9.4 Passaggio di parametri per indirizzo

Abbiamo osservato nel precedente capitolo che in C non è possibile passare un array a una funzione. Eppure esistono molti casi in cui è necessario non solo passare un array ma anche restituire una struttura dati più complessa della semplice variabile `char` o `int`.
All'apparenza le funzioni sembrano essere limitate dal meccanismo del passaggio parametri per valore. Il programmatore C risolve questa apparente pecca con un metodo semplice: passa per valore l'indirizzo della variabile – array o altro – che si vuol leggere o modificare tramite la funzione. Passare un indirizzo a una funzione significa renderle nota la locazione dell'oggetto corrispondente all'indirizzo. In tale maniera le istruzioni all'interno di una funzione possono modificare il contenuto della variabile il cui indirizzo è stato passato alla funzione. Questo meccanismo è noto con il nome di *passaggio di parametri per indirizzo*.
Consideriamo, per esempio, nel Listato 9.1 la funzione `scambia`, che ha l'effetto di scambiare il valore dei suoi parametri.

```
#include <stdio.h>

void scambia(int, int);

main()
{
  int x, y;

  x = 8;
  y = 16;
  printf("Prima dello scambio\n");
  printf("x = %d, y = %d\n", x, y);

  scambia(x, y);

  printf("Dopo lo scambio\n");
  printf("x = %d, y = %d\n", x, y);
}


/* Versione KO di scambia */
void scambia(int a, int b)
{
  int temp;

  temp = a;

  a = b;
  b = temp;
}
```

Listato 9.1 Il passaggio dei parametri per indirizzo

La chiamata di questa funzione non produce alcun effetto sui parametri attuali; cioè la chiamata

```
scambia(x, y);
```

non ha effetto sulle variabili intere `x` e `y`. Infatti i valori di `x` e `y` sono copiati nei parametri formali `a` e `b` e, quindi, sono stati scambiati i valori dei parametri formali, non i valori originali di `x` e `y`! Affinché `scambia` abbia un qualche effetto deve essere modificata in modo da ricevere gli indirizzi, anziché i valori, delle variabili (Listato 9.2). La relativa rappresentazione grafica del passaggio di parametri per indirizzo è data in Figura 9.2.

```
#include <stdio.h>

void scambia(int *, int *);
```

```
main()
{
  int x, y;

  x = 8;
  y = 16;
  printf("Prima dello scambio\n");
  printf("x = %d, y = %d\n", x, y);

  scambia(&x, &y);

  printf("Dopo lo scambio\n");
  printf("x = %d, y = %d\n", x, y);
}

/* Versione OK di scambia */
void scambia(int *a, int *b)
{
  int temp;

  temp = *a;
  *a = *b;
  *b = temp;
}
```

Listato 9.2 Ancora sullo scambio di valori

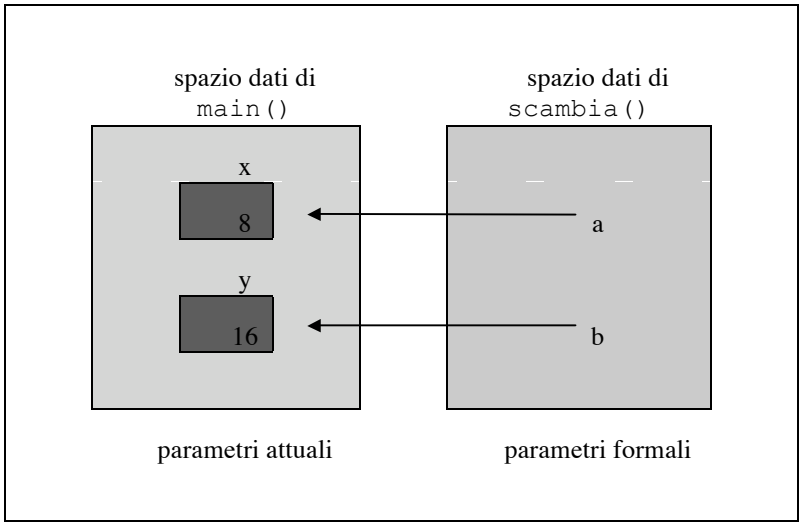

Figura 9.2 Passaggio di parametri per indirizzo

La simbologia `int *a` e `int *b`, usata nella definizione dei parametri formali, dichiara `a` e `b` come variabili di tipo puntatore a un intero. L'invocazione della funzione `scambia` deve essere modificata in modo da passare l'indirizzo delle variabili da scambiare:

```
scambia(&x, &y);
```

