Programmazione 1

Lezione 6

Vincenzo Marra

vincenzo.marra@unimi.it

Dipartimento di Matematica *Federigo Enriques*
Università degli Studi di Milano

5 aprile 2016

Calendario

- Lezioni frontali. La lezione di mercoledì 19 aprile è annullata per dare la possibilità agli studenti di concentrarsi sulle prove intermedie che si terranno nell'interruzione didattica 20-28 aprile.
- Laboratori. I due turni di laboratorio di mercoledì 12 aprile si terranno regolarmente. I due turni di laboratorio di mercoledì 19 aprile si terranno, ma saranno *riservati agli studenti dei due turni del giovedì*, che si potranno suddividere liberamente fra l'Aula 2 e l'Aula 307 di via Celoria. Gli studenti che decideranno di fare laboratorio in Aula 2 devono ricordarsi di ritirare le credenziali d'accesso dal sig. Luca Galizzi (piano -1), se non l'hanno ancora fatto. Da mercoledì 3 maggio tutte le lezioni riprendono regolarmente.

I puntatori: Introduzione

I puntatori nel linguaggio C sono variabili che permettono di fare riferimento al contenuto di altre variabili, o più generalmente al contenuto di aree della memoria centrale.

I puntatori: Introduzione

I puntatori nel linguaggio C sono variabili che permettono di fare riferimento al contenuto di altre variabili, o più generalmente al contenuto di aree della memoria centrale.

Il loro uso nella programmazione C è importante: alcune cose si possono realizzare solo tramite puntatori; altre, sebbene siano realizzabili altrimenti, possono essere implementate in modo più efficiente e compatto per mezzo dei puntatori.

I puntatori: Introduzione

I puntatori nel linguaggio C sono variabili che permettono di fare riferimento al contenuto di altre variabili, o più generalmente al contenuto di aree della memoria centrale.

Il loro uso nella programmazione C è importante: alcune cose si possono realizzare solo tramite puntatori; altre, sebbene siano realizzabili altrimenti, possono essere implementate in modo più efficiente e compatto per mezzo dei puntatori.

Si ricordi che la memoria centrale del calcolatore è organizzata come una successione numerata di celle o locazioni di memoria, ciascuna atta a memorizzare una quantità di informazione prefissata e dipendente dall'architettura della macchina — per esempio, un byte = 8 bit.

I puntatori: Introduzione

I puntatori nel linguaggio C sono variabili che permettono di fare riferimento al contenuto di altre variabili, o più generalmente al contenuto di aree della memoria centrale.

Il loro uso nella programmazione C è importante: alcune cose si possono realizzare solo tramite puntatori; altre, sebbene siano realizzabili altrimenti, possono essere implementate in modo più efficiente e compatto per mezzo dei puntatori.

Si ricordi che la memoria centrale del calcolatore è organizzata come una successione numerata di celle o locazioni di memoria, ciascuna atta a memorizzare una quantità di informazione prefissata e dipendente dall'architettura della macchina — per esempio, un byte = 8 bit.

La posizione di una data cella nella memoria è il suo indirizzo.

Si ricordi inoltre che una variabile è un nome simbolico per uno specifico indirizzo di memoria.

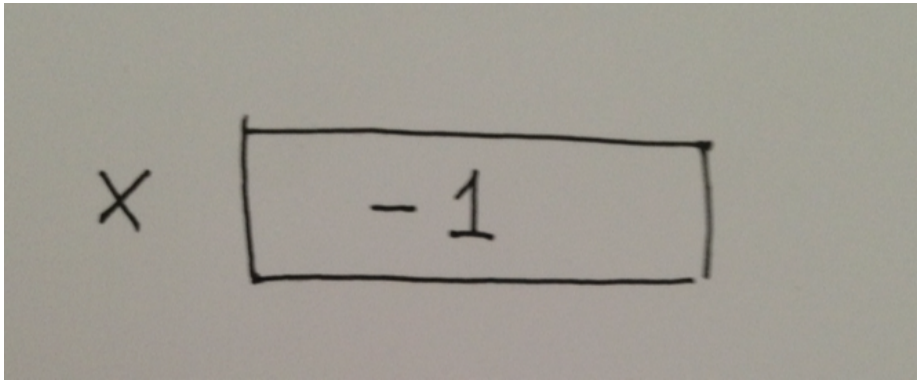
Si ricordi inoltre che una variabile è un nome simbolico per uno specifico indirizzo di memoria.

Il contenuto della variabile è il dato contenuto nella cella di memoria di cui la variabile è un nome.

Si ricordi inoltre che una variabile è un nome simbolico per uno specifico indirizzo di memoria.

Il contenuto della variabile è il dato contenuto nella cella di memoria di cui la variabile è un nome.

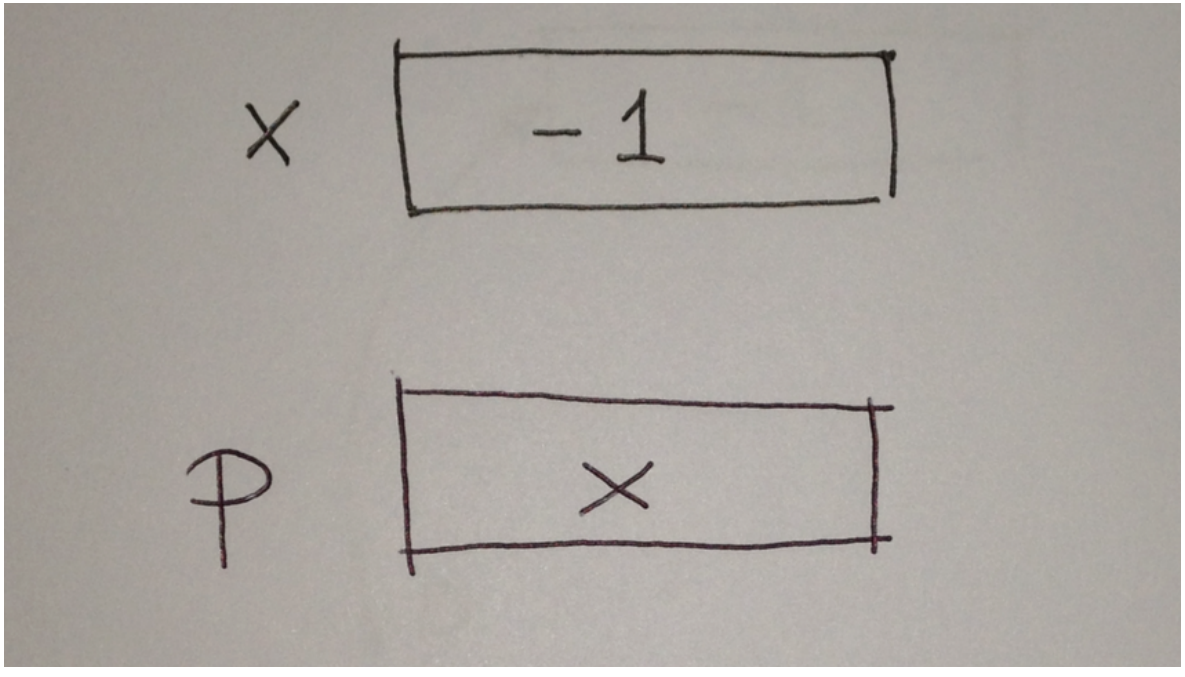
La definizione `int x=-1`, per esempio, genera questa situazione in memoria:



