Laboratorio di Programmazione

08:30 – 12:30 lunedì – aula Omega e Sigma

08:30 – 10:30 martedì – aula Omega e Delta

Lezione 1 – Primi esercizi introduttivi

Marco Anisetti (teoria)
Dipartimento di Informatica
marco.anisetti@unimi.it

Matteo Luperto (lab. turno A)
Dipartimento di Informatica
matteo.luperto@unimi.it

Nicola Bena (lab. turno B)
Dipartimento di Informatica
nicola.bena@unimi.it

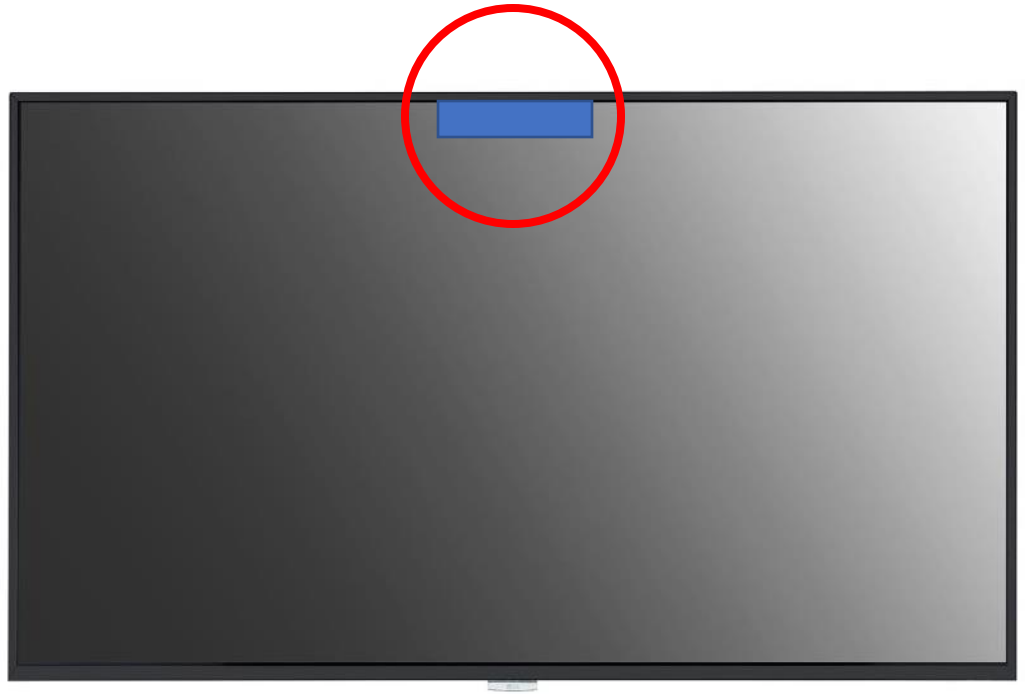
Come funziona il laboratorio

- Esercizi al computer
- Elenco/lista di esercizi forniti a inizio della lezione
  - No soluzioni / alcune soluzioni fornite lez. successive, alcune discusse in aula
- Svolti singolarmente
- Il tempo richiesto per svolgere tutti gli esercizi è spesso maggiore del tempo di lezione
- Gli esercizi rimanenti sono da svolgersi individualmente (a casa, al computer) e discussi alla lezione successiva

Ricevimento:
mail al docente di laboratorio, lunedì/martedì pomeriggio ricevimento

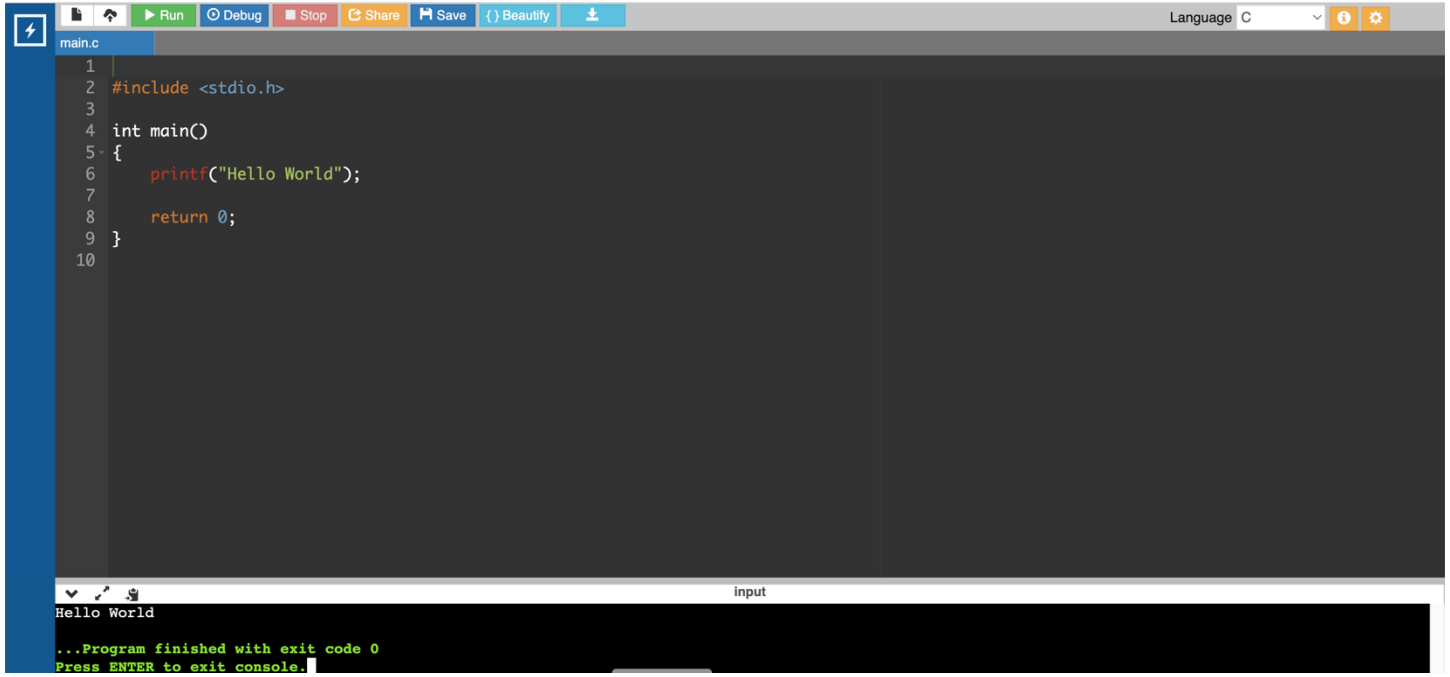
Setup in presenza

- Fate login usando Linux come sistema operativo:
  - In alto, al centro trovate una linguetta;
  - Vi apparirà un menù per scegliere tra Windows e Linux; scegliete il secondo
  - Effettuate il login usando la vostra mail e password @studenti.unimi.it



Setup da remoto

- Opzione 1 (temporanea/backup):
  Compilatore online su Online GDB https://www.onlinegdb.com/
  Pro: semplice da configurare, non dovete installare nulla, web-based
  Contro: setup semplificato rispetto ad esame



Setup personale

- Opzione 2 (consigliata):
  Installare sul vostro computer personale
  - Compilatore
  - Editor di testo (minimale)

  Pro: setup uguale a quello del lab
  Contro: il setup è specifico per la vostra configurazione
  (Linux consigliato, anche con macchina virtuale, macOs e windows OK)

Avvisate / chiedete consulto in caso di problemi

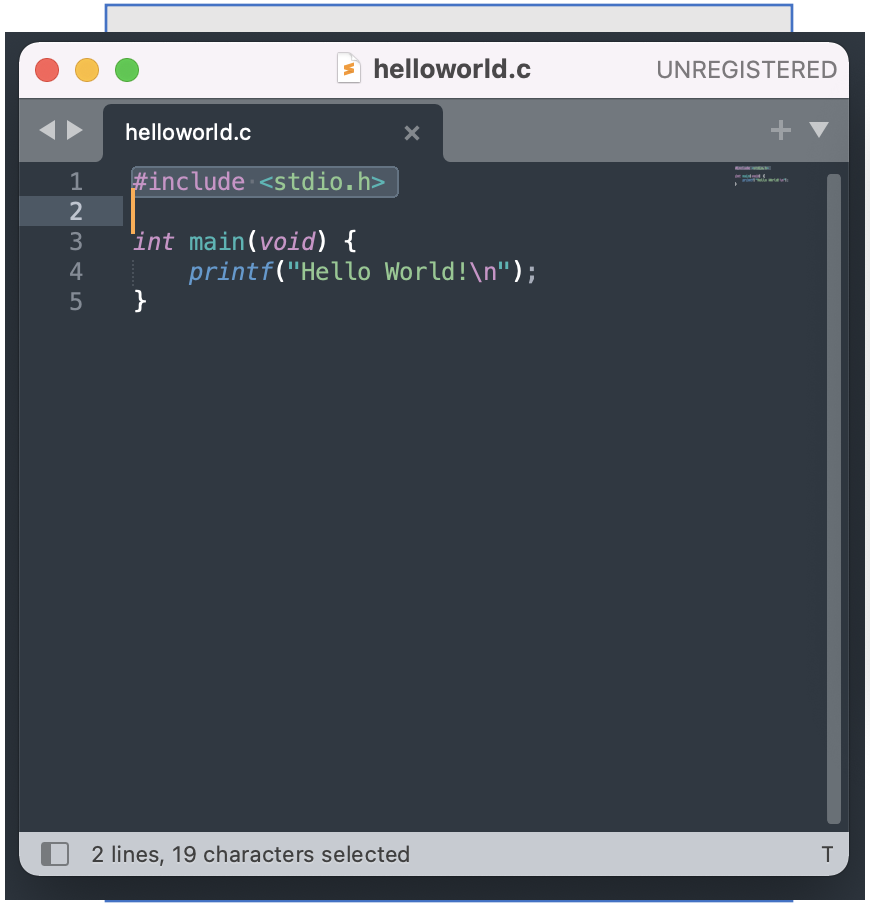
Come fare gli esercizi

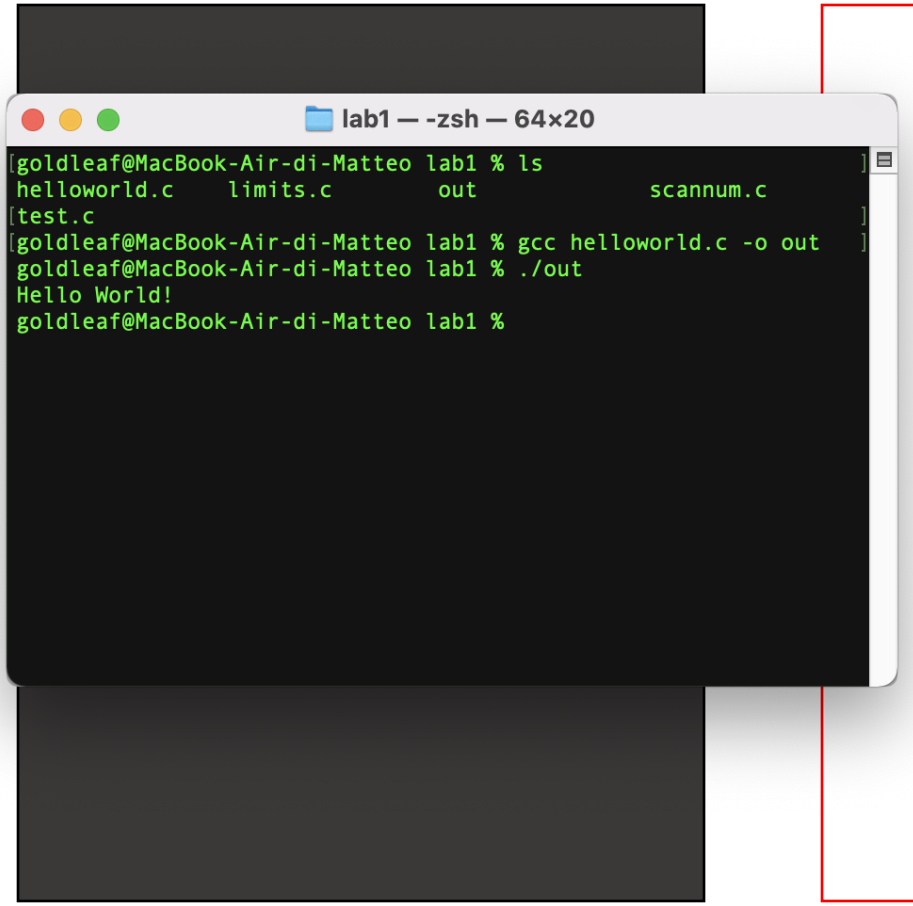
Editor di testo:
qui scriverete il vostro codice sorgente
Consigliato:
Pluma

