



UNIVERSITÀ DI PARMA

Tipi di dato in C

*What's in a name? That which we call a rose,
By any other name would smell as sweet.*

William Shakespeare, Romeo and Juliet,
Act II, Scene II)



- Definizione variabile
- Tipi di dato in C
- Dati scalari
 - Virgola mobile
 - Interi
- Combinazione differenti tipi di dato
 - Conversioni
- Campo di visibilità delle variabili



- Qualcosa che “varia”...
- Definizione:
 - Spazio in memoria dotato di nome in cui è possibile scrivere e leggere dati durante l'esecuzione del programma
- Spazio in memoria → indirizzo e dimensione
 - Gestiti automaticamente in linguaggi di alto livello

- Contenuto ovvero valore di una variabile → right value o rvalue
- Indirizzo in memoria → left value o lvalue
- Spazio occupato

```
int a = 173;
```

- Il nome di una variabile o identificatore può essere costituito da lettere, cifre o underscore “_”
- il primo carattere **deve** essere una lettera
- lettere minuscole o maiuscole sono differenti
- non può essere una parola chiave

- Alcune convenzioni:
 - il nome di una variabile è normalmente minuscolo
 - il nome deve essere mnemonico
 - quando il nome è formato da più parole è opportuno usare l'underscore o alcune lettere maiuscole

```
int somma          = 173;  
int numero_studenti = 144;  
int NumeroAula      = 10;
```