Un puntatore è una variabile il cui contenuto è l'indirizzo di una cella di memoria.

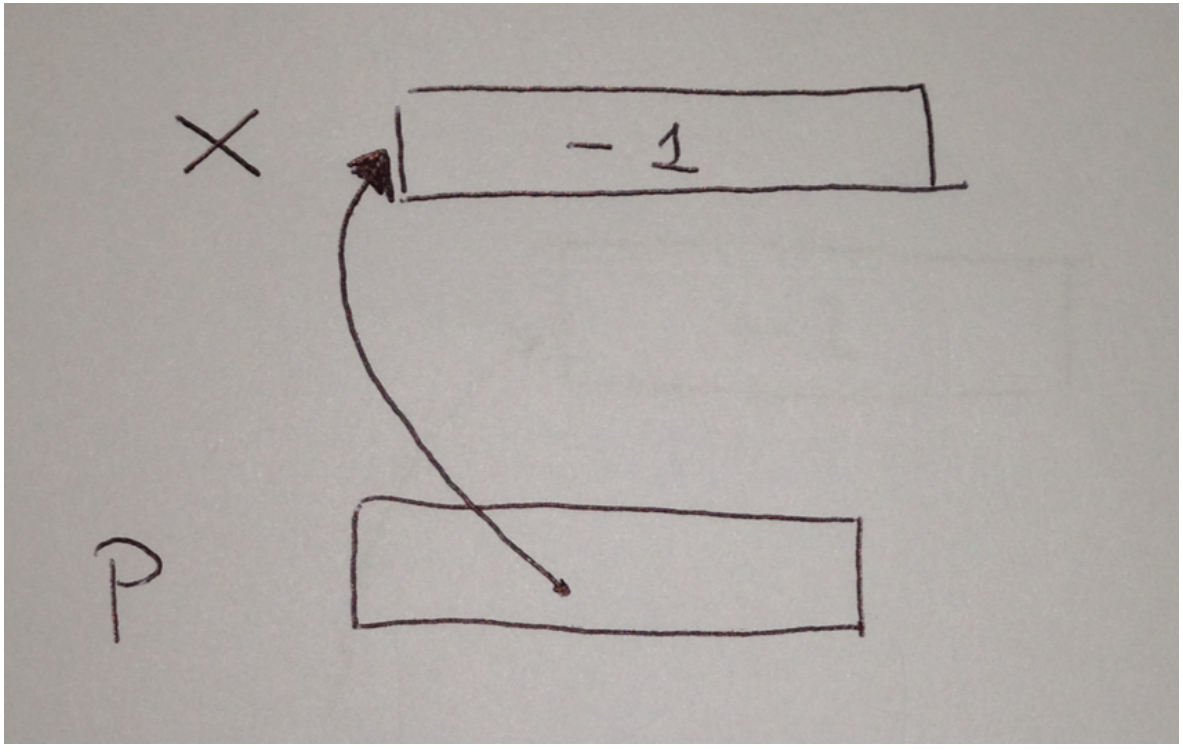
Un puntatore è una variabile il cui contenuto è l'indirizzo di una cella di memoria.

Ad esempio, un puntatore di nome p il cui contenuto sia l'indirizzo della cella di memoria denominata dalla variabile x si può raffigurare così:



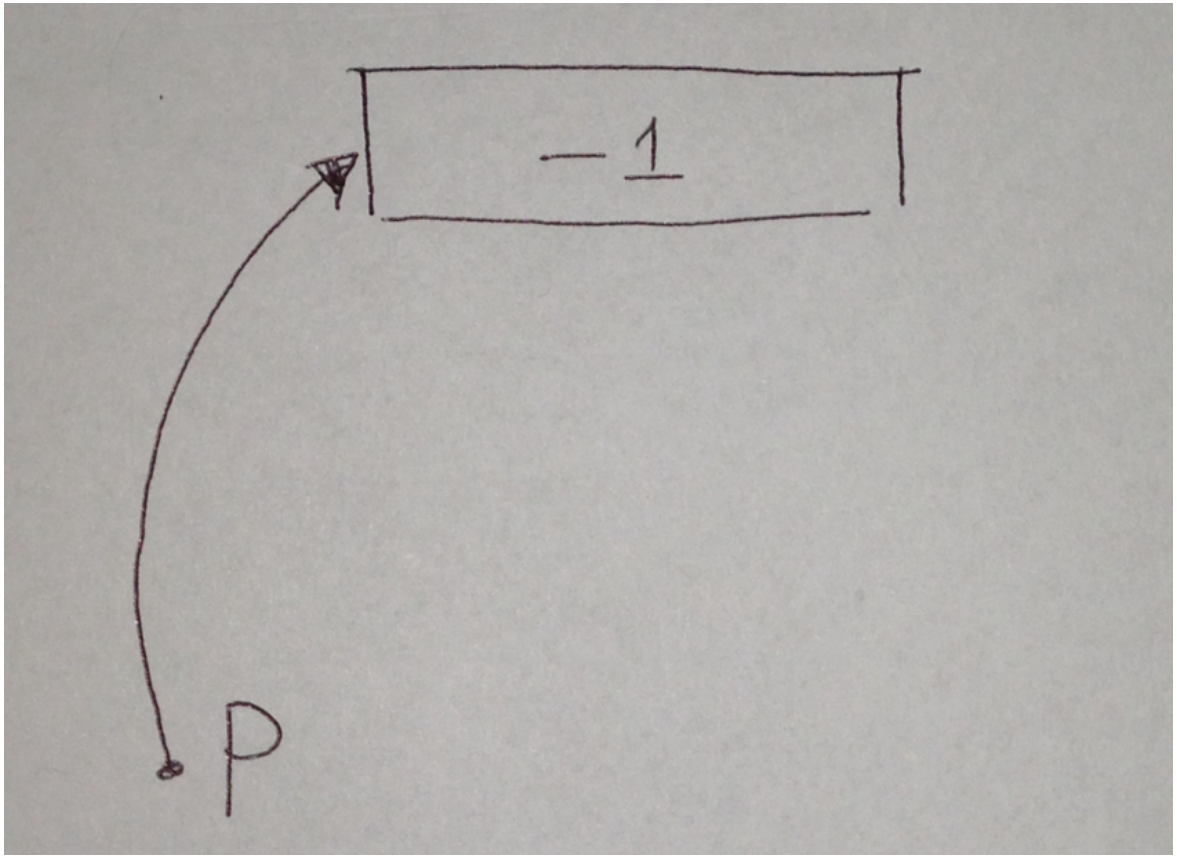
Un puntatore è una variabile il cui contenuto è l'indirizzo di una cella di memoria.

