Azzolini Riccardo 2021-03-15

Puntatori a funzione

1 Puntatori a funzione

In generale, un puntatore è una variabile che contiene un indirizzo. Allora, ha senso che possano esistere puntatori a tutti gli oggetti dotati di indirizzo. Finora, si è visto che le variabili hanno un indirizzo, mentre ad esempio le espressioni non lo hanno. Un altro oggetto caratterizzato da un indirizzo sono le *funzioni*: una funzione è rappresentata da una sequenza di istruzioni in memoria, dunque il suo indirizzo è l'indirizzo della prima istruzione (quella a cui si salta quando la funzione viene chiamata). Perciò, il linguaggio C consente la definizione di **puntatori a funzione**.

1.1 Definizione

Esistono tanti tipi di puntatori a funzione — uno per ogni possibile prototipo di funzione (cioè per ogni combinazione di tipo restituito e tipi dei parametri). La sintassi per definire un puntatore a funzione è

tipo_restituito (**nome*)(*tipi_parametri*)

dove:

- *tipo_restituito* è il tipo di valore restituito (eventualmente `void`) dalle funzioni a cui il puntatore potrà puntare;
- *nome* è il nome della variabile puntatore che si sta definendo;
- *tipi_parametri* è la sequenza (eventualmente `void`) di tipi dei parametri formali accettati dalle funzioni a cui il puntatore potrà puntare.

Questa sintassi è ispirata alla sintassi dei prototipi di funzione e al modo in cui il puntatore viene usato una volta definito: (**nome*) è l'oggetto puntato da *nome*, ed è appunto una funzione con un prototipo corrispondente a quello indicato nella definizione di *nome*.

Ad esempio, data una funzione `f` con prototipo

```
int f(short, double);
```

un puntatore chiamato `ptr_to_f` che è del tipo giusto per puntare a `f` viene definito nel modo seguente:

```
int (*ptr_to_f)(short, double);
```

1.2 Uso

In C, così come il nome di un array può essere interpretato come un puntatore al suo primo elemento, anche il nome di una funzione corrisponde a un puntatore alla funzione stessa (ovvero alla sua prima istruzione). Di conseguenza, dati il puntatore `ptr_to_f` e la funzione `f` mostrati prima, per assegnare a `ptr_to_f` l'indirizzo di `f` è sufficiente scrivere:[1]

```
ptr_to_f = f;
```

Una volta assegnato un indirizzo, quando si vuole invocare la funzione puntata, si usa prima l'operatore di indirizzamento indiretto per passare dal puntatore alla funzione puntata, e poi si usa la normale sintassi di chiamata con le parentesi. Ad esempio, se `ptr_to_f` contiene l'indirizzo di `f`, l'invocazione

```
int x = (*ptr_to_f)(5, 8.64);
```

equivale a

```
int x = f(5, 8.64);
```

Inoltre, data la corrispondenza tra nomi di funzioni e puntatori, è consentito omettere l'operatore di indirizzamento indiretto (e quindi anche le parentesi intorno all'uso di tale operatore):

```
int x = ptr_to_f(5, 8.64);
```

2 Esempio: ordinamento con ordine parametrico

I puntatori a funzione sono utili, ad esempio, per il riuso del codice, in quanto consentono la scrittura di procedure nelle quali si astraggono alcuni aspetti degli algoritmi implementati, lasciando che essi vengano specificati come parametri.

Il concetto è illustrato dalla seguente procedura `selection_sort`, che implementa l'ordinamento per selezione di un array di interi senza fissare uno determinato ordine, poiché l'ordine viene invece specificato fornendo come parametro un puntatore a un'opportuna funzione di confronto:

```
void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
```

[1] È anche consentito, e del tutto equivalente, scrivere `ptr_to_f = &f`. Ciò è diverso da quanto accade per gli array: se `A` è una variabile di tipo array, allora le espressioni `A` e `&A` *non* sono equivalenti.

```c
void selection_sort(int arr[], int n, int (*comp)(int, int)) {
    for (int i = 0; i < n - 1; i++) {
        // Trova il minimo elemento (secondo comp)
        int min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (comp(arr[j], arr[min_idx])) {
                min_idx = j;
            }
        }
        // Scambia il minimo trovato con il primo elemento
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Così, la stessa implementazione dell'algoritmo può essere usata, ad esempio, per effettuare l'ordinamento in ordine crescente e in ordine decrescente: il programma

```c
#include <stdio.h>

void swap(int *xp, int *yp) {
    // ...
}

void selection_sort(int arr[], int n, int (*comp)(int, int)) {
    // ...
}

int minore(int a, int b) { return a < b; }

int maggiore(int a, int b) { return a > b; }

int main(int argc, const char *argv[]) {
    int v1[] = {3, 1, 5, 2, 7, 4, 6, 9, 8};
    int v2[] = {3, 1, 5, 2, 7, 4, 6, 9, 8};

    selection_sort(v1, 9, minore);
    for (int i = 0; i < 9; i++) {
        printf("v1[%d] = %d\n", i, v1[i]);
    }
    printf("\n");
    selection_sort(v2, 9, maggiore);
    for (int i = 0; i < 9; i++) {
        printf("v2[%d] = %d\n", i, v2[i]);
    }
```

```
    return 0;
}
```

produce l'output

```
v1[0] = 1
v1[1] = 2
v1[2] = 3
v1[3] = 4
v1[4] = 5
v1[5] = 6
v1[6] = 7
v1[7] = 8
v1[8] = 9

v2[0] = 9
v2[1] = 8
v2[2] = 7
v2[3] = 6
v2[4] = 5
v2[5] = 4
v2[6] = 3
v2[7] = 2
v2[8] = 1
```