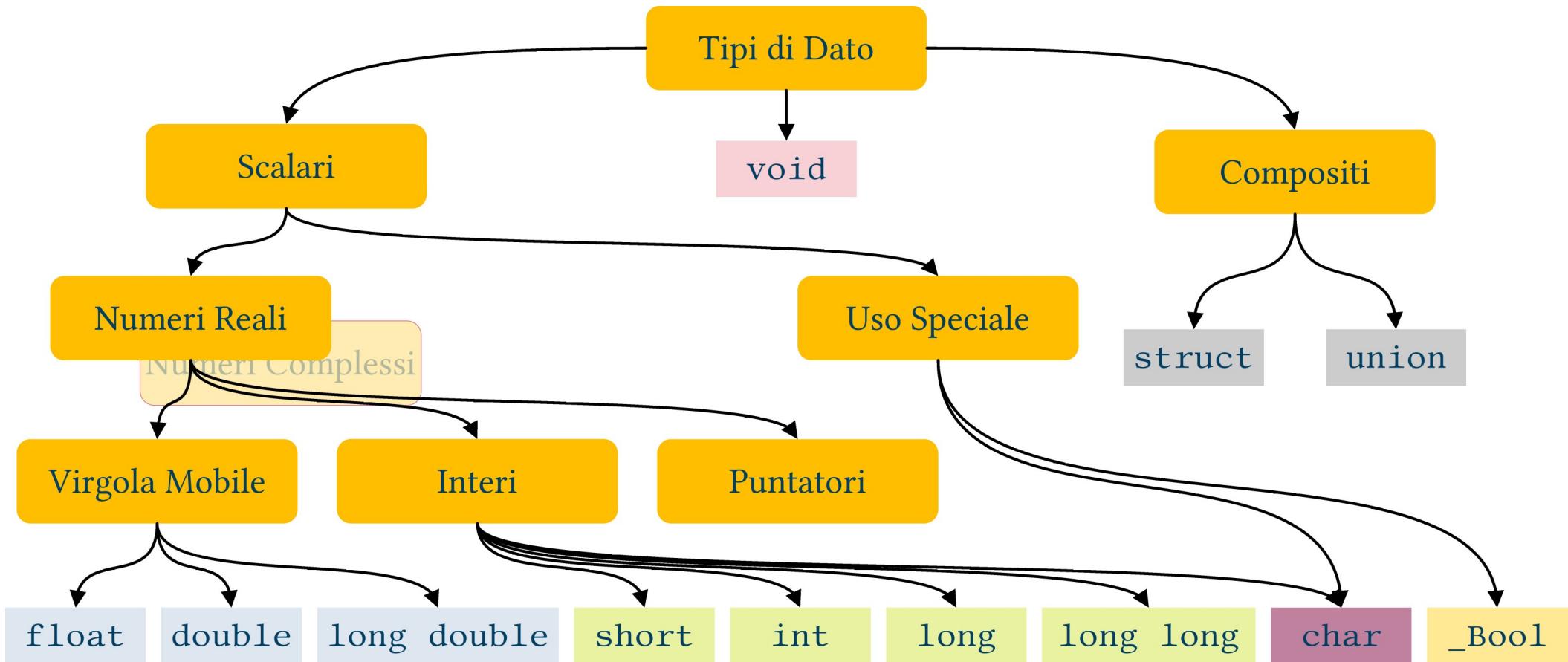
Definizione di una variabile

```
// le variabili si definiscono come
// <tipo> <nome_variabile>

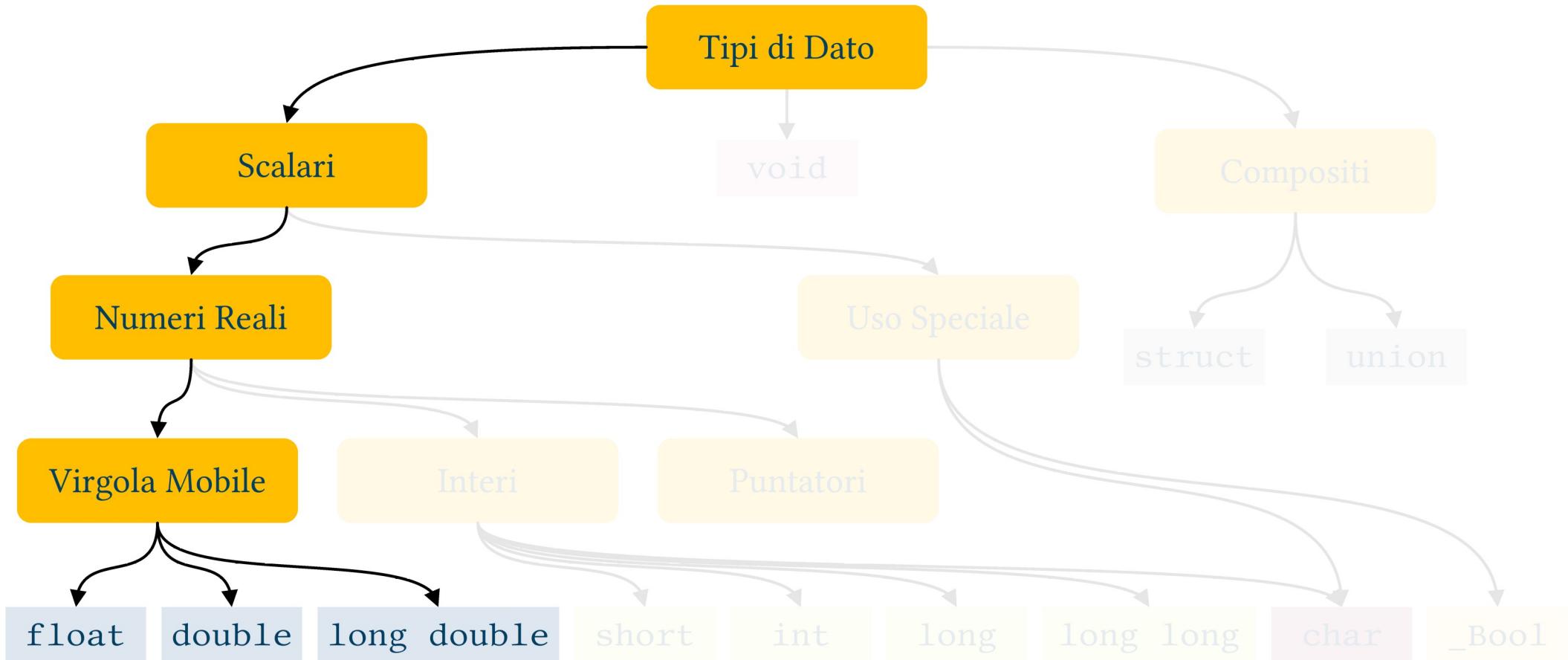
int a;
    // definisco variabile di tipo "int"
    // a contiene valore casuale
    // in C la definizione NON inizializza

int b=17;
    // definisco E inizializzo variabile di tipo "int"
```

C99 Albero dei Tipi di Dato



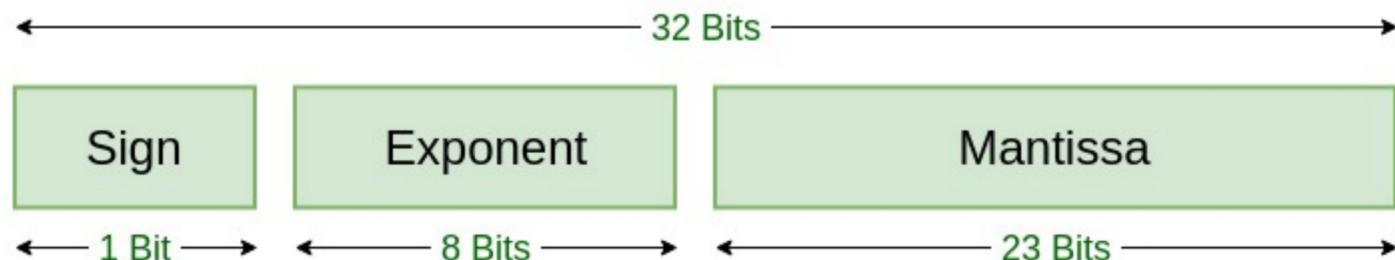
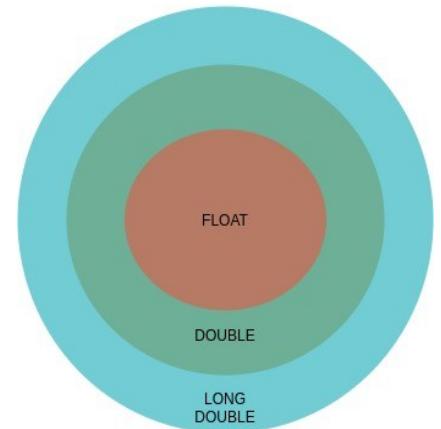
C99 Albero dei Tipi di Dato



