

ELEMENTI DI PROGRAMMAZIONE E GESTIONE DATI

Ing. Giancarlo Degani

2025

INDICE

1. [Welcome to Slidev](#)
2. [Indice](#)
3. [1 - Fondamenti di Informatica](#)
4. [2 - Introduzione al “C”](#)
5. [3 - Istruzioni condizionali e cicli](#)
6. [4 - Vettori e matrici](#)
7. [7 - Fondamenti di Informatica](#)

1 - FONDAMENTI DI INFORMATICA

Ing. Giancarlo Degani

IL CORSO

- **Programma:**
 - Prima parte: cultura informatica
 - Seconda parte: programmazione in linguaggio 'C'
- **Durata:** 30 ore
- **Verifica:** test finale con domande a risposta multipla

STRUMENTI

- Dispense delle lezioni
- Classroom
- CLion

CLION

- CLion Download
- Educational license

The screenshot shows the CLion IDE interface. The top bar displays the project name "LLVM" and branch "master". The left sidebar has a "Structure" tab selected, showing a tree view of the project's source code. The main editor area shows a file named "GlobalCompilationDatabase.h" with several functions listed. A specific function, "getFallbackCommand(PathRef) const", is highlighted with a blue selection bar. The bottom status bar indicates the current file path: "llvm-project > clang-tools-extra > clangd > GlobalCompilationDatabase.h".

```
18 namespaces
19 namespaces
58 toolin
59 @↓ ↵ Global
60 std
61 // C
62 // L
63 if
64 Arg
65 too
66
67
68 Cmd
69 retu
70 }
71 }
72 }
73 Direct
N clang > N clang
```

Structure

- o clang
- o clangd
- o <unnamed>
- o clangd
- o adjustArguments(CompileCommand)
- o getStandardResourceDir() : string
- o getFallbackClangPath() : string
- o GlobalCompilationDatabase
- o getFallbackCommand(PathRef) const
- > DirectoryBasedGlobalCompilationDatabase
- > OverlayCDB

GlobalCompilationDatabase.h

18 namespaces
19 namespaces
58 toolin
59 @↓ ↵ Global
60 std
61 // C
62 // L
63 if
64 Arg
65 too
66
67
68 Cmd
69 retu
70 }
71 }
72 }
73 Direct
N clang > N clang

llvm-project > clang-tools-extra > clangd > GlobalCompilationDatabase.h

LICENZA PER STUDENTI

- Creare un account con l'email @itsmeccatronico.it e richiedere una licenza educational
- Scaricare ed installare CLion
- Aprire il programma e registrare la licenza inserendo le credenziali dell'account in **Help > Register**



JetBrains Product Pack for St

Download ▾

Licensed to:

License restriction: For educational use only

Valid through: February 27, 2025

Following products included:

- [Aqua](#)
- [dotMemory](#)
- [PyCharm](#)
- [RustRover](#)
- [CLion](#)
- [dotTrace](#)
- [ReSharper](#)
- [WebStorm](#)

After downloading and installing the software, simply log in to your Account.

RIFERIMENTI

- <https://cplusplus.com/reference/clibrary/>
- https://en.wikibooks.org/wiki/C_Programming
- <https://archive.org/details/Apress.Beginning.C.5th.Edition.2013>

ALGORITMO

- Il termine deriva dalla trascrizione latina del nome del matematico persiano al-Khwarizmi, vissuto nel IX secolo d.C. È considerato uno dei primi autori ad aver fatto riferimento a questo concetto, scrivendo il libro “Regole di ripristino e riduzione”.
- In matematica e informatica, un algoritmo è la specificazione di una sequenza finita di operazioni (dette anche istruzioni) che consente di risolvere una classe di problemi specifici o di calcolare il risultato di un'espressione matematica.

PROPRIETÀ DI UN ALGORITMO

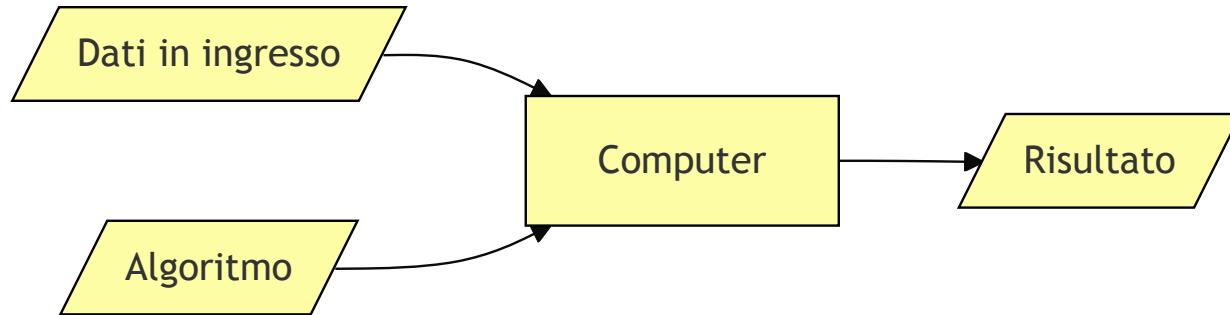
- **Finito:** costituito da un numero finito di istruzioni.
- **Deterministico:** partendo dagli stessi dati di ingresso, ottengo gli stessi risultati.
- **Generale:** applicabile a tutti i problemi della classe a cui si riferisce.
 - Ad esempio, l'algoritmo per il calcolo dell'area di un rettangolo deve essere applicabile a tutti i rettangoli.
- **Eseguibile:** esiste un esecutore in grado di eseguire tutte le istruzioni in un tempo finito.

CARATTERISTICHE DEGLI ESECUTORI

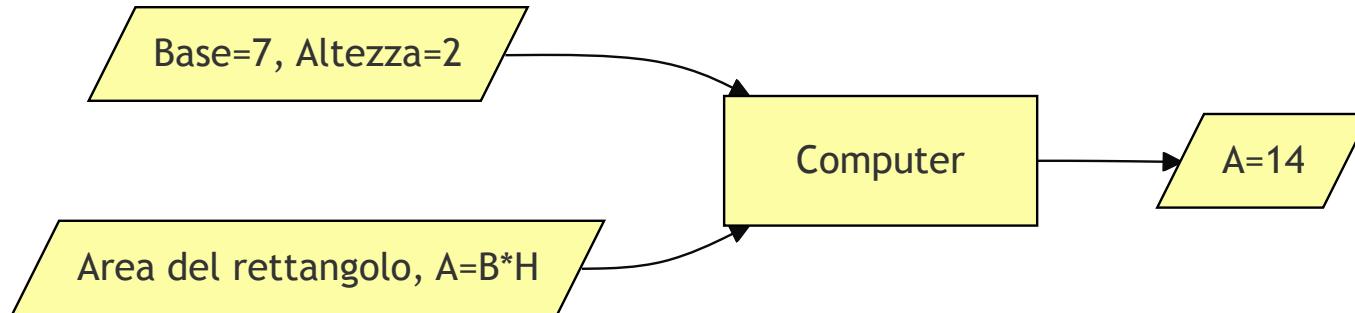
- Il linguaggio che possono comprendere (italiano, inglese, C, TypeScript, ecc.)
- Le azioni che possono eseguire
- Le regole che associano alle istruzioni fornite le azioni da eseguire



CALCOLATORE COME ESECUTORE



ESEMPIO



STRUTTURE DATI

- I contenitori usati per contenere i dati in ingresso sono detti variabili.
- Le variabili:
 - Hanno un nome o identificatore.
 - Possono essere usate come parte di una istruzione.
 - Possono essere caratterizzate dal tipo di dato che contengono
 - es. variabili per numeri interi, numeri reali, sequenze di numeri, etc...

ISTRUZIONI DI ASSEGNAZIONE

- Consentono di inserire un valore all'interno di una variabile.
- Cambiano a seconda del linguaggio utilizzato, ma solitamente usano l'operatore “=”.
 - $x=5$ assegna il valore 5 alla variabile x.
 - $y=x$ assegna il valore contenuto nella variabile x alla variabile y.

ESPRESSIONI ARITMETICHE

Sono costituite da:

- Operandi: variabili, costanti, espressioni aritmetiche
- Operatori: addizione '+', sottrazione '-', moltiplicazione '*', divisione intera '/', resto o modulo '%'
- Parentesi: per definire l'ordine con cui vengono elaborate
- Risultato: un numero

ESEMPI

X = 5 Assegna alla variabile x il valore 5

X = 5+3 Assegna ad x il valore 8

Y = 5%3 Assegna ad y il valore 2

X = X *3 Assegna ad x il valore precedente moltiplicato per 3

ESPRESSIONI RELAZIONALI

Sono costituite da:

- Operandi: variabili, costanti, espressioni.
- Operatori: uguaglianza ==, disuguaglianza !=, maggiore di >, minore di <.
- Parentesi: per definire l'ordine con cui vengono elaborate.
- Risultato: vero o falso.

ESEMPI

X = 5 Assegna alla variabile x il valore 5

X == 5 Vero

X != 5 Falso

X > 0 Vero

ESPRESSIONI LOGICHE

Sono costituite da:

- Operandi: variabili, costanti, espressioni
- Operatori: somma logica (OR), moltiplicazione logica (AND), negazione (NOT)
- Parentesi: per definire l'ordine con cui vengono elaborate
- Risultato: un valore logico, vero o falso

OPERATORI LOGICI

Negazione

A	NOT A
0	1
1	0

Moltiplicazione

A	B	A AND B
0	0	0
0	1	0
1	0	0

OPERATORI LOGICI

Somma

A	B	$A \text{ OR } B$
---	---	-------------------

0	0	0
---	---	---

0	1	1
---	---	---

1	0	1
---	---	---

1	1	1
---	---	---

Disuguaglianza

A	B	$A \text{ XOR } B$
---	---	--------------------

0	0	0
---	---	---

0	1	1
---	---	---

1	0	1
---	---	---

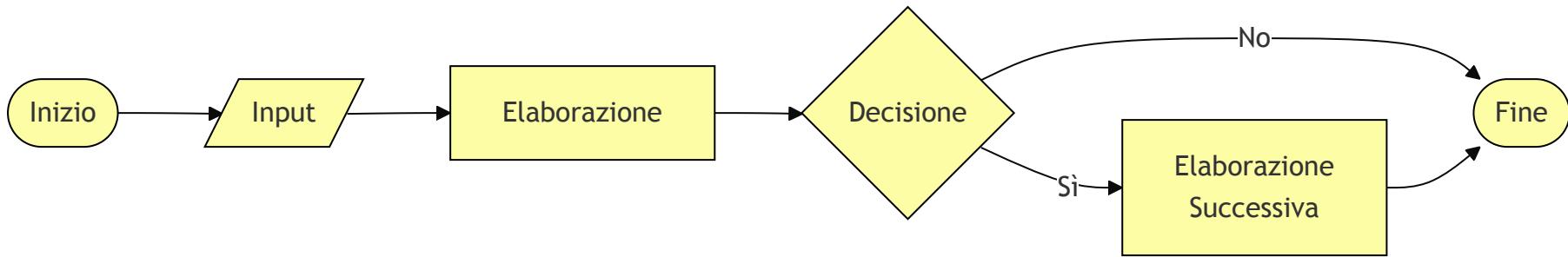
1	1	0
---	---	---

DIAGRAMMA DI FLUSSO/FLOW CHART

- Consentono di rappresentare visivamente un algoritmo.
- Sono indipendenti dal linguaggio di programmazione.
- Esistono diversi standard grafici.

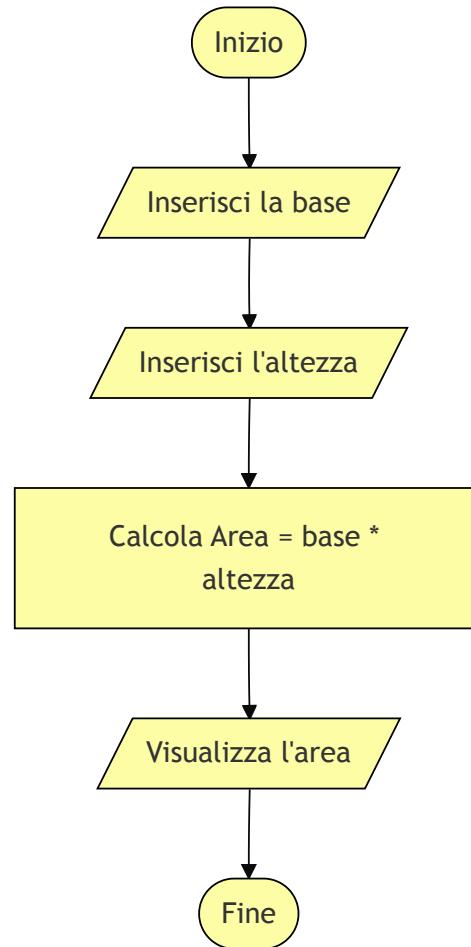
ANSI/ISO Shape	Name	Description
	Flowline (Arrowhead) ^[17]	Shows the process's order of operation. A line coming from one symbol and pointing at another. ^[16] Arrowheads are added if the flow is not the standard top-to-bottom, left-to right. ^[17]
	Terminal ^[16]	Indicates the beginning and ending of a program or sub-process. Represented as a stadium, ^[16] oval or rounded (fillet) rectangle. They usually contain the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit inquiry" or "receive product".
	Process ^[17]	Represents a set of operations that changes value, form, or location of data. Represented as a rectangle. ^[17]
	Decision ^[17]	Shows a conditional operation that determines which one of the two paths the program will take. ^[16] The operation is commonly a yes/no question or true/false test. Represented as a diamond (rhombus). ^[17]
	Input/Output ^[17]	Indicates the process of inputting and outputting data, ^[17] as in entering data or displaying results. Represented as a rhomboid. ^[16]
	Annotation ^[16] (Comment) ^[17]	Indicating additional information about a step in the program. Represented as an open rectangle with a dashed or solid line connecting it to the corresponding symbol in the flowchart. ^[17]
	Predefined Process ^[16]	Shows named process which is defined elsewhere. Represented as a rectangle with double-struck vertical edges. ^[16]
	On-page Connector ^[16]	Pairs of labeled connectors replace long or confusing lines on a flowchart page. Represented by a small circle with a letter inside. ^{[16][20]}
	Off-page Connector ^[16]	A labeled connector for use when the target is on another page. Represented as a home plate-shaped pentagon. ^{[16][20]}

