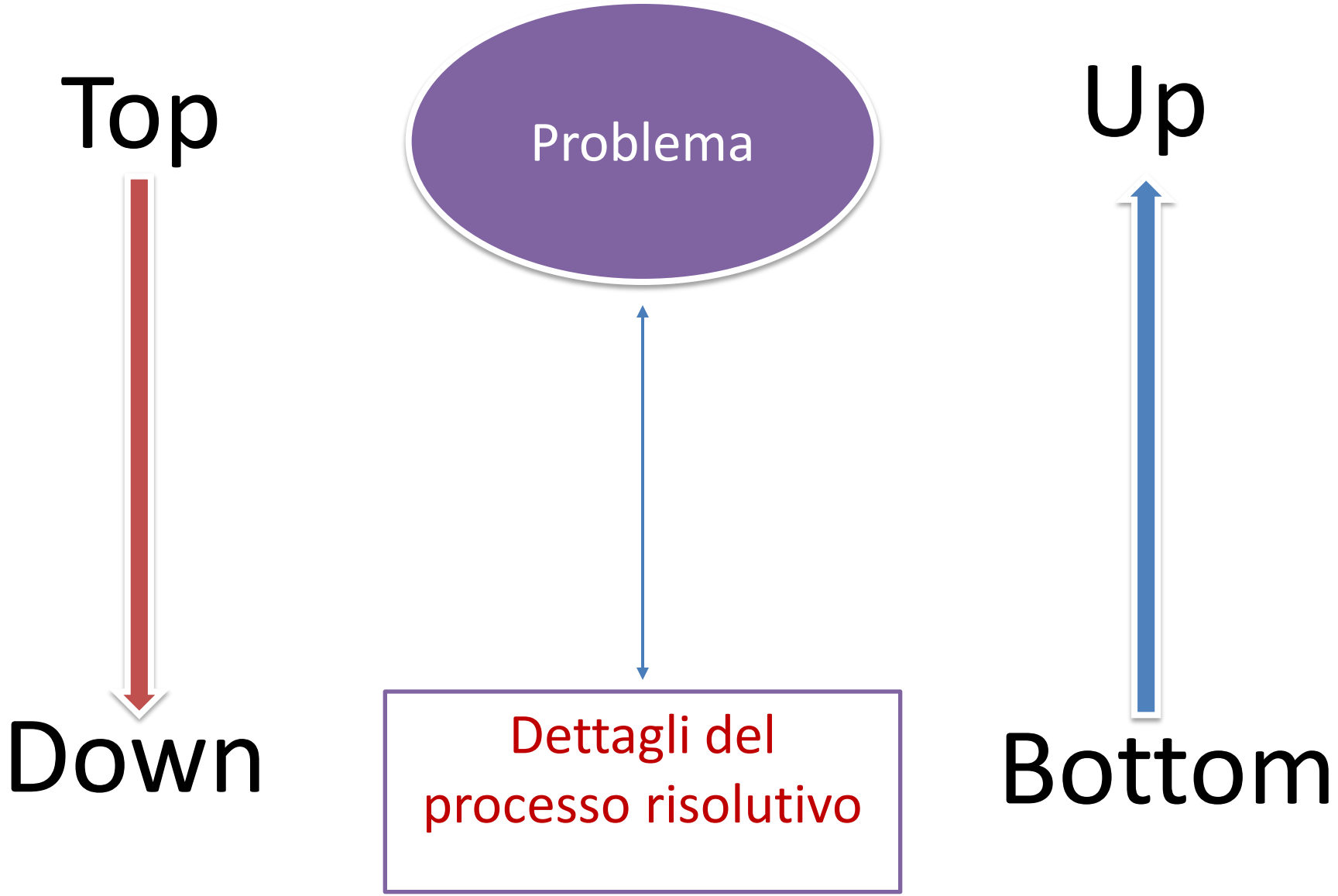
Metodologie di progettazione e sottoprogrammi

Autore: Prof. Rio Chierego

Ultimo aggiornamento: 22 settembre 2024

Metodologie di progettazione



Metodologie di progettazione

- per affrontare problemi complessi si ricorre alla tecnica dei ***raffinamenti successivi*** che suggerisce di ***scomporre*** il problema in problemi più semplici (***sottoproblemi***)
- … e di applicare anche a questi sottoproblemi la stessa tecnica fino ad ottenere problemi facilmente risolvibili
- questa tecnica è definita ***top-down***:
  - si parte da una ***visione globale*** del problema (alto livello di astrazione) [***top***]
  - poi si scende nel ***dettaglio*** dei sottoproblemi diminuendo il livello di astrazione [***down***]
- viene fornita inizialmente una soluzione del problema che non si basa però su operazioni elementari, ma sulla soluzione di sottoproblemi

Metodologie di progettazione

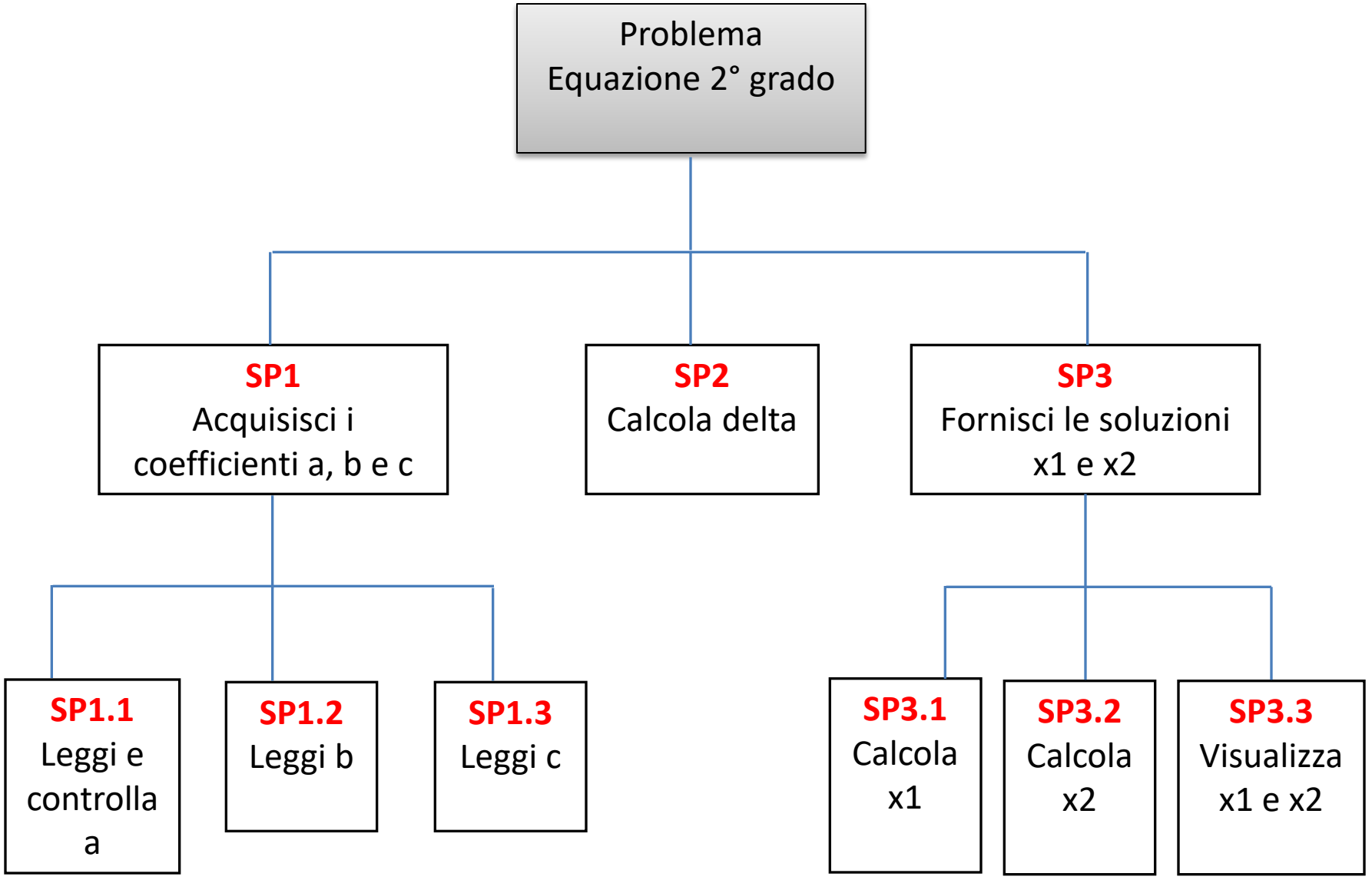
- i modelli top-down e bottom-up (dall'alto verso il basso e dal basso verso l'alto) sono strategie di elaborazione dell'informazione e di gestione delle conoscenze, riguardanti principalmente il software …
- nel modello ***top-down*** è formulata una visione generale del sistema senza scendere nel dettaglio di alcuna delle sue parti
  - ogni parte del sistema è successivamente rifinita aggiungendo maggiori dettagli dalla progettazione.
- nella progettazione ***bottom-up*** parti individuali del sistema sono specificate in dettaglio
  - queste parti vengono poi connesse tra loro in modo da formare componenti più grandi, che vengono a loro volta interconnessi fino a realizzare un sistema completo

Fonte: Wikipedia

Metodologia di progettazione: TOP-DOWN

- se il sottoproblema è ***semplice*** allora viene ***risolto***, viene cioè scritto l'algoritmo di risoluzione
- se il sottoproblema è ***complesso*** viene riapplicato lo stesso procedimento ***scomponendolo*** in sottoproblemi più semplici
- ***diminuisce*** il livello di ***astrazione*** (si affrontano problemi sempre più concreti)
- ***diminuisce*** il livello di ***complessità*** *(i sottoproblemi devono essere più semplici del problema che li ha originati)*
- fino ad arrivare alla stesura di tutti gli algoritmi necessari

Tale metodologia di progettazione utilizza una strategia di tipo DEDUTTIVO

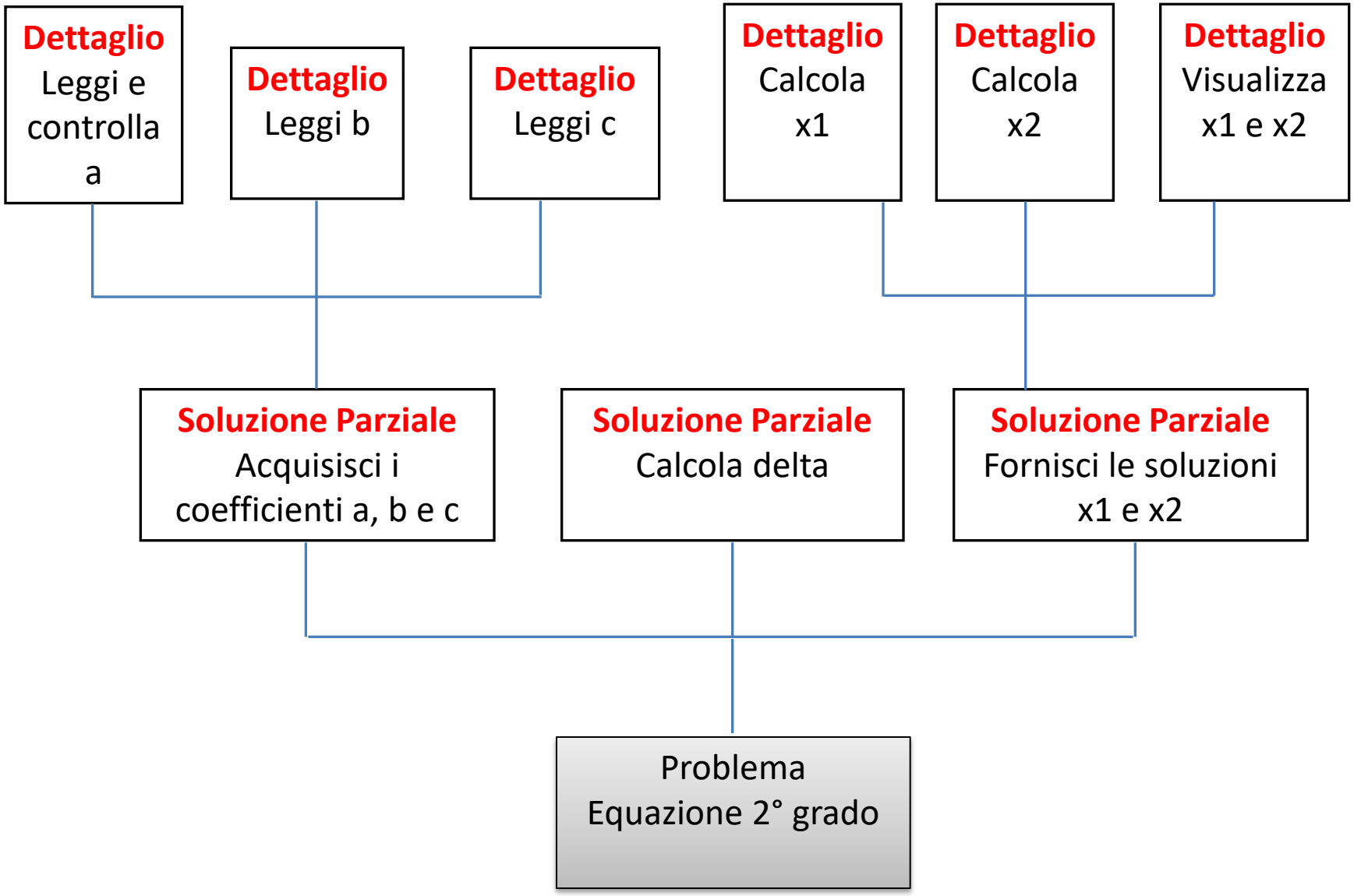


Esempio di metodologia di progettazione TOP-DOWN

Metodologia di progettazione: BOTTOM-UP

- la metodologia bottom-up concentra l'attenzione sui dettagli più caratteristici del problema e ne propone una soluzione
- in un secondo momento le soluzioni parziali verranno integrati con le altre per ottenere la soluzione del problema iniziale
- spesso la soluzione ad alcuni sottoproblemi è già disponibile (librerie, classi, funzioni)
- parola chiave: riutilizzo o riuso

Tale metodologia di progettazione utilizza una strategia di tipo INDUTTIVO



Esempio di metodologia di progettazione BOTTOM-UP

Implementare un SOTTOPROGRAMMA
Quando conviene e quando non conviene utilizzarlo

Non esiste una formula in grado di stabilire quanti programmi occorrono per risolvere un problema e quando essi vanno utilizzati. E' possibile fornire soltanto delle linee guida basate sull'esperienza che vanno considerate come indicazioni operative:

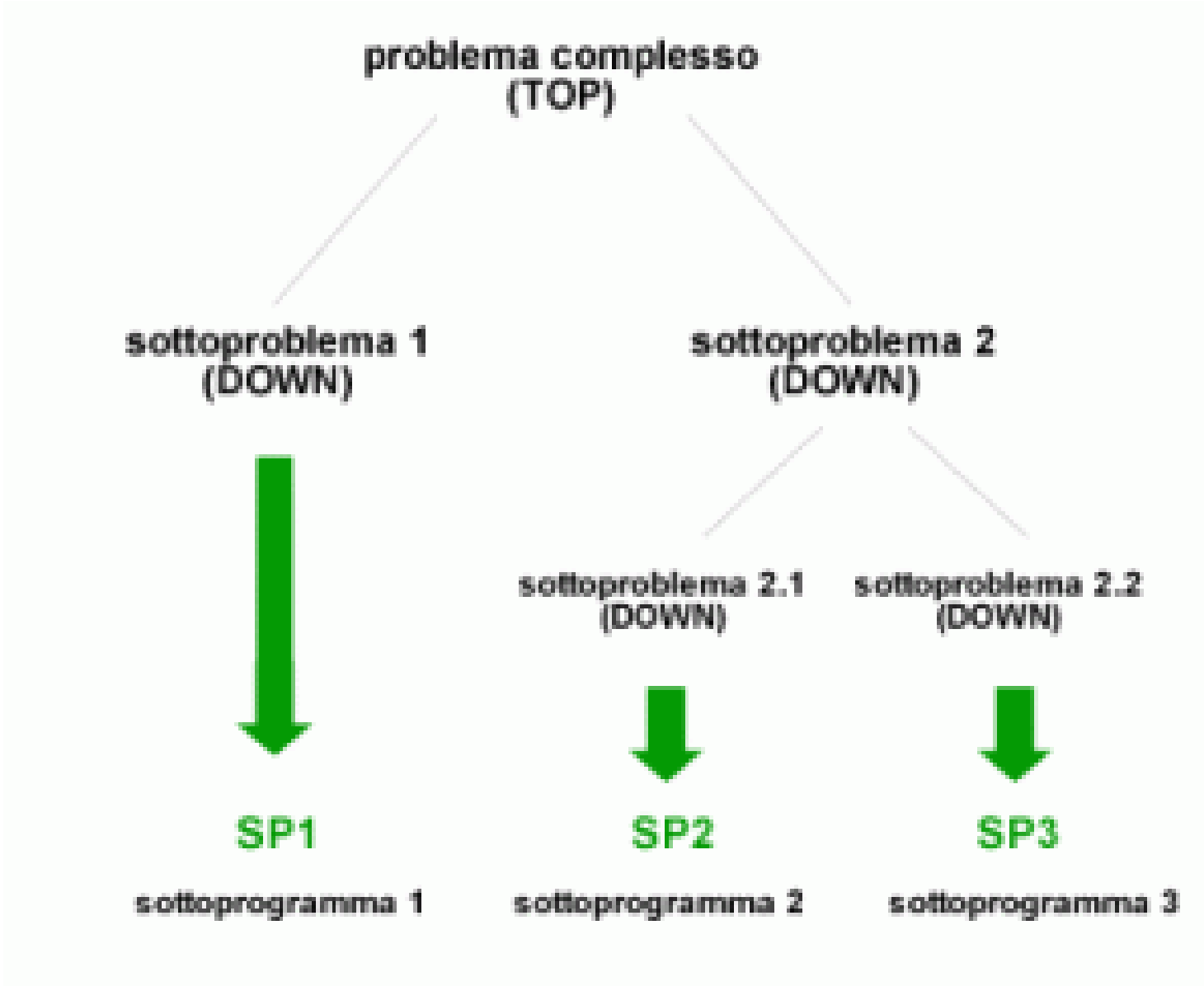
a) CONVIENE descrivere un sottoproblema per mezzo di un sottoprogramma se
- è di interesse generale;
- pur non essendo di interesse generale si presenta più volte all'interno del programma;
- pur essendo di scarso interesse generale, permette una maggiore leggibilità del programma.

b) NON CONVIENE descrivere un sottoproblema per mezzo di un sottoprogramma se
- è di scarso interesse generale;
- non migliora la leggibilità del programma, anzi la complica;
- non garantisce un risparmio di tempo, soprattutto se si tratta di programma breve.

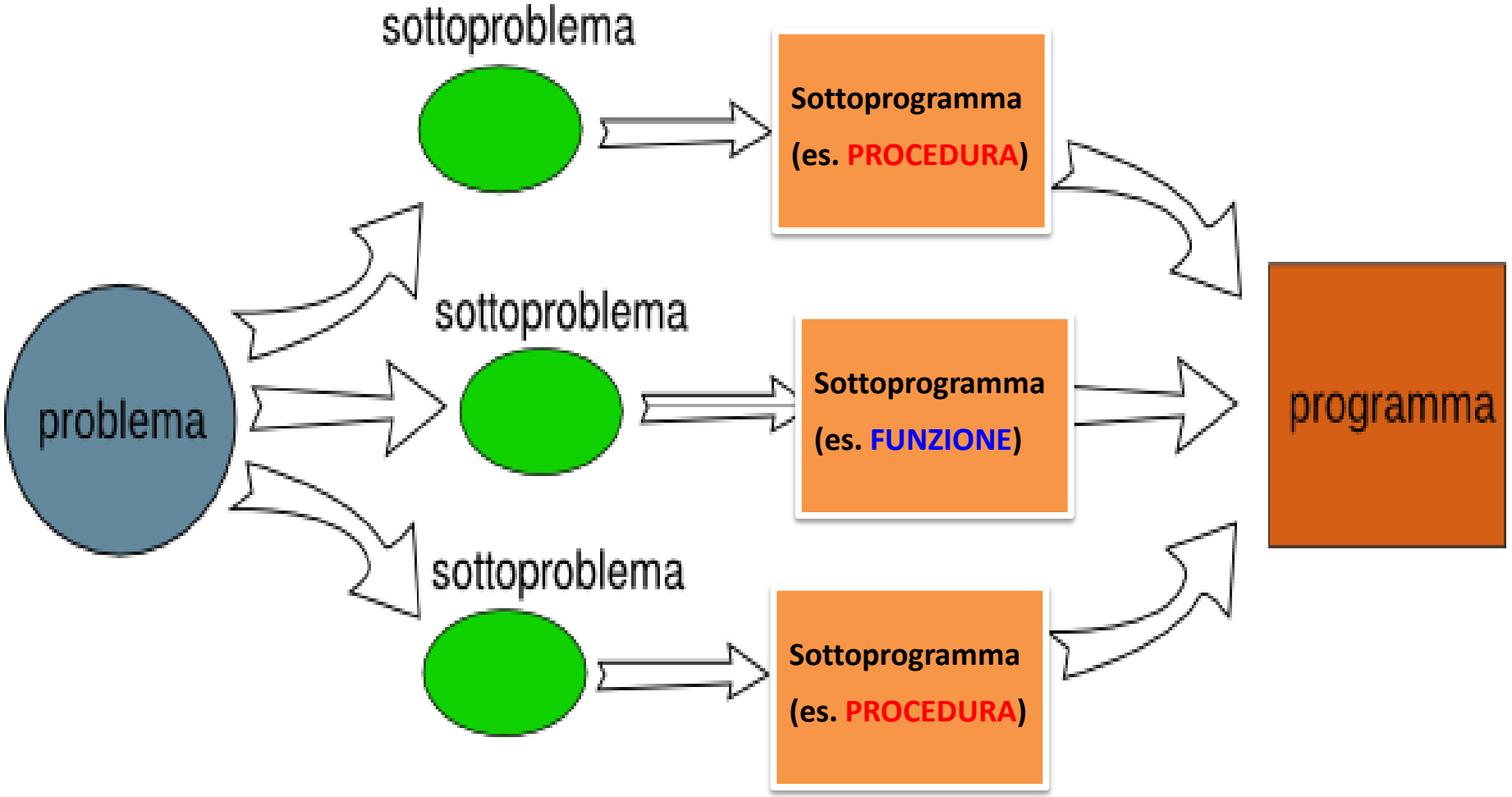
Implementare un SOTTOPROGRAMMA: vantaggi

Riassumendo i **vantaggi** derivanti dall'uso dei sottoprogrammi sono:

- migliorano la leggibilità del programma;
- permettono l'astrazione (ossia il sottoprogramma permette al programmatore di interessarsi di "cosa" fare e non di "come" farlo);
- consentono di scrivere meno codice (e quindi al programma eseguibile di occupare meno memoria);
- sono riutilizzabili anche in altri contesti.


problema complesso
(TOP)
sottoproblema 1
(DOWN)
sottoproblema 2
(DOWN)
sottoproblema 2.1
(DOWN)
sottoproblema 2.2
(DOWN)
SP1
SP2
SP3
sottoprogramma 1
sottoprogramma 2
sottoprogramma 3

Schematizzazione della modularità di un programma



Esecuzione di un SOTTOPROGRAMMA: Funzionamento

Quando un programma non è in esecuzione risiede su una **MEMORIA DI MASSA** e subito dopo la compilazione ed il linkaggio, sarà costituito esclusivamente dal codice o istruzioni e dai dati ed occuperà un'area di memoria (la cui dimensione in BYTE dipende esclusivamente dalle istruzioni e dai dati utilizzati) che è possibile essere pensata come suddivisa in **due segmenti**:

- **il Segmento "CODICE"** o "ISTRUZIONI"**:** area contenente le istruzioni del programma (codice) scritte in linguaggio macchina

- **il Segmento "DATI":** area contenente variabili e costanti allocate staticamente

Quando un programma è in esecuzione (chiamato anche **TASK** o **processo)** viene allocato in memoria centrale (**RAM**) e gli viene assegnato, dal sistema operativo (un particolate programma chiamato **loader**), nella memoria di lavoro (**RAM**) anche una zona di memoria aggiuntiva rispetto a quella posseduta quando è "in quiete".

