

I Puntatori

Prof. Salvatore Venticinque

Prof. Mario Magliulo

Salvatore Venticinque

Puntatori

- L'operatore di indirizzo &
- Indirizzi, puntatori
- Aritmetica dei puntatori
- L'operatore di dereferenziazione *

Operatore di indirizzo &

L'operatore unario di indirizzo & restituisce l'indirizzo della locazione di memoria dell'operando

Es.:

&a	<u>ammesso</u>
&(a+1)	<u>non ammesso</u>
&a = b	<u>non ammesso</u>

L'indirizzo di memoria di una variabile non può essere modificato in un'istruzione, ma solo usato come riferimento in quanto è predeterminato e non modificabile.

Il puntatore come tipo

Il puntatore è un tipo di variabile il cui valore è un indirizzo

Es:

```
int* pointer;
```

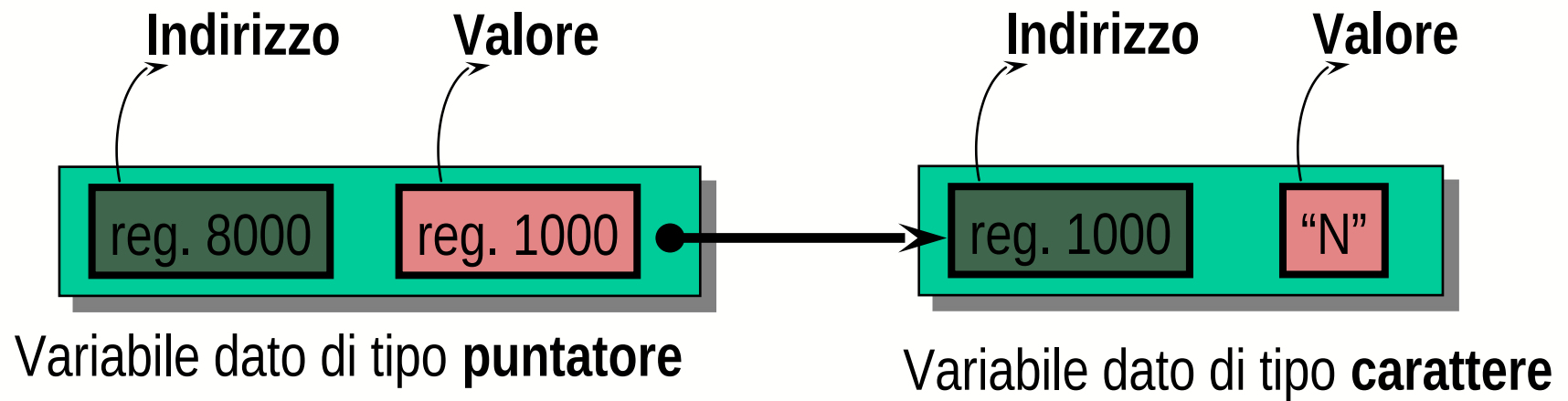
dichiara la variabile **pointer**, **puntatore** *ad una qualunque variabile di tipo **int***

Un dato **puntatore** può puntare solo a un determinato **tipo** di variabili, quello specificato nella dichiarazione

Esistono puntatori per ogni tipo di variabile puntata

Puntatori

Una variabile di **tipo puntatore** é designata a contenere l'indirizzo di memoria di un'altra variabile (detta **variabile puntata**), la quale a sua volta può essere di qualunque **tipo**, anche non **nativo** (persino un altro **puntatore**!).



Assegnazione di valore a un **puntatore**

NON si possono assegnare **valori** a un **puntatore**, salvo che in questi tre casi:

1. a un **puntatore** é assegnato il valore **NULL** (non punta a “niente”)
2. a un **puntatore** è assegnato l'**indirizzo** di una variabile esistente, restituito dall'operatore **&**

(Es. `int a; int* p; p=&a;`)

Che cosa è :

Es.: **double** pointer;**

Come lo inizializziamo?

Deferenziazione

L'**operatore unario di dereferenziazione *** di un **puntatore** restituisce il valore della variabile puntata dall'**operando**:

-

utilizzato a destra dell'assegnazione, esegue un'operazione di **estrazione**

Es.: **a = *p;** assegna ad **a** il valore della variabile puntata da **p**

utilizzato a sinistra dell'assegnazione, esegue un'operazione di **inserimento**

Es.: ***p = a;** assegna il valore di **a** alla variabile puntata da **p**

Deferenziamento

l'operazione di **deref.** é inversa a quella di **indirizzo**.

