Puntatori in C

Puntatori

Variabili tradizionali

Esempio: `int a = 5;`

- Proprietà della variabile a:
  - nome: a
  - tipo: int
  - valore: 5
  - indirizzo: A010

indirizzo	memoria
A00E	...
A010	**5**
A012	…
A014	…

Finora abbiamo usato solo le prime tre proprietà

Come si usa l'indirizzo?

&a → **operatore indirizzo** "&" applicato alla variabile a → ha valore 0xA010 (ovvero, 61456 in decimale)

Puntatori

- Gli indirizzi si utilizzano nelle variabili di tipo puntatore, dette anche **puntatori**.

Variabili di tipo puntatore

Esempio: `int *pi;`

- Proprietà della variabile pi:
  - nome: pi
  - tipo: **puntatore ad intero** (ovvero, indirizzo di un intero)
  - valore: inizialmente casuale
  - indirizzo: fissato una volta per tutte

Puntatori

Sintassi della dichiarazione di variabili puntatore

```
<tipo> *<identificatore>;
```

Esempio:

```
int *pi1;
int *pi2, i, *pi3, j;
float *pf1, f, *pf2;
```

NOTA:

```
int *pi;
int * pi;
int* pi;
```

Sono tutte scritture equivalenti!

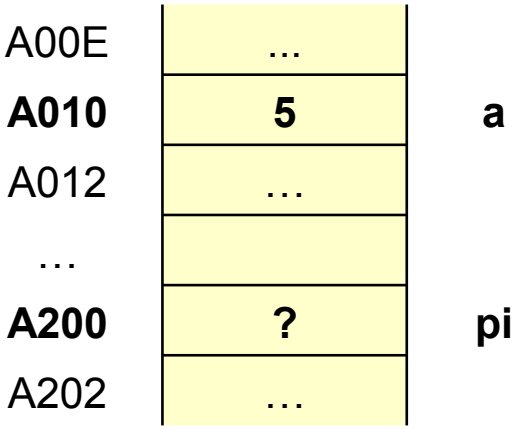
- pi1, pi2, pi3 sono di tipo puntatore ad int
- i, j sono di tipo int
- pf1, pf2 sono di tipo puntatore a float
- f è di tipo float

Puntatori

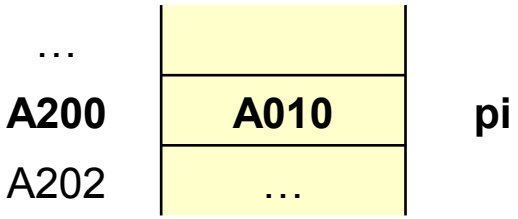
- Una variabile puntatore può essere inizializzata usando l'operatore di indirizzo.

Esempio: **`pi = &a;`**

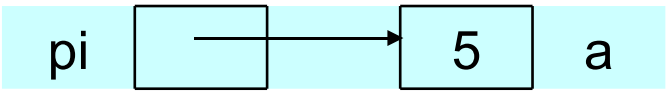
- il valore di pi viene posto pari all'indirizzo di a
- ovvero, pi **punta** ad a
- ovvero, a è l'**oggetto puntato** da pi



dopo l'assegnazione…



graficamente lo indichiamo così



Puntatori

Operatore di dereferenziamento "*"

- Applicato ad una variabile puntatore fa riferimento all'oggetto puntato.

Esempio:

```
int *pi;       /* dichiarazione di un puntatore ad intero */
int a = 5, b;        /* dichiarazione di variabili intere */
pi = &a;      /* pi punta ad a
                 ==> *pi e' un altro modo di denotare a */
b = *pi;    /* assegna a b il valore della variabile puntata
               da pi, ovvero il valore di a, ovvero 5 */
*pi = 9;      /* assegna 9 alla variabile puntata da pi,
                 ovvero ad a */
```

- N.B. Se **`pi`** è di tipo **`int *`**, allora **`*pi`** è di tipo **`int`**.

Puntatori

ATTENZIONE:

- Non bisogna confondere le due occorrenze di "*":

  - * in una dichiarazione serve per dichiarare una variabile di tipo puntatore

  Es.: int *pi;

  - * in una espressione è l'operatore di dereferenziamento

  Es.: b = *pi;

Operatori di dereferenziamento * e di indirizzo &

Hanno priorità più elevata degli operatori binari

- **"*" è associativo a destra**
  - **p è equivalente a *(*p)
- **"&" può essere applicato solo ad una variabile**
  - &a non è una variabile → "&" **non è associativo**

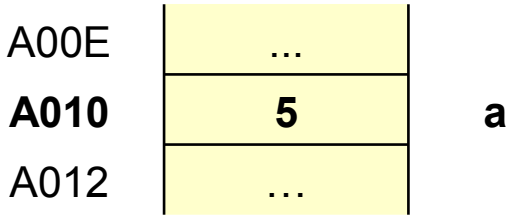
- **"*" e "&" sono uno l'inverso dell'altro**
  - data la dichiarazione int a;
    *&a è un alias per a (*sono entrambi variabili*)
  - data la dichiarazione int *pi;
    &*pi ha valore uguale a pi

  però:
  - pi è una variabile
  - &*pi non lo è (ad esempio, non può essere usato a sinistra di "=")

Puntatori

Stampa di puntatori

- I puntatori si possono stampare con printf e specificatore di formato **"%p"** (stampa in formato esadecimale).
- Poco utile in applicazioni reali!



