



UNIVERSITÀ DI PARMA

# Breve introduzione al C

*For 'tis the sport to have the engineer  
Hoist with his own petar*

Hamlet, Act III, Scene IV, Shakespear



- Perché il C?
  - Come e quando è nato il C
  - Principali caratteristiche
- Scheletro di un programma C
- Introduzione agli elementi base
  - Preprocessore
  - Variabili
  - Espressioni
  - Funzioni
  - I/O → Input/Output



- Perché il C?
  - Non nasconde la macchina
    - basso/medio livello
  - Efficiente
  - Consolidato
  - Utilizzato per: driver, programmi semplici, sistemi operativi, sistemi embedded
- A partire dal C sviluppati altri linguaggi e/o sintassi molto simili
  - C++, JAVA, C#, (PERL)



- Articolo di Joel Spolsky, 2005
  - Stackoverflow

*And thus, the ability to understand **pointers** and **recursion** is directly correlated with the ability to be a great programmer.*

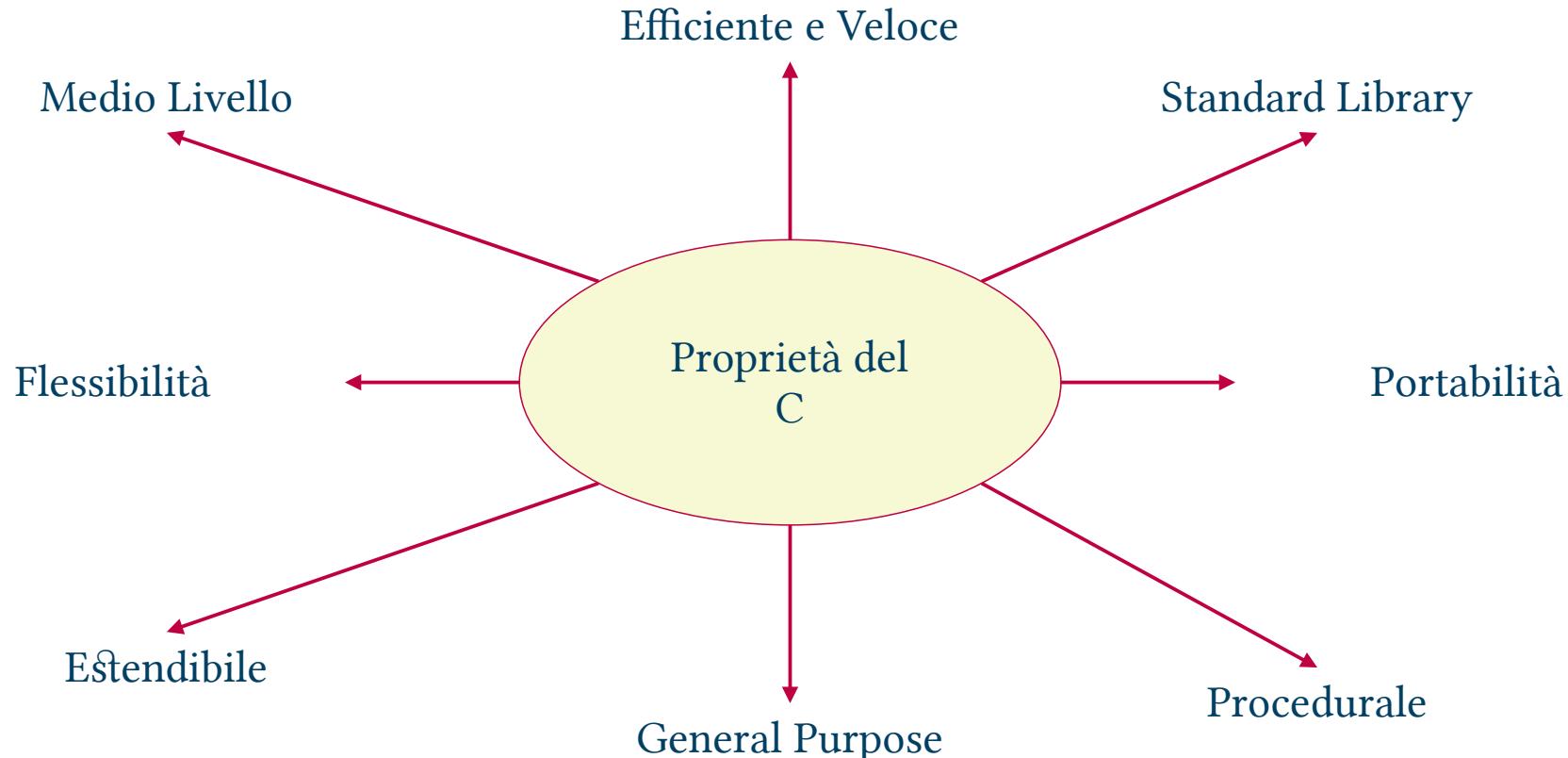
*Nothing about an all-Java CS degree really weeds out the students who lack the mental agility to deal with these concepts. As an employer, I've seen that the 100% Java schools have started churning out quite a few CS graduates who are simply **not smart enough to work as programmers***



