I Puntatori

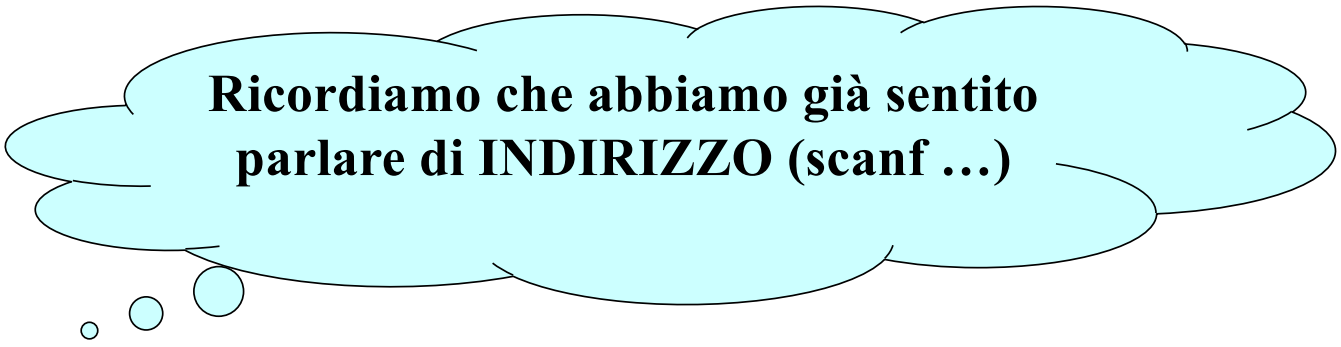
Programmazione I

Prof. Elisa Quintarelli

Variabili rivisitate

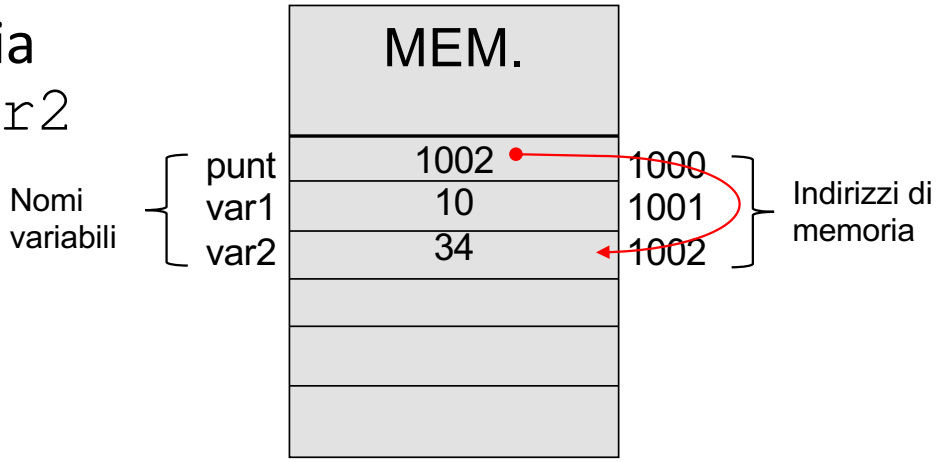
- Finora una variabile (cella di memoria) era accessibile mediante un nome simbolico
- `x = a`

  il valore della cella identificata da `a` va copiato nella cella identificata da `x`



Puntatori

- Il puntatore è una variabile che contiene un indirizzo di memoria
- Il puntatore serve per puntare ad un'altra variabile e quindi accedervi in modo indiretto
- Nell'esempio il puntatore `punt` contiene l'indirizzo di memoria di `var2` quindi «punta» a `var2`



Perchè introdurre i puntatori?

- Per poter gestire le situazioni in cui dobbiamo restituire più valori con una funzione (vedremo a breve)
- Per gestire problemi in cui non si sa stimare a priori la cardinalità dei dati (argomento principale di questa seconda parte del corso).

