

# Introduzione alla programmazione in linguaggio C

Marco Alberti

Dipartimento di Matematica e Informatica



**Università  
degli Studi  
di Ferrara**

Programmazione e Laboratorio, A.A. 2020-2021

Ultima modifica: 25 settembre 2020

Attenzione! Questo materiale didattico è per uso personale dello studente ed è coperto da copyright.  
Ne sono vietati la riproduzione e il riutilizzo anche parziale, ai sensi e per gli effetti della legge sul diritto d'autore.

# Sommario

- 1 Sintassi
- 2 Semantica
- 3 Espressioni intere
- 4 Priorità ed associatività
- 5 Flusso

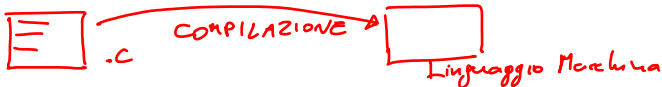
## Editor (di testo)

- Un **editor** è qualsiasi programma in grado di creare e modificare file di testo (in formato ASCII o nelle sue estensioni, come ISO-8859-1 e UTF-8), quindi adatto a scrivere programmi in quasi tutti i linguaggi di programmazione
- Alcuni editor supportano particolarmente bene la scrittura di file sorgenti C e l'integrazione con gli strumenti di programmazione (compilatore, debugger, version control): Emacs, Vim, Sublime Text *80\$*
- Noi useremo Visual Studio Code (<https://code.visualstudio.com>)
- Estensione (plugin) per lo sviluppo in C documentata qui:  
<https://code.visualstudio.com/docs/languages/cpp>.

## 030\_introduzione\_al\_c/helloWorld.c

```
1 #include <stdio.h>
2 main() {
3     printf("Hello, World!\n");
4 }
```

- 1 Utilizzando il terminale, creare, se non esiste, la cartella /home/studente/~~home~~\_\_matricola\_\_ (ad esempio, se il proprio numero di matricola è 123456, il path è /home/studente/123456).
- 2 Scrivere con l'editor il programma mostrato sopra e salvarlo con nome helloWorld.c nella cartella appena creata.



Trasforma un programma (cioè, nel caso più semplice, un file di testo contenente un programma, ad esempio in linguaggio C) in un file di formato **eseguibile** dal computer, in modo che possa essere **invocato** da terminale come un altro comando.

Il compilatore che useremo durante il corso è GCC (comando **gcc**).

- GNU's Not Unix! (<https://www.gnu.org>)
- GNU Compiler Collection (originariamente, GNU C Compiler) (<https://gcc.gnu.org>)
- Compilatori per C, C++ , altri linguaggi
- Disponibile per quasi tutte le piattaforme
- Molto ben documentato (**gcc -h**, **man gcc**, **info gcc**)

↑                    ↑                    ↑  
GUIDA            MAN PAGES            INFO  
IN LINEA                                    PAGES

gcc helloWorld.c ↵

gcc \_\_nome\_\_.c

compila (e linka) il file sorgente C \_\_nome.c\_\_, generando il file eseguibile a.out

COLLEGAMENTO

↳ SORGENTE

gcc -o helloWorld helloWorld.c

gcc -o \_\_eseguibile\_\_ \_\_nome.c\_\_

compila (e linka) il file sorgente C \_\_nome.c\_\_, generando il file eseguibile \_\_eseguibile\_\_.

Vedremo altre opzioni del comando gcc.

Su Windows: a.exe

# Terminologia

- Il verbo compilare si usa sia in modo transitivo (“io compilo il programma”) che intransitivo (“il programma compila”, cioè è compilabile, cioè rispetta le regole grammaticali del linguaggio C).
- Un testo compila se segue le regole grammaticali del linguaggio C, che sono formali e non ambigue.
- Un testo che non compila non è un programma C (per questo è valutato 0 nella prova pratica).

# Output del compilatore



Il compilatore riporta i problemi che incontra con due tipi di messaggio:

- **warning** (avvertimento): non impedisce la compilazione, ma segnala un potenziale problema. E' bene modificare il programma in modo da eliminarlo.

