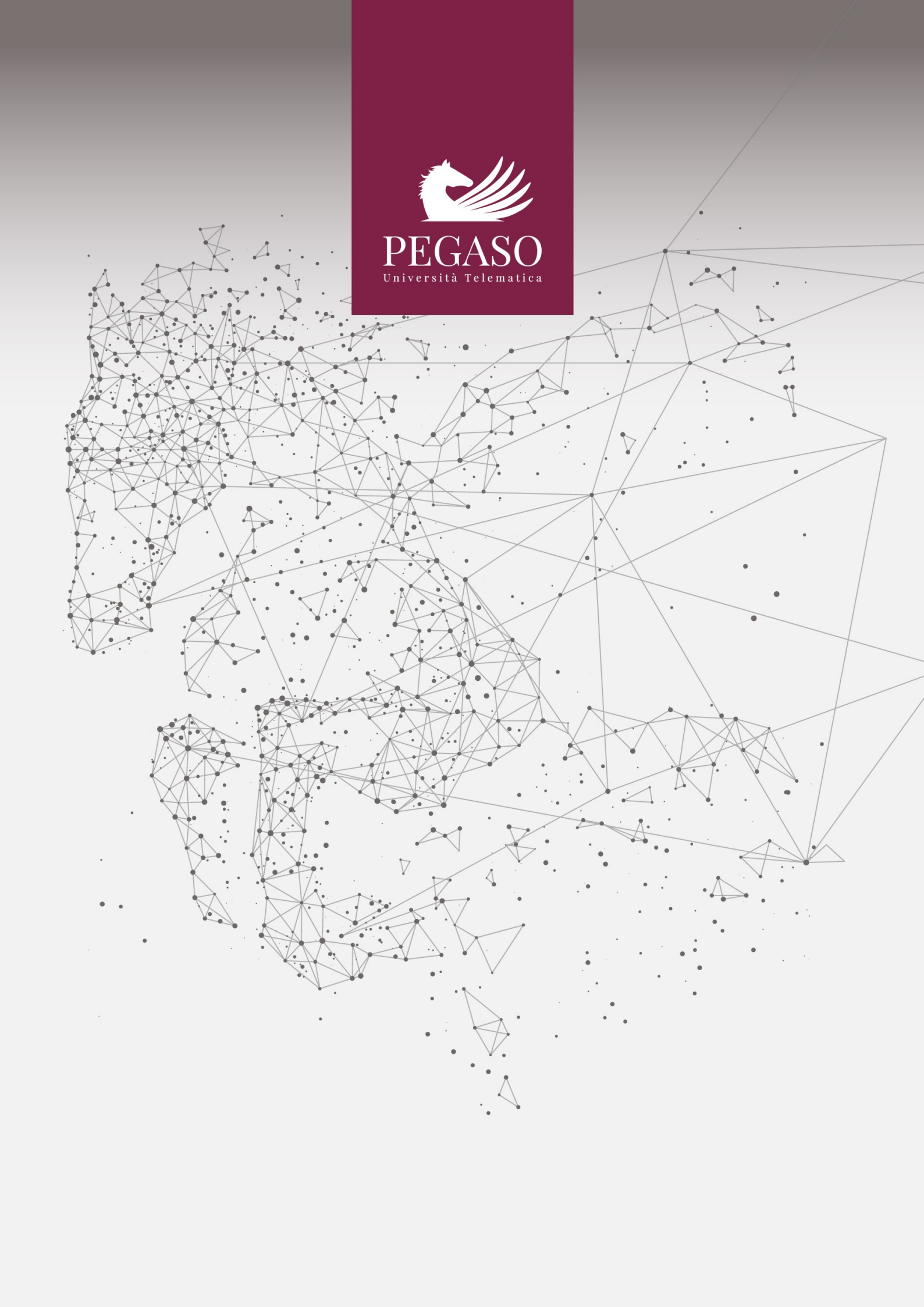

PEGASO
Università Telematica

Indice

1. Definizione e inizializzazione di variabili puntatore

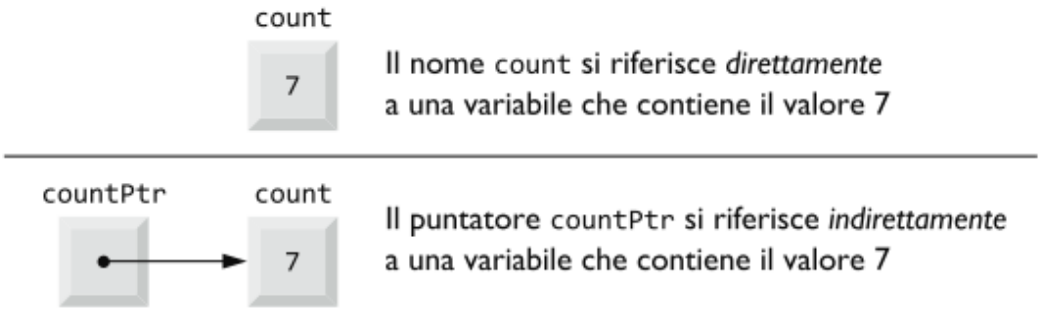
I puntatori sono variabili i cui valori sono *indirizzi di memoria*.

Normalmente, una variabile contiene direttamente un valore specifico.

Un puntatore, tuttavia, contiene un *indirizzo* di una variabile che contiene un valore specifico.

In questo senso, il nome di una variabile fa *direttamente* riferimento a un valore, mentre un puntatore fa *indirettamente* riferimento a un valore.

Far riferimento a un valore per mezzo di un puntatore si dice **indirezione**.



Dichiarare i puntatori

I puntatori, come tutte le variabili, devono essere definiti prima di essere utilizzati.

La definizione

```
int *countPtr, count;
```

specifica che la variabile `countPtr` è del tipo `int *` (cioè un puntatore a un intero) e si legge (da destra a sinistra) "`countPtr` è un puntatore a un `int`" oppure "`countPtr` punta a un'area di memoria che contiene una variabile di tipo `int`".

Inoltre, la variabile `count` è definita di tipo `int`, non come un puntatore a un `int`.

Il simbolo `*` si applica nella definizione solo a `countPtr`.

Quando il simbolo `*` è usato in questo modo in una definizione, indica che la variabile che viene definita è un puntatore.

I puntatori possono essere definiti per puntare a oggetti di qualsiasi tipo.

Per evitare l'ambiguità che si crea dichiarando nella stessa dichiarazione le variabili puntatore e non puntatore, come mostrato in precedenza, è opportuno dichiarare sempre soltanto una variabile per dichiarazione.

☹ *Errore comune di programmazione*