ESEMPIO DI FLOWCHART



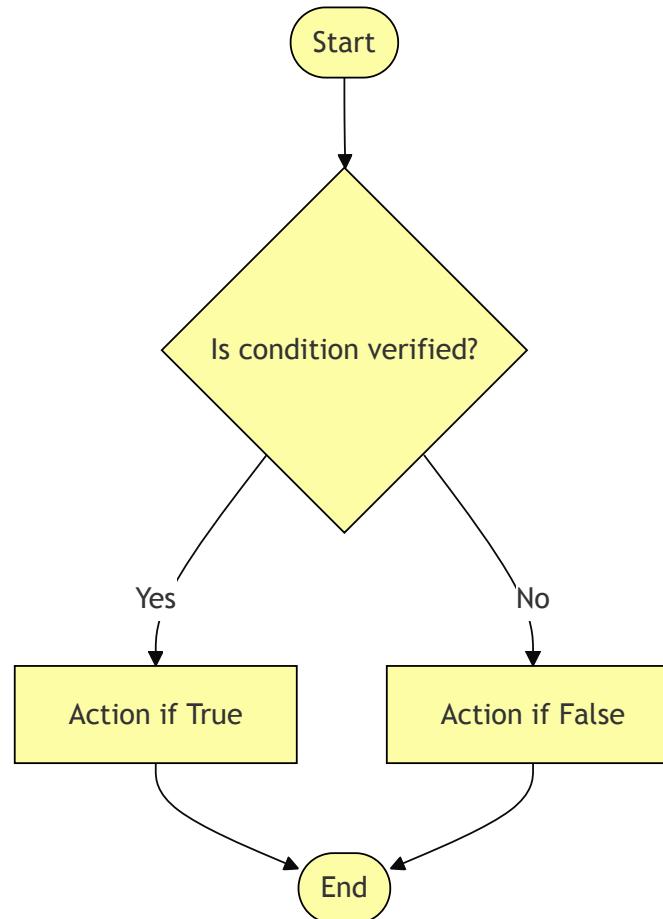
ESEMPIO: AREA DI UN RETTANGOLO

- Calcolo dell'area di un rettangolo
- Input: base ed altezza
- Algoritmo: $\text{Area} = \text{base} * \text{altezza}$



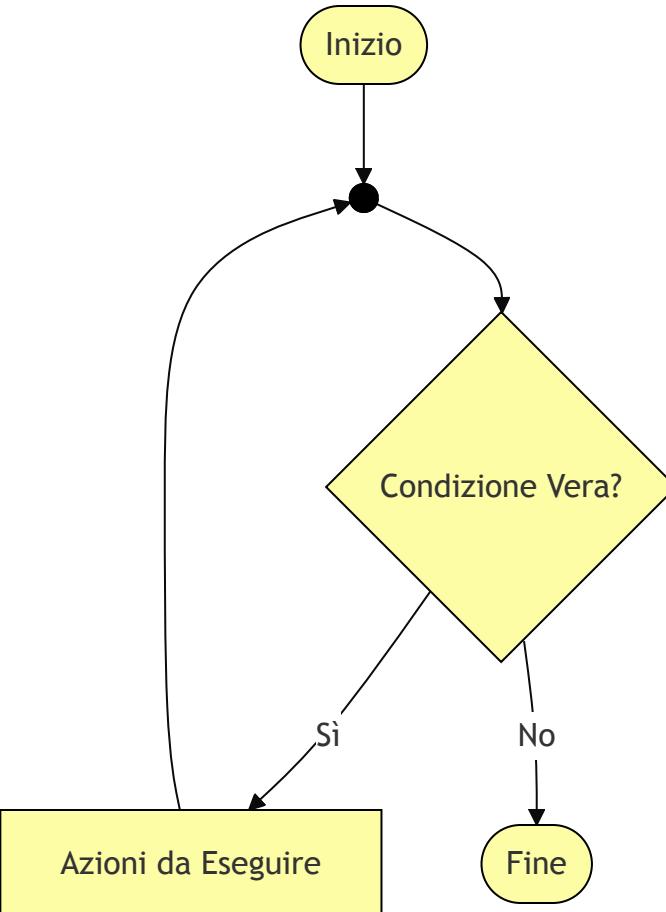
SELEZIONE - IF THEN ELSE

- Viene valutata una condizione.
- Se la condizione è vera,
l'elaborazione prosegue con il ramo
di sinistra.
- Se la condizione è falsa,
l'elaborazione prosegue con il ramo
di destra.



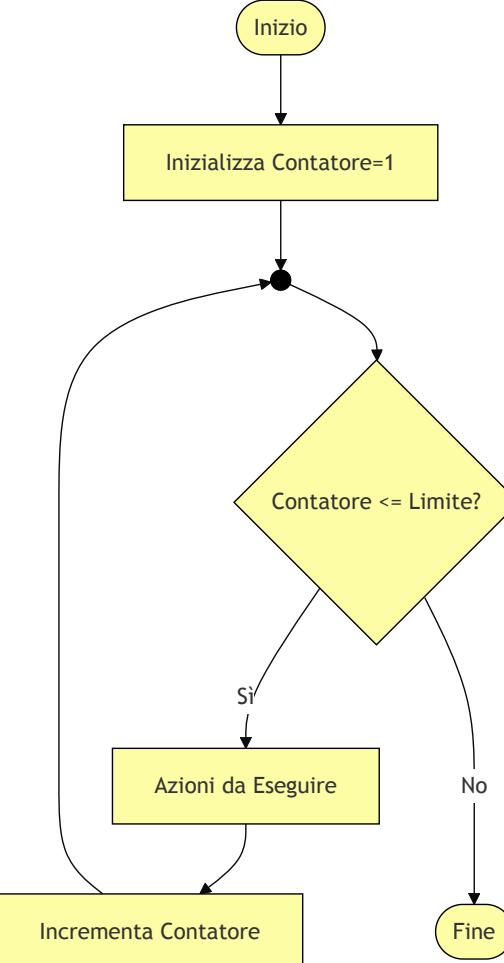
CICLO WHILE

- Viene valutata una condizione.
- Se la condizione è vera, viene eseguita l'azione e poi viene rivalutata la condizione.
- Si esce dal ciclo quando la condizione diventa falsa.



CICLO FOR

- Viene valutata la condizione e, se vera, si esegue l'azione.
- L'azione viene eseguita un numero finito di volte.



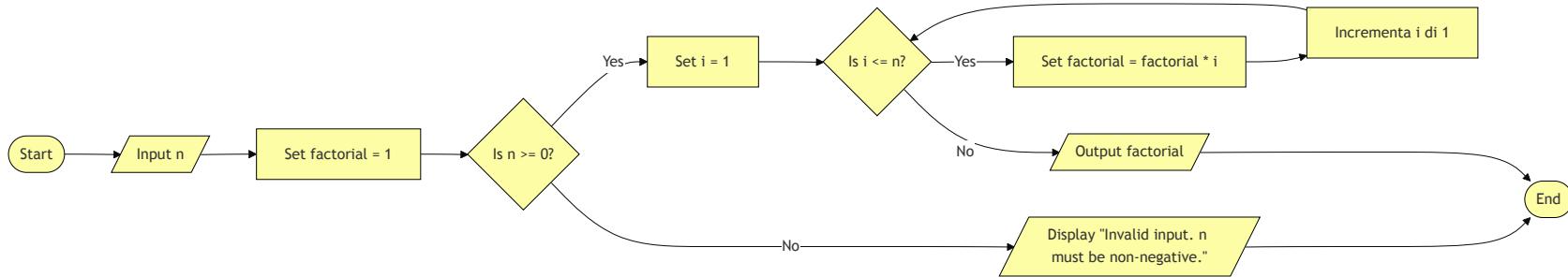
ESEMPIO: CALCOLO DEL FATTORIALE

Dato un numero intero, calcolarne il fattoriale.

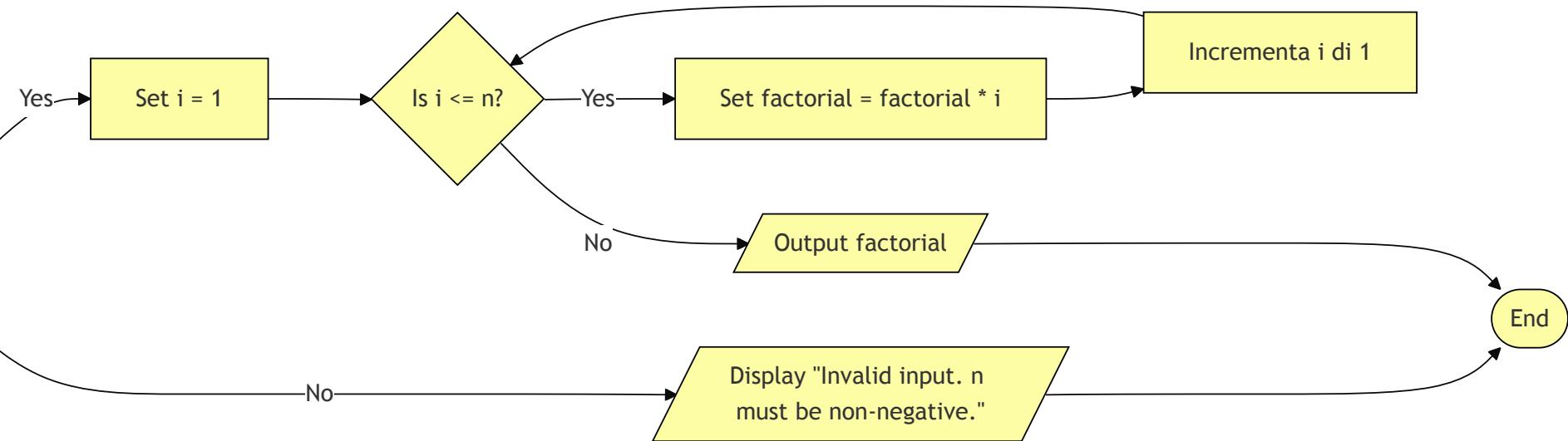
- Input: un numero intero maggiore o uguale a zero
- Algoritmo: moltiplico n per tutti i numeri minori di n, fino a 2

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1) \times \cdots \times 2 \times 1 & \text{se } n > 0 \end{cases}$$

ESEMPIO: CALCOLO DEL FATTORIALE



ESEMPIO: CALCOLO DEL FATTORIALE



ESERCIZIO

Rappresentare un algoritmo per il calcolo del costo di un prodotto che soddisfi i seguenti requisiti:

- Input: costo unitario, quantità acquistata
- Se il numero di elementi acquistati è superiore a 10, applicare uno sconto del 20%
- Output: costo totale

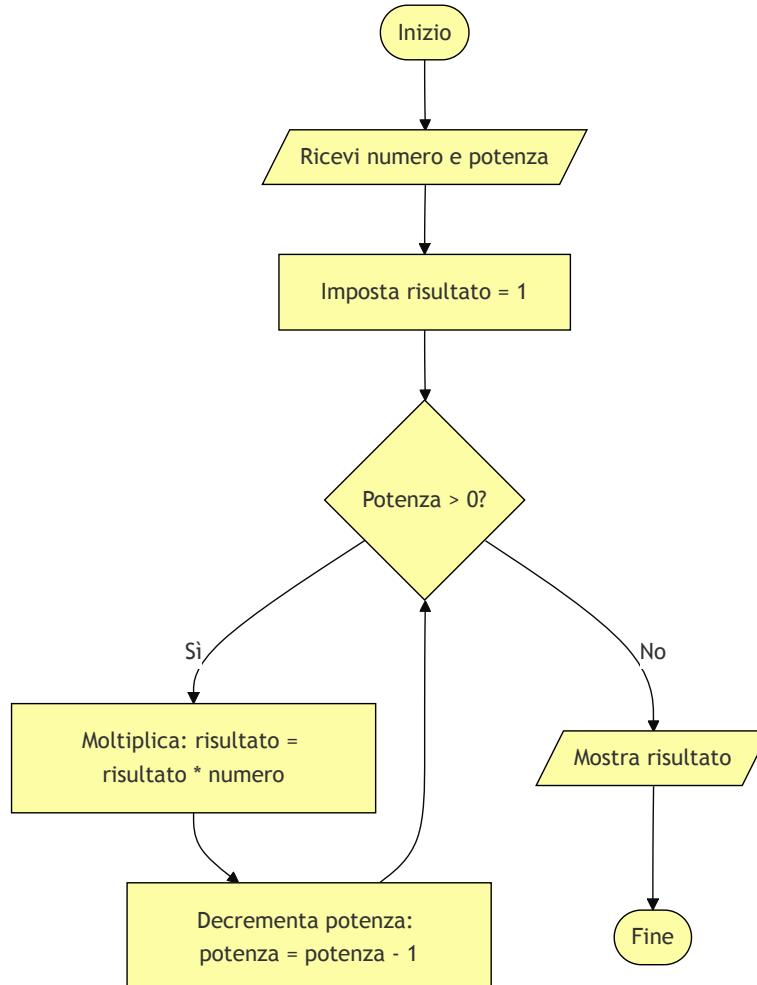
ESERCIZIO

Rappresentare un algoritmo per il calcolo della potenza n-esima di un numero intero che soddisfi i seguenti requisiti:

- Input: numero, potenza
- Output: numero^{potenza}
- Usare solo le operazioni elementari: +, -, *, /

FLOW-CHART

Flowchart della soluzione



CODIFICHE NUMERICHE

Sistema	Base	Simboli
Binario	2	{0,1}
Ottale	8	{0,1,2,3,4,5,6,7}
Decimale	10	{0,1,2,3,4,5,6,7,8,9}
Esadecimale	16	{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

ESEMPI

Decimale	Binario	Esadecimale
2	10	2
10	1010	A
16	1 0000	10
255	1111 1111	FF

CODIFICA NUMERICA POSIZIONALE

$$n = \sum_{0 < i < m} r_j \cdot b^j$$

$$1024 = 1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 = 1024_{10}$$

$$10 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 1010_2$$

$$\begin{aligned} 1024_{10} = & 1 \times 2^{10} + 0 \times 2^9 + 0 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 \\ & + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 10000000000_2 \end{aligned}$$

CONVERSIONE IN BINARIO

- Per convertire un numero da base decimale a base binaria si utilizzano le operazioni di divisione intera e modulo (quoziente e resto):
 - Divido il numero per 2 e salvo il resto della divisione.
 - Ripeto fino a che il quoziente diventa zero.

$$1972_{10} = 111;1011;0111_2$$

N div B	N mod B	
1972 / 2	986	0 ← LSB
986 / 2	493	0
493 / 2	246	1
...
15 / 2	7	1
7 / 2	3	1
3 / 2	1	1

CONVERSIONE IN ESADECIMALE

- La stessa procedura usata per la conversione in binario può essere applicata anche alla conversione in base 16.
- Considerato che 16 è una potenza di 2, solitamente si converte prima in binario e poi si raggruppano i bit in gruppi di quattro ($16 = 2^4$).

$$255_{10} = 1111; 1111_2 = FF_{16}$$

UNITÀ DI MISURA

- In binario, l'unità di misura fondamentale è l'informazione rappresentabile con una sola cifra binaria, zero o uno, detta bit (Binary digit).
- Una sequenza di 8 bit è definita Byte.