Un programma in esecuzione quindi occuperà una zona della memoria di lavoro che è possibile pensare ora suddivisa in **quattro segmenti**:

- **il Segmento "CODICE"** o "ISTRUZIONI"**:** *vedi descrizione già data in precedenza*

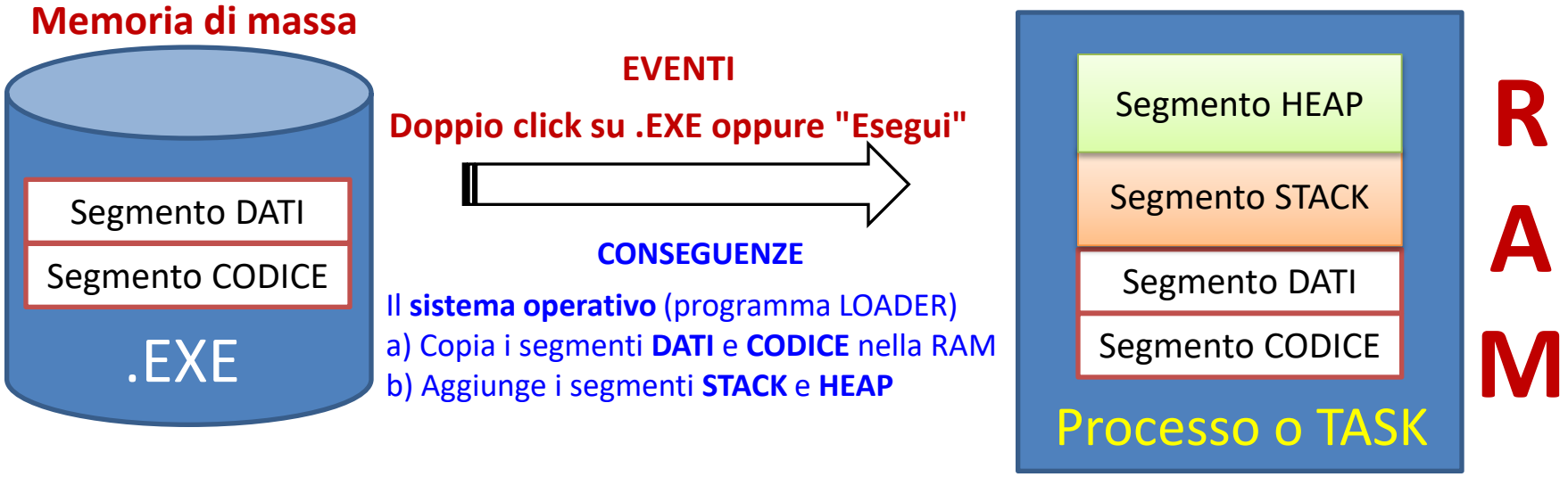
- **il Segmento "DATI":** *vedi descrizione già data in precedenza*

- **il Segmento "STACK":** area destinata a gestire la "**PILA DELLE ATTIVAZIONI**"

- **il Segmento HEAP** (lett. mucchio) **di sistema:** area destinata a raccogliere i dati gestiti dinamicamente che come tali verranno allocati e deallocati dinamicamente (che verranno illustrati IN SEGUITO)

Esecuzione di un SOTTOPROGRAMMA: Funzionamento

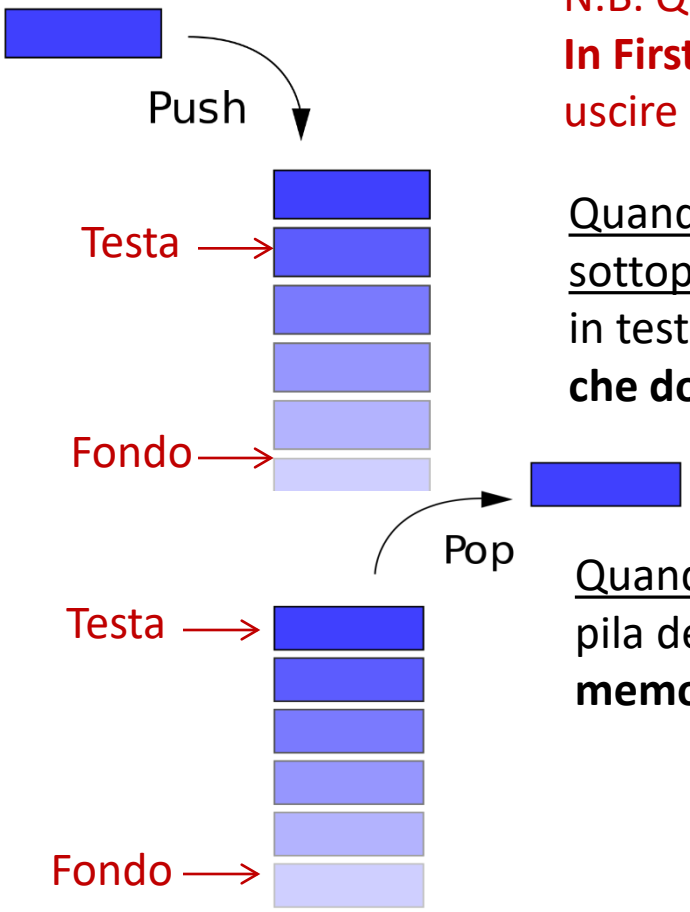
Quando un programma è in esecuzione, tutte le sue parti (sottoprogrammi) vengono caricate in memoria centrale in un'apposita zona di memoria nella RAM per poi essere attivate al momento della chiamata. Una volta terminata l'esecuzione del programma questa zona di memoria verrà rilasciata liberando la memoria precedentemente occupata **(allocazione dinamica del codice da parte del sistema operativo).**



Quando la CPU incontra **una istruzione di chiamata** a sottoprogramma (nel nostro esempio successivo indicata con **SP1, SP2, SP3 ossia SP<n>) sospende l'esecuzione del programma corrente** e passa ad eseguire le istruzioni contenute nel sottoprogramma chiamato. Terminata l'esecuzione, la CPU quando arriva all'istruzione di FINE riprende l'esecuzione del programma ripartendo dall'istruzione successiva a quella di chiamata.

Esecuzione di un SOTTOPROGRAMMA: Funzionamento

Per ricordare da quale istruzione va ripresa l'esecuzione dopo un sottoprogramma, la CPU si serve di una apposita STRUTTURA DATI detta **PILA delle attivazioni o STACK** dalla quale i dati possono essere inseriti o estratti solo da una estremità che viene detta **testa della pila**.

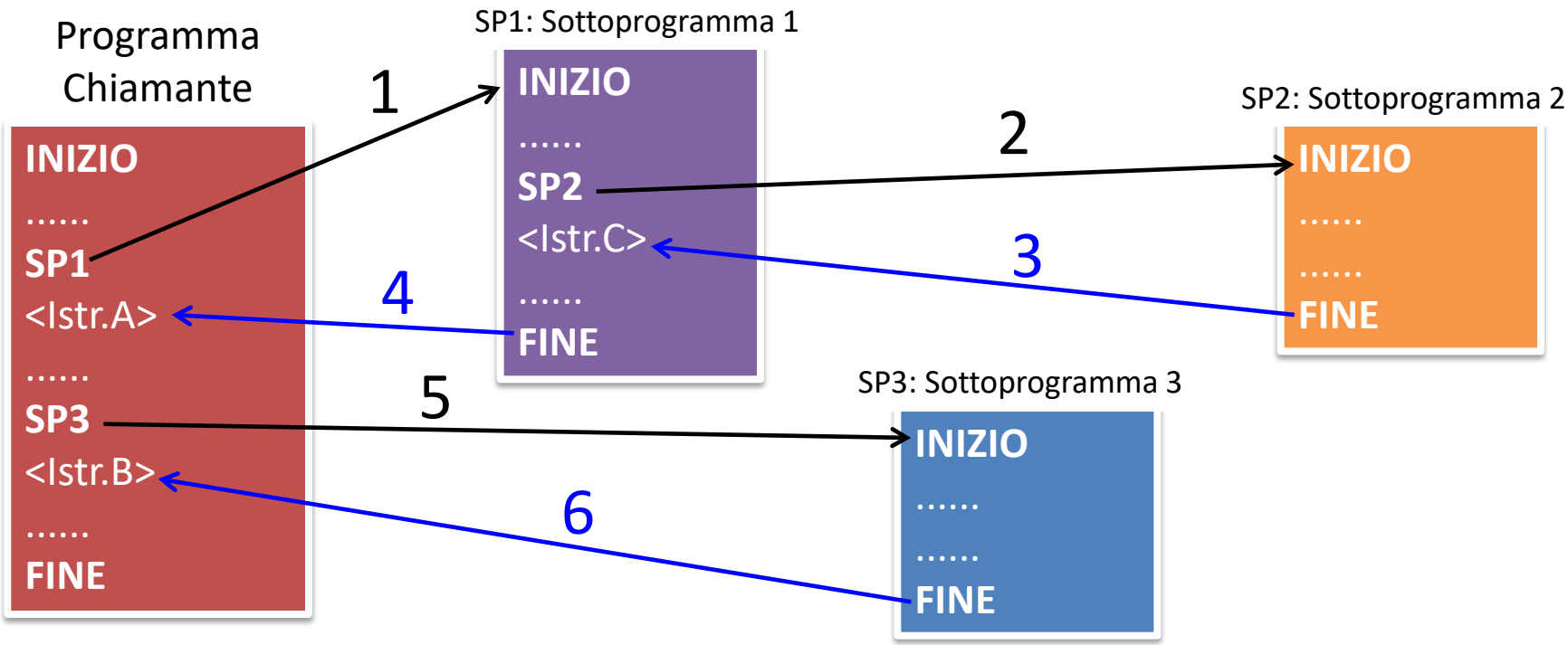


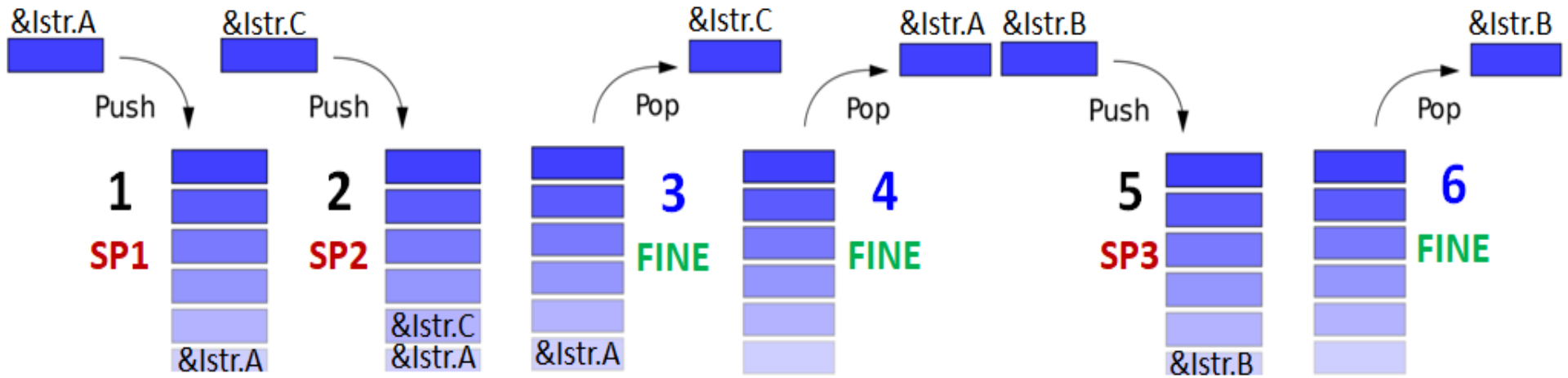
N.B. Questa struttura dati ha una struttura di tipo **LIFO o Last In First Out** nel senso che l'ultimo ad entrare è il primo ad uscire (esempio pila di piatti o di cd).

Quando la CPU esegue una istruzione di chiamata a sottoprogramma allora inserisce (**Push**) nella pila delle attivazioni in testa **l'indirizzo della cella di memoria contenente l'istruzione che dovrà essere eseguita al rientro dal sottoprogramma.**

Quando la CPU esegue una istruzione di FINE allora utilizza la pila delle attivazioni per estrarre dalla testa (**Pop**) **l'indirizzo di memoria in esso contenuto da dove riprendere l'esecuzione.**

Esecuzione di un SOTTOPROGRAMMA: SCHEMATIZZAZIONE





Ambiente di visibilità di un SOTTOPROGRAMMA

DEF: Con il termine "Ambiente (di visibilità) di un sottoprogramma" definiamo l'insieme delle risorse (variabili, costanti, tipi di dato, sottoprogrammi, parametri) alle quali esso può accedere.

Tale ambiente è costituito da:

- un **Ambiente** (di visibilità) **Locale** ossia costituito dalle risorse dichiarate al suo interno (**risorse locali**);
- un **Ambiente** (di visibilità) **Globale** ossia costituito dalle risorse utilizzabili da tutti i sottoprogrammi (**risorse globali**).

N.B. Un corretto stile di programmazione impone di non utilizzare l'ambiente (di visibilità) **globale di un sottoprogramma ma di privilegiare quello locale**

Regole di visibilità o "SCOPE"

PREMESSA Per **oggetti** informatici qui intenderemo **costanti**, **variabili** e **tipi di dato**

Esistono delle **regole** per determinare il campo di **visibilità** (o **«scope»**) degli **oggetti *globali*** e ***locali*** di un programma (algoritmo).

Si parte dai seguenti tre principi:

1) Un qualsiasi **oggetto** (**globale** e/o **locale**) non può essere usato se prima non viene dichiarato;

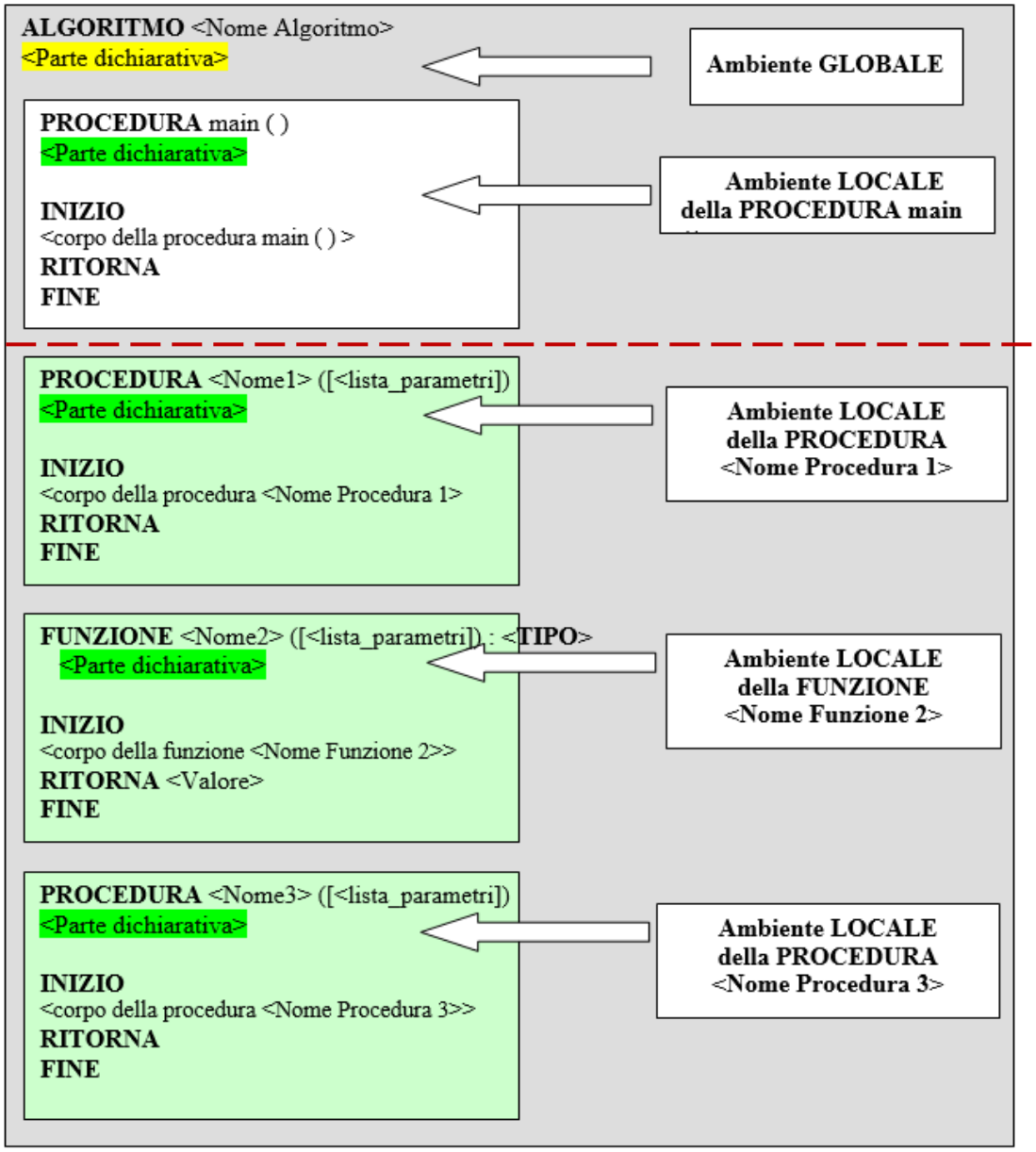
2) Gli **oggetti globali** sono accessibili a tutto il programma (algoritmo) ed a tutti i sottoprogrammi che lo compongono;

3) Un **oggetto** dichiarato in un sottoprogramma (**locale**) ha significato solo in quel sottoprogramma ed in tutti quelli in esso dichiarati.

CASO PARTICOLARE: L'OMONIMIA: Nella descrizione di un programma (algoritmo) può accadere che una variabile sia dichiarata **con lo stesso nome** (il tipo potrebbe essere anche non uguale) **tanto a livello globale che a livello locale** all'interno di un sottoprogramma. **COME EVITARE L'AMBIGUITA'?**

RISOLUZIONE CASO PARTICOLARE: Nel caso che essa venga usata in una o più istruzioni all'interno del sottoprogramma, tale variabile locale più interna **oscurerà** l'omonima variabile più esterna (la globale), **impedendone la visibilità** (**SHADOWING**)

Ambiente di visibilità di un SOTTOPROGRAMMA

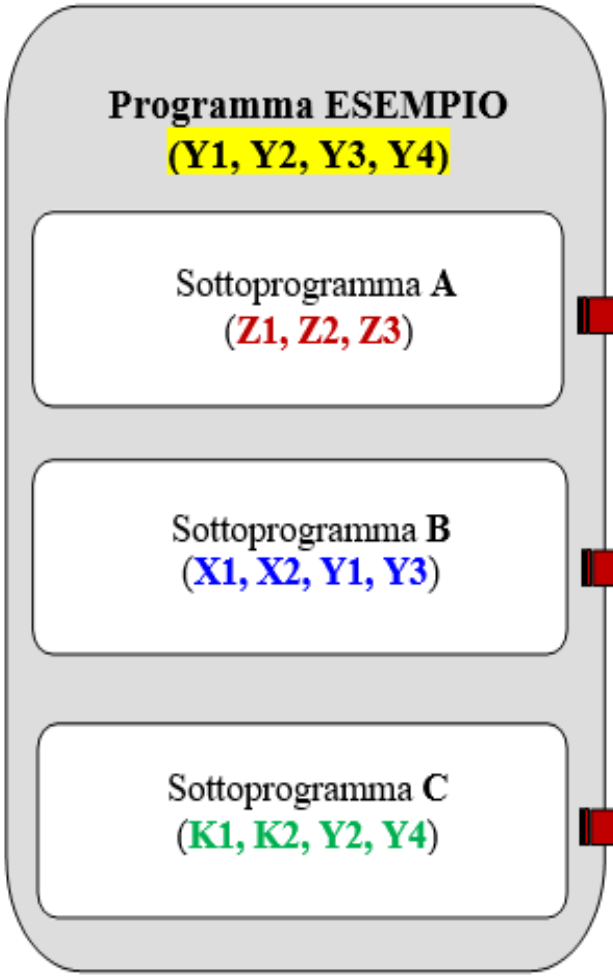


N.B.

Finora abbiamo implementato programmi derivanti da algoritmi costituiti dalla sola **PROCEDURA main()**

D'ora in avanti implementeremo programmi derivanti da algoritmi costituiti da più sottoprogrammi (PROCEDUREe/o FUNZIONI)

Ambiente di visibilità di un SOTTOPROGRAMMA



Il **sottoprogramma A** vede le sue variabili **locali Z1, Z2, Z3** e le variabili **globali Y1, Y2, Y3, Y4**.
Non vede le variabili locali dichiarate nei sottoprogrammi **B** e **C**

Il **sottoprogramma B** vede le sue variabili **locali X1,X2, Y1,Y3** e le variabili **globali Y2** e **Y4** con le sue variabili locali **Y1** e **Y3** che oscurano (shadowing) le omonime variabili globali.
Non vede le variabili locali dichiarate nei sottoprogrammi **A** e **C**

Il **sottoprogramma C** vede le sue variabili **locali K1,K2,Y2,Y4** e le variabili **globali Y1** e **Y3** con le sue variabili locali **Y2** e **Y4** che oscurano (shadowing) le omonime variabili globali.
Non vede le variabili dichiarate nei sottoprogrammi **A** e **B**

SOTTOPROGRAMMA

Appare evidente che è possibile realizzare un **sottoprogramma per ogni sottoproblema non più ulteriormente scomponibile** e che unendo alla fine tutti i sottoprogrammi si ottiene il **programma che risolve il problema principale.**

Definizione: **Il sottoprogramma è una parte del programma in cui viene dettagliata una particolare attività descritta da un sottoalgoritmo**

In altre parole un **sottoprogramma** nell'ambito della programmazione, è un particolare costrutto sintattico di un determinato **linguaggio di programmazione** che permette di **raggruppare una sequenza di istruzioni in un unico blocco**, espletando così una specifica (e in generale più complessa) operazione, azione o elaborazione sui dati del programma stesso in modo tale che, a partire da determinati **input**, restituisca determinati **output**.

Esistono due tipi di **sottoprogramma**:

a) PROCEDURA

b) FUNZIONE

La pseudocodifica dei sottoprogrammi: LA PROCEDURA

DEF: La PROCEDURA è un sottoprogramma che, attivato dall'apposita istruzione di chiamata, svolge le azioni in esso specificate allo scopo di risolvere il (sotto)problema per il quale è stato realizzato.

Con la PSEUDOCODIFICA la procedura viene indicata come segue:

PROCEDURA <Nome Procedura> **(** [**REF** | **VAL** <Nome param 1>**:** <Tipo param 1> **,**
REF | **VAL** <Nome param 2**>:** <Tipo param 2> ,
……..,
REF | **VAL** <Nome param N**>:** <Tipo param N>] **)**

< Sezione dichiarativa Procedura>

INIZIO

< Corpo della Procedura>

RITORNA

FINE

N.B. Gli specificatori di passaggio **REF** e **VAL** verranno illustrati nelle slide seguenti

Una PROCEDURA è caratterizzata da:

- **un nome**, grazie al quale è possibile richiamarla ed identificarla univocamente;
- **una lista di parametri** che è *opzionale* e permette lo scambio in input e/o in output di informazioni tra il *programma chiamante* ed la procedura stessa ossia il *programma chiamato*.

La pseudocodifica dei sottoprogrammi: LA FUNZIONE

DEF: La FUNZIONE è un sottoprogramma che, attivato dall'apposita istruzione di chiamata, oltre a svolgere le azioni in esso specificate allo scopo di risolvere il (sotto)problema per il quale è stato realizzato, può restituire un valore.
Questo valore è restituito nel nome della funzione e può essere usato direttamente come elemento di una istruzione di assegnazione o in una espressione oppure come output.

Con la PSEUDOCODIFICA la funzione viene indicata come segue:

FUNZIONE <Nome Funzione> **(** **[** **REF** **|** **VAL** <Nome param 1>**:** <Tipo param 1> **,**
REF **|** **VAL** <Nome param 2**>:** <Tipo param 2> ,
……..,
REF **|** **VAL** <Nome param n**>:** <Tipo param N> **]** **) : < Tipo Risultato>**

< Sezione dichiarativa Funzione>

INIZIO

< Corpo della Funzione>

RITORNA <Valore>

FINE

N.B. Gli specificatori di passaggio **REF** e **VAL** verranno illustrati nelle slide seguenti

Una FUNZIONE è dunque caratterizzata da:

- **un nome**, grazie al quale è possibile richiamarla ed identificarla univocamente;
- **una lista di parametri** che è *opzionale* e permette lo scambio in input e/o in output di informazioni tra il *programma chiamante* ed la procedura stessa ossia il *programma chiamato*;
- **un valore** ritornato direttamente nel nome della funzione

Lo scambio di dati con un sottoprogramma: I PARAMETRI

Definizione: I parametri sono oggetti messi a disposizione da tutti i linguaggi di programmazione per rendere i sottoprogrammi autonomi ed indipendenti (funzionalmente indipendenti) dai dati del programma chiamante.

Come le comuni variabili, i parameri sono caratterizzati da:

- **un identificatore o nome;**
- **un tipo;**
- **un valore**.

Inoltre per svolgere il ruolo di **INTERFACCIAMENTO** tra **programma chiamante** e **programma chiamato**, i paramentri sono caratterizzati da:

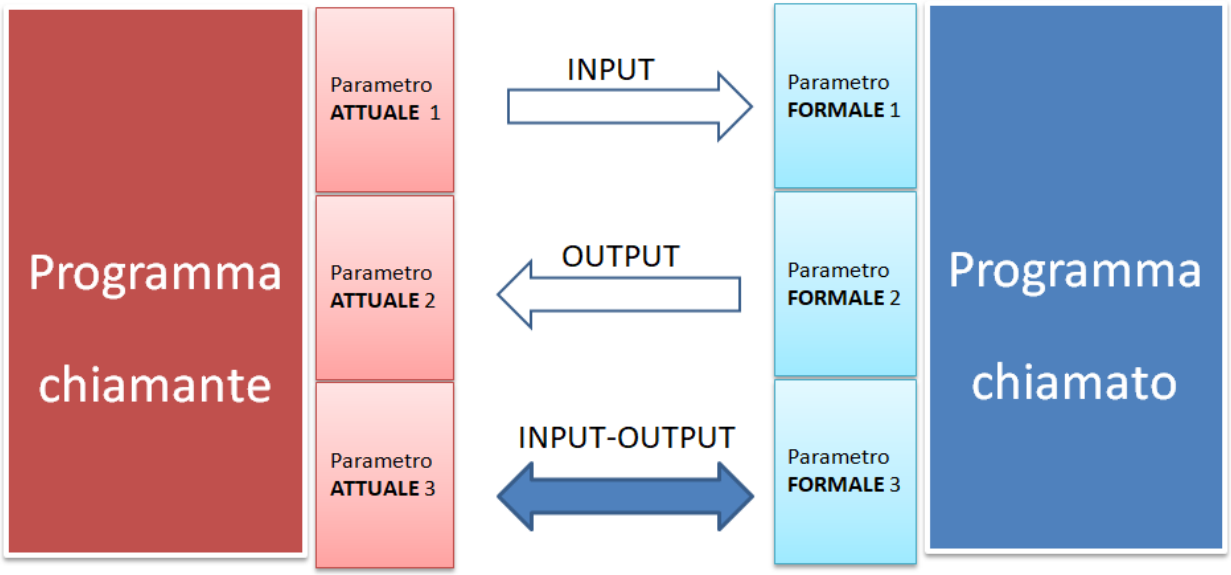
- **un numero;**
- **una posizione;**
- **una direzione.**

Sono dunque oggetti attraverso i quali averrà l'input dei dati (dal programma chiamante al sottoprogramma) ed il conseguente output dei risultati (dal sottoprogramma al programma chiamante).

