# 10 - Puntatori

## Definizione e Concetti Base

Le variabili in un programma rappresentano porzioni di memoria utilizzate per contenere valori elaborati da un algoritmo. Ogni variabile è descritta da un nome, un tipo e un indirizzo di memoria. Sebbene nome e tipo siano esplicitamente dichiarati, l'indirizzo è nascosto al programmatore, ma può essere recuperato usando l'operatore `&`. Ad esempio, con `scanf`, il simbolo `&` permette di accedere all'indirizzo di memoria per salvare un valore.

Un **puntatore** è una particolare variabile che memorizza un indirizzo di memoria invece di un valore diretto. Questo indirizzo punta a un'altra variabile che contiene un valore. Per dichiarare un puntatore si utilizza la sintassi `<tipo>* <nome_variabile>`. Ad esempio:

```
int* a;  // Puntatore a un intero
float* b;  // Puntatore a un float
```

L'asterisco `*` può essere posizionato anche vicino al nome della variabile.

### Caratteristiche dei Puntatori

Un puntatore è caratterizzato dal tipo di dati che punta, dall'indirizzo a cui è legato e dal valore memorizzato in quell'indirizzo. È importante inizializzare i puntatori prima dell'uso, poiché un puntatore non inizializzato contiene un indirizzo casuale che potrebbe causare corruzione della memoria. Per evitare ciò, si può inizializzare un puntatore a `NULL`:

```
int* pi = NULL;
```

## Operatori Relativi ai Puntatori

Il linguaggio C fornisce due operatori fondamentali:

1. **Operatore di indirizzo** `&`: restituisce l'indirizzo di una variabile.
2. **Operatore di indirezione** `*`: accede al valore memorizzato nell'indirizzo a cui punta il puntatore.
   Esempio:

```
int x = 5;
int* p = &x;  // p contiene l'indirizzo di x
int y = *p;   // y contiene il valore di x (5)
```

## Passaggio dei Parametri

Di default, in C, i parametri delle funzioni vengono passati per valore, cioè la funzione lavora su una copia del valore originale, e le modifiche non sono riflessive. Questo comportamento può essere problematico quando si vuole modificare direttamente i dati originali, come nello scambio di valori. I puntatori permettono di simulare il passaggio per riferimento, passando l'indirizzo delle variabili:

```c
void scambia(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 10, y = 20;
    scambia(&x, &y);  // Passo gli indirizzi di x e y
    return 0;
}
```

In questo esempio, i valori di `x` e `y` vengono effettivamente scambiati.

## Applicazioni dei Puntatori

I puntatori sono fondamentali per:

- Modificare direttamente i dati originali in una funzione.
- Restituire più valori da una funzione.

### Esempio Pratico

Un esercizio interessante riguarda la gestione dei tempi sul giro di auto da corsa. Il programma genera casualmente i tempi in millisecondi per cinque auto, converte questi tempi in minuti, secondi e decimi, e li ordina dal più veloce al più lento.

```c
void converti(int millisecondi, int* minuti, int* secondi, int* decimi) {
    *secondi = millisecondi / 1000;
    *decimi = millisecondi % 1000;
    *minuti = *secondi / 60;
    *secondi %= 60;
}

int main() {
    int tempi[5] = {82738, 84219, 83567, 81024, 89012};
    int minuti, secondi, decimi;
    for (int i = 0; i < 5; i++) {
        converti(tempi[i], &minuti, &secondi, &decimi);
        printf("Auto %d: %d'%d''%d\n", i + 1, minuti, secondi, decimi);
    }
```

```
    return 0;
}
```

## Note Importanti

Gli operatori di dereferenziazione ( `*` ) e di indirizzo ( `&` ) sono inversi tra loro. Questo significa che applicarli consecutivamente restituisce il valore originale. Tuttavia, è essenziale evitare errori, come usare puntatori non inizializzati o puntatori che non rispettano il tipo di dato al quale puntano.