6.2 Esempi di uso delle stringhe

Il programma del Listato 6.2 copia il contenuto di una stringa in un'altra.

```
/* Copia di una stringa su un'altra */

#include <stdio.h>

char frase[] = "Analisi, requisiti ";

main()
{
int i;
char discorso[80];
for(i=0; (discorso[i]=frase[i])!='\0'; i++)
  ;
printf(" originale: %s \n copia:     %s \n", frase, discorso);
}
```

Listato 6.2 Copia di un array di caratteri

La variabile intera `i` viene utilizzata come indice degli array, per fare la scansione delle due stringhe carattere per carattere e per effettuare la copia. Nell'istruzione `for` viene inizializzato il valore di `i` a 0:

```
i = 0;
```

Successivamente il ciclo assegna a `discorso[0]` il valore di `frase[0]` e controlla che tale valore non sia uguale a `\0` (marca di fine stringa), nel qual caso il ciclo ha termine. A ogni nuova iterazione il valore di `i` viene incrementato di 1:

```
i++;
```

Viene quindi assegnato a `discorso[i]` il valore di `frase[i]`,

```
(discorso[i]=frase[i])
```

e controllato che il carattere in questione non sia `\0`:

```
(discorso[i]=frase[i])!='\0'
```

nel qual caso l'iterazione ha termine. Si noti che anche il carattere terminatore viene copiato nell'array `discorso`, che sarà così correttamente delimitato.

L'istruzione `printf` stampa la stringa di partenza e quella di arrivo, per cui l'esecuzione del programma visualizzerà:

```
originale: Analisi, requisiti
     copia:     Analisi, requisiti
```

✓ NOTA

Abbiamo evidenziato il fatto che il costrutto `for` non contiene alcuna istruzione esplicita all'interno del ciclo ponendo il punto e virgola nella riga successiva:

```
for(i=0; (discorso[i]=frase[i])!='\0'; i++)
  ;
```

Questa una notazione è utilizzata spesso in C. In realtà, come abbiamo visto esaminando il programma, qualcosa di concreto accade a ogni iterazione: un elemento di `frase` viene assegnato a `discorso`; l'operazione relativa è però contenuta (nascosta) all'interno di un'espressione della parte controllo del `for`. È chiaro che si poteva ottenere lo stesso risultato scrivendo:

```
for(i=0; frase[i]!='\0'; i++)
  discorso[i]=frase[i];
discorso[i]='\0';
```

Anche se questa notazione è più familiare, si esorta a utilizzare la precedente per sfruttare al meglio le caratteristiche del C.

Se si desidera copiare soltanto alcuni caratteri della prima stringa sulla seconda si deve modificare l'istruzione `for` del Listato 6.2. Scrivendo

```
for(i=0; ((discorso[i]=frase[i])!='\0') && (i<7); i++)
```

si è inserita la condizione `i<7` messa in AND (`&&`) con la verifica di fine stringa; in questo modo verranno copiati solamente i primi sette caratteri. L'istruzione `printf` visualizzerà allora:

```
originale: Analisi, requisiti
     copia:     Analisi
```

Il programma del Listato 6.3 permette invece di aggiungere a una variabile stringa i caratteri presenti in un'altra.

```
/* Concatenazione di due stringhe */

#include <stdio.h>

char frase[160] = "Analisi, requisiti ";
```

```
main()
{
char dimmi[80];
int i, j;

printf("Inserisci una parola: ");
scanf("%s", dimmi);
for(i=0; (frase[i])!='\0'; i++)
  ;
for(j=0; (frase[i]=dimmi[j])!='\0'; i++,j++)
  ;
printf("frase: %s \n", frase);
}
```

Listato 6.3 Concatenazione di array di caratteri

In questo caso indichiamo esplicitamente il numero di elementi (160) che compongono la variabile `frase`, poiché desideriamo definire un array che possa contenere più caratteri di quelli presenti nella stringa assegnatagli all'inizio (20):

```
char frase[160] = "Analisi, requisiti ";
```

In questo modo `frase` potrà contenere i caratteri che gli verranno concatenati. La prima istruzione `printf` richiede all'utente una stringa e `scanf` la inserisce nell'array di caratteri `dimmi`. In generale non si conosce il numero di caratteri che attualmente costituiscono la stringa di partenza, per cui si deve scorrerla fino a posizionare l'indice sul carattere terminatore:

```
for(i=0; (frase[i])!='\0'; i++)
       ;
```

Alla fine del ciclo `i` conterrà l'indice dell'elemento del vettore dov'è presente il carattere `\0`. Nel caso specifico, avendo assegnato a `frase` la stringa `"Analisi, requisiti "`, `i` avrà valore 20.