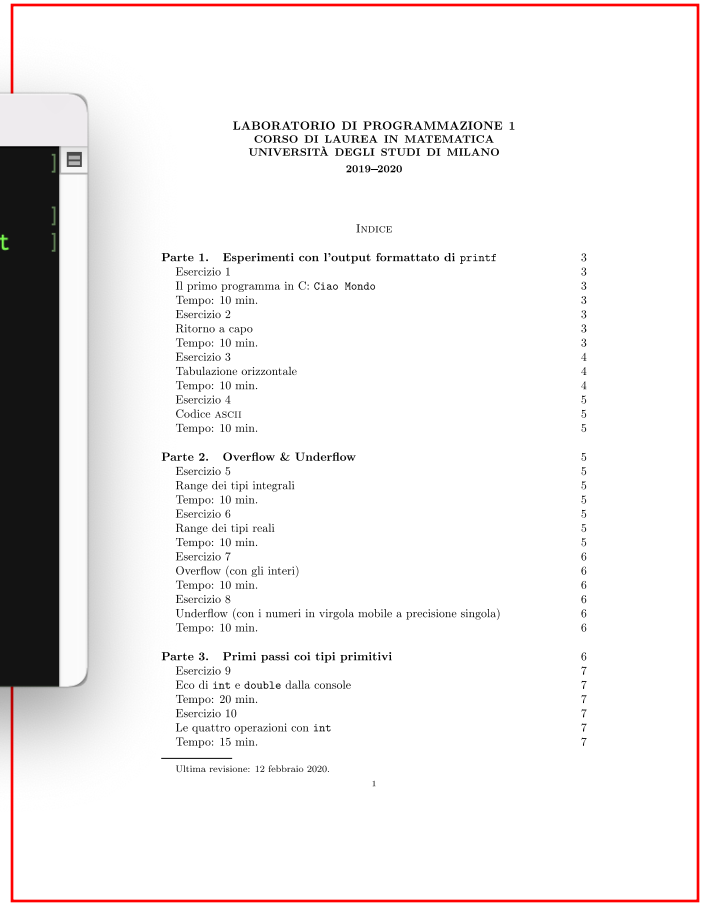
Terminale:
qui compilate il vostro codice sorgente, creerete eseguibile, lo lancerete

Testo degli esercizi

Come fare gli esercizi







LABORATORIO DI PROGRAMMAZIONE 1
CORSO DI LAUREA IN MATEMATICA
UNIVERSITÀ DEGLI STUDI DI MILANO
2019–2020

INDICE

Ultima revisione: 12 febbraio 2020.

Struttura del laboratorio

- Segue il programma visto a teoria

1. Prime lezioni introduttive su linguaggio `C` → Ottobre
2. Concetti più avanzati, sempre in linguaggio `C` → Novembre
3. Programmazione ad oggetti in `Java` → Dicembre

- Simulazione esame ed esercizi riassuntivi a fine blocco 2. e 3.

Modalità d'esame di laboratorio:

Esercizi da svolgersi in `C` e `Java`, al computer, ~3.5h a disposizione, open book + slide del corso fornite in pdf.

Storia del Linguaggio C

- Fu inventato nel 1972 da Dennis Ritchie presso i laboratori della AT&T Bell
- Serviva a scrivere il codice del sistema operativo UNIX per un processore DEC PDP-11
- Nel 1989 l'American National Standards Institute (ANSI) ha approvato lo standard C89, noto anche come ANSI C
- L'anno seguente ISO ha adottato lo standard ANSI C
    - Revisioni in 1999 (C99), 2011 (C11) e 2018 (C18)

Struttura di un programma

- Un programma è come una **funzione** che prende degli ingressi e ritorna delle uscite
- Possiamo iniziare a pensare che tale **funzione** debba essere esplicitata quando scriviamo del codice
    - Formato: `<output> <nomefunzione> ‘‘(’’ <input> ‘‘)’’…`

- Il programma può essere composto a sua volta da sequenze di funzioni

- La funziona principale, quella che ‘‘contiene’’ tutto il programma, viene per convenzione chiamata **`main`**

- Fisicamente un programma scritto in un linguaggio di programmazione risiede su uno o più file di testo
    - Di solito ogni file contiene una o più funzioni differenti

Struttura di un programma

- Questi file che contengono il programma si chiamano anche ***sorgenti***
- Un programma può essere fatto da più file sorgente che sono tutti necessari per la sua esecuzione
    - Utile per leggere il codice e per riusarlo

- Prima di eseguire il programma quindi vengono in modo automatico riuniti i sorgenti in un'unica entità che poi viene tradotta nel codice eseguibile

- Questo processo si chiama ***compilazione***

- La traduzione può avvenire in più fasi avvicinandosi al vero linguaggio del calcolatore che è chiamato **linguaggio macchina**

Assembler

- Il linguaggio più vicino al linguaggio macchina, che non richiede passaggi intermedi nel suo processo di traduzione, è l'assembly

- Gli altri linguaggi ad alto livello vengono prima tradotti in assembly e poi in linguaggio macchina

- Alto livello vuol dire che il linguaggi si avvicina di più al modo di pensare delle persone che al modo di agire del computer

Esempio: M.C.D in assembly

```
label1:  LOAD R01, 101
         LOAD R02, 102
         DIV R01, R02
         MUL R01, R02
         LOAD R02, 101
         SUB R02, R01
         JZERO R02, fine
         LOAD R01, 102
         STORE R01, 101
         STORE R02, 102
         JUMP label1
fine:    LOAD R01, 102
         STORE R01, 103
```

Compilazione

- Processo di traduzione dal codice in linguaggio ad alto livello (anche più file) al programma in linguaggio macchina (un file eseguibile)

- E’ scomposto in fasi

1. **precompilazione o preprocessing**: fonde codici, modifica o cancella brani di codice

2. **compilazione**:
   traduce un codice in un file oggetto, di solito passando per il codice assembly

3. **linking** (da oggetti a eseguibile): lega file oggetto ed eventuali librerie esterne in un programma

Steps preliminari

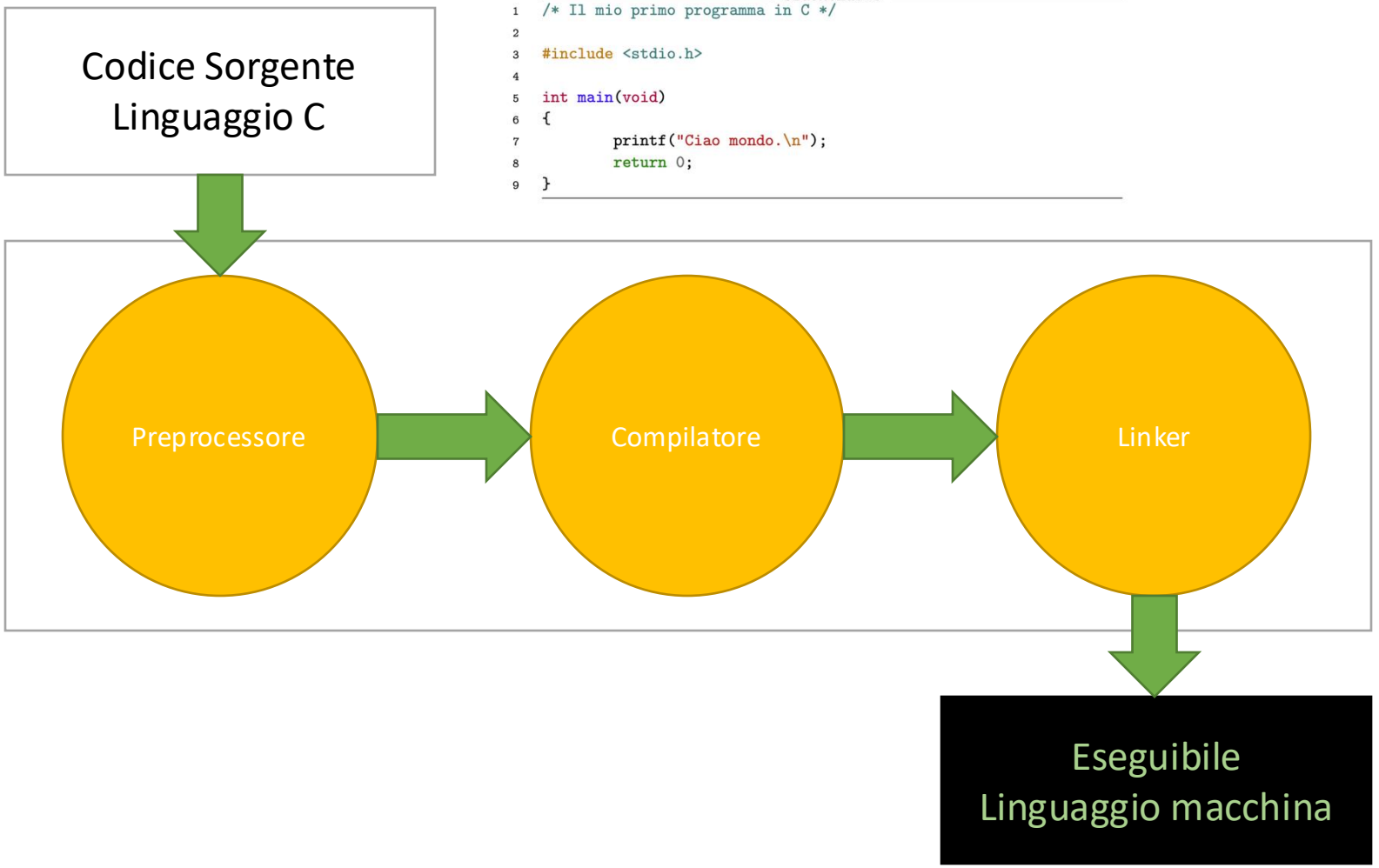
- Create una cartella in una directory a vostro piacere:
  es: `Documents/Laboratorio/Laboratorio1`
- Salvate i file sorgenti in quella cartella
- Aprite il terminale e entrate in quella cartella dal terminale

