Puntatori a puntatori

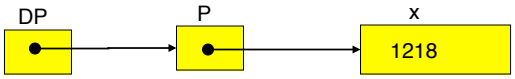
Un puntatore può *puntare* a variabili di tipo qualunque (semplici o strutturate):

➔ può *puntare* anche a un puntatore:

```
[typedef] TipoDato      **TipoPunt;
```

Ad esempio:

```
int x, *P, **DP;
P = &x;
Dp = &P;
**DP=1218;
```



➔ DP è un **doppio puntatore** (o *handle*): *dereferenziando* 2 volte DP, si accede alla variabile puntata dalla "catena" di riferimenti.

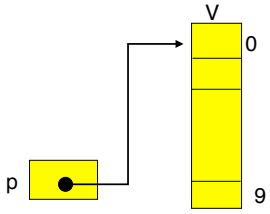
Vettori & Puntatori

Nel linguaggio C, i vettori sono rappresentati mediante puntatori:

➔ il **nome** di una variabile di tipo vettore denota **l'indirizzo del primo elemento** del vettore.

Ad esempio:

```
float V[10]; /*V è una costante di tipo puntatore:
         V equivale a &V[0];
         V e` un puntatore (costante!) a float
         */
float *p;
p=V;   /* p punta a V[0] */
*p=0.15; /* equivale a V[0]=0.15 */
```



Vettori & Puntatori

Nel linguaggio C, i vettori sono rappresentati mediante puntatori:

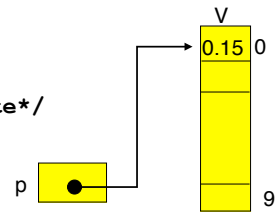
il **nome** di una variabile di tipo vettore denota **l'indirizzo del primo elemento** del vettore.

Ad esempio:

```
float V[10]; /*V è una costante di tipo puntatore:
         V equivale a &V[0];
         V e` un puntatore (costante!) a float
         */
float *p;
p=V;   /* p punta a V[0] */
*p=0.15; /* equivale a V[0]=0.15 */
```

≠

```
V = p; /*ERRORE! V è un puntatore costante*/
```



Operatori *aritmetici* su puntatori a vettori

Nel linguaggio C, gli elementi di un vettore vengono allocati in memoria in **parole consecutive** (cioe`, in celle fisicamente adiacenti), la cui dimensione dipende dal tipo dell'elemento.

➔ Conoscendo l'indirizzo del primo elemento e la dimensione dell'elemento, e` possibile calcolare l'indirizzo di qualunque elemento del vettore:

Operatori *aritmetici* (somma e sottrazione) su puntatori a vettori:

Se V e W sono puntatori ad elementi di vettori ed i è un intero:

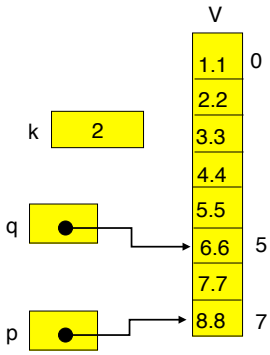
- **(V+i)** restituisce l'indirizzo dell'elemento spostato di i posizioni in avanti rispetto a quello puntato da V;
- **(V-i)** restituisce l'indirizzo dell'elemento spostato di i posizioni all'indietro rispetto a quello puntato da V;
- **(V-W)** restituisce l'intero che rappresenta il numero di elementi compresi tra V e W.

Operatori *aritmetici* su puntatori a vettori

Esempio:

```
#include <stdio.h>

main()
{ float V[8]={1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8};
   int k;
   float *p, *q;
   p=V+7;
   q=p-2;
   k=p-q;
   printf("%f,\t%f,\t%d\n",*p, *q, k);
}
```



➔ **Stampa:** `8.8, 6.6, 2`

Vettori e Puntatori

Durante l'esecuzione di ogni programma C, ogni riferimento ad un elemento di un vettore è tradotto in un puntatore dereferenziato; per esempio:

`V[0]`	**viene tradotto in**	`*(V)`
`V[1]`	**viene tradotto in**	`*(V + 1)`
`V[i]`	**viene tradotto in**	`*(V + i)`
`V[expr]`	**viene tradotto in**	`*(V + expr)`

Esempio:

```
#include <stdio.h>

main ()
{       char a[ ] = "0123456789"; /* a è un vettore di 10 char */
   int i = 5;
   printf("%c%c%c%c%c\n",a[i],a[5],i[a],5[a],(i-1)[a]); /* !!! */
}
```

➔ **Stampa:** **5 5 5 5 4**

NB: Per il compilatore `a[i]` **e** `i[a]` **sono lo stesso elemento**, perché viene sempre eseguita la conversione: `a[i] =>*(a+i)` (senza eseguire alcun controllo né su a, né su i).

Complementi sui puntatori

3

Vettori di puntatori:

Il costruttore [ ] ha precedenza rispetto al costruttore *. Quindi:

`char *a[10];` equivale a `char *(a[10]);`

➔ a è un vettore di 10 puntatori a carattere.

NB: Per un puntatore ad un vettore di caratteri è necessario forzare la precedenza (con le parentesi): `char (* a) [10];`

Puntatori a strutture:

E' possibile utilizzare i puntatori per accedere a variabili di tipo struct, tenendo conto che il punto della notazione postfissa ha la precedenza sull'operatore di dereferencing *.

Esempio:

```
typedef struct{ int Campo_1,Campo_2; } TipoDato;
TipoDato        S, *P;
P = &S;
(*P).Campo1=75; /* assegnamento della costante 75 al Campo1 della
                   struct puntata da P* (e` necessario usare le
                   parentesi) */
```

Operatore ->:

L'operatore -> consente di accedere ad un campo di una struttura referenziata da un puntatore in modo più sintetico:

```
P->Campo1=75;
```