Multipli	Valore	Valore
Kilobyte - KB	2^{10}	1 024
Megabyte - MB	$(2^{10})^2$	1 048 576
Gigabyte - GB	$(2^{10})^3$	1 073 741 824
Terabyte - TB	$(2^{10})^4$	1 099 511 627 776

RAPPRESENTAZIONE DI INTERI IN MODULO E SEGNO

- Supponiamo di avere un registro di 4 byte = 32 bit.
- I bit sono convenzionalmente numerati da 0 a 31 da destra verso sinistra, dove un indice maggiore corrisponde a un peso maggiore.
- Si utilizza il primo bit a sinistra per rappresentare il segno:
 - 0: numero positivo
 - 1: numero negativo
- I bit da 0 a 30 rappresentano il valore assoluto del numero.

RAPPRESENTAZIONE DI INTERI IN MODULO E SEGNO

Si possono rappresentare i seguenti numeri:

- Positivi: da +0 a $+2^{31}-1 = 2.147.483.647$
- Negativi: da -0 a $-(2^{31}-1) = -2.147.483.647$

Cifra	1	0	1	...	0	1	1	0
Indice	31	30	29	...	3	2	1	0

IL COMPLEMENTO A DUE

- Metodo per rappresentare numeri interi con segno nei computer.
- Calcolo del complemento a due:
 - Rappresentare il numero in forma binaria: 0000 0101 (5).
 - Invertire tutti i bit (sostituendo 0 con 1 e viceversa): 1111 1010
 - Ovvero eseguire il complemento a uno).
 - Aggiungere il valore 1: 1111 1011 (che rappresenta -5).

RAPPRESENTAZIONE DI INTERI IN COMPLEMENTO A 2

- Nella codifica in complemento a due:
 - I numeri positivi sono rappresentati in modulo e segno.
 - I numeri negativi sono rappresentati in complemento a due.
- Con n bit si possono rappresentare numeri da -2^{n-1} a $2^{n-1}-1$.
- Con 8 bit, ad esempio, si possono rappresentare i numeri da -128 a +127.

CODIFICA DI CARATTERI

- La prima codifica utilizzata fu lo standard ASCII (American Standard Code for Information Technology).
- Prevedeva l'uso di 7 bit per rappresentare caratteri alfabetici, simboli grafici e alcuni caratteri speciali.
- Successivamente fu estesa a 8 bit.

Binario	Oct	Dec	Hex	Glifo	Binario	Oct	Dec	Hex	Glifo	Binario	Oct	Dec	Hex	Glifo
010 0000	040	32	20	Spazio	100 0000	100	64	40	@	110 0000	140	96	60	`
010 0001	041	33	21	!	100 0001	101	65	41	A	110 0001	141	97	61	a
010 0010	042	34	22	"	100 0010	102	66	42	B	110 0010	142	98	62	b
010 0011	043	35	23	#	100 0011	103	67	43	C	110 0011	143	99	63	c
010 0100	044	36	24	\$	100 0100	104	68	44	D	110 0100	144	100	64	d
010 0101	045	37	25	%	100 0101	105	69	45	E	110 0101	145	101	65	e
010 0110	046	38	26	&	100 0110	106	70	46	F	110 0110	146	102	66	f
010 0111	047	39	27	'	100 0111	107	71	47	G	110 0111	147	103	67	g
010 1000	050	40	28	(100 1000	110	72	48	H	110 1000	150	104	68	h
010 1001	051	41	29)	100 1001	111	73	49	I	110 1001	151	105	69	i

UNICODE

- Unicode è uno standard per la rappresentazione e gestione di testi di ogni lingua e simboli utilizzati in tutto il mondo. È stato creato per risolvere i problemi di compatibilità tra i diversi sistemi di codifica (come ASCII o Latin-1).
- Ogni carattere in Unicode è identificato da un numero unico chiamato code-point. Questi numeri possono essere rappresentati nel computer con varie codifiche, tra cui le più comuni sono UTF-8, UTF-16 e UTF-32.
- Supporta anche la rappresentazione di alfabeti complessi come il cinese, il giapponese e il coreano.

UNICODE

- Gestisce simboli matematici, emoticon e caratteri speciali.
- È essenziale per l'internazionalizzazione e lo sviluppo di applicazioni moderne, soprattutto con la diffusione di Internet.
- Esempi di code-point:
 - Carattere “A” → U+0041
 - Carattere “😊” → U+1F60A

“ALGORITMI + STRUTTURE DATI = PROGRAMMI”

Niklaus Wirth

PROGRAMMAZIONE

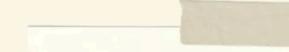
- Il programma è una sequenza di istruzioni che produce un obiettivo desiderato in un tempo finito, implementando un algoritmo.
- La programmazione è il processo che porta alla realizzazione di un programma o software.

RICETTA

- Input:
 - Lista di ingredienti
 - Procedimento da eseguire
- Output:
 - Pancake

Pancake

Ricetta per 2 persone

	<ul style="list-style-type: none">• 200 gr di farina• 250 gr di latte• 2 uova• 30 gr di zucchero• 30 gr di olio di semi• 1 bustina di lievito• 1 pizzico di sale• una noce di burro per la cottura• 3-4 cucchiai di sciroppo d'acero• 3 cucchiai di mirtilli
---	---



Mescolare in una ciotola le uova, l'olio e il latte con una frusta a mano. A parte setacciare farina e lievito e mescolare insieme a zucchero e sale. Unire gli ingredienti secchi a quelli liquidi, girare con la frusta a mano pochi secondi. Infine incorporare gli albumi delicatamente con una spatola.

Servitevi di una padella antiaderente e passate il burro in padella. Aggiungete un mestolo di impasto senza schiacciare. Appena vedete

PROBLEMA

- Per fornire istruzioni a un computer è necessario utilizzare un linguaggio comune.
- Il computer comprende solo sequenze di 0 e 1, ovvero sequenze binarie.
- Il programmatore comprende il linguaggio naturale: “fai, leggi, scrivi.”

PROGRAMMA

- Il computer mette a disposizione delle istruzioni elementari.
- Il programma è una sequenza di istruzioni elementari scritta da un programmatore per risolvere un problema.
- Il programmatore utilizza un linguaggio di programmazione per fornire la sequenza di istruzioni al computer, ovvero per scrivere il programma.
- Il linguaggio di programmazione è un linguaggio compreso sia dal computer che dal programmatore.

TRADUTTORI IN LINGUAGGIO MACCHINA

- Interprete: Le istruzioni vengono tradotte una alla volta ed eseguite immediatamente dal calcolatore.
- Compilatore: Tutte le istruzioni vengono tradotte in linguaggio macchina e memorizzate in un file eseguibile dal calcolatore (programma).

PRO E CONTRO

Velocità di esecuzione:

- L'interprete deve tradurre il programma ogni volta che lo esegue.
- Il programma compilato viene tradotto solo una volta.
- Il compilatore è più efficiente ed ottimizza il codice tradotto.

PRO E CONTRO

Prerequisiti:

- L'interprete deve essere installato su ogni macchina che userà il programma.
- Il compilatore viene acquistato ed usato solo dal programmatore.

PRO E CONTRO

Proprietà intellettuale:

- L'interprete richiede la distribuzione del codice sorgente in chiaro.
- Il programma compilato può essere distribuito in linguaggio macchina, senza il codice sorgente.

LINGUAGGI DI PROGRAMMAZIONE

Calcolatore e programmatore, per comprendersi, devono avere un linguaggio comune:

- Linguaggi a basso livello: (es. Assembly)
- Linguaggi ad alto livello: (es. JavaScript, Python)
- Linguaggi compilati: (es. TypeScript, Java)
- Linguaggi interpretati: (es. JavaScript, Python)

Il documento contenente le istruzioni scritte in un linguaggio di programmazione si chiama **codice sorgente**.

LINGUAGGIO MACCHINA

Sequenza binaria comprensibile solo da uno specifico microprocessore o da una famiglia di microprocessori.

10	00	00	FF	FF	00	00	MZI	YY
10	00	00	00	00	00	00	8	..
10	00	00	00	00	00	00		
10	00	00	80	00	00	00		I
38	01	4C	CD	21	54	68	II	, LI
5D	20	63	61	6E	6E	6F	is program can		
59	6E	20	44	4F	53	20	t be run in DO		
10	00	00	00	00	00	00	mode	S
3A	91	44	00	10	00	00	PE	. L	I'D
31	02	38	00	06	00	00	A	8
L1	00	00	00	10	00	00		5
L0	00	00	00	02	00	00	8	
10	00	00	00	00	00	00		
10	00	00	03	00	00	00	8	
10	10	00	00	10	00	00		
10	00	00	00	00	00	00		
10	00	00	00	00	00	00	0	E
10	00	00	00	00	00	00		
10	00	00	00	00	00	00		

LINGUAGGIO ASSEMBLY

Stampa la scritta “hello world” in linguaggio assembly per microprocessore Intel 8086.

```
; The easiest way to print out "Hello, World!"  
  
name "hi"  
  
org 100h  
  
jmp start ; jump over data declaration  
  
msg: db "Hello, World!", 0Dh, 0Ah, 24h  
  
start: mov dx, msg ; load offset of msg into dx.  
       mov ah, 09h ; print function is 9.  
       int 21h      ; do it!  
  
       mov ah, 0  
       int 16h      ; wait for any key....  
  
ret ; return to operating system.
```

LINGUAGGIO C

Stampa la scritta “hello world”.

```
1  /*
2   Simple C program to display "Hello World"
3   This is a comment
4  */
5
6  // Header file for input output functions like printf
7 #include <stdio.h>
8
9 /**
10  * main function - where the execution of program begins
11  * int - return type of the function
12  * void - indicates that the function takes no arguments
13  * main - name of the function
14  */
15 int main(void) {
16
17     // prints hello world on the console
18     printf("Hello, World!\n"); // \n is used to move to the next line
19     return 0; // ; is used to end the statement
20 }
```

LINGUAGGIO PYTHON

Stampa la scritta “hello world” in Python.

```
1  """
2  Python example that print the "Hello World!"
3  This is a comment
4
5  """
6
7  # next statement will print the string "Hello, World!"
8  print("Hello, World!")
```

2 - INTRODUZIONE AL “C”

Ing. Giancarlo Degani

IL LINGUAGGIO C

- Sviluppato da Dennis Ritchie ai Bell Labs nel 1972 per realizzare il sistema operativo UNIX
- Linguaggio compilato
- Compilatore disponibile per tutte le piattaforme
- Codice molto efficiente



Brian W. Kernighan • Dennis M. Ritchie

IL LINGUAGGIO C, CARATTERISTICHE

- Adatto sia come linguaggio ad alto livello che a basso livello (operazioni sui bit)
- Tantissime librerie disponibili
- Linguaggio procedurale, non ad oggetti (Aggiunti nel C++)
- Gestione della memoria “manuale” (Non c’è garbage collector)

LIBRERIE

- In un linguaggio ad alto livello le funzioni di base sono fornite dal linguaggio (lettura da tastiera, scrittura su schermo, lettura/scrittura da file)
- Queste operazioni elementari sono disponibili sotto forma di funzioni
- Le funzioni sono raccolte e distribuite sotto forma di librerie

COME SI SCRIVE UN PROGRAMMA?

- Bastano un editor di testo per scrivere il file sorgente ed un compilatore o interprete per tradurre il sorgente in linguaggio macchina
- Solitamente si usa uno strumento definito Integrated Development Environment (IDE)
- CLion è un IDE specifico per C/C++

HELLO, WORLD

```
1  /*
2   Simple C program to display "Hello World"
3   This is a comment
4  */
5
6  // Header file for input output functions like printf
7 #include <stdio.h>
8
9 /**
10  * main function - where the execution of program begins
11  * int - return type of the function
12  * void - indicates that the function takes no arguments
13  * main - name of the function
14 */
15 int main(void) {
16
17     // prints hello world on the console
18     printf("Hello, World!\n"); // \n is used to move to the next line
19     return 0; // ; is used to end the statement
20 }
```

HELLO, WORLD

```
1  /*
2   Simple C program to display "Hello World"
3   This is a comment
4  */
5
6  // Header file for input output functions like printf
7 #include <stdio.h>
8
9 /**
10  * main function - where the execution of program begins
11  * int - return type of the function
12  * void - indicates that the function takes no arguments
13  * main - name of the function
14 */
15 int main(void) {
16
17     // prints hello world on the console
18     printf("Hello, World!\n"); // \n is used to move to the next line
19     return 0; // ; is used to end the statement
20 }
```

PROCESSO DI BUILD DI UN PROGRAMMA C

CREAZIONE DI UN ESEGUIBILE

Il processo di creazione di un eseguibile a partire dai sorgenti è composto da 2 fasi:

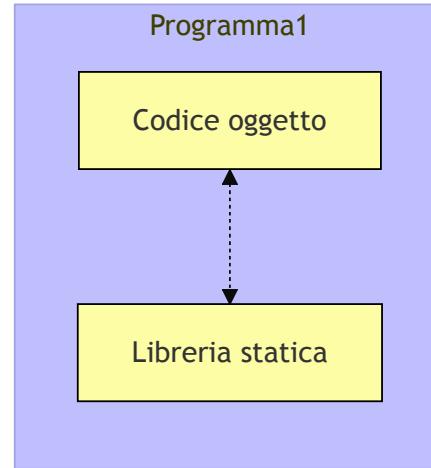
- **compilazione**: il sorgente viene compilato, ma alcune parti (le “funzioni di base”) sono ancora mancanti; viene generato un file intermedio detto file oggetto
- **linking**: il file oggetto e le librerie vengono unite (collegate – “link”) così da aggiungere al file oggetto le parti mancanti e costituire un unico file eseguibile. La fase di link può creare un eseguibile collegando insieme più file oggetto e più librerie.