È tuttavia più utile raffigurarlo così:



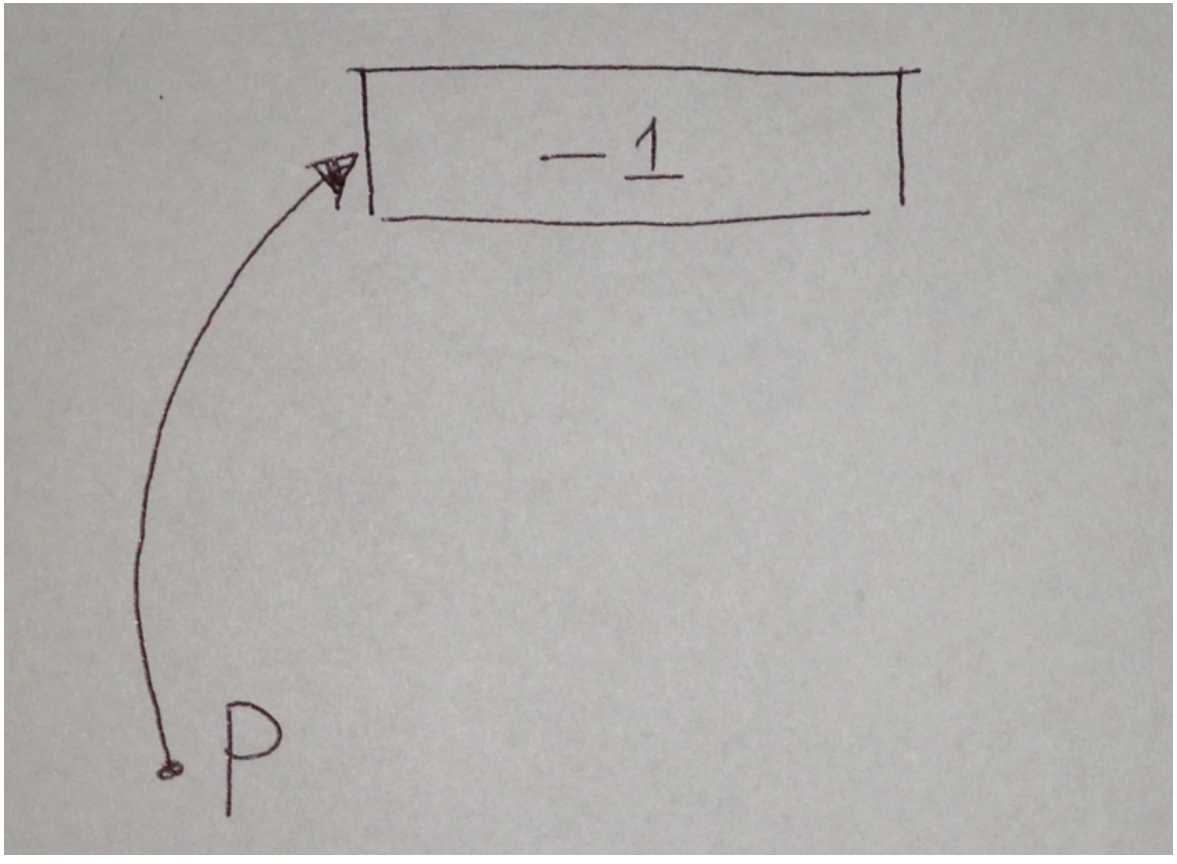
Un puntatore è una variabile il cui contenuto è l'indirizzo di una cella di memoria.

E anzi addirittura così:



