



UNIVERSITÀ DI PARMA

Allocazione Dinamica della Memoria e Puntatori

*Memoria est thesaurus omnium rerum et
custos*

De Oratore, Cicerone, I sec a.C.

- Limite array
- Variable Length Array
- I puntatori
- Il tipo void
- Funzioni di gestione memoria
- Errori Comuni

SUMMARY



- Fino ad ora abbiamo usato array con dimensioni costanti
- Ipotesi su quantità dati da gestire
 - Spreco memoria
 - Possibilità di errori → buffer overrun
- Come risolvere?
 - Variable Length Array (VLA)
 - Allocazione dinamica della memoria



- Il C99 ammette di usare una variabile per la dimensione degli array
- Dimensione array decisa in esecuzione
 - Non in codifica!
- Durata sempre automatica
- Non possibile inizializzazione

- Fino al C99 → dimensione array costante
- Per il C99 → dimensione array anche variabile
- Dopo il C99 → dimensione array variabile opzionale

- C++? → ad oggi non previsti nello standard

- Pro
 - Sintassi semplice
- Contro
 - Non portabile
 - C++
 - Non posso dinamicamente cambiare dimensione array
 - Non posso liberare la memoria in uso
 - Dimensioni spesso limitate
 - Durata automatica!

- Il C fornisce primitive di allocazione
 - Sintassi diversa
 - Più flessibili rispetto VLA
- Ma prima dobbiamo introdurre altri elementi
 - Puntatori
 - void
 - size_t

- Principali primitive

```
void *malloc(size_t size);
```

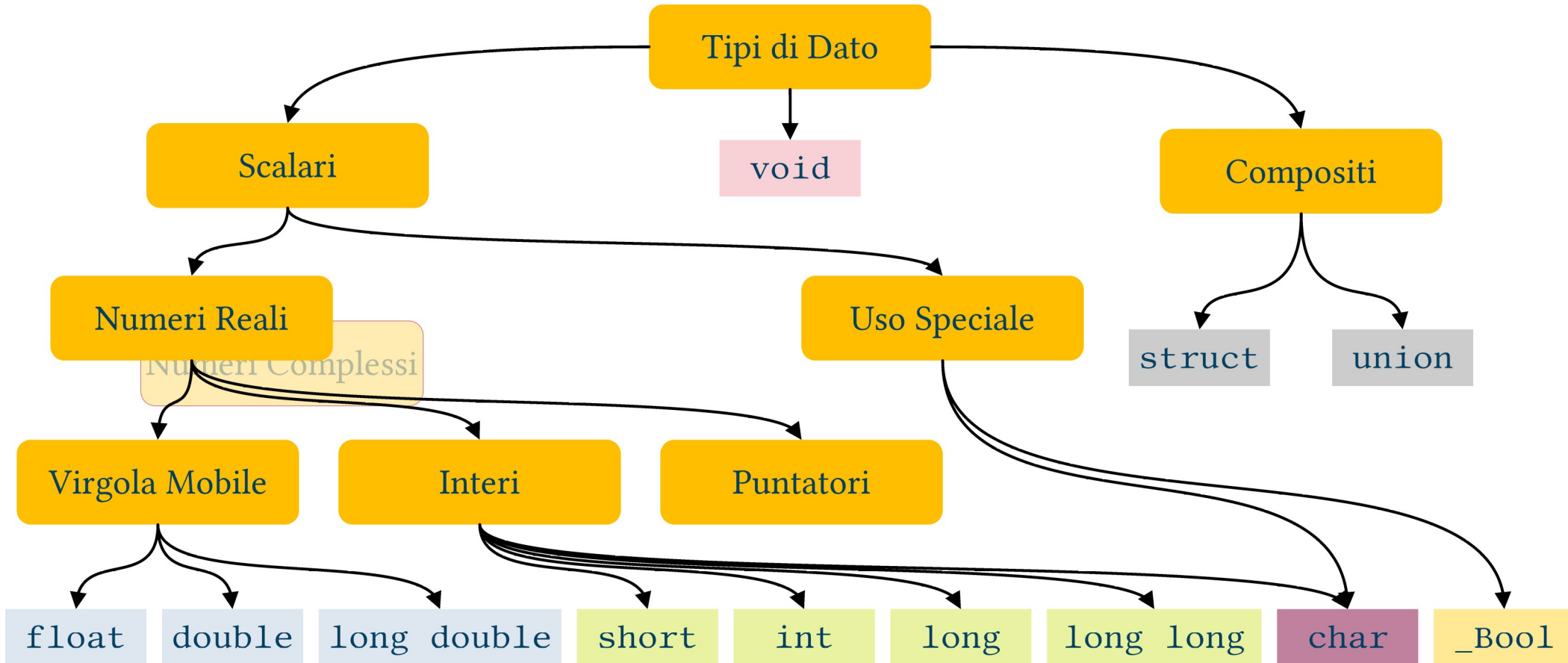
```
void free(void *ptr);
```

```
void *calloc(size_t nmemb, size_t size);
```