Lo scambio di dati con un sottoprogramma: I PARAMETRI

Al momento della dichiarazione del sottoprogramma si deve specificare la tipologia dei parametri da utilizzare (**identificatore o nome, tipo e posizione**): essi prendono il nome di **parametri formali.**

Al momento della chiamata del sottoprogramma occorrerà specificare la tipologia dei parametri da trasmettere (**identificatore o nome, tipo e posizione**): essi prendono il nome di **parametri attuali.**



- il **numero** dei parametri attuali deve coincidere con il numero dei parametri formali;
- i parametri attuali ed i parametri formali che si trovano in **posizione** omologa, devono avere lo stesso **tipo**;
- la **direzione** ha a che fare con l'essere parametro solo di input, solo di output o di input ed output.

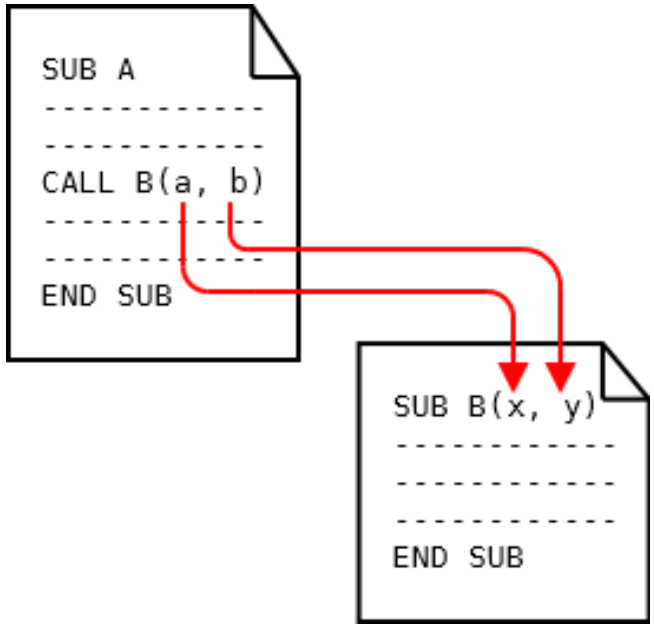
Passaggio dei parametri

Con il termine **passaggio o trasmissione dei parametri** intendiamo l'operazione con la quale il valore dei **parametri attuali** (appartenenti al programma chiamante) viene associato (trasmesso) a quello dei **parametri formali** (appartenenti al programma chiamato)

Esistono **due tipologie di passaggio di parametri**:

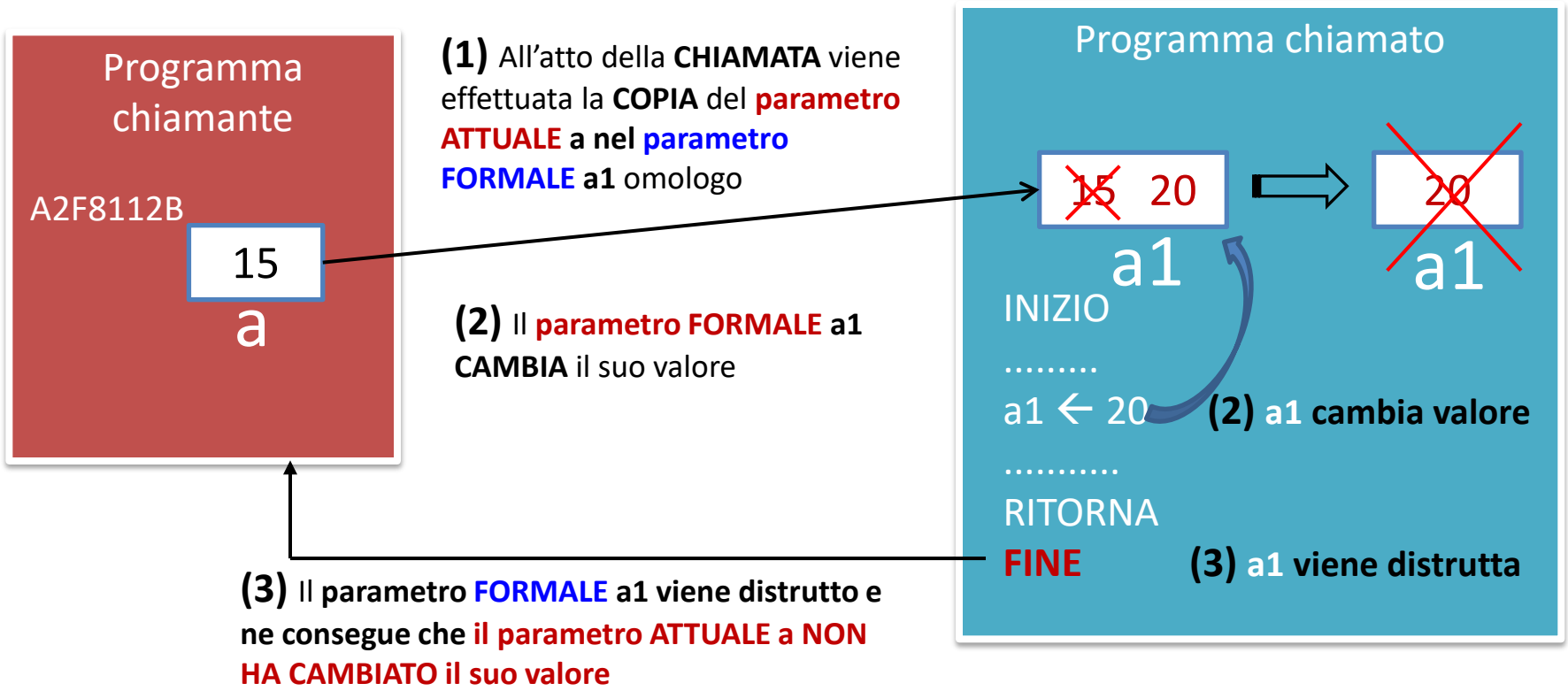
a) **Passaggio dei parametri per VALORE o BY VALUE**

b) **Passeggio dei parametri per RIFERIMENTO o BY REFERENCE (o per INDIRIZZO)**



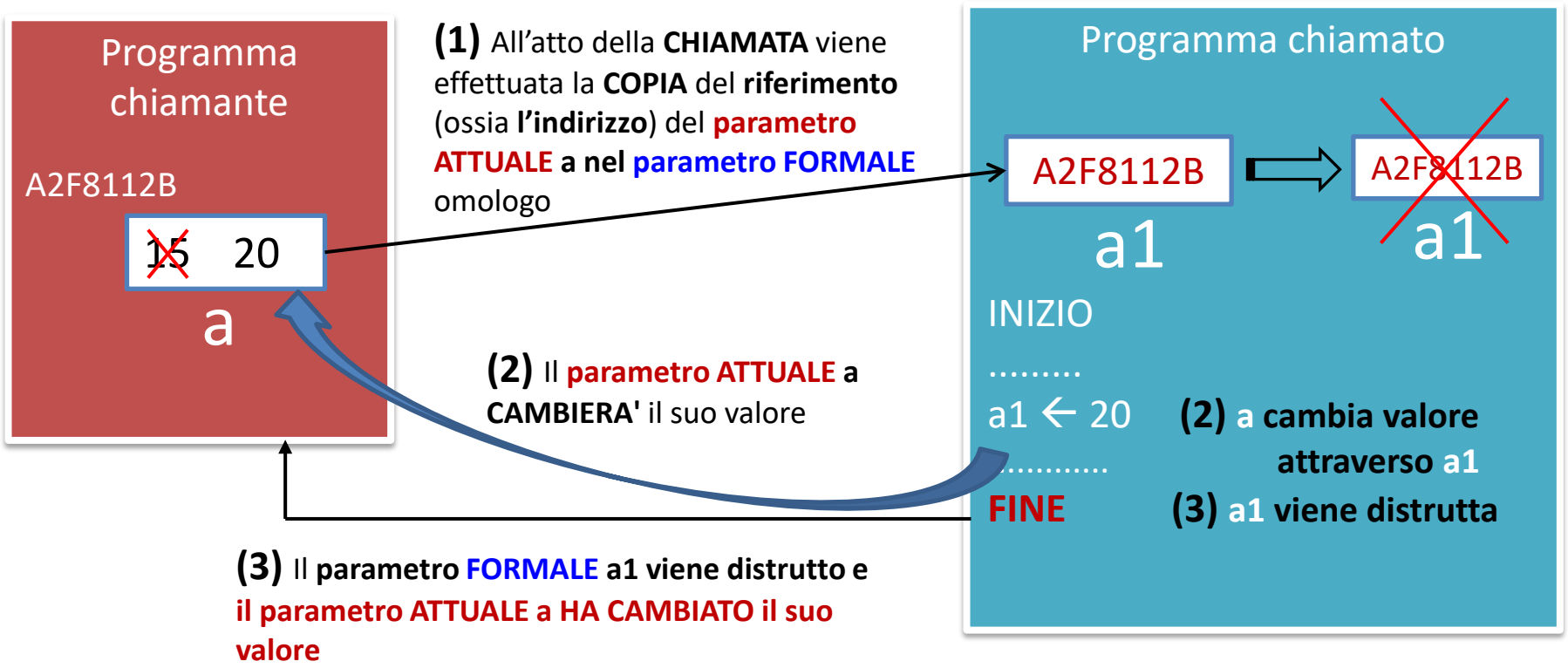
Passaggio dei parametri per VALORE o BY VALUE

a) Passaggio dei parametri per VALORE o BY VALUE



Passaggio dei parametri per riferimento o BY REFERENCE

b) Passaggio dei parametri per RIFERIMENTO o BY REFERENCE (o per INDIRIZZO)



Variabili (parametri) di tipo PUNTATORE

Non tutti i linguaggi di programmazione permettono l'uso dell'allocazione dinamica della memoria da parte del programmatore.
E' indubbio che questa possibilità fornisca a tali linguaggi una marcia in più.

Sia il linguaggio C sia il linguaggio il C++ fortunatamente permettono al programmatore di gestire l'allocazione dinamica della memoria a disposizione un particolare tipo di dato chiamato **puntatore**.

Definizione: una variabile di tipo puntatore contiene un valore intero (espresso in esadecimale) che rappresenta l'indirizzo della locazione di memoria nella quale è memorizzato il dato cui si riferisce

Conseguenze:

1) Un puntatore quindi non contiene direttamente dati come le altre variabili di altri tipi, ma contiene un indirizzo di memoria dove reperire i dati.

2) Per riferirci al valore del dato puntato da un puntatore useremo il simbolo ***** davanti al nome della variabile puntatore

3) Per assegnare l'indirizzo di una cella di memoria ad un puntatore useremo il simbolo **&** davanti al nome della variabile

4) Una variabile puntatore occupa sempre la stessa quantità di memoria indipendentemente dal tipo di dato puntato.

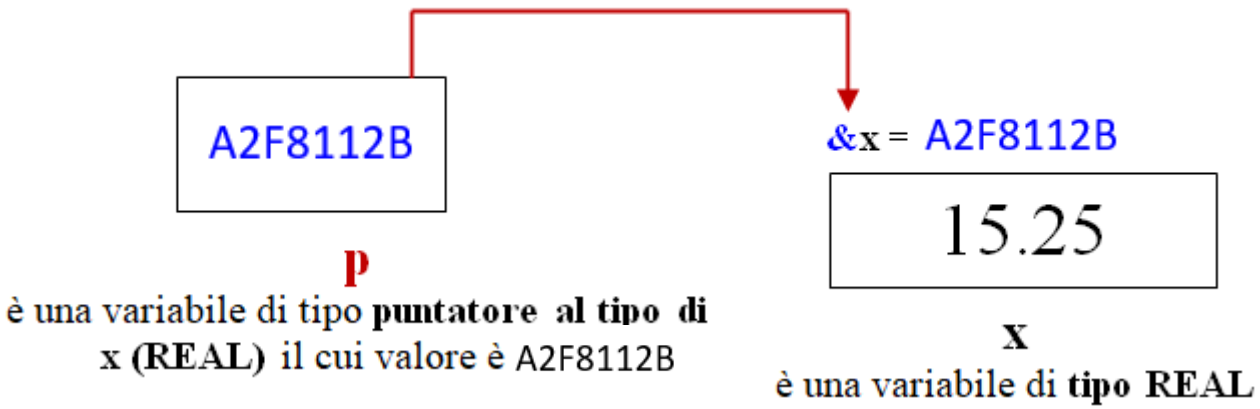
Variabili (parametri) di tipo PUNTATORE

Esempio:

.....
p : **PUNTATORE A REAL**
x : **REAL**
.....
x ← 15.25
p ← **NULL** // **NULL** rappresenta “lo zero” dei puntatori

p ← &x // si realizza il collegamento tra puntatore **p** e variabile puntata **x**
Scrivi (*p) // è possibile utilizzare il contenuto del dato attraverso un suo puntatore
.....

Graficamente



Variabili (parametri) di tipo REFERENCE o di tipo PUNTATORE

NOTA BENE: ENTRAMBI CONTENGONO INDIRIZZI MA…….

Una variabile (parametro) **REFERENCE** contiene l'indirizzo di un'altra variabile (parametro) e permette di accedere al dato "riferito" o "puntato" **SENZA DOVER UTILIZZARE nella (PSEUDO)codifica l'OPERATORE *** (detto operatore di "INDIREZIONE" o "DEFERENZIAZIONE")
(N.B. Noi utilizzeremo questa semplificazione)

invece

Una variabile (parametro) **PUNTATORE** contiene anch'essa l'indirizzo di un'altra variabile (parametro) ma permette di accedere al dato "riferito" o "puntato" **ESCLUSIVAMENE UTILIZZANDO nella (PSEUDO)codifica l'OPERATORE *** sopra citato

NON TUTTI I LINGUAGGI DI PROGRAMMAZIONE PREVEDONO AMBEDUE I TIPI DI DATO (REFERENCE e PUNTATORE)

Ad esempio:
il **LINGUAGGIO C** ha solo variabili **PUNTATORI** per svolgere di passaggio di parametri per **RIFERIMENTO** o **BY REFERENCE (per INDIRIZZO)**, mentre il **LINGUAGGIO C++** le possiede entrambi

Esempio svolto: SOMMA DI DUE INTERI (PROCEDURA)

Esercizio A: Scrivere la PSEUDOCODIFICA di una PROCEDURA che effettua la somma di due numeri interi (PROCEDURA SommaP())

Breve Analisi: per poter calcolare la somma di due numeri, indipendentemente dal tipo, abbiamo bisogno in INPUT del valore del primo addendo e del valore del secondo addendo.
Inoltre il sottoprogramma (in questo caso una PROCEDURA) dovrà produrre come OUTPUT il valore della somma calcolato addizionando i due addendi

DATI DI INPUT DEL SOTTOPROBLEMA: PROCEDURA SommaP()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
a	INT	STATICA	TUTTI	Primo parametro FORMALE, PASSATO PER VALORE, che conterrà il valore del primo addendo
b	INT	STATICA	TUTTI	Secondo parametro FORMALE, PASSATO PER VALORE, che conterrà il valore del secondo addendo

DATI DI OUTPUT DEL SOTTOPROBLEMA: PROCEDURA SommaP()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
s	INT	STATICA	TUTTI	Terzo parametro FORMALE, PASSATO PER RIFERIMENTO, che conterrà il valore della somma da calcolare dei primi due addendi

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: PROCEDURA SommaP()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

Esempio svolto: SOMMA DI DUE INTERI (PROCEDURA)

Esercizio A: Scrivere la PSEUDOCODIFICA di una PROCEDURA che effettua la somma di due numeri interi (PROCEDURA SommaP())

Breve Analisi: per poter calcolare la somma di due numeri, indipendentemente dal tipo, abbiamo bisogno in INPUT del valore del primo addendo e del valore del secondo addendo. Inoltre il sottoprogramma (in questo caso una PROCEDURA) dovrà produrre come OUTPUT il valore della somma calcolato addizionando i due addendi

Prototipo o segnatura della **PROCEDURA**

PROCEDURA SommaP (VAL a : INT, VAL b : INT, REF s : INT)
INIZIO
s ← a + b
RITORNA
FINE

TIPOLOGIA DI **PASSAGGIO**

PARAMETRI **FORMALI**

Esempio svolto: SOMMA DI DUE INTERI (FUNZIONE)

Esercizio B: Scrivere la PSEUDOCODIFICA di una FUNZIONE che effettua la somma di due numeri interi (FUNZIONE SommaF())

Breve Analisi: per poter calcolare la somma di due numeri, indipendentemente dal tipo, abbiamo bisogno in INPUT del valore del primo addendo e del valore del secondo addendo. Inoltre il sottoprogramma (in questo caso una FUNZIONE) dovrà produrre come OUTPUT il valore della somma calcolato addizionando i due addendi

DATI DI INPUT DEL SOTTOPROBLEMA: FUNZIONE SommaF()

Nome variabile	Tipo dati	Tipo Allocazione	Valori ammessi	Descrizione
a	INT	STATICA	TUTTI	Primo parametro FORMALE, PASSATO PER VALORE che conterrà il valore del primo addendo
b	INT	STATICA	TUTTI	Secondo parametro FORMALE, PASSATO PER VALORE che conterrà il valore del secondo addendo

DATI DI OUTPUT DEL SOTTOPROBLEMA: FUNZIONE SommaF()

Nome variabile	Tipo dati	Tipo Allocazione	Valori ammessi	Descrizione

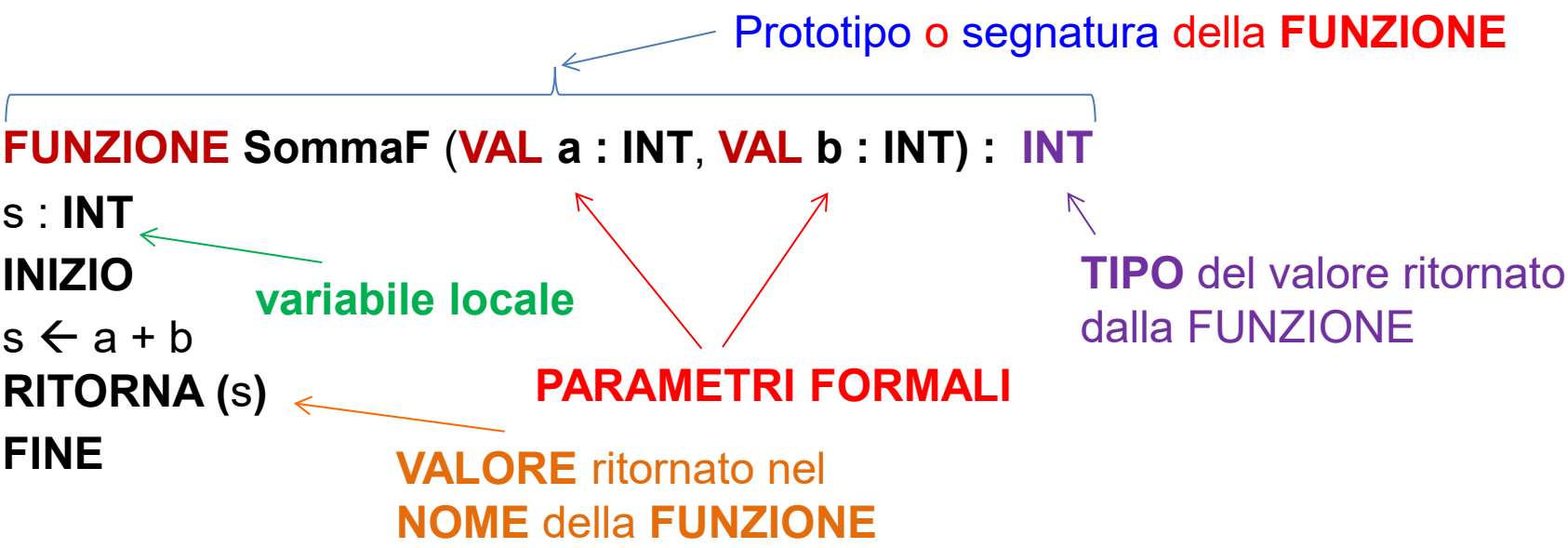
DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: FUNZIONE SommaF()

Nome variabile oppure nome costante	Tipo dati	Tipo Allocazione	Valori ammessi	Descrizione
s	INT	STATICA	TUTTI	Variabile contenente il valore della somma dei due addendi passati da restituire nel nome della funzione

Esempio svolto: SOMMA DI DUE INTERI (FUNZIONE)

Esercizio B: Scrivere la PSEUDOCODIFICA di una FUNZIONE che effettua la somma di due numeri interi (FUNZIONE SommaF())

Breve Analisi: per poter calcolare la somma di due numeri, indipendentemente dal tipo, abbiamo bisogno in INPUT del valore del primo addendo e del valore del secondo addendo. Inoltre il sottoprogramma (in questo caso una FUNZIONE) dovrà produrre come OUTPUT il valore della somma calcolato addizionando i due addendi



N.B. Ci si sarebbe anche potuto risparmiare l'utilizzo della variabile locale **s** eseguendo il calcolo **a + b** direttamente con l'istruzione **RITORNA**

Esempio svolto: SOMMA DI DUE INTERI (con main())

Esercizio Riepilogativo: Scrivere un algoritmo che sia in grado di mostrare a video il valore della somma di due numeri interi qualsiasi **x** e **y** calcolato prima attraverso la procedura **SommaP()**, poi attraverso la funzione **SommaF**().

Esempio:

Se all'interno della procedura main() l'utente dovesse inserire i seguenti valori:

x = 3 e **y = 5**

allora:

- dopo la chiamata al sottoprogramma di tipo PROCEDURA **SommaP()** dovrò ottenere **8**;

- dopo la chiamata al sottoprogramma di tipo FUNZIONE **SommaF()** dovrò ottenere ugualmente **8**.