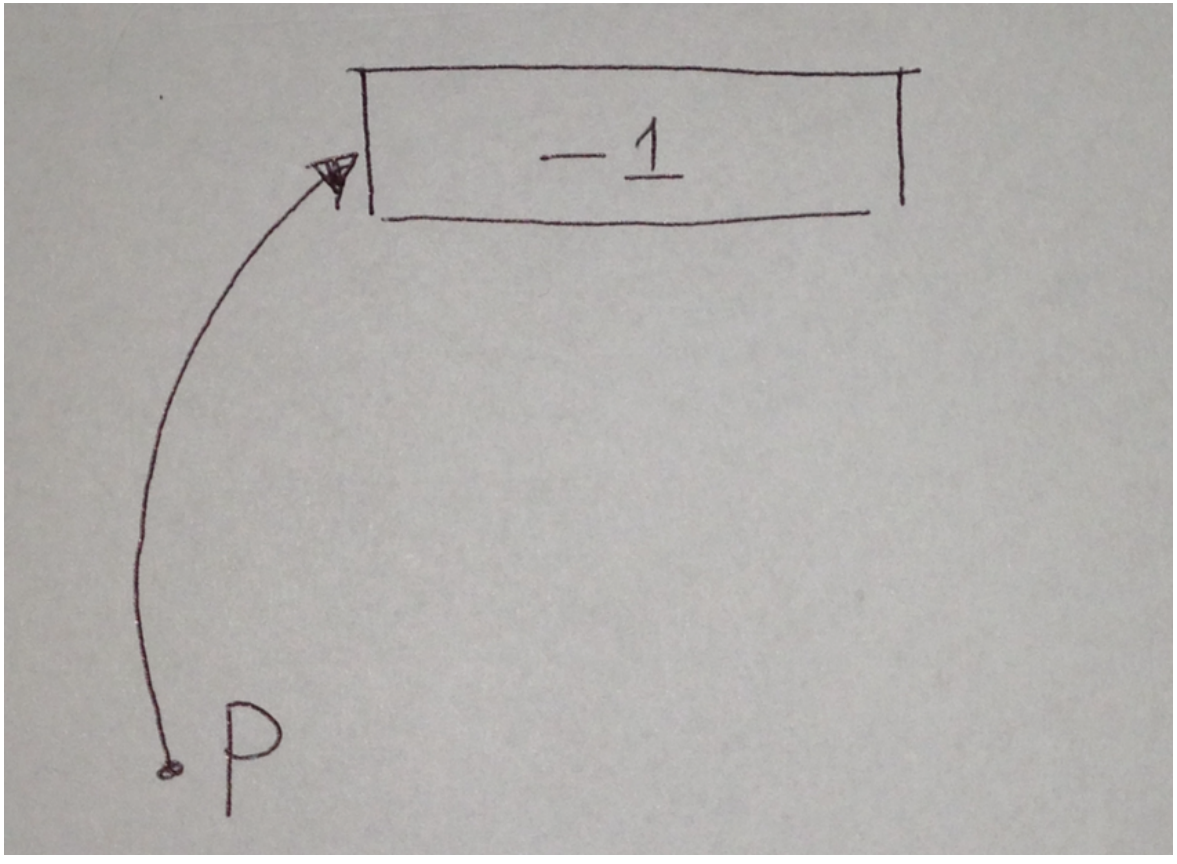
Astrattamente

Un puntatore è il nome di una freccia che punta a una cella di memoria.



Concretamente

Un puntatore è una variabile il cui contenuto è l'indirizzo di una cella di memoria.



I puntatori in C: *Referencing* (&)

L'operatore unario &, prefisso al nome di un oggetto che risiede in memoria — una variabile — costituisce un'espressione che rappresenta l'indirizzo in memoria dell'oggetto.

I puntatori in C: *Referencing* (&)

L'operatore unario `&`, prefisso al nome di un oggetto che risiede in memoria — una variabile — costituisce un'espressione che rappresenta l'indirizzo in memoria dell'oggetto.
Il brano di codice seguente stampa l'indirizzo in memoria (espresso in notazione esadecimale) della variabile intera `x`, assieme al valore della variabile, cioè il contenuto del suo indirizzo.

I puntatori in C: *Referencing* (&)

L'operatore unario &, prefisso al nome di un oggetto che risiede in memoria — una variabile — costituisce un'espressione che rappresenta l'indirizzo in memoria dell'oggetto.
Il brano di codice seguente stampa l'indirizzo in memoria (espresso in notazione esadecimale) della variabile intera x, assieme al valore della variabile, cioè il contenuto del suo indirizzo.

punt1.c

```
1  #include <stdio.h>
2  int main(void)
3  {
4          int x=-2;
5          printf("&x=%p\n", &x); // indirizzo di x
6          printf("x=%d\n", x); // valore di x
7          return 0;
8  }
```

I puntatori in C: *Referencing* (&)

L'operatore unario &, prefisso al nome di un oggetto che risiede in memoria — una variabile — costituisce un'espressione che rappresenta l'indirizzo in memoria dell'oggetto.

La specifica di conversione %p di `printf` indica che si intende stampare un indirizzo.

```
5       printf("&x=%p\n", &x); // indirizzo di x
```

I puntatori in C: *Referencing* (&)

Si noti che non ha senso premettere & ad un'espressione che non denoti una variabile.

I puntatori in C: *Referencing* (&)

Si noti che non ha senso premettere & ad un'espressione che non denoti una variabile.

Per esempio, il brano seguente produce un errore in compilazione.

I puntatori in C: *Referencing* (&)

Si noti che non ha senso premettere & ad un'espressione che non denoti una variabile.

Per esempio, il brano seguente produce un errore in compilazione.

punt2.c

```
1  #include <stdio.h>
2  int main(void)
3  {
4          printf("&10=%p\n", &10); // ERRORE
5          return 0;
6  }
```

I puntatori in C: *Dereferencing* (*)

L'operatore unario *, prefisso a un indirizzo, costituisce un'espressione che rappresenta il contenuto in memoria dell'indirizzo.

I puntatori in C: *Dereferencing* (*)

L'operatore unario *, prefisso a un indirizzo, costituisce un'espressione che rappresenta il contenuto in memoria dell'indirizzo.
Il brano di codice seguente stampa il contenuto (*) dell'indirizzo in memoria (&x) della variabile intera x, ossia stampa semplicemente il valore della variabile.

I puntatori in C: *Dereferencing* (*)

L'operatore unario *, prefisso a un indirizzo, costituisce un'espressione che rappresenta il contenuto in memoria dell'indirizzo.
Il brano di codice seguente stampa il contenuto (*) dell'indirizzo in memoria (&x) della variabile intera x, ossia stampa semplicemente il valore della variabile.

punt3.c

```
1  #include <stdio.h>
2  int main(void)
3  {
4          int x=-2;
5          printf("*(&x)=%d\n", *(&x)); // equivalente alla riga seguente.
6          printf("x=%d\n", x);
7          return 0;
8  }
```

I puntatori in C: *Dereferencing* (*)

Si noti che non ha senso premettere * ad un'espressione che non denoti un indirizzo di memoria.

I puntatori in C: *Dereferencing* (*)

Si noti che non ha senso premettere * ad un'espressione che non denoti un indirizzo di memoria.

Per esempio, il brano seguente produce un errore in compilazione. (*Nota terminologica.* L'operatore * è anche detto di *indirection*.)

I puntatori in C: *Dereferencing* (*)

Si noti che non ha senso premettere * ad un'espressione che non denoti un indirizzo di memoria.

