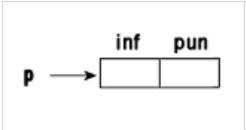
Gestione di una lista – II

17 marzo 2006

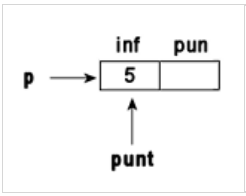
Fabrizio Ciacchi

Vediamo che è stata usata la funzione **malloc()** insieme a **sizeof** per allocare dinamicamente la memoria necessaria ad un elemento della lista (vedere lezione 14); il valore ritornato viene castato ad un puntatore allo spazio allocato, che viene assegnato a p.



Assumendo che sia stato inserito un valore di n (il numero di elementi della lista) uguale a 3; nella prima iterazione, viene creato il primo elemento della lista con il codice sopra esposto e viene assegnato (tramite tastiera), ad esempio, il valore 5;

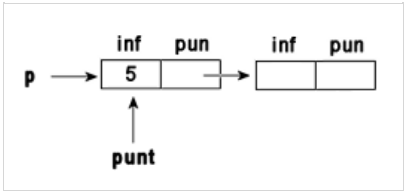
Successivamente viene creato un altro puntatore alla prima posizione, di nome “punt”, tale puntatore ausiliario, servirà per scorrere gli elementi della lista e mantenere il collegamento all’ultimo elemento inserito.



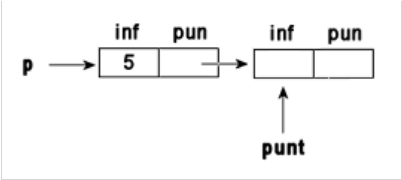
Proseguendo con l’esecuzione del codice, arriviamo ad inserire il secondo ed il terzo elemento (grazie al ciclo for):

```
      /* creazione elementi successivi */
      for(i=2; i<=n; i++)
      {
         punt->pun = (struct elemento *)malloc(sizeof(struct elemento));
         punt = punt->pun;
         printf(“nInserisci il %d elemento: “, i);
         scanf(“%d”, & punt->inf);
      } // chiudo il for
      punt->pun = NULL; // marcatore fine lista
   } // chiudo l’if-else
   return(p);
} // chiudo la funzione
```

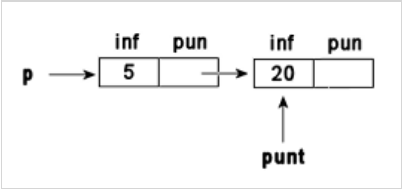
Per prima cosa viene creato un altro oggetto della lista, identificato con punt->pun,



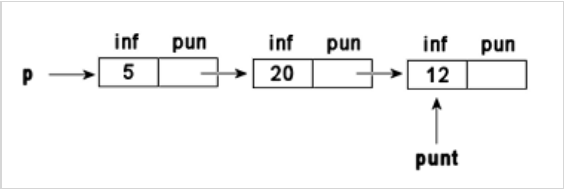
poi “punt”, il puntatore ausiliario, viene fatto puntare, non più al primo elemento, bensì al secondo, all’atto pratico “punt” diventa il puntatore dell’oggetto da lui puntato (punt = punt->pun;).



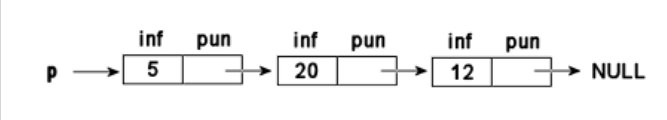
Quindi viene inserito il campo informazione dell'elemento tramite l'input da tastiera dell'utente; in questo caso viene inserito il valore 20;



La procedura, ovviamente, si ripete per il terzo elemento, e, se vi fossero stati altri elementi da aggiungere, si sarebbe ripetuta fino all'inserimento dell'ultimo elemento;



All'ultimo elemento, alla fine del ciclo, viene fatto puntare il valore NULL che serve come marcatore di fine lista.



La funzione **visualizza _lista()** risulta essere molto più semplice di quanto si pensi; essa prende in input la lista da scorrere, viene considerato il primo oggetto puntato da p (che è il primo elemento) e viene stampata la sua informazione; l'iterazione inizia e si assegna a p il suo oggetto puntato, ovvero il secondo elemento, di cui verrà stampata l'informazione relativa; il ciclo continua fino a quando il puntatore p non vale NULL, cioè non coincide con il marcatore di fine lista, momento in cui si esce dal ciclo e la funzione visualizza_lista() termina il suo compito.

```
void visualizza_lista(struct elemento *p)
{
   printf("nlista –> ");

   /* ciclo di scansione */
   while(p != NULL)
   {
      printf("%d", p->inf); // visualizza l'informazione
      printf(" –> ");
      p = p->pun; // scorre di un elemento
   }

   printf("NULLnn");
}
```

ESEMPIO PRATICO

(**Visualizza il sorgente completo**)

Le liste vengono usate ampiamente all'interno del programma, perché permettono una lunghezza dinamica per il numero di contatti presenti. Qui sotto proponiamo le parti di codice principali per la creazione e l'uso di tale lista:

```
38      // Creo la struttura per creare la lista
39      struct elemento
40      {
41        t_contatto inf;
42        struct elemento *pun;
43      };
[...]
[Dichiaro la lista vuota]
63      struct elemento *lista = NULL;
[...]
[Utilizziamo delle funzioni per modificare la lista]
110           lista = aggiungiContatto(lista);
[...]
[Parte di codice che inserisce i nuovi elementi]
255         /* creazione elementi successivi */
256         // Alloco la memoria necessaria
257         punt = (struct elemento *)malloc(sizeof(struct elemento));
258         // Metto daInserire nell'informazione del puntatore

259         punt->inf = daInserire;
```

```
260         // Metto il puntatore in testa alla lista
261         punt->pun = p;
```

INTRODUZIONE INPUT E OUTPUT SU FILE ›

1 ... 34 35 36 ... 47

TUTTE LE LEZIONI