ALGORITMO Somma_2_Interi_SOTTOPROGRAMMA

PROCEDURA main()

DATI DI INPUT DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
x	INT	STATICA	TUTTI	Primo addendo acquisito da tastiera
y	INT	STATICA	TUTTI	Secondo addendo acquisito da tastiera

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
somma	INT	STATICA	TUTTI	Risultato della somma di **x** e d **y** mostrato a video, calcolato prima con la procedura **SommaP()** e poi con la funzione **SommaF()**

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

ALGORITMO Somma_2_Interi_SOTTOPROGRAMMA

PROCEDURA SommaP()

DATI DI INPUT DEL SOTTOPROBLEMA: PROCEDURA SommaP()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
a	INT	STATICA	TUTTI	Primo parametro FORMALE, PASSATO PER VALORE, che conterrà il valore del primo addendo
b	INT	STATICA	TUTTI	Secondo parametro FORMALE, PASSATO PER VALORE, che conterrà il valore del secondo addendo

DATI DI OUTPUT DEL SOTTOPROBLEMA: PROCEDURA SommaP()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
s	INT	STATICA	TUTTI	Terzo parametro FORMALE, PASSATO PER RIFERIMENTO, che conterrà il valore della somma da calcolare dei primi due addendi

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: PROCEDURA SommaP()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

ALGORITMO Somma_2_Interi_SOTTOPROGRAMMA

PROCEDURA SommaF()

DATI DI INPUT DEL SOTTOPROBLEMA: FUNZIONE SommaF()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
a	INT	STATICA	TUTTI	Primo parametro FORMALE, PASSATO PER VALORE che conterrà il valore del primo addendo
b	INT	STATICA	TUTTI	Secondo parametro FORMALE, PASSATO PER VALORE che conterrà il valore del secondo addendo

DATI DI OUTPUT DEL SOTTOPROBLEMA: FUNZIONE SommaF()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: FUNZIONE SommaF()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
s	INT	STATICA	TUTTI	Variabile contenente il valore della somma dei due addendi passati da restituire nel nome della funzione

ALGORITMO Somma_2_Interi_SOTTOPROGRAMMA

```
PROCEDURA main ( )
x, y, somma : INT
INIZIO
Leggi(x)
Leggi(y)
/* Chiamata alla PROCEDURA */
SommaP (x, y, somma)
Scrivi(somma)
/* Chiamata alla FUNZIONE */
somma ← SommaF(x, y)
Scrivi(somma)
RITORNA
FINE
```

```
PROCEDURA SommaP (VAL a : INT, VAL b : INT, REF s : INT)
INIZIO
s ← a + b
RITORNA
FINE
```

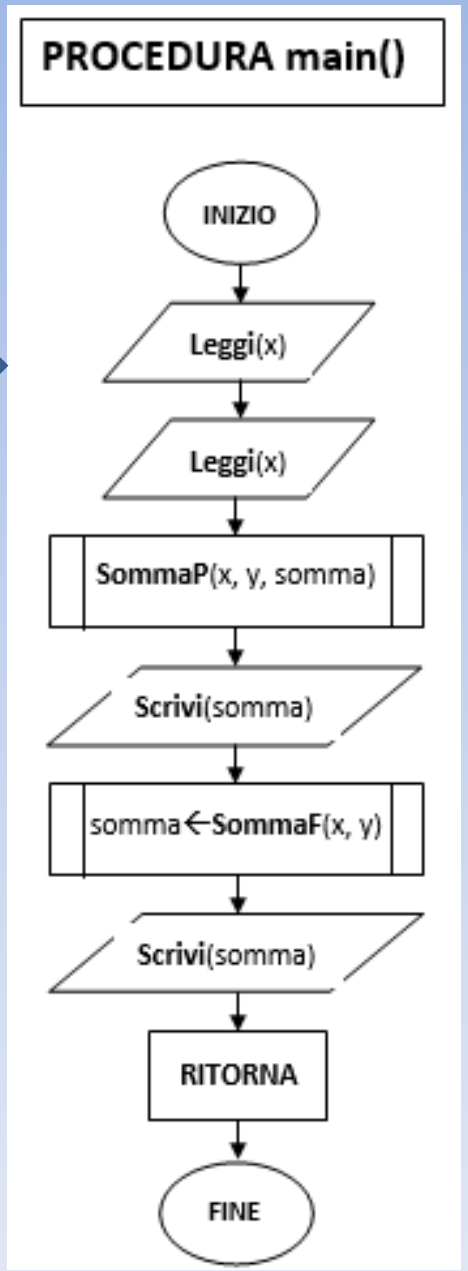
```
FUNZIONE SommaF (VAL a : INT, VAL b : INT) :  INT
s : INT
INIZIO
s ← a + b
RITORNA (s)
FINE
```

SOLUZIONE PROPOSTA
(PSEUDOCODIFICA)

ALGORITMO Somma_2_Interi_SOTTOPROGRAMMA

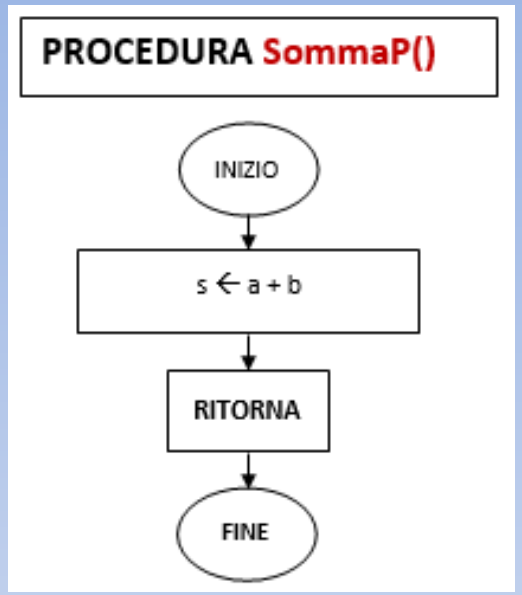
PROCEDURA main ( )
x, y, somma : INT
INIZIO
Leggi(x)
Leggi(y)
/* Chiamata alla PROCEDURA */
SommaP (x, y, somma)
Scrivi(somma)
/* Chiamata alla FUNZIONE */
somma ← SommaF(x, y)
Scrivi(somma)
RITORNA
FINE

SOLUZIONE PROPOSTA
(FLOWCHART)



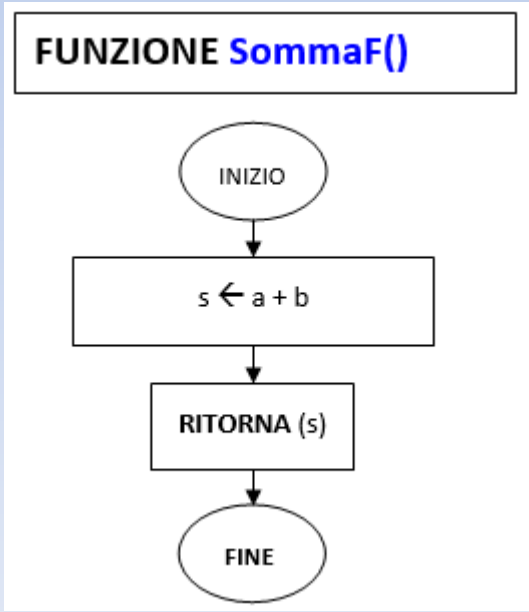
ALGORITMO Somma_2_Interi_SOTTOPROGRAMMA

```
PROCEDURA SommaP (VAL a : INT, VAL b : INT, REF s : INT)
INIZIO
s ← a + b
RITORNA
FINE
```



SOLUZIONE PROPOSTA
(FLOWCHART)

```
FUNZIONE SommaF (VAL a : INT, VAL b : INT) :  INT
s : INT
INIZIO
s ← a + b
RITORNA (s)
FINE
```



ALGORITMO Somma_2_Interi_SOTTOPROGRAMMA

```
PROCEDURA main ( )
x, y, somma : INT
INIZIO
Leggi(x)
Leggi(y)
/* Chiamata alla PROCEDURA */
SommaP (x, y, somma)
Scrivi(somma)
/* Chiamata alla FUNZIONE */
somma ← SommaF(x, y)
Scrivi(somma)
RITORNA
FINE
```

```
PROCEDURA SommaP (VAL a : INT, VAL b : INT, REF s : INT)
INIZIO
s ← a + b
RITORNA
FINE
```

```
FUNZIONE SommaF (VAL a : INT, VAL b : INT) :  INT
s : INT
INIZIO
s ← a + b
RITORNA (s)
FINE
```

E' abbastanza semplice intuire che se **x = 3** e **y = 5** i due sottoprogrammi dovranno essere entrambi in grado di fornire al programma chiamante (in questo caso la procedura main) **il valore 8**

Ma qual è il meccanismo messo in piedi attraverso il passaggio dei parametri e come possiamo controllarlo?

Ancora una volta sarà indispensabile l'utilizzo delle tabelle di traccia

Esempio svolto: SOMMA DI DUE INTERI (tabelle di traccia)

```
ALGORITMO Somma
PROCEDURA main ( )
x, y, somma : INT
INIZIO
Leggi(x)
Leggi(y)
SommaP (x, y, somma)
Scrivi(somma)
somma ← SommaF(x, y)
Scrivi(somma)
RITORNA
FINE
```

3° param. ATTUALE nel 3° param. FORMALE

2° param. ATTUALE nel 2° param. FORMALE

1° param. ATTUALE nel 1° param. FORMALE

```
PROCEDURA SommaP (VAL a : INT, VAL b : INT, REF s : INT)
INIZIO
s ← a + b
RITORNA
FINE
```

N.B. Al momento della chiamata lo scambio di valori tra parametri ATTUALI e FORMALI , INDIPENDENTEMENTE DAL TIPO DI PASSAGGIO UTILIZZATO, **avviene esclusivamente per POSIZIONE** nel rispetto del TIPO posseduto **senza assolutamente guardare il NOME** del parametro (regole di visibilità)

Programma chiamante

Programma chiamato

x	y	somma
3	5	-

3 5 -

SommaP(x, y, somma)

Val a	Val b	Ref s
3	5	-
3	5	8

Calcoli nella PROCEDURA
s ← a + b (s = 3 + 5 = 8)

RITORNA

x	y	somma
3	5	-
3	5	8

Esempio svolto: SOMMA DI DUE INTERI (tabelle di traccia)

```
ALGORITMO Somma
PROCEDURA main ( )
x, y, somma : INT
INIZIO
Leggi(x)
Leggi(y)
……..
somma ← SommaF(x, y)
Scrivi(somma)
RITORNA
FINE
```

2° param. ATTUALE nel 2° param. FORMALE

1° param. ATTUALE nel 1° param. FORMALE

```
FUNZIONE SommaF (VAL a : INT, VAL b : INT) :  INT
s : INT
INIZIO
s ← a + b
RITORNA (s)
FINE
```

Programma chiamante

Programma chiamato

x	y	somma
3	5	-

3 5

somma ←SommaF(x, y)

Val a	Val b	s
3	5	-
3	5	8

x	y	somma
3	5	-
3	5	8

8

RITORNA (s)

Calcoli nella FUNZIONE
s ← a + b (s = 3 + 5 = 8)

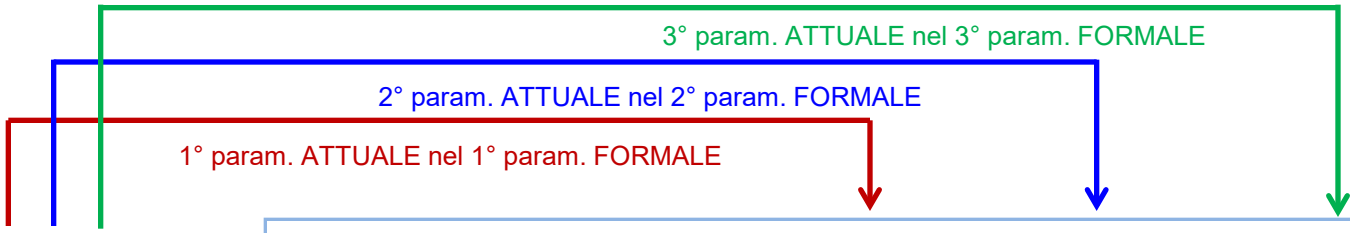
N.B. Al momento della chiamata lo scambio di valori tra parametri ATTUALI e FORMALI , INDIPENDENTEMENTE DAL TIPO DI PASSAGGIO UTILIZZATO, **avviene esclusivamente per POSIZIONE** nel rispetto del TIPO posseduto **senza assolutamente guardare il NOME** del parametro (regole di visibilità)

Esercizi svolti sul passaggio dei parametri: PROCEDURA

Programma chiamante

N.B. Al momento della chiamata lo scambio di valori tra parametri ATTUALI e FORMALI , INDIPENDENTEMENTE DAL TIPO DI PASSAGGIO UTILIZZATO, **avviene esclusivamente per POSIZIONE** nel rispetto del TIPO posseduto **senza assolutamente guardare il NOME** del parametro (regole di visibilità)

```
ALGORITMO Passaggio1
PROCEDURA main ( )
x, y, z : INT
INIZIO
Leggi (x)
Leggi (y)
Leggi (z)
/* 1° chiamata */
ChangeMe1 (y, z, x)
Scrivi (x)
Scrivi (y)
Scrivi (z)
/* 2° chiamata */
ChangeMe1 (z, y, x)
Scrivi (x)
Scrivi (y)
Scrivi (z)
RITORNA
FINE
```




```
PROCEDURA ChangeMe1 (REF x: INT, VAL y: INT, REF z: INT)
i: INT
INIZIO
PER i ← 1 A z ESEGUI
    x ← 2*y – x + z
    y ← 2*x – y – z
    i ← i + 1
FINE PER
z ← z - 1
RITORNA
FINE
```

Programma chiamato

Esercizio: Supponendo che inizialmente le variabili siano così valorizzate: **x = 2, y = 2, z = 3 cosa verrà mostrato a video dopo ciascuna chiamata?**

Esercizi svolti sul passaggio dei parametri: PROCEDURA

Per risolvere l'esercizio proposto devo utilizzare **le tabelle di traccia** facendo attenzione rispetto quanto detto finora

La scelta dei nomi e dell'ordine nel quale inserire i **parametri ATTUALI** (programma chiamante) nella tabella di traccia è **ARBITRARIO**.
N.B. Però una volta fissato, meglio non modificarlo per evitare inutile confusione.

La scelta dei nomi e dell'ordine nel quale inserire i **parametri FORMALI** (programma chiamato) nella tabella di traccia è **OBBLIGATO** in quanto **DEVE CORRISPONDERE** all'ordine scelto dal progettista nel PROTOTIPO o SEGNATURA del SOTTOPROGRAMMA

Programma chiamante

Programma chiamato

x	y	z
2	2	3

2 3 2

ChangeMe1(y, z, x)

/ 1° CHIAMATA */*

Ref	Val	Ref
x	y	z
2	3	2

Esercizi svolti sul passaggio dei parametri

Programma chiamante

Programma chiamato

x	y	z
2	2	3

2 3 2

ChangeMe1(y, z, x)

/ 1° CHIAMATA */*

		Ref	Val	Ref
i	x	y	z	
-	2	3	2	
1	6	7	2	
2	10	11	2	
3	10	11	1	

Calcoli: 1° Chiamata (nella procedura **ChangeMe1**)

i ← 1(i=1) prima che inizi ciclo PER

TEST PER (i ≤ z) ossia (1 ≤ 2) VERO

i=1 x ← 2*y – x + z (x = 2*3 – 2 + 2 = 6)

y ← 2*x – y – z (y = 2*6 – 3 – 2 = 7)

i ← i + 1 (i = 1 + 1 = 2)

TEST PER (i ≤ z) ossia (2 ≤ 2) VERO

i=2 x ← 2*y – x + z (x = 2*7 – 6 + 2 = 10)

y ← 2*x – y – z (y = 2*10 – 7 – 2 = 11)

i ← i + 1 (i = 2 + 1 = 3)

TEST PER (i ≤ z) ossia (3 ≤ 2) FALSO exit ciclo PER

Fuori ciclo PER z ← z – 1 (z = 2 – 1 = 1)

RITORNA

x	y	z
2	2	3
1	10	3

Esercizi svolti sul passaggio dei parametri: PROCEDURA

Programma chiamante

Programma chiamato

x	y	z
2	2	3
1	10	3

3 10 1

ChangeMe2(z, y, x)

/ 2° CHIAMATA */*

Ref Val Ref

i	x	y	z
-	3	10	1
1	18	25	1
2	18	25	0

Calcoli: 2° Chiamata (nella procedura **ChangeMe1**)

i ← 1(i=1) prima che inizi ciclo PER

TEST PER (i ≤ z) ossia (1 ≤ 1) VERO

i=1 x ← 2*y – x + z (x = 2*10 – 3 + 1 = 18)

y ← 2*x – y – z (y = 2*18 – 10 – 1 = 25)

i ← i + 1 (i = 1 + 1 = 2)

TEST PER (i ≤ z) ossia (2 ≤ 1) FALSO exit ciclo PER

Fuori ciclo PER z ← z – 1 (z = 1 – 1 = 0)

RITORNA

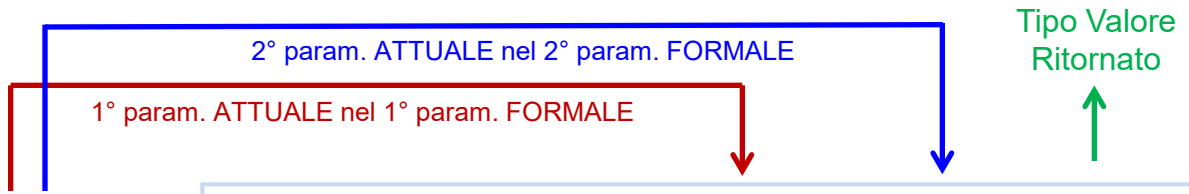
x	y	z
2	2	3
1	10	3
0	10	18

Esercizi svolti sul passaggio dei parametri: FUNZIONE

Programma chiamante

```
ALGORITMO Passaggio2
PROCEDURA main ( )
x, y, z : INT
INIZIO
Leggi (x)
Leggi (y)
Leggi (z)
/* Prima chiamata */
y ← ChangeMe2 (x, z)
Scrivi (x)
Scrivi (y)
Scrivi (z)
/* Seconda chiamata */
x ← ChangeMe2 (z, y)
Scrivi (x)
Scrivi (y)
Scrivi (z)
RITORNA
FINE
```

N.B. Al momento della chiamata lo scambio di valori tra parametri ATTUALI e FORMALI , INDIPENDENTEMENTE DAL TIPO DI PASSAGGIO UTILIZZATO, **avviene esclusivamente per POSIZIONE** nel rispetto del TIPO posseduto **senza assolutamente guardare il NOME** del parametro (regole di visibilità).
Occorre anche tenere presente il valore espressamente ritornato nel NOME della funzione che potrà essere utilizzato in un'assegnazione o un'espressione




```
FUNZIONE ChangeMe2 (VAL x: INT, REF y: INT) : INT
z: INT
INIZIO
SE (x < y)
   ALLORA
      x ← x + 2*y
      y ← y + 1
   ALTRIMENTI
      y ← y + 2*x
      x ← x - 2
FINE SE
z ← x - y
RITORNA (z)
FINE
```

Programma chiamato

Esercizio: Supponendo che inizialmente le variabili siano così valorizzate:
x = 2, y = 2, z = 3 cosa verrà mostrato a video dopo ciascuna chiamata?

Esercizi svolti sul passaggio dei parametri: FUNZIONE

Anche per risolvere l'esercizio proposto devo utilizzare **le tabelle di traccia** facendo attenzione rispetto quanto detto finora

La scelta dei nomi e dell'ordine nel quale inserire i **parametri ATTUALI** (programma chiamante) nella tabella di traccia è **ARBITRARIO**.
N.B. Però una volta fissato, meglio non modificarlo per evitare inutile confusione.

La scelta dei nomi e dell'ordine nel quale inserire i **parametri FORMALI** (programma chiamato) nella tabella di traccia è **OBBLIGATO** in quanto **DEVE CORRISPONDERE** all'ordine scelto dal progettista nel PROTOTIPO o SEGNATURA del SOTTOPROGRAMMA

Programma chiamante

Programma chiamato

x	y	z
2	2	3

y ←**ChangeMe2**(x, z)

/ 1° CHIAMATA */*

2 3

Val Ref

z	x	y
-	2	3

Esercizi svolti sul passaggio dei parametri: FUNZIONE

Programma chiamante

Programma chiamato

x	y	z
2	2	3

2 3

Val Ref

y ←**ChangeMe2**(x, z)

/ 1° CHIAMATA */*

z	x	y
-	2	3
-	8	4
4	8	4

Calcoli: 1° Chiamata (nella funzione **ChangeMe2**)
Test SE (x < y) ossia (2 < 3) VERO - RAMO ALLORA
x ← x + 2*y (x = 2 + 2*3 = 8)
y ← y + 1 (y = 3 + 1 = 4)
z ← x – y (z = 8 – 4 = 4)

x	y	z
2	2	3
2	4	4

4

RITORNA (z)

Esercizi svolti sul passaggio dei parametri: FUNZIONE

Programma chiamante

x	y	z
2	2	3
2	4	4

x ←**ChangeMe2**(z, y)

/ 2° CHIAMATA */*

4 4

Programma chiamato

Val Ref

z	x	y
-	4	4
-	2	12
-10	2	12

Calcoli: 2° Chiamata (nella funzione **ChangeMe2**)
Test SE (x < y) ossia (4 < 4) FALSO - RAMO ALTRIMENTI
y ← y + 2*x (y = 4 + 2*4 = 12)
x ← x – 2 (x = 4 – 2 = 2)
z ← x – y (z = 2 – 12 = -10)

-10

RITORNA (z)

x	y	z
2	2	3
2	4	4
-10	12	4

Speciale: IMPLEMENTAZIONE IN C

Nelle slide seguenti implementiamo ora, a titolo di esempio, l'algoritmo **Somma** progettato in precedenza, utilizzando **il linguaggio di programmazione C**.

PREMESSA: Nel linguaggio C non si parla di PROCEDURE e FUNZIONI bensì si utilizza per entrambe le tipologie di SOTTOPROGRAMMI, il medesimo termine "FUNZIONE".

Per **il linguaggio** C le **PROCEDURE** sono **FUNZIONI** che non restituiscono "nulla" nel proprio nome, dando a questo "**nulla**" la specificità di un tipo di dato, chiamato "**void**".

Nel **linguaggio C**, a differenza della pseudocodifica, oltre alla **chiamata** ed al **corpo** di ogni "funzione" utilizzata, andrà sempre specificato in ambiente globale il suo **prototipo o segnatura**.

Ora utilizzeremo l'I.D.E. DEV-C++ per sviluppare **in due modi differenti** tale esempio :

a) **Creazione** di un **Progetto** DEV-C++ **monofile** (il codice di tutte le funzioni in un unico file **.C**)

b) **Creazione** di un **Progetto** DEV-C++ **multifile** (un file **.C** per ogni funzione, un file **.H** per i prototipi delle funzioni implementate, un file **.H** per le eventuali costanti utilizzate, un file **.H** per gli eventuali tipi utente definiti)

N.B. La soluzione b) è senza dubbio da preferire alla a) in quanto permette lo sviluppo in "parallelo" (invece che in "serie") di un programma tra più programmatori a patto di avere definito in modo rigoroso i prototipi o le segnature di tutti i sottoprogrammi coinvolti

ALGORITMO Somma

```
PROCEDURA main ( )
x, y, somma : INT
INIZIO
Leggi(x)
Leggi(y)
/* Chiamata alla PROCEDURA */
SommaP (x, y, somma)
Scrivi(somma)
/* Chiamata alla FUNZIONE */
somma ← SommaF(x, y)
Scrivi(somma)
RITORNA
FINE
```

```
PROCEDURA SommaP (VAL a : INT, VAL b : INT, REF s : INT)
INIZIO
s ← a + b
RITORNA
FINE
```

```
FUNZIONE SommaF (VAL a : INT, VAL b : INT) :  INT
s : INT
INIZIO
s ← a + b
RITORNA (s)
FINE
```

Ricordiamo qui la **pseudocodifica** dell'algoritmo **Somma** da implementare **nel linguaggio C** .....

.... e ricordiamo che il **SOTTOPROGRAMMA** **SommaP()** è stato progettato come **PROCEDURA** .....

.... mentre ricordiamo che il **SOTTOPROGRAMMA** **SommaF()** è stato progettato come **FUNZIONE**.....

Speciale: IMPLEMENTAZIONE IN C

a) Progetto DEV-C++ MONOFILE (tutto il codice di tutte le funzioni in un unico file .C)

Creiamo un progetto DEV-C++ in cui ci sia un unico file .C il cui listato è il seguente:

unico_main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  //definizione eventuali COSTANTI
5   ..............................
6  //definizione eventuali TIPI di dato utente (typedef)
7   ..............................
8  //definizione dei PROTOTIPI delle funzioni
9  void SommaP (int a, int b, int* s);     //PROCEDURA SommaP
10 int SommaF (int a, int b);              //Funzione SommaF
11
12 //Funzione main() ----> corpo
13 int main(int argc, char *argv[])
14 {
15 int x, y, somma;
16
17 scanf("%d", &x);
18 scanf("%d", &y);
19
20 //PROCEDURA SommaP  ----> corpo
21 SommaP (x, y, &somma);
22 printf("%d", somma);
23
24 //FUNZIONE SommaF  ----> corpo
25 somma = SommaF(x, y);
26 printf("\n%d", somma);
27
28 return 0;
29 }
```

Qui vanno definite le eventuali **costanti**

Qui vanno definiti gli eventuali **tipi di dato** definiti dall'utente tramite istruzione **typedef**

Qui vanno definiti i **prototipi** o **segnature** delle **FUNZIONI** definite dall'utente

Istruzione di chiamata (o CALL) alla FUNZIONE SommaP *(progettata come PROCEDURA)*

Istruzione di chiamata (o CALL) alla FUNZIONE SommaF *(progettata come FUNZIONE)*

Speciale: IMPLEMENTAZIONE IN C

a) Progetto DEV MONOFILE (tutto il codice di tutte le funzioni in un unico file .C)

Continuazione.....