La medesima strategia del passaggio per valore di un indirizzo si può sfruttare con gli array (Listato 9.3).

```
#include <stdio.h>

char str[] = "BATUFFO";
int strlen(char *);
```

```
main()
{
printf("la stringa %s è lunga %d\n", str, strlen( str ));
}

int strlen( char *p)
{
  int i = 0;
  while (*p++) i++;
  return i;
}
```

Listato 9.3 Passaggio di un array

L'array dell'esempio è una stringa, cioè un array di `char` che termina con il carattere terminatore `\0`. Con l'inizializzazione

```
char str[] = "BATUFFO";
```

il primo elemento dell'array di caratteri `str` è inizializzato a puntare al primo elemento della costante di tipo stringa `"BATUFFO"`.
L'accorgimento di valutare una stringa per mezzo del puntatore `char *` è particolarmente utile nella scrittura di funzioni che manipolano stringhe. Nell'esempio la funzione `strlen` conta il numero di caratteri (escluso `\0`) di una stringa:

```
int strlen( char *p)
{
  int i = 0;
  while (*p++) i++;
  return i;
}
```

La funzione pone il contatore `i` a zero e comincia a contare caratteri finché non trova il carattere nullo. Una implementazione alternativa di `strlen` che usa la sottrazione di puntatori è:

```
int strlen( char *p )
{
  char *q = p;
  while ( *q++);
  return (q-p-1);
}
```

Le funzioni di manipolazione stringa gestiscono le stringhe sempre per mezzo di puntatori a carattere. Il C fornisce un vasto insieme di funzioni di manipolazione stringa, dichiarate nel file di *include* `<string.h>`. Perciò, per poter usare la libreria di manipolazione stringhe del C, occorre premettere la direttiva:

```
#include <string.h>
```

Si riportano di seguito le dichiarazioni delle più importanti funzioni di manipolazione stringa, alcune delle quali utilizzate nel Capitolo 5, allo scopo di riflettere sull'uso che viene fatto del passaggio dei parametri per indirizzo.

`char *strcat(char *`*`string1`*`, const char *`*`string2`*`);`
Concatena le stringhe *`string1`* e *`string2`* attaccando *`string2`* in coda a *`string1`*.

`char *strncat(char *`*`string1`*`, const char *`*`string2`*`, int n);`
Concatena le stringhe *`string1`* e *`string2`* attaccando *n* caratteri della stringa *`string2`* in coda a *`string1`*.

`int strcmp(const char *`*`string1`*`, const char *`*`string2`*`);`

Confronta *string1* con *string2*. Ritorna 0 se le stringhe sono identiche, un numero minore di zero se *string1* è minore di *string2*, e un numero maggiore di zero se *string1* è maggiore di *string2*.

`int strncmp(const char *`*`string1`*`, const char *`*`string2`*`, int n);`
Confronta i primi *n* caratteri di *string1* con *string2*. Ritorna 0 se le sottostringhe di *n* caratteri sono identiche, un numero minore di zero se *sottostring1* è minore di *sottostring2*, e un numero maggiore di zero se *sottostring1* è maggiore di *sottostring2*.

`char *strcpy(char *`*`string1`*`, const char *`*`string2`*`);`
Copia *string2* su *string1*.

`char *strncpy(char *`*`string1`*`, const char *`*`string2`*`, int n);`
Copia i primi *n* caratteri di *string2* su *string1*.

`int strlen(const char *`*`string`*`);`
Conta il numero di caratteri di *string*, escluso il carattere nullo.

`char *strchr(const char *`*`string`*`, int c);`
Ritorna il puntatore alla prima occorrenza in *string* del carattere `c`.

`char *strrchr(const char *`*`string`*`, int c);`
Ritorna il puntatore all'ultima occorrenza del carattere `c` nella stringa *string*.

`char *strpbrk(const char *`*`string1`*`, const char *`*`string2`*`);`
Ritorna un puntatore alla prima occorrenza della stringa *string2* in *string1*.

`int strspn(const char *`*`string1`*`, const char *`*`string2`*`);`
Trova la posizione del primo carattere in *string1* che non appartiene all'insieme di caratteri di *string2*.

`char *strtok(char *`*`string1`*`, const char *`*`string2`*`);`
Trova la prossima sequenza di caratteri (*token*) circoscritta dai caratteri *string2* nella stringa *string1*.

Il lettore può facilmente verificare l'uso di puntatori a `char` che si ha nelle funzioni di manipolazione stringa. Si osservi inoltre l'uso della parola chiave C `const`. Essa sta a indicare che, anche se l'indirizzo `char *` è passato alla funzione, la funzione non può andare a modificare le locazioni di memoria puntate da tale indirizzo, può solamente andare a leggere le locazioni puntate da quell'indirizzo. Con tale precisazione, semplicemente leggendo il prototype della funzione si capisce quali sono le stringhe che vengono modificate dalla corrispondente funzione di manipolazione.