- Laboratori Bell
  - UNIX
  - DEC PDP-7 con ben 8 kByte di RAM
- Inizialmente sviluppo in Assembly
- Linguaggio ad alto livello → B derivato dal BCPL
- Dennis Ritchie sviluppa il C a partire dal B
  - Inizialmente NB (New B)
- 1973 UNIX riscritto in C

- Differenti standard
  - K&R (da evitare)
  - ANSI C o ISO C:
    - C89/C90 largo supporto
    - C99 diffusione recente, si sta sempre di più affermando
    - C11 aka “do not use VLA”
    - C18/17 mostly C11 bugfixes
    - C2x (2023)
- Code::Blocks > C89/C90

# Caratteristiche principali





- “Trust the Programmer” → i programmatore sono infidi!
- Codice difficile da interpretare
  - E quindi modificare
- Procedurale ↔ OOP
- No gestione eccezioni
- No namespace
- No run time checking
- No garbage collection



```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char **argv){
    /* stampa Hello World! A schermo */
    printf("Hello World!");
    return 0;
}
```



```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char *argv) {
    /* stampa Hello World */
    printf("Hello World\n");
    return 0;
}
```

- Tutto quanto inizia con #
  - Direttiva preprocessore
- Nel caso degli “include<>” serve per inserire informazioni necessarie al compilatore
  - File .h
  - Librerie
- A rigore solo stdio.h è necessaria



```
#include<stdio.h>
#include<stdlib.h>
```

- Funzione “main()”
- Punto di inizio dell'esecuzione
  - Si parte da qui!
- Capiremo più avanti i singoli elementi

```
int main(int argc, char **argv){  
    /* stampa Hello World! A schermo */  
    printf("Hello World!");  
    return 0;  
}
```



```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char **argv){
    /* stampa Hello World! A schermo */
    printf("Hello World!");
    return 0;
}
```

- Commento
- Tutto ciò tra /\* e \*/
  - Altri modi
- Ignorato dal compilatore
- Ma importante per chi deve capire il codice!



```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char **argv)
    /* stampa Hello World! A schermo */
    printf("Hello World!");
    return 0;
}
```

- Stampa “Hello World!”
  - Senza doppie virgolette
- printf() non è una istruzione del C ma una funzione
  - Come la main()
- La libreria del C ne fornisce diverse



```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char **argv){
    /* stampa Hello World! A schermo */
    printf("Hello World!");

    return 0;
}
```

- Termina la main()
- E quindi il programma
- Restituisce “0”
- A chi?
  - Al Sistema Operativo



```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char **argv) {
    /* stampa Hello World! A schermo */
    printf("Hello World!");
    return 0;
}
```

- Syntactic Syrup
- Addolciscono la sintassi per noi
- Comunque obbligatorie

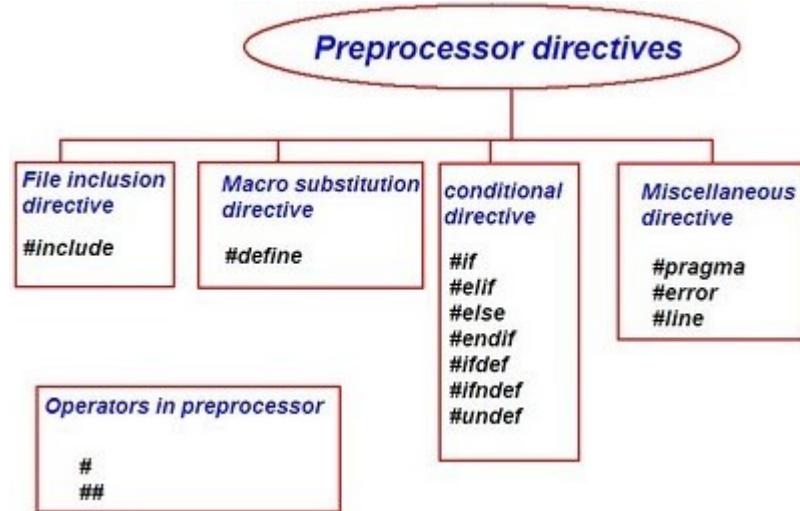
- Indentazione
- Rende il codice più leggibile
- Obbligatoria per noi (-2 punti)

# Introduzione agli elementi base del C

- Direttive preprocessore
- Commenti
- Variabili
- Stringhe
- Nominativi
- Funzioni
- Espressioni
- I/O



- Tutte quelle che iniziano con ‘#’
  - Ce ne sono una dozzina
  - Ma noi ne vedremo solo alcune
- Per ora limitiamoci a:
  - `#include <??>` oppure `#include "???"`
    - Permette di includere file necessari al compilatore
  - `#define <identificativo> <rimpiazzo>`
    - Per costanti e macro

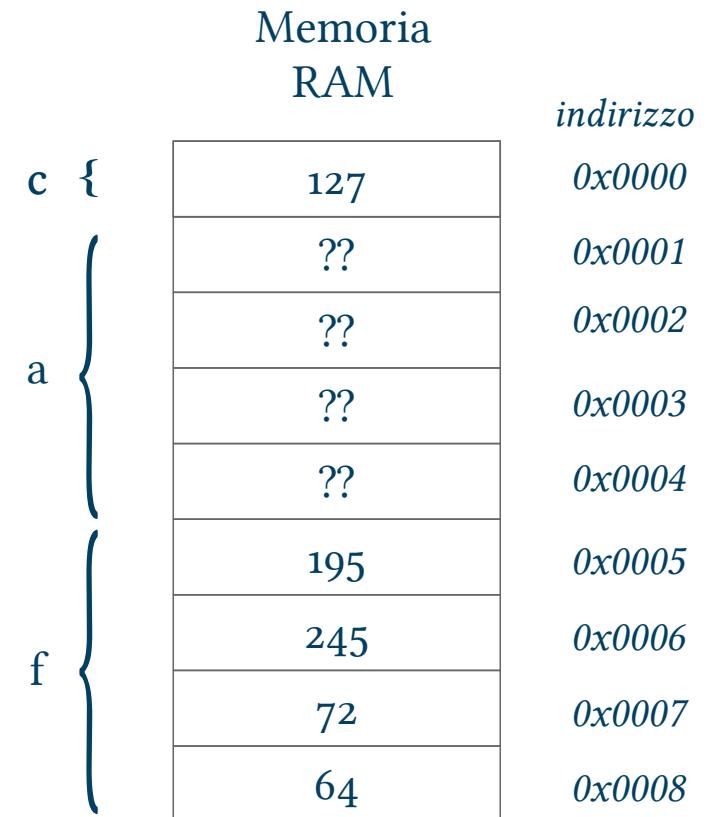


- Spesso sottovalutati
  - Ma in sede di esame ne tengo conto
- Multiriga:
  - Tutto ciò che si trova tra /\* e \*/
- Singola riga
  - Tutto ciò che segue //
  - Nello standard dal C99

- Nel modello di von Neuman → memoria
  - Celle con indirizzo
  - Scomodo ad alto livello
- Il C può accedere alla memoria tramite le variabili
  - Sistema per memorizzare dati
  - In pratica parte di memoria a cui associo un nome
    - Non mi devo preoccupare di indirizzo o spazio occupato
    - Ci pensa il compilatore

# Introduzione alle variabili

- Differenti tipi
  - Numeriche ovvero Scalari
    - Interi, virgola mobile...
  - Composite
  - ...
- In C devo sempre indicare un tipo di dato
- Esempio di definizione di variabili:
  - `char c =127`
  - `int a;`
  - `float f=159.37;`



# Introduzione alle stringhe

- In C particolare tipo di dato
  - Sequenza di caratteri racchiusi tra “”
  - `printf("Hello world!");`
  - `char c[]="Informatica";`
- Approfondiremo meglio
- Libreria apposita
  - `#include<string.h>`
- Stringhe di formato

- Programmando sceglieremo dei nomi per variabili, costanti, funzioni e altri elementi
- Ci sono però regole
  - Solo lettere, cifre o “\_”
  - Deve iniziare per lettera
  - Maiuscole e Minuscole sono considerate lettere differenti
  - Non deve essere una parola riservata del linguaggio

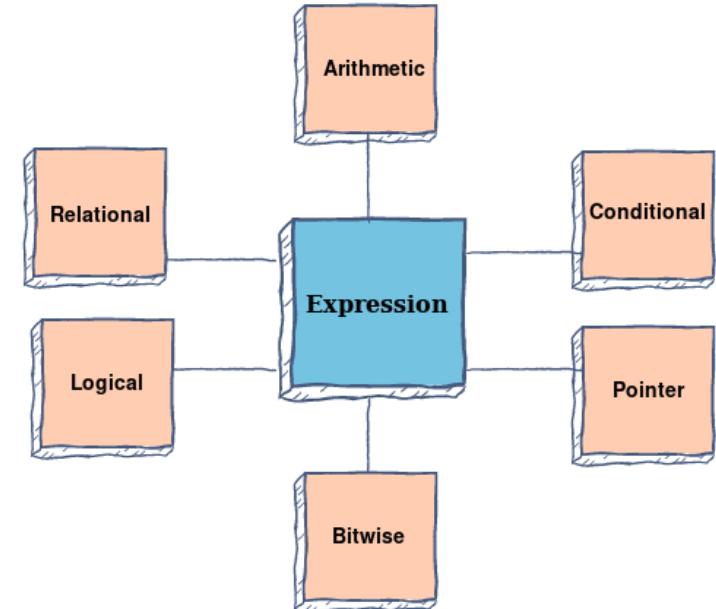
# Introduzione agli identificativi

<i>auto</i>	enum	<i>restrict</i>	unsigned
break	extern	return	void
case	float	short	volatile
char	for	signed	while
const	goto	sizeof	<i>_Bool</i>
continue	if	static	<i>_Complex</i>
default	<i>inline</i>	struct	<i>_Imaginary</i>
do	int	switch	<i>_Packed</i>
double	long	typedef	
else	register	union	

# Introduzione alle espressioni



- Combinazione di 1 o più “operandi”
  - Variabili, costanti, chiamate di funzioni ecc.
- Esempi
  - 5
  - j (se j è una variabile)
  - printf(“ciao”)
  - $5+j^*6$ 
    - +, -, /, \*. %



- Input/Output
- Noi ci limiteremo inizialmente a
  - Input → Tastiera
  - Output → Video
- Due funzioni principali
  - printf()
  - scanf()

- È una funzione
  - Complessa
- Print Formatted
  - Stampa in accordo a una “stringa di formato”
    - Testo
    - Variabili
    - Simboli
    - Ecc.

```
int printf(const char *format, ...);
```

- Stringa di formato
- Può contenere
  - Testo → stampato così come è
  - Combinazioni di caratteri → rimpiazzate da altro quando stampo
    - Sequenze di escape
    - Specificatori di formato
      - Caratteri di controllo

- Stampare testo “semplice”
- Racchiuso tra doppi apici “”

```
printf("Hello world!");
```

- Caratteri non ottenibili da tastiera o per uso specifico
- Tipicamente carattere preceduto da “\”

```
printf("Hello world! \n");
```

```
printf("Hello world!
"); // error: missing terminating "
character
```

# Sequenze di “escape”



<b>\n</b>	newline
<b>\r</b>	carriage return
<b>\f</b>	form feed
<b>\a</b>	bell
<b>\t</b>	horizontal tab
<b>\v</b>	vertical tab
<b>\”</b>	doppie virgolette
<b>\nnn</b>	carattere corrispondente all'ASCII nnn (ottale)
<b>\xNN</b>	carattere corrispondente all'ASCII NN (esadecimale)
<b>\\"</b>	backslash
<b>%%</b>	singolo %

- Indicano a printf() come stampare
  - Valori numerici
  - Contenuto variabili
  - ...
- Preceduti da “%”

```
printf("11x67 is equal to %d\n", 11*67);
```

# Specificatori di formato



%d	intero decimale
%o %x %X	intero ottale o esadecimale
%f	numero a virgola mobile in notazione decimale
%e %E	numero a virgola mobile in notazione scientifica
%g %G	numero a virgola mobile (automatico)
%c	singolo carattere
%s	stringa
%p	indirizzo di memoria (void *)
%q %L	long long
%[ ]	elenco di caratteri, solo <b>scanf()</b>

- Gli specificatori di formato indicano cosa stampare
- I caratteri di controllo permettono di fare “fine tuning”
  - Stampo comunque il segno?
  - Quanti caratteri devo usare?
  - Quante cifre dopo la virgola?
  - ...

```
printf("1 g di oro vale %f al grammo", 1770.77/31.103);
```

```
printf("1 g di oro vale %.2f al grammo", 1770.77/31.103);
```

<b>+</b>	stampa comunque il segno
<b>spazio</b>	antepone a numeri positivi uno spazio
<b>n</b>	minima ampiezza campo (riempimento con spazi)
<b>0n</b>	minima ampiezza campo (riempimento con 0)
<b>.n</b>	numero di cifre decimali (minimo o fisso)
<b>-</b>	allineamento a sinistra
<b>l</b>	long
<b>h</b>	short
<b>ll</b>	long long

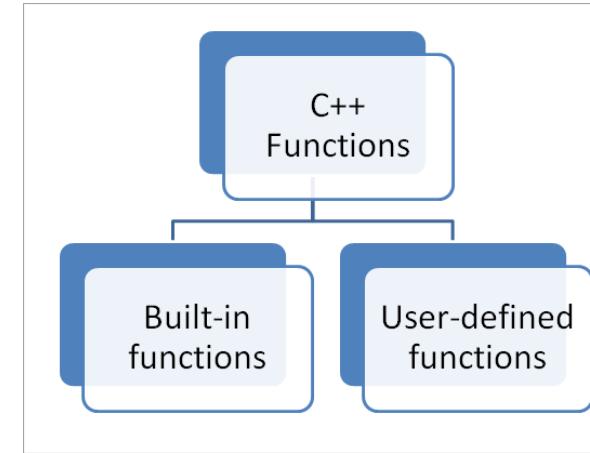
- È una funzione
  - `int scanf(const char *format, ...);`
- Mi permette di leggere da tastiera
  - Stringa di formato → pattern matching
  - Ciò che legge → variabile (ovvero memoria)

```
int a;  
scanf(“%d”, &a);
```

- Simile a printf() ma:
  - Stringa di formato tipicamente solo specificatore di formato
  - I dati coinvolti vanno preceduti da &
- Occhio agli spazi e al buffer di ingresso!

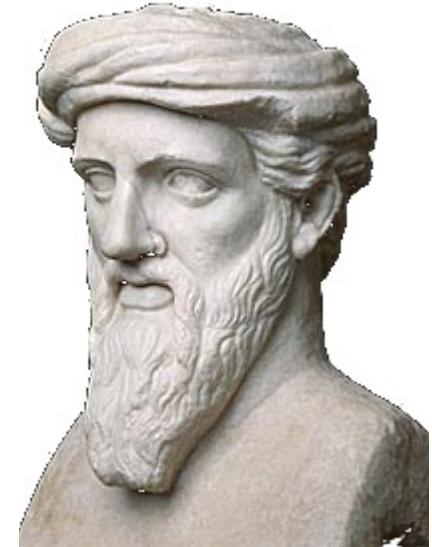


- Anche “procedure” o “subroutine”
- Istruzioni C “troppo” semplici
- Tante anche per fare cose banali
  - Soluzione → creo “gruppi” di istruzioni
- Concetto base di molti linguaggi ad alto livello
  - “Building blocks”
  - Programmazione procedurale





- Funzioni predefinite
  - Esempio: printf()
- Il C mette a disposizione una “ricca” libreria di funzioni predefinite
  - Non devo reinventare la ruota!
  - Sono già ottimizzate
  - Funzionano
  - Portabili

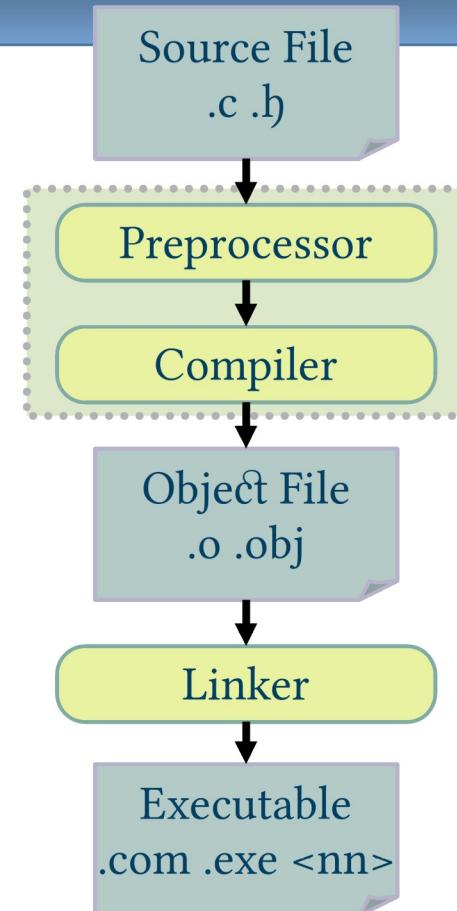


```
double hypot(double x, double y);
```

- Funzioni definite dal programmatore
- Fattorizzazione
  - le definisco → 1 volta
  - le uso → quante volte voglio
- Miglior leggibilità
- Miglior facilità di organizzazione e debugging

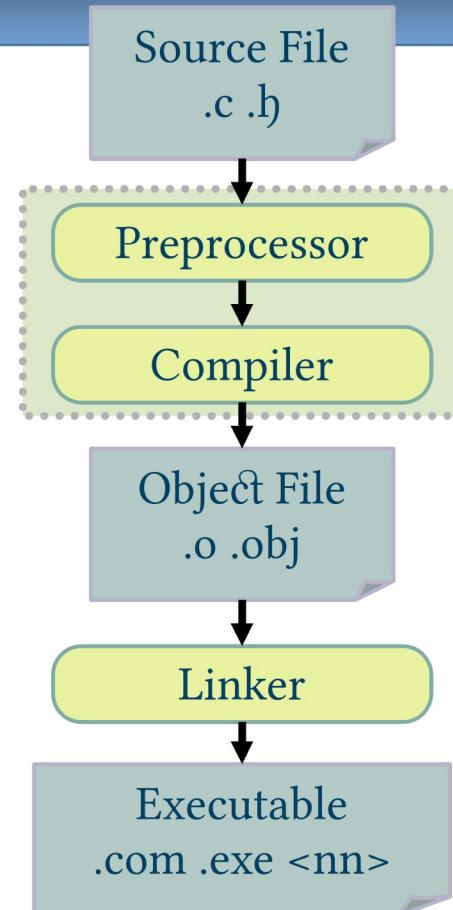
# Compilazione

- Sorgente → Codice Macchina
- Noi useremo “ambiente integrato”
  - Nasconde il processo
- In realtà differenti “attori”
  - Preprocessore
  - Compilatore
  - Linker



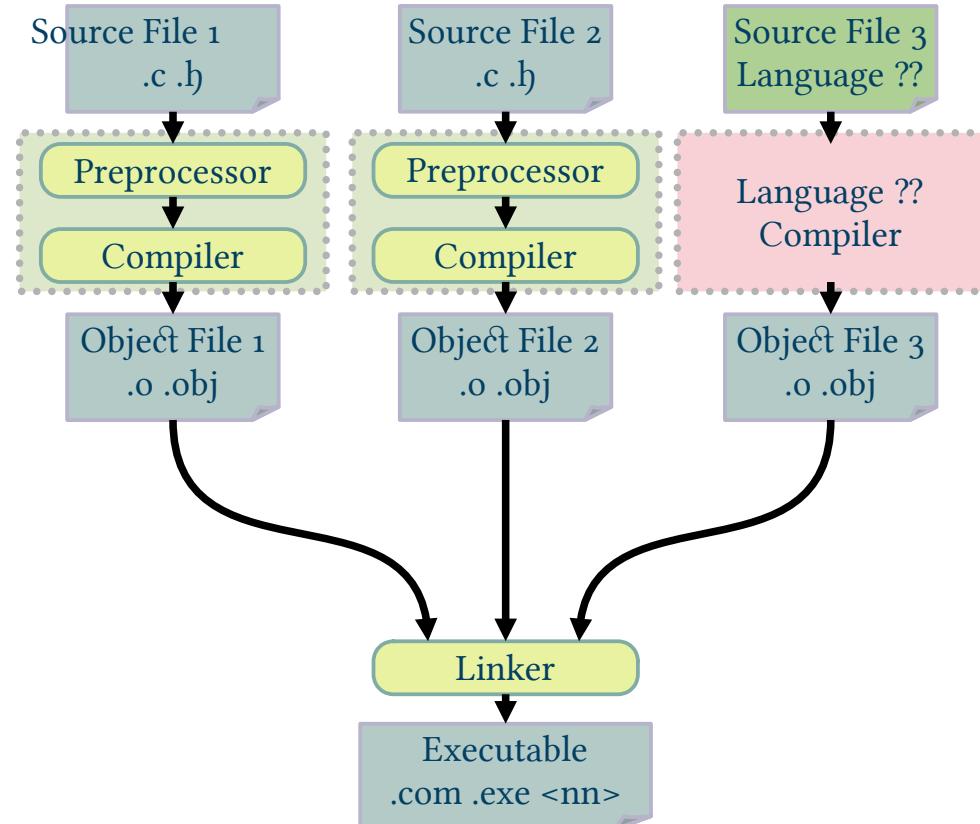
# Fasi di compilazione

- Preprocessore
  - Mera sostituzione direttive che iniziano con ‘#’
  - Spesso integrato con compilatore
- Compilatore
  - Traduce il sorgente ottenendo codice macchina
    - Operazione complessa
  - Ottimizzazione
  - Supporto differenti architetture





- **Linker**
  - Riunisce tutti i file oggetto
  - Risoluzione indirizzi memoria
  - Necessario per progetti multifile
  - Può gestire sistemi con più linguaggi
    - Tutti “compilati”





UNIVERSITÀ DI PARMA

# Breve introduzione al C



*For 'tis the sport to have the engineer  
Hoist with his own petar*

Hamlet, Act III, Scene IV, Shakespear