```
30
31  //Funzione SommaP() ----> corpo
32  void SommaP (int a, int b, int* s)
33  {
34  *s = a + b;
35  return;
36  }
37
38  //Funzione SommaF() ----> corpo
39  int SommaF (int a, int b)
40  {
41  int s;
42
43  s= a + b;
44  return (s);
45  }
```

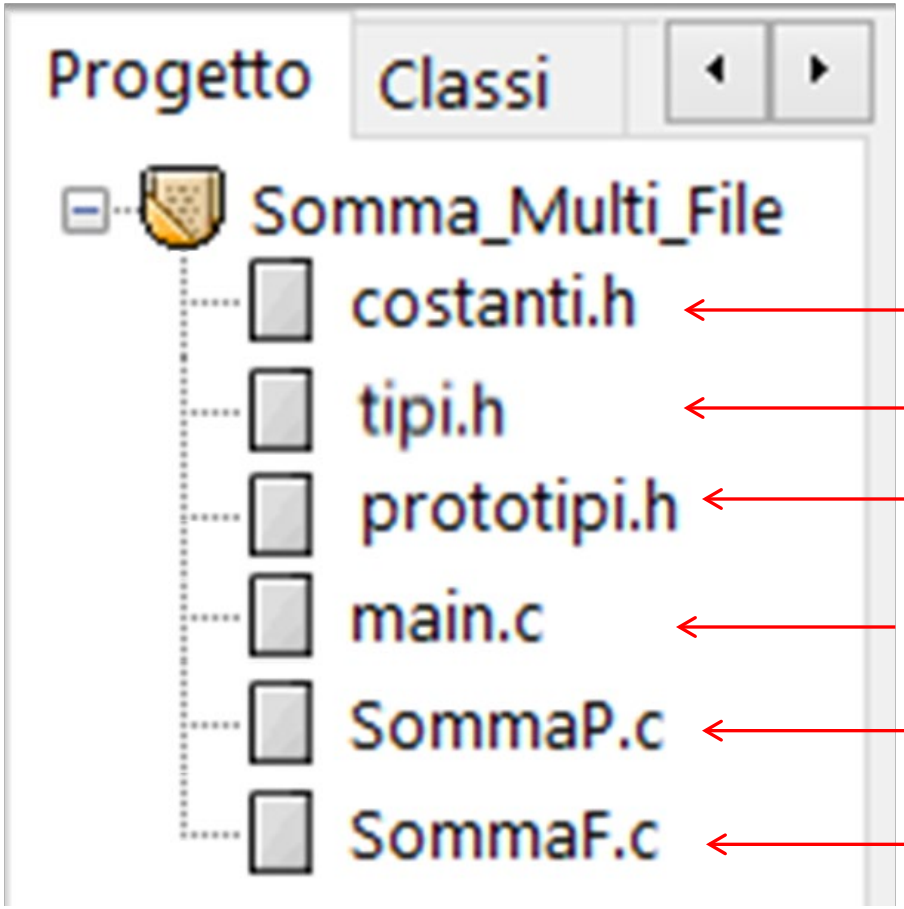
Qui è dettagliato il **corpo** della **funzione SommaP** *(progettata come PROCEDURA)*

Qui è dettagliato il **corpo** della **funzione SommaF** *(progettata come FUNZIONE)*

Speciale: IMPLEMENTAZIONE IN C

b) Progetto DEV-C++ MULTIFILE (tutto il codice di tutte le funzioni in più file)

Creiamo un progetto DEV-C++ (Somma_Multi_file.dev) in cui ci siano i seguenti file:



File .H contenente definite le eventuali **costanti**

File .H contenente gli eventuali **tipi di dato** definiti dall'utente tramite istruzione **typedef**

File .H contenente i **prototipi** o **segnature** delle funzioni definite dall'utente

File .C contenente il corpo della FUNZIONE main()
(n.b. funzione di partenza di ogni programma in linguaggio C)

File .C contenente il corpo della FUNZIONE SommaP
(progettata come PROCEDURA)

File .C contenente il corpo della FUNZIONE SommaF
(progettata come FUNZIONE)

Speciale: IMPLEMENTAZIONE IN C

b) Progetto DEV-C++ MULTIFILE (tutto il codice di tutte le funzioni in più file)

Continuazione.....

HEADER FILE

1 FILE costanti.h: N.B. Nonostante in questo esempio non ci siano costanti, il file è presente nel progetto ma è stato creato vuoto per farne comprendere meglio il ruolo ed il relativo significato...

2 FILE tipi.h: N.B. Nonostante in questo esempio non ci siano tipi definiti dall'utente, il file è presente nel progetto ma è stato creato vuoto per farne comprendere meglio il ruolo ed il relativo significato...

3 FILE prototipi.h: Il file è presente nel progetto e conterrà i prototipi delle due funzioni SommaP() e SommaF()

costanti.h | tipi.h | prototipi.h | main.c | SommaP.c | SommaF.c

```
1  //definizione dei PROTOTIPI delle funzioni
2  void SommaP (int a, int b, int* s);     //PROCEDURA SommaP
3  int SommaF (int a, int b);              //Funzione SommaF
```

Speciale: IMPLEMENTAZIONE IN C

b) Progetto DEV-C++ MULTIFILE (tutto il codice di tutte le funzioni in più file)

Continuazione..... **4 FILE main.c**

costanti.h | tipi.h | prototipi.h | main.c | SommaP.c | SommaF.c

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

//definizione eventuali COSTANTI
#include "costanti.h"
//definizione eventuali TIPI di dato utente (typedef)
#include "tipi.h"
//definizione dei PROTOTIPI delle funzioni
#include "prototipi.h"

//Funzione main() ----> corpo
int main(int argc, char *argv[])
{
int x, y, somma;

scanf("%d", &x);
scanf("%d", &y);

//PROCEDURA SommaP  ----> corpo
SommaP (x, y, &somma);
printf("%d", somma);

//FUNZIONE SommaF  ----> corpo
somma = SommaF(x, y);
printf("\n%d", somma);

return 0;
}
```

In **AMBIENTE GLOBALE** va inserita l'Inclusione degli **HEADER file .h** definiti dall'utente i cui nomi devono essere racchiusi tra **"** e **"** se non presenti nel path delle librerie di sistema (se invece lo fossero, si potrebbero utilizzare anche in questo caso per la loro inclusione i simboli **<** e **>**)

Istruzione di chiamata (o CALL) alla FUNZIONE SommaP *(progettata come PROCEDURA)*

Istruzione di chiamata (o CALL) alla FUNZIONE SommaF *(progettata come FUNZIONE)*

Speciale: IMPLEMENTAZIONE IN C

b) Progetto DEV-C++ MULTIFILE (tutto il codice di tutte le funzioni in più file)

Continuazione..... **5 FILE SommaP.c**

costanti.h | tipi.h | prototipi.h | main.c | SommaP.c | SommaF.c

```
1  //Funzione SommaP() ----> corpo
2  void SommaP (int a, int b, int* s)
3  {
4  *s = a + b;
5  return;
6  }
```

In questo file .C è dettagliato il **corpo** della **funzione SommaP** *(progettata come PROCEDURA)*

Continuazione..... **6 FILE SommaF.c**

costanti.h | tipi.h | prototipi.h | main.c | SommaP.c | SommaF.c

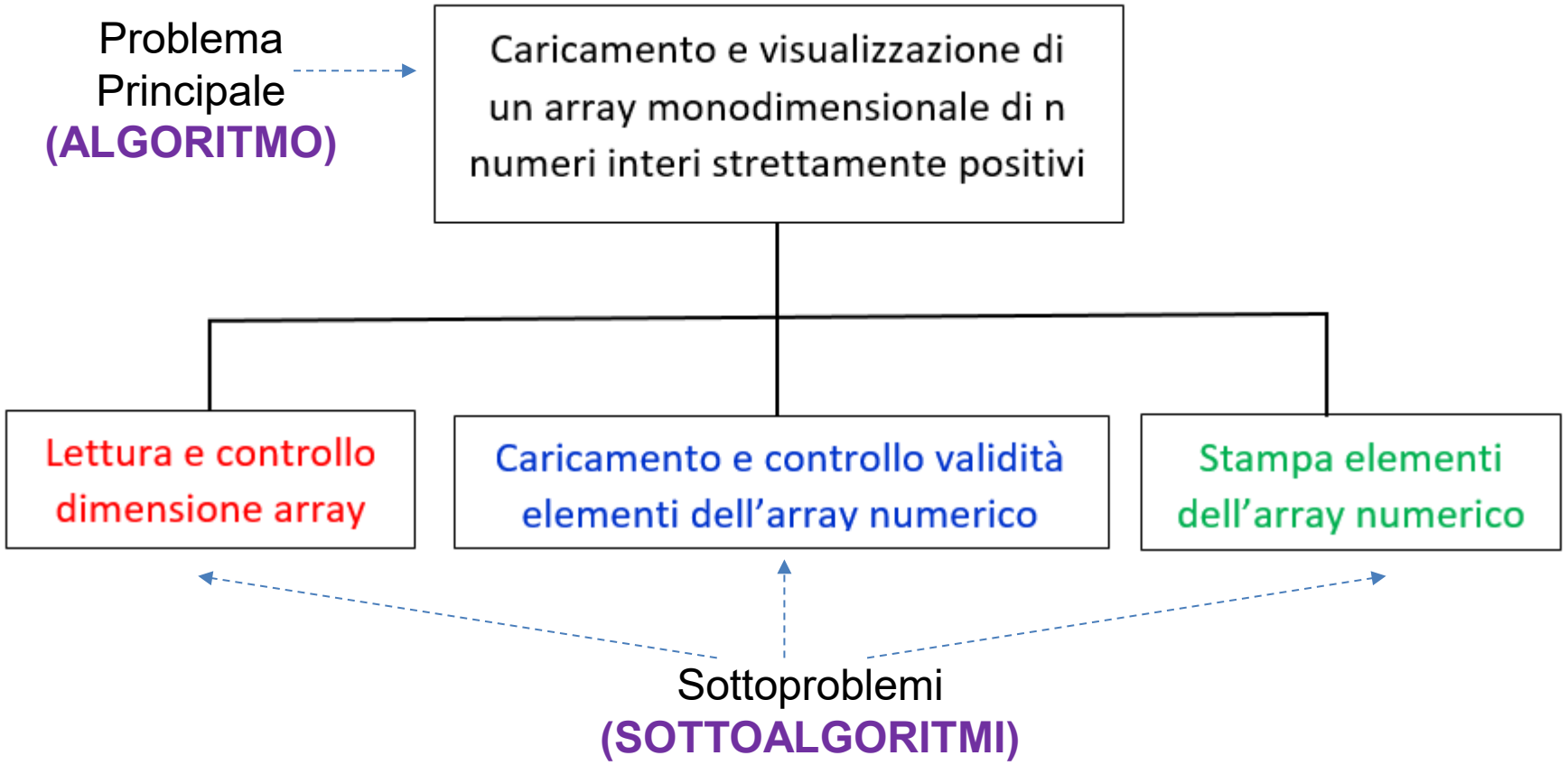
```
1  //Funzione SommaF() ----> corpo
2  int SommaF (int a, int b)
3  {
4  int s;
5
6  s= a + b;
7  return (s);
8  }
```

In questo file .C è dettagliato il **corpo** della **funzione SommaF** *(progettata come FUNZIONE)*

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

Esercizio: Caricamento e visualizzazione di un array monodimensionale di interi strettamente positivi

N.B. Utilizzando la METODOLOGIA di PROGETTAZIONE TOP-DOWN il problema assegnato potrebbe essere scomposto nei seguenti sottoproblemi:



USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

Grazie alla metodologia di progettazione top-down, abbiamo dunque individuato i seguenti SOTTOPROGRAMMI (procedura o funzione) da implementare, che si devono occupare delle seguenti azioni:

1) la lettura ed il controllo della dimensione dell'array;
2) il caricamento (con eventuale controllo del valore) dei suoi elementi;
3) la stampa dei suoi elementi.

Prototipo o segnatura se **FUNZIONE**

1 **FUNZIONE Check_Dim** () : **INT**

Prototipo o segnatura se **PROCEDURA**

2 **PROCEDURA Leggi_V** (**VAL** n **: INT**, **REF** v : **ARRAY** [MAXDIM] **DI INT**)

Prototipo o segnatura se **PROCEDURA**

3 **PROCEDURA Stampa_V** (**VAL** n **: INT**, **VAL** v : **ARRAY** [MAXDIM] **DI INT**)

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

TABELLE DEI DATI: PROCEDURA main()

DATI DI INPUT DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
MAXDIM	INT	STATICA	**10**	Massimo numero di elementi dell'array monodimensionale **v**
v	ARRAY[MAXDIM] DI INT	STATICA	**v[i] > 0**	Vettore monodimensonale di interi positivi da acquisire da tastiera
n	REAL	STATICA	**1 ≤ n ≤ MAXDIM** ossia **(n ≥ 1) AND (n ≤ MAXDIM)**	Dimensione del vettore di interi positivi da acquisire da tastiera

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

ALGORITMO Array_SottoProgrammi

MAXDIM **10**

PROCEDURA main()

/* dati di elaborazione o lavoro */
v : **ARRAY[**MAXDIM**] DI INT**
n : **INT**

INIZIO

/* CALL alla FUNZIONE Check_Dim() */
n ← Check_Dim()

/* CALL alla PROCEDURA Leggi_V() */
Leggi_V (n, v)

/* CALL alla PROCEDURA Stampa_V() */
Stampa_V (n, v)

RITORNA

FINE

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

TABELLE DEI DATI: FUNZIONE Check_Dim()

DATI DI INPUT DEL SOTTOPROBLEMA: FUNZIONE Check_Dim()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI OUTPUT DEL SOTTOPROBLEMA: FUNZIONE Check_Dim()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: FUNZIONE Check_Dim()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
n	**INT**	**STATICA**	**$1 \le n \le MAXDIM$ ossia** $(n \ge 1)$ **AND** $(n \le MAXDIM)$	Dimensione del vettore di interi il cui valore sarà restituito nel nome della funzione stessa

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

```
FUNZIONE Check_Dim () : INT
n : INT
INIZIO
/* Leggo e controllo la dimensione dell'array */
RIPETI
    Scrivi("Inserisci la dimensione dell'array: ")
    Leggi(n)
    SE (n < 1) OR (n > MAXDIM)
      ALLORA
          Scrivi("ERRORE: la dimensione dell'array non e' corretta!")
    FINE SE
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
RITORNA (n)
FINE
```

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

TABELLE DEI DATI: PROCEDURA Carica_V ()

DATI DI INPUT DEL SOTTOPROBLEMA: PROCEDURA Leggi_V()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
n	**INT**	**STATICA**	**1 ≤ n ≤ MAXDIM ossia (n ≥ 1) AND (n ≤ MAXDIM)**	Primo parametro FORMALE, PASSATO PER VALORE, che conterrà la dimensione del vettore di interi

DATI DI OUTPUT DEL SOTTOPROBLEMA: PROCEDURA Leggi_V()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY[MAXDIM] DI INT**	**STATICA**	**v[i] > 0**	Secondo parametro FORMALE, PASSATO PER RIFERIMENTO che conterrà il vettore di interi che si intende caricare

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: PROCEDURA Leggi_V()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
i	**INT**	**STATICA**	**1 ≤ i ≤ n + 1 ossia (i ≥ 1) AND (i ≤ n + 1)**	Indice necessario a navigare gli elementi del vettore **v**

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

```
PROCEDURA Leggi_V (VAL n : INT, REF v : ARRAY[MAXDIM] DI INT)
INIZIO
i : INT
/* Leggo e controllo i valori dell'array */
PER i ← 1 A n ESEGUI
    RIPETI
        Scrivi("Inserisci l'elemento del vettore: ")
        Leggi(v[i])
        SE (v[i] ≤ 0)
            ALLORA
                Scrivi("ERRORE: l'elemendo dell'array deve essere strettamente positivo!")
        FINE SE
    FINCHE' (v[i] > 0)
    i ← i + 1
FINE PER
RITORNA
FINE
```

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

TABELLE DEI DATI: PROCEDURA Stampa_V ()

DATI DI INPUT DEL SOTTOPROBLEMA: PROCEDURA Stampa_V ()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
n	**INT**	**STATICA**	**1 ≤ n ≤ MAXDIM ossia (n ≥ 1) AND (n ≤ MAXDIM)**	Primo parametro FORMALE, PASSATO PER VALORE, che conterrà la dimensione del vettore di interi
v	**ARRAY[MAXDIM] DI INT**	**STATICA**	**v[i] > 0**	Secondo parametro FORMALE, PASSATO PER VALORE che conterrà il vettore di interi che si intende stampare

DATI DI OUTPUT DEL SOTTOPROBLEMA: PROCEDURA Stampa_V ()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY[MAXDIM] DI INT**	**STATICA**	**v[i] > 0**	Secondo parametro FORMALE, PASSATO PER VALORE che conterrà il vettore di interi stampato

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: PROCEDURA Stampa_V ()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
i	**INT**	**STATICA**	**1 ≤ i ≤ n + 1 ossia (i ≥ 1) AND (i ≤ n + 1)**	Variabile che conterrà l'indice necessario a navigare gli elementi del vettore v

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY numerico

```
PROCEDURA Stampa_V (VAL n : INT, VAL v : ARRAY[MAXDIM] DI INT)
INIZIO
i : INT
/* Stampo i valori dell'array */
Scrivi("Gli elementi del vettore sono: ")
PER i ← 1 A n ESEGUI
    Scrivi(v[i])
    Scrivi(" ")
    i ← i + 1
FINE PER
RITORNA
FINE
```

ALGORITMO Array_SottoProgrammi

```
MAXDIM 10
PROCEDURA main()
/* dati di elaborazione o lavoro */
v : ARRAY[MAXDIM] DI INT
n : INT
INIZIO
/* CALL alla FUNZIONE Check_Dim () */
n ← Check_Dim();
/* CALL alla PROCEDURA Leggi_V () */
Leggi_V (n, v)
/* CALL alla PROCEDURA Stampa_V () */
Stampa_V (n, v)
RITORNA
FINE
```

```
FUNZIONE Check_Dim () : INT
n : INT
INIZIO
/* Leggo e controllo la dimensione dell'array */
RIPETI
   Scrivi("Inserisci la dimensione dell'array: ")
   Leggi(n)
   SE (n < 1) OR (n > MAXDIM)
     ALLORA
        Scrivi("ERRORE: la dimensione dell'array non e' corretta!")
   FINE SE
FINCHE' (n ≥ 1) AND (n  ≤ MAXDIM)
RITORNA (n)
FINE
```

SOLUZIONE PROPOSTA (PSEUDOCODIFICA) Parte 1

```
PROCEDURA Leggi_V (VAL n : INT, REF v : ARRAY[MAXDIM] DI INT)
INIZIO
i : INT
/* Leggo e controllo i valori dell'array */
PER i ← 1 A n ESEGUI
   RIPETI
      Scrivi("Inserisci l'elemento dell'array: ")
      Leggi(v[i])
      SE (v[i] ≤ 0)
         ALLORA
            Scrivi("ERRORE: l'elemento dell'array deve essere strettamente positivo!")
      FINE SE
   FINCHE' (v[i] > 0)
   i ← i + 1
FINE PER
RITORNA
FINE
```

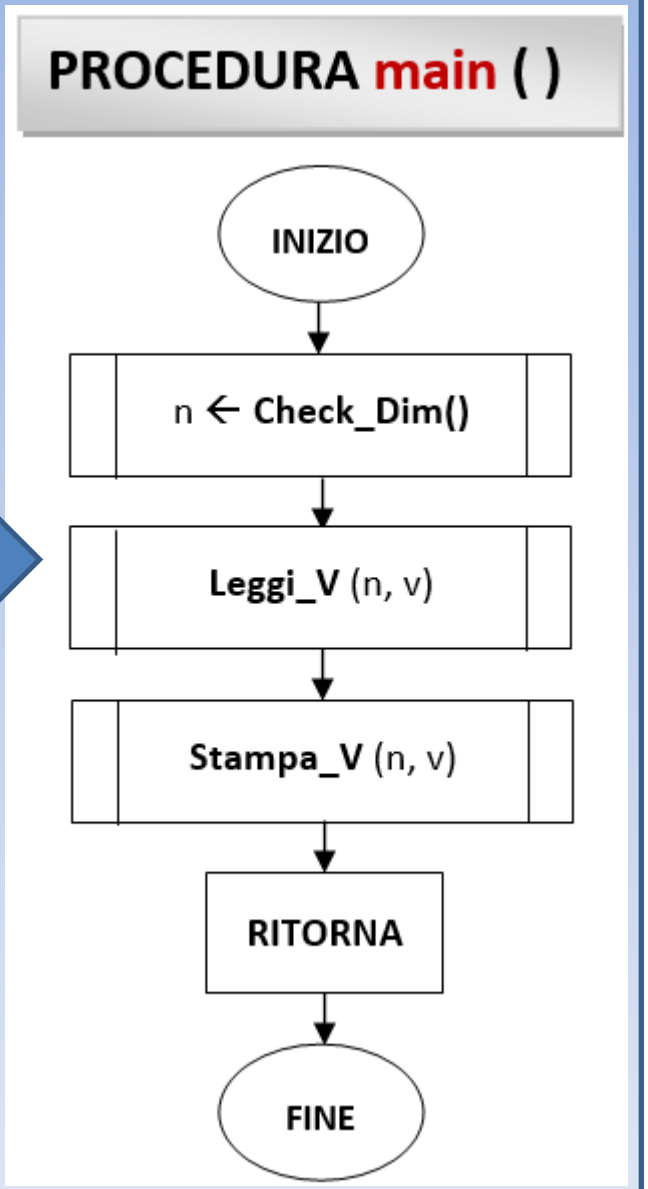
```
PROCEDURA Stampa_V (VAL n : INT, VAL v : ARRAY[MAXDIM] DI INT)
INIZIO
i : INT
/* Stampo i valori dell'array */
Scrivi("Gli elementi dell'array sono: ")
PER i ← 1 A n ESEGUI
   Scrivi(v[i])
   Scrivi(" ")
   i ← i + 1
FINE PER
RITORNA
FINE
```

SOLUZIONE PROPOSTA (PSEUDOCODIFICA) Parte 2

ALGORITMO Array_SottoProgrammi
MAXDIM 10

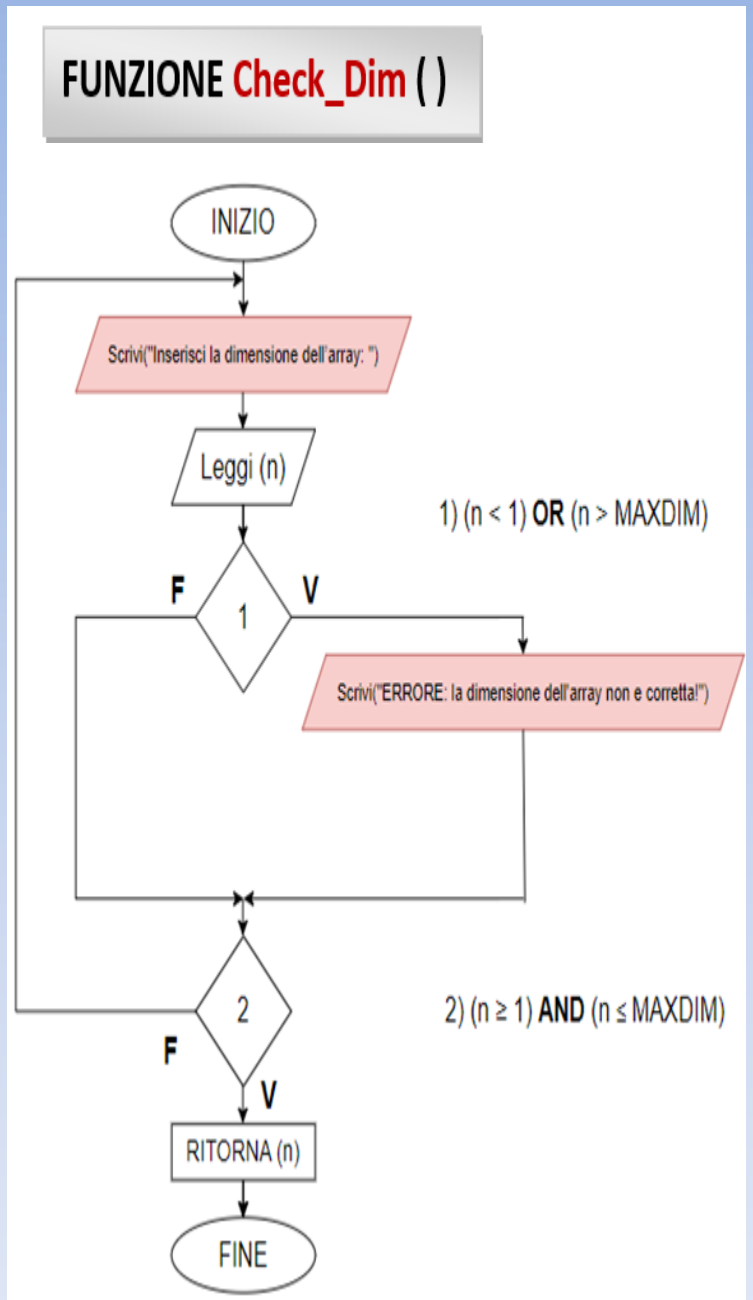
```
PROCEDURA main()
/* dati di elaborazione o lavoro */
v : ARRAY[MAXDIM] DI INT
n : INT
INIZIO
/* CALL alla FUNZIONE Check_Dim () */
n ← Check_Dim();
/* CALL alla PROCEDURA Leggi_V () */
Leggi_V (n, v)
/* CALL alla PROCEDURA Stampa_V () */
Stampa_V (n, v)
RITORNA
FINE
```

SOLUZIONE PROPOSTA
(FLOWCHART)



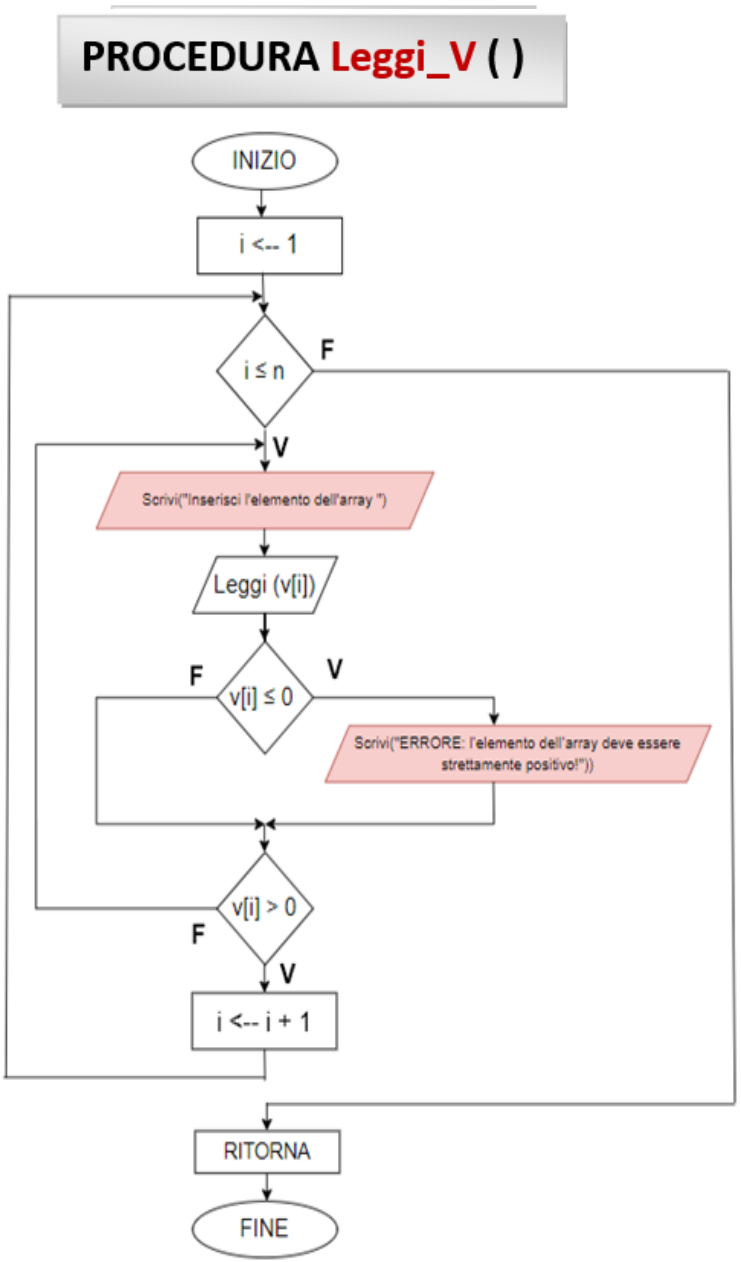
ALGORITMO Array_SottoProgrammi

```
FUNZIONE Check_Dim () : INT
n : INT
INIZIO
/* Leggo e controllo la dimensione dell'array */
RIPETI
    Scrivi("Inserisci la dimensione dell'array: ")
    Leggi(n)
    SE (n < 1) OR (n > MAXDIM)
      ALLORA
          Scrivi("ERRORE: la dimensione dell'array non e' corretta!")
     FINE SE
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
RITORNA (n)
FINE
```