  Comandi:
  - «`cd cartellaDestinazione`» vi permette di cambiare cartella ed entrare in `cartellaDestinazione`
  - «`ls`» vi permette di vedere cosa c'è nella cartella (comprese sottocartelle)
  - «`pwd`» vi dice quale è la cartella corrente in cui siete
  - «`mkdir nomeCartella`» vi permette di creare una cartella di nome `nomeCartella`

Dal sorgente all'eseguibile

CiaoMondo.c

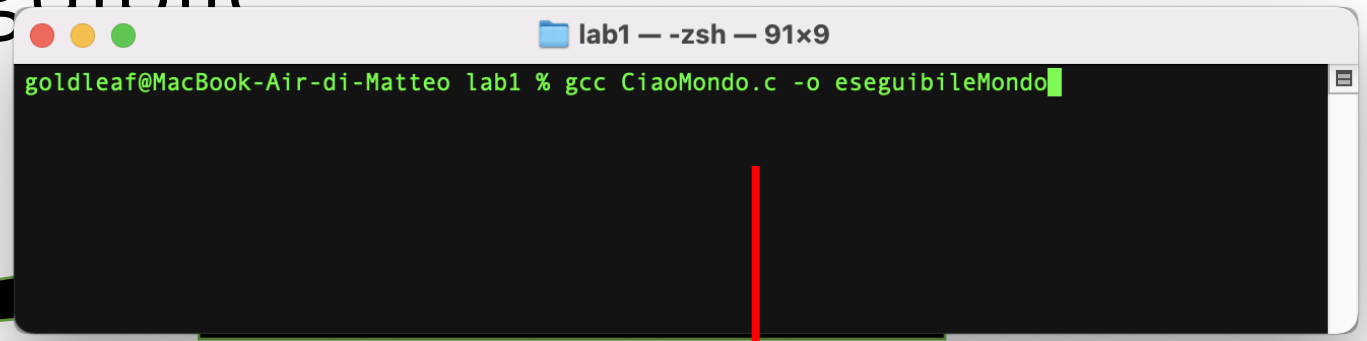
```c
/* Il mio primo programma in C */

#include <stdio.h>

int main(void)
{
        printf("Ciao mondo.\n");
        return 0;
}
```

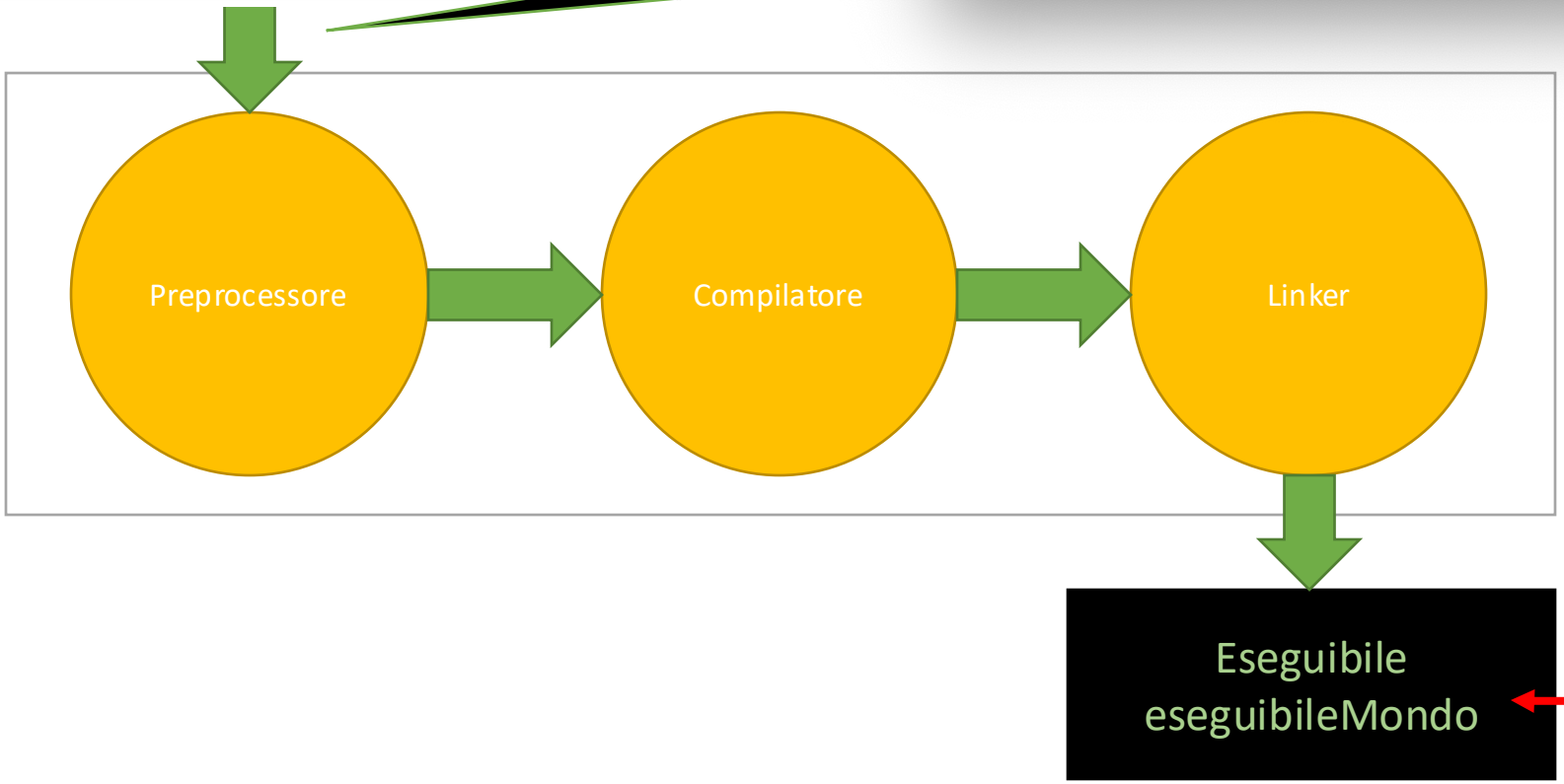


Dal sorgente all'eseguibile

CiaoMondo.c

```c
/* Il mio primo programma in C */

#include <stdio.h>

int main(void)
{
        printf("Ciao mondo.\n");
        return 0;
}
```

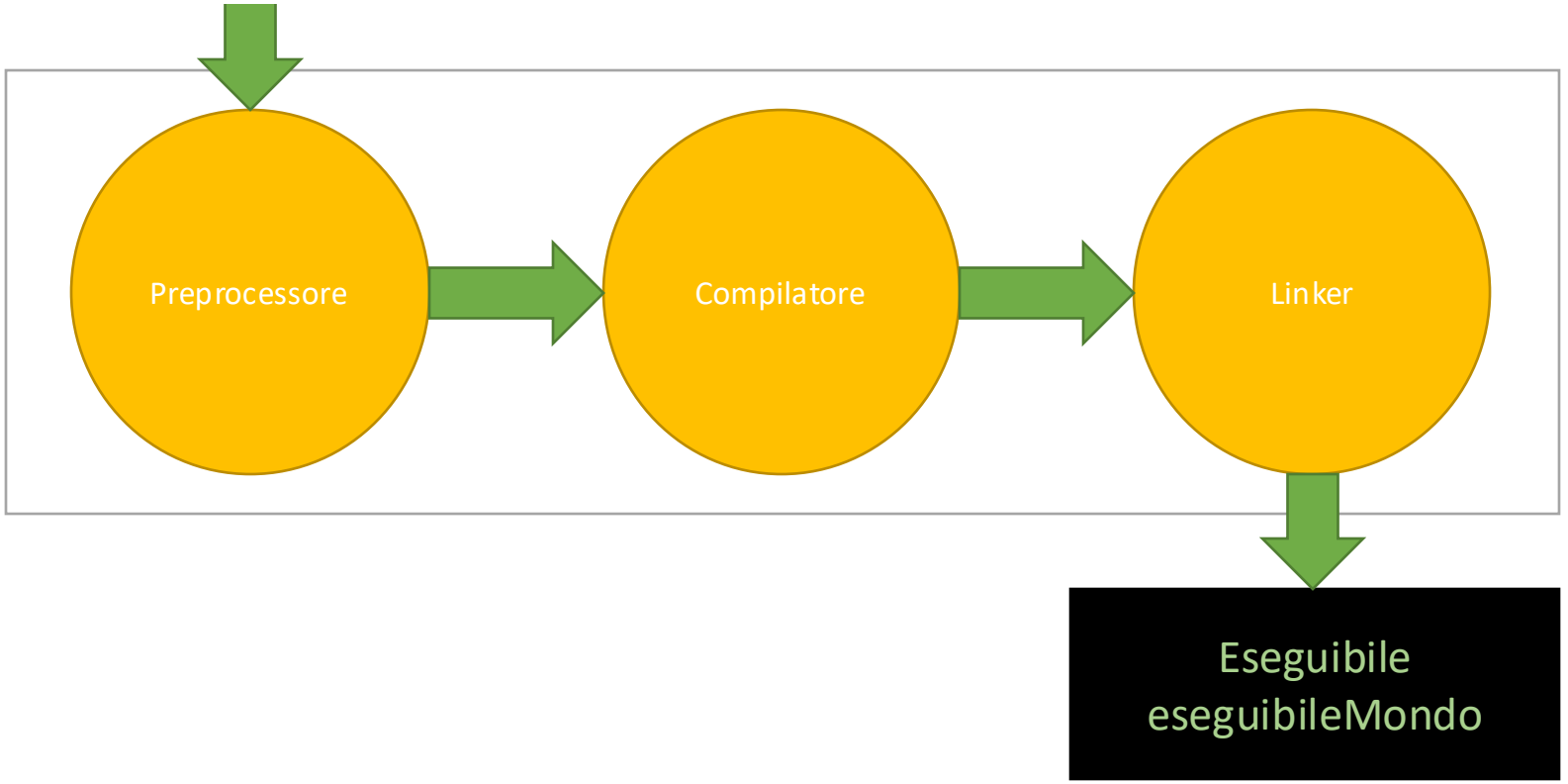




Dal sorgente all'eseguibile

CiaoMondo.c

```c
/* Il mio primo programma in C */

#include <stdio.h>

int main(void)
{
        printf("Ciao mondo.\n");
        return 0;
}
```



Siate ordinati!

