

PUNTATORI

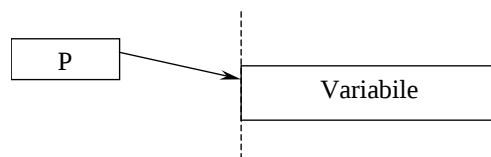
[1]

○ NUOVO COSTRUTTORE DI TIPO

- ↗ Un puntatore e' una variabile il cui contenuto e' l'indirizzo di memoria di un'altra variabile;
- ↗ Si parla quindi di puntatori a variabile;
- ↗ Ogni puntatore conserva implicitamente informazioni sul tipo di variabile a cui punta (compatibilita' ...).

○ DIVERSO MODO DI ACCEDERE AI DATI

- ↗ Accedo ai dati manipolando il contenuto delle variabili del programma attraverso i loro indirizzi di memoria.



PUNTATORI

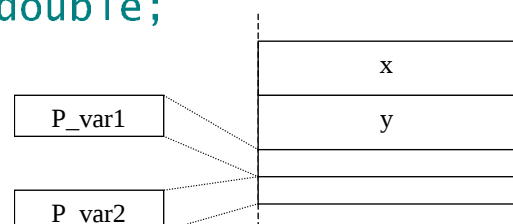
[2]

○ SINTASSI

- ↗ Dichiarazione di Tipo Puntatore
`typedef <Tipo_Var> *<Tipo_Punt>`
- ↗ Dichiarazione di Variabili Puntatore
`<Tipo_Var> *<Nome_Punt>`

○ ESEMPIO:

```
typedef double *Punt_var_double;  
double x, y, *p_var1;  
Punt_var_double p_var2;
```



PUNTATORI

[3]

○ OPERAZIONI SUI PUNTATORI

- ↗ Assegnamento attraverso **&** (ricava l'indirizzo);
- ↗ Assegnamento del valore di un altro puntatore;
- ↗ Assegnamento del valore **NULL**;
- ↗ Assegnamento di indirizzi dopo allocazione dinamica;
- ↗ Operazione di dereferenziazioni (operatore *****);
- ↗ Confronti: **==**, **!=**, **<**, **>**, ...;
- ↗ Operazioni aritmetiche.

○ OSSERVAZIONI

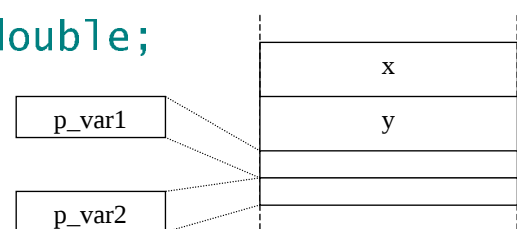
- ↗ I puntatori sono sostanzialmente riferimenti (etichette) per delle locazioni di memoria;
- ↗ Per accedere al contenuto devo usare *****;
- ↗ Ricavo l'indirizzo di memoria di una variabile con **&**.

PUNTATORI

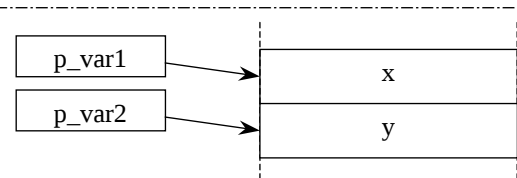
[4]

○ ESEMPIO:

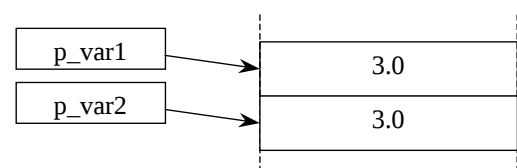
```
typedef double *Punt_var_double;  
double x, y, *p_var1;  
Punt_var_double p_var2;
```



```
p_var1 = &x;  
p_var2 = &y;
```



```
*p_var1 = 3.0;  
*p_var2 = x;
```

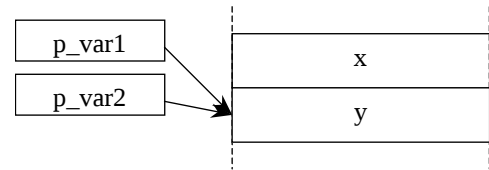


PUNTATORI

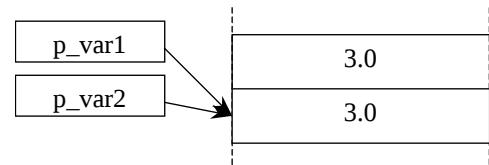
[5]

○ ASSEGNAZIONE TRA PUNTATORI

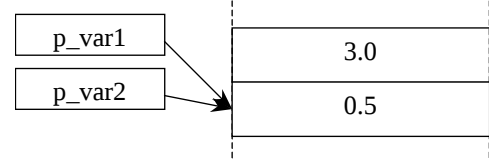
```
p_var1 = p_var2;  
// sposto la cella puntata
```



```
x=3.0;  
y=x; // assegno i contenuti
```



```
*p_var1 = 0.5;  
// assegno alla cella puntata
```



PUNTATORI

[6]

○ RISCHI NELL'USO DI PUNTATORI

↪ *Side-effect* problemi dovuti alla doppia identificazione di uno stesso oggetto ...