Per esempio, il brano seguente produce un errore in compilazione. (*Nota terminologica.* L'operatore * è anche detto di *indirection*.)

punt4.c

```
1  #include <stdio.h>
2  int main(void)
3  {
4          int x=-2;
5          printf("*x=%d\n", *x)); // ERRORE
6          return 0;
7  }
```

Riassunto: & e *

- Gli operatori unari & e * forniscono rispettivamente l'indirizzo di memoria di un oggetto, e il contenuto di un indirizzo di memoria. I due operatori sono uno l'inverso dell'altro.

Riassunto: & e *

- Gli operatori unari & e * forniscono rispettivamente l'indirizzo di memoria di un oggetto, e il contenuto di un indirizzo di memoria. I due operatori sono uno l'inverso dell'altro.
- Un'espressione cui si è applicato (legittimamente) & diviene dunque un puntatore.

Riassunto: & e *

- Gli operatori unari & e * forniscono rispettivamente l'indirizzo di memoria di un oggetto, e il contenuto di un indirizzo di memoria. I due operatori sono uno l'inverso dell'altro.
- Un'espressione cui si è applicato (legittimamente) & diviene dunque un puntatore.
- Un'espressione cui si è applicato (legittimamente) * è dunque un valore di tipo coincidente col tipo del valore denotato.

Riassunto: & e *

- Gli operatori unari & e * forniscono rispettivamente l'indirizzo di memoria di un oggetto, e il contenuto di un indirizzo di memoria. I due operatori sono uno l'inverso dell'altro.
- Un'espressione cui si è applicato (legittimamente) & diviene dunque un puntatore.
- Un'espressione cui si è applicato (legittimamente) * è dunque un valore di tipo coincidente col tipo del valore denotato.
- Nella pratica, & si applica tipicamente a variabili per ottenerne l'indirizzo, mentre * si applica tipicamente a puntatori per ottenere il valore cui essi puntano.

Tipo dei puntatori

- Si è visto che un puntatore può essere pensato come una freccia che punta a un valore. Poiché i valori nel linguaggio C hanno un tipo, anche i puntatori ereditano un tipo.

Tipo dei puntatori

- Si è visto che un puntatore può essere pensato come una freccia che punta a un valore. Poiché i valori nel linguaggio C hanno un tipo, anche i puntatori ereditano un tipo.
- Più precisamente, se un certo dato in memoria ha tipo `T`, e se `p` è un puntatore a quel dato, allora il tipo di `p` è:

puntatore a `T`.

Tipo dei puntatori

- Si è visto che un puntatore può essere pensato come una freccia che punta a un valore. Poiché i valori nel linguaggio C hanno un tipo, anche i puntatori ereditano un tipo.
- Più precisamente, se un certo dato in memoria ha tipo `T`, e se `p` è un puntatore a quel dato, allora il tipo di `p` è:

  puntatore a `T`.

- Molto importante. Non confondere i due tipi:
  - 'T' e
  - 'puntatore a T'.

  Si tratta di due tipi diversi.

Dichiarazioni di puntatori

- Sintassi. Se `T` è un tipo, per dichiarare un puntatore di nome `ptr` a `T` si usa l'istruzione:

```
T *ptr;
```

Dichiarazioni di puntatori

- Sintassi. Se `T` è un tipo, per dichiarare un puntatore di nome `ptr` a `T` si usa l'istruzione:

```
T *ptr;
```

- Semantica. La dichiarazione definisce l'identificatore `ptr` e alloca adeguata memoria per esso, ma senza inizializzarlo. Dopo la dichirazione il valore del puntatore è indeterminato.

Dichiarazioni di puntatori

- Sintassi. Se `T` è un tipo, per dichiarare un puntatore di nome `ptr` a `T` si usa l'istruzione:

```
T *ptr;
```

- Semantica. La dichiarazione definisce l'identificatore `ptr` e alloca adeguata memoria per esso, ma senza inizializzarlo. Dopo la dichirazione il valore del puntatore è indeterminato.
- *Esempio.* Dichiarazione di un puntatore a `char`:

```
char *pcar;
```

Puntatori `NULL`

- Ad un puntatore si assegna di solito l'indirizzo di un oggetto già allocato in memoria in precedenza, e non certo un intero qualunque. Fa eccezione l'intero 0: si può sempre assegnare questo valore a un qualunque puntatore di qualunque tipo. Il C definisce la costante simbolica `NULL`, il cui valore è appunto 0. In coppia con i puntatori si usa sempre `NULL` al posto del valore 0.

Puntatori NULL

- Ad un puntatore si assegna di solito l'indirizzo di un oggetto già allocato in memoria in precedenza, e non certo un intero qualunque. Fa eccezione l'intero 0: si può sempre assegnare questo valore a un qualunque puntatore di qualunque tipo. Il C definisce la costante simbolica `NULL`, il cui valore è appunto 0. In coppia con i puntatori si usa sempre `NULL` al posto del valore 0.
- La definizione (ossia, dichiarazione e inizializzazione) seguente:

```
char *pcar = NULL;
```

  dice che `pcar` è un puntatore a `char` che al momento non punta ad alcunché. (Nota differenza con valore indeterminato.)

Assegnazioni fra puntatori

Le assegnazioni fra puntatori seguono l'usuale semantica per copia degli assegnamenti. Per non sbagliare occorre però sempre tenere presente che il valore di un puntatore è un indirizzo, ossia, più astrattamente, una freccia.

Assegnazioni fra puntatori

Le assegnazioni fra puntatori seguono l'usuale semantica per copia degli assegnamenti. Per non sbagliare occorre però sempre tenere presente che il valore di un puntatore è un indirizzo, ossia, più astrattamente, una freccia.

Esempio. Siano `x` e `y` variabili di tipo `int` e valore 0 e 1, rispettivamente. Siano inoltre `px` e `py` puntatori a `int` di valore `&x` e `&y`, rispettivamente.

Assegnazioni fra puntatori

Le assegnazioni fra puntatori seguono l'usuale semantica per copia degli assegnamenti. Per non sbagliare occorre però sempre tenere presente che il valore di un puntatore è un indirizzo, ossia, più astrattamente, una freccia.

Esempio. Siano `x` e `y` variabili di tipo `int` e valore 0 e 1, rispettivamente. Siano inoltre `px` e `py` puntatori a `int` di valore `&x` e `&y`, rispettivamente.