- Rappresentazione approssimata dei numeri reali
 - Positivi e negativi con e senza decimali
- Perchè approssimata?
 - Massimo e minimo valore rappresentabile
 - Limite al numero di cifre decimali

Tipi di dato a virgola mobile

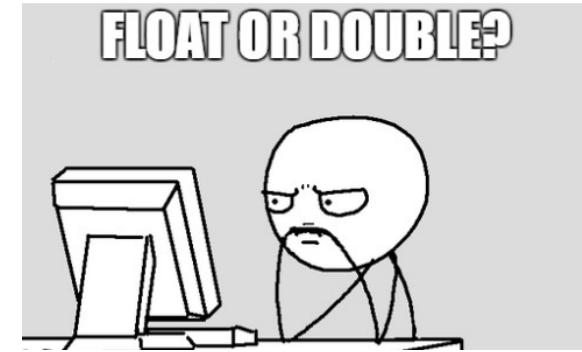
- Tre tipi
 - float → precisione singola
 - double → precisione “doppia”
 - long double → precisione estesa
- Cosa cambia?
 - numero di byte utilizzati



Float vs Double vs Long Double



- Quale tipo scelgo?
- Come sempre compromesso:
 - Precisione
 - Velocità esecuzione
 - Occupazione memoria



- Il C stabilisce di usare lo standard IEEE 754?
 - No e non è importante
- Il C fissa quanti bit vengono usati?
 - No
- Dipende tutto dal compilatore
 - In alcuni casi non vi è differenza tra float e double

- Il C fornisce un sistema per capire quanto spazio occupa una variabile
- `sizeof(<tipo di dato>)`
 - Restituisce il numero di byte occupati da quel tipo di dato
 - Tecnicamente è una macro
- Per stampare prefisso ‘z’ per gli specificatori di formato
 - `%zd` C99
 - `%ld` C89

- Come li stampiamo o leggiamo?

- Formati

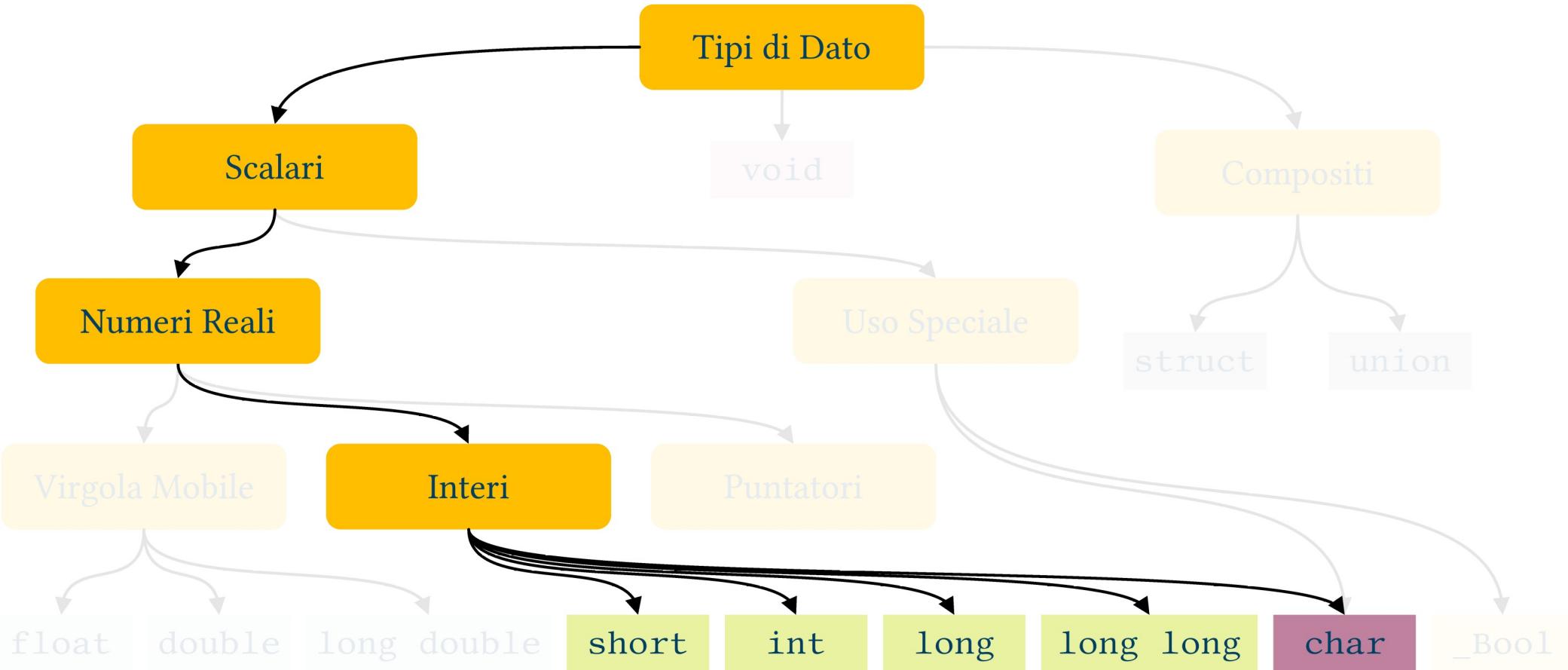
- %f → formato decimale 123.456000
 - %e → formato scientifico 1.23456e+02
 - %E → formato scientifico 1.23456E+02
 - %g → rappresentazione più breve tra %f e %e 123.456