Puntatori

- Dichiarazione di un puntatore
```
int *punt;
```
  - `*` alla sinistra del nome della variabile indica che si tratta di un puntatore
  - È necessario specificare il tipo di dato a cui si punta

- Assegnamento di un indirizzo ad un puntatore
```
int var1, var2;
punt=&var2;
```
  - L'operatore `&` calcola l'indirizzo di una variabile dato il suo nome

- Accesso al contenuto della cella puntata (dereferenziazione)
```
*punt=2;
printf("%d\n", *punt);
```
  - L'operatore * permette di accedere al contenuto della variabile puntata

Puntatori: esempio

```
#include<stdio.h>

void main(){
  int *punt, var1, var2;

  punt=&var2;
  scanf("%d", punt);
  *punt = *punt +1;
  printf("%d\n", *punt);

}
```

Dichiarazione di una variabile puntatore

Assegnamento dell'indirizzo di `var2` a `punt`

Lettura del valore da salvare in `var2`. non c'è bisogno di specificare `&` perché `punt` contiene già l'indirizzo di `var2 (non usate questa modalità)`

Accesso alla variabile puntata mediante l'operatore di dereferenziazione

Definizione di un puntatore

```
tipo *nomevarPuntatore; int *p;
typedef tipo *tipoRef;
```

`tipoRef` è un tipo puntatore (indirizzo, riferimento) a un dato di tipo **`tipo`**

```
typedef int *intRef; intRef myRef, yourRef;
```

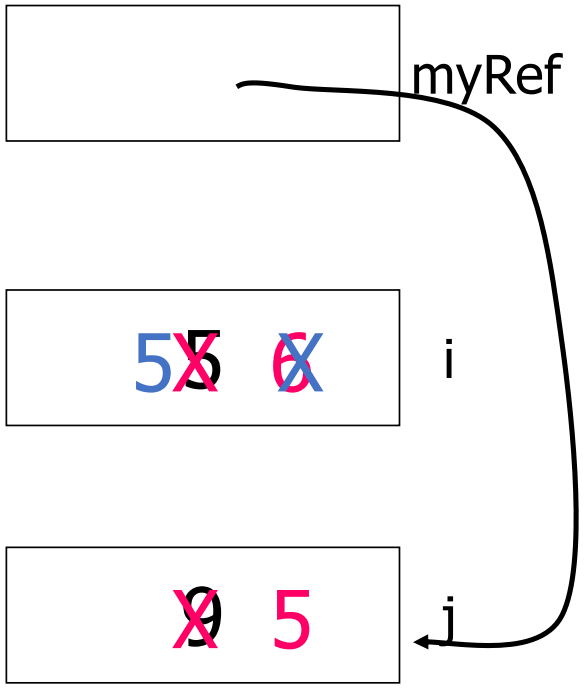
Suggerimento: usare “Ref” (o “Punt”) in coda al nome **per denotare un puntatore**

Esempi

```
typedef int *intRef;

intRef myRef;
int i=5; int j=9;


myRef = &j;
```




```
*myRef = i;
i++;
 i = *myRef;
```

Puntatori: riassumiamo

- Consideriamo un intero i
- Chiamiamo &i il suo indirizzo.

```
int *puntint = &i;
```

- `int *` è la notazione per dichiarare un puntatore a un intero.
- & è l'operatore che applicato **a una variabile** ritorna l'indirizzo della variabile stessa (operatore di referencing). Perciò prima ho *dichiarato* e *assegnato*.
- L'operatore opposto * restituisce il valore puntato (operatore di dereferencing):

```
i = *puntint;
```

- Attenzione!!! Non confondiamo i molteplici usi dell'asterisco (moltiplicazione, dichiarazione di puntatore, dereferencing)

Esercizio

- Supponiamo che **`p`** e **`q`** siano puntatori a **`int`**
- Che differenza esiste tra

```
p = q;
```

e

```
*p = *q;
```

???

