14.3 Gestione di una lista

Consideriamo il problema di memorizzare e successivamente visualizzare una sequenza di *n* interi. Il valore di *n* non è conosciuto a priori, ma è determinato in fase di esecuzione. Se si decide di utilizzare la struttura informativa array si deve prefissare il numero massimo di valori della sequenza. Si propende quindi per una soluzione che utilizzi la memoria in modo dinamico. Il tipo `elemento` è una struttura composta da due campi, un campo `inf` di tipo `int` e un campo `pun` puntatore alla struttura stessa:

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

Il problema presentato è divisibile nei due sottoproblemi:

- memorizzare la sequenza;
- visualizzare la sequenza.

Si affida quindi la soluzione dei due sottoproblemi a due funzioni, la cui dichiarazione prototype è:

```
struct elemento *crea_lista();
     void visualizza_lista(struct elemento *);
```

Nel `main` viene definito il puntatore che conterrà il riferimento al primo elemento della lista:

```
struct elemento *punt_lista;
```

Le due funzioni vengono chiamate in sequenza dallo stesso `main`; il programma completo è riportato nel Listato 14.1.

```
punt_lista = crea_lista();
     visualizza_lista(punt_lista);
```

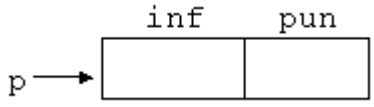
La procedura `crea_lista` restituisce al `main` il puntatore alla lista che è assegnato a `punt_lista` e che viene successivamente passato a `visualizza_lista`.
La funzione `crea_lista` è di tipo puntatore a strutture `elemento`. Non prevede il passaggio di valori dal chiamante, per cui una sua dichiarazione più appropriata sarebbe stata:

```
struct elemento *crea_lista(void);
```

dove `void` esplicita la mancanza di parametri. Nel nostro caso, dunque, il compilatore potrebbe segnalare un *warning* (attenzione!).
La funzione `crea_lista` deve comprendere la dichiarazione di `p`, puntatore alla testa della lista, e di `paus`, puntatore ausiliario, che permette la creazione degli elementi successivi al primo, senza perdere il puntatore iniziale alla lista. In primo luogo si deve richiedere all'utente di inserire il numero di elementi da cui è composta la lista. Questa informazione viene memorizzata nella variabile `n`, di tipo `int`. Dopo di ciò, se `n` è uguale a zero, si assegna a `p` il valore `NULL`, che corrisponde a lista vuota. In questo caso il sottoprogramma termina. Si osservi che `n` è una variabile locale alla funzione.
Se il numero di elementi (valore di `n`) è maggiore di zero `crea_lista` deve creare il *primo elemento*:

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

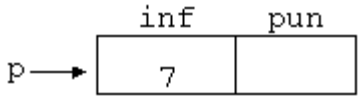
L'operatore `sizeof` restituisce la quantità di memoria occupata da un elemento e `malloc` alloca uno spazio corrispondente di memoria libera. Successivamente viene effettuato un *cast* del valore ritornato da `malloc`, in modo da trasformarlo in un puntatore allo spazio allocato che viene assegnato a `p`.



Per richiamare `malloc` si deve includere nel programma il riferimento alla libreria `malloc.h` e/o `stdlib.h`; in implementazioni del C meno recenti la libreria da includere è stddef.h. Si richiede all'utente di inserire la prima informazione che viene assegnata `p->inf`, campo `inf` del primo elemento della lista:

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

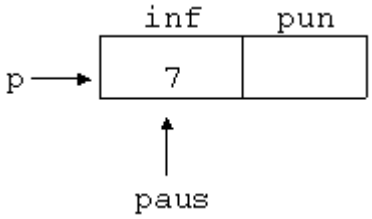
Si supponga che l'utente abbia passato precedentemente il valore 3 come lunghezza della sequenza e successivamente immetta i seguenti valori: 7, 3, 20; dopo la prima immissione avremo la seguente situazione:



dopo di che viene assegnato a `paus` il valore di `p`:

```
paus = p;
```

Come per ogni altro tipo di variabile semplice, è possibile fare assegnamenti fra puntatori purché si riferiscano allo stesso tipo di oggetto. Da questo momento sia `p` sia `paus` (puntatore ausiliario) fanno riferimento al primo elemento della lista.



Finita la gestione del primo elemento, deve iniziare un ciclo per la creazione degli *elementi successivi al primo*. Questo ciclo si ripete *n*−1 volte, dove *n* è la lunghezza della sequenza in ingresso. Il ciclo è così costituito:

```
for(i = 2; i<=n; i++) {
```
a) *crea il nuovo elemento concatenato al precedente*
b) *sposta di una posizione avanti* paus
c) *chiedi all'utente la nuova informazione della sequenza*; *inserisci l'informazione nel campo* `inf` *del nuovo elemento*
```
}
```

Nel caso dell'esempio proposto, il ciclo si ripete due volte (da `i=2` a `i=n=3`); a ogni iterazione le operazioni descritte corrispondono a:

```
a) paus->pun = (struct elemento *)malloc(sizeof(struct elemento));
b) paus = paus->pun;
c) printf("\nInserisci la %d informazione: ", i);
   scanf("%d", &paus->inf);
```

In a) viene creato un nuovo elemento di lista, connesso al precedente. In questo caso il puntatore ritornato dall'operazione di allocazione di memoria e successivo *cast* viene assegnato al campo puntatore della variabile puntata da `paus`. Infatti `paus->` specifica che si tratta dell'elemento puntato da `paus` e `pun` indica il campo della struttura cui si fa riferimento (Figura 14.3a). In b) viene attualizzato il puntatore ausiliario `paus`, in modo da farlo puntare all'elemento successivo (Figura 14.3b). In c) viene richiesta all'utente l'immissione del prossimo valore della sequenza e tale valore viene assegnato al campo informazione dell'elemento (Figura 14.3c). Il ciclo verrà ripetuto un'altra volta ottenendo il risultato di Figura 14.4.

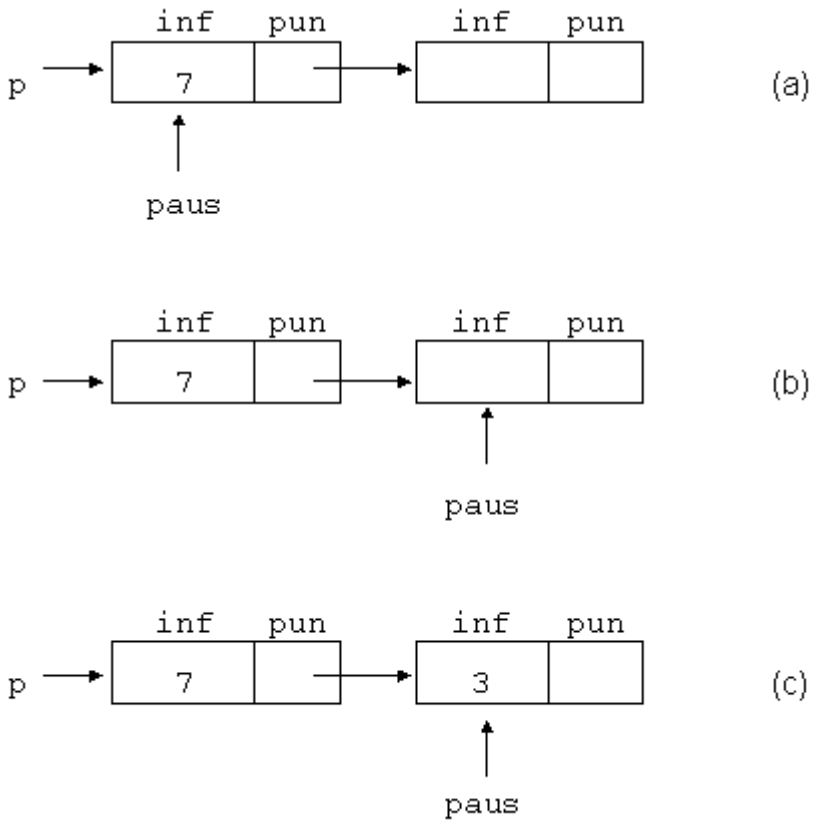


Figura 14.3 Sequenza di creazione del secondo elemento della lista

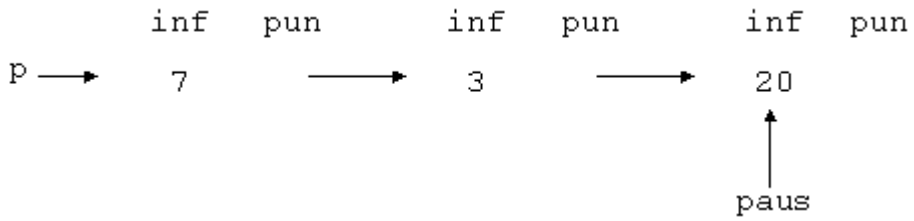


Figura 14.4 Situazione dopo l'ultimo ciclo

Infine la funzione assegna al campo puntatore dell'ultimo elemento il valore `NULL` e termina passando al chiamante il valore di `p`, cioè il puntatore alla lista:

```
return(p);
```

Il controllo passa quindi al `main`, che ora dispone della lista configurata come in Figura 14.5. Le variabili `paus` e `n` non esistono più perché sono state dichiarate locali alla procedura.

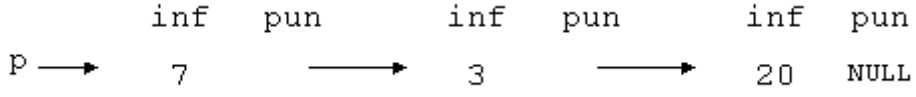


Figura 14.5 Lista completa

La procedura `visualizza_lista` effettua la scansione. È previsto un ciclo che visualizzi il campo informazione di ogni elemento a iniziare dal primo; la funzione è stata dichiarata di tipo `void` poiché non restituisce un valore di ritorno.

Il `main` passa alla procedura il puntatore iniziale alla lista:

```
visualizza_lista(punt_lista);
```

Il parametro attuale `punt_lista` corrisponde al parametro formale `p`. Per effettuare la scansione della lista utilizziamo il seguente ciclo:

```
while(p!=NULL) {
  printf("%d", p->inf);    /* Visualizza il campo
                              informazione */
  printf("---> ");
  p = p->pun;              /* Scorri di un elemento
                              in avanti */
}
```

Il ciclo di scansione è controllato dal test sopra il puntatore `p`: se `p!=NULL` continua l'iterazione. Questo controllo ci è permesso perché abbiamo avuto cura di porre il segnale di fine lista nella funzione `crea_lista`. La scansione degli elementi è consentita dall'operazione di assegnamento:

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

Si provi a eseguire manualmente l'algoritmo di scansione sulla lista di Figura 14.5. La procedura funziona anche nel caso particolare di lista vuota. Non c'è bisogno di un puntatore ausiliario, dato che il riferimento all'inizio della lista è nella variabile `punt_lista` del `main`, mentre `p` è una variabile locale al sottoprogramma ▬.

```
/* Accetta in ingresso una sequenza di interi e li memorizza in
   una lista. Il numero di interi che compongono la sequenza
   è richiesto all'utente. La lista creata viene visualizzata */

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

/* struttura degli elementi della lista */
struct elemento {
  int inf;
  struct elemento *pun;
};

struct elemento *crea_lista();
void visualizza_lista(struct elemento *);

main()
{
struct elemento *punt_lista;  /* Puntatore alla testa
                                 della lista           */
punt_lista = crea_lista();    /* Chiamata funzione per
                                 creare la lista       */
visualizza_lista(punt_lista); /* Chiamata funzione per
                                 visualizzare la lista */
}


/* Funzione per l'accettazione dei valori immessi
   e la creazione della lista. Restituisce il puntatore alla testa */
struct elemento *crea_lista()
{
struct elemento *p, *paus;
int i, n;
```

```
printf("\n Di quanti elementi è composta la sequenza?  ");
scanf("%d", &n);

if(n==0)  p = NULL;           /* lista vuota */
else
{
  /* Creazione del primo elemento */
  p = (struct elemento *)malloc(sizeof(struct elemento));
  printf("\nInserisci la 1 informazione: ");
  scanf("%d", &p->inf);
  paus = p;

  /* creazione degli elementi successivi */
  for(i=2; i<=n; i++) {
   paus->pun = (struct elemento *)malloc(sizeof(struct elemento));
   paus = paus->pun;
   printf("\nInserisci la %d informazione: ", i);
   scanf("%d", &paus->inf);
  }
  paus->pun = NULL;           /* Marca di fine lista */
}
return(p);
}


/* Funzione per la visualizzazione della lista.
   Il parametro in ingresso è il puntatore alla testa */
void visualizza_lista(struct elemento *p)
{
printf("\npunt_lista---> ");

/* Ciclo di scansione della lista */
while(p!=NULL) {
  printf("%d", p->inf);    /* Visualizza il campo informazione */
  printf("---> ");
  p = p->pun;              /* Scorri di un elemento in avanti  */
}
printf("NULL\n\n");
}
```

Listato 14.1 Creazione e visualizzazione di una lista