- Suffissi per tipo

- “niente” → float
 - l → double (obbligatorio solo per input)
 - L → long double

- Valori particolari
- $\pm\infty \rightarrow$ Infinite
 - Può essere il risultato di alcune operazioni
 - Esempio: divisione per zero
- NaN \rightarrow not a number
 - Di fatto non capita se la variabile è inizializzata correttamente
- Funzioni per controllo
 - isnan(), isinf()...

C99 Albero dei Tipi di Dato



- Quattro+1 tipi:
 - `char` → in realtà si usa raramente per “numeri”
 - `short int`
 - `int` → il piú usato
 - `long int`
 - `long long int`
- Anche qui cambia dimensione memoria occupata
 - Dati n bit riesco a rappresentare numeri non negativi tra 0 e $2^n - 1$

- Complicazione: il segno!
 - Cambia l'intervallo possibile (da $0-2^n$ a $-2^{n-1}-2^{n-1}-1$)
- Sono tutti con segno ad eccezione di...
 - char
 - Che a seconda del compilatore può o meno avere segno
- Nel definirli posso cambiare comportamento
 - unsigned
 - (signed)
- I tipi possibili diventano quindi 10!

Tipi di dato interi (esempio 64 bit)



Type	Byte	Smallest Value	Largest Value
signed char	1	-128	127
unsigned char	1	0	255
short	2	-32.768	32.767
unsigned short	2	0	65.535
int	4	-2.147.483.648	2.147.483.647
unsigned int	4	0	4.294.967.295
long	8	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
unsigned long	8	0	18.446.744.073.709.551.615
long long	8	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
unsigned long long	8	0	18.446.744.073.709.551.615

- Come li stampiamo o leggiamo?

- Formati

• %d	→ formato decimale	-45
• %x	→ formato esadecimale	fffffd3
• %X	→ formato esadecimale	FFFFFD3
• %u	→ formato decimale senza segno	4294967251