1. Date nomi diversi al codice sorgente di esercizi diversi
2. Date nomi diversi al codice eseguibile

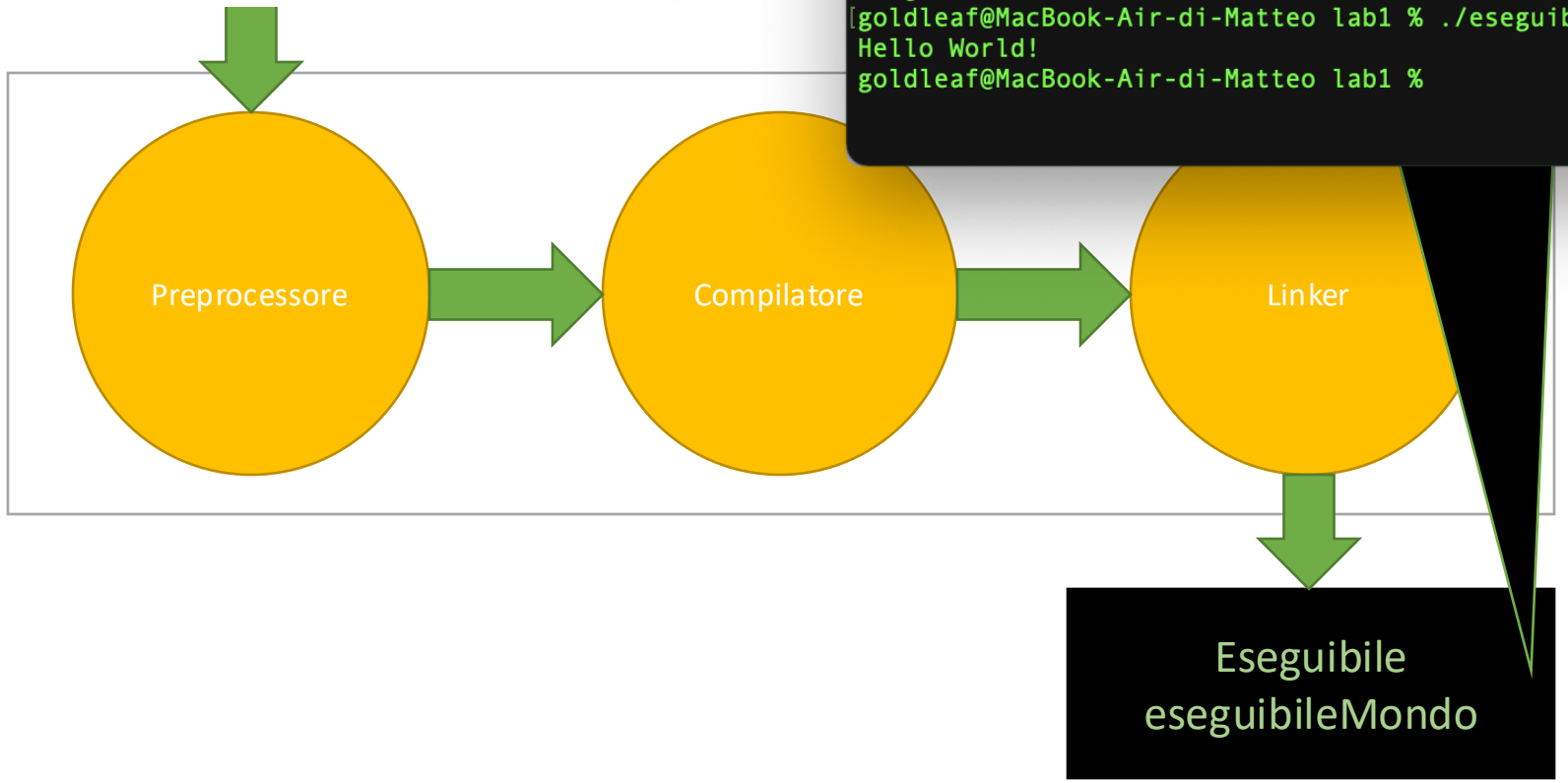
Attenzione!
Dare lo stesso nome del file contenente il codice sorgente all'eseguibile cancellerà il codice sorgente (verrà sovrascritto dall'eseguibile)

Eseguire il file compilato

CiaoMondo.c

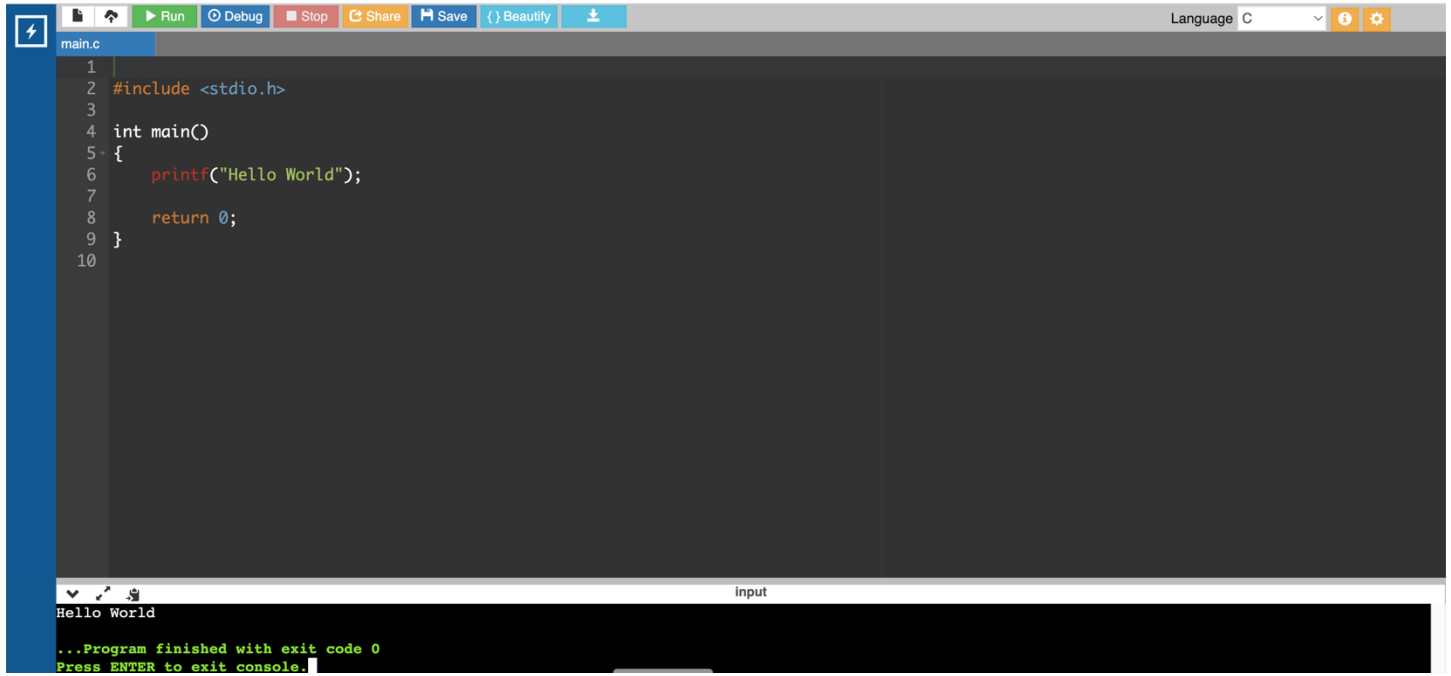
```c
/* Il mio primo programma in C */

#include <stdio.h>

int main(void)
{
        printf("Ciao mondo.\n");
        return 0;
}
```

```
lab1 — -zsh — 91×9
[goldleaf@MacBook-Air-di-Matteo lab1 % gcc CiaoMondo.c -o eseguibileMondo
[goldleaf@MacBook-Air-di-Matteo lab1 % ls
CiaoMondo.c     helloworld.c    out             test.c
eseguibileMondo limits.c        scannum.c
[goldleaf@MacBook-Air-di-Matteo lab1 % ./eseguibileMondo
Hello World!
goldleaf@MacBook-Air-di-Matteo lab1 %
```



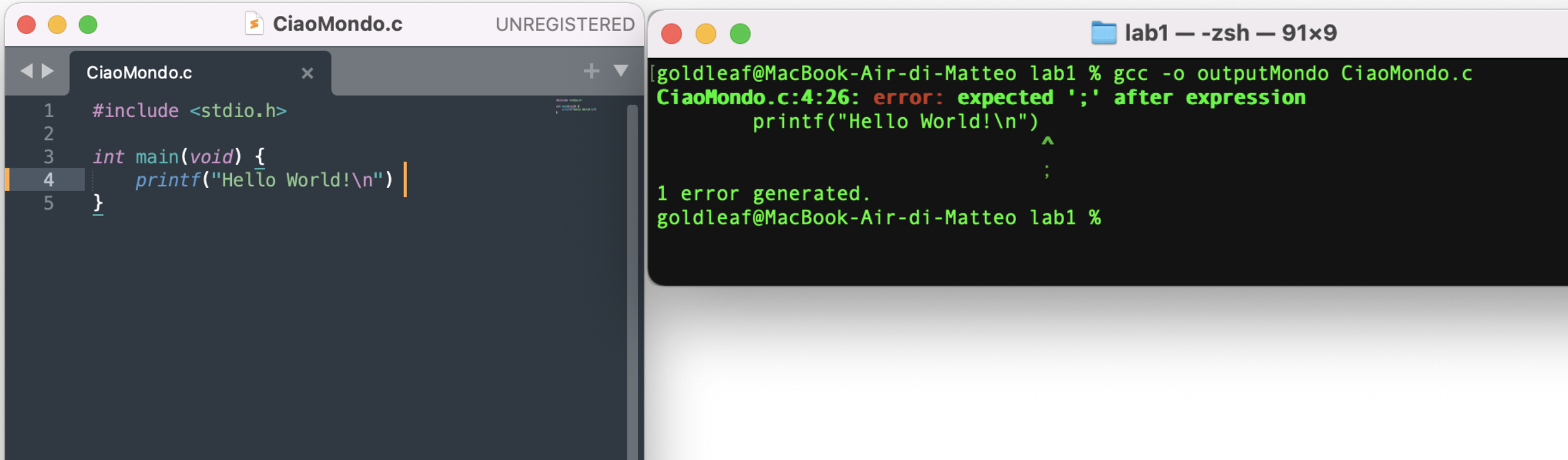
Dal sorgente all’eseguibile – setup remoto

- onlineGDB fa tutti gli step in maniera automatica
- Compilazione ed esecuzione vengono fatti premendo RUN
- Attenzione! Questa è una semplificazione (va bene, ma tenetene conto)



Comprendere output del compilatore

- Se ci sono errori di sintassi, il compilatore non produce l'eseguibile
- Inoltre vi dice dove ha incontraro l'errore:
  usate questa informazione per correggere

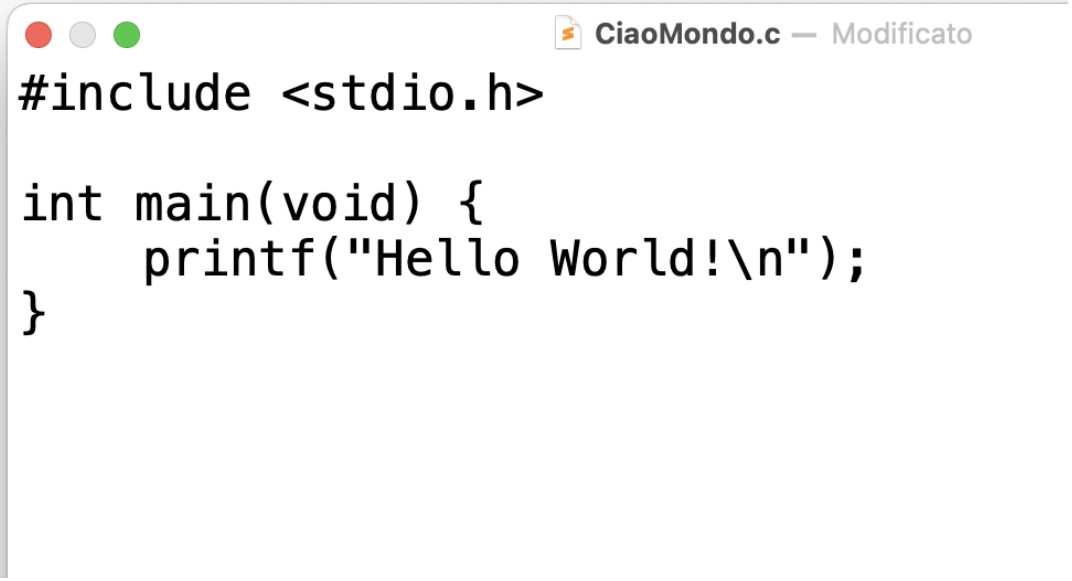


Setup in lab:

- Linux
- Pluma (editor – o un altro editor di testo minimale)
- Compilatore su terminale

Setup da remoto:

- https://www.onlinegdb.com/ (online via web browser)
- Replica di setup del lab su vostra macchina o virtual machine



Comandi utili per il usare il terminale:

- **`cd nomeCartella`** si sposta in una cartella di nome **`nomeCartella`**
- **`mkdir nomeCartella`** crea una cartella di nome **`nomeCartella`**
- **`pwd`** nome della cartella corrente
- **`ls`** mostra contenuto cartella
- **`cd ..`** Torna alla cartella precedente

Comandi utili per il compilare ed eseguire il vostro programma:

- (scrivete il vostro programma su di un editor di testo, salvatelo)
- entrate nella cartella dove è salvato **`programma.c`**
- **`gcc programma.c -o eseguibile`** compila e crea eseguibile (leggere output di compilazione per errori)
- **`./eseguibile`** esegue il programma (stampa a terminale il risultato)

Il primo programma in C

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

int main(int argc, char *argv[]) {

    printf("Hello World\n");
    return 0;
}
```

Il primo programma in C

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

int main(int argc, char *argv[]) {

    printf("Hello World\n");
    return 0;
}
```

Librerie che vengono aggiunte e che contengono le funzioni utilizzate nel codice, scritte da altri (es, `printf`)

Usare funzioni da libreria vi permette di non «reinventare la ruota»

Il primo programma in C

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

int main(int argc, char *argv[]) {

    printf("Hello World\n");
    return 0;
}
```

Funzione principale con relativi input e output
Per l'output si specifica solo il tipo
Per l'input anche un nome della variabile
L'input è opzionale può essere omesso

```
int main ()
```

Il primo programma in C

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

int main(int argc, char *argv[]) {

    printf("Hello World\n");
    return 0;
}
```

{ } racchiudono un **blocco di codice** in questo caso il blocco che fa parte della funzione main.

Indentare il codice correttamente aiuta a gestire i blocchi di codice e a rendere il codige leggibile (e debuggabile)

Il primo programma in C

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

int main(int argc, char *argv[]) {

    printf("Hello World\n");
    return 0;
}
```

- `printf` chiama una funzione che stampa del testo
- il “;” delimita la fine di una linea di codice definendo la **sequenza** di esecuzione del programma
  (in questo caso, il programma ha due istruzioni)

Il primo programma in C

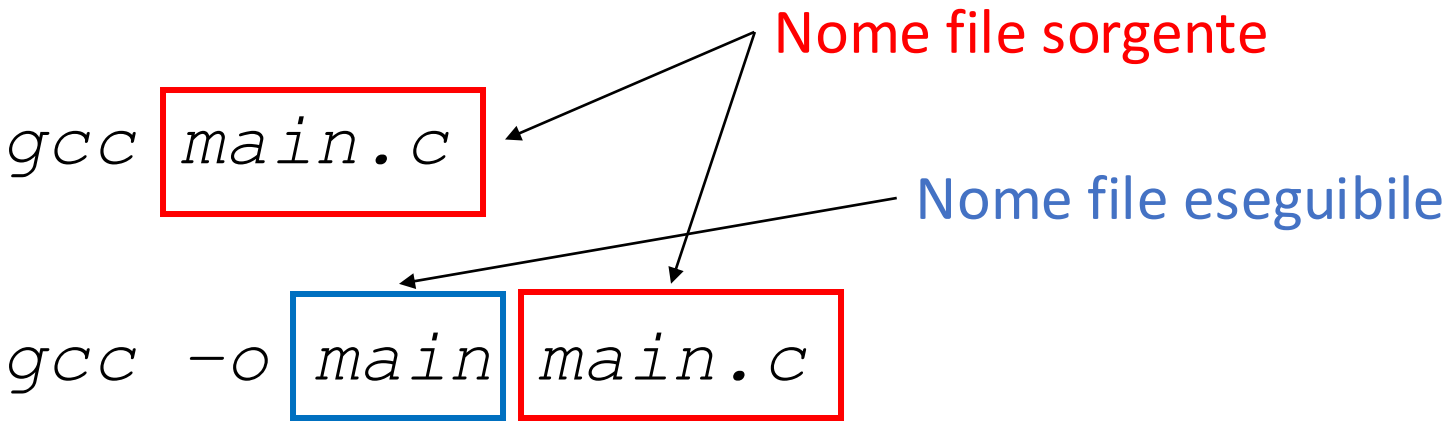
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

    printf("Hello World\n");
    return 0;
}
```

Istruzione che ~~ritorna~~ restituisce l'output della funzione, in questo caso torna l'intero 0

Compilazione

Per eseguire il programma occorre prima compilarlo.

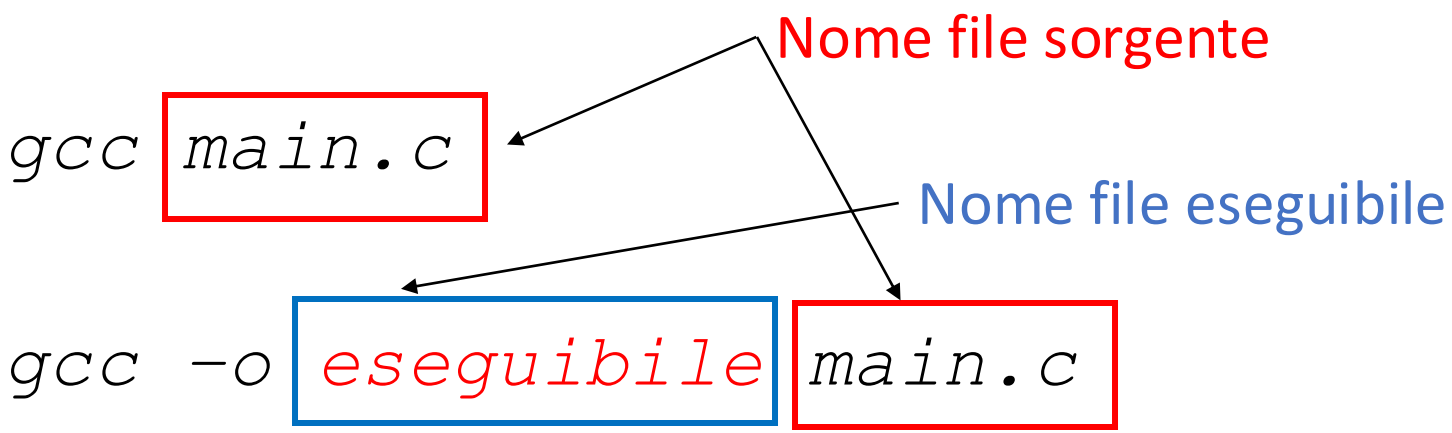
Ogni linguaggio ha il suo compilatore per il `C` si usa di norma il `gcc` (GNU/LINUX compiler)



Compilazione

Per eseguire il programma occorre prima compilarlo.

Ogni linguaggio ha il suo compilatore per il `C` si usa di norma il `gcc` (GNU/LINUX compiler)



Meglio dare all'eseguibile un nome diverso dal sorgente, se per sbaglio scrivo
`gcc -o es1.c es1.c`
sovrascrivo irrimediabilmente il mio sorgente, devo rifare da capo!

Precompilatore

Per capire cosa fa la precompilazione, eseguire:

`gcc main.c -E`

per averlo su file: `gcc main.c -E -o main.txt`

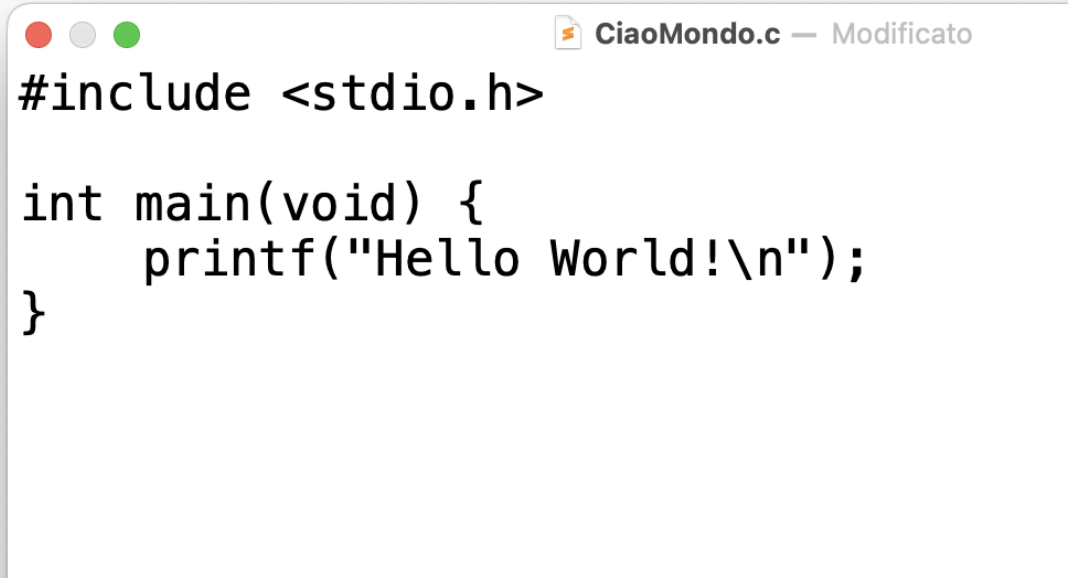
Il precompilatore ha «*esploso*» tutte le `include` che erano specificate

Setup in lab:

- Linux
- Pluma (editor – o un altro editor di testo minimale)
- Compilatore su terminale

Setup da remoto:

- https://www.onlinegdb.com/ (online via web browser)
- Replica di setup del lab su vostra macchina o virtual machine



Comandi utili per il usare il terminale:

- **`cd nomeCartella`** si sposta in una cartella di nome **`nomeCartella`**
- **`mkdir nomeCartella`** crea una cartella di nome **`nomeCartella`**
- **`pwd`** nome della cartella corrente
- **`ls`** mostra contenuto cartella
- **`cd ..`** Torna alla cartella precedente

Comandi utili per il compilare ed eseguire il vostro programma:

- (scrivete il vostro programma su di un editor di testo, salvatelo)
- entrate nella cartella dove è salvato **`programma.c`**
- **`gcc programma.c -o eseguibile`** compila e crea eseguibile (leggere output di compilazione per errori)
- **`./eseguibile`** esegue il programma (stampa a terminale il risultato)