La notazione con l'asterisco (`*`) usata per dichiarare le variabili puntatore non viene distribuita a tutti i nomi delle variabili in una dichiarazione.

Ogni puntatore deve essere dichiarato con il simbolo `*` prefissato al nome; ad esempio, se desiderate dichiarare `xPtr` e `yPtr` come puntatori a oggetti di tipo `int`, usate

```
int *xPtr, *yPtr; .
```

😊 *Buona pratica di programmazione*

È preferibile includere le lettere `Ptr` nei nomi delle variabili puntatore per rendere chiaro che queste variabili sono puntatori e di conseguenza devono essere trattate adeguatamente.

Inizializzare e assegnare valori ai puntatori

I puntatori devono essere inizializzati quando sono definiti, oppure assegnando loro un valore.

Un puntatore può essere inizializzato a `NULL`, 0 o a un indirizzo.

Un puntatore con il valore `NULL` non punta a niente.

NULL è una costante simbolica definita nell'intestazione `<stddef.h>` (e in diverse altre intestazioni, come `<stdio.h>`) .

Inizializzare un puntatore a 0 equivale a inizializzare un puntatore a NULL, ma NULL è preferibile, poiché evidenzia il fatto che la variabile è di un tipo puntatore.

Quando si assegna 0, questo viene prima convertito a un puntatore del tipo appropriato. Il valore 0 è l'unico valore intero che si può assegnare direttamente a una variabile puntatore.

😊 *Prevenzione di errori*

Inizializzate i puntatori per prevenire risultati inaspettati.

2. Operatori per i puntatori

Presentiamo ora gli operatori di indirizzo (`&`) e di indirezione (`*`), e la relazione che intercorre tra essi.

Operatore di indirizzo `&`

L'operatore di indirizzo & è un operatore unario che restituisce l'indirizzo del suo operando.