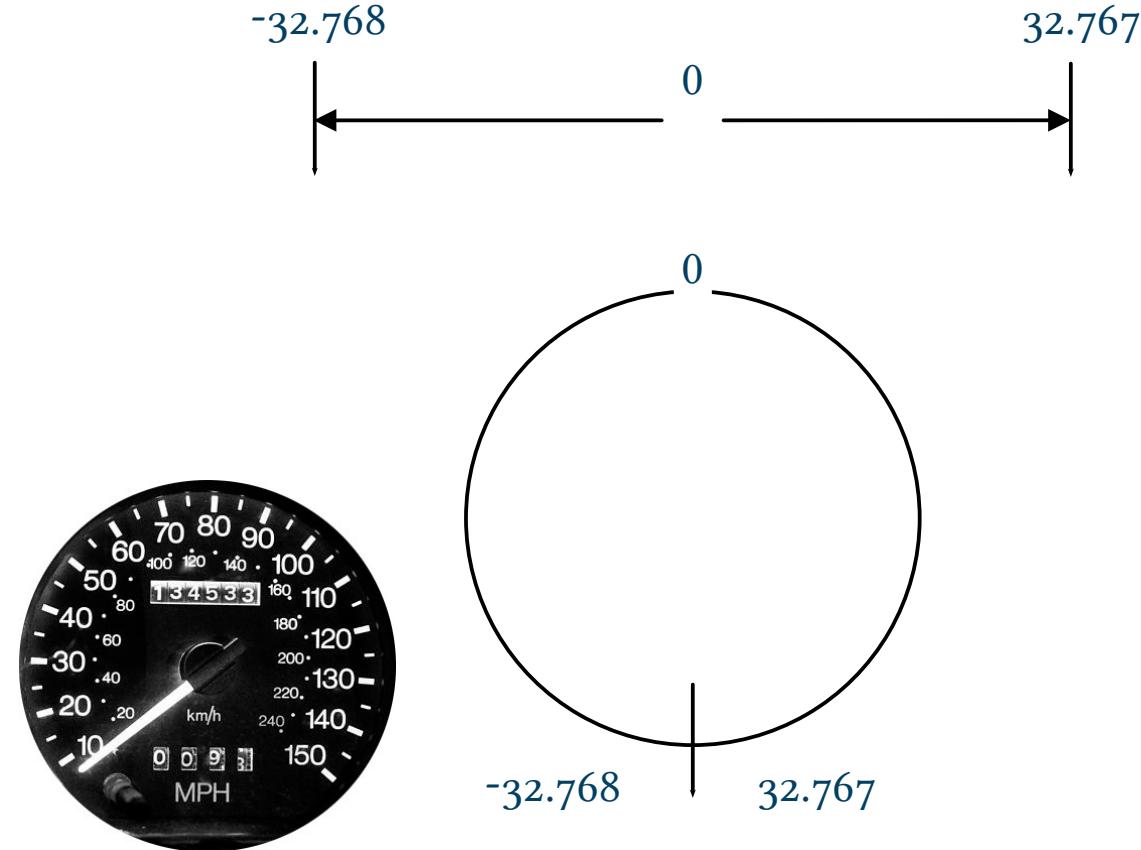
- Suffissi per tipo

• “niente”	→	int
• hh	→	char
• h	→	short
• l	→	long
• ll	→	long long

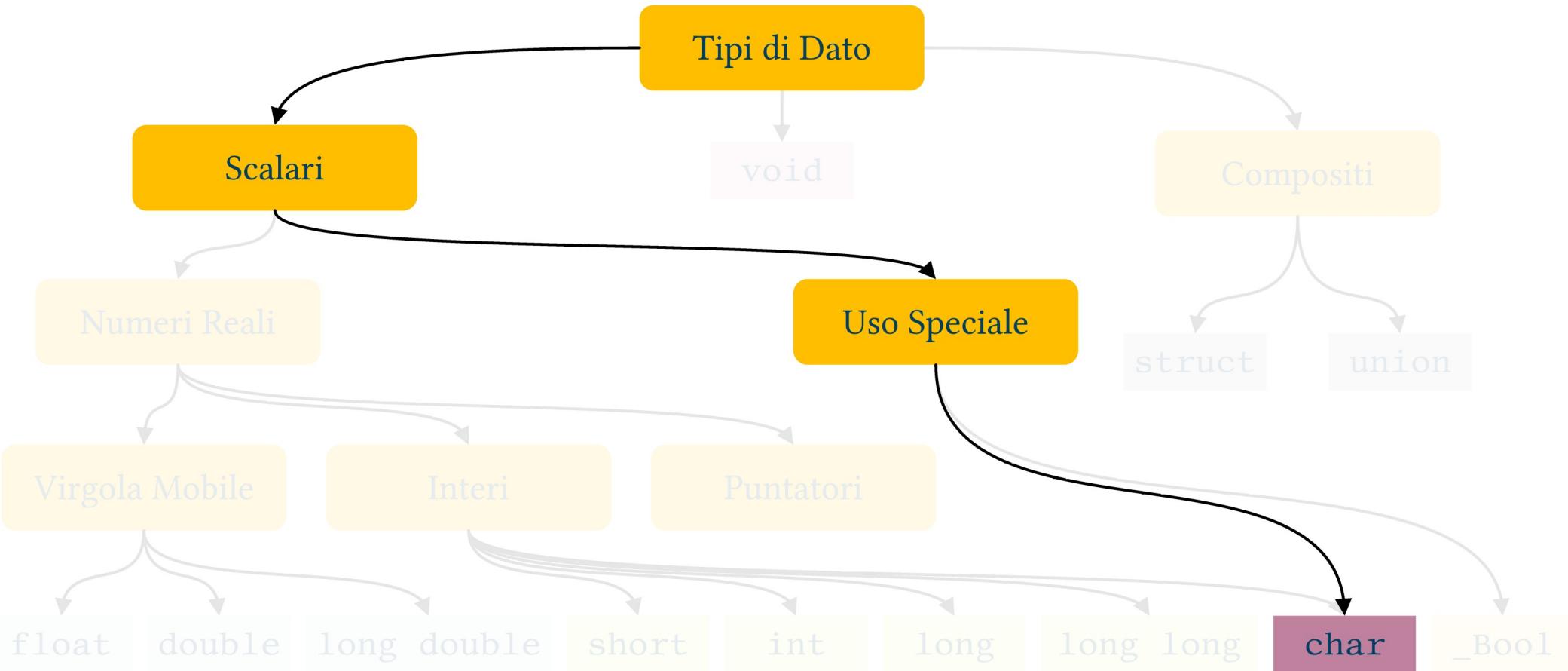
Tipi di dato interi



- Overflow
 - Avviene quando vado oltre i limiti possibili
 - Non viene fornito errore!
 - Perdo riporto o prestito