Nota: a questo punto, l'eventuale warning

`warning: return type defaults to 'int' [-Wimplicit-int]`

è da considerarsi normale. Vedremo in seguito come evitarlo.

- • **error**: in fase di compilazione, indica un errore di sintassi; compromette la compilazione. Deve essere risolto.

I messaggi riportano (solitamente in modo affidabile) la posizione del problema: ad esempio, `helloWorld.c:2:1` indica la riga `2` e la colonna `1` del file `helloWorld.c`. Poichè un problema può confondere il compilatore, i messaggi successivi al primo possono essere dovuti al primo problema rilevato. Perciò è bene risolvere il primo problema e ricompilare, anziché tentare di risolverli tutti prima di ricompilare.



~~gcc helloWorld.c~~

- 1 Da terminale, compilare il programma salvato nell'esercizio alla slide 3 in un file di nome `helloWorld`.

~~gcc -o helloWorld helloWorld.c~~

- 2 Invocare il file così creato.

~~./a.out~~

~~./helloWorld~~

# Sommario

- 1 Sintassi
- 2 Semantica
- 3 Espressioni intere
- 4 Priorità ed associatività
- 5 Flusso

abA12\*a  
STRINGA

- I linguaggi di programmazione sono **linguaggi formali**.
- La **sintassi** di un linguaggio formale è un insieme di regole che determina se una qualsiasi sequenza di simboli di un alfabeto (ad esempio quello composto da lettere latine, cifre arabe e segni di punteggiatura) è un'espressione del linguaggio.
- Si dice che una grammatica **SINTASSI** **riconosce** un testo se il testo è corretto rispetto alle regole della grammatica.
- Esistono vari formalismi per esprimere sintassi; noi useremo la (Extended) Backus - Naur Form (EBNF).

# Extended Backus-Naur Form

TURING AWARD

- **simbolo terminale**: elemento che compare come tale nel testo (esempio: 0)
- **simbolo nonterminale**: elemento che deve essere sostituito da altri simboli terminali o non terminali (esempio:  $\langle \text{numero} \rangle$ )
- **produzioni (o regole di produzione)**: indicano da cosa (a destra del  $::=$ ) può essere sostituito il nontermiale a sinistra del  $::=$

Nelle produzioni si possono usare i seguenti simboli:

- **alternativa**: ( $\langle \text{numero} \rangle \mid a$  significa "il nonterminale  $\langle \text{numero} \rangle$  oppure il terminale a")
- **zero o più**: ( $\llbracket \langle \text{numero} \rangle \rrbracket^*$  significa "zero o più occorrenze del nonterminale  $\langle \text{numero} \rangle$ ")
- **uno o più**: ( $\llbracket \langle \text{numero} \rangle \rrbracket^+$  significa "una o più occorrenze del nonterminale  $\langle \text{numero} \rangle$ ")

# Esempio di Backus-Naur Form: Numero naturale

- $\langle \text{naturale} \rangle ::= 0 \mid \langle \text{cifra non nulla} \rangle [ \langle \text{cifra} \rangle ]^*$ 

*Handwritten notes:* "SIMBOLO NON TERMINALE" (circled around ::), "ESNF" (above the first dot), "OPPURE" (above the vertical bar), "PRODUZIONE" (to the right), "SIMBOLO TERMINALE" (under the first dot).

Un numero naturale è 0, oppure ("|") una cifra non nulla seguita da zero o più ("[" ... "]"\*) cifre

- $\langle \text{cifra non nulla} \rangle ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Una cifra non nulla è 1, oppure 2, ..., oppure 9

$\langle \text{cifra} \rangle ::= 0 \mid \langle \text{cifra non nulla} \rangle$

*Handwritten:*  $\langle \text{cifra} \rangle = 0 \mid 1 \mid 2 \dots \mid 9$

Una cifra è 0 oppure una cifra non nulla

*Handwritten:* 123

*Handwritten:* 0

*Handwritten:* a25

*Handwritten:* 0

*Handwritten:* 4010

Quali dei seguenti testi sono riconosciuti dalla grammatica alla slide 11?