LIBRERIE

- In un linguaggio ad alto livello le funzioni di base sono fornite dal linguaggio (lettura da tastiera, scrittura su schermo, lettura/scrittura da file)
- Queste operazioni elementari sono disponibili sotto forma di funzioni
- Le funzioni sono raccolte e distribuite sotto forma di librerie

LIBRERIE STATICHE

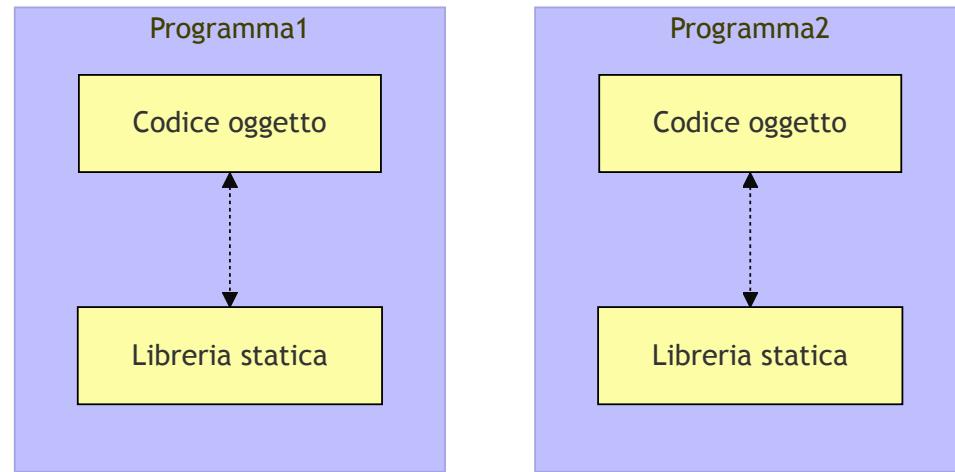
Nella creazione del programma eseguibile, il codice oggetto e le librerie vengono uniti (collegati) a formare un unico file binario.



LIBRERIE STATICHE

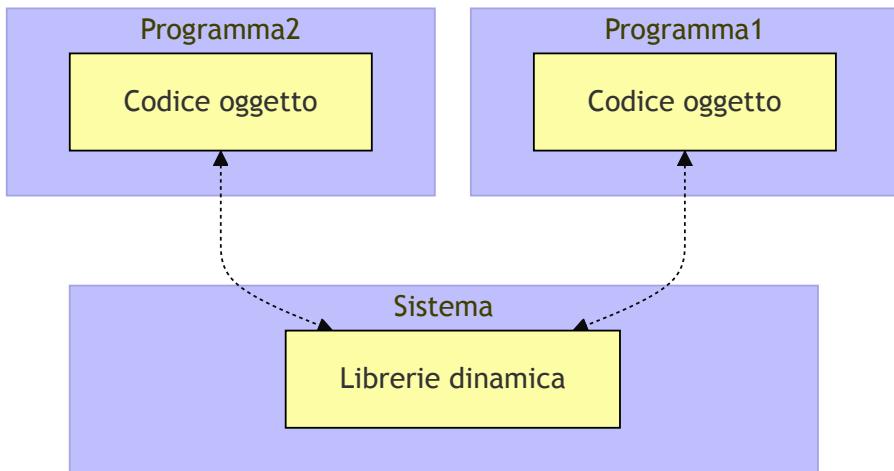
Le librerie statiche vengono incluse in ogni programma che le usa:

- Spreco di memoria
- Manutenzione onerosa



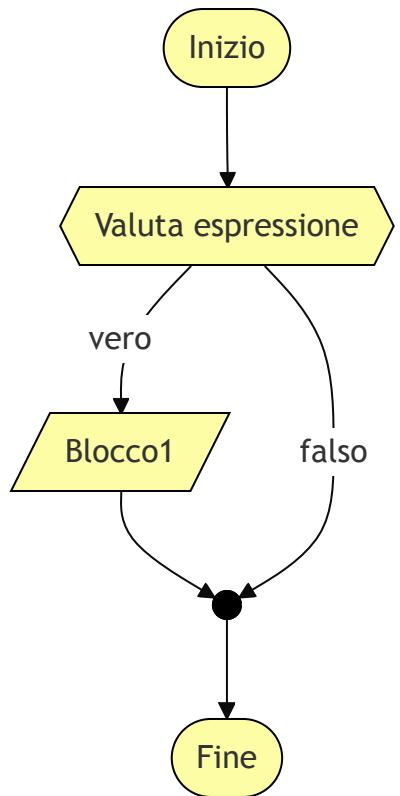
LIBRERIE DINAMICHE

- Nella creazione del programma eseguibile le librerie vengono referenziate, non incluse
- La libreria viene caricata in memoria al momento dell'esecuzione
- La libreria può essere condivisa da più programmi eseguibili
- Manutenzione semplificata



title

char	H	e	I	I	o	!	\0
Dec	72	101	108	108	111	33	0
Hex	48	65	6C	6C	6F	21	0



CODE

left

```
1  /*
2  * Print the "Hello World!" string
3  */
4
5 #include <stdio.h>
6
7 int main(void)
8 {
9     printf("Hello, World!\n");
10    return 0;
11 }
```

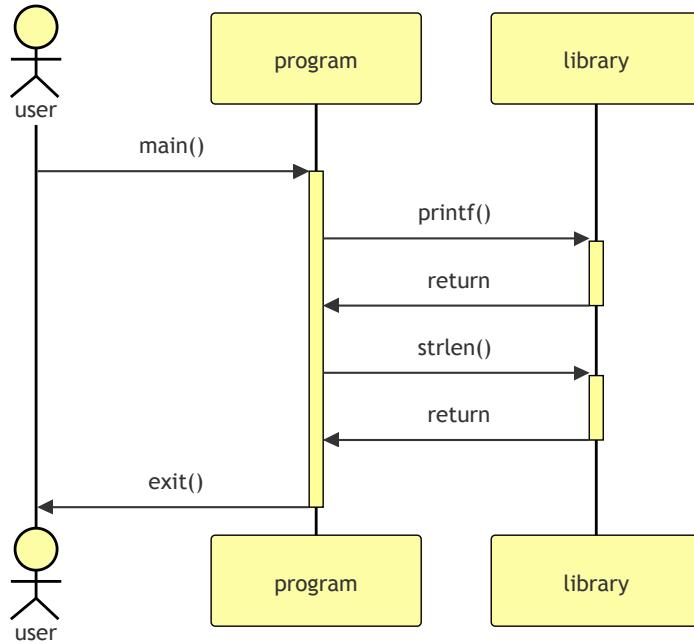
LATEX

asd ad a sadsd

CODE

```
1  /*
2   * Print the "Hello World!" string
3   */
4
5 #include <stdio.h>
6
7 int main(void)
8 {
9     printf("Hello, World!\n");
10    return 0;
11 }
```

ESEMPIO FOR



3 - ISTRUZIONI CONDIZIONALI E CICLI

Ing. Giancarlo Degani

ESEMPIO

- Scope di visibilità
- Stampa su schermo di numeri interi
%d
- Overriding

```
1 #include <stdio.h>
2
3 int main(void) {
4     int base = 2, altezza = 3;
5
6     printf("Hello world!\n");
7     {
8
9         int base = 4;
10        int area = base * altezza;
11        printf("Base = %d \n", base);
12        printf("Altezza = %d \n", altezza);
13        printf("Area = %d \n", area);
14    }
15    printf("Base = %d \n", base);
16
17    getchar();
18    return 0;
19
20 }
```

OPERATORI DI CONFRONTO O RELAZIONALI

Operator name	Syntax
Equal to	$a == b$
Not equal to	$a != b$
Greater than	$a > b$
Less than	$a < b$
Greater than or equal to	$a >= b$
Less than or equal to	$a <= b$

COSTANTI INTERE DECIMALI

- Sono sequenze di cifre decimali eventualmente precedute da ‘+’ o ‘-’
- Sono del tipo int se possibile, altrimenti long se possibile, altrimenti unsigned long
- Se non compatibili con un unsigned long si ha un errore
- Sono di tipo long se seguite dalla lettera L
- Sono unsigned se seguite dalla lettera U

123 -> int

123L -> long int

123U -> unsigned int

123UL -> unsigned long int

COSTANTI INTERE ESADECIMALI

Una costante intera è considerata essere in base 16 se è preceduta da 0x oppure da 0X (zero X) e può contenere le cifre da 0 a 9 e da A a F (maiuscole e minuscole):

$$0xA == 10_{10}$$

$$0x10 == 16_{10}$$

COSTANTI NUMERICHE

- Il modificatore const nella definizione delle variabili informa il compilatore che queste non dovranno essere modificate
- Il valore della variabile const deve essere specificato nell'inizializzazione, può anche essere un'espressione di cui viene calcolato il risultato

```
const double Pi = 3.141592653;  
const double Pi = 4.0*atan(1.0);
```

COSTANTI SIMBOLICHE

- E' possibile dare un nome (un identificatore) ad una quantità costante, questo nome è detto simbolo (per convenzione in maiuscolo)
- Prima della compilazione, il pre-processore cerca i simboli definiti con direttive #define e sostituisce ogni occorrenza del simbolo nome con la corrispondente sequenza_di_caratteri
- Esempi:

```
#define nome sequenza_di_caratteri

#define TRUE 1
#define FALSE 0
#define DIM 80
#define BANNER "#####\n"
```

COSTANTI SIMBOLICHE

- La sostituzione dei simboli inizia a partire dalla riga dove è presente la `#define` e continua fino a fine file (ignorando la struttura a blocchi del programma)
- Non è un'istruzione C di assegnazione, ma una direttiva del pre-processore, quindi:
 - non bisogna mettere il carattere '=' tra il nome del simbolo e la sequenza_di_caratteri
 - non si deve mettere il ';' a fine riga
- La sequenza di caratteri può contenere spazi e termina a fine riga

RIFERIMENTI

- <https://man7.org/linux/man-pages/man3/printf.3.html>
- <https://en.cppreference.com/w/c/language>

OPERATORI LOGICI

Operator name	Syntax
Logical negation (NOT)	$!a$ not a
Logical AND	$a \&& b$ a or b
Logical OR	$a b$ a or b

VALUTAZIONE DELLE ESPRESSIONI LOGICHE

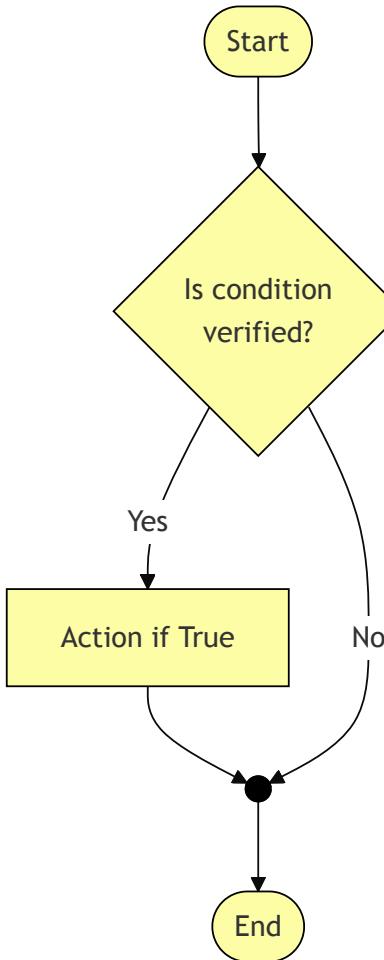
- Vengono valutate da sx a dx
- Si possono usare le parentesi per alterare l'ordine di valutazione
- La valutazione termina appena è possibile stabilire il risultato complessivo

ALTRI OPERATORI

Operator name	Syntax
Function call	<code>f(a, b)</code>
Comma	<code>a, b</code>
Sizeof	<code>sizeof(a)</code>
Conversion (C-style cast)	<code>(a valid C Type)</code>

ESECUZIONE CONDIZIONALE - IF

- Viene valutata una condizione
- Se la condizione è vera,
l'elaborazione prosegue con il ramo
di sinistra
- Se la condizione è falsa,
l'elaborazione prosegue con il ramo
di destra



ESECUZIONE CONDIZIONALE

- Sintassi (minima):

```
if (condizione) <br>
    blocco istruzioni
```

- Il blocco di istruzioni deve essere racchiuso tra se contiene più istruzioni
- Se c'è una sola istruzione, non serve ";" al termine dell'istruzione
- Non serve il ";" dopo la parentesi "}"

ESEMPI

```
if( condizione )  
    istruzione;
```

```
if( condizione ) {  
    istruzione 1;  
    istruzione 2;  
}
```