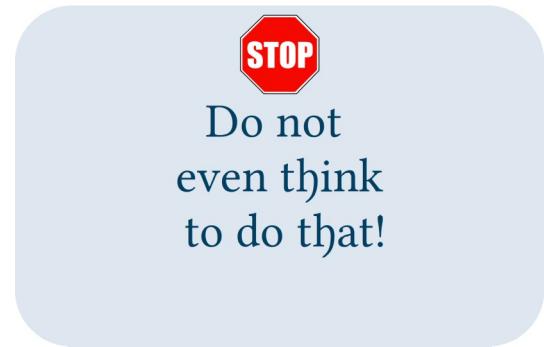
C99 Albero dei Tipi di Dato



Il tipo char



- È un tipo numerico intero a tutti gli effetti
 - 0–255
 - -128–127
- Di fatto usato solo per codice ASCII
- Specificatore di formato
 - %c
- Costanti:
 - Numero diretto, esempio 65
 - Indicazione simbolo tramite apici singoli, esempio 'A'



Il tipo char e scanf()



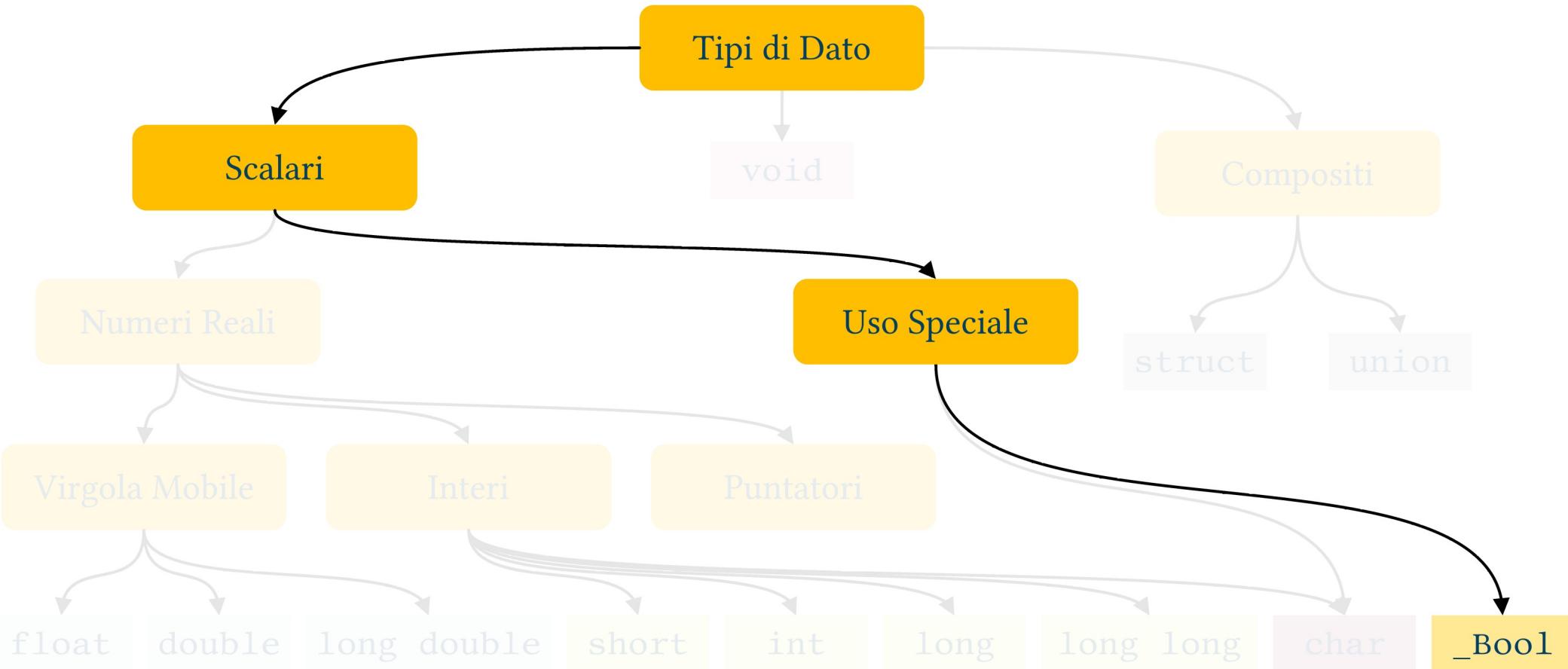
- Leggere da tastiera i char (e derivati) richiede qualche cautela
- Tutto ciò che viene premuto finisce in input al programma
 - Incluso, ad esempio, eventuali caratteri di invio
 - Buffer di input



```
fscanf("%c", &x); // leggo eventuale invio di inserimento precedente
```