- 0 Sì
- 23 Sì
- 023 No
- ventitre No
- 300  
  ↓  
  No

# Grammatica (provvisoria) del linguaggio C

$\langle \text{programma} \rangle ::=$  *colpo*  $\{$   
 $\quad \llbracket \langle \text{direttiva} \rangle \rrbracket^*$   
 $\quad \text{main}() \{$   
 $\quad \quad \llbracket \langle \text{istruzione} \rangle \rrbracket^*$   
 $\quad \quad \}$   
 $\quad \}$

•  $\langle \text{direttiva} \rangle ::= \text{\#include} < \langle \text{header} \rangle >$

$\langle \text{header} \rangle ::= \text{stdio.h}$

•  $\langle \text{istruzione} \rangle ::= \langle \text{istruzioneSemplice} \rangle$

$\langle \text{istruzioneSemplice} \rangle ::= \langle \text{espressione} \rangle ;$



*#include <stdio.h>*

*main() {*

*printf("Hello, World\n");*

*}*

# Sommario

- 1 Sintassi
- 2 Semantica**
- 3 Espressioni intere
- 4 Priorità ed associatività
- 5 Flusso



# Semantica

La sintassi ci dice come scrivere programmi C corretti.

Noi però vogliamo poter scrivere programmi non solo corretti, ma anche funzionali, cioè che facciano quello che vogliamo.

La **semantica** definisce il significato dei programmi, cioè che cosa succede quando il file eseguibile prodotto dalla compilazione di un programma viene eseguito.

Conoscendo la semantica di ogni programma, possiamo scrivere quello con la semantica che ci serve.

## La macchina astratta C

- E' un modello (definito dagli standard ANSI C) che ci permette di prevedere l'effetto dell'esecuzione di un programma C.
- In realtà i programmi C vengono trasformati in file eseguibili in linguaggio macchina, e sono questi ultimi ad essere eseguiti. Tuttavia il modello è accurato: l'effetto dell'esecuzione di un programma nella macchina astratta è (per definizione, salvo errori nel compilatore) lo stesso che si ottiene compilando ed eseguendo il programma.
- Per leggere correttamente i programmi, bisogna “mettersi nei panni” della macchina astratta.
- Per programmare correttamente è necessario conoscere la macchina astratta. La verifica ultima della propria comprensione della macchina astratta è l'esecuzione del programma.

# Espressioni

~~7 x 15~~  
-

Le conosciamo dall'algebra elementare.

## Example

$$7 + 2 \times (5 - 3)$$

Diagram illustrating the evaluation of the expression  $7 + 2 \times (5 - 3)$  using the order of operations (PEMDAS/BODMAS):

- Step 1: Evaluate the innermost parentheses:  $5 - 3 = 2$ .
- Step 2: Evaluate the multiplication:  $2 \times 2 = 4$ .
- Step 3: Evaluate the addition:  $7 + 4 = 11$ .

Applicando a un'espressione un determinato procedimento (valutazione), si giunge un risultato che corrisponde al valore dell'espressione.

# Concetto di espressione nei linguaggi di programmazione

Costrutto (frammento di codice scritto secondo regole sintattiche) la cui **valutazione** produce

- un **valore** (ad esempio un numero), come nell'algebra elementare; ma anche
- un **effetto** (side effect) su memoria, input e output
- La semantica di un'espressione è <sup>RAM</sup> interamente caratterizzata dal suo valore e dal suo effetto.

Notazione: **\_\_espressione\_\_**  $\rightarrow$  **\_\_valore\_\_** significa che il valore di **\_\_espressione\_\_** è **\_\_valore\_\_**.

Esistono vari tipi di espressione. Iniziamo con due famiglie di espressioni, che consentono di usare la macchina astratta come semplice calcolatrice:

$\langle \text{espressione} \rangle ::= \langle \text{espressioneDiOutput} \rangle$   
           |  $\langle \text{espressioneIntera} \rangle$

# Capacità della macchina astratta C

Cominciamo da un modello semplice, che arricchiremo quando necessario. Ha le seguenti capacità:

- Valuta espressioni (calcolandone il valore e producendone l'effetto)
- Esegue un'istruzione semplice valutando l'espressione che si ottiene togliendo il `;` e scartando il valore