Funzione `printf`

È la versione `C` della funzione write usata negli esempi in pseudocodice

È definita in `stdio.h`, che viene incluso nel `main`, e contiene il *prototipo della funzione*

Lavora su standard output (il monitor)

Per ora abbiamo visto la funzione prendere un parametro d'ingresso, ovvero una stringa costante

Tra i caratteri di questa stringa possono esserci dei caratteri di controllo chiamati *sequenze di escape*

ESEMPIO: cambiare il programma `main` rimuovendo la sequenza di escape "`\n`" dopo "`Hello`"

Funzione `printf`

E’ una funzione piuttosto articolata in realtà perché può avere anche altri parametri

Serve per esempio se voglio stampare a video un valore

Per esempio una *costante* numerica

```
printf(‘‘la rendita è di %d euro’’, 250);
```

Oppure il valore di una *variabile* chiamata `rendita`

```
printf(‘‘la rendita è di %d euro’’, rendita);
```

Il «`%`» indica un placeholder (segnaposto) per un valore il cui tipo è intero `(d)`,
ne definisce il formato con cui deve essere stampato a video

Funzione `printf` sequenze di escape

\a	Bell (speaker beeps)
\b	Backspace (non-erase)
\f	Form feed/clear screen
\n	New line
\r	Carriage Return
\t	Tab
\v	Vertical tab
\\	Backslash
\?	Question mark
\'	Single quote
\"	Double quote
\x*nn*	Hexadecimal character code *nn*
\o*nn*	Octal character code *nn*

Funzione `printf`

Una espressione matematica genera un valore

Esempio: 12+5 è un valore, se lo stampo avrò 17

Quindi può essere stampata come

```
printf(‘‘la rendita è di %d*%d=%d euro’’,50,5,50*5);
```

NOTA: le espressioni matematiche non sempre generano un valore che ci aspettiamo

Cosa succede in una divisione per zero?

Variabili, costanti e tipi

- Variabile: il **valore** assunto da una variabile può (**deve**) cambiare nel corso del programma
- Ad ogni variabile è associato un **tipo**
- La variabile può cambiare però solo all'interno dell'insieme di valori possibili definiti dal suo tipo
- Il C mette a disposizione dei tipi predefiniti detti **primitivi**, ma permette anche di definirne di nuovi
- Una **costante** non varia dopo che ha assunto un valore ma ha comunque associato un tipo

Tipi primitivi

- I tipi **`char` (carattere in codice `ASCII` – 8bit)** e **`int` (numero intero – 16bit)** sono detti tipi integrali.

- I loro valori sono interi e `char` è un sottotipo di `int`
    - i valori di `char` sono sottoinsieme dei valori di `int`

- L'unica differenza è come vengono interpretati questi valori, ma si tratta a tutti gli effetti di numeri
    - Reagiscono allo stesso modo alle operazioni aritmetiche

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	*[NULL]*
1	1	1	1	*[START OF HEADING]*
2	2	10	2	*[START OF TEXT]*
3	3	11	3	*[END OF TEXT]*
4	4	100	4	*[END OF TRANSMISSION]*
5	5	101	5	*[ENQUIRY]*
6	6	110	6	*[ACKNOWLEDGE]*
7	7	111	7	*[BELL]*
8	8	1000	10	*[BACKSPACE]*
9	9	1001	11	*[HORIZONTAL TAB]*
10	A	1010	12	*[LINE FEED]*
11	B	1011	13	*[VERTICAL TAB]*
12	C	1100	14	*[FORM FEED]*
13	D	1101	15	*[CARRIAGE RETURN]*
14	E	1110	16	*[SHIFT OUT]*
15	F	1111	17	*[SHIFT IN]*
16	10	10000	20	*[DATA LINK ESCAPE]*
17	11	10001	21	*[DEVICE CONTROL 1]*
18	12	10010	22	*[DEVICE CONTROL 2]*
19	13	10011	23	*[DEVICE CONTROL 3]*
20	14	10100	24	*[DEVICE CONTROL 4]*
21	15	10101	25	*[NEGATIVE ACKNOWLEDGE]*
22	16	10110	26	*[SYNCHRONOUS IDLE]*
23	17	10111	27	*[ENG OF TRANS. BLOCK]*
24	18	11000	30	*[CANCEL]*
25	19	11001	31	*[END OF MEDIUM]*
26	1A	11010	32	*[SUBSTITUTE]*
27	1B	11011	33	*[ESCAPE]*
28	1C	11100	34	*[FILE SEPARATOR]*
29	1D	11101	35	*[GROUP SEPARATOR]*
30	1E	11110	36	*[RECORD SEPARATOR]*
31	1F	11111	37	*[UNIT SEPARATOR]*
32	20	100000	40	*[SPACE]*
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	*[DEL]*

Tabella ASCII

Funzione `printf` sequenze di escape

Sequenze di *escape*

Negli esercizi di questa prima parte sperimenterete con l'output formattato di **`printf`**. Fate riferimento alla seguente tabella di sequenze escape del linguaggio C.

Escape	Nome	Effetto
`\n`	newline	va a capo
`\r`	carriage return	sposta il cursore a inizio riga
`\t`	horizontal tab	tabulazione orizzontale
`\v`	vertical tab	tabulazione verticale
`\b`	backspace	sposta il cursore indietro di un carattere
`\'`	single quote	visualizza un apice
`\"`	double quote	visualizza doppio apice
`\?`	question mark	visualizza punto interrogativo
`\`*ooo*	octal code	carattere in notazione ottale
`\x`*hh*	hexadecimal code	carattere in notazione esadecimale
`\u`*hhhh*	Unicode	carattere in notazione esadecimale Unicode

Alcune sequenze di escape saranno illustrate negli esercizi di questa dispensa.

Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_

Funzione `printf` sequenze di escape

Sequenze di *escape*

Negli esercizi di questa prima parte sperimenterete con l'output formattato di **`printf`**. Fate riferimento alla seguente tabella di sequenze escape del linguaggio C.

Escape	Nome	Effetto
`\n`	newline	va a capo
`\r`	carriage return	sposta il cursore a inizio riga
`\t`	horizontal tab	tabulazione orizzontale
`\v`	vertical tab	tabulazione verticale
`\b`	backspace	sposta il cursore indietro di un carattere
`\'`	single quote	visualizza un apice
`\"`	double quote	visualizza doppio apice
`\?`	question mark	visualizza punto interrogativo
`\`*ooo*	octal code	carattere in notazione ottale
`\x`*hh*	hexadecimal code	carattere in notazione esadecimale
`\u`*hhhh*	Unicode	carattere in notazione esadecimale Unicode

Alcune sequenze di escape saranno illustrate negli esercizi di questa dispensa.

Ad esempio `\101` stampa `A`, mentre `\x4C` stampa `L`

Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_

Funzione `printf` sequenze di escape

Sequenze di *escape*

Negli esercizi di questa prima parte sperimenterete con l'output formattato di **`printf`**. Fate riferimento alla seguente tabella di sequenze escape del linguaggio C.

Escape	Nome	Effetto
`\n`	newline	va a capo
`\r`	carriage return	sposta il cursore a inizio riga
`\t`	horizontal tab	tabulazione orizzontale
`\v`	vertical tab	tabulazione verticale
`\b`	backspace	sposta il cursore indietro di un carattere
`\'`	single quote	visualizza un apice
`\"`	double quote	visualizza doppio apice
`\?`	question mark	visualizza punto interrogativo
`\`*ooo*	octal code	carattere in notazione ottale
`\x`*hh*	hexadecimal code	carattere in notazione esadecimale
`\u`*hhhh*	Unicode	carattere in notazione esadecimale Unicode

Alcune sequenze di escape saranno illustrate negli esercizi di questa dispensa.

Ad esempio `\101` stampa `A`, mentre `\x4C` stampa `L`

Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_

Tipi primitivi

- I tipi **`char` (carattere in codice `ASCII` – 8bit)** e **`int` (numero intero – 16bit)** sono detti tipi integrali.

- I loro valori sono interi e `char` è un sottotipo di `int`
  - i valori di `char` sono sottoinsieme dei valori di `int`

- L'unica differenza è come vengono interpretati questi valori, ma si tratta a tutti gli effetti di numeri
  - Reagiscono allo stesso modo alle operazioni aritmetiche

- I tipi **`float` (numero reale – 32bit precisione singola)** e **`double` (numero reale – 32bit precisione doppia)** sono detti tipi reali. Hanno valori nel dominio dei numeri razionali rappresentati con espansione decimale

Costanti

- Le costanti si indicano nel sorgente come dei numeri es. 123 o -123
- Le costanti reali con la notazione con il punto es. 12.456
- Si può usare la notazione esponenziale es. 2.5E12
- Le costanti caratteri sono in sostanza dei numeri ma si possono scrivere anche usando gli ‘’ indicando il carattere es. ‘A’
- Verrà usata la tabella `ASCII` per convertire quel carattere in numero
- In `C`, per convenzione, quando dichiarate una costante dovete indicarla con lettere MAIUSCOLE (lettere minuscole → variabili, lettere MAIUSCOLE → costanti)

Operazioni sui caratteri

- Per comprendere come i caratteri vengano trattati effettivamente dal C usiamo ancora la funzione `printf`

- Sfruttiamo il carattere di conversione `%c` oltre che `%d` per visualizzare il carattere `ASCII` o il numero relativo a dei `char`

```
printf(‘‘il codice ASCII di %c è %d’’, ‘A’, ‘A’);
printf(‘‘il codice ASCII di %c è %d’’, ‘A’ +1, ‘B’);
```

Altri caratteri di conversione

***Table 7.1** Basic Printf Conversions*

Character	Argument type; Printed As
`d,i`	`int`; decimal number
`o`	`int`; unsigned octal number (without a leading zero)
`x,X`	`int`; unsigned hexadecimal number (without a leading `0x` or `0X`), using `abcdef` or `ABCDEF` for 10, ...,15.
`u`	`int`; unsigned decimal number
`c`	`int`; single character
`s`	`char *`; print characters from the string until a `'\0'` or the number of characters given by the precision.
`f`	`double`; `[-]`*m.dddddd*, where the number of *d*'s is given by the precision (default 6).
`e,E`	`double`; `[-]`*m.dddddd*`e+/-`*xx* or `[-]`*m.dddddd*`E+/-`*xx*, where the number of *d*'s is given by the precision (default 6).
`g,G`	`double`; use `%e` or `%E` if the exponent is less than -4 or greater than or equal to the precision; otherwise use `%f`. Trailing zeros and a trailing decimal point are not printed.
`p`	`void *`; pointer (implementation-dependent representation).
`%`	no argument is converted; print a %

Kerningham & Ritchie

Dichiarazione di variabili

- In `C` le variabili vanno **sempre** dichiarate prima di essere usate
- La dichiarazione prevede di esprimere il ***tipo*** il ***nome*** ed un eventuale valore di inizializzazione attraverso un assegnamento

```
int  somma=0;
```

- L'assegnamento in C è codificato dal simbolo "="
- Assegna il valore che sta alla destra al contenitore (variabile di norma) che sta alla sinistra

Proviamo ad alterare l'esempio delle `printf` *dichiarando delle variabili che poi stamperemo…*

Overflow e Underflow

- Essendo i numeri rappresentati in uno spazio finito di memoria con un numero prefissato di bit, può succedere che una operazione porti ad un numero non rappresentabile da quella quantità di bit
- Es. se i numeri interi (`int`) fossero a 3 bit

```
int a=8
a=a+1 /*con interi a 3 bit porta a sforare il massimo numero rappresentabile*/
```

1	2	3		
0	0	0	→	0
0	0	1	→	1
0	1	0	→	2
0	1	1	→	3
1	0	0	→	4
1	0	1	→	5
1	1	0	→	6
1	1	1	→	7

- Si parla di **overflow**
- The number glitch that can lead to catastrophe - BBC Future
- **Underflow** e l'incapacità di rappresentare un valore reale troppo piccolo, che viene quindi rappresentato come zero.