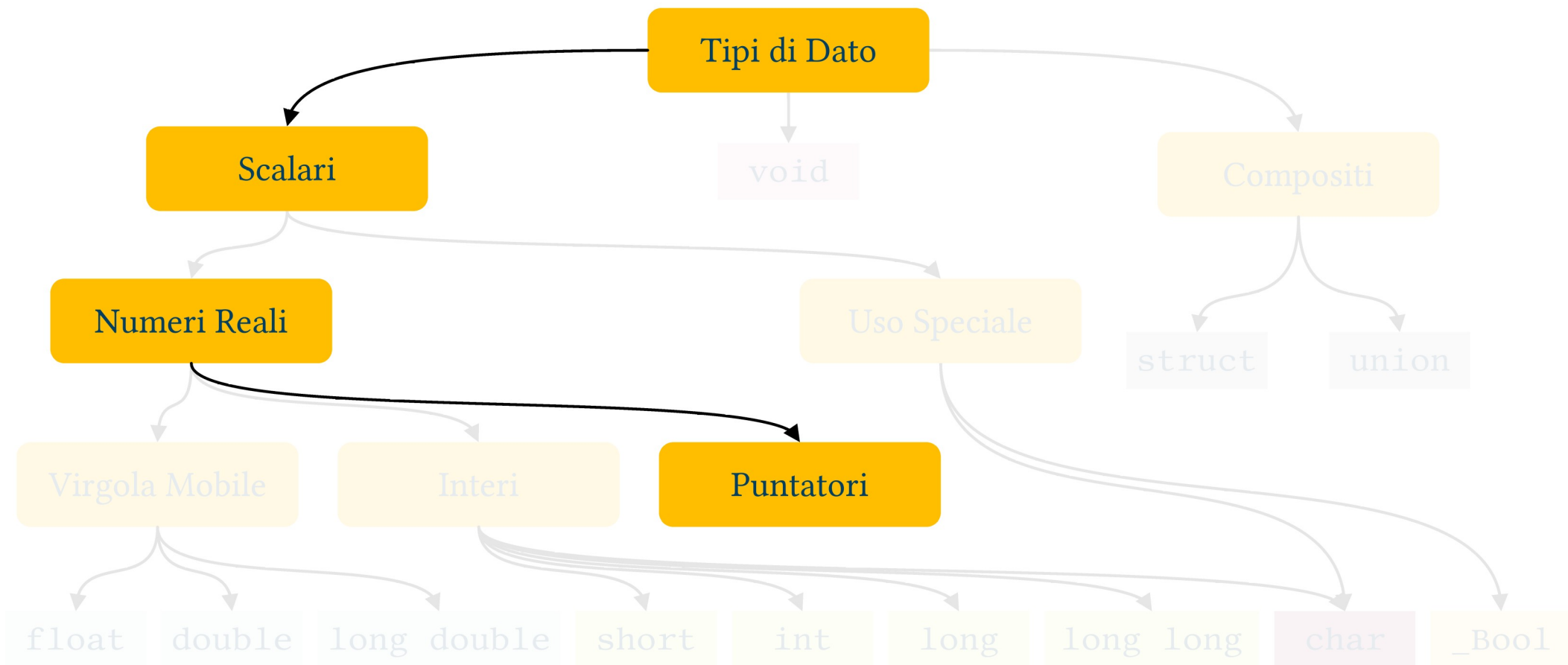
```
void *realloc(void *ptr, size_t size);
```

- Non abbiamo ancora tutti gli elementi per capirle
 - void? *? size_t?

C99 Albero dei Tipi di Dato



C99 Albero dei Tipi di Dato



- Variabile \rightarrow spazio in memoria con nome
 - Dimensione
 - Indirizzo
- Esistono casi in cui indirizzo può servire
- Come lo gestisco? \rightarrow puntatore!
- Un puntatore è un tipo di variabile usato per memorizzare indirizzo di altri dati o elementi del programma

- Come ricavo indirizzo memoria di una variabile?
 - Operatore & per variabili singole
 - Niente per array
 - `printf("L'indirizzo della variabile a e' %p\n", &a);`
 - `printf("L'indirizzo dell'array n e' %p\n", n);`
- Lo posso memorizzare in una variabile puntatore:
 - `int a; // variabile di tipo int`
 - `int *x; // puntatore a un dato di tipo int`
 - `x = &a; // ora x contiene indirizzo di a`