Esempio:

```
int a = 5;
int *pi;
pi = &a;
printf("indirizzo di a = %p\n", &a);     /* stampa 0xA010 */
printf("valore di pi = %p\n", pi);       /* stampa 0xA010 */
printf("valore di &*pi = %p\n", &*pi);   /* stampa 0xA010 */
printf("valore di a = %d\n", a);         /* stampa 5 */
printf("valore di *pi = %d\n", *pi);     /* stampa 5 */
printf("valore di *&a = %d\n", *&a);     /* stampa 5 */
```

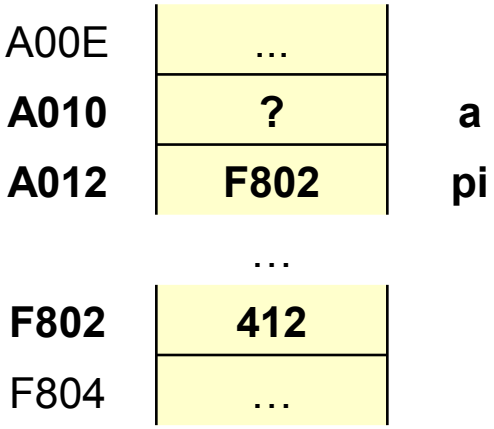
Inizializzazione di puntatori

- I puntatori (come tutte le altre variabili) devono venire inizializzati prima di poter essere usati.
- È un errore dereferenziare una variabile puntatore non inizializzata.

Esempio:

```
int a;
int *pi;

a = *pi;
*pi = 500;
```



ad a viene assegnato il valore 412

Scrive 500 nella cella memoria di indirizzo F802
… ma non sappiamo a cosa corrisponde questa cella di memoria!
… la memoria puo essere corrotta!
… **ERRORE** a run-time!

Tipo di variabili puntatore

- Il tipo di una variabile puntatore è **puntatore a *tipo***.
- Il suo valore è un **indirizzo**.
- I tipi puntatore sono **indirizzi** e **non interi**.

Esempio:
```
int a;
int *pi;
a = pi;
```
compilando si ottiene un warning:
"assignment makes integer from pointer without a cast"

- Due variabili puntatore a tipi diversi **non sono compatibili** tra loro.

Esempio:
```
int x;
int *pi;
float *pf;
```
`x = pi;` assegnazione `int*` a `int`
→ warning: "assignment makes integer from pointer without a cast"

`pf = x;` assegnazione `int` a `float*`
→ warning: "assignment makes pointer from integer without a cast"

`pi = pf;` assegnazione `float*` a `int*`
→ warning: "assignment from incompatible pointer type"

Tipo di variabili puntatore

Perchè il C distingue tra puntatori di tipo diverso?

- Se tutti i tipi puntatore fossero identici (ad es. puntatore a void), non sarebbe possibile determinare a tempo di compilazione il tipo di *p.

Esempio: void *p;
int i; char c; float f;

Potrei scrivere: p = &c;
p = &i;
p = &f;

- Il tipo di ***p** verrebbe a dipendere dall'ultima assegnazione che è stata fatta!!!
- Qual è il significato di **i/*p** (divisione intera oppure divisione reale)?
- Il C permette di definire un puntatore a void (tipo void*)
  - è compatibile con tutti i tipi puntatore
  - **non** può essere dereferenziato (bisogna prima fare un cast esplicito)

Operatore *sizeof* con puntatori

- La funzione sizeof restituisce l'occupazione in memoria in byte di una variabile.
- Può anche essere applicata anche ad un tipo.
- Tutti i puntatori sono indirizzi *…quindi…* occupano lo spazio di memoria di un indirizzo.
- L'oggetto puntato ha dimensione del tipo puntato.

