7.10 La scomposizione funzionale

Vediamo ora alcuni criteri euristici per la progettazione di una funzione in C.
Il primo criterio riguarda la scelta del nome. Il nome di una funzione deve esprimere in sintesi la semantica, cioè qual è il compito della funzione. Saranno allora considerati opportuni nomi del tipo:

- `calcola_media`
- `acquisisci_valore`
- `converti_stringa`
- `tx_dati`
- `rx_dati`

mentre si sconsigliano nomi del tipo:

```
x_139       /* oscuro */
gestore     /* generico */
trota       /* umoristico ma poco significativo */
```

La scelta del nome è anche un test della bontà della funzione. Se non si trova un nome che descrive sinteticamente il compito della funzione, probabilmente quest'ultima fa troppe cose, e quindi potrebbe essere ulteriormente scomposta, oppure non ha un compito preciso, nel qual caso potrebbe valere la pena eliminarla!
Una volta scelto il nome di una funzione si definiscono gli ingressi, che possono essere passati esplicitamente per valore o implicitamente per mezzo di variabili globali. In generale è sconsigliato, anche per problemi di leggibilità, avere una lista di parametri esageratamente nutrita. I parametri attuali e le variabili locali hanno vita temporanea, vengono creati all'atto dell'esecuzione della funzione e rimossi all'atto del ritorno al chiamante. Essi vengono posti dinamicamente in memoria; si ricordi che avere molti parametri significa avere molto consumo di memoria e di tempo di elaborazione.
La zona di memoria riservata alle chiamate di funzione viene gestita con la logica di una *pila* (*stack*), di cui parleremo diffusamente nel Capitolo 14 ■. Cerchiamo comunque di rappresentare quello che accade in memoria durante il ciclo di vita di una funzione.
Quando viene invocata una funzione il sistema alloca uno spazio di memoria libero in testa alla pila, spazio riservato ai parametri formali e alle variabili locali. Per esempio, nel programma che calcola le potenze, dopo che il `main` ha chiamato la funzione `pote` la situazione è quella della Figura 7.4*a*. Se l'utente ha immesso il valore 2, `pote` chiama la funzione `quad` per determinare il quadrato del numero immesso e il sistema alloca spazio per i suoi dati (Figura 7.4*b*). Quando `quad` restituisce il controllo a `pote` la situazione ritorna a essere ancora quella di Figura 7.4*a*.

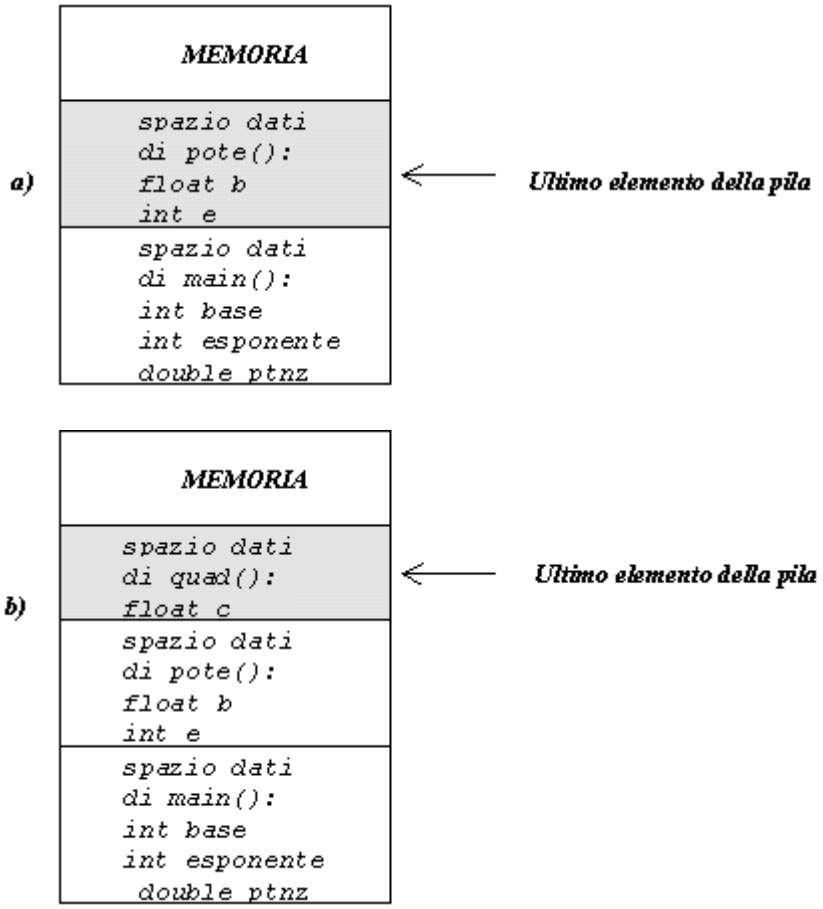

Figura 7.6 Allocazione dinamica di spazio di memoria sulla pila

Quindi, per esempio, definire una funzione come:

```
int f(void)
{
   char a[1000000];
   ...
}
```

potrebbe creare qualche problema! Ecco dunque una possibile eccezione alla sacre regole della programmazione strutturata: nei casi in cui una funzione debba lavorare su strutture di memoria costose in termini di occupazione è preferibile usare variabili globali.
L'ultima osservazione riguarda le uscite di una funzione. Abbiamo visto che in C una funzione ritorna al più un valore. Come ci si deve comportare allora quando è necessario che ci sia più di un valore di ritorno? Esistono due soluzioni:

1. 1. usare delle variabili globali;
2. 2. rendere note alla funzione delle locazioni in cui andare a depositare le uscite.

Per saperne di più su questo secondo tipo di soluzione, si rimanda al Capitolo 9 sui puntatori .