Allora l'assegnamento:

```
px=py
```

fa sì che a stampare il valore di `*px` e `*py` si ottenga 1 e 1, rispettivamente, mentre i valori di `x` e `y` sono invariati.

Puntatori ed array

Nel linguaggio C esiste una relazione molto stretta fra puntatori ed array. *Qualunque operazione eseguibile usando gli indici degli array si può anche eseguire usando i puntatori.*

Puntatori ed array

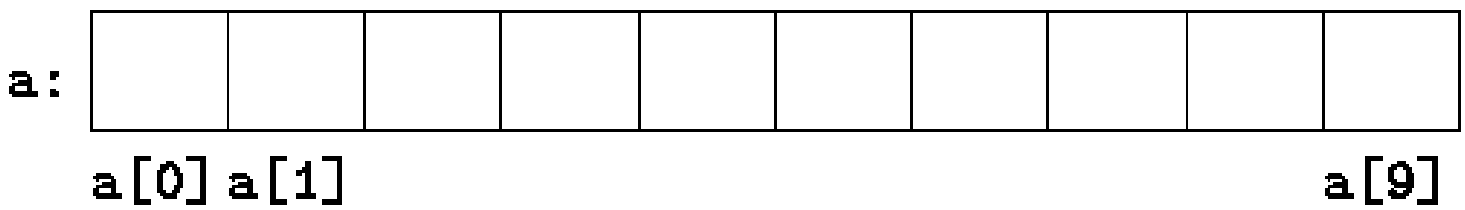
Nel linguaggio C esiste una relazione molto stretta fra puntatori ed array. *Qualunque operazione eseguibile usando gli indici degli array si può anche eseguire usando i puntatori.*

Abbiamo visto il significato della dichiarazione

```
int a[10];
```

e dell'espressione

```
a[i].
```



Se pa è un puntatore a un intero, dichiarato con

```
int *pa;
```

l'assegnazione

```
pa=&a[0];
```

fa puntare pa al primo elemento (indice 0) dell'array a. Quindi pa contiene l'indirizzo di a[0].

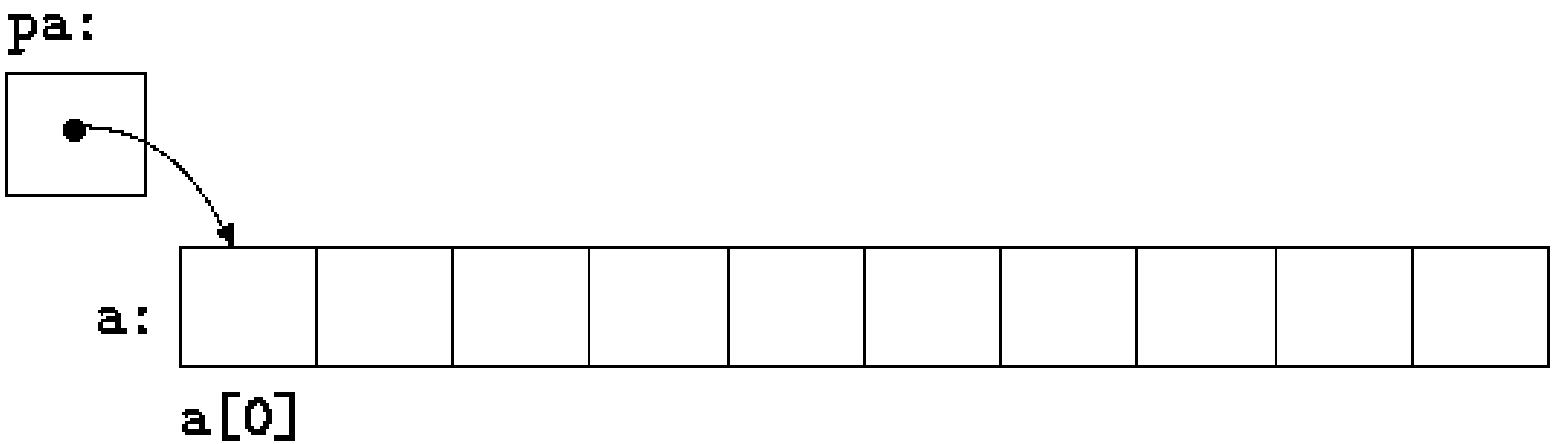
Se pa è un puntatore a un intero, dichiarato con

```
int *pa;
```

l'assegnazione

```
pa=&a[0];
```

fa puntare pa al primo elemento (indice 0) dell'array a. Quindi pa contiene l'indirizzo di a[0].



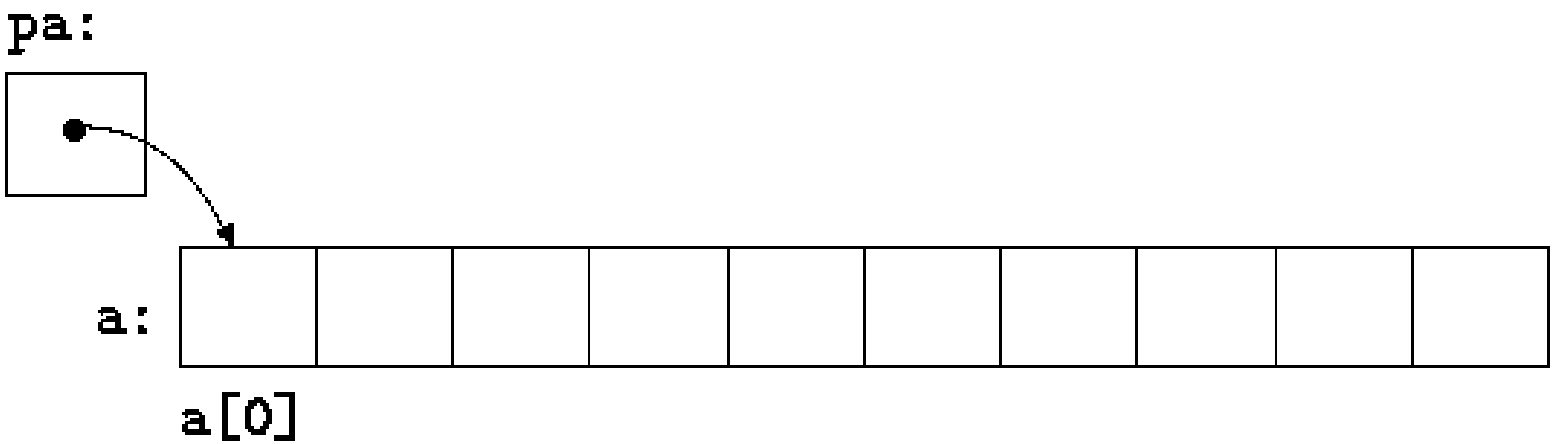
Se pa è un puntatore a un intero, dichiarato con

```
int *pa;
```

l'assegnazione

```
pa=&a[0];
```

fa puntare pa al primo elemento (indice 0) dell'array a. Quindi pa contiene l'indirizzo di a[0].