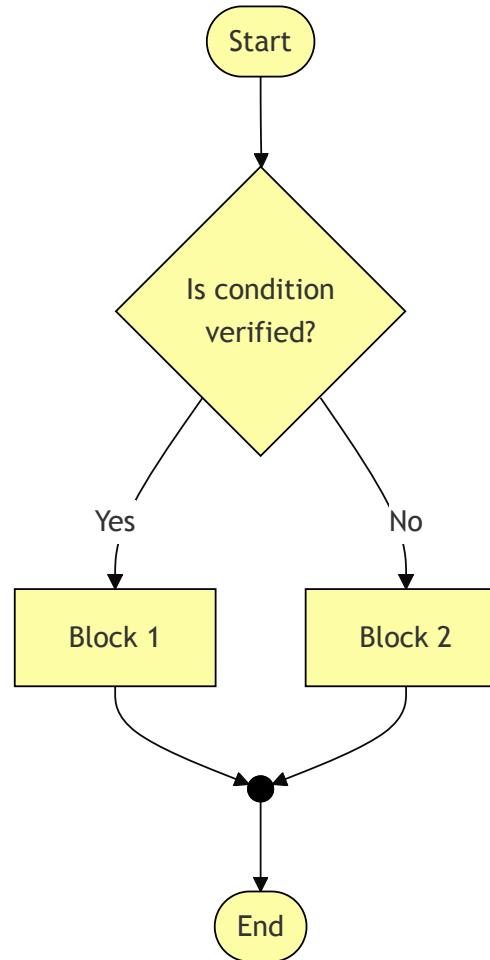
ESEMPIO

- Chiede un numero e, se è positivo, lo stampa
- Le parentesi sono richieste avendo un blocco di più istruzioni

```
1 // Applicazione dello sconto del 10% se il numero di oggetti è maggiore di 10
2 if (numero_oggetti > 10) {
3     sconto = imponibile * 0.10; // Calcolo dello sconto
4     imponibile -= sconto;      // Aggiornamento dell'imponibile
5 }
```

ESECUZIONE CONDIZIONALE - IF- ELSE

- Se la condizione è verificata esegue il blocco 1 altrimenti il blocco 2
- È richiesta quando si hanno più di due casi che necessitano di elaborazioni diverse



ESERCIZIO

Scrivere un programma per la gestione elementare di un carrello della spesa. Il programma deve chiedere input:

- Il numero di oggetti contenuti nel carrello
- Il prezzo unitario

Il programma deve calcolare il costo totale del carrello:

- Se il numero di oggetti è maggiore di 10, applicare lo sconto del 10%
- Calcolare il costo al lordo dell'IVA del 22%
- Stampare a schermo (vedi esempio):

Il dettaglio del carrello, l'imponibile, l'IVA, ed il totale lordo

ESERCIZIO

Output richiesto:

===== Dettaglio del Carrello =====

Numero di oggetti: 12

Prezzo unitario: 200.00

Totale prima dello sconto: 2400.00

Sconto applicato: 240.00

Imponibile (dopo sconto): 2160.00

IVA (22%): 475.20

Totale lordo: 2635.20

=====

SELEZIONE MULTIPLA - SWITCH

```
switch ( espressione ){
    case valore1:
        blocco di istruzioni;

    case valore2:
        blocco di istruzioni;

    default:
        Blocco di 'default';
}
```

SELEZIONE MULTIPLA

- L'espressione deve restituire un valore intero (char, short, int, long)
- I valori sono costanti note al momento della compilazione
- L'elaborazione inizia in corrispondenza del primo 'case' verificato.
- Si possono inserire più statement case con un solo blocco di istruzione;
- Usare **break** per uscire dalla selezione multipla terminato un blocco di istruzioni !

ITERAZIONI

Problema: Visualizzare i numeri interi da 0 a 100

```
printf("0\n");
printf("1\n");
printf("2\n");
...
...
```

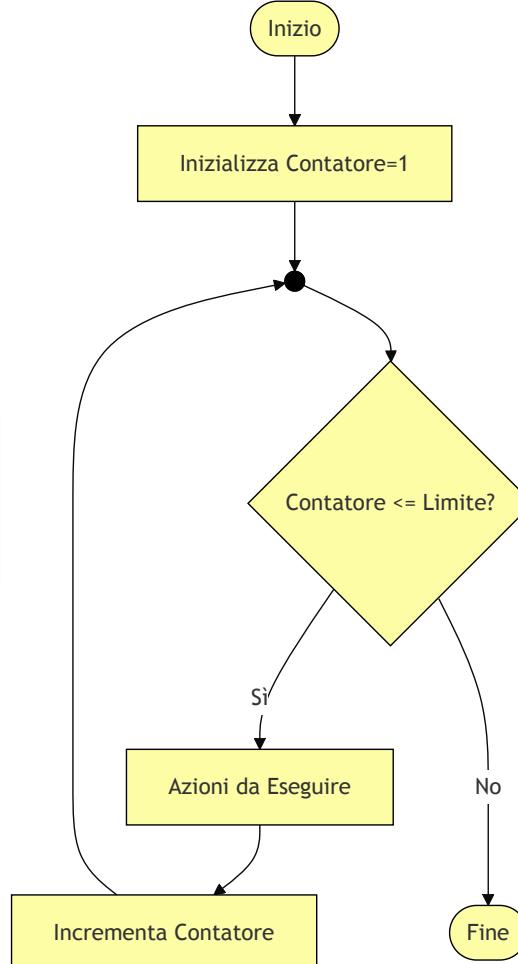
ITERAZIONI

- Si parla di iterazioni quando una istruzione, o un blocco di istruzioni, vengono eseguite più volte
- Le strutture iterative sono comunemente dette cicli o loop
- Sono controllati da una condizione di permanenza nel ciclo

ITERAZIONI: IL CICLO FOR

Il ciclo **for** ripete l'esecuzione del blocco di istruzioni fintanto che la condizione è verificata

```
int j ;  
for ( j = 0; j < 100; j++) {  
    blocco di istruzioni  
}
```



ITERAZIONI: IL CICLO FOR

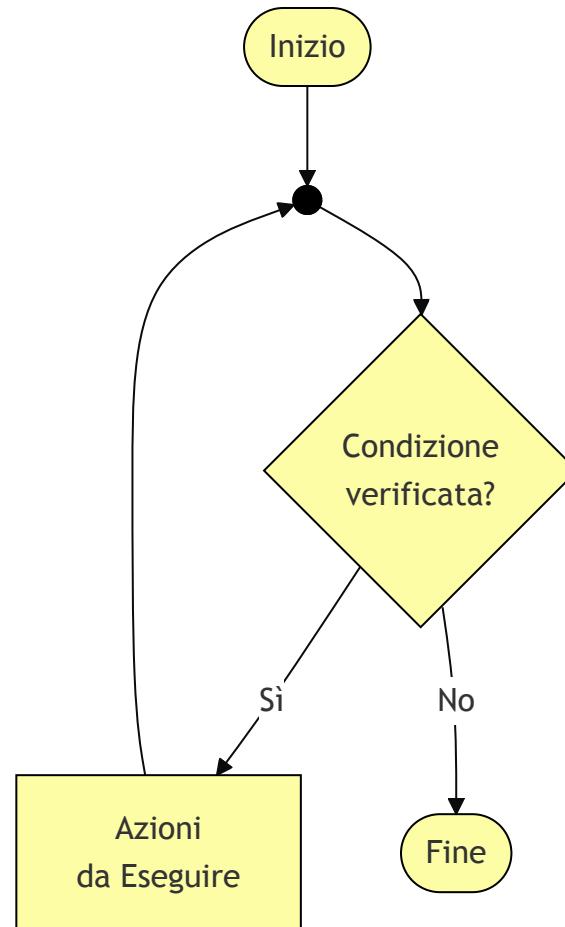
for (Inizializzazione; controllo; incremento)

Blocco di istruzioni

}

ITERAZIONI: IL CICLO WHILE

- Fa eseguire un blocco di codice fino a che una certa condizione è verificata
- Valuta la condizione prima di eseguire il blocco
- Se la condizione è inizialmente falsa, il blocco non viene eseguito neppure una volta
- La condizione deve essere sempre presente



ITERAZIONI: IL CICLO WHILE

```
while ( condizione ) {
```

Blocco di istruzioni

```
}
```

ESEMPIO: IL CICLO WHILE

- Stampa i numeri da 0 a 1000
- Terminato il ciclo, il valore di i è 1001

```
int i = 0;  
while (i <= 1000) {  
    printf("%d ", i);  
    i++;  
}
```

ESEMPIO: IL CICLO WHILE

Somma dei valori introdotti finché non viene immesso il valore 0

```
1 int somma = 0;
2 scanf("%d", &v);
3 while (v != 0) {
4     somma += v;
5     scanf("%d", &v);
6 }
7 printf("Somma: %d", somma);
```

EQUIVALENZA DI FOR E WHILE

For e While consentono di esprimere lo stesso comportamento.

I due esempi a fianco sono equivalenti.

```
1  i = 0;
2  while (i <= 1000) {
3      printf("%d", i);
4      i++;
5  }
6
7
8  for (i = 0; i <= 1000; i++)
9      printf("%d", i);
10 }
```


ITERAZIONI: IL CICLO DO-WHILE

```
do {
```

Blocco di istruzioni

```
} while ( condizione );
```

ITERAZIONI: IL CICLO DO-WHILE

Stampa i numeri da 0 a 1000

```
1 int i = 0;
2 do {
3     printf("%d ", i);
4     i++;
5 } while (i <= 1000);
```

CODIFICA DI NUMERI REALI

Se convertiamo il numero in notazione esponenziale, le informazioni da memorizzare sono:

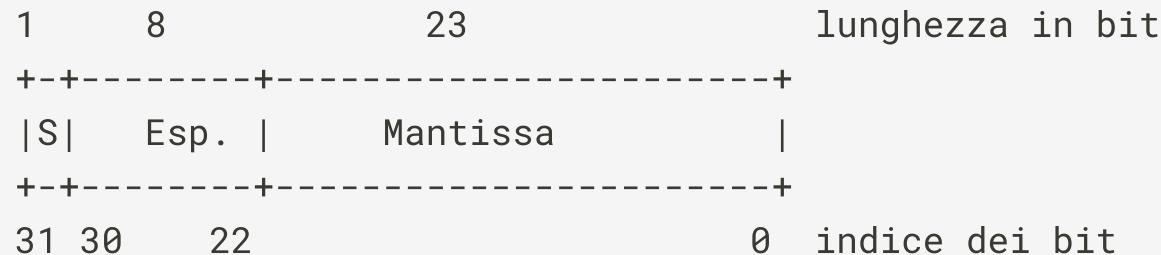
- Segno
- Mantissa
- Esponente

Esempio:

$$-12.34 == -0.1234 * 10^2 == [\text{segno}] 0. [\text{mantissa}] * 10^{[\text{esponente}]}$$

CODIFICA FLOATING-POINT DEI NUMERI REALI

Un numero in virgola mobile, secondo lo standard IEEE è rappresentato su parole di 32, 64 o 128 bit divisi in tre parti: segno, mantissa ed esponente



Il valore del numero rappresentato è calcolabile come:

$$(-1)^s * 2^E * M$$

ESEMPIO

Conversione in precisione semplice (32bit):

$$-5,82812510 = 1 \mid 1000\ 0001 \mid 0111\ 0101\ 0000\ 0000\ 0000\ 000$$

Se la mantissa eccede i 23 bit, viene troncata, per questo si parla di precisione finita.

Per comodità di lettura il numero viene solitamente rappresentato in esadecimale:

$$1100\ 0000\ 1011\ 1010\ 1000\ 0000\ 0000\ 0000 = C0\ BA\ 80\ 00$$

TIPI DI DATI PRIMITIVI PER NUMERI IN VIRGOLA MOBILE

Type	Description	Format
float	Real floating-point type, usually referred to as a single-precision floating-point type	%f %e
double	Real floating-point type, usually referred to as a double-precision floating-point type	%lf %le
long double	Real floating-point type, usually mapped to an extended precision floating-point number format	%Lf %Le

ESERCIZIO

Scrivere un programma che calcoli la radice quadrata x di un numero n :

- Usare l'algoritmo di bisezione per risolvere l'equazione $x^2=n$
 - Non usare la funzione di libreria `sqrt`
 - Calcolare la soluzione con una precisione pari a 0.000 001
- Il numero in input deve essere maggiore di 1
 - In caso contrario stampare un messaggio di errore e terminare il programma
- Prima di scrivere il codice, progettare l'algoritmo con un flowchart

Consegnare: il flowchart, il codice, lo screenshot dell'output

ITERAZIONI: IL CICLO FOR ANNIDATO

```
1 int i, j;
2 for (i = 1; i <= 4; i++)
3 {
4     // Ciclo esterno
5     for (j = 1; j < 4; j++)
6     {
7         // Ciclo interno
8         printf("%d, %d ", i, j);
9     }
10    printf("\n");
11 }
12 printf("%d, %d \n", i, j);
```

FORMULA DI NEWTON

$$\begin{cases} x_{i+1} = \frac{1}{2}(x_i + \frac{A}{x_i}) \\ x_{i=0} = A \end{cases}$$

$$M_n = \frac{(n-1) * M_{n-1} + a_n}{n}$$

4 - VETTORI E MATRICI

Ing. Giancarlo Degani

SOLUZIONE ES. CARRELLO

```
1 // Richiesta dati all'utente
2 printf("Inserisci il numero di oggetti nel carrello: ");
3 scanf("%d", &numero_oggetti);
4
5 printf("Inserisci il prezzo unitario degli oggetti: ");
6 scanf("%f", &prezzo_unitario);
7
8 // Calcolo del costo totale prima dell'applicazione di sconti o IVA
9 imponibile = numero_oggetti * prezzo_unitario;
10
11 // #region blocco_if
12 // Applicazione dello sconto del 10% se il numero di oggetti è maggiore di 10
13 if (numero_oggetti > 10) {
14     sconto = imponibile * 0.10;    // Calcolo dello sconto (10% del totale)
15     imponibile -= sconto;        // Aggiornamento dell'imponibile dopo lo sconto
16 }
```