- Per definire un puntatore si usa il carattere ‘*’ prima del nome

```
int *p;    // puntatore a dato di tipo int
```

```
float *x; // puntatore a dato di tipo float
```

```
long a, *b; // solo b è un puntatore!
```

- Dato un puntatore che contiene l'indirizzo di un dato come vi accedo?
- Operatore '*'

```
int a;
```

```
...
```

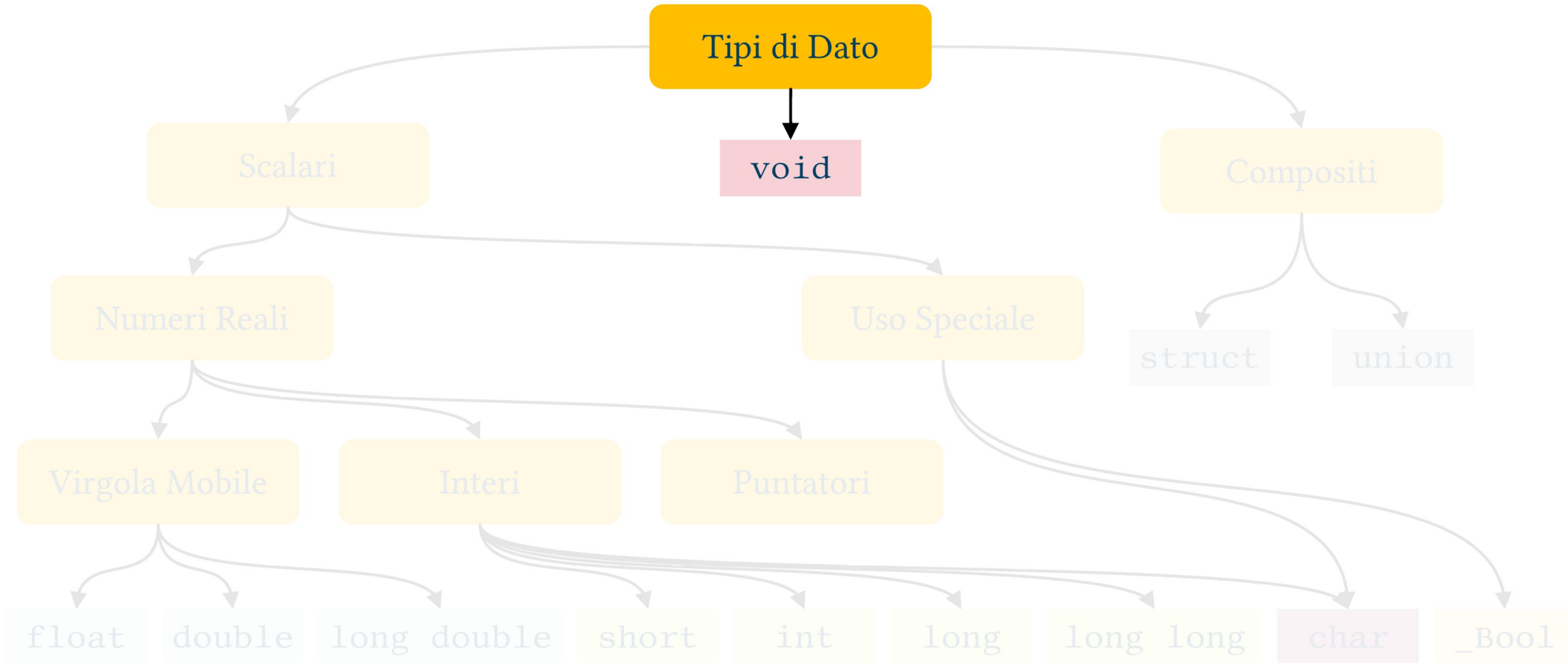
```
int *x=&a;
```

```
printf("All'indirizzo %p e' memorizzato l'int %d\n", x, *x);
```

- Posso ritenere array come puntatori costanti
- Sintassi interscambiabile
 - In particolare []
 - Eccezione sizeof()

- Valore convenzionale usato per indicare indirizzo non valido
- Uso tipico
 - Funzioni che restituiscono indirizzo, se falliscono o non trovano risultato utile restituiscono NULL
 - Come argomento per alcune funzioni

C99 Albero dei Tipi di Dato



- È un *non tipo*
- Non usabile per definizione variabili
 - Funzioni
 - No value
 - No parameters
 - Puntatori
 - No type

- Tecnicamente non è un tipo di dato
- Alias del tipo di dato in grado di contenere la dimensione in byte massima gestibile dal sistema
 - Tipo restituito da sizeof()
- unsigned
- Nei sistemi 64 bit di solito unsigned long long o unsigned long
 - Per stampare %zt (C99, in laboratorio non funziona → %ld)