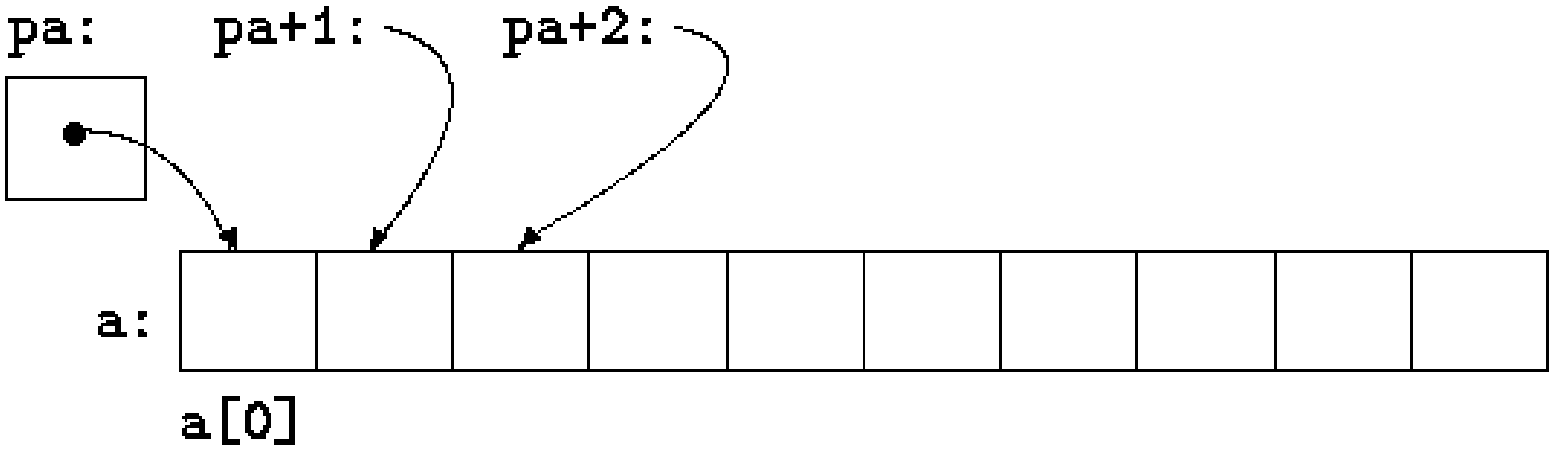


In questa situazione, se x è di tipo int, l'assegnazione x=*pa; copia in x il valore di a[0].

Se `pa` punta a una specifica posizione di un array (del tipo appropriato), allora per definizione `pa+1` punta alla prossima posizione di quell'array.

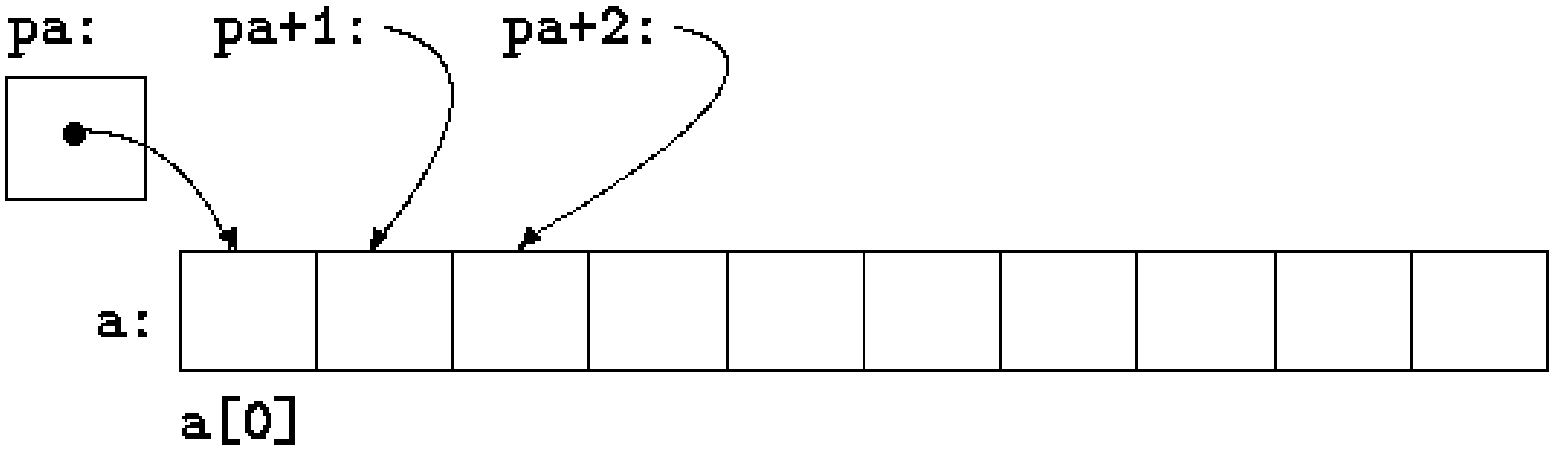
Se `pa` punta a una specifica posizione di un array (del tipo appropriato), allora per definizione `pa+1` punta alla prossima posizione di quell'array.

Analogamente, `pa+i` punta `i` elementi dopa `pa`, e `pa-i` punta `i` elementi prima di `a`.



Se `pa` punta a una specifica posizione di un array (del tipo appropriato), allora per definizione `pa+1` punta alla prossima posizione di quell'array.

Analogamente, `pa+i` punta `i` elementi dopa `pa`, e `pa-i` punta `i` elementi prima di `a`.