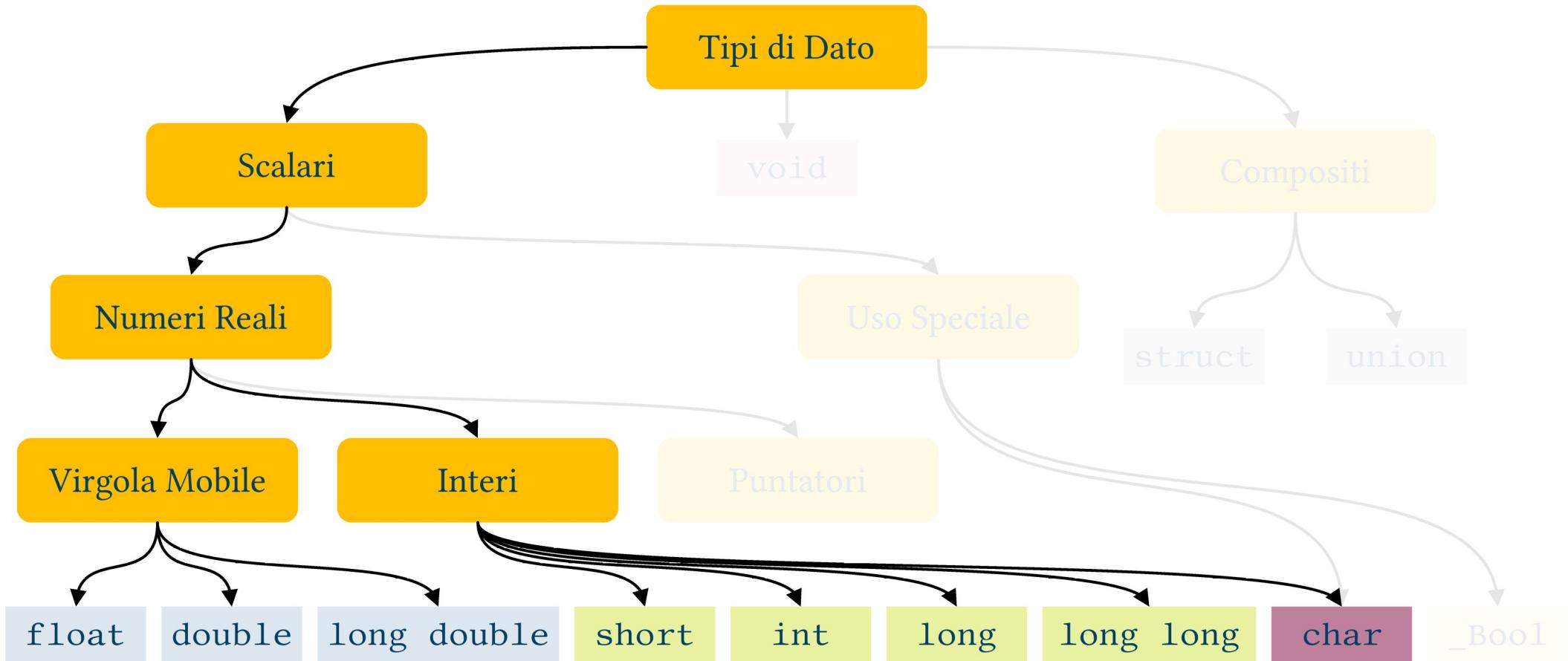
```
Fscanf(" %c", &x); // salto eventuali spaziature ('invii' inclusi!)
```

C99 Albero dei Tipi di Dato



- Specifico del C99
- Tipo di dato intero senza segno pensato per memorizzare solo “vero” e “falso”
 - 0 → falso
 - 1 → vero
- Quindi assume solo valori 1 o 0