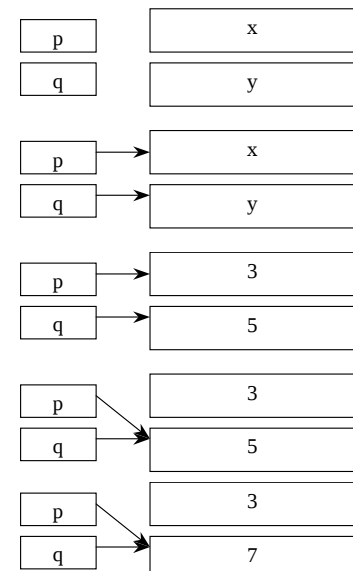
```
int x, y, *p, *q;
```

```
p=&x;  
q=&y;
```

```
*p=3; // x vale 3  
*q=5; // y vale 5
```

```
p=q // ora entrambe puntano a y
```

```
*q=7 // y vale 7, x vale 3, // ma *p vale 7
```



PUNTATORI

[7]

○ STRUTTURE DATI

- ↪ Base per la costruzione di strutture dati dinamiche
- ↪ Esistono notazioni equivalenti per l'accesso ai dati

$$\mathbf{P} \rightarrow \equiv (\mathbf{*P}).$$

○ ESEMPIO

```
typedef struct
    { int    x;
      ...    y;} TipoDato;
TipoDato T1,*p;
p=&T1;
```

```
T1.x=12;
(*p).x=12;
p->x=12;
```

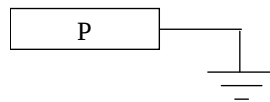
Equivalenti

PUNTATORI

[8]

○ PUNTATORI A NULL

- ↪ Un puntatore **P==NULL** e' tale per cui ***P** non e' indefinito



○ UTILITA' DEI PUNTATORI

- ↪ Legame tra **array** e puntatori (aritmetica dei puntatori)
- ↪ Passaggio parametri nelle funzioni (passaggio per indirizzo)
- ↪ Allocazione dinamica della memoria (e riferimento ...)
- ↪ Definizione di strutture dati dinamiche

ARRAY E PUNTATORI

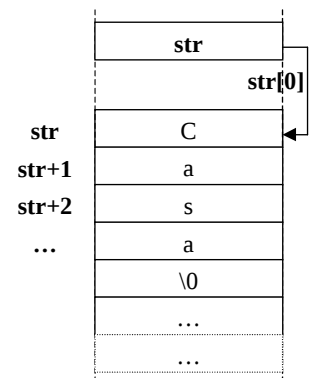
[1]

○ ALLOCAZIONE DI UN ARRAY

- ↪ Gli elementi occupano in memoria un numero di celle che dipende dal tipo del contenuto;
- ↪ Gli elementi sono allocati in celle di memoria contigue.

○ ACCESSO AGLI ARRAY ED ARITMETICA DEI PUNTATORI

- ↪ Puntatore costante alla prima cella di memoria che contiene il primo elemento dell'array
`char str[5] = "Casa";`
- ↪ Se P e' un puntatore ad una variabile di un determinato tipo l'espressione P+1 fornisce l'accesso alla successiva



ARRAY E PUNTATORI

[2]

○ ACCESSO AGLI ELEMENTI DI UN ARRAY

- ↪ Se P punta al primo elemento di un array P+n punta all'elemento n posizioni successive (o precedenti)
- ↪ Si ottengono quindi alcune notazioni equivalenti

`a[0] = *a`

`a[i] = *(a+i)`

`a = &a[0]`

`a+i = &a[i]`

○ ASSEGNAZIONE TRA ARRAY

- ↪ Non e' possibile assegnare due array poiché si assegnerebbe al primo l'indirizzo in memoria del secondo: puntano alla stessa variabile (non sono due copie) le celle di memoria del primo non sono piu' utilizzabili !!!!

FUNZIONE E PUNTATORI

[1]

○ PASSAGGIO PER VALORE (UNICO IN C)

- ⇒ Copio i parametri attuali nei parametri formali
- ⇒ I parametri attuali non vengono modificati, comunico attraverso variabili globali (tipico nelle procedure) o attraverso l'istruzione return.

○ PASSAGGIO PER INDIRIZZO

- ⇒ Modifico il contenuto dei parametri attuali della funzione;
- ⇒ Viene realizzato in C passando il puntatore alle celle di memoria contenenti i parametri formali e modificandone il contenuto accedendo tramite esso.

FUNZIONE E PUNTATORI

[2]

○ ESEMPIO: SCANF

- ⇒ Si vuole modificare il contenuto della variabile passata scrivendoci all'**interno** cio' che viene immesso da tastiera
- ⇒ Alla funzione scanf viene quindi passato l'indirizzo di memoria della variabile.

```
int Num;  
scanf("%d", &Num);
```

○ PASSAGGIO DI ARRAY

- ⇒ Se si vuole leggere una stringa e metterla all'interno di un array, si deve quindi passare il riferimento a tale struttura dati

```
char ArrayDiChar[100];  
scanf("%s", ArrayDiChar);
```

FUNZIONE E PUNTATORI

[3]

○ ESEMPIO:

- Definita la struttura dati per un generico parallelepipedo, definto da tre lati ed un volume, si vuole scrivere una funzione che dato un parallelepipedo ne calcoli il volume e lo memorizzi nell'apposito campo della struttura.
- Come definisco le funzioni e cosa devo passargli ??

```
typedef struct
{
    double l1;
    double l2;
    double l3;
    double volume;
} Parallelepipedo;
```