Se assegniamo a un **puntatore p** l'**indirizzo** di una variabile **a**,

$$p = \&a;$$

allora

$$*p == a$$

cioè la **deref.** di **p** coincide con **a**

L'operatore di **dereferenziazione** *

ATTENZIONE: non é detto il contrario !!!

se si assegna alla **deref.** di **p** il valore di **a**,

***p = a ;**

ciò non comporta automaticamente che in **p** si ritrovi l'**indirizzo** di **a**, ma semplicemente che il valore della variabile puntata da **p** coinciderà con **a**

Esempi

- `int a=4`
- `int *p=NULL;`
- `printf(“%d\n”, a);`
- `printf(“%p\n”,p);`
- `p=&a;`
- `printf(“%p\n”,p);`
- `printf(“%p\n”,&a);`
- `printf(“%p\n”,&p)`
- `printf(“%d\n”, *p);`

Operazioni sui puntatori

Assegnazione di valore (indirizzo) ad un puntatore

int* pi = &i

p = q

Riferimento all'oggetto puntato *pi=3

Confronto tra puntatori (==, !=)

Operazioni sui puntatori

E' possibile specificare che un dato puntatore non deve essere usato per modificare l'oggetto puntato attraverso la parola chiave `const`:

```
int i;
```

```
const int* p=&i;
```

Le istruzioni in cui si usa *p* per aggiornare *i* vengono segnalate come erronee

Operazioni sui puntatori

E' possibile specificare con la parola chiave `const` che un puntatore non deve essere modificato:

```
int i;  
int* const p=&i;
```

Le istruzioni in cui si vuole modificare il *valore di p* vengono segnalate come erronee

Aritmetica dei puntatori

Il valore assunto da un **puntatore**

- **Può essere visto come** un numero **intero** che rappresenta, un **indirizzo** di memoria
- le operazioni di **somma** fra un **puntatore** e un **valore intero** (con risultato **puntatore**), oppure di **sottrazione** fra due **puntatori** (con risultato **intero**) *vengono eseguite tenendo conto del **tipo** della variabile puntata*

Es.: se si incrementa un **puntatore-a-**float**** di 3 unità, il suo valore viene incrementato di 12 byte.

Aritmetica dei puntatori

Se l'espressione p rappresenta l'indirizzo di un oggetto di tipo T , allora l'espressione $p+1$ rappresenta l'indirizzo di un oggetto di tipo T allocato *consecutivamente* in memoria.

In generale: se i è un intero, se p rappresenta l'indirizzo addr di T che occupa n locazioni di memoria, allora l'espressione $p+i$ ha valore $\text{addr} + n \times i$

Aritmetica dei puntatori

Le regole dell'**aritmetica** dei **puntatori** assicurano che il risultato sia sempre corretto, qualsiasi sia la lunghezza in byte della variabile puntata.

Es.: se **p** punta a un **elemento** di un **array**, **p++** punterà all'elemento successivo, qualunque sia il **tipo** (anche non **nativo**) dell'array

Attenzione!

la dichiarazione di un **puntatore** comporta allocazione di memoria per la **variabile puntatore**, ma non per la **variabile puntata**.

Es.: **int* lista;**

alloca memoria per **lista** ma non per la variabile puntata da **lista**

Array e puntatori

Il nome di un array è in realtà un puntatore costante all'indirizzo di partenza dell'array stesso, per cui non può essere modificato nelle operazioni di aritmetica dei puntatori.

I puntatori possono essere utilizzati in tutte le operazioni di indicizzazione di un array

Array e puntatori

La dichiarazione di un **array**:

- allocazione di memoria di una **variabile puntatore** (il **nome** dell'**array**),
- Allocazione dell'**area puntata**, di lunghezza predefinita

*il **puntatore** é dichiarato **const** e inizializzato con l'**indirizzo** dell'**area puntata** (cioè del primo elemento dell'**array**)*

Es.: **int lista[5];**

- alloca memoria per il **puntatore costante lista**;
- alloca memoria per **5** valori di tipo **int**;
- inizializza **lista** con **&lista[0]**

Array e puntatori (1)

Il **nome** di un **array** è il **puntatore** al primo **elemento** dell'array

Ogni altro elemento é accessibile:

- tramite la **deref.** del **puntatore-array** incrementato di una quantità pari all'**indice/offset** dell'**elemento**
- tramite le parentesi quadre

A[i] e ***(A+i)**

conducono ad identico risultato.

Array e puntatori (2)

Se $A[N]$ è un array di indirizzo ind ,
 $A[i]$ il suo elemento i -esimo (i intero),
l'indirizzo di $A[i]$ sarà dato dall'espressione $*(ind+i)$

Array di puntatori

I puntatori, come qualsiasi altra variabile, possono essere raggruppati in array e dichiarati come:

Es.: **int* A[10] ;**

dichiara un array di 10 puntatori a int

Come il nome di un array equivale a un puntatore, così un array di puntatori equivale a un puntatore a puntatore (con in più l'allocazione della memoria puntata, come nel caso di array generico)

Array di Stringhe

Il caso più frequente di array di puntatori é quello dell'array di stringhe, che consente anche l'inizializzazione (atipicamente) delle stringhe che costituiscono l'array

Es.: **char* colori[3] = {"Blu", "Rosso", "Verde"} ;**

- Le stringhe possono essere di differente lunghezza;
- in memoria sono allocate consecutivamente;
- sono riservati tanti bytes quant'è la rispettiva lunghezza (terminatore compreso).

Array di puntatori

`suit [4]` indica un array di 4 elementi

```
char* suit[4] = {"Heart",  
                 "Diamond", "Clubs",  
                 "Spades"}
```

- Ogni elemento dell'array è un puntatore ad un **char**, l'array contiene in realtà soltanto i puntatori al primo carattere di ogni stringa.