C99 Albero dei Tipi di Dato



Combinazione di tipi differenti

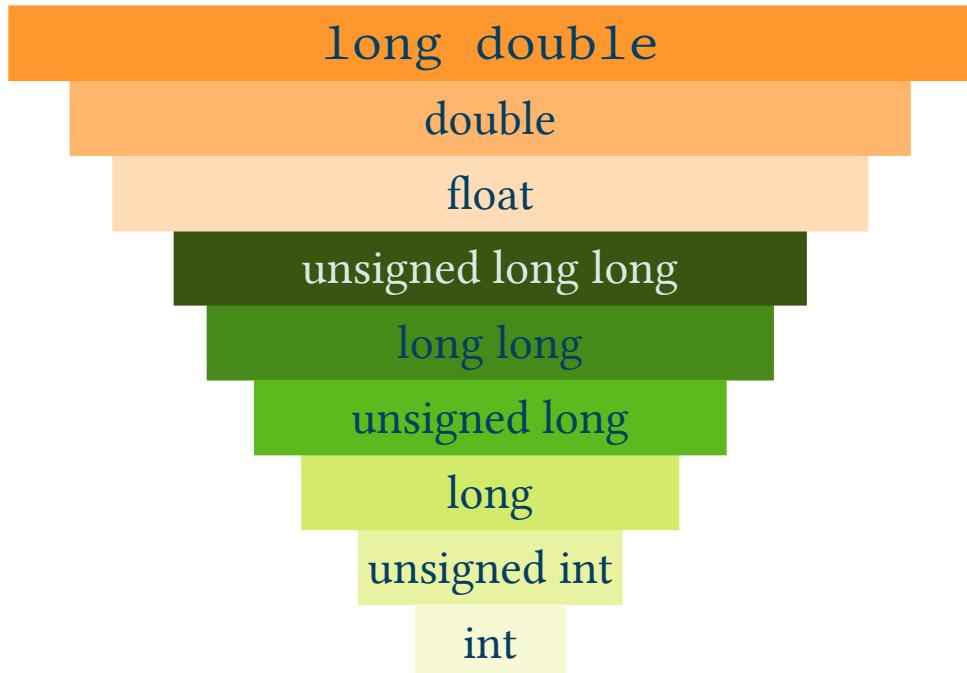
- 3 tipi a virgola mobile 4–5 tipi interi
- Come si comporta il C quando li combiniamo?
- Conversioni
 - Assegnamento
 - Implicite o automatiche

- Quando combino tipi differenti in espressioni
- Operandi binari richiedono operatori dello stesso tipo
 - Aritmetici: *, /, %, +, -; confronti: <, >, <=, >=, ==, !=; bit a bit: &, ^, |,
- Esempio: $a+b$ se a è int e b è float che tipo di risultato fornisce?
 - Spoiler: float
- Regole di conversione
- Alta probabilità di errore!

Regole conversioni implicite



- Promozione implicita a int
 - `_Bool`, `char` e `short` → int
 - Oppure `unsigned int`
 - Anche se operandi stesso tipo
- Piramide conversioni





- Casting
- Decido io la conversione
 - Meno facile sbagliare
- Formato:

(tipo) espressione



- Definita una variabile posso usarla ovunque?
 - No!
- Ambito di visibilità di una variabile o *scope*
 - Parte di codice in cui quella variabile è usabile
- 3 Casi:
 - Programma (variabili globali)
 - Singolo file
 - Blocco (variabili locali)





- Definite al di fuori di qualunque blocco
 - {} oppure definizione in for(;;)
- Posso modificarle in qualunque punto del codice
 - Dati NON protetti
 - Difficoltà debug
 - Incertezza
- Il voto di esame ne risente...
- Unico vantaggio → inizializzazione



- Definite
 - Interno blocchi ovvero dentro un {}
 - In un for(;;)
 - Argomenti funzioni
- Usabili solo dove definite
- Durata automatica
 - Create quando il flusso di esecuzione arriva in quel punto
 - Distrutte quando l'esecuzione esce da quella parte di codice

- Variabili locali
 - Focus su parti di codice piccole
 - E autosufficienti!
 - Codice tipicamente più comprensibile e gestibile
 - Modularità
 - Minori problemi di portabilità
- Variabili globali
 - Devo tener traccia di tutti i punti del codice dove sono usate/modificate
 - Se lavoro di gruppo devo coordinare gestione
 - Memoria sempre in uso!
 - Conflitti nomi



- Una variabile non detto che sia per sempre
- Durata fissa (*static lifetime*):
 - Variabili globali
 - Memoria occupata ad inizio esecuzione e mai liberata
 - In questo caso inizializzazione a 0
- Durata automatica (*automatic lifetime*)
 - Variabili locali
 - La variabile viene creata ogni volta si accede a quella parte di codice
 - Distrutta ogni volta il flusso di esecuzione esce da quel blocco
 - Posso cambiare la durata? Sì → **static**



- Ci sono situazioni in cui si usano valori costanti
- Soluzione quick ma molto dirt:
 - Riempo il programma di “numeri magici”
 - Modifiche difficili, codice non parametrico
- Soluzione corretta:
 - Definisco costanti
 - Questo permette di ottenere codice parametrico e facilmente modificabile

- In C tre soluzioni per definire una costante:
 - `#define`
 - Esempio: `#define N_MAX_STUDENTI (100)`
 - `const`
 - Stessa sintassi variabili, obbligatoria inizializzazione
 - Sensata visibilità globale
 - Esempio: `const int N_MAX_STUDENTI=100;`



UNIVERSITÀ DI PARMA

Tipi di dato in C



*What's in a name? That which we call a rose,
By any other name would smell as sweet.*

William Shakespeare, Romeo and Juliet,
Act II, Scene II)