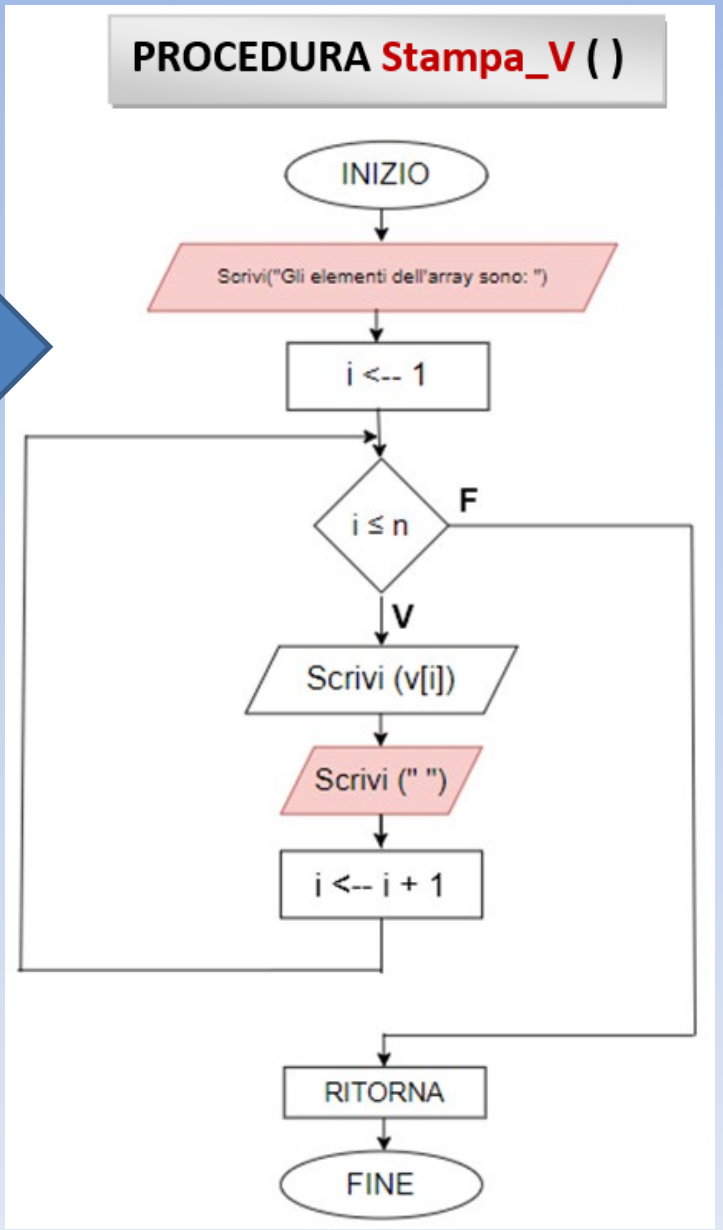
ALGORITMO Array_SottoProgrammi

```
PROCEDURA Leggi_V (VAL n : INT,
                   REF v : ARRAY[MAXDIM] DI INT)
INIZIO
i : INT
/* Leggo e controllo i valori dell'array */
PER i ← 1 A n ESEGUI
   RIPETI
      Scrivi("Inserisci l'elemento dell'array: ")
      Leggi(v[i])
      SE (v[i] ≤ 0)
         ALLORA
            Scrivi("ERRORE: l'elemento dell'array deve
            essere strettamente positivo!")
      FINE SE
   FINCHE' (v[i] > 0)
   i ← i + 1
FINE PER
RITORNA
FINE
```



ALGORITMO Array_SottoProgrammi

```
PROCEDURA Stampa_V (VAL n : INT, VAL v : ARRAY[MAXDIM] DI INT)
INIZIO
i : INT
/* Stampo i valori dell'array */
Scrivi("Gli elementi dell'array sono: ")
PER i ← 1 A n ESEGUI
    Scrivi(v[i])
    Scrivi(" ")
    i ← i + 1
FINE PER
RITORNA
FINE
```



Speciale: IMPLEMENTAZIONE IN C

a) Progetto DEV-C++ MONOFILE (tutto il codice di tutte le funzioni in un unico file .C)

Creiamo un progetto DEV-C++ in cui ci sia un unico file .C il cui listato è il seguente:

```
#include <stdio.h>
#include <stdlib.h>
#define MAXDIM 10

// definizione del prototipo della funzione Check_Dim (teoricamente è una FUNZIONE)
int Check_Dim ();
// definizione del prototipo delle funzione Leggi_V (teoricamente è una PROCEDURA)
void Leggi_V (int n , int v[MAXDIM]);
// definizione dei prototipi delle funzione Stampa_V (teoricamente è una PROCEDURA)
void Stampa_V (int n , int v[MAXDIM]);

int main(int argc, char*argv[])
{
/* dati di input/output */
int v[MAXDIM];
int n;

/* CALL alla FUNZIONE Check_Dim () */
n = Check_Dim();
/* CALL alla PROCEDURA Leggi_V () */
Leggi_V (n, v);
/* CALL alla PROCEDURA Stampa_V () */
Stampa_V (n, v);

return 0;
}
```

Qui sono definite le eventuali **costanti**

Indicazione di MAXDIM non necessaria

Qui vanno definiti i prototipi o segnature delle FUNZIONI e/o PROCEDURE definite dall'utente

Istruzione di chiamata (o CALL) alla FUNZIONE Check_Dim *(progettata come FUNZIONE)*

Istruzione di chiamata (o CALL) alla FUNZIONE Leggi_V *(progettata come PROCEDURA)*

Istruzione di chiamata (o CALL) alla FUNZIONE Stampa_V *(progettata come PROCEDURA)*

Speciale: IMPLEMENTAZIONE IN C

```
////////////////////////////////////
//     FUNZIONE Check_Dim ()      //   //(teoricamente è una FUNZIONE)
////////////////////////////////////
int Check_Dim (void)
{
//dati di input
int n;

/* Leggo e controllo la dimensione dell'array */
do
   {
   printf ("Inserisci la dimensione dell'array: ");
   scanf ("%d", &n);
   if ((n < 1) || (n > MAXDIM))
      {
      printf("ERRORE: la dimensione dell'array non e' corretta!\n");
      }
   }
while ((n < 1) || (n > MAXDIM));

return (n);
}
```

N.B. Indicazione di void NON NECESSARIA….

Speciale: IMPLEMENTAZIONE IN C

```c
////////////////////////////////////
//     FUNZIONE Leggi_V ()      //    //(teoricamente è una PROCEDURA)
////////////////////////////////////
void Leggi_V (int n, int v[MAXDIM])
{
//dati di lavoro
int i;

/* Leggo e controllo i valori dell'array */
for(i = 0; i < n; i++)
   {
   do
      {
      printf ("Inserisci l'elemento del vettore: ");
      scanf ("%d", &v[i]);
      if (v[i] <= 0)
         {
         printf ("ERRORE: l'elemento dell'array deve essere strettamente positivo!\n");
         }
      }
   while (v[i] <= 0);
   }

return;
}
```

Indicazione di MAXDIM non necessaria

Speciale: IMPLEMENTAZIONE IN C

```
////////////////////////////////////
//     FUNZIONE Stampa_V ()       //    //(teoricamente è una PROCEDURA)
////////////////////////////////////
void Stampa_V (int n, int v[MAXDIM])
{
//dati di lavoro
int i;

/* Leggo e controllo i valori dell'array */
for(i = 0; i < n; i++)
   {
   printf ("%d", v[i]);
   printf (" ");
   }

return;
}
```

Indicazione di MAXDIM non necessaria

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

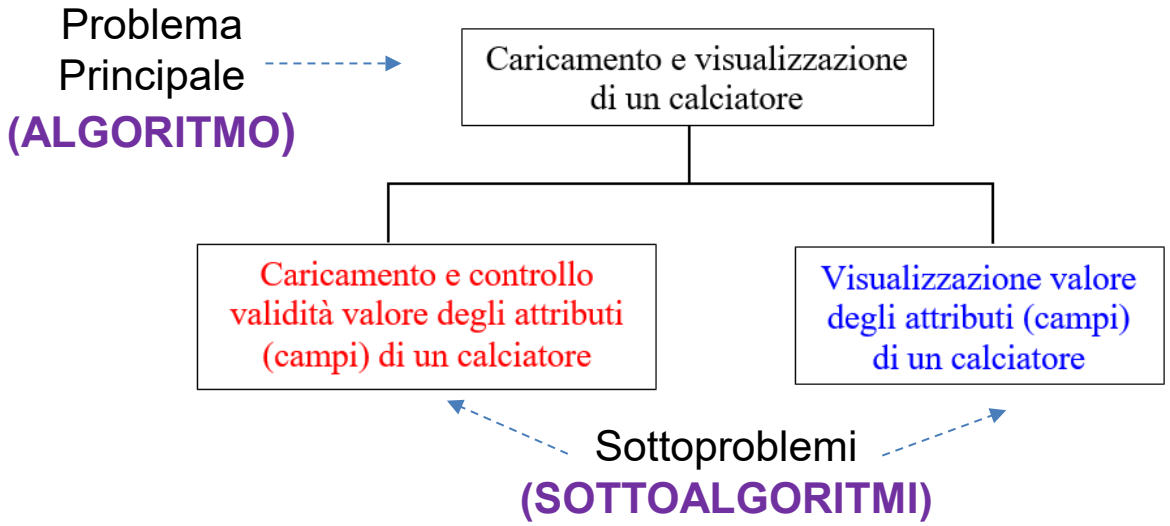
Esercizio: Caricamento e visualizzazione di un record di tipo "Calciatore" il cui tracciato è assegnato secondo la seguente rappresentazione tabellare:

Numero	Nome Campo	Tipo Campo	Lunghezza[1]	Descrizione	FlagSubRec[2]
1	Cognome	ARRAY DI CHAR	30	Cognome del calciatore	
2	Nome	ARRAY DI CHAR	30	Nome del calciatore	
3	Maglia	INT	2	Num. maglia del calciatore	

1. In caso di valori numerici decimali scrivere **5,2** vuol dire **5** cifre intere in totale di cui **2** decimali (max 999.99)
In caso di valori numerici interi scrivere **4** vuol dire prevedere un massimo di **4 cifre significative** (max 9999)

2. Se nel campo FlagSubRec viene posta una X vuol dire che quel campo deve essere considerato parte di un altro record (vedi SOTTORECORD)

N.B. Utilizzando la METODOLOGIA di PROGETTAZIONE TOP-DOWN il problema assegnato potrebbe essere scomposto nei seguenti sottoproblemi:



USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

Grazie alla metodologia di progettazione top-down, abbiamo dunque individuato i seguenti SOTTOPROGRAMMI (procedura o funzione) da implementare, che si devono occupare delle seguenti azioni:
1) il caricamento (con eventuale controllo del valore di tutti gli attributi (i campi))
2) la visualizzazione di tutti gli attributi (campi)
di un RECORD con il seguente tracciato (PSEUDOCODIFICA):

```
TIPO Calciatore = RECORD
                      Cognome : ARRAY[30] DI CHAR
                      Nome :    ARRAY[30] DI CHAR
                      Maglia :  INT
                  FINE RECORD
```

Prototipo o segnatura se **PROCEDURA**

1.a PROCEDURA Carica_Record_P (REF c : Calciatore)

Prototipo o segnatura se **FUNZIONE**
(N.B. IN ALTERNATIVA alla PROCEDURA Carica_Record_P())

1.b FUNZIONE Carica_Record_F ( ) : Calciatore

Prototipo o segnatura se **PROCEDURA**

2 PROCEDURA Visualizza_Record_P (VAL c : Calciatore)

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

TABELLE DEI DATI: PROCEDURA main()

DATI DI INPUT DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
C	**Calciatore**	**STATICA**	**Vedi TRACCIATO record**	Istanza generica del tipo Calciatore
MAXNUMCHAR	**INT**	**STATICA**	**30**	Massimo numero di caratteri per Nome e Cognme
MINNUMMAGLIA	**INT**	**STATICA**	**1**	Minimo numero maglia
MAXNUMMAGLIA	**INT**	**STATICA**	**99**	Massimo numero maglia

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

```
ALGORITMO  Record_E_Sottoprogrammi

MAXNUMCHAR      30
MINNUMMAGLIA    1
MAXNUMMAGLIA    99

TIPO Calciatore = RECORD
                      Cognome : ARRAY[MAXNUMCHAR] DI CHAR
                      Nome : ARRAY[MAXNUMCHAR] DI CHAR
                      Maglia : INT
                  FINE RECORD
PROCEDURA main()
c : Calciatore
INIZIO
//Utilizziamo la procedura 1.a
Carica_Record_P (c)
//Oppure potremmo utilizzare IN ALTERNATIVA la funzione 1.b
//c ← Carica_Record_F()
//Utilizziamo la procedura 2
Visualizza_Record_P (c)
RITORNA
FINE
```

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

TABELLE DEI DATI: PROCEDURA Carica_Record_P()

DATI DI INPUT DEL SOTTOPROBLEMA: PROCEDURA Carica_Record_P()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI OUTPUT DEL SOTTOPROBLEMA: PROCEDURA Carica_Record_P()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
c	**Calciatore**	**STATICA**	**Vedi TRACCIATO record**	Parametro FORMALE, PASSATO PER RIFERIMENTO, che conterrà il record di tipo Calciatore da valorizzare

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: PROCEDURA Carica_Record_P()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

```
PROCEDURA Carica_Record_P (REF c : Calciatore)
INIZIO
RIPETI
    Scrivi("Cognome = ")
    Leggi (c.Cognome)
FINCHE' (Lunghezza(c.Cognome) ≠ 0) AND
        (Lunghezza(c.Cognome) ≤ MAXNUMCHAR)
RIPETI
    Scrivi("Nome = ")
    Leggi (c.Nome)
FINCHE' (Lunghezza(c.Nome) ≠ 0)
        AND (Lunghezza(c.Nome) ≤ MAXNUMCHAR)
RIPETI
    Scrivi("Maglia = ")
    Leggi (c.Maglia)
FINCHE' (c.Maglia ≥ MINNUMMAGLIA) AND (c.Maglia ≤ MAXNUMMAGLIA)
RITORNA
FINE
```

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

ALTERNATIVA POSSIBILE - TABELLE DEI DATI: FUNZIONE Carica_Record_F()

DATI DI INPUT DEL SOTTOPROBLEMA: FUNZIONE Carica_Record_F()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI OUTPUT DEL SOTTOPROBLEMA: FUNZIONE Carica_Record_F()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: FUNZIONE Carica_Record_F()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
C	Calciatore	STATICA	Vedi TRACCIATO record	Calciatore generico restituito nel nome della funzione

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

```
FUNZIONE Carica_Record_F ( ) : Calciatore
c : Calciatore
INIZIO
RIPETI
    Scrivi("Cognome = ")
     Leggi (c.Cognome)
FINCHE' (Lunghezza(c.Cognome) ≠ 0) AND
            (Lunghezza(c.Cognome) ≤ MAXNUMCHAR)
RIPETI
    Scrivi("Nome=  ")
     Leggi (c.Nome)
FINCHE' (Lunghezza(c.Nome) ≠ 0)
            AND (Lunghezza(c.Nome) ≤ MAXNUMCHAR)
RIPETI
    Scrivi("Maglia = ")
     Leggi (c.Maglia)
FINCHE' (c.Maglia ≥ MINNUMMAGLIA) AND (c.Maglia ≤ MAXNUMMAGLIA)
RITORNA (c)
FINE
```

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

TABELLE DEI DATI: PROCEDURA Visualizza_Record_P()

DATI DI INPUT DEL SOTTOPROBLEMA: PROCEDURA Visualizza_Record_P()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
c	**Calciatore**	**STATICA**	**Vedi TRACCIATO record**	Parametro passato per VALORE contenente il calciatore da stampare

DATI DI OUTPUT DEL SOTTOPROBLEMA: PROCEDURA Visualizza_Record_P()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
c	**Calciatore**	**STATICA**	**Vedi TRACCIATO record**	Parametro passato per VALORE contenente il calciatore stampato

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: PROCEDURA Visualizza_Record_P()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un RECORD

```
PROCEDURA Visualizza_Record_P (VAL c : Calciatore)
INIZIO
Scrivi("Cognome immesso: ")
Scrivi (c.Cognome)
Scrivi("Nome immesso: ")
Scrivi (c.Nome)
Scrivi("Maglia immessa: ")
Scrivi (c.Maglia)
RITORNA
FINE
```

ALGORITMO Record_E_Sottoprogrammi

```
MAXNUMCHAR     30
MINNUMMAGLIA   1
MAXNUMMAGLIA   99
TIPO Calciatore = RECORD
                      Cognome : ARRAY[MAXNUMCHAR] DI CHAR
                      Nome : ARRAY[MAXNUMCHAR] DI CHAR
                      Maglia : INT
                  FINE RECORD
```

```
PROCEDURA main()
c : Calciatore
INIZIO
//Utilizziamo la procedura 1.a
Carica_Record_P (c)
//Oppure potremmo utilizzare IN ALTERNATIVA la funzione 1.b
//c ← Carica_Record_F()
//Utilizziamo la procedura 2
Visualizza_Record_P (c)
RITORNA
FINE
```

SOLUZIONE PROPOSTA
(PSEUDOCODIFICA)
Parte 1

```
PROCEDURA Carica_Record_P (REF c : Calciatore)
INIZIO
RIPETI
    Scrivi("Cognome = ")
    Leggi (c.Cognome)
FINCHE' (Lunghezza(c.Cognome) ≠ 0) AND
          (Lunghezza(c.Cognome) ≤ MAXNUMCHAR)
RIPETI
    Scrivi("Nome = ")
    Leggi (c.Nome)
FINCHE' (Lunghezza(c.Nome) ≠ 0)
          AND (Lunghezza(c.Nome) ≤ MAXNUMCHAR)
RIPETI
    Scrivi("Maglia = ")
    Leggi (c.Maglia)
FINCHE' (c.Maglia ≥ MINNUMMAGLIA) AND (c.Maglia ≤ MAXNUMMAGLIA)
RITORNA
FINE
```

SOLUZIONE PROPOSTA
(PSEUDOCODIFICA)
Parte 2

```
FUNZIONE Carica_Record_F ( ) : Calciatore
c : Calciatore                                   POSSIBILE ALTERNATIVA
INIZIO
RIPETI
   Scrivi("Cognome = ")
    Leggi (c.Cognome)
FINCHE' (Lunghezza(c.Cognome) ≠ 0) AND
          (Lunghezza(c.Cognome) ≤ MAXNUMCHAR)
RIPETI
   Scrivi("Nome=  ")
    Leggi (c.Nome)
FINCHE' (Lunghezza(c.Nome) ≠ 0)
          AND (Lunghezza(c.Nome) ≤ MAXNUMCHAR)
RIPETI
   Scrivi("Maglia = ")
    Leggi (c.Maglia)
FINCHE' (c.Maglia ≥ MINNUMMAGLIA) AND (c.Maglia ≤ MAXNUMMAGLIA
RITORNA (c)
FINE
```

SOLUZIONE PROPOSTA
(PSEUDOCODIFICA)
Parte 3

```
PROCEDURA Visualizza_Record_P (VAL c : Calciatore)
INIZIO
Scrivi("Cognome immesso: ")
Scrivi (c.Cognome)
Scrivi("Nome immesso: ")
Scrivi (c.Nome)
Scrivi("Maglia immessa: ")
Scrivi (c.Maglia)
RITORNA
FINE
```

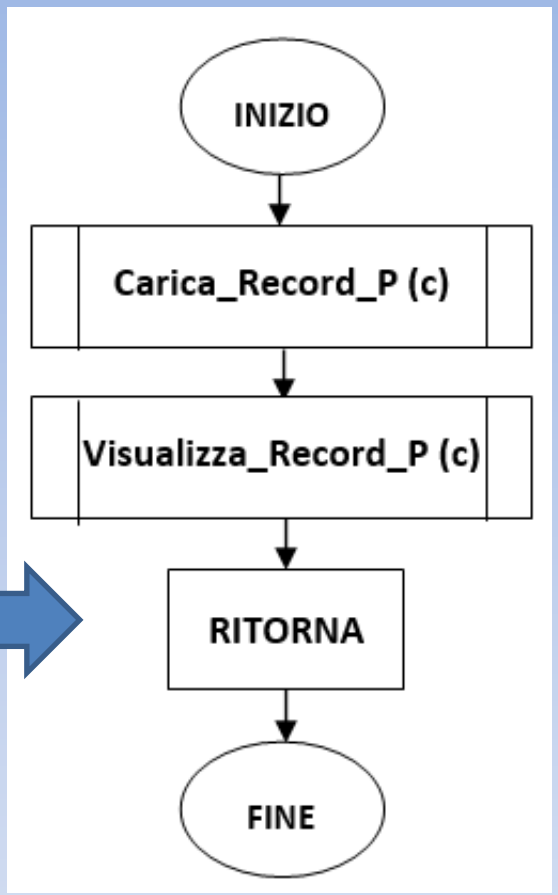
SOLUZIONE PROPOSTA
(PSEUDOCODIFICA)
Parte 4

ALGORITMO Record_E_Sottoprogrammi

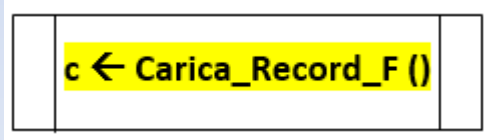
```
MAXNUMCHAR     30
MINNUMMAGLIA   1
MAXNUMMAGLIA   99

TIPO Calciatore = RECORD
                    Cognome : ARRAY[MAXNUMCHAR] DI CHAR
                    Nome : ARRAY[MAXNUMCHAR] DI CHAR
                    Maglia : INT
                  FINE RECORD
```

```
PROCEDURA main()
c : Calciatore
INIZIO
//Utilizziamo la procedura 1.a
Carica_Record_P (c)
//Oppure potremmo utilizzare IN ALTERNATIVA la funzione 1.b
//c ← Carica_Record_F()
//Utilizziamo la procedura 2
Visualizza_Record_P (c)
RITORNA
FINE
```

SOLUZIONE PROPOSTA
(FLOWCHART)

