15.8 Trasformazione di alberi

Da un albero ordinato A di *n* nodi è possibile ricavare un equivalente albero binario B di *n* nodi con la seguente regola:

- la radice di A coincide con la radice di B;
- ogni nodo *b* di B ha come radice del sottoalbero sinistro il primo figlio di *b* in A e come sottoalbero destro il fratello successivo di *b* in A.

In Figura 15.5 vediamo la trasformazione dell'albero di Figura 15.3 in albero binario.

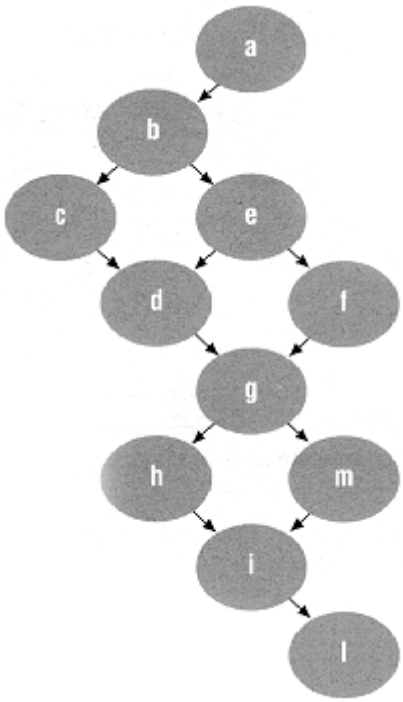

Figura 15.5 Albero binario corrispondente all'albero di Figura 15.3

Dalla regola di trasformazione si deduce che in un albero binario, ricavato da un albero ordinato, la radice può avere solamente il figlio sinistro.

Il programma del Listato 15.6 accetta in ingresso la rappresentazione parentetica di un albero ordinato e memorizza il corrispondente albero binario nella forma che abbiamo visto nei primi paragrafi di questo capitolo. Le funzioni utilizzate sono di tipo non ricorsivo e pertanto esiste il problema di memorizzare i puntatori degli elementi di livello superiore. A questo scopo si utilizza una pila di puntatori, dove si va a inserire il puntatore di un certo livello prima di scendere al livello sottostante e da dove si preleva il puntatore una volta che si debba risalire di livello.
La pila è implementata mediante una lista. Rimangono dunque valide le funzioni di `inserimento`, `eliminazione` e `pila_vuota` già trattate. L'unica differenza sta nella struttura di ogni elemento: la parte informazione è di tipo puntatore a un nodo dell'albero:

```
struct elemento {              /* Elemento della lista lineare  */
  struct nodo     *inf;        /* con cui è implementata la pila */
  struct elemento *pun;
};
```

Le visite le abbiamo realizzate come al solito in forma ricorsiva. Si può verificare che la visita dell'albero di partenza in ordine anticipato è uguale alla visita in ordine anticipato del corrispondente albero binario. La visita in ordine differito corrisponde invece alla visita in ordine simmetrico dell'albero binario.

```
/* Dalla rappresentazione parentetica di un albero ricava il
   corrispondente albero binario, che visita in ordine simmetrico,
   anticipato e differito.
   Per la creazione usa una funzione iterativa (non ricorsiva)
   con l'ausilio di una pila gestita mediante una lista lineare
   il cui campo inf è un puntatore ai nodi dell'albero binario.
   Per le  visite in ordine simmetrico, anticipato e differito
   rimangono valide le funzioni ricorsive già esaminate         */

#include <stdio.h>
#include <stddef.h>
```

```
struct nodo {                   /* Nodo dell'albero binario */
  char inf;
  struct nodo *alb_sin;
  struct nodo *alb_des;
};

struct nodo *alb_bin_par();    /* Funzione per la creazione */

void anticipato(struct nodo *);         /* Visite */
void simmetrico(struct nodo *);
void differito(struct nodo *);


struct elemento {             /* Elemento della lista lineare  */
  struct nodo     *inf;       /* con cui è implementata la pila */
  struct elemento *pun;
};

/* Funzioni per la gestione della pila */
struct elemento *inserimento(struct elemento *, struct nodo *);
struct elemento *eliminazione( struct elemento *, struct nodo **);
int pila_vuota(struct elemento *);

main()
{
struct nodo *radice;

radice = alb_bin_par();

printf("\nVISITA IN ORDINE SIMMETRICO\n");
simmetrico( radice );
printf("\nVISITA IN ORDINE ANTICIPATO\n");
anticipato( radice );
printf("\nVISITA IN ORDINE DIFFERITO\n");
differito( radice );
}


/* Dalla rappresentazione parentetica di un albero crea
   il corrispondente albero binario                      */
struct nodo *alb_bin_par()
{
struct nodo *p;
struct nodo *paus, *pp;
char car;
int logica = 1;

struct elemento *punt_testa = NULL;    /* Inizializzazione pila */

printf("\nInserisci la rappresentazione dell'albero: ");
scanf("%c", &car);

/* creazione della radice */
p = (struct nodo *) malloc(sizeof(struct nodo));

scanf("%c", &p->inf);

p->alb_sin = NULL;     /* Inizializzazione dei puntatori */
p->alb_des = NULL;     /* ai sottoalberi */
```

```
paus = p;  logica = 1;

do {
  scanf("%c", &car);
  if(car=='(') {
    pp = (struct nodo *) malloc(sizeof(struct nodo));
    scanf("%c", &pp->inf);
    pp->alb_sin = NULL;
    pp->alb_des = NULL;
    if(logica) {
      paus->alb_sin = pp;
      /* Inserimento in pila */
      punt_testa = inserimento(punt_testa, paus);
    }
    else {
      paus->alb_des = pp;
      logica = 1;
    }
    paus = pp;
   }
   else
     if(logica)
       logica = 0;
     else
       /* Eliminazione dalla pila */
       punt_testa = eliminazione(punt_testa, &paus);
}
while(!pila_vuota(punt_testa));

return(p);
}


/* Funzioni per la gestione della pila */

struct elemento *inserimento(struct elemento *p, struct nodo *ele)
{
struct elemento *paus;

paus = (struct elemento *)malloc(sizeof(struct elemento));
if(paus==NULL) return(NULL);

paus->pun = p;
p = paus;
p->inf = ele;

return(p);
}


struct elemento *eliminazione(struct elemento *p, struct nodo **ele)
{
struct elemento *paus;

*ele = p->inf;
paus = p;
p = p->pun;
free(paus);
```

```
return(p);
}


int pila_vuota(struct elemento *p)
{
if(p ==NULL)
  return(1);
else
  return(0);
}


/* Devono essere incluse le funzioni ricorsive di visita dell'albero binario già esaminate, dove il valore visualizzato è di tipo char /*
. . .
```

Listato 15.6 Creazione di un albero binario corrispondente all'albero ordinato (non binario) immesso in forma parentetica dall'utente