Ciò vale indipendentemente dai tipi coinvolti: in questo esempio `a` è un array di `int`, ma il tutto varrebbe anche per `float`, `double`, `char`, ecc.

La stretta correlazione fra array e puntati nel linguaggio C dipende principalmente dal fatto che, nel nostro esempio `int a[10]`, l'espressione

`a`

ha valore pari all'*indirizzo* dell'elemento di indice zero dell'array.

La stretta correlazione fra array e puntati nel linguaggio C dipende principalmente dal fatto che, nel nostro esempio `int a[10]`, l'espressione

`a`

ha valore pari all'*indirizzo* dell'elemento di indice zero dell'array.

In generale: per definizione, il valore del nome di un array (senza quadre) — come pure di un'espressione di tipo array — è l'indirizzo dell'elemento zero dell'array.

La stretta correlazione fra array e puntati nel linguaggio C dipende principalmente dal fatto che, nel nostro esempio `int a[10]`, l'espressione

`a`

ha valore pari all'*indirizzo* dell'elemento di indice zero dell'array.

In generale: per definizione, il valore del nome di un array (senza quadre) — come pure di un'espressione di tipo array — è l'indirizzo dell'elemento zero dell'array.

L'assegnazione

`pa=&a[0]`

ha l'effetto di far puntare `pa` al primo elemento di `a`. Ma la stessa cosa si può scrivere:

La stretta correlazione fra array e puntati nel linguaggio C dipende principalmente dal fatto che, nel nostro esempio `int a[10]`, l'espressione

```
a
```

ha valore pari all'*indirizzo* dell'elemento di indice zero dell'array.

In generale: per definizione, il valore del nome di un array (senza quadre) — come pure di un'espressione di tipo array — è l'indirizzo dell'elemento zero dell'array.

L'assegnazione

```
pa=&a[0]
```

ha l'effetto di far puntare `pa` al primo elemento di `a`. Ma la stessa cosa si può scrivere:

```
pa=a;
```

C'è però una differenza importante fra `pa` e `a`.

C'è però una differenza importante fra pa e a.

- Un puntatore è una variabile: istruzioni come pa=pa+1, pa++ e pa=a sono perfettamente legittime.

C'è però una differenza importante fra pa e a.

- Un puntatore è una variabile: istruzioni come pa=pa+1, pa++ e pa=a sono perfettamente legittime.
- Il nome di un array non è una variabile: istruzioni come a=a+1, a++ e a=pa non sono legittime.

C'è però una differenza importante fra pa e a.

- Un puntatore è una variabile: istruzioni come pa=pa+1, pa++ e pa=a sono perfettamente legittime.
- Il nome di un array non è una variabile: istruzioni come a=a+1, a++ e a=pa non sono legittime.

Si può quindi pensare al nome di un array come a un puntatore (del tipo appropriato) che sia però costante.

C'è però una differenza importante fra `pa` e `a`.

- Un puntatore è una variabile: istruzioni come `pa=pa+1`, `pa++` e `pa=a` sono perfettamente legittime.
- Il nome di un array non è una variabile: istruzioni come `a=a+1`, `a++` e `a=pa` non sono legittime.

Si può quindi pensare al nome di un array come a un puntatore (del tipo appropriato) che sia però costante.

Possiamo ora illustrare l'affermazione iniziale di questa sezione. L'espressione `a[i]` si può riscrivere con i puntatori in questo modo

```
*(a+i).
```

C'è però una differenza importante fra `pa` e `a`.

- Un puntatore è una variabile: istruzioni come `pa=pa+1`, `pa++` e `pa=a` sono perfettamente legittime.
- Il nome di un array non è una variabile: istruzioni come `a=a+1`, `a++` e `a=pa` non sono legittime.

Si può quindi pensare al nome di un array come a un puntatore (del tipo appropriato) che sia però costante.

Possiamo ora illustrare l'affermazione iniziale di questa sezione. L'espressione `a[i]` si può riscrivere con i puntatori in questo modo

```
*(a+i).
```

In effetti, le due espressioni hanno significato identico per il compilatore C, che considera la prima come un'abbreviazione della seconda. Applicando `&` a entrambe le espressioni, otteniamo che `&a[i]` e `&(*(a+i))=a+i` sono pure identiche: `a+i` è l'indirizzo dell'elemento di indice `i` dell'array.

Per converso, è possibile applicare al puntatore pa l'operazione di indicizzazione `[]`.

Per converso, è possibile applicare al puntatore pa l'operazione di indicizzazione `[]`.

Nel nostro esempio, assumendo di aver eseguito l'assegnazione

```
pa=a;
```

avremo che l'espressione

```
pa[2]
```

è del tutto equivalente all'espressione

```
a[2].
```

Per converso, è possibile applicare al puntatore pa l'operazione di indicizzazione `[]`.

Nel nostro esempio, assumendo di aver eseguito l'assegnazione

```
pa=a;
```

avremo che l'espressione

```
pa[2]
```

è del tutto equivalente all'espressione

```
a[2].
```

Morale: *Qualunque operazione eseguibile usando gli indici degli array si può anche eseguire usando i puntatori.*

Inizializzazione degli array

arrinit.c

```c
#include <stdio.h>

int main(void)
{
        //Inizializzazione degli array
        double f[5]={-1, 2.4, 0, 50, -12.34};
        int g[]={1,2,3}; //Alloca int[] di 3 elementi
        int h[3]={1,2}; //Elementi non specificati (h[2]) messi a 0
        double k[100]={0.0}; //Tutto a 0.0
        //ERRATO: Errore  in compilazione
        //int a[2];
        //a[2]={1,2};
        //ERRATO: Errore in compilazione
        //double b[2]={0.0, 1.1, 2.2};
        return 0;
}
```

Inizializzazione degli array di char (stringhe)

stringinit.c

```c
#include <stdio.h>

int main(void)
{
        //Inizializzazione degli array di char (stringhe)
        char s[]={'P', 'i', 'p', 'p', 'o'}; // No \0 alla fine
        char t[]={'P', 'i', 'p', 'p', 'o', '\0'};// \0 alla fine
        char u[]="Pippo";// \0 alla fine

        printf("%s\n%s\n",t,u);
        //ERRATO: manca \0
        //printf("%s\n",s);
        return 0;
}
```