9.1 Definizione di puntatore

A ogni variabile corrisponde un nome, una locazione di memoria, e l'indirizzo della locazione di memoria. Il nome di una variabile è il simbolo attraverso cui si fa riferimento al contenuto della corrispondente locazione di memoria. Così, per esempio, nel frammento di programma:

```
int a;
     ...
     a = 5;
     printf("%d", a);
```

viene assegnato il valore costante 5 alla variabile di tipo intero `a`.

L'operatore `&`, introdotto con la funzione `scanf`, restituisce l'indirizzo di memoria di una variabile. Per esempio l'espressione `&a` è un'espressione il cui valore è l'indirizzo della variabile `a`. Un indirizzo può essere assegnato solo a una speciale categoria di variabili dette *puntatori*, le quali sono appunto variabili abilitate a contenere un indirizzo. La sintassi di definizione è

```
tipo_base *var_punt;
```

dove `var_punt` è definita come variabile di tipo "puntatore a *`tipo_base`*"; in sostanza *`var_punt`* è creata per poter mantenere l'indirizzo di variabili di tipo *`tipo_base`*, che è uno dei tipi fondamentali già introdotti: `char`, `int`, `float` e `double`.
Il tipo puntatore è un classico esempio di tipo derivato; infatti, non ha senso parlare di tipo puntatore in generale, ma occorre sempre specificare a quale tipo esso punta. Per esempio, in questo caso:

```
int   a;
char  c;

int  *pi;
char *pc;

pi = &a;
pc = &c;
```

si ha che `pi` è una variabile di tipo puntatore a `int`, e `pc` è una variabile di tipo puntatore a `char`. Le variabili `pi` e `pc` sono inizializzate rispettivamente con l'indirizzo di `a` e di `c`.

La capacità di controllo di una variabile o, meglio, la capacità di controllo di una qualsiasi regione di memoria per mezzo di puntatori è una delle caratteristiche salienti del C. Il lettore avrà modo di sperimentare quanto si accrescano le capacità del linguaggio con l'introduzione dei puntatori.
Questi ultimi, d'altra parte, sarebbero poca cosa se non esistesse l'operatore unario `*`. L'operatore `*`, detto *operatore di indirezione*, si applica a una variabile di tipo puntatore e restituisce il contenuto dell'oggetto puntato. Se effettuiamo le operazioni

```
a = 5;
c = 'x';
```

in memoria abbiamo la situazione illustrata di seguito.

Le istruzioni

```
printf("a = %d    c = %c", a, c);
printf("a = %d    c = %c", *pa, *pc);
```

hanno esattamente lo stesso effetto, quello di visualizzare:

```
a = 5    c = x
```

Vediamo un altro esempio:

```
char  c1, c2;
char  *pc;
...
c1 = 'a';
c2 = 'b';
printf(" c1 = %c, c2 = %c \n", c1, c2);

pc = &c1;          /* pc contiene l'indirizzo di c1 */
c2 = *pc;          /* c2 contiene il carattere 'a'  */
printf(" c1 = %c, c2 = %c \n", c1, c2);
```

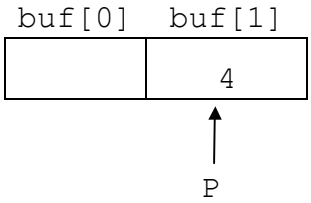
Dopo l'assegnazione `pc=&c1`; i nomi `c1` e `*pc` sono perfettamente equivalenti (*alias*), e si può accedere allo stesso oggetto creato con la definizione `char c1` sia con il nome `c` sia con l'espressione `*pc`. L'effetto ottenuto con l'assegnazione `c2=*pc` si sarebbe ottenuto, equivalentemente, con l'assegnazione

```
c2 = c1;
```

Un ulteriore esempio di uso di puntatori e dell'operatore di indirezione, riferiti a elementi di un array, è il seguente:

```
int  buf[2];
int  *p;
...
p  = &buf[1];
*p = 4;
```

Con il puntatore a intero `p` e l'operatore `*` si è modificato il contenuto della locazione di memoria `buf[1]`, questa volta preposta a contenere un valore di tipo `int`.



Il lettore avrà certo notato che l'operatore `*` è usato nella definizione di variabili di tipo "puntatore a":

```
int  *pi;
char *pc;
```

La notazione è perfettamente coerente con la semantica dell'operatore di indirezione. Infatti, se `*pi` e `*pc` occupano tanta memoria quanto rispettivamente un `int` e un `char`, allora `pi` e `pc` saranno dei puntatori a `int` e a `char`.