oper
```

A questo punto il record è strutturato nel primo modo; è dunque possibile utilizzare il campo `segno` (che inizialmente avrà, come sempre, valore indefinito).

Altri esempi di assegnamenti, sono riportati qui di seguito:

```
   VAR
      n, t: tipoexpr;

BEGIN
   new(n);
   n^.sx := NIL;
   n^.dx := NIL;
   n^.nodo := oper;
   n^.segno := piu;

   new(t);
   t^.sx := NIL;
   t^.dx := NIL;
   t^.nodo := val;
   t^.valore := 10;
   t^.valore := t^.valore + ord(n^.segno)

   ...
END.
```

Un record con varianti occupa in memoria il massimo tra gli spazi richiesti dalle proprie varianti. Ad esempio, nell'ipotesi che puntatori e `integer` vengano memorizzati su 2 byte, e valori dei tipi `operatori` e `generenodi` su un solo byte, il record richiederà 6 byte se l'etichetta contiene `oper`, e 7 byte se l'etichetta contiene `val`. Pertanto, lo spazio utilizzato per memorizzare variabili di tipo `nodoexpr`, è sempre di 7 byte.

In un record con varianti è possibile omettere il campo etichetta e specificare solo un nome di un tipo per definire le possibili alternative. Ad esempio:

```
TYPE
   r = RECORD
         a: char;
         CASE boolean OF
            true: (
                  b: char;
                  c: char;
                  d: char;
                  e: char;
            );
            false: (
                  g: integer
            )
      END;
```

In questo caso il record può essere strutturato in uno dei seguenti modi:

1. cinque campi di tipo `char`, di nomi `a`, `b`, `c`, `d` ed `e`;
2. un campo di tipo `char`, di nome `a`, e un campo di tipo `integer` di nome `g`.

Non esiste alcun campo che tenga traccia di quale struttura sia attualmente in uso. Poiché le varianti di un record vengono implementate sulla stessa area di memoria, in questo modo è possibile interpretare il contenuto della stessa area secondo tipi diversi. Si consideri ad esempio il seguente programma:

```
PROGRAM p (input, output);
   TYPE
      r = RECORD
            a: char;
            CASE boolean OF
               true: (
                     b: char;
                     c: char;
                     d: char;
                     e: char;
               );
               false: (
                     g: integer
               )
         END;

   VAR
      x: r;

BEGIN
   readln(x.a, x.b, x.c, x.d, x.e);
   writeln(x.a, x.b, x.c, x.d, x.e, x.g);
   x.g := x.g + 1;
   writeln(x.a, x.b, x.c, x.d, x.e, x.g);
END.
```

Nella prima riga, vengono letti cinque valori di tipo `char`, assegnati rispettivamente ai campi `a`, `b`, `c`, `d` ed `e`. Nella seconda riga viene riscritto in `output` l'input letto, insieme al contenuto del campo `g`. Quest'ultimo campo, in realtà, utilizza la stessa area di memoria dei campi `b`, `c`, `d` ed `e`. L'istruzione successiva incrementa di 1 il campo `g`. In questo modo, eseguendo l'istruzione di scrittura alla fine dell'esecuzione, risultano modificati anche alcuni dei campi della prima variante.

Ad esempio, eseguendo il programma compilato con Think Pascal, e con la stringa `gatto` in `input`, si è ottenuto il seguente output:

```
gatto      97
gbtto      98
```

Lo stesso programma, compilato con HP-Pascal, ha fornito su input `gatto`, l'output:

```
gatto  1635021935
gattp  1635021936
```

Si noti che in entrambi casi, l'incremento della componente `g` ha provocato un "incremento" di una delle componenti dell'altra variante. Il comportamento del programma è strettamente legato alla rappresentazione utilizzata per i vari tipi. Chiaramente, l'uso dei record con varianti in questa maniera deve essere evitato.

Esercizi

1. Costruire (in versione ricorsiva e iterativa) una procedura che ricevendo come primo parametro il puntatore a una lista di interi, restituisca nel secondo parametro il puntatore a una nuova lista contenente gli stessi elementi della prima, ma in ordine inverso.

2. Costruire (in versione ricorsiva e iterativa) una procedura che ricevendo come primo parametro il puntatore a una lista di interi, restituisca nel secondo parametro il puntatore a una nuova lista contenente gli stessi elementi della prima, ordinati in modo non decrescente.

3. Costruire una procedura che ricevendo come unico parametro il puntatore a una lista di interi, riordini gli elementi della lista in modo non decrescente. Nota: mentre nell'esercizio precedente si chiede la creazione di una copia, in questo esercizio non si devono allocare nuove variabili dinamiche, ma si deve invece riorganizzare la lista data.

4. Costruire (in versione ricorsiva e iterativa) una procedura che ricevendo come unico parametro il puntatore a una lista di interi, elimini da tale lista tutti gli elementi che si trovano in posizione pari.

5. Costruire (in versione ricorsiva e iterativa) una procedura che ricevendo come unico parametro il puntatore a una lista di interi, elimini da tale lista tutti gli elementi che si trovano in posizione dispari.

6. (Dal tema d'esame del 21 febbraio 1997.) Si vuole costruire un ambiente per lo studio e la prova di sottoprogrammi che manipolano sequenze di interi *ordinate* in modo non decrescente. In base ad un menù, l'utente può scegliere di volta in volta quali operazioni effettuare manipolando una sequenza, inizialmente vuota. Le operazioni previste sono:

   (a) Inserimento di un intero all'interno della sequenza. Ad esempio, se la sequenza contiene `2 4 4 37 98 462` e l'utente vuole inserire `45`, la nuova sequenza sarà `2 4 4 37 45 98 462`. Si noti che uno stesso elemento può apparire nella sequenza più di una volta.

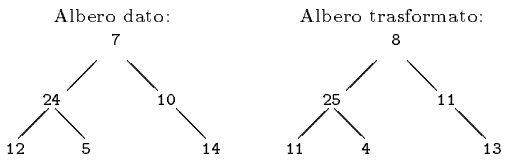
   (b) Stampa del contenuto della sequenza.

   (c) Ricerca di un elemento all'interno della sequenza: dato in ingresso un intero, il programma indica le posizioni in cui appare, se presente. Ad esempio, se la sequenza contiene `2 4 4 37 98 462` e l'utente inserisce per la ricerca `4`, il programma dovrà rispondere che l'elemento si trova in seconda e in terza posizione.

   (d) Rimozione delle ripetizioni. Ad esempio, se la sequenza contiene `2 2 4 4 37 98 98 98 462 462`, la nuova sequenza sarà `2 4 37 98 462`.

   Il programma *deve essere privo* di vincoli sulla lunghezza massima della sequenza trattata e deve risultare facilmente espandibile nel caso si vogliano introdurre nuove funzionalità.

7. Scrivere la dichiarazione del tipo albero binario di interi. Costruire poi una procedura che ricevendo come parametro il puntatore alla radice di un albero di interi, incrementi di `1` il valore contenuto in ogni nodo interno e decrementi di `1` il valore contenuto in ogni foglia. Esempio:



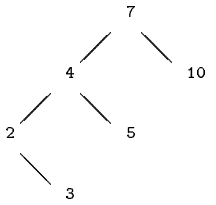
8. Scrivere la dichiarazione del tipo lista di caratteri. Costruire poi una procedura che ricevendo come parametro il puntatore al primo elemento di una lista di caratteri, la rovesci. Se ad esempio la lista contiene inizialmente la sequenza `r o m a`, dopo l'esecuzione della procedura dovrà contenere `a m o r`.

9. (Dal tema d'esame del 13 giugno 1997.) Si vuole costruire un ambiente per lo studio e la prova di sottoprogrammi che manipolano alberi binari di ricerca contenenti numeri interi. In base ad un menù, l'utente può scegliere di volta in volta quali operazioni effettuare manipolando un albero, inizialmente vuoto. Le operazioni previste sono:

   (a) Ricerca con inserimento: dato in ingresso un intero, il programma indica se sia già presente e, in caso negativo, lo inserisce.

   (b) Stampa del contenuto dell'albero: vengono stampati in ordine crescente i valori contenuti nei nodi; accanto ad ogni valore viene stampato il *livello* del nodo. Si ricordi che la radice dell'albero si trova a livello 1, e che i figli di un nodo a livello i si trovano a livello $i+1$. Esempio:

se l'albero è:



si dovrà stampare:

valore	livello
2	3
3	4
4	2
5	3
7	1
10	2

   (c) Calcolo della lunghezza media dei cammini, cioè della media dei livelli dei nodi (nell'esempio è `2.5`).

   (d) Rimozione dell'elemento minimo contenuto nell'albero.

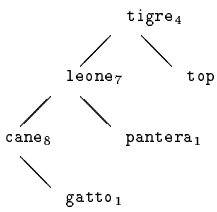
   Il programma deve risultare facilmente espandibile nel caso si vogliano introdurre nuove funzionalità.
   Nota. Mentre l'operazione di rimozione di un nodo da un albero è in generale piuttosto complicata, perché il nodo rimosso può avere due figli da ricollegare al resto dell'albero, nel caso del nodo contenente il valore minimo il sottoalbero sinistro è sempre vuoto...

10. (Dal tema d'esame dell'11 luglio 1997.) Si vuole costruire un ambiente per lo studio e la prova di sottoprogrammi che manipolano alberi binari di ricerca contenenti nomi. Ad ogni nome è associato un contatore che indica quante volte il nome sia stato oggetto di un'operazione d'inserimento. In base ad un menù, l'utente può scegliere di volta in volta quali operazioni effettuare manipolando un albero, inizialmente vuoto. Le operazioni previste sono:

   (a) Inserimento: dato in ingresso un nome, il programma lo inserisce nell'albero se non è già presente, altrimenti ne incrementa il contatore.

   (b) Stampa del contenuto dell'albero in ordine alfabetico. Oltre ai nomi contenuti nei nodi, vengono stampati anche i contatori. Esempio:

se l'albero è:



si dovrà stampare:

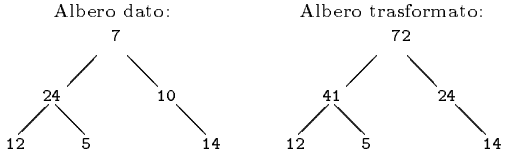
nome	cont.
cane	8
gatto	1
leone	7
pantera	1
tigre	4
topo	3

   (c) Calcolo della frequenza media dei nomi, cioè della media dei contatori associati ai nomi (nell'esempio è `4`).

(d) Rimozione dell'elemento massimo (in ordine alfabetico) contenuto nell'albero.

Il programma deve risultare facilmente espandibile nel caso si vogliano introdurre nuove funzionalità. È possibile fissare un limite superiore alla lunghezza di nomi trattati (ad esempio 10 caratteri).

11. Scrivere la dichiarazione del tipo albero binario di interi. Costruire poi una procedura che ricevendo come parametro il puntatore alla radice di un albero di interi, sostituisca il valore di ciascun nodo con la somma dei valori contenuti nel relativo sottoalbero. Esempio:



12. 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 dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

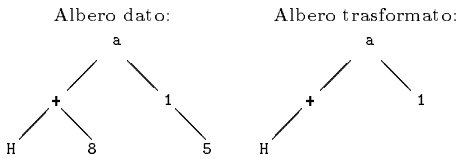
    - ```
      r^:= a[r^] > ord(r^);
      ```
    - ```
      x:= chr(y^);
      v[x,y^]:= ord(succ(x));
      ```
    - ```
      new(q.a);
      q.a^:= q.b[q.c];
      q.c:= q.c + 1;
      ```

13. Scrivere la dichiarazione del tipo lista di interi. Costruire poi una procedura che ricevendo come parametro il puntatore a una lista di interi, modifichi tale lista duplicandone ogni elemento. Ad esempio, se prima della chiamata la lista contiene i valori `5 15 6 76`, dopo l'esecuzione della procedura dovrà contenere `5 5 15 15 6 6 76 76`.

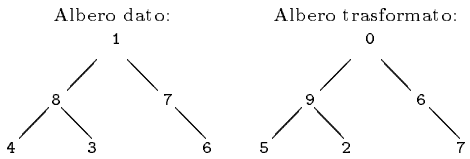
14. Dopo avere dichiarato il tipo *lista di interi*, scrivere in Pascal una `PROCEDURE` che riceva come parametri il puntatore a una lista di interi e un intero x ed elimini dalla lista il primo elemento che sia multiplo di x. Ad esempio, se la lista contiene inizialmente i valori `10 7 15 12` e x vale `3`, dopo l'esecuzione della `PROCEDURE` la lista dovrà contenere `10 7 12`.

15. Dopo avere dichiarato il tipo *lista di interi*, scrivere in Pascal una `PROCEDURE` che riceva come parametro il puntatore a una lista di interi e modifichi tale lista scambiando il minimo valore dispari presente con il massimo valore dispari presente. Ad esempio, se la lista contiene inizialmente i valori `8 1 18 14 13 2 7`, dopo l'esecuzione della `PROCEDURE` la lista dovrà contenere `8 13 18 14 1 2 7`.

16. Dopo avere dichiarato il tipo *albero binario di caratteri*, scrivere in Pascal una `PROCEDURE` che riceva come parametro il puntatore a un albero binario di caratteri e modifichi tale albero eliminando tutte le foglie che contengono cifre.
    Esempio:


      ```

17. Dopo avere dichiarato il tipo *albero binario di interi*, scrivere in Pascal una `PROCEDURE` che riceva come parametro il puntatore a un albero binario di interi e modifichi tale albero incrementando di 1 tutti i valori pari contenuti nell'albero e decrementando di 1 tutti gli altri valori.
Esempio:



18. Dopo avere dichiarato il tipo *lista di interi*, scrivere in Pascal una `PROCEDURE` che riceva come parametri il puntatore a una lista di interi e un intero x ed inserisca, all'inizio della lista, un nuovo elemento contenente il numero di valori maggiori di x presenti nella lista data. Ad esempio, se la lista contiene inizialmente i valori `8 17 4 2` e x vale `5`, dopo l'esecuzione della `PROCEDURE` la lista dovrà contenere `2 8 17 4 2`.

19. Dopo avere dichiarato il tipo *lista di interi*, scrivere in Pascal una `PROCEDURE` che riceva come parametro il puntatore a una lista di interi e modifichi la lista, inserendo, dopo ogni nodo, un nuovo nodo contenente lo stesso valore. Ad esempio, se la lista contiene inizialmente i valori `8 17 4 2` dopo l'esecuzione della `PROCEDURE` la lista dovrà contenere `8 8 17 17 4 4 2 2`.

20. Prova scritta del 16 aprile 1998 (Parte A)

    (a) Scrivere la dichiarazione del tipo lista di interi. Costruire poi una procedura che ricevendo come parametro il puntatore a una lista di interi, modifichi tale lista eliminando tutti gli elementi minimi e massimi. Ad esempio, se prima della chiamata la lista contiene i valori `5 15 6 5 76 18`, dopo l'esecuzione della procedura dovrà contenere `15 6 18`.

    (b) 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 dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

    - ```
      new(p);
      p:= p + 3;
      ```
    - ```
      new(p);
      p^:= p^ + 3;
      ```
    - ```
      new(p);
      p^.info:= p^.info + 3;
      p^.pros:= p;
      ```
    - ```
      new(p);
      p^.pros:= p^.pros + 3;
      p^.info:= p;
      ```

    (c) Disegnare l'albero di ricerca ottenuto inserendo uno dopo l'altro in un albero inizialmente vuoto i numeri `15 2 7 16 13 1 4 20 17`.
    Scrivere gli output prodotti visitando tale albero in ordine simmetrico.
    Si supponga ora di disporre di una procedura ricorsiva che, quando l'albero non è vuoto, visiti ricorsivamente prima il sottoalbero destro, poi visualizzi la radice e quindi visiti ricorsivamente di nuovo il sottoalbero destro. Scrivere l'output prodotto da tale procedura sull'albero considerato sopra.

21. Prova scritta del 12 giugno 1998 (Parte A)

    (a) Scrivere la dichiarazione del tipo albero binario di caratteri. Costruire poi una procedura ricorsiva che ricevendo come parametro il puntatore alla radice di un albero di caratteri, elimini dall'albero tutte le foglie (si ricordi che una foglia è un nodo privo di figli).

(b) 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 dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

- ```
  new(p);
  p^.alfa:= p;
  p^.beta:= p^.alfa;
  p^.gamma:= NIL;
  p^.delta:= p = NIL;
  ```
- ```
  i:= i + 1;
  new(p[i]);
  p[i]^:= i;
  ```
- ```
  p:= 'a';
  a:= ord(p) = ord('p');
  v[p]:= x;
  x[a]:= 1;
  ```

(c) Per ognuno dei seguenti programmi, disegnare le variabili presenti nello stack e nello heap durante l'esecuzione, con i rispettivi contenuti. Indicare poi l'output prodotto.

- ```
  PROGRAM p1 (input,output);
  VAR r, s: ^integer;
  BEGIN
     new (r);
     new (s);
     s^:= 7;
     r^:= 15;
     r^:= r^ + s^;
     s^:= r^ - s^;
     r^:= r^ - s^;
     writeln (r^, s^)
  END.
  ```
- ```
  PROGRAM p2 (input,output);
  VAR r, s: ^integer;
  BEGIN
     new (r);
     new (s);
     s^:= 7;
     r^:= 15;
     r:= s;
     r^:= r^ + s^;
     s^:= r^ - s^;
     r^:= r^ - s^;
     writeln (r^, s^)
  END.
  ```
- ```
  PROGRAM p3 (input,output);
  VAR r, s: ^integer;
  BEGIN
     new (r);
     new (s);
     s^:= 7;
     r^:= 15;
     r^:= s^;
     r^:= r^ + s^;
  ```

```
    s^:= r^ - s^;
    r^:= r^ - s^;
    writeln (r^, s^)
END.
```

22. Prova scritta del 14 luglio 1998 (Parte A)

    (a) Scrivere la dichiarazione del tipo lista di interi. Costruire poi una procedura che ricevendo come parametro il puntatore a una lista di interi, modifichi tale lista eliminando tutti gli elementi superiori alla media degli elementi contenuti inizialmente nella lista. Ad esempio, se prima della chiamata la lista contiene i valori `5 15 6 5 40 18`, dopo l'esecuzione della procedura dovrà contenere `5 6 5`, essendo la media uguale a `14.83...`.

    (b) 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 dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

    - ```
      new(p);
      FOR i:= 'a' TO 'z' DO
         p^[i]:= ord(i);
      ```
    - ```
      FOR i:= 'a' TO 'z' DO
      BEGIN
         new(p[i]);
         p[i]^:= ord(i)
      END;
      ```
    - ```
      c:= a DIV b + a MOD b;
      d:= a / b + a MOD b;
      e:= c + d;
      ```
    - ```
      y:= succ(100);
      x:= chr(y);
      z[x,y]:= x;
      ```

    (c) Disegnare l'albero di ricerca ottenuto inserendo uno dopo l'altro in un albero inizialmente vuoto i numeri `8 2 7 16 1 4 33 15 12`.
    Scrivere gli output prodotti visitando tale albero nei tre ordini anticipato, simmetrico e posticipato.

23. Prova scritta del 25 settembre 1998 (Parte A)

    (a) Scrivere la dichiarazione del tipo lista di interi. Costruire poi una procedura che ricevendo come parametro il puntatore a una lista di interi, modifichi tale lista eliminando tutti gli elementi che si trovano in posizione pari. Ad esempio, se prima della chiamata la lista contiene i valori `5 15 6 5 40 18 27`, dopo l'esecuzione della procedura dovrà contenere `5 6 40 27`.

    (b) 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 dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

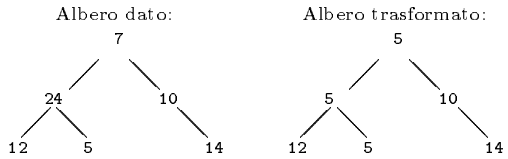
    - ```
      a:= 10;
      z:= 100;
      FOR i:= 'a' TO 'z' DO
         writeln(succ(i));
      ```
    - ```
      a:= 10;
      z:= 100;
      FOR i:= a TO z DO
         writeln(succ(i));
      ```

- ```
  a:= 10;
  z:= 100;
  FOR i:= a TO 'z' DO
     writeln(succ(i));
  ```
- ```
  new(p);
  p^.info:= p;
  p^.pros:= p;
  p^.dato:= p = NIL;
  ```
- ```
  x:= y;
  x[y[j]]:= y[x[i]] + 1;
  ```

(c) Disegnare l'albero di ricerca ottenuto inserendo uno dopo l'altro in un albero inizialmente vuoto i numeri `11 14 18 7 3 8 10 4 15`.
Scrivere gli output prodotti visitando tale albero nei tre ordini anticipato, simmetrico e posticipato.

24. Prova scritta del 30 ottobre 1998 (Parte A)

(a) Scrivere la dichiarazione del tipo albero binario di interi. Costruire poi una procedura che ricevendo come parametro il puntatore alla radice di un albero di interi, sostituisca il valore di ciascun nodo con il minimo dei valori contenuti nel relativo sottoalbero. Esempio:



(b) Per ognuno dei seguenti tre casi, 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:

- il programma principale possa richiamare la procedura `a` e la procedura `b`; la procedura `a` possa richiamare la procedura `c`, la procedura `b` possa richiamare sia la procedura `a` che la procedura `c`.
- il programma principale possa richiamare la procedura `a` e la procedura `b`, ma *non* la procedura `c`; la procedura `a` possa richiamare la procedura `c`; la procedura `b` possa richiamare la procedura `a`;
- il programma principale possa richiamare la procedura `b` e la procedura `c`; la procedura `a` possa richiamare la procedura `b`; la procedura `b` possa richiamare sia la procedura `a` che la procedura `c`.

(c) Dopo avere scritto le dichiarazioni di tipo e variabile opportune, riscrivere i seguenti frammenti di codice sostituendo i cicli `WHILE` e `FOR` con cicli `REPEAT`, ed eventualmente istruzioni di selezione.

- ```
  a := pred(x) = succ(y);
  WHILE a DO
     writeln('ciclo while in corso');
  writeln('ciclo while finito');
  ```
- ```
  read(a);
  FOR a := succ(a) DOWNTO pred('a') DO
     writeln(a);
  ```
- ```
  FOR x := 1 TO p['a']['a'] DO
     r[x] := p[chr(x)];
  ```