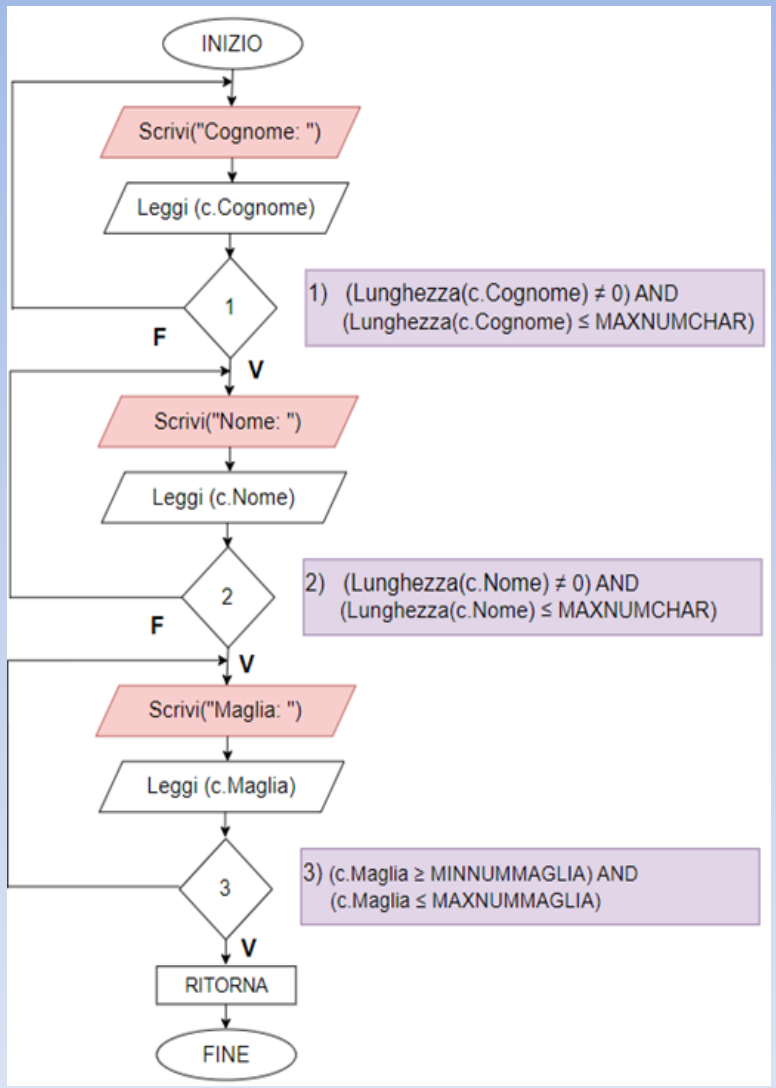


POSSIBILE ALTERNATIVA



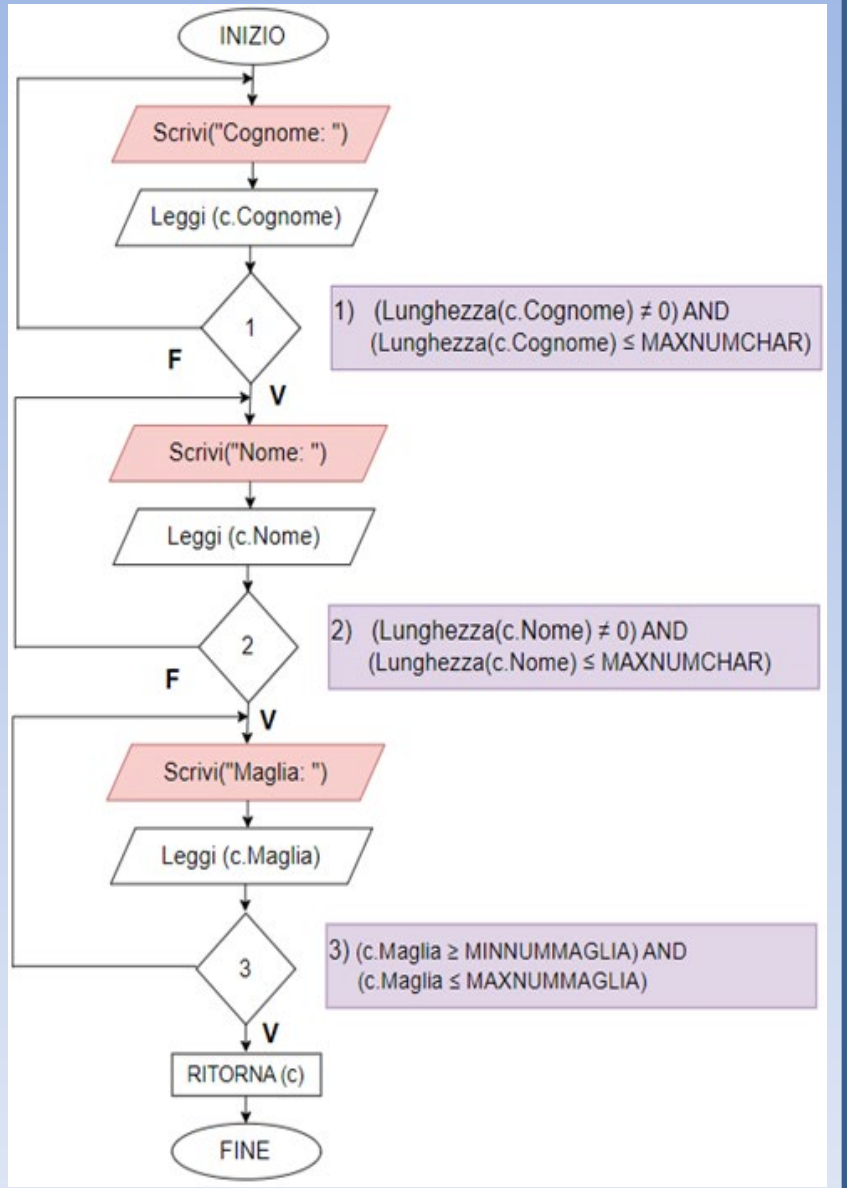
ALGORITMO Record_E_Sottoprogrammi

```
PROCEDURA Carica_Record_P (REF c : Calciatore)
INIZIO
RIPETI
    Scrivi("Cognome = ")
    Leggi (c.Cognome)
FINCHE' (Lunghezza(c.Cognome) ≠ 0) AND
          (Lunghezza(c.Cognome) ≤ MAXNUMCHAR)
RIPETI
    Scrivi("Nome = ")
    Leggi (c.Nome)
FINCHE' (Lunghezza(c.Nome) ≠ 0)
          AND (Lunghezza(c.Nome) ≤ MAXNUMCHAR)
RIPETI
    Scrivi("Maglia = ")
    Leggi (c.Maglia)
FINCHE' (c.Maglia ≥ MINNUMMAGLIA) AND (c.Maglia ≤ MAXNUMMAGLIA)
RITORNA
FINE
```



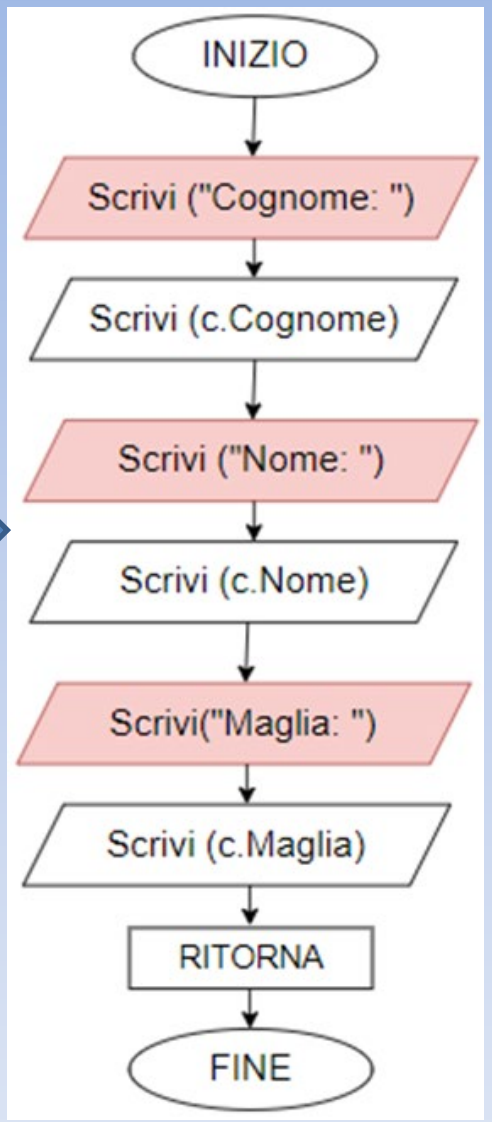
ALGORITMO Record_E_Sottoprogrammi

```
FUNZIONE Carica_Record_F ( ) : Calciatore
c : Calciatore                                   POSSIBILE ALTERNATIVA
INIZIO
RIPETI
   Scrivi("Cognome = ")
   Leggi (c.Cognome)
FINCHE' (Lunghezza(c.Cognome) ≠ 0) AND
         (Lunghezza(c.Cognome) ≤ MAXNUMCHAR)
RIPETI
   Scrivi("Nome=  ")
   Leggi (c.Nome)
FINCHE' (Lunghezza(c.Nome) ≠ 0)
         AND (Lunghezza(c.Nome) ≤ MAXNUMCHAR)
RIPETI
   Scrivi("Maglia = ")
   Leggi (c.Maglia)
FINCHE' (c.Maglia ≥ MINNUMMAGLIA) AND (c.Maglia ≤ MAXNUMMAGLIA)
RITORNA (c)
FINE
```



ALGORITMO Record_E_Sottoprogrammi

```
PROCEDURA Visualizza_Record_P (VAL c : Calciatore)
INIZIO
Scrivi("Cognome immesso: ")
Scrivi (c.Cognome)
Scrivi("Nome immesso: ")
Scrivi (c.Nome)
Scrivi("Maglia immessa: ")
Scrivi (c.Maglia)
RITORNA
FINE
```



Speciale: IMPLEMENTAZIONE IN C

a) Progetto DEV-C++ MONOFILE (tutto il codice di tutte le funzioni in un unico file .C)

Creiamo un progetto DEV-C++ in cui ci sia un unico file .C il cui listato è il seguente:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXNUMCHAR 30
#define MINNUMMAGLIA 1
#define MAXNUMMAGLIA 99

typedef struct
   {
   char Cognome [MAXNUMCHAR + 1];
   char Nome [MAXNUMCHAR + 1];
   int Maglia;
   } Calciatore;

// prototipi funzioni C necessarie
void Carica_Record_P (Calciatore *c);
Calciatore Carica_Record_F ();
void Visualizza_Record_P (Calciatore c);

// PROCEDURA main()
int main(int argc, char*argv[])
{
/* dati di input/output */
Calciatore c;

/* CALL alla PROCEDURA Carica_Record_P() */
Carica_Record_P (&c);

/* CALL alla FUNZIONE Carica_Record_F() */
//c = Carica_Record_F();

/* CALL alla PROCEDURA Visualizza_Record_P() */
Visualizza_Record_P (c);

return 0;
}
```

Qui sono definite le eventuali **costanti**

Qui va definito il **tipo di dato** definito dall'utente tramite istruzione **typedef (RECORD)**

Qui vanno definiti i **prototipi** o **segnature** delle **FUNZIONI** definite dall'utente

Istruzione di chiamata (o CALL) alla FUNZIONE Carica_Record_P *(progettata come PROCEDURA)*

Istruzione di chiamata (o CALL) alla FUNZIONE Carica_Record_F *(progettata come FUNZIONE)*

Istruzione di chiamata (o CALL) alla FUNZIONE Visualizza_Record_P *(progettata come PROCEDURA)*

Speciale: IMPLEMENTAZIONE IN C

CORPO della FUNZIONE Carica_Record_P *(progettata come PROCEDURA)*

```
// PROCEDURA Carica_Record_P()
void Carica_Record_P (Calciatore *c)
{
// caricamento e controllo campi del record
do
   {
   printf ("Cognome: ");
   gets ((*c).Cognome);
   }
while ( (strlen((*c).Cognome) == 0) || (strlen((*c).Cognome) > MAXNUMCHAR) );

do
   {
   printf ("Nome: ");
   gets ((*c).Nome);
   }
while ( (strlen((*c).Nome) == 0) || (strlen((*c).Nome) > MAXNUMCHAR) );

do
   {
   printf ("Maglia: ");
   scanf("%d", &(c->Maglia));    //alternativa scanf("%d", &((*c).Maglia));
   }
while ( ((*c).Maglia < MINNUMMAGLIA) || ((*c).Maglia > MAXNUMMAGLIA) );

return;
}
```

Speciale: IMPLEMENTAZIONE IN C

CORPO della FUNZIONE Carica_Record_F *(progettata come FUNZIONE)*

```
// FUNZIONE Carica_Record_F()
Calciatore Carica_Record_F ()
{
Calciatore c;

// caricamento e controllo campi del record
do
   {
   printf ("Cognome: ");
   gets (c.Cognome);
   }
while ( (strlen(c.Cognome) == 0) || (strlen(c.Cognome) > MAXNUMCHAR) );

do
   {
   printf ("Nome: ");
   gets (c.Nome);
   }
while ( (strlen(c.Nome) == 0) || (strlen(c.Nome) > MAXNUMCHAR) );

do
   {
   printf ("Maglia: ");
   scanf ("%d", &(c.Maglia));
   }
while ( (c.Maglia < MINNUMMAGLIA) || (c.Maglia > MAXNUMMAGLIA) );

return (c);
}
```

Speciale: IMPLEMENTAZIONE IN C

CORPO della FUNZIONE Visualizza_Record_P *(progettata come PROCEDURA)*

```
// PROCEDURA Visualizza_Record_P()
void Visualizza_Record_P (Calciatore c)
{
// Visualizzazione campi del record
// Cognome
printf ("\nCognome immesso: ");
puts (c.Cognome);

// Nome
printf ("\nNome immesso: ");
puts (c.Nome);

// Maglia
printf ("\nMaglia immessa: ");
printf ("%d",c.Maglia);

return;
}
```

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

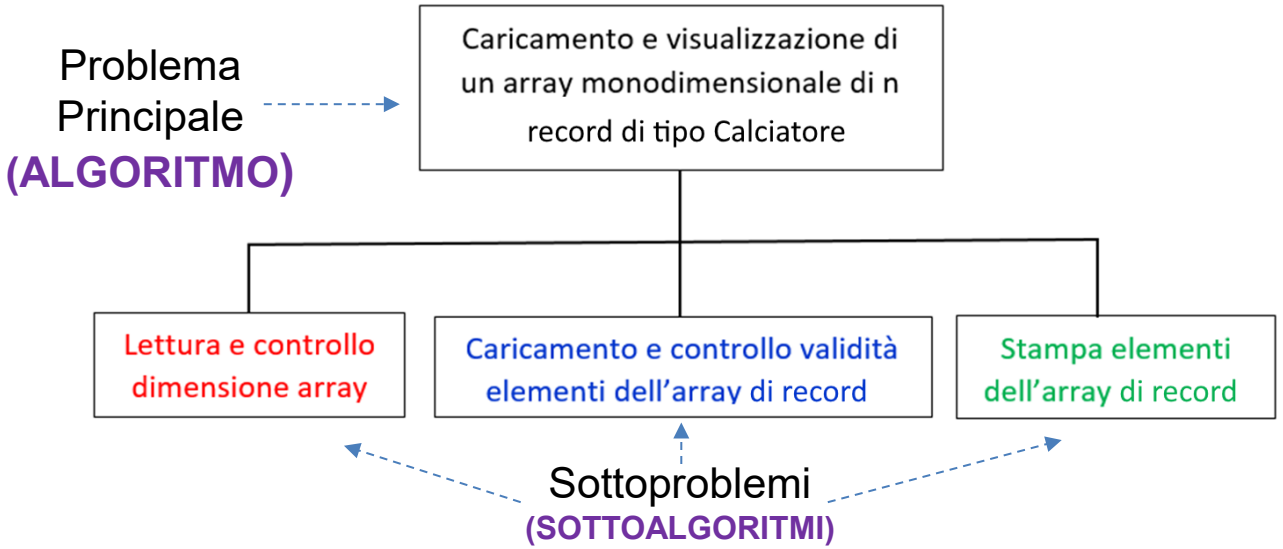
Esercizio: Caricamento e visualizzazione di un ARRAY MONODIMENSIONALE di record di tipo "Calciatore" il cui tracciato è assegnato secondo la seguente rappresentazione tabellare:

Numero	Nome Campo	Tipo Campo	Lunghezza[1]	Descrizione	FlagSubRec[2]
1	Cognome	ARRAY DI CHAR	30	Cognome del calciatore	
2	Nome	ARRAY DI CHAR	30	Nome del calciatore	
3	Maglia	INT	2	Num. maglia del calciatore	

1. In caso di valori numerici decimali scrivere **5,2** vuol dire **5** cifre intere in totale di cui **2** decimali (max 999.99)
In caso di valori numerici interi scrivere **4** vuol dire prevedere un massimo di **4 cifre significative** (max 9999)

2. Se nel campo FlagSubRec viene posta una X vuol dire che quel campo deve essere considerato parte di un altro record (vedi SOTTORECORD)

N.B. Utilizzando la METODOLOGIA di PROGETTAZIONE TOP-DOWN il problema assegnato potrebbe essere scomposto nei seguenti sottoproblemi:



USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

Grazie alla metodologia di progettazione top-down, abbiamo dunque individuato i seguenti SOTTOPROGRAMMI (procedura o funzione) da implementare, che si devono occupare delle seguenti azioni:

1) la lettura ed il controllo della dimensione dell'array;
2) il caricamento (con eventuale controllo del valore) dei suoi elementi;
3) la stampa dei suoi elementi.

Prototipo o segnatura se **FUNZIONE**

1 **FUNZIONE Leggi_Dim** () : **INT**

Prototipo o segnatura se **PROCEDURA**

2 **PROCEDURA Carica_V** (**VAL** n **: INT**, **REF** v : **ARRAY** [MAXDIM] **DI Calciatore**)

Prototipo o segnatura se **PROCEDURA**

3 **PROCEDURA Visualizza_V** (**VAL** n **: INT**, **VAL** v : **ARRAY** [MAXDIM] **DI Calciatore**)

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

TABELLE DEI DATI: PROCEDURA main()

DATI DI INPUT DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE PROCEDURA main()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
MAXNUMCHAR	INT	STATICA	30	Massimo numero di caratteri per Nome e Cognme
MINNUMMAGLIA	INT	STATICA	1	Minimo numero maglia
MAXNUMMAGLIA	INT	STATICA	99	Massimo numero maglia
MAXDIM	INT	STATICA	10	Massimo numero di elementi del vettore monodimensionale v
v	ARRAY[MAXDIM] DI Calciatore	STATICA	Vedi tracciato record	Vettore monodimensionale di tipo Calciatore
n	INT	STATICA	1 ≤ n ≤ MAXDIM ossia (n ≥ 1) AND (n≤ MAXDIM)	Dimensione effettiva del vettore di tipo Calciatore

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

ALGORITMO **Array_Record_E_Sottoprogrammi**

MAXNUMCHAR 30
MINNUMMAGLIA 1
MAXNUMMAGLIA 99
MAXDIM 10

TIPO Calciatore = **RECORD**
Cognome : **ARRAY**[MAXNUMCHAR] **DI CHAR**
Nome : **ARRAY**[MAXNUMCHAR] **DI CHAR**
Maglia : **INT**
FINE RECORD

PROCEDURA main()
v : ARRAY[MAXDIM] di Calciatore
INIZIO
/* CALL alla FUNZIONE Leggi_Dim() */
n ← Leggi_Dim()
/* CALL alla PROCEDURA Carica_V() */
Carica_V (n, v)
/* CALL alla PROCEDURA Visualizza_V() */
Visualizza_V (n, v)
RITORNA
FINE

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

TABELLE DEI DATI: FUNZIONE Leggi_Dim()

DATI DI INPUT DEL SOTTOPROBLEMA: FUNZIONE Leggi_Dim()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI OUTPUT DEL SOTTOPROBLEMA: FUNZIONE Leggi_Dim()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: FUNZIONE Leggi_Dim()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
n	**INT**	**STATICA**	**$1 \le n \le MAXDIM$** **ossia** **$(n \ge 1)$ AND $(n \le MAXDIM)$**	Dimensione del vettore di record da restituire nel nome della funzione stessa

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

```
FUNZIONE Leggi_Dim () : INT
n : INT
INIZIO
/* Leggo e controllo la dimensione dell'array */
RIPETI
    Scrivi("Inserisci la dimensione dell'array: ")
    Leggi(n)
    SE (n < 1) OR (n > MAXDIM)
      ALLORA
          Scrivi("ERRORE: la dimensione dell'array non e' corretta!")
    FINE SE
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
RITORNA (n)
FINE
```

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

TABELLE DEI DATI: PROCEDURA Carica_V ()

DATI DI INPUT DEL SOTTOPROBLEMA: PROCEDURA Carica_V()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
n	**INT**	**STATICA**	**$1 \le n \le MAXDIM$ ossia $(n \ge 1)$ AND $(n \le MAXDIM)$**	Dimensione del vettore di record passato per VALORE

DATI DI OUTPUT DEL SOTTOPROBLEMA: PROCEDURA Carica_V()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY[MAXDIM] DI Calciatore**	**STATICA**	**Vedi tracciato record**	Vettore monodimensionale di tipo Calciatore passato per RIFERIMENTO

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: PROCEDURA Carica_V()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
i	**INT**	**STATICA**	**$1 \le i \le n + 1$ ossia $(i \ge 1)$ AND $(i \le n + 1)$**	Variabile contatore per array di record

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

```
PROCEDURA Carica_V (VAL n : INT, REF v : ARRAY[MAXDIM] DI Calciatore)

i : INT
INIZIO
PER i ← 1 A n ESEGUI
    RIPETI
      Scrivi("Cognome = ")
      Leggi (v[i].Cognome)
    FINCHE' (Lunghezza(v[i].Cognome) ≠ 0) AND (Lunghezza(v[i].Cognome) ≤ MAXNUMCHAR)
    RIPETI
      Scrivi("Nome = ")
      Leggi (v[i].Nome)
    FINCHE' (Lunghezza(v[i].Nome) ≠ 0) AND (Lunghezza(v[i].Nome) ≤ MAXNUMCHAR)
    RIPETI
      Scrivi("Maglia = ")
      Leggi (v[i].Maglia)
    FINCHE' (v[i].Maglia ≥ MINNUMMAGLIA) AND (v[i].Maglia ≤ MAXNUMMAGLIA)
  i ← i + 1
FINE PER
RITORNA
FINE
```

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

TABELLE DEI DATI: PROCEDURA Visualizza_V ()

DATI DI INPUT DEL SOTTOPROBLEMA: PROCEDURA Visualizza_V()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
n	**INT**	**STATICA**	**$1 \le n \le MAXDIM$ ossia $(n \ge 1)$ AND $(n \le MAXDIM)$**	Dimensione del vettore di record passato per VALORE
v	**ARRAY[MAXDIM] DI Calciatore**	**STATICA**	**Vedi tracciato record**	Vettore monodimensionale di tipo Calciatore passato per VALORE da stampare

DATI DI OUTPUT DEL SOTTOPROBLEMA: PROCEDURA Visualizza_V()				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY[MAXDIM] DI Calciatore**	**STATICA**	**Vedi tracciato record**	Vettore monodimensionale di tipo Calciatore passato per VALORE stampato

DATI DI ELABORAZIONE o DI LAVORO DEL SOTTOPROBLEMA: PROCEDURA Visualizza_V()				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
i	**INT**	**STATICA**	**$1 \le i \le n + 1$ ossia $(i \ge 1)$ AND $(i \le n + 1)$**	Variabile contatore per array di record

USO DEI SOTTOPROGRAMMI
Caricamento e visualizzazione di un ARRAY DI RECORD

```
PROCEDURA Visualizza_V (VAL n : INT, VAL v : ARRAY[MAXDIM] DI Calciatore)
i : INT
INIZIO
PER i ← 1 A n ESEGUI
    Scrivi("Cognome = ")
    Scrivi (v[i].Cognome)
    Scrivi("Nome = ")
    Scrivi (v[i].Nome)
    Scrivi("Maglia = ")
    Scrivi (v[i].Maglia)
     i ← i + 1
FINE PER
RITORNA
FINE
```

ALGORITMO Array_Record_E_Sottoprogrammi

```
MAXNUMCHAR     30
MINNUMMAGLIA   1
MAXNUMMAGLIA   99
MAXDIM         10
TIPO Calciatore = RECORD
                      Cognome : ARRAY[MAXNUMCHAR] DI CHAR
                      Nome : ARRAY[MAXNUMCHAR] DI CHAR
                      Maglia : INT
                  FINE RECORD
```

```
PROCEDURA main()
v : ARRAY[MAXDIM] di Calciatore
INIZIO
/* CALL alla FUNZIONE Leggi_Dim() */
n ← Leggi_Dim()
/* CALL alla PROCEDURA Carica_V() */
Carica_V (n, v)
/* CALL alla PROCEDURA Visualizza_V() */
Visualizza_V (n, v)
RITORNA
FINE
```

SOLUZIONE PROPOSTA
(PSEUDOCODIFICA)
Parte 1

```
FUNZIONE Leggi_Dim () : INT
n : INT
INIZIO
/* Leggo e controllo la dimensione dell'array */
RIPETI
   Scrivi("Inserisci la dimensione dell'array: ")
   Leggi(n)
   SE (n < 1) OR (n > MAXDIM)
     ALLORA
        Scrivi("ERRORE: la dimensione dell'array non e' corretta!")
   FINE SE
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
RITORNA (n)
FINE
```

```
PROCEDURA Carica_V (VAL n : INT, REF v : ARRAY[MAXDIM] DI Calciatore)
i : INT
INIZIO
PER i ← 1 A n ESEGUI
   RIPETI
     Scrivi("Cognome = ")
     Leggi (v[i].Cognome)
   FINCHE' (Lunghezza(v[i]. Cognome) ≠ 0) AND
            (Lunghezza(v[i].Cognome) ≤ MAXNUMCHAR)
   RIPETI
     Scrivi("Nome = ")
     Leggi (v[i].Nome)
   FINCHE' (Lunghezza(v[i] Nome) ≠ 0) AND
            (Lunghezza(v[i].Nome) ≤ MAXNUMCHAR)
   RIPETI
     Scrivi("Maglia = ")
     Leggi (v[i].Maglia)
   FINCHE' (v[i].Maglia ≥ MINNUMMAGLIA) AND (v[i].Maglia ≤ MAXNUMMAGLIA)
  i ← i + 1
FINE PER
RITORNA
FINE
```

SOLUZIONE PROPOSTA
(PSEUDOCODIFICA)
Parte 2