Adesso dobbiamo assegnare agli elementi successivi di `frase` il contenuto di `dimmi`:

```
for(j=0; (frase[i]=dimmi[j])!='\0'; i++,j++)
       ;
```

L'indice `j` scorre `dimmi` a partire dalla prima posizione, mentre `i` scorre `frase` a partire dal suo carattere terminatore; alla prima iterazione il carattere `\0` di `frase` viene sostituito dal primo carattere di `dimmi`. A ogni ciclo successivo viene assegnato a `frase[i]` il valore di `dimmi[j]`. All'ultima iterazione il carattere `\0` di `dimmi` viene posto in `frase`, così da chiuderla correttamente.

L'istruzione `printf` visualizza il nuovo contenuto di `frase`.

```
printf("frase: %s \n", frase);
```

Osserviamo di seguito una possibile esecuzione del programma.

```
Inserisci una parola: funzionali
     frase: Analisi, requisiti funzionali
```

In Figura 6.2, A e B corrispondono agli stati degli array immediatamente prima e dopo l'esecuzione del secondo `for` del programma che effettua la concatenazione.

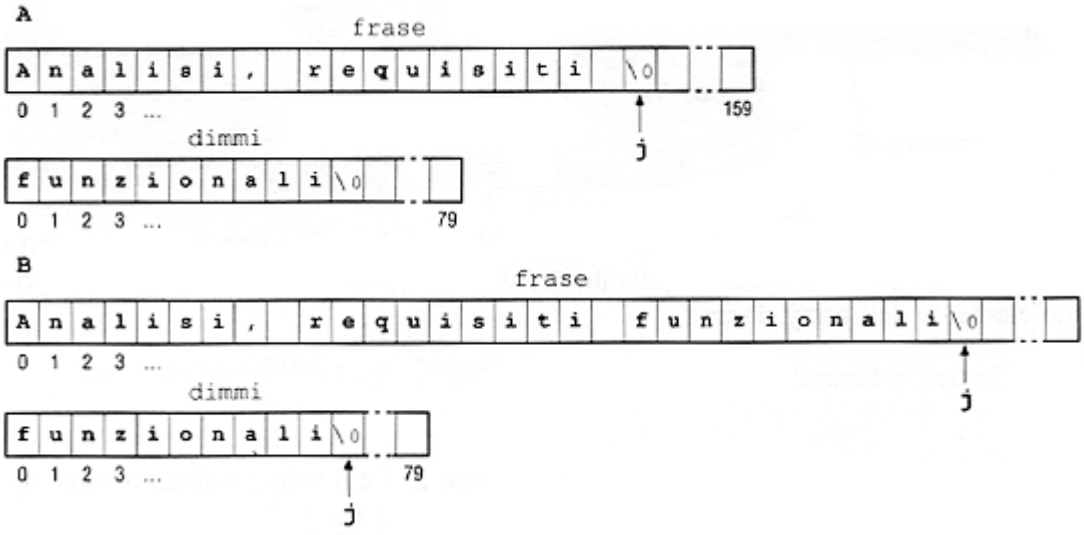

Figura 6.2 Stato degli array prima e dopo la concatenazione

Un altro modo per memorizzare una stringa è l'uso, all'interno di un ciclo, della funzione `getchar`, che cattura il carattere passatole in ingresso (Listato 6.4).

```
/* Concatenazione di due stringhe
   introduzione della seconda stringa con getchar */

#include <stdio.h>

char frase[160] = "Analisi, requisiti ";

main()
{
char dimmi[80];
int i, j;

printf("Inserisci una parola: ");
for(i=0; (dimmi[i]=getchar())!='\n'; i++)
  ;
dimmi[i]='\0';
for(i=0; frase[i]!='\0'; i++)
  ;
for(j=0; (frase[i]=dimmi[j])!='\0'; i++,j++)
  ;
printf(" frase: %s \n", frase);
}
```

Listato 6.4 Immissione di caratteri con `getchar()`

Il ciclo `for` che sostituisce l'istruzione `scanf` inizializza l'indice `i` a zero, cattura il carattere passato da tastiera mediante la funzione `getchar` e lo inserisce nel primo elemento dell'array `dimmi`:

```
for(i=0; (dimmi[i]=getchar())!='\n'; i++)
        ;
```

A ogni iterazione il valore di `i` viene incrementato di 1 e il valore immesso viene inserito nel corrispondente elemento dell'array. Il ciclo si ripete finché il carattere immesso è diverso da `\n`, cioè fino a quando l'utente non batte un Invio.

La stringa memorizzata in `dimmi` non contiene il carattere terminatore, che va esplicitamente assegnatogli nella posizione appropriata:

```
dimmi[i] = '\0';
```

È chiaro che potremmo decidere d'interrompere l'inserimento al verificarsi di un altro evento; per esempio, quando l'utente batte un punto esclamativo. In questo modo potremmo memorizzare più linee nello stesso array: ogni Invio dato dal terminale corrisponde infatti all'assegnamento di un `\n` a un elemento dell'array; evidentemente una successiva visualizzazione dell'array mostrerebbe la stringa con gli accapo inseriti dall'utente.

> ✓ **NOTA**
> Nel programma abbiamo definito `dimmi` di 80 caratteri. Se l'utente ne inserisse un numero maggiore, come abbiamo già evidenziato, i sovrabbondanti andrebbero a sporcare zone contigue di memoria centrale. Il C non fa infatti nessun controllo automatico del rispetto dei margini dell'array. È il programmatore che si deve preoccupare di verificare che gli assegnamenti vengano effettuati su elementi definiti dell'array, per cui un più corretto ciclo d'inserimento del programma sarebbe:
>
> ```
> for(i=0; ((dimmi[i]=getchar())!='\n') && (i<80)) ;i++)
>   ;
> ```
>
> Il ciclo prosegue finché il carattere catturato è diverso da `\n` e contemporaneamente `i` è minore di 80.

Scriviamo adesso un programma che confronta due stringhe rivelando se la prima è uguale, maggiore o minore della seconda (Listato 6.5). L'ordinamento seguito è quello definito dal codice di rappresentazione dei caratteri, che nella maggior parte delle macchine è il codice ASCII.

```
#include <stdio.h>

/* Confronto fra due stringhe */

char prima[160] = "mareggiata";

main()
{
char seconda[80];
int i;

printf("Inserisci una parola: ");
for(i=0; ((seconda[i]=getchar()) != '\n') && (i<80) ;i++)
  ;
seconda[i]='\0';
for(i=0; (prima[i] == seconda[i]) && (prima[i] != '\0') &&
   (seconda[i] != '\0'); i++)
  ;
if(prima[i]==seconda[i])
  printf("Sono uguali\n");
else
  if(prima[i]>seconda[i])
    printf("La prima è maggiore della seconda\n");
  else
    printf("La seconda è maggiore della prima\n");
}
```

Listato 6.5 Confronto fra array di caratteri

L'istruzione `for`

```
for(i=0; (prima[i]==seconda[i]) && (prima[i]!='\0') &&
```

```
        (seconda[i]!='\0'); i++);
```

scorre in parallelo gli elementi dei due array e li confronta; il ciclo si interrompe quando `prima[i]` non risulta essere uguale a `seconda[i]` oppure quando finisce una delle due stringhe. L'`if` seguente serve a determinare la ragione per cui il ciclo `for` si è interrotto; si noti che l'unica possibilità per cui `prima[i]` è uguale a `seconda[i]` si presenta quando entrambi sono uguali a `\0`, il che significa che le stringhe hanno la stessa lunghezza e sono uguali.