Limiti

Per il controllo di questi limiti esistono delle costanti definite da delle librerie

In particolare:

`INT_MAX, INT_MIN, CHAR_MAX, CHAR_MIN` in `limits.h`

`FLT_MAX, FLT_MIN, DBL_MAX, DBL_MIN` in `float.h`

Input dei dati

E’ un argomento molto complesso da affrontare al momento – ne avrete consapevolezza più avanti.

Per fare esercizio assumeremo di utilizzare delle funzioni senza chiederci molto come mai sono fatte in un certo modo

- `getchar()`: leggere un carattere
- `scanf()`: legge `char int float` e `double`

Esempio:

```
char c;
c=getchar();
```

In realtà quando si usa `getchar` si preme anche invio dopo il carattere inserito da tastiera quindi i caratteri sono due (\n è un carattere)

La selezione

Costrutto di selezione in C

```
if (<espressione>) <blocco then>

if (<espressione>) <blocco then> else <blocco else>
```

- Blocco di codice: se è costituito da più di una istruzione è definito da { }
  Se è di una sola istruzione, si possono omettere le { }

- Di <espressione> si valuta il valore logico (vero o falso)

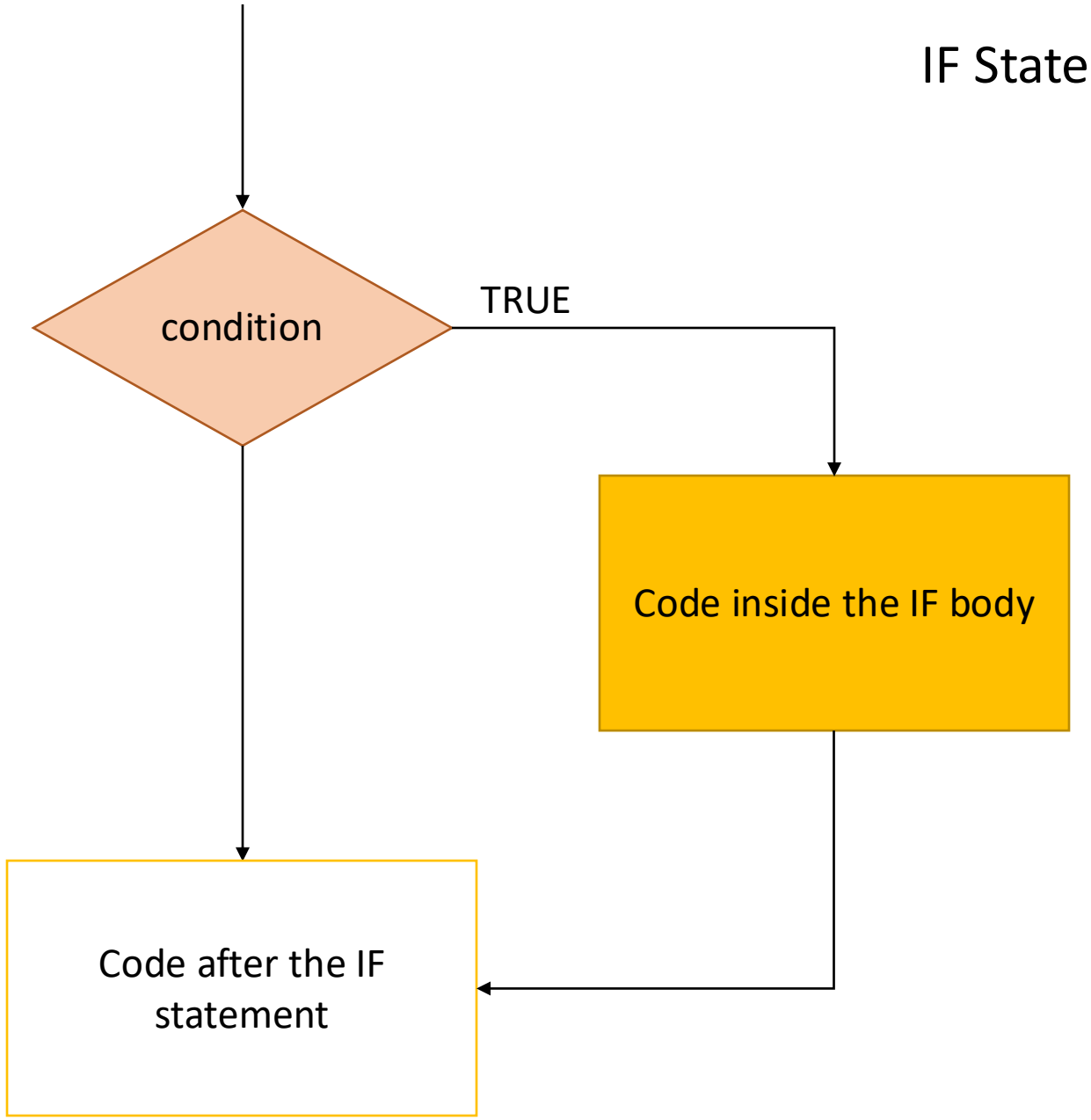
- Per C, una espressione, è per convenzione lavora un numero
  - (il tipo boolean VERO/ FALSO è stato introdotto dopo nel C99)

- Per il C il numero 0 è falso e qualsiasi altro numero è vero (per convenzione si intende l'1)

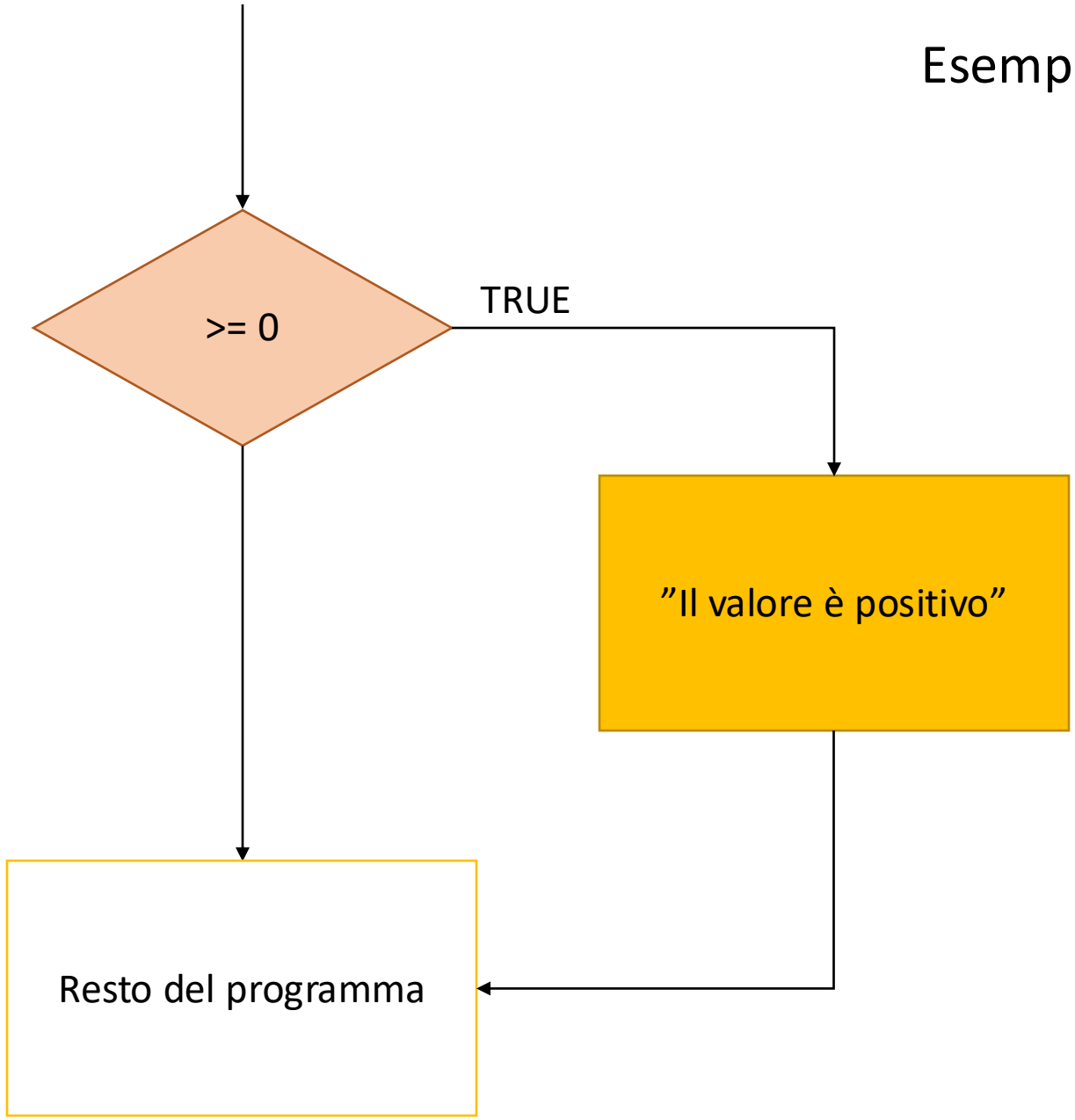
Espressione: una prima approssimazione

- *E' una sequenza di simboli a cui è associato un valore*
- Per il momento ci accontentiamo di supporre che siano valide le regole della matematica per ottenere il valore dell'espressione
- Abbiamo operatori relazionali (<,>,<=, >=, ==, !=)
- Operatori aritmetici (+,-,*,/,%)
- Operatori logici: (!, &&, ||, …)
- Si possono usare le parentesi tonde per chiarire le precedenze anche se esiste un ordine di precedenza