```
PROCEDURA Visualizza_V (VAL v : ARRAY[MAXDIM] DI Calciatore)
i : INT
INIZIO
PER i ← 1 A n ESEGUI
    Scrivi("Cognome = ")
    Scrivi (v[i].Cognome)
    Scrivi("Nome = ")
    Scrivi (v[i].Nome)
    Scrivi("Maglia = ")
    Scrivi (v[i].Maglia)
    i ← i + 1
FINE PER
RITORNA
FINE
```

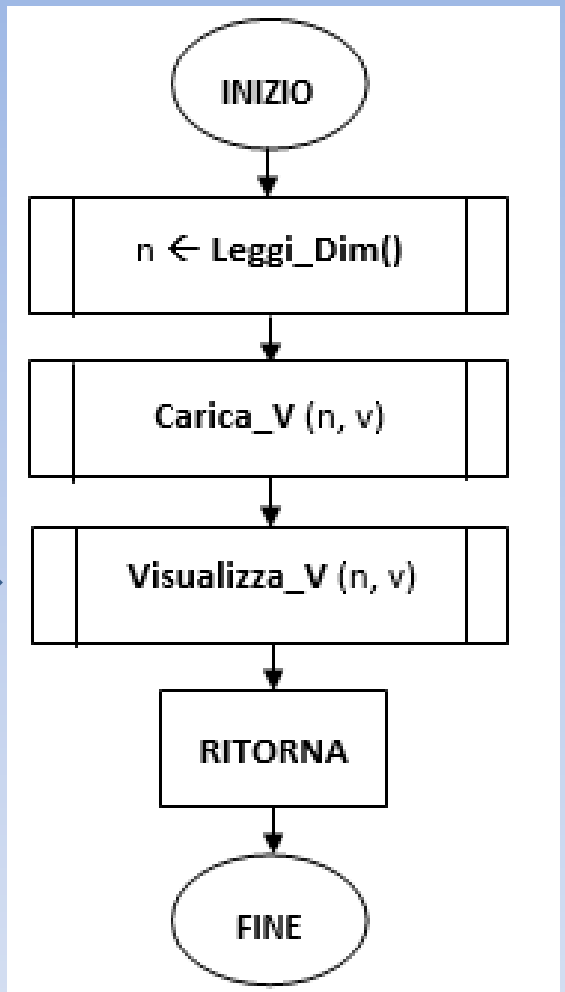
SOLUZIONE PROPOSTA
(PSEUDOCODIFICA)
Parte 3

ALGORITMO Array_Record_E_Sottoprogrammi

```
MAXNUMCHAR     30
MINNUMMAGLIA   1
MAXNUMMAGLIA   99
MAXDIM         10
TIPO Calciatore = RECORD
                    Cognome : ARRAY[MAXNUMCHAR] DI CHAR
                    Nome : ARRAY[MAXNUMCHAR] DI CHAR
                    Maglia : INT
                  FINE RECORD
```

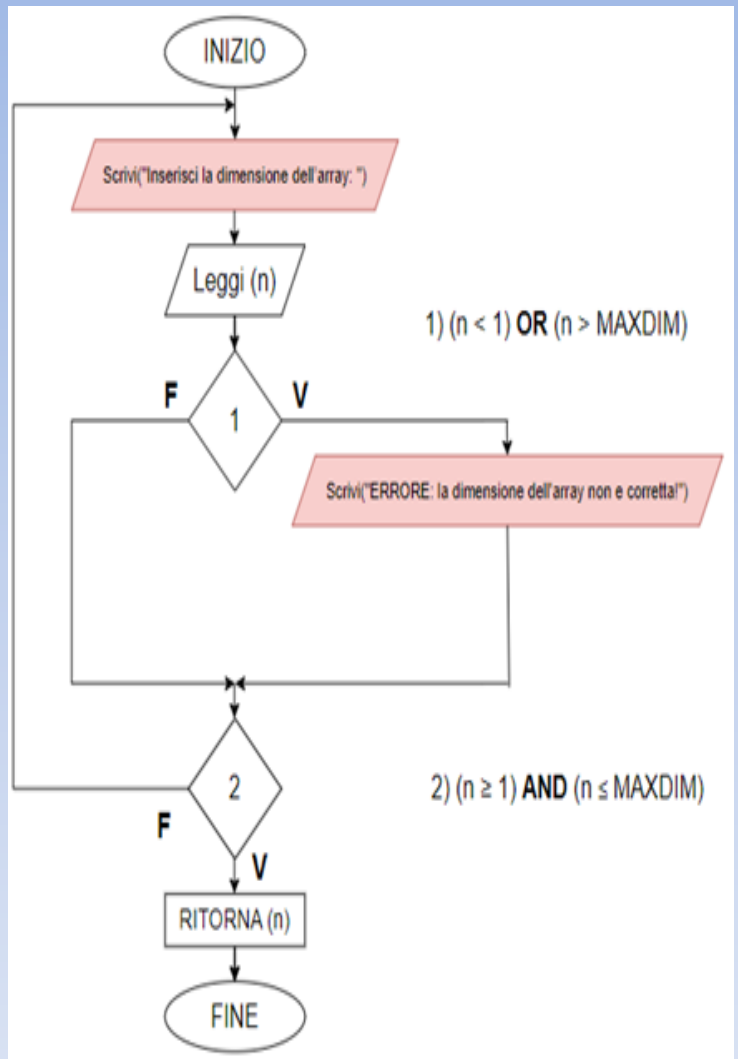
```
PROCEDURA main()
v : ARRAY[MAXDIM] di Calciatore
INIZIO
/* CALL alla FUNZIONE Leggi_Dim() */
n ← Leggi_Dim()
/* CALL alla PROCEDURA Carica_V() */
Carica_V (n, v)
/* CALL alla PROCEDURA Visualizza_V() */
Visualizza_V (n, v)
RITORNA
FINE
```

SOLUZIONE PROPOSTA
(FLOWCHART)



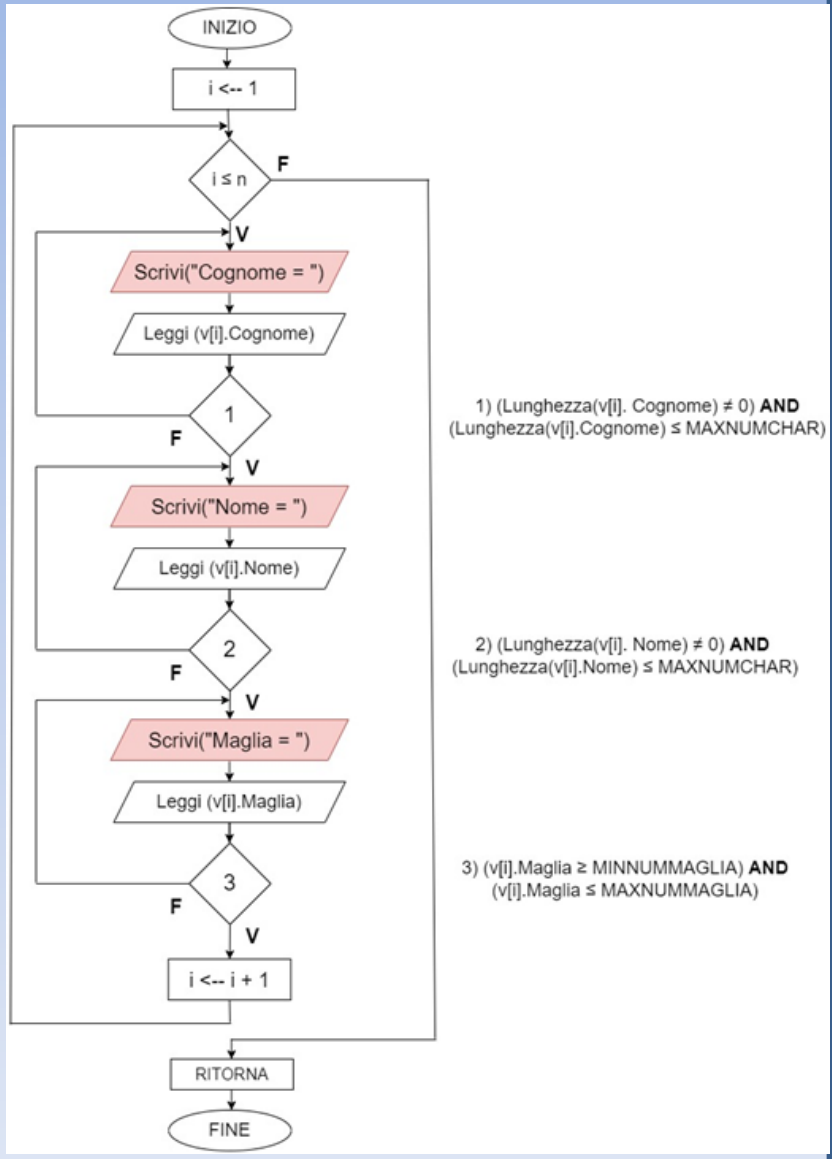
ALGORITMO Array_Record_E_Sottoprogrammi

```
FUNZIONE Leggi_Dim () : INT
n : INT
INIZIO
/* Leggo e controllo la dimensione dell'array */
RIPETI
   Scrivi("Inserisci la dimensione dell'array: ")
   Leggi(n)
   SE (n < 1) OR (n > MAXDIM)
     ALLORA
        Scrivi("ERRORE: la dimensione dell'array non e' corretta!")
   FINE SE
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
RITORNA (n)
FINE
```



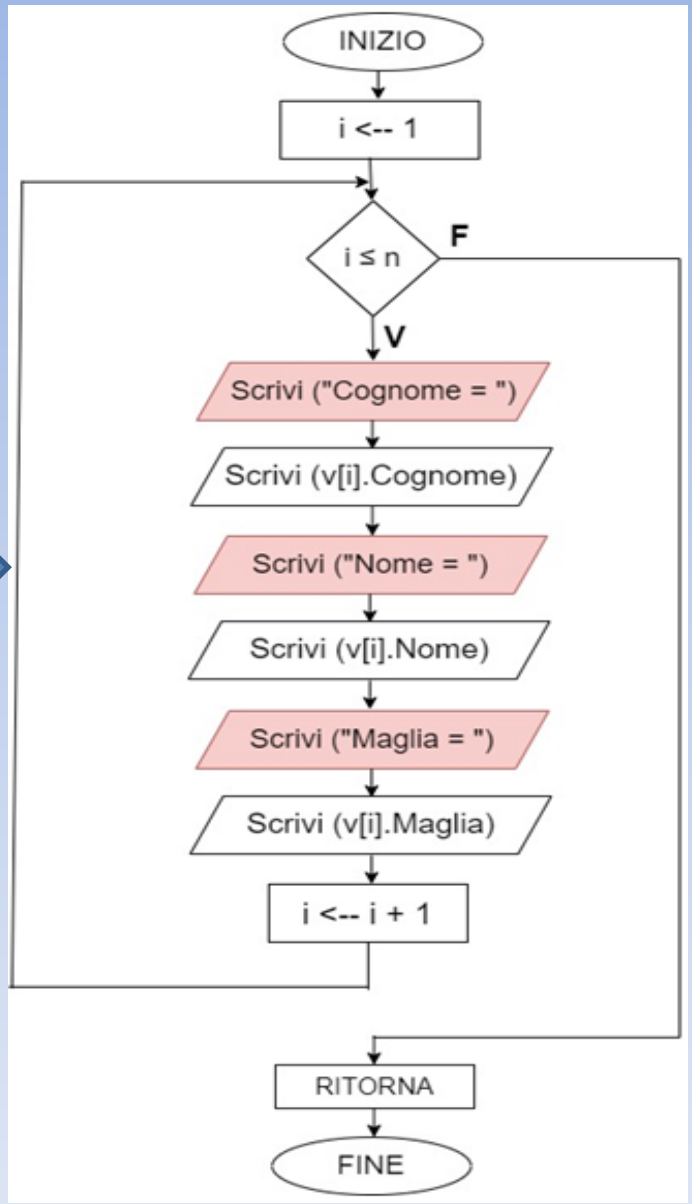
ALGORITMO Array_Record_E_Sottoprogrammi

```
PROCEDURA Carica_V (VAL n : INT, REF v : ARRAY[MAXDIM] DI Calciatore)
i : INT
INIZIO
PER i ← 1 A n ESEGUI
   RIPETI
     Scrivi("Cognome = ")
     Leggi (v[i].Cognome)
   FINCHE' (Lunghezza(v[i]. Cognome) ≠ 0) AND
            (Lunghezza(v[i].Cognome) ≤ MAXNUMCHAR)
   RIPETI
     Scrivi("Nome = ")
     Leggi (v[i].Nome)
   FINCHE' (Lunghezza(v[i] Nome) ≠ 0) AND
            (Lunghezza(v[i].Nome) ≤ MAXNUMCHAR)
   RIPETI
     Scrivi("Maglia = ")
     Leggi (v[i].Maglia)
   FINCHE' (v[i].Maglia ≥ MINNUMMAGLIA) AND (v[i].Maglia ≤ MAXNUMMAGLIA)
  i ← i + 1
FINE PER
RITORNA
FINE
```



ALGORITMO Array_Record_E_Sottoprogrammi

```
PROCEDURA Visualizza_V (VAL v : ARRAY[MAXDIM] DI Calciatore)
i : INT
INIZIO
PER i ← 1 A n ESEGUI
   Scrivi("Cognome = ")
   Scrivi (v[i].Cognome)
   Scrivi("Nome = ")
   Scrivi (v[i].Nome)
   Scrivi("Maglia = ")
   Scrivi (v[i].Maglia)
    i ← i + 1
FINE PER
RITORNA
FINE
```



Speciale: IMPLEMENTAZIONE IN C

a) Progetto DEV-C++ MONOFILE (tutto il codice di tutte le funzioni in un unico file .C)

Creiamo un progetto DEV-C++ in cui ci sia un unico file .C il cui listato è il seguente:

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

#define MAXNUMCHAR 30
#define MINNUMMAGLIA 1
#define MAXNUMMAGLIA 99

#define MAXDIM 10

typedef struct
   {
   char Cognome [MAXNUMCHAR + 1];
   char Nome [MAXNUMCHAR + 1];
   int Maglia;
   } Calciatore;

// prototipi funzioni C necessarie
int Leggi_Dim ();
void Carica_V (int n, Calciatore v[]);
void Visualizza_V (int n, Calciatore v[]);

// PROCEDURA main()
int main(int argc, char*argv[])
{
/* dati di lavoro */
int n;
Calciatore v[MAXDIM];

/* CALL alla FUNZIONE Leggi_Dim() */
n = Leggi_Dim();
/* CALL alla PROCEDURA Carica_V() */
Carica_V (n, v);
/* CALL alla PROCEDURA Visualizza_V() */
Visualizza_V (n, v);

return 0;
}
```

Qui sono definite le eventuali **costanti**

Qui va definito il **tipo di dato** definito dall'utente tramite istruzione **typedef (RECORD)**

Qui vanno definiti i **prototipi** o **segnature** delle **FUNZIONI** definite dall'utente

Istruzione di chiamata (o CALL) alla FUNZIONE Leggi_Dim *(progettata come FUNZIONE)*

Istruzione di chiamata (o CALL) alla FUNZIONE Carica_V *(progettata come PROCEDURA)*

Istruzione di chiamata (o CALL) alla FUNZIONE Visualizza_V *(progettata come PROCEDURA)*

Speciale: IMPLEMENTAZIONE IN C

CORPO della FUNZIONE Leggi_Dim *(progettata come FUNZIONE)*

```
// FUNZIONE Leggi_Dim()
int Leggi_Dim (void)
{
int n;

/* Leggo e controllo la dimensione dell'array */
do
   {
   printf ("Inserisci la dimensione dell'array: ");
   scanf ("%d", &n);

   if ((n < 1) || (n > MAXDIM))
      {
      printf ("ERRORE: la dimensione dell'array non e' corretta\n");
      }
   }
while ((n < 1) || (n > MAXDIM));

return (n);
}
```

Speciale: IMPLEMENTAZIONE IN C

CORPO della FUNZIONE Carica_V *(progettata come PROCEDURA)*

```
// PROCEDURA Carica_V()
void Carica_V (int n, Calciatore v[MAXDIM])
{
int i;

for (i=0 ; i < n; i++)
    {
    // caricamento e controllo campi del record
    do
        {
        printf ("Cognome = ");
        fflush (stdin);
        gets (v[i].Cognome);
        }
    while ( (strlen(v[i].Cognome) == 0) || (strlen(v[i].Cognome) > MAXNUMCHAR) );

    do
        {
        printf ("Nome = ");
        fflush (stdin);
        gets (v[i].Nome);
        }
    while ( (strlen(v[i].Nome) == 0) || (strlen(v[i].Nome) > MAXNUMCHAR) );

    do
        {
        printf ("Maglia = ");
        scanf("%d", &(v[i].Maglia));
        }
    while ( (v[i].Maglia < MINNUMMAGLIA) || (v[i].Maglia > MAXNUMMAGLIA) );
    }

return;
}
```

Speciale: IMPLEMENTAZIONE IN C

CORPO della FUNZIONE Visualizza_V *(progettata come PROCEDURA)*

```
// PROCEDURA Visualizza_V()
void Visualizza_V (int n, Calciatore v[MAXDIM])
{
int i;

for (i=0 ; i < n; i++)
   {
   // Visualizzazione campi del record
   // Cognome
   printf ("\nCognome = ");
   puts (v[i].Cognome);

   // Nome
   printf ("Nome = ");
   puts (v[i].Nome);

   // Maglia
   printf ("Maglia = ");
   printf ("%d",v[i].Maglia);

   printf ("\n");
   }

return;
}
```

Ricorsione e sottoprogrammi ricorsivi

Abbiamo visto che nel corpo di un **sottoprogramma** è possibile chiamare (attivare) altri sottoprogrammi dichiarati esternamente o localmente ad esso, **ma è anche possibile, in determinate condizioni, fornire l'istruzione per chiamare (riattivare) se stesso.**

In questo caso si parla di **sottoprogrammi ricorsivi**, una caratteristica prevista da alcuni linguaggi di programmazione quali il **C** o il **C++.**

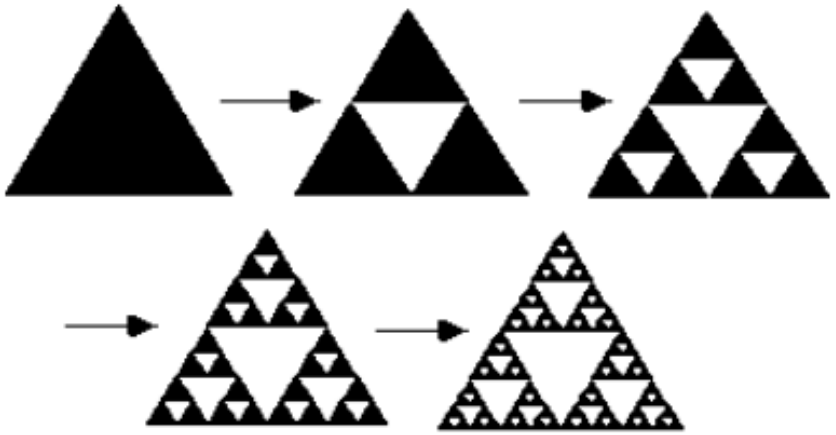
DEF: In generale si definisce **ricorsività o ricorsione** la possibilità di definire un problema riferendosi (ossia ricorrendo) alla sua stessa definizione

Ricorsione e frattali

Il **frattale** è una figura auto-simile, un oggetto geometrico dotato di omotetia interna, ovvero si ripete nella sua forma allo stesso modo su scale diverse, e dunque ingrandendo una qualunque sua parte si ottiene una figura simile all'originale.

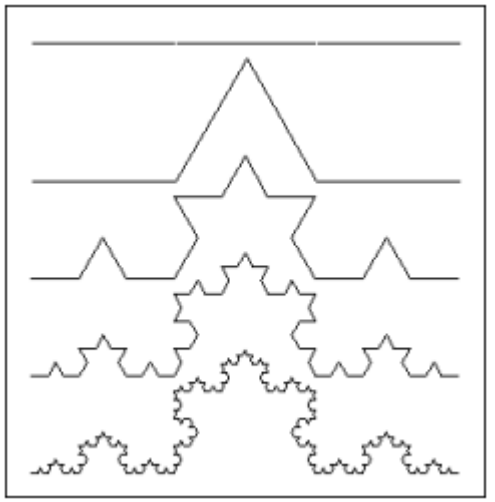
Triangolo di Sierpinski

Si prenda un triangolo equilatero ABC e si uniscano i punti medi di ciascun lato. Si otterranno 4 triangoli simili. Si elimini quello centrale. Si proceda nello stesso modo per ciascun triangolo generato, e si iteri questo processo in ogni passaggio seguente ad ogni triangolo neo–generato.



Curva di Koch

Si prenda un segmento di lunghezza ℓ e lo si divida in 3 segmenti rispettivamente lunghi ℓ/3. Si rimuova il segmento centrale e al suo posto si costruisca un triangolo equilatero privo di base di lato ℓ/3. Si iteri tale procedimento per ciascun lato-segmento formato



Ricorsione e frattali

Una **funzione** in grado di descrivere un **frattale** è una **funzione ricorsiva**.

Infatti, la sua costruzione si basa su un **algoritmo** che permette, attraverso un processo di iterazione (teoricamente infinito), di costruire una curva che si avvicina sempre più al risultato finale, al punto che eventuali modifiche di essa non siano più distinguibili dall'occhio umano.

Frattali nel/sul nostro corpo

Alveoli polmonari

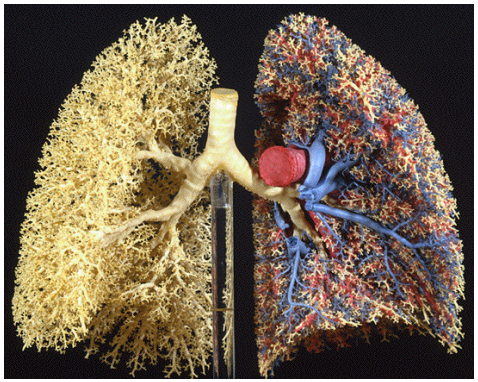


Figure di **Lichtenberg**

Possono anche apparire sulla pelle delle vittime di un fulmine. Si tratta di motivi rossastri simili a felci che possono persistere per ore o giorni.

Frattali in natura

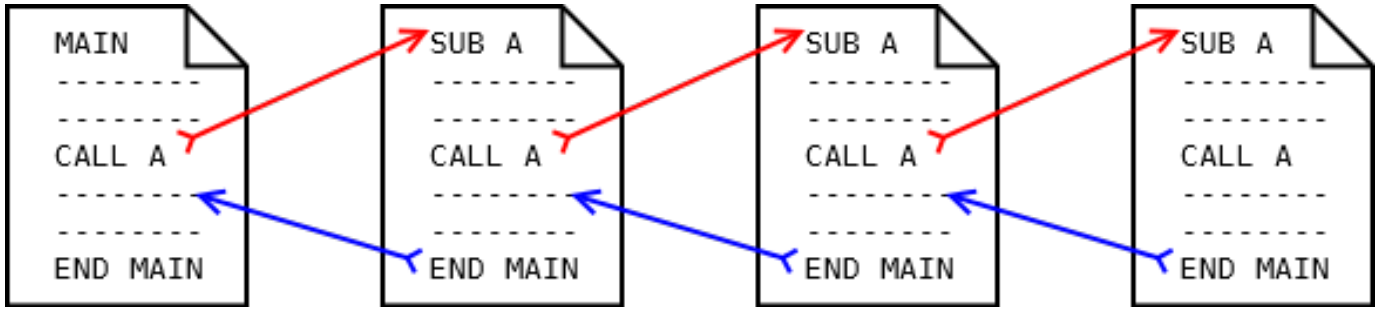
Cavolo romano

Cristallo di ghiaccio

I tre requisiti FONDAMENTALI per l'attivazione di un processo ricorsivo

Per potere attivare un **processo risolutivo di tipo ricorsivo** occorre rispettare **i seguenti tre requisiti**:

1. il problema principale (**caso generale**) può essere scomposto in sottoproblemi dello stesso tipo, ognuno dipendente dall'altro in base ad una **scala gerarchica** (ossia con ordine di complessità inferiore);
2. è necessario conoscere la soluzione di un **caso particolare** del problema principale; ciò è indispensabile per poter arrestare la ricorsione (**condizione di terminazione** della ricorsione);
3. devono essere note le relazioni funzionali che legano il problema principale (**caso generale**) ai sottoproblemi simili



N.B. Il sottoprogramma A chiama il sottoprogramma A … finché non succede qualcosa (l'istanza diventa un caso particolare di cui conosco la soluzione e che quindi non necessita di un'ulteriore chiamata ricorsiva) e si ritorna all'indietro fino alla prima chiamata…

Le tre tipologie di ricorsione: la RICORSIONE DIRETTA

Esistono tre tipologie di ricorsione:

A. Ricorsione **DIRETTA** 

B. Ricorsione **MULTIPLA**

C. Ricorsione **INDIRETTA**

A. DEF: Un sottoprogramma implementa la **ricorsione DIRETTA** quando nella sua definizione compare **UNA SOLA CHIAMATA** al sottoprogramma stesso.

Esempi classici *di problemi che ammettono una soluzione* ***ricorsiva DIRETTA****:*

- ***Fattoriale di un numero***
- ***Potenza ennesima di un numero***

A1. Esempio di RICORSIONE DIRETTA: la potenza di un numero

In matematica sono possibili due definizioni generali di **potenza n-sima** di un numero **a** intero non nullo elevato ad un esponente **n** intero non negativo.

1) definizione **non ricorsiva**

$$\begin{cases} a^n = 1 & \text{se } n = 0 \text{ e } a \neq 0 \\ a^n = \underbrace{a * a * \ldots\ldots\ldots * a}_{\text{n volte}} & \text{se } n > 0 \end{cases}$$

2) definizione **ricorsiva**

$$\begin{cases} a^n = 1 & \text{se } n = 0 \text{ e } a \neq 0 \\ a^n = a * a^{(n-1)} & \text{se } n > 0 \end{cases}$$

Questo problema è stato espresso in termini ricorsivi, in quanto risponde ai tre requisiti enunciati poco fa. Infatti:

- sappiamo che calcolare a^n dipende esclusivamente dal calcolo di $a^{(n-1)}$ **(scala gerarchica);**
- conosciamo la soluzione del