Finora abbiamo considerato programmi composti da una sola istruzione semplice. Quindi per prevedere l'effetto dell'esecuzione del programma è sufficiente conoscere la semantica dell'espressione contenuta nell'istruzione.

```
#include <stdio.h>
main() {
    printf(" . ");
}
```

espressione

## Espressione di output

$\langle \text{espressioneDiOutput} \rangle ::=$

`printf(  $\langle \text{costanteStringa} \rangle$  [ ,  $\langle \text{espressione} \rangle$  ]* )`

$\langle \text{costanteStringa} \rangle ::=$  " [  $\langle \text{carattere} \rangle$  ]\* "

La `costanteStringa` è detta **stringa di formato**.

Vedremo che l'espressione di output non è che un'istanza di una classe di espressioni più generale (dette **chiamate di funzioni**) ma per ora supponiamo che sia un'espressione sui generis.

Per usare l'espressione `printf` è necessario includere l'header `stdio.h` con la direttiva `#include <stdio.h>`.

*Handwritten example:* `printf("Hello")` → 5

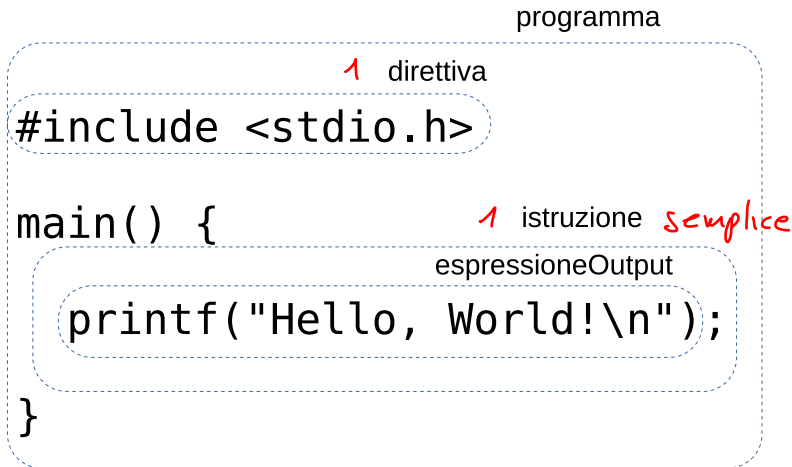
### Semantica

- Effetto: se non contiene nessuna espressione, stampa a video `costanteStringa`; fra poco tratteremo il caso con espressioni
- Valore: il numero di caratteri stampati

Di solito il valore di un'espressione di output non interessa.

## Parsing dell'esempio alla slide 3

**Parsing** è il riconoscimento di un testo secondo una grammatica.



# Sommario

- 1 Sintassi
- 2 Semantica
- 3 Espressioni intere**
- 4 Priorità ed associatività
- 5 Flusso



# Espressione intera

Rappresenta operazioni su numeri interi.

$\langle \text{espressioneIntera} \rangle ::= \langle \text{costanteIntera} \rangle$  125 0

|  $\langle \text{espressioneUnaria} \rangle$  -12

|  $\langle \text{espressioneBinaria} \rangle$  15 + 12

|  $( \langle \text{espressioneIntera} \rangle )$  (12)

$\langle \text{espressioneUnaria} \rangle ::= [ + | - ] \langle \text{espressioneIntera} \rangle$

$\langle \text{espressioneBinaria} \rangle ::=$

$\langle \text{espressioneIntera} \rangle [ + | - | * | / | \% ] \langle \text{espressioneIntera} \rangle$

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$  sono detti **operatori**; le espressioni a cui sono applicati sono dette **operandi**. Gli operandi sono espressioni intere, quindi possono a loro volta essere costituite da operatori ed operandi.

5+6 (5+6)\*3

((15+12)\*(5-6))

## Semantica

- Effetto: nessuno NO INPUT, NO OUTPUT, NO CAMBIAMENTI MEMORIA
- Valore: come nell'algebra elementare

## Costante intera

Un'espressione costante intera è un numero naturale, secondo la grammatica alla slide 11.

Il suo valore è il numero stesso.

0  $\rightarrow$  0

1  $\rightarrow$  1

12345  $\rightarrow$  12345

# Espressione unaria

Il valore di  $+ \text{__espressione__}$  è lo stesso di  $\text{__espressione__}$ .