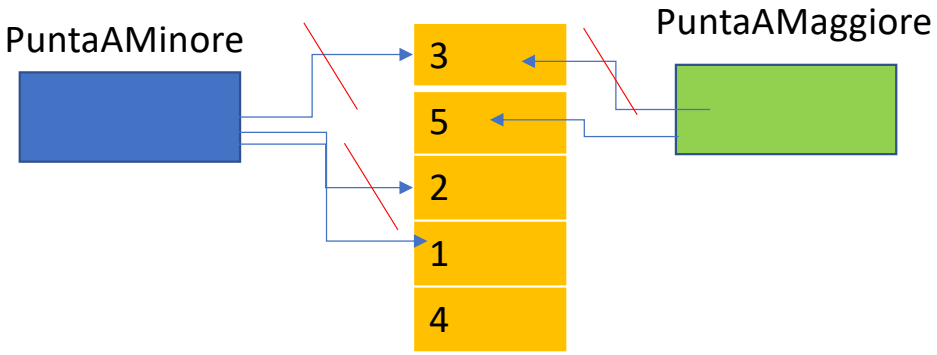
- nel primo caso impongo che il puntatore p punti alla stessa variabile cui punta q
- nel secondo caso assegno il valore della variabile puntata da q alla variabile puntata da p

Valore iniziale

- Il valore iniziale di un puntatore **dovrebbe essere NULL**
- NULL significa che **non riferisce ad alcuna cella di memoria**
- dereferenziando NULL si ha un **errore in esecuzione**
- *come al solito, non fare MAI affidamento sulle inizializzazioni implicite delle variabili che C potrebbe fare (alcune implementazioni inizializzano a NULL)*

Esempio: assegna a due puntatori l'indirizzo delle componenti con valore minimo e massimo

```
 /* Programma Puntatori */
#define LunghezzaArray 5
void main() {
     int       i;
     int       ArrayDiInt[LunghezzaArray];
     int       *PuntaAMinore;
     int       *PuntaAMaggiore;
     . . . .
     PuntaAMinore = &ArrayDiInt[0];
     i = 1;
     while (i < LunghezzaArray)
     {  if (ArrayDiInt[i] < *PuntaAMinore)
           PuntaAMinore = &ArrayDiInt[i];
        i = i + 1;
     }
     PuntaAMaggiore = &ArrayDiInt[0];
     i = 1;
     while (i < LunghezzaArray) {
        if (ArrayDiInt[i] > *PuntaAMaggiore)
           PuntaAMaggiore = &ArrayDiInt[i];
        i = i + 1;
     }
}
```



Puntatori e tipizzazione delle variabili puntate

Il compilatore segnala utilizzo di puntatori a dati di tipo differente da quello a cui dovrebbero puntare

Esercizio Puntatori

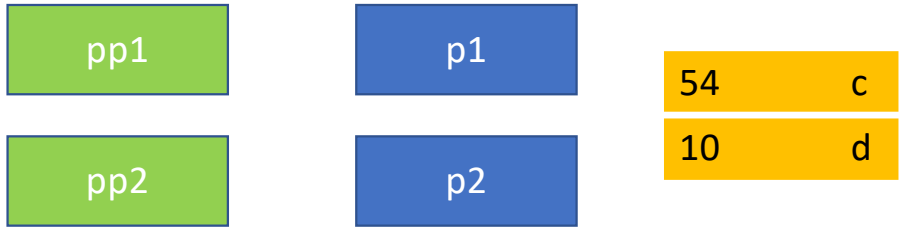
Dato il seguente codice, indicare a fianco ad ogni printf il risultato stampato.

```
typedef int *intptr;
typedef intptr *int2ptr;

void main()
{     int c, d;
      intptr p1, p2;
      int2ptr pp1, pp2;

      c = 54; d = 10;

      p1 = &c; p2 = p1;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54  10  54
54
      p1 = &d;
      *p1 = *p1 + *p2;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54  64  64
54
      pp1 = &p1;
      pp2 = &p2;
      *pp2 = *pp1;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54  64  64
64
}
```



Esercizio Puntatori (guardare le slide seguenti)