Inserisci il numero di oggetti nel carrello: 12

Inserisci il prezzo unitario degli oggetti: 1000

===== Dettaglio del Carrello =====

Numero di oggetti: 12

Prezzo unitario: 1000.00

Totale prima dello sconto: 12000.00

Sconto applicato: 1200.00

Imponibile (dopo sconto): 10800.00

IVA (22%): 2376.00

Totale lordo: 13176.00

=====

SOLUZIONE ES. RADICE

```
1 // Input dell'utente
2 printf("Inserisci un numero maggiore di 1: ");
3 scanf("%lf", &n);
4 // Controllo se il numero è valido
5 if (n <= 1) {
6     printf("Errore: il numero deve essere maggiore di 1.\n");
7     return 1; // Termina il programma con errore
8 }
9 // Inizializzazione dei limiti per il metodo di bisezione
10 a = 1.0;
11 b = n;
12 // Iterazione dell'algoritmo di bisezione
13 while ((b - a) > EPSILON) {
14     x = (a + b) / 2.0; // Calcolo del punto medio
15     printf("Punto medio: %.7f\n", x);
16     if (x * x > n)
17         b = x; // Se  $x^2$  è maggiore di n, aggiorna il limite superiore
18     else
19         a = x; // Altrimenti aggiorna il limite inferiore
20 }
21 // Stampa del risultato
22 printf("La radice quadrata approssimata di %.1f è: %.7f\n", n, (a + b) / 2.0);
```

Inserisci un numero maggiore di 1: 5

Punto medio: 3.000000

Punto medio: 2.000000

Punto medio: 2.500000

Punto medio: 2.250000

Punto medio: 2.125000

Punto medio: 2.187500

Punto medio: 2.2187500

Punto medio: 2.2360535

Punto medio: 2.2360687

Punto medio: 2.2360611

Punto medio: 2.2360649

Punto medio: 2.2360668

Punto medio: 2.2360678

Punto medio: 2.2360682

Punto medio: 2.2360680

Punto medio: 2.2360679

Punto medio: 2.2360680

La radice quadrata approssimata di 5.0 è: 2.2360680

VETTORI

- Variabile Scalare: contiene 1 singolo valore:

```
tipo identificatore = valore;  
int numero = 3 ;
```

- Variabili vettoriali: contengono più valori dello stesso tipo:

```
tipo identificatore [ dimensione ] = valore ;  
int numeri[ 3 ] = { 0, 1, 2 };
```

- **dimensione** deve essere una costante intera, positiva, e nota al momento della compilazione
- Contengono elementi dello stesso tipo scalare (int, double, char,...)
- L'indici è di tipo intero e non negativo
- Il primo elemento ha indice 0 (posizione)
- L'ultimo elemento ha indice N-1 (N è la dimensione del vettore)

VETTORI

- Gli elementi del vettore sono allocati in locazioni di memoria contigue e successive
- Si accede ai singoli elementi indicando il nome del vettore seguito dall'indice fra parentesi quadre
- Poiché ciascun elemento del vettore è del tipo indicato nella definizione, può essere utilizzato in tutti i contesti in cui si può usare una variabile di quel tipo

```
int vett[10];
scanf("%d", &vett[4]);
x = vett[4] * 5;
```

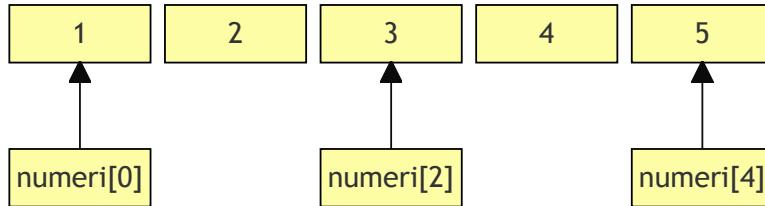
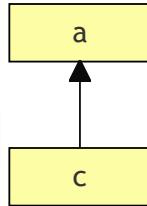
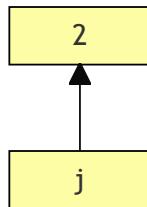
ESEMPI

- Valori scalari

```
int j = 2  
char c = 'a'
```

- vettore di 5 elementi

```
int numeri[5] = {1, 2, 3, 4, 5}
```



VETTORI

- I vettori possono essere attraversati agevolmente mediante un ciclo **for**
- Il nome di un vettore è usato dal compilatore come sinonimo dell'indirizzo di memoria del primo elemento del vettore
- Si “sfora” il vettore quando si accede a elementi oltre i limiti del vettore

```
#define N 10
int vett[N];
for (i=0; i<N; i++){
    scanf("%d", &vett[i]);
}
for (i=N-1; i>=0; i--){
    printf("%d\n", vett[i]);
}
```

ESERCIZI

- Scrivere un programma che chieda quanti valori verranno introdotti dalla tastiera (max 100), li chieda tutti e successivamente visualizzi prima tutti i valori pari nell'ordine in cui sono stati inseriti e poi tutti i valori dispari nell'ordine inverso. (*see example07*)
- Scrivere un programma che, dati in input N numeri reali, con N che al massimo vale 100, stampi quanti di essi sono maggiori della media e successivamente li stampi a video

IL CRIVELLO DI ERATOSTENE

Il crivello di Eratostene è un metodo
che consente di trovare i numeri
primi fino ad un certo n prefissato.

- si scrivono tutti i numeri naturali a partire da 2 fino n
- si cancellano tutti i multipli del primo numero
- si passa al successivo numero non cancellato e si ripete l'operazione con i numeri che seguono

2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

source: wikipedia.org

ESERCIZIO

Scrivere un programma che richieda un numero n positivo e, usando il crivello di Eratostene, trovi e stampi i numeri primi minori o uguali a n (see *example08*).

SOLUZIONE CRIVELLO DI ERATOSTENE

```
1 #include <stdio.h>
2 #define MAX_SIZE 1000
3 int main(void) {
4     int n;
5     printf("Enter the value of n: ");
6     scanf("%d", &n);
7
8     if (n >= MAX_SIZE){
9         return 1;
10    }
11    int prime[MAX_SIZE];
12    for (int i = 0; i <= n; i++) {
13        prime[i] = 1; // true
14    }
15
16    for (int p = 2; p * p <= n; p++) {
17        if (prime[p] == 1) {
18            for (int i = p * p; i <= n; i += p) {
19                prime[i] = 0; // false
20            }
21        }
22    }
}
```

MATRICI

- Sono variabili vettoriali con due dimensioni
- Definizione

```
tipo identificatore [ numero_righe ] [ numero_colonne ] ;
```

- Es: matrice con 10 righe e 20 colonne:

```
int matrice [ 10 ] [ 20 ] ;
```

- Gli indici di riga e colonna vanno da 0 a N-1, dove N è la dimensione
- Matrice con 5 righe (da 0 a 4) e 10 colonne (da 0 a 9)

```
int matrice [ 5 ][ 10 ] ;
```

MATRICI

Come per i vettori, il ciclo **for** si presta per attraversare righe e colonne:

```
int matrice[RIGHE][COLONNE];
for (r=0; r<RIGHE; r++)
{
    for (c=0; c<COLONNE; c++)
        printf("%d ", matrice[r][c]);
    printf("\n");
}
```

MATRICI

- Le matrici sono memorizzate in un'area di memoria contigua per righe
- La matrice m[10][20] è memorizzata come 20 vettori consecutivi di 10 elementi
- Un vettore o una matrice possono essere inizializzati elencando i valori delle singole celle della matrice o del vettore

```
int matrice [2][3] = {1, 2, 3, 4, 5, 6};
```

	0	1	2
0	1	2	3
1	4	5	6

MATRICI

- Non c'è limite al numero delle dimensioni

```
int matrice [DIM_1][DIM_2]...[DIM_N] ;
```

- Solitamente si usano costanti simboliche (#define) per definire le dimensioni dei vettori o delle matrici
- Non è possibile copiare o confrontare due generici vettori (multidimensionali) usando gli operatori = o == sui nomi dei vettori stessi

CARATTERI

- Per memorizzare i simboli grafici corrispondenti ai caratteri bisogna associare un numero intero a ciascuno di essi
- Lo standard ASCII definisce una codifica a 7 o 8 bit
- I caratteri ASCII sono gestiti in C con variabili di tipo **char**, ovvero numeri interi ad 8 bit
- Esempio:

```
char character;
char character = 'A'; // assegnazione con carattere
char character = 65; // assegnazione con codice ASCII decimale
char character = 0x41; // assegnazione con codice ASCII esadecimale
```

CARATTERI

- Caratteri “speciali” sono rappresentati con le sequenze di escape, ovvero premettendo il carattere '\:
 - \'
 - \"
 - \?
 - \\
- Alcuni caratteri di controllo
 - \n - nuova linea
 - \r - ritorno a capo
 - \t - tabulazione

STRINGHE

- Sono vettori di **char** terminate dal carattere **null**
- Null è un carattere speciale rappresentato con \0 (Ottale) o Ox00 (Esadecimale)
- Attenzione:

Simbolo	Decimale	Esadecimale
Null	0	0x00
'0' (zero)	48	0x30

STRINGHE

- Poiché le stringhe sono terminate da null, una stringa di n caratteri, richiede n+1 byte di memoria.

Esempio:

Char	H	e	I	I	o	!	\0
Dec	72	101	108	108	111	33	0
Hex	48	65	6C	6C	6F	21	0

STRINGHE

Costanti

Le stringhe costanti (string literal) sono sequenze di char racchiuse da doppi apici Esempi:

- "Ciao"
- "Hello World!"

Variabili

- Le stringhe variabili sono vettori di char di dimensione nota al momento della compilazione.
- il vettore deve contenere anche il terminatore null.

STRINGHE VARIABILI

- Inizializzazione

```
#define MAX_LENGTH 100
char str[100] = "Hello";
char str[MAX_LENGTH+1] = {'H', 'e', 'l', 'l', 'o'} ;
```

- La dimensione massima viene solitamente gestita con #define
- La lunghezza della stringa è data dal numero di caratteri fino al null escluso
- La stringa è un vettore, posso quindi usare la notazione dei vettori per accedere ai singoli caratteri
 - str0 è il primo carattere
 - str1 è il secondo carattere

STRINGHE VARIABILI

- Attenzione agli apici:
 - 'a' è un carattere che occupa 1 byte
 - "a" è una stringa di 2 char, il caratter 'a' ed il terminatore \0
- Come per i vettori, il nome della stringa rappresenta per il compilatore una variabile contenente l'indirizzo di memoria del primo carattere della stringa.
- Una stringa non può essere copiata con l'operatore '=', devo usare delle funzioni apposite.

I/O DI STRINGHE

`puts(nome_stringa)`

Visualizza *nome_stringa* e aggiunge un '\n' alla fine.

`gets(nome_stringa)`

Legge da tastiera tutta la stringa in input fino al ritorno a capo incluso , la mette in *nome_stringa* senza il '\n' ed aggiunge '\0' alla fine.

`printf("%s", nome_stringa)`

%s visualizza la stringa *nome_stringa* fino al '\0'

STRINGHE - CTYPE.H

Character classification

Defined in header `<ctype.h>`

<code>isalnum</code>	checks if a character is alphanumeric (function)
<code>isalpha</code>	checks if a character is alphabetic (function)
<code>islower</code>	checks if a character is lowercase (function)
<code>isupper</code>	checks if a character is an uppercase character (function)
<code>isdigit</code>	checks if a character is a digit (function)
<code>isxdigit</code>	checks if a character is a hexadecimal character (function)
<code>iscntrl</code>	checks if a character is a control character (function)
<code>isgraph</code>	checks if a character is a graphical character (function)
<code>isspace</code>	checks if a character is a space character (function)
<code>isblank</code> (C99)	checks if a character is a blank character (function)
<code>isprint</code>	checks if a character is a printing character (function)
<code>ispunct</code>	checks if a character is a punctuation character (function)

source: cppreference.com

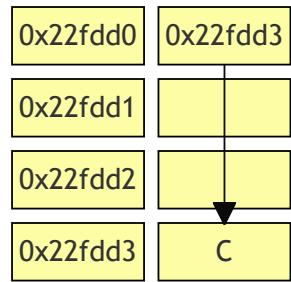
STRINGHE - STRING.H

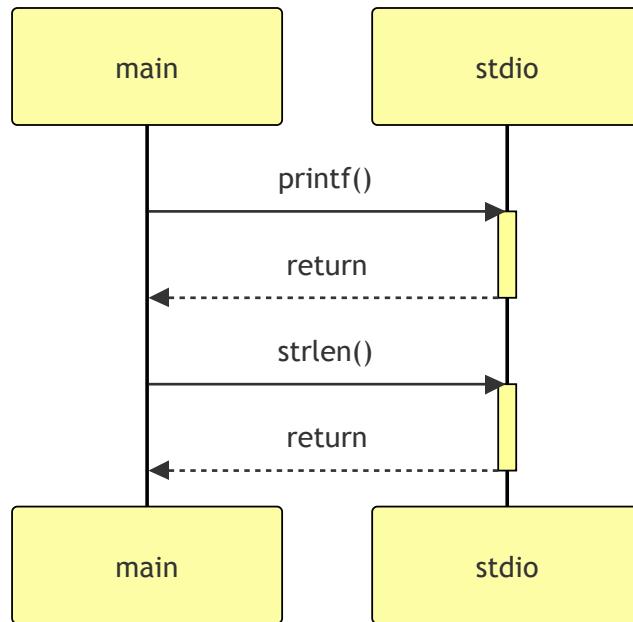
String manipulation

Defined in header `<string.h>`

<code>strcpy</code>	copies one string to another
<code>strcpy_s</code> (C11)	(function)
<code>strncpy</code>	copies a certain amount of characters from one string to another
<code>strncpy_s</code> (C11)	(function)
<code>strcat</code>	concatenates two strings
<code>strcat_s</code> (C11)	(function)
<code>strncat</code>	concatenates a certain amount of characters of two strings
<code>strncat_s</code> (C11)	(function)
<code>strxfrm</code>	transform a string so that <code>strcmp</code> would produce the same result as <code>strcoll</code>
	(function)
<code>strdup</code> (C23)	allocates a copy of a string
	(function)
<code>strndup</code> (C23)	allocates a copy of a string of specified size
	(function)

source: cppreference.com





ESERCIZI STRINGHE

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void) {
5     char str1[100], str2[100];
6
7     printf("Inserisci la prima stringa: ");
8     fgets(str1, sizeof(str1), stdin);
9     printf("Inserisci la seconda stringa: ");
10    fgets(str2, sizeof(str2), stdin);
11
12    // Rimuove il carattere di newline alla fine delle stringhe
13    str1[strcspn(str1, "\n")] = '\0';
14
15    int posizione = strcspn(str2, "\n");
16    str2[posizione] = '\0';
17
18    if (strlen(str1) > strlen(str2)) {
19        printf("La stringa più lunga è: %s\n", str1);
20    } else if (strlen(str1) < strlen(str2)) {
21        printf("La stringa più lunga è: %s\n", str2);
22    } else {
23        printf("Le stringhe hanno la stessa lunghezza. La prima stringa è: %s\n", str1);
24    }
```

ESERCIZI STRINGHE

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_PERSONE 100
6 #define MAX_NOME 50
7
8 int main(void) {
9
10
11     int numero_persone, numero_gruppi;
12
13     // Richiedi il numero di persone
14     do {
15         printf("Inserisci il numero di persone (2 <= numero_persone < 100): ");
16         scanf("%d", &numero_persone);
17     } while (numero_persone < 2 || numero_persone >= 100);
18
19     // Richiedi i nomi delle persone
20     char nomi[MAX_PERSONE][MAX_NOME];
21     for (int i = 0; i < numero_persone; i++) {
22         printf("Inserisci il nome della persona %d: ", i + 1);
23         scanf("%s", nomi[i]);
24     }
```