## Example

$$+1 \rightarrow 1$$

Il valore di  $- \text{__espressione__}$  è il valore di  $\text{__espressione__}$  cambiato di segno.

## Example

$$-1 \rightarrow -1$$

$$-(-1) \rightarrow 1$$

## Esercizio

$$- (+ (- (- (+ (-2)))))) \rightarrow +2 \quad (\text{oppure } 2)$$

$$\begin{array}{c}
 \underbrace{-2}_{-2} \\
 \underbrace{+2}_{-2} \\
 \underbrace{-2}_{-2} \\
 \underbrace{-2}_{+2}
 \end{array}$$

# Somma, differenza, prodotto

Come nell'aritmetica elementare (ma il prodotto è indicato da  $*$ ):

- il valore di  $\_\_\text{espr1}\_\_ + \_\_\text{espr2}\_\_$  è la somma dei valori di  $\_\_\text{espr1}\_\_ e \_\_\text{espr2}\_\_.$   
 $\quad \quad \quad -2 \quad \quad \quad 5 \quad \quad \rightarrow 3$

$$3 + 5 \rightarrow 8$$

- il valore di  $\_\_\text{espr1}\_\_ - \_\_\text{espr2}\_\_$  è la differenza dei valori di  $\_\_\text{espr1}\_\_ e \_\_\text{espr2}\_\_.$   
 $\quad \quad \quad -2 \quad \quad \quad 5 \quad \quad \rightarrow -7$

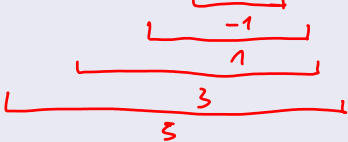
$$3 - 5 \rightarrow -2$$

- il valore di  $\_\_\text{espr1}\_\_ * \_\_\text{espr2}\_\_$  è il prodotto dei valori di  $\_\_\text{espr1}\_\_ e \_\_\text{espr2}\_\_.$   
 $\quad \quad \quad (-2) \quad * \quad 5 \quad \quad \rightarrow -10$

$$3 * 5 \rightarrow 15$$

## Esercizio

$$2 + (3 * (- (4 - 5))) \rightarrow 5$$



Handwritten annotations for the expression  $2 + (3 * (- (4 - 5)))$  showing the order of operations:

- A bracket under  $(4 - 5)$  with  $-1$  written below it, indicating the innermost operation.
- A bracket under  $-(4 - 5)$  with  $1$  written below it, indicating the next operation.
- A bracket under  $3 * (- (4 - 5))$  with  $3$  written below it, indicating the multiplication.
- A final bracket under the entire expression  $2 + (3 * (- (4 - 5)))$  with  $5$  written below it, indicating the final result.

## Quoziente intero

$$5 / 2 \rightarrow 2$$

Il valore di `__espr1__ / __espr2__` è il quoziente intero della divisione del valore di `__espr1__` per il valore di `__espr2__`

### Example

$$7 / 3 \rightarrow 2$$

### Esercizio

$$4 / 2 \rightarrow 2$$

$$5 / 3 \rightarrow 1$$

$$2 / 0 \rightarrow \text{ERRORE}$$

# Resto della divisione intera (o modulo)

$$5/2 \rightarrow 2$$

$$5\%2 \rightarrow 1$$

Il valore di `__espr1__ % __espr2__` è il resto della divisione del valore di `__espr1__` per il valore di `__espr2__`

## Example

$$7 \% 3 \rightarrow 1$$

## Esercizio

$$4 \% 2 \rightarrow 0$$

$$5 \% 3 \rightarrow 2$$

$$2 \% 0 \rightarrow \text{ERRORE}$$

$$-7 \% 3 \rightarrow -1$$

$$-7 / 3 \rightarrow -2$$

~~$$-7 / 3 = -3$$

$$-7 \% 3 = 2$$~~



# Output di espressioni aritmetiche

*printf("%d", 2+3)*  
 (The `%d` is underlined, and `2+3` is underlined with a bracket pointing to the `5` result.)

`printf("...%d...", __espressione__)`

`%d` è uno **specificatore di conversione**. Viene sostituito dal valore di `__espressione__`; il resto della stringa di formato viene scritto così com'è.

`printf("%d", 2)` stampa 2

E' possibile scrivere il valore di più di una espressione in una sola stringa di formato, usando tante espressioni quanti sono gli specificatori di conversione.

`printf("La somma di %d e %d è %d\n", 2, 3, 2+3)` stampa La somma di 2 e 3 è 5 e va a capo.

*printf("%d%d%d", 3, 5, 7) stampa 357*

*printf("%d %d", 3, 5) stampa 3 5*

$2+3;$       $\text{printf}(\text{"\%d", } 2+3),$

La slide 18 dice che nell'eseguire un'istruzione la macchina astratta scarta il valore dell'espressione in essa contenuta.

Perché allora ci interessa il valore delle espressioni? Perché un'espressione può contenerne altre e il valore delle espressioni interne determina l'effetto dell'espressione esterna.

## Example

L'effetto dell'espressione `printf("%d", 2)` (cioè il numero stampato) è determinato dal valore dell'espressione contenuta al suo interno.



## Espressioni intere

Verificare, con opportuni programmi, la correttezza delle soluzioni degli esercizi nelle diapositive precedenti stampando a video il valore delle espressioni e quello valutato manualmente.

## Example

Per verificare che  $7 \% 3 \rightarrow 1$ , si può usare il seguente programma:

```
#include <stdio.h>
main() {
    printf ("%d = %d", 7 % 3, 1);
}
```

# Sommario


- 1 Sintassi
- 2 Semantica
- 3 Espressioni intere
- 4 **Priorità ed associatività**
- 5 Flusso

$$\begin{array}{c} 2 + 3 * 4 \\ | \\ 2 + (3 * 4) \end{array} \quad \begin{array}{l} 20 \\ \textcircled{14} \end{array}$$

# Parentesi

Si possono scrivere espressioni complesse in cui gli operandi sono a loro volta espressioni racchiuse fra parentesi (solo tonde: le quadre e le graffe in C si usano per altri scopi).

## Example

$$(2 + 6) * (8 - (7 + 2))$$


Le parentesi che racchiudono le espressioni sono opzionali. Se sono omesse, l'ordine di valutazione è determinato dalle regole di priorità e associatività.

$$2 + 3 * 4$$

$$2 + (3 * 4)$$

In algebra, si ottiene un'espressione equivalente raggruppando (cioè mettendo fra parentesi) le potenze prima di moltiplicazioni e divisioni, e queste prima di addizioni e sottrazioni. In C funziona allo stesso modo.

Formalmente, in C ogni operatore è caratterizzato da un numero intero, detto **priorità**. Ad esempio,  $*$  ha priorità 3 e  $+$  ha priorità 4.

Se ci sono due modi di leggere un'espressione, viene preferito quello che corrisponde al mettere fra parentesi l'espressione con l'operatore di priorità minore.

## Example

L'espressione  $2 + (3 * 4)$  si può leggere come  $(2 + 3) \times 4$  (addizione fra parentesi) o come  $2 + (3 \times 4)$  (moltiplicazione fra parentesi). Le regole di priorità del C assicurano che l'espressione venga letta dalla macchina astratta come è consueto in algebra.

# Associatività

L'associatività (a sinistra o a destra) di un operatore indica come vengono raggruppate espressioni con operatori della stessa priorità in assenza di parentesi.

## Example

- L'associatività del  $+$  binario è a sinistra. Quindi l'espressione  $2 + 3 + 4 + 5$  si legge come  $((2 + 3) + 4) + 5$ .
- L'associatività del  $-$  unario è a destra. Quindi l'espressione  $- - 2$  si legge come  $- (-2)$ .

$$-(-3)$$

$$(((2 + 3) + 4) + 5)$$

## Priorità e associatività degli operatori aritmetici

Famiglia	Operatore	Priorità	Associatività
Unari	$+, -$	2	Destra
Binari Moltiplicativi	$*, /, \%$	3	Sinistra
Binari Additivi	$+, -$	4	Sinistra

$$2 - (3 - 4)$$

# Priorità, associatività e parentesi

$$2 + \cancel{3} * \cancel{4}$$

Le parentesi si possono omettere se indicano un raggruppamento delle espressioni uguale a quello stabilito dalle regole di priorità e associatività.