Ad esempio, presupponendo le definizioni

```
int y = 5;
int *yPtr;
```

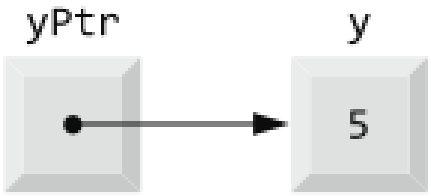
l'istruzione

```
yPtr = &y;
```

assegna l'indirizzo della variabile y alla variabile puntatore `yPtr`.

Si dice pertanto che la variabile `yPtr` "punta a" `y`.

La Figura successiva mostra una rappresentazione schematica della memoria dopo l'esecuzione dell'assegnazione precedente.



Rappresentazione di un puntatore in memoria

La Figura successiva mostra la rappresentazione del puntatore precedente in memoria, supponendo che la variabile intera y sia memorizzata alla locazione 600000 e che la variabile puntatore `yPtr` sia memorizzata alla locazione 500000.

L'operando dell'operatore di indirizzo deve essere una variabile; l'operatore di indirizzo non può essere applicato a costanti o a espressioni.

Operatore di indirezione *

L'operatore unario *, comunemente detto operatore di indirezione o operatore di dereferenziazione, restituisce il valore dell'oggetto al quale punta il suo operando (un puntatore). Ad esempio, l'istruzione

```
printf("%d", *yPtr);
```

stampa il valore della variabile `y`, cioè 5.

Usare l'operatore * in questo modo equivale a dereferenziare un puntatore.

☹ *Errore comune di programmazione*

Dereferenziare un puntatore che non è stato correttamente inizializzato o a cui non è stato assegnato l'indirizzo di una specifica locazione di memoria è un errore.

Ciò potrebbe determinare un errore irreversibile in fase di esecuzione o potrebbe causare la modifica accidentale di dati importanti, permettendo al programma di completare l'esecuzione con risultati scorretti.

3. Relazioni tra gli operatori per puntatori & e *

Il programma successivo illustra l'uso degli operatori `&` e `*`.

Lo specificatore di conversione `%p` di `printf` invia in uscita la locazione di memoria come un intero esadecimale sulla maggior parte delle piattaforme.

Nell'output del programma, notate che l'indirizzo di `a` e il valore di `aPtr` sono identici nell'output, confermando così che l'indirizzo di a è davvero assegnato alla variabile puntatore `aPtr` (riga 8).

Gli operatori `&` e `*` sono l'uno il complemento dell'altro: quando entrambi sono applicati consecutivamente ad `aPtr` in un ordine o nell'altro (riga 18), viene stampato lo stesso risultato. Gli indirizzi mostrati nell'output varieranno a seconda del sistema.

```
1 // Programma con
 2  // uso degli operatori & e *.
 3  #include <stdio.h>
 4
 5  int main(void)
 6  {
 7     int a = 7;
 8     int *aPtr = &a; // imposta aPtr all'indirizzo di a
 9
10     printf("The address of a is %p\n", &a);
11     printf("The value of aPtr is %p\n", aPtr);
12
13     printf("The value of a is %d", a);
14     printf("The value of *aPtr is %d\n", *aPtr);
15
16     printf("Thus, * and & are complements of each other\n");
17     printf("&*aPtr = %p\n", &*aPtr);
18     printf("*&aPtr = %p\n", *&aPtr);
```

```
19  }
```

```
The address of a is 0028FEC0

The value of aPtr is 0028FEC0

The value of a is 7

The value of *aPtr is 7

Thus, * and & are complements of each other

&*aPtr = 0028FEC0

*&aPtr = 0028FEC0
```

La Figura successiva elenca la precedenza e l'associatività degli operatori introdotti fino a questo punto.

Operatori	Associatività	Tipo
() [] ++ (*postfisso*) -- (*postfisso*)	da sinistra a destra	postfisso
+ - ++ -- ! * & (*tipo*)	da destra a sinistra	unario
* / %	da sinistra a destra	moltiplicativo
+ -	da sinistra a destra	additivo
< <= > >=	da sinistra a destra	relazionale
== !=	da sinistra a destra	di uguaglianza
&&	da sinistra a destra	AND logico
||	da sinistra a destra	OR logico
?:	da destra a sinistra	condizionale
= += -= *= /= %=	da destra a sinistra	di assegnazione
,	da sinistra a destra	virgola

Riferimenti bibliografici

- Paul Deitel, Harvey Deitel, "Il linguaggio C – Fondamenti e tecniche di programmazione", Libro edito da Pearson Italia. Include anche utili esercizi di autovalutazione.