ESEMPIO 11

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100 // Define a constant for the maximum length of the string
5
6 int main(int argc, char *argv[]) {
7     char vettore[MAX]; // Declare an array to store the input string
8     char *ptr = vettore; // Initialize a pointer to the start of the array
9
10    printf("Dammi una stringa: "); // Prompt the user for a string
11    fgets(vettore, MAX, stdin); // Read the input string from the user
12
13    // Loop through the string until the null terminator is found
14    while(*ptr != '\0') {
15        ptr++;
16    }
17
18    // Print the length of the string (excluding the null terminator)
19    printf("La stringa e' lunga %d caratteri.\n", ptr - vettore - 1);
20    return EXIT_SUCCESS; // Return success status
21 }
```

ESEMPIO 12

```
1 #include <stdio.h>
2
3 #define MAX_SIZE 100
4
5 int main(void) {
6     int n, i, j, temp;
7     int arr[MAX_SIZE];
8
9     // Richiede in input il numero di elementi (N < 100)
10    printf("Inserisci il numero di elementi (N < 100): ");
11    scanf("%d", &n);
12
13    // Controlla se N è maggiore o uguale a 100
14    if (n >= 100) {
15        printf("Errore: N deve essere minore di 100.\n");
16        return 1;
17    }
18
19    // Richiede in input N numeri interi
20    printf("Inserisci %d numeri interi:\n", n);
21    for (i = 0; i < n; i++) {
22        scanf("%d", &arr[i]);
23    }
24
```

FUNZIONE

```
float areaTriangolo( float base, float altezza) {  
    float risultato = base * altezza / 2 ;  
    return risultato ;  
}
```

ESEMPIO 14

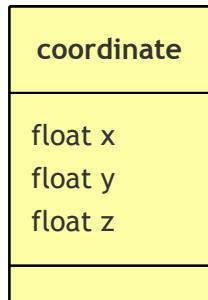
```
1  #ifndef EXAMPLE14_UTILITY_H
2  #define EXAMPLE14_UTILITY_H
3
4  int isPalindrome(char *str);
5
6  /**
7   * Function to print a slice of a vector of integers
8   * @param vettore - Pointer to the vector
9   * @param sx - Pointer to the left character
10  * @param dx - Pointer to the right character
11  */
12 void stampaVettore( int vettore[ ] , int sx, int dx );
13
14
15 /**
16  * Function to remove spaces from a string
17  * @param strin input string
18  * @param strout output string
19  * @param length input string length
20  */
21 void togliSpazi( char *strin, char *strout, int length);
22
23 #endif //EXAMPLE14_UTILITY_H
```

ESEMPIO 14

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "utility.h"
5
6 #define MAX 1000
7
8 int main(int argc, char *argv[]) {
9     char vettore[MAX];
10
11    // Prompt the user to enter a string
12    printf("Introduci la stringa: ");
13
14    // Use fgets instead of gets for safer input handling
15    if (fgets(vettore, MAX, stdin) != NULL) {
16        // Remove the newline character if present
17        size_t len = strlen(vettore);
18        if (len > 0 && vettore[len - 1] == '\n') {
19            vettore[len - 1] = '\0';
20        }
21
22        // Check if the string is a palindrome and print the result
23        if (isPalindrome(vettore)) {
24            printf("La stringa e' palindroma\n");
25        }
26    }
27}
```

STRUCT

```
struct coordinate {  
    float x;  
    float y;  
    float z;  
};  
  
struct coordinate punto1, punto2, *punto3;  
struct coordinate punto4 = {1.0, 2.2, 3.4};  
  
punto2.x = punto4.x;  
punto2.y = punto4.y;  
  
(*punto3).x = punto4.x;  
punto3->x = punto4.x;
```



TYPEDEF

```
typedef struct record {
    char nome[20];
    char cognome[20];
    int eta;
} record_utente;

record_utente utente1, utente2;

utente1.eta = 20;
strcpy(utente1.nome, "Mario");
```

ESEMPIO 15

```
1 #include <stdio.h>
2 #include <limits.h>
3 #include <stdlib.h>
4
5 // Define a type for a pointer to unsigned char
6 typedef unsigned char *byte_pointer;
7
8 // Function to display bytes of a given memory area
9 void show_bytes(byte_pointer start, int len) {
10     for (int i = 0; i < len; i++)
11         printf("%2x ", start[i]);
12     printf("\n");
13 }
14
15 // Function to display bytes of an integer
16 void show_int(int x) {
17     show_bytes((byte_pointer) &x, sizeof(int));
18 }
19
20 // Function to print a byte as bits
21 void print_byte_as_bits(char val) {
22     for (int i = 7; i >= 0; i--) {
23         printf("%c", (val & (1 << i)) ? '1' : '0');
24     }
25 }
```

STRUCT

```
1  typedef struct record {
2      char nome[20];
3      char cognome[20];
4      int eta;
5  } record_utente ;
6
7  record_utente utenti[10] ={ {"bilbo", "baggins", 100 } };
8
9
10
11 struct record {
12     char nome[20];
13     char cognome[20];
14     int eta;
15 } utente1, utente2;
16
17 if ( utente1.eta == utente2.eta
18     && strcmp(utente1.nome, utente2.nome) == 0
19     && strcmp(utente1.cognome, utente2.cognome) == 0 ) {
20     // code block
21
22 }
```

FILE

```
// stdio.h
typedef struct __sFILE {
    ...
} FILE;

FILE *my_file;
```

ESEMPIO FILE

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define FILENAME "input.txt"
5
6 int main(void) {
7     // Declare a file pointer
8     FILE *file;
9
10    // Open the file in read mode
11    file = fopen(FILENAME, "r");
12
13    // Check if the file was opened successfully
14    if (file == NULL) {
15        // Print an error message and exit if the file could not be opened
16        perror("Error opening file");
17        return EXIT_FAILURE;
18    }
19
20    // Declare a character variable to store each character read from the file
21    char ch;
22
23    // Read and print each character until the end of the file is reached
24    while ((ch = fgetc(file)) != EOF) {
```

STRUCT & FUNC

```
1 #include <stdio.h>
2
3 // Definizione della struct
4 struct Studente {
5     char nome[50];
6     int eta;
7     float voto;
8 };
9
10 // Funzione che prende una struct come argomento
11 void stampaStudente(struct Studente s) {
12     printf("Nome: %s\n", s.nome);
13     printf("Età: %d\n", s.eta);
14     printf("Voto: %.2f\n", s.voto);
15 }
16
17 int main() {
18     // Dichiarazione e inizializzazione di una struct
19     struct Studente studente1 = {"Marco Rossi", 20, 28.5};
20
21     // Chiamata alla funzione con la struct come argomento
22     stampaStudente(studente1);
23 }
```

FILES

```
1 1;Bilbo;19;  
2 2;Frodo;7;  
3 3;Sam;10;  
4 4;Merry;30;
```

```
1 Student Records:  
2 ID          Name           Grade  
3 -----  
4 4          Merry          30  
5 1          Bilbo          19
```

FILES

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAXLENGTH 100
6 #define MAXLINES 1000
7 #define DELIMITER ";"
8
9 // Define a structure to hold the record data
10 struct Record {
11     unsigned int id;
12     unsigned int grade;
13     char name[MAXLENGTH];
14 };
15
16 // Function to parse a line of input and populate a Record structure
17 void parseLine(char *line, struct Record *currentRecord) {
18     int counter = 0;
19     char *tok;
20     char *rest = line;
21
22     // Loop through the tokens in the line
23     while ((tok = strsep(&rest, DELIMITER)) != NULL && counter < 3) {
24         switch (counter) {
```

7 - FONDAMENTI DI INFORMATICA

Ing. Giancarlo Degani

ALLOCAZIONE DELLA MEMORIA

- **Allocare** memoria significa chiedere al sistema operativo di assegnare un blocco di memoria RAM ad un programma
- La memoria può essere allocata:
 - Al momento della compilazione (compile-time)
 - Durante l'esecuzione del programma (run-time)
- **Deallocare** memoria significa rilasciare questa memoria al sistema operativo per renderla disponibile ad altri programmi

ALLOCAZIONE STATICÀ

- Usata per variabili static o locali
- Avviene durante la compilazione
- Non può essere rilasciata o deallocata durante l'esecuzione del programma

ALLOCAZIONE AUTOMATICA

- Usata per variabili locali
- Viene allocata durante l'esecuzione
- Viene gestita automaticamente dal compilatore

ALLOCAZIONE DINAMICA

- La memoria viene allocata a run-time
- Può essere deallocata a run-time
- La responsabilità della deallocazione è del programmatore

FUNZIONI DI GESTIONE DELLA MEMORIA

- Allocano un generico blocco contiguo di byte
- Restituiscono l'indirizzo di memoria (di tipo puntatore-a-void) del primo byte del blocco
- Il blocco di byte non ha di per sé alcun tipo, il cast sul puntatore restituito fa sì che il blocco di byte sia considerato dal compilatore come avente il tipo indicato nel cast
- Non si può applicare l'operatore **sizeof** a un blocco di memoria allocato dinamicamente in quanto sizeof viene valutato dal compilatore

<STDLIB.H>

- **void * malloc(size_t size);**
 - The malloc() function allocates size bytes of memory and returns a pointer to the allocated memory.
- **void * calloc(size_t count, size_t size);**
 - The calloc() function contiguously allocates enough space for count objects that are size bytes of memory each and returns a pointer to the allocated memory. The allocated memory is filled with bytes of value zero
- **void free(void *ptr);**
 - The free() function deallocates the memory allocation pointed to by ptr. If ptr is a NULL pointer, no operation is performed.

ESEMPI DI ALLOCAZIONE

Variabile scalare

- Istanziazione di una variabile scalare:

```
double *p;  
p=(double *)malloc(sizeof(double));
```

- Utilizzo:

```
*p = 1.9;
```

- Deallocazione:

```
free(p);
```

ESEMPI DI ALLOCAZIONE

Vettore unidimensionale

- Istanziazione di una variabile scalare:

```
int *p;  
p=(int *)malloc(sizeof(int)*100);
```

- Utilizzo:

```
*(p+12) = 19;  
p[12] = 19;
```

ESEMPI DI ALLOCAZIONE

Vettore di strutture

- Istanziazione di una variabile scalare:

```
int *p;  
p=(int *)malloc(sizeof(int)*100);
```

- Utilizzo:

```
*(p+12) = 19;  
p[12] = 19;
```

ESEMPIO

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(void) {
5     int *a;
6     int n, i;
7
8     printf("Array size? ");
9     scanf("%i", &n);
10
11    // allocate memory for new array a
12    a = (int *) malloc(n * sizeof(int));
13
14    // assign values to array elements
15    for(i=0; i<n; i++) {
16        a[i] = i;
17        printf("%i ", a[i]);
18    }
19    printf("\n");
20
21    // release memory
22    free(a);
23    a = NULL;      // <-- make sure that a is no longer
24
25    // end program
26    return 0;
27 }
```

ESEMPIO

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Define a structure to store course information
5 struct course {
6     int marks;
7     char subject[30];
8 };
9
10 int main() {
11     struct course *ptr;
12     int noOfRecords;
13
14     // Ask the user for the number of records
15     printf("Enter the number of records: ");
16     scanf("%d", &noOfRecords);
```

ESEMPIO

```
17 // Allocate memory for the number of records entered by the user
18 ptr = (struct course *)malloc(noOfRecords * sizeof(struct course));
19 // Loop to get the subject name and marks for each record
20
21 for (int i = 0; i < noOfRecords; i++) {
22     printf("Enter subject %d: ", i + 1);
23     scanf("%s", (ptr + i)->subject);
24     printf("Enter marks for subject %d: ", i + 1);
25     scanf("%d", &(ptr + i)->marks);
26 }
27
28 // Display the entered information
29 printf("Displaying Information:\n");
30 for (int i = 0; i < noOfRecords; ++i) {
31     printf("%s\t%d\n", (ptr + i)->subject, (ptr + i)->marks);
32 }
33
34 // Free the allocated memory
35 free(ptr);
36 return 0;
```