## Example

$2 + (3 * 4)$  è equivalente a  $2 + 3 * 4$ .

$2 * (3 + 4)$  **non** è equivalente a  $(2 * 3) + 4$  (priorità).

$(2 + \cancel{3}) + \cancel{4}$  è equivalente a  $(2 + 3) + 4$ .

$2 - (3 - 4)$  **non** è equivalente a  $(2 - 3) - 4$  (associatività).

Fortunatamente, la valutazione funziona come nell'algebra elementare.

$$(2 - 3) - 4$$

# Esercizio

Togliere dalle seguenti espressioni le parentesi non necessarie:

sì (i)  $\{2 + 3\}$

NO (ii)  $(3 + 2) * 4$        $3 + (2 * 4)$

sì (iii)  $2 + (2 / 6) - 5$        $2 + (2 / 6) - 5$

(iv)  $4 / (2 * 3)$

(v)  $4 * (5 / 2)$

(vi)  $2 + (6 * (7 / 2) - 5)$

(vii)  $6 - ((2 * (4 + 3)) - 5)$



## Parentesi

Verificare la correttezza della valutazione delle espressioni nelle diapositive precedenti stampandone il valore per mezzo di opportuni programmi.

```
#include <stdio.h>
main() {
```

```
    printf("%d = %d\n",
           2 + (2 / 6) - 5, 2 + 2 / 6 - 5),
}
```

# Raggruppamento e ordine di valutazione

$$(2 * 5) + (6 * 7)$$

Non bisogna confondere raggruppamento e ordine di valutazione delle espressioni. Le regole di priorità e associatività determinano il raggruppamento, cioè *quali* espressioni vengono valutate, ma non l'ordine di valutazione.

## Example

Nell'espressione  $(2 + 4) + (7 - 5)$  quale dei due addendi viene valutato per primo? Il linguaggio non lo specifica.

L'ordine di valutazione dei componenti di una espressione non è influente sul risultato finale *se le espressioni non hanno effetto*.

## Example

Qual è l'effetto della seguente espressione?

```
printf("%d\n", printf("Primo") + printf("Secondo"))
```

Dipende dall'ordine di valutazione degli addendi.

PrimoSecondo 12      SecondoPrimo 12

# Sommario

- 1 Sintassi
- 2 Semantica
- 3 Espressioni intere
- 4 Priorità ed associatività
- 5 Flusso**

# Flusso predefinito

Secondo la grammatica alla slide 13, non è detto che un programma abbia esattamente una istruzione: può averne zero o più.

Se un programma ha più di una istruzione, la macchina astratta le esegue una alla volta, dalla prima all'ultima.

Il **flusso** è l'ordine in cui vengono eseguite le istruzioni.

Il **flusso predefinito** (cioè quello seguito dalla macchina astratta in assenza di indicazioni diverse) coincide quindi con l'ordine testuale delle istruzioni.

Il flusso si può modificare rispetto a quello predefinito con le istruzioni di **controllo di flusso**, che vedremo più avanti.



## La macchina astratta C: nuova capacità

- Valuta espressioni (calcolandone il valore e producendone l'effetto)
- Esegue un'istruzione semplice valutando l'espressione che si ottiene togliendo il ; e scartando il valore
- • Segue il flusso predefinito del programma (cioè eseguita un'istruzione passa alla successiva)

Che cosa fa il seguente programma?

030\_introduzione\_al\_c/helloWorld2.c

```
1 #include <stdio.h> DIRETTIVA
2 main() {
3     printf("Hello,");
4     printf(" World!\n"); non c'è \n
5 }
```

In cosa differisce l'output di questo programma (che contiene due istruzioni) da quello della diapositiva 3 (che ne contiene una sola)?

*Hello, World! ↵*

## Operazioni

Scrivere un programma che stampi a video la somma, la differenza, il prodotto e il quoziente intero dei due numeri interi 5 e 2. Verificarne la correttezza compilandolo ed eseguendolo.