14.9 Gestione di una pila mediante lista lineare

Il problema di questo paragrafo è: gestire una struttura astratta di tipo pila per mezzo delle operazioni di inserimento ed eliminazione degli elementi e visualizzazione della pila. L'informazione presente in un elemento della pila è di tipo intero. Bisogna implementare la pila per mezzo di una struttura a lista lineare.

L'analisi del problema, la scomposizione in sottoproblemi e la struttura del programma sono analoghe a quelle esaminate nel paragrafo precedente. Perciò la trattazione che segue si limita a individuare le differenze rispetto a quelle, soffermandosi sull'implementazione con la lista lineare. In questo caso non si ha bisogno di dimensionare la struttura, si utilizzerà tanto spazio in memoria quanto necessario.

La verifica di pila piena deve essere modificata perché l'allocazione dinamica della memoria non richiede una predefinizione di lunghezza massima; ciò nonostante la memoria disponibile potrebbe a un certo punto finire, caso che deve essere gestito dal programmatore. A questo proposito si veda la funzione `inserimento`. La verifica di pila vuota è banale: si tratta di verificare se il puntatore alla lista è uguale a `NULL`. La visualizzazione della pila corrisponde a una scansione della lista che abbiamo già più volte incontrato. Nel Listato 14.5 riportiamo il programma completo.

La funzione `gestione_pila` passa alla funzione `inserimento` il puntatore alla testa della lista e l'informazione da inserire:

```
punt_testa = inserimento(punt_testa, ele);
```

I parametri formali corrispondenti si chiamano `p` ed `ele`:

```
struct elemento *inserimento(struct elemento *p, int ele)
```

Deve essere creato un nuovo elemento e questo deve venire inserito in testa alla lista lineare. Per far ciò si ha bisogno di un puntatore ausiliario `paus`, per mezzo del quale viene creato il nuovo elemento (Figura 14.10a).

```
paus = (struct elemento *)malloc(sizeof(struct elemento));

     if(paus==NULL) return(NULL);
```

Se la memoria dinamica a disposizione è terminata il sistema ritorna `NULL`, la funzione si chiude e restituisce questo valore al programma chiamante, che comunica all'utente l'impossibilità di effettuare l'inserimento. Altrimenti si deve connettere l'elemento al puntatore alla testa della pila (Figura 14.10b):

```
paus->pun = p;
```

Il puntatore alla testa della pila deve riferirsi al nuovo elemento creato (Figura 14.10c):

```
p = paus;
```

Finalmente inseriamo nel nuovo elemento il valore dell'informazione passata dall'utente (Figura 14.10d):

```
p->inf = ele;
```

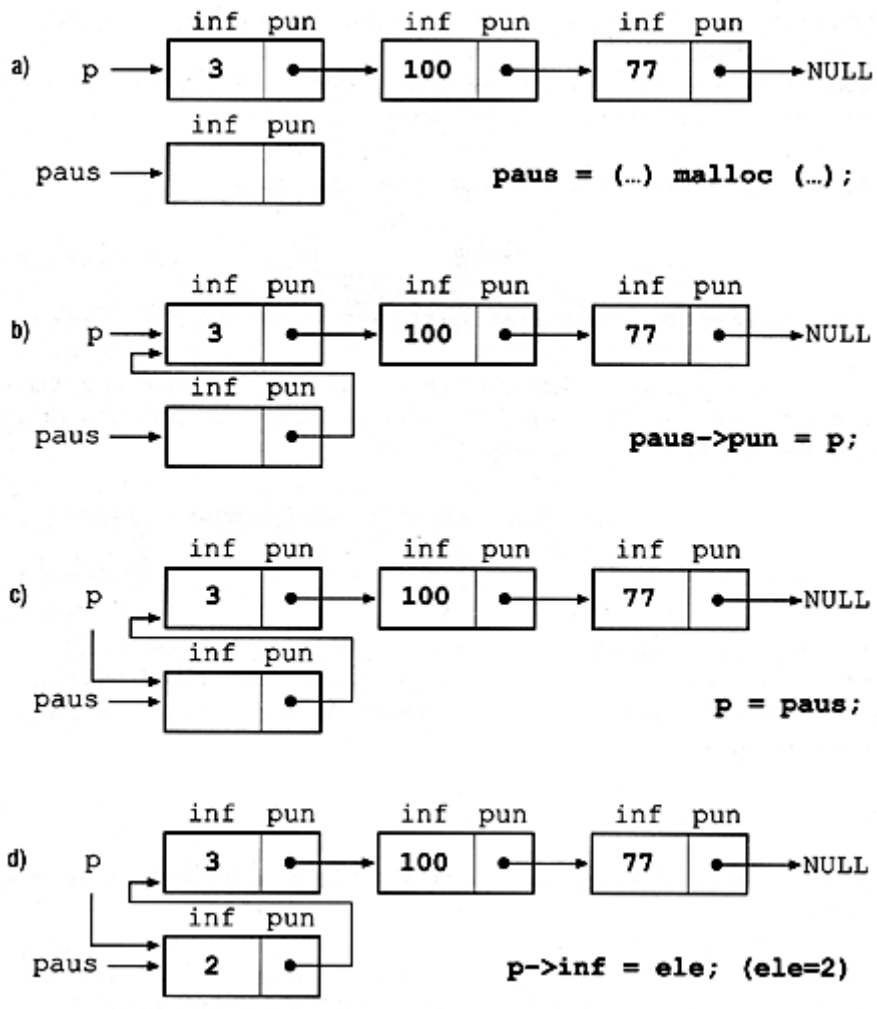

Figura 14.10 Le fasi a, b, c e d corrispondono all'inserimento di un nuovo elemento in testa alla pila

I parametri passati da `gestione_pila` a `eliminazione` sono gli stessi di `inserimento`, anche se in questo caso `ele` viene passata per indirizzo, per far in modo che la funzione restituisca l'informazione eliminata. L'eliminazione avviene semplicemente assegnando al puntatore alla lista il valore del puntatore del primo elemento (Figura 14.11a):

```
/* GESTIONE DI UNA PILA
   Operazioni di inserimento, eliminazione e
   visualizzazione. Utilizza una lista lineare
   per implementare la pila                    */

#include <stdio.h>
#include <malloc.h>

struct elemento {
  int inf;
  struct elemento *pun;
};

void gestione_pila(void);
struct elemento *inserimento(struct elemento *, int ele);
struct elemento *eliminazione(struct elemento *, int *);
int pila_vuota(struct elemento *);
void visualizzazione_pila(struct elemento *);

main()
{
  gestione_pila();
}


void gestione_pila(void)
{
struct elemento *punt_testa = NULL;
int scelta = -1, ele;
char pausa;

while(scelta!=0) {
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\tESEMPIO UTILIZZO STRUTTURA ASTRATTA: PILA");
  printf("\n\n\n\t\t\t 1. Per inserire un elemento");
  printf("\n\n\t\t\t 2. Per eliminare un elemento");
  printf("\n\n\t\t\t 3. Per visualizzare la pila");
  printf("\n\n\t\t\t 0. Per finire");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

  switch(scelta) {
    case 1:
       printf("Inserire un elemento: ");
       scanf("%d", &ele);
       punt_testa = inserimento(punt_testa, ele);
       if(punt_testa==NULL) {
         printf("Inserimento impossibile: ");
         printf("memoria disponibile terminata");
         printf("\n\n Qualsiasi tasto per continuare...");
         scanf("%c%c", &pausa, &pausa);
       }
       break;
```

```
    case 2:
       if(pila_vuota(punt_testa)) {
         printf("Eliminazione impossibile: pila vuota");
         printf("\n\n Qualsiasi tasto per continuare...");
         scanf("%c%c", &pausa, &pausa);
       }
       else {
         punt_testa = eliminazione(punt_testa, &ele);
         printf("Eliminato: %d", ele );
         printf("\n\n Qualsiasi tasto per continuare...");
         scanf("%c%c", &pausa, &pausa);
       }
       break;
    case 3:
       visualizzazione_pila(punt_testa);
       printf("\n\n Qualsiasi tasto per continuare...");
       scanf("%c%c", &pausa, &pausa);
       break;
  }
}
}


void visualizzazione_pila(struct elemento *p)
{
struct elemento *paus = p;

printf("\n<------ Testa della pila ");
while(paus!=NULL) {
  printf("\n%d", paus->inf);
  paus = paus->pun;
}
}


struct elemento *inserimento(struct elemento *p, int 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, int *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);
}
```

Listato 14.5 Gestione di una pila implementata mediante una lista