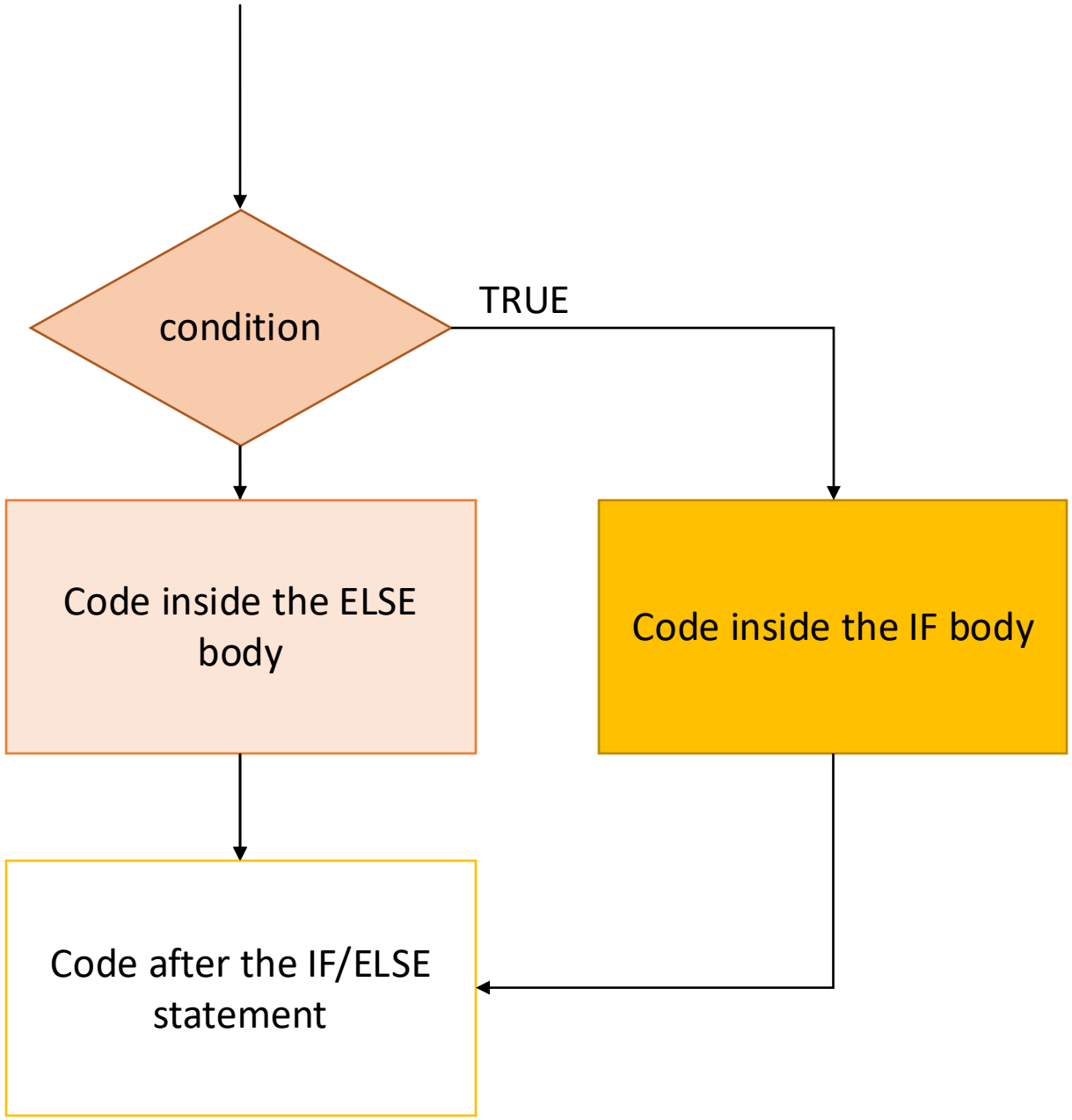
IF Statement



Esempio: check positivo

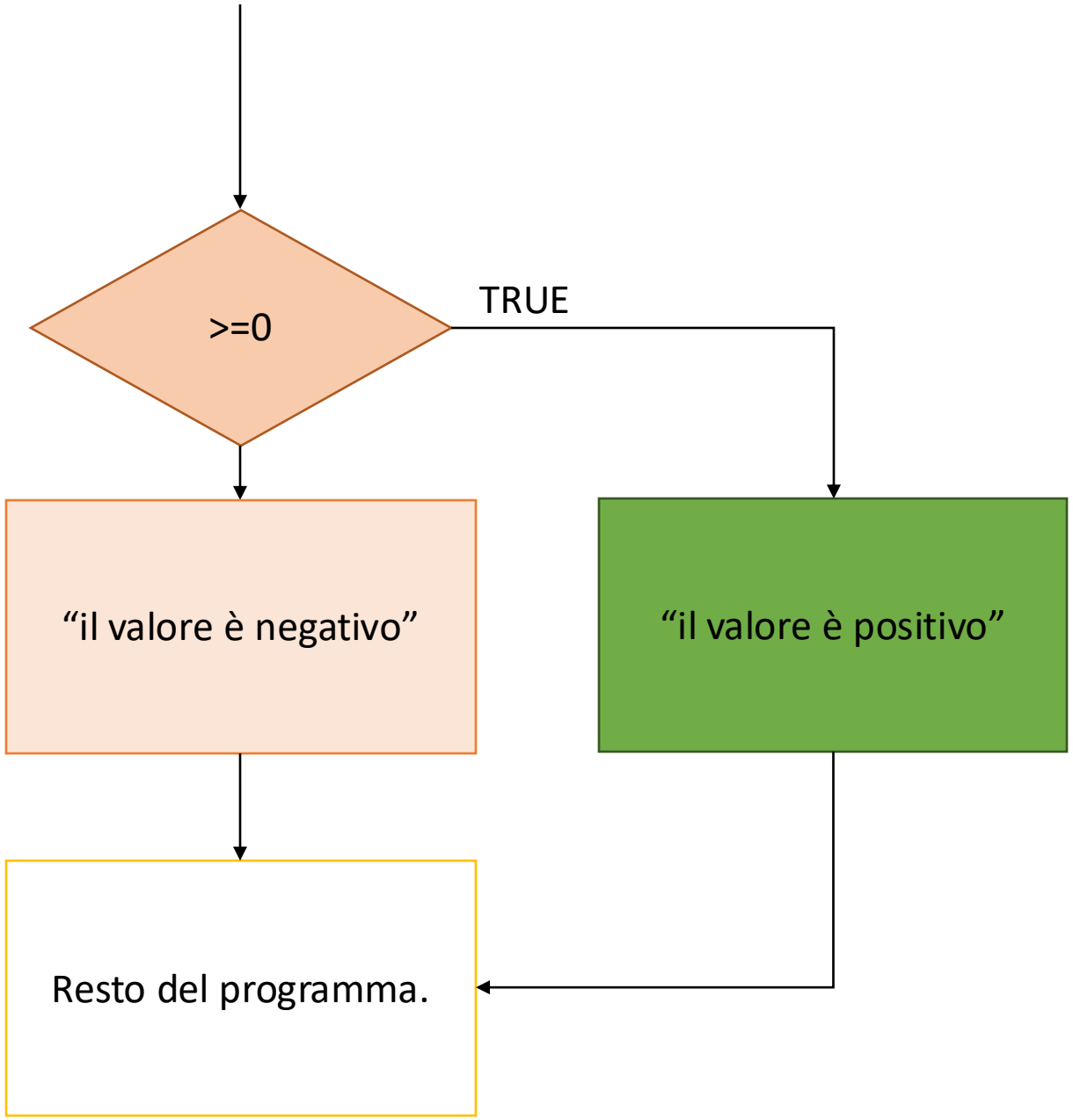


IF Statement




```
if (condition)
        // blocco di codice
        // condizione verificata
        // (Obbligatorio)
else
        // blocco di codice
        // condizione NON verificata
        // (Opzionale)
return 0;
}
```

Esempio: check positivo




```
if (valore >= 0)
        printf(“Il valore è positivo\n”);
else
        printf(“Il valore è negativo\n”);
```

Come impostare un programma

Quando scrivete un nuovo programma (di brevi dimensioni, come quelli oggetto di questo corso), cercate di dividere logicamente le parti principali del programma (dove possibile)

1. Dichiarazione delle variabili
2. Ottenimento degli input (es, `scanf`)
3. Fase principale – calcolo
4. Restituzione degli output (es, `printf`)
5. Conclusione: `return`

Inoltre

1. Cercate di scomporre il programma in più funzioni/sottoprogrammi
2. Evitate la duplicazione del codice
   (se dovete duplicare il codice in più parti, è un buon indizio che potete usare una funzione al posto del codice duplicato)

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
	/* Questo è un commento, può essere su più linee */
	// Questo è un commento che sta su una linea sola
	int numero;

	printf("Inserisci un valore intero\n");
	scanf("%d",&numero);

	if (numero < 0)
		numero = -numero;

	printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
	return 0;
}
```

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Commentare il codice ne aumenta la *leggibilità*, e vi aiuta a trovare eventuali errori

Permette di specificare assunzioni fatte

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Fase iniziale/1:
dichiaro le variabili

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Fase iniziale/2:
inizializzo le variabili

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Quando utilizzate la scanf, anteponete al nome della variabile una “&”.
Questo perché alla scanf dovete fornire un *puntatore* alla variabile (concetto che vedremo più avanti)

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Parte principale del programma

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Indentare il codice ci fa capire che l'istruzione è parte di questo `if`

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Parte conclusiva:
restituisco il risultato

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

/*Scrivere un programma che riceve in ingresso un numero intero, inserito da utente,
e ne stampa il valore assoluto*/
int main(){
        /* Questo è un commento, può essere su più linee */
        // Questo è un commento che sta su una linea sola
        int numero;

        printf("Inserisci un valore intero\n");
        scanf("%d",&numero);

        if (numero < 0)
                numero = -numero;

        printf("Il valore assoluto del numero che hai inserito è %d.\n", numero);
        return 0;
}
```

Il programma termina segnalando che è andato tutto bene

Esercizi

1. Si scriva un programma che produca in uscita la scritta: *Ciao, Mondo*
2. Si scriva un programma che produca la scritta
   Ciao,
   Mondo
3. Si scriva un programma che visualizzi sul terminale tre colonne di dati i cui campi siano allineati a sinistra, come in questo esempio

```
Lun  10:40    Architetture
Mar  12:10    Basi di dati
Mer  13:30    Programmazione
```

Esercizi

4. Si scriva un programma che produca in uscita la scritta “`Ciao`” usando una sola chiamata a printf e le sequenze di escape relative ai codici ASCII (\ooo)

5. Appurare i range dei tipi visti a lezione e disponibili su `limits.h`. Le costanti da verificare sono:
   `CHAR_MAX, CHAR_MIN, INT_MAX, e INT_MIN`

6. Fare la stessa cosa per i tipi razionali includendo `float.h` (attenzione al selettore del formato scelto per la stampa)

Esercizi

7. Scrivere un programma che assegni `INT_MAX` ad una variabile intera e poi la incrementi di 1. Visualizzarne il risultato. Cosa è successo?

8. Scrivete un programma che assegni alla variabile `num` di tipo float il valore `FLT_MIN` definito `in float.h`. Visualizzate il valore di float. Dividete num per una costante reale grande per esempio `FLT_MAX`. Visualizzate di nuovo il valore di `num`.

Esercizi

9. Si scriva un programma che chieda all'utente di inserire un valore di tipo `int`, e lo riscriva subito dopo sul terminale. Il programma prosegue facendo la stessa cosa per un valore di tipo double.

10. Si scriva un programma che chieda all'utente di inserire due valori di tipo `int`, e visualizzi poi la loro somma, il valore del primo meno il secondo, il loro prodotto, il valore del primo diviso il secondo (divisione intera /), e il valore del resto della divisione del primo per il secondo (operatore modulo ''%'').

11. Fare lo stesso programma usando il tipo double

12. Si scriva un programma che dica se l'esito di una moltiplicazione tra due numeri interi è positivo o negativo e, successivamente, ne stampi il risultato.