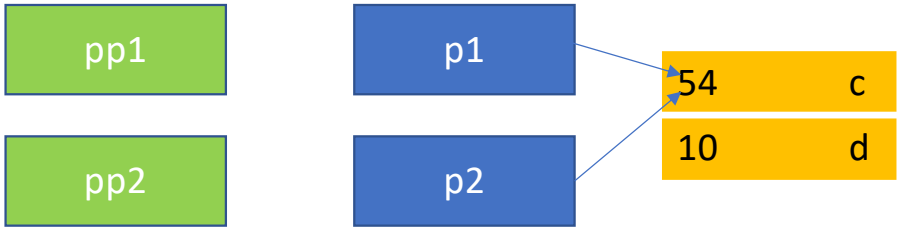
Dato il seguente codice, indicare a fianco ad ogni printf il risultato stampato.

```
typedef int *intptr;
typedef intptr *int2ptr;

void main()
{     int c, d;
      intptr p1, p2;
      int2ptr pp1, pp2;

      c = 54; d = 10;

      p1 = &c; p2 = p1;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   10   54
54
      p1 = &d;
      *p1 = *p1 + *p2;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   64   64
54
      pp1 = &p1;
      pp2 = &p2;
      *pp2 = *pp1;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   64   64
64
}
```



Esercizio Puntatori (guardare le slide seguenti)

Dato il seguente codice, indicare a fianco ad ogni printf il risultato stampato.

```
typedef int *intptr;
typedef intptr *int2ptr;

void main()
{     int c, d;
      intptr p1, p2;
      int2ptr pp1, pp2;

      c = 54; d = 10;

      p1 = &c; p2 = p1;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   10   54
54
      p1 = &d;
      *p1 = *p1 + *p2;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   64   64
54
      pp1 = &p1;
      pp2 = &p2;
      *pp2 = *pp1;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   64   64
64
}
```

Esercizio Puntatori (guardare le slide seguenti)

Dato il seguente codice, indicare a fianco ad ogni printf il risultato stampato.

```
typedef int *intptr;
typedef intptr *int2ptr;

void main()
{     int c, d;
      intptr p1, p2;
      int2ptr pp1, pp2;

      c = 54; d = 10;

      p1 = &c; p2 = p1;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   10   54
54
      p1 = &d;
      *p1 = *p1 + *p2;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   64   64
54
      pp1 = &p1;
      pp2 = &p2;
      *pp2 = *pp1;
      printf ("%d %d %d %d", c, d, *p1, *p2);   54   64   64
64
}
```

Array e puntatori

- In C stretta parentela tra array e puntatori
- il nome di variabile di tipo array (es. **`a`**) è una *costante* di tipo puntatore (al tipo componente l'array) con valore "indirizzo della prima parola di memoria allocata all'array"
- **`int a[3]`** definisce **`a`** come **`int *const`**

  (cioè un **puntatore costante** a una variabile intera)

  `a=&a[0]`
- perciò **`a[i]`** è equivalente a **`*(a+i)`** !!!

Riassumiamo: array e puntatori

- In presenza di
  `int a[5]; int i; int * p;`
- **`a[i]`** equivale a **`*(a+i)`** (i+1-esimo elemento)
- **`p = a`** è equivalente a **`p = &a[0];`**
- **`p = a+1`** è equivalente a **`p = &a[1];`**
- **`a = p;`** ERRORE !!
- **`a = a+1;`** ERRORE !!

(cioè ricordiamoci che comunque `a` era un array!!)

Puntatori e array

- Il nome di un array (senza parentesi quadre) rappresenta un puntatore costante alla base dell'array
- Si può usare un puntatore e l'aritmetica dei puntatori per accedere ad un array

```
...
int *punt, arr[10];
...
punt=arr;
*punt=9;
punt=punt+2;
*punt=7;
*(punt+1)=17;
...
```

Assegnamento dell'indirizzo base (cioè della prima cella) dell'array al puntatore

Scrittura nella prima cella arr[0] dell'array del valore 9

Il puntatore viene spostato sulla terza cella (punta a[2])

Scrittura nella terza cella dell'array del valore 7

Scrittura nella quarta cella dell'array del valore 17

Puntatori e struct

- Possiamo usare i puntatori anche per puntare a variabili di tipo struttura

```
typedef struct{
  int x,y;
}punto_t;

void main()
  punto_t *p, valore;
  ...
  p=&valore;
  p->x = 1;
  ...
```

L'operatore -> su un puntatore a struttura accede ad un campo della struttura puntata. Codice equivalente a:
`(*p).x=1;`