Esempio:

```
char *pc;
int *pi;
double *pd;
printf("%d %d %d ", sizeof(pc), sizeof(pi), sizeof(pd));
printf("%d %d %d\n", sizeof(char *),sizeof(int *),sizeof(double *));

printf("%d %d %d ", sizeof(*pc), sizeof(*pi), sizeof(*pd));
printf("%d %d %d\n", sizeof(char), sizeof(int), sizeof(double));
```

Ecco cosa stampa questo codice

```
4 4 4  4 4 4
1 2 8  1 2 8
```

Aritmetica dei puntatori

Sui puntatori si possono effettuare diverse **operazioni**:

- di **dereferenziamento**

Esempio:
```
int *p, i;
i = *p;
```

- di **assegnamento**

Esempio:
```
int *p, *q;
p = q;
```

N.B. p e q devono essere dello stesso tipo (altrimenti bisogna usare l'operatore di cast).

Aritmetica dei puntatori

- di **confronto**

Esempio: if (p == q) …

Esempio: if (p > q) … **Ha senso?**

Con quello che abbiamo visto finora no.

Vedremo tra poco che ci sono situazioni in cui ha senso.

- **aritmetiche**, con opportune limitazioni
  - incremento (++) o decremento (--)
  - somma (+=) o sottrazione (-=) di un intero
  - sottrazione di un puntatore da un altro

Significato delle operazioni aritmetiche sui puntatori

- Il **numero di byte** di cui viene modificato il puntatore **dipende dal suo tipo**.

Esempio:

```
int *pi;
*pi = 15;
pi++;
*pi = 20;
```

→ `pi` punta al prossimo `int` (4 byte dopo)

Esempio:

```
double *pd;
*pd = 12.2;
pd += 3;
```

→ `pd` punta a 3 `double` dopo (24 byte dopo)

Esempio:

```
char *pc;
*pc = 'A';
pc -= 5;
```

→ `pc` punta a 5 `char` prima (5 byte prima)

Si può anche scrivere:

```
pi = pi + 1;
pd = pd + 3;
pc = pc - 5;
```

Relazione tra vettori e puntatori

- **Attenzione:** in generale non sappiamo cosa contengono le celle di memoria adiacenti ad una data cella.

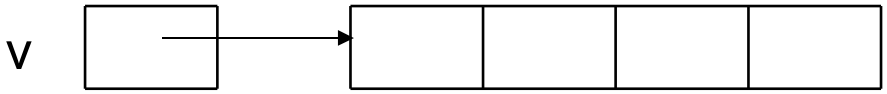
- L'unico caso in cui sappiamo quali sono le locazioni di memoria successive e cosa contengono è quando utilizziamo dei vettori.

Relazione tra vettori e puntatori

Abbiamo detto che …

- Un vettore è un insieme finito di **N** variabili dello stesso tipo, ognuna identificata da un indice intero compreso fra **0** e **N-1**

- In C un array è in realtà **un puntatore che punta a un'area di memoria pre-allocata,** di dimensione prefissata



Relazione tra vettori e puntatori

- Pertanto, in C, il **nome di un vettore** è in realtà un **sinonimo per il suo indirizzo iniziale** (elemento di indice 0)

```
int vet[10];
```

`vet` e `&vet[0]` **hanno lo stesso valore**.

```
printf("%p %p", vet, &vet[0]);
```
stampa 2 volte lo stesso indirizzo.

Vettori: implementazione in C

- Il fatto che il nome dell'array non indichi l'array, ma l'indirizzo iniziale dell'area di memoria ad esso associata ha una conseguenza:

È impossibile denotare un array nella sua globalità, in qualunque contesto

- Questa è la spiegazione, ad esempio, del motivo per cui non si può assegnare un array ad un altro
- *Vi sono altre conseguenze che vedremo più avanti…*

Accesso agli elementi di un vettore

Esempio:

```
int vet[5];
int *pi = vet;
*(pi + 3) = 28;
```

`pi+3` punta all'elemento di indice 3 del vettore.

- 3 viene detto **offset** (o scostamento) del puntatore.
- N.B. Servono le "()" perchè "*" ha priorità maggiore di "+".
  Che cosa denota `*pi + 3` ?

Osservazione:

- `&vet[3]` equivale a `pi+3` equivale a `vet+3`
- `*&vet[3]` equivale a `*(pi+3)` equivale a `*(vet+3)`

- Inoltre, `*&vet[3]` equivale a `vet[3]`.
- Quindi in C, `vet[3]` è semplicemente un modo alternativo di scrivere `*(vet+3)`.

Accesso ai campi di una struttura (struct) dato il puntatore

- Data una variabile di tipo struct, sappiamo che per accedere ai suoi campi si usa la notazione puntata
- Se ho un puntatore ad una struct, per accedere ai suoi campi posso usare l'operatore **->** (*freccia*)
  (anziché dereferenziare il puntatore e poi usare la notazione puntata)

```
struct data {
   int giorno, mese, anno;
}
struct data d;
…
d.giorno = 25;
d.mese = 12;
d.anno = 2003;
```

```
struct data {
   int giorno, mese, anno;
}
struct data d, *p;
…
p->giorno = 15;
p->mese = 8;
p->anno = 2005;
```