- Principali primitive

```
void *malloc(size_t size);
```

```
void free(void *ptr);
```

```
void *calloc(size_t nmemb, size_t size);
```

```
void *realloc(void *ptr, size_t size);
```

- Sintassi adesso spiegabile

```
void *malloc(size_t size);
```

- Funzione
 - Prende in ingresso → quantità di memoria
 - Alloca tale quantità di memoria
 - Se possibile
 - Restituisce → indirizzo del buffer allocato
 - NULL se non possibile allocare

void free(void *ptr);

- Funzione
 - Prende in ingresso → indirizzo buffer allocato
 - “Disalloca” la memoria
 - Non restituisce niente
- In C non esiste Garbage Collector
 - Se non libero memoria → occupata per sempre!



```
void *calloc(size_t nmemb, size_t size);
```

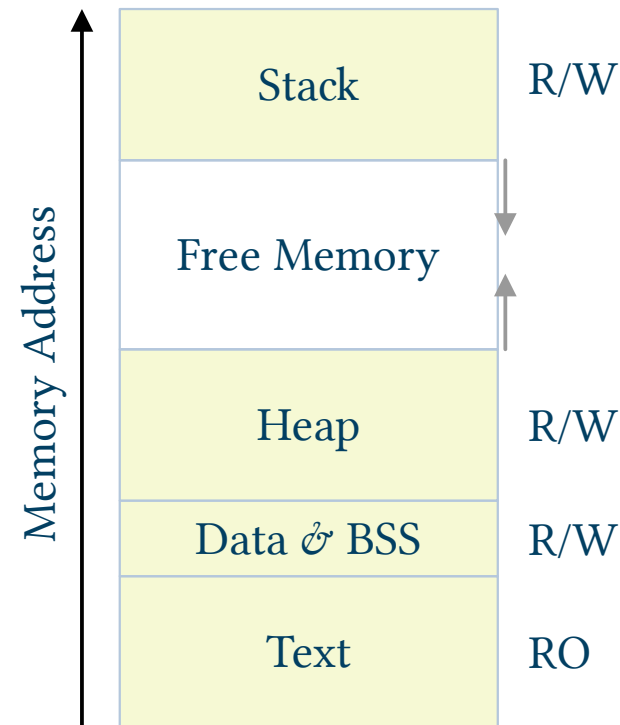
- Funzione
 - Prende in ingresso → numero elementi e loro dimensione
 - Alloca tale “array”
 - Se possibile
 - Lo inizializza a 0
 - Restituisce → indirizzo del buffer allocato
 - NULL se non possibile allocare

```
void *realloc(void *ptr, size_t size);
```

- Funzione

- Prende in ingresso → buffer da ridimensionare, nuova dimensione
- Cambia dimensione buffer
- Ricopia dati
- Restituisce → nuovo indirizzo del buffer allocato
 - NULL se non possibile ridimensionare

- Diverse aree
 - Stack (.stack) → variabili locali, VLA
 - Heap (.heap) → buffer allocati (malloc()...)
 - Code (.text) → codice
 - Data → variabili globali o statiche
 - .data & .bss



- Tipo di dato scalare \rightarrow possibile utilizzo operatori $+$ e $-$
 - Ed equivalenti ($++$, $--$, $+=$ ecc.)
- Risultato dipendente da tipo di dato puntato
- Sommare x non significa incrementare indirizzo di x byte
 - Ma di x “posizioni”

- Se il puntatore punta al tipo di dato `<type>` sommare `x` equivale a sommare `x` volte i byte occupati da `<type>`
- In pratica si usa solo se puntatore contiene indirizzo array
- Di fatto implicita nella sintassi `x[y]`
 - $x[y] \rightarrow *(x+y)$

- Come per array facile commettere errori
 - Dimensioni buffer
 - Durata variabili
 - Perdita informazioni su aree allocate



- Come per gli array rischio di andare oltre i limiti di quanto allocato
- Inoltre → puntatore non inizializzato o NULL



- malloc()/calloc() restituiscono indirizzo di quanto allocato
 - Serve per usare area memoria
- E se perdo tale indirizzo?
 - Memoria diventa inutilizzabile
 - Ma comunque “in uso”
- Cause
 - Errori Assegnazione
 - Durate Automatiche



- È il duale del precedente
- Non perdo l'informazione di dove si trova il buffer ma...
 - Il buffer non è più valido
- Cause
 - Durata automatica
 - Errori uso free()





UNIVERSITÀ DI PARMA

Allocazione Dinamica della Memoria e Puntatori



Ma noi paghiamo i byte? No, e allora...

Z, VisLab srl, 2014