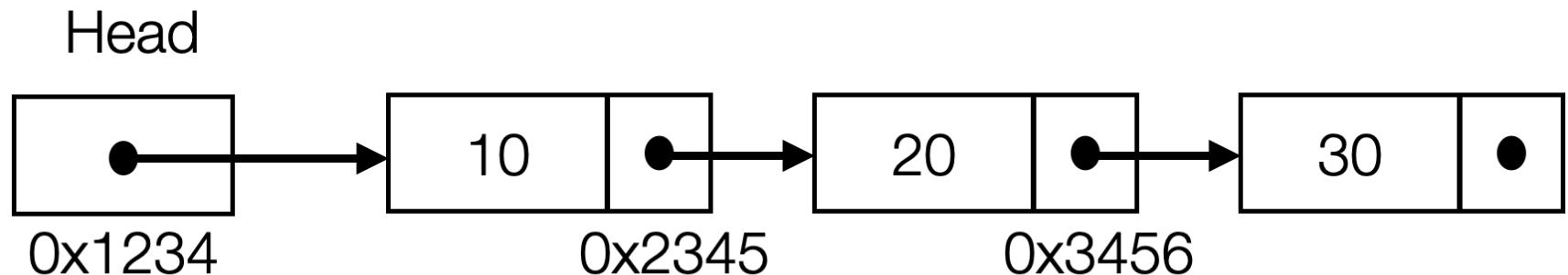
ESERCIZIO

Si scriva un programma che ordini in senso crescente i valori contenuti in un file di testo e li scriva in un'altro. Non è noto a priori quanti siano i valori contenuti nel file. Si utilizzi una funzione per l'ordinamento. Il programma, per allocare un vettore dinamico di dimensione appropriata, nel main:

- conta quanti sono i valori leggendoli dal file e scartandoli
- crea il vettore dinamico di dimensione adeguata
- lo riempie rileggendo il file
- lo passa alla funzione di ordinamento
- scrive il file di output con il contenuto del vettore riordinato

LISTE

- Si definisce lista una struttura dati in cui ogni oggetto ha un collegamento ad un altro oggetto (Linked list)
- Il collegamento è costituito da un puntatore all'area di memoria contenente l'elemento successivo della lista



VANTAGGI E SVANTAGGI RISPETTO AI VETTORI

- La dimensione della lista può essere modificata a runtime
- Lo spostamento di elementi è molto veloce
- L'accesso all'i-esimo elemento è più lento perchè devo scorrere tutta o parte della lista

IMPLEMENTAZIONE CON STRUCT

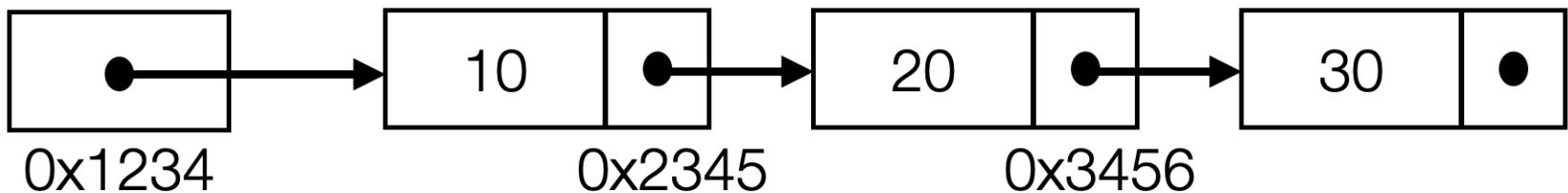
- **valore** è l'informazione contenuta nel singolo elemento della lista
- **head** è il puntatore al primo elemento della lista (testa della lista)
 - Inizialmente la lista è vuota, quindi head è NULL
- **next** è il puntatore al prossimo elemento della lista
 - next è NULL nell'ultimo elemento della lista (coda della lista)

```
1 static struct nodo {  
2     int valore;  
3     struct nodo *next; // pointer to the next instance  
4 } *head = NULL, *p, *q; // 3 pointers to a nodo
```

INSERIMENTO IN TESTA ALLA LISTA 1

Situazione iniziale con una lista non vuota

Head



puntatoreNodo



nuovoValore



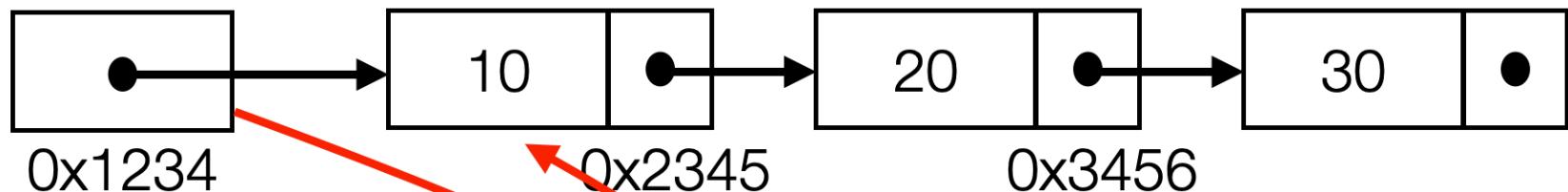
INSERIMENTO IN TESTA ALLA LISTA 2

Creo una nuova variabile per contenere il nuovo elemento della lista

INSERIMENTO IN TESTA ALLA LISTA 3

Copro il valore ed il puntatore alla testa della lista nel nuovo elemento

Head



puntatoreNodo

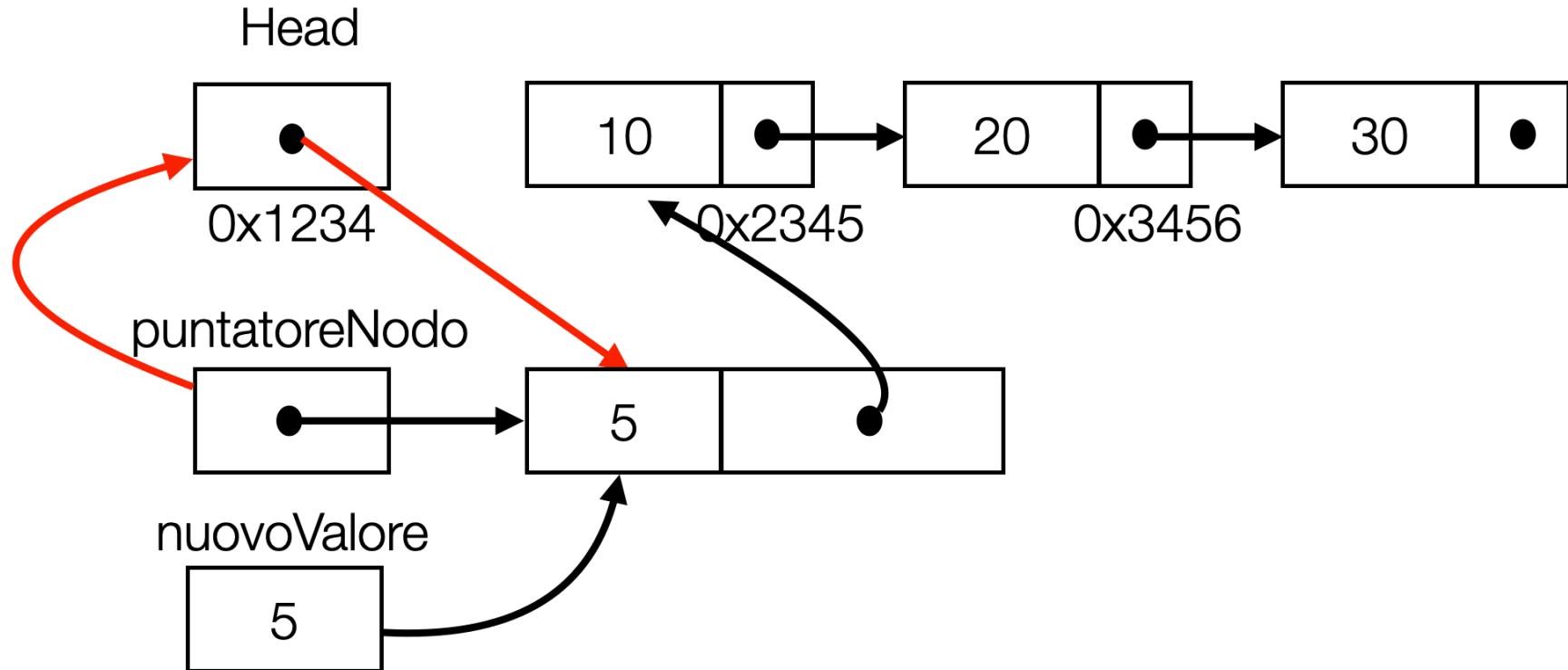


nuovoValore



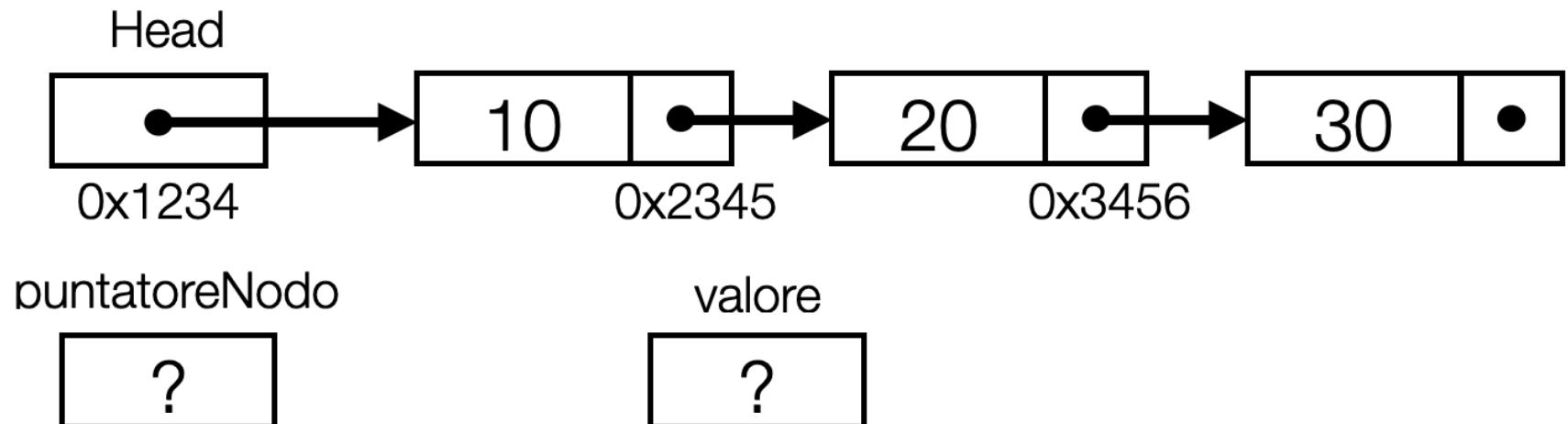
INSERIMENTO IN TESTA ALLA LISTA 4

Cambio il valore di Head



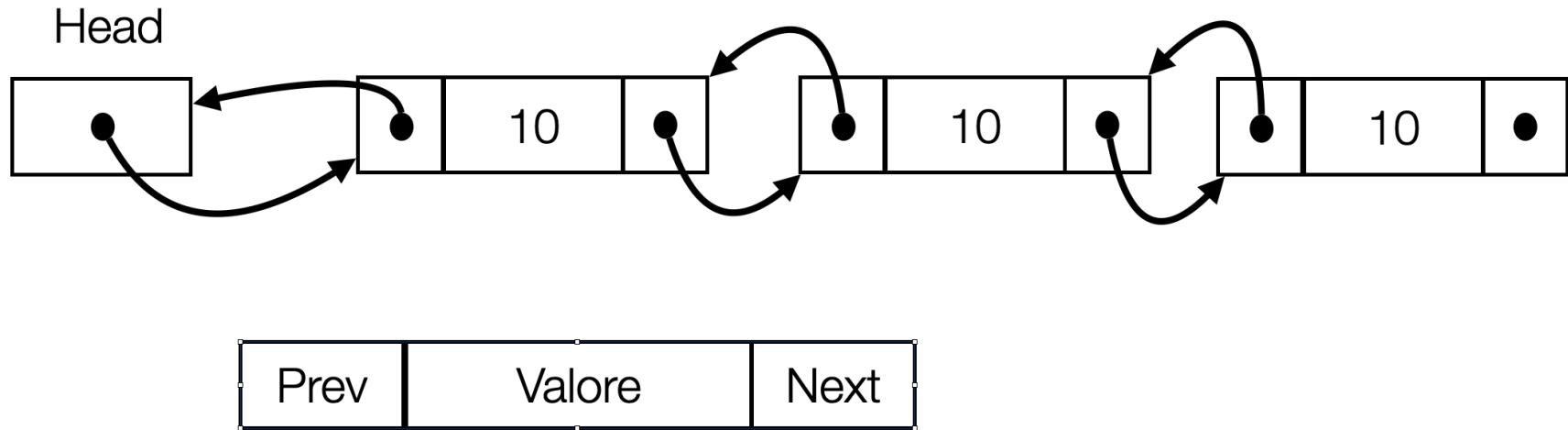
RIMOZIONE DALLA TESTA

- Copio il valore dell'elemento in testa in una variabile temporanea
- Copio il puntatore all'elemento in testa (head) in una variabile temporanea
- Copio il puntatore al secondo elemento in head
- Rilascio la memoria allocata per l'elemento rimosso



LISTE BIDIREZIONALI

Situazione iniziale con una lista non vuota



LISTE DI LISTE

- Posso suddividere la lista in parti per rendere più veloce la navigazione all'interno della lista
- Uso un algoritmo di hashing per definire il partizionamento e quindi la ricerca

ESERCIZIO

- Implementare delle funzioni per la gestione di una lista dinamica di numeri interi
 - lista.h dichiarazione delle funzioni
 - lista.c implementazione delle funzioni
- Scrivere un main che proponga un menu per testarle.
- Ogni funzione, quando richiamata, deve stampare a video l'intera lista

```
1 int AddToHead(int);  
2 int AddToTail(int);  
3 int RemoveFromHead(int *);  
4 int RemoveFromTail(int *);  
5 void ClearAll(void);  
6 void printAll();
```

ESEMPIO DI MENÙ

Operazioni possibili:

- a. AddToHead
- b. AddToTail
- c. RemoveFromHead
- d. RemoveFromTail
- e. ClearAll
- p. PrintAll
- x. Exit

Scelta:

ESEMPIO DI OUTPUT

```
0) address: [0x6000039dc000], value: [22222], next: [0x6000039c4000]
1) address: [0x6000039c4000], value: [11111], next: [0x0]
```

Operazioni possibili:

- a. AddToHead
- b. AddToTail
- c. RemoveFromHead
- d. RemoveFromTail
- e. ClearAll
- p. PrintAll
- x. Exit

Scelta: