I Puntatori

I puntatori sono la parte più importante della programmazione in C, quella che permette di lavorare "a basso livello" (cioè agendo su singole istruzioni del processore), mantenendo però una praticità unica nel suo genere.

Cosa è un puntatore? **Un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile**. Quando dichiariamo una variabile, a questa verrà riservato un indirizzo di memoria, ad esempio la posizione 1000; un puntatore contiene, appunto, l'indirizzo di tale variabile (quindi il valore 1000). L'importanza risiede nel fatto che si possono manipolare sia il puntatore che la variabile puntata (cioè la variabile memorizzata a quell'indirizzo di memoria).

Per definire un puntatore è necessario seguire la seguente sintassi:

```
// variabile normale
int variabile;

// puntatore
int *puntatore;
```

L'asterisco (*) viene chiamato **operatore di indirezione o deferenziazione** e restituisce il contenuto dell'oggetto puntato dal puntatore; mentre l'operatore **&** restituisce l'indirizzo della variabile e va usato nella seguente forma:

```
// assegno al puntatore l'indirizzo di variabile
puntatore = &variabile;
```

Per fare un esempio pratico, assumiamo di avere due variabili, **alfa** e **beta** ed un puntatore di nome **pointer**; assumiamo anche che alfa risieda alla locazione di memoria "100", beta alla locazione "200" e pointer alla locazione "1000" e vediamo, eseguendo il codice sotto proposto, il risultato ottenuto:

```
#include <stdio.h>

int main()
{
	int alfa = 4;
	int beta = 7;
	int *pointer;

	pointer = &alfa;
	printf("alfa -> %d, beta -> %d, pointer -> %d\n", alfa, beta, pointer);
	beta = *pointer;
	printf("alfa -> %d, beta -> %d, pointer -> %d\n", alfa, beta, pointer);
	alfa = pointer;
	printf("alfa -> %d, beta -> %d, pointer -> %d\n", alfa, beta, pointer);
	*pointer = 5;
	printf("alfa -> %d, beta -> %d, pointer -> %d\n", alfa, beta, pointer);
}
```

Questo stamperà a video i valori corretti di alfa e beta (4 e 7) e l'indirizzo di alfa memorizzato nel puntatore (100):

alfa -> 4, beta -> 7, pointer -> 100

poi assegniamo a beta il valore dell'oggetto puntato da pointer (alfa), e quindi il valore 4:

alfa -> 4, beta -> 4, pointer -> 100

successivamente assegniamo ad alfa il valore memorizzato in pointer e quindi l'indirizzo di memoria di alfa stesso,

alfa -> 100, beta -> 4, pointer -> 100

continuiamo memorizzando in alfa (l'oggetto puntato da pointer) il valore 5,

alfa -> 5, beta -> 4, pointer -> 100

Esiste anche un aritmetica base dei puntatori ai quali è possibile aggiungere o togliere dei valori, che generalmente vengono rappresentati come blocchi di memoria; per intendersi, un puntatore esiste sempre in funzione del tipo di oggetto puntato, se creo un puntatore ad int, il blocco di memoria vale 4 byte, un puntatore a char, invece, usa blocchi di 1 byte. In questo modo se utilizzo l'operatore **++**, incremento del blocco a seconda del tipo di variabile puntata.

Come mostreremo di seguito i puntatori possono essere usati con successo in combinazione con **funzioni**, **array** e **strutture**.

I puntatori risultano molto utili anche usati con le funzioni. Generalmente vengono passati alle funzioni gli argomenti (le variabili) per **valore** (utilizzando return). Ma questo modo di passare gli argomenti, non modifica gli argomenti stessi e quindi potrebbe risultare limitativo quando, invece, vogliamo che vengano modificate le variabili che passiamo alle funzioni.

Pensiamo ad un caso esemplare, uno su tutti, l'uso della funzione **swap (alfa, beta)** che scambia il valore di alfa con beta e viceversa; in questo caso il valore restituito dalla funzione (con return) non va minimamente ad intaccare i valori delle variabili alfa e beta, cosa che vogliamo, invece, accada per poter effettivamente fare lo scambio. In questo caso si passano alla funzione, non i valori delle variabili, ma il loro indirizzo, trovandoci, quindi, ad operare con i puntatori a tali valori. Facciamo un esempio pratico per chiarire:

```
#include <stdio.h>

void swap(int *apt, int *bpt);

int main()
{
	int alfa = 5;
	int beta = 13;

	printf("alfa -> %d, beta -> %d\n", alfa, beta);

	swap(&alfa, &beta);

	printf("alfa -> %d, beta -> %d\n", alfa, beta);
}

void swap(int *apt, int *bpt)
{
	int temp;
	temp = *apt;
	*apt = *bpt;
	*bpt = temp;
}
```

Una digressione su array e puntatori potrebbe portare via moltissimo tempo; in questo ambito ci limitiamo ad accennare alla correlazione tra i due strumenti a disposizione nel linguaggio C.

L'aspetto che accomuna i puntatori e gli array è sostanzialmente il fatto che entrambi sono memorizzati in locazioni di memoria sequenziali, ed è quindi possibile agire su un array (se lo si vede come una serie di blocchi ad indirizzi di memoria sequenziali) come se si stesse agendo su un puntatore (e viceversa); ad esempio se dichiariamo un array "alfa" ed un puntatore "pointer":

```
// definisco un array ed una variabile intera
	int alfa[20], x;

// creo il puntatore;
	int *pointer;

// puntatore all'indirizzo di a[0]
	pointer = &a[0];
// x prende il valore di pointer, cioè di a[0];
	x = *pointer;
```

Se volessi scorrere, ad esempio, alla posizione i-ma dell'array, basterebbe incrementare il puntatore di i; cioè

pointer + i "è equivalente a" a[i]

Graficamente questo viene rappresentato come:

```
          0  1  2  3  4 .  .. n
          __ __ __ __ __ __ __
alfa      |__|__|__|__|__|__|__|

pointer  +1 +2 .. +i
```

Comunque array e puntatori hanno delle sostanziali differenze:

- Un puntatore è una variabile, sulla quale possiamo eseguire le più svariate operazioni (come l'assegnamento).
- Un array non è una variabile convenzionale (è un contenitore di variabili) ed alcune operazioni da array a puntatore potrebbero non essere permesse.

Nell'uso congiunto di strutture e puntatori, come mostrato nell'esempio seguente:

```
struct PIPPO { int x, y, z; } elemento;
struct PIPPO *puntatore;

puntatore = &elemento;

puntatore->x = 6;
puntatore->y = 8;
puntatore->z = 5;
```

si può notare che abbiamo creato una struttura di tipo PIPPO e di nome "elemento", ed un puntatore ad una struttura di tipo PIPPO. Per accedere ai membri interni della struttura "elemento" abbiamo usato l'operatore **->** sul puntatore alla struttura. Inoltre è possibile utilizzare i puntatori anche per, ad esempio, le liste semplici (tratteremo le liste nella lezione 15), in cui per collegare due elementi, basta scrivere:

```
typedef struct { int inf; ELEMENTO *pun; } ELEMENTO;

ELEMENTO ele1, ele2;

// in questo modo faccio puntare il primo elemento al secondo
ele1.pun = &ele2;
```

ESEMPIO PRATICO

Il largo uso fatto di puntatori fa capire quanto siano importanti questi strumenti per scrivere codice potente (in termini di capacità), ma allo stesso tempo estremamente ridotto (rispetto ad una versione senza puntatori). Ad esempio è possibile modificare gli elementi di una lista (spiegato successivamente) semplicemente "giocando" con i puntatori.

```
415         // Trovato l'elemento gli faccio puntare l'oggetto puntato dal suo
416         // puntatore, in poche parole "salto" l'elemento da eliminare
417         if(subscelta == n)
418         { // IF - OPEN
419
420           aus->pun=aus->pun->pun;
421
422         } else { // ELSE
423
424           // Nel caso in cui il puntatore fosse NULL, per non creare casini
425           // glielo assegniamo direttamente
426           aus=aus->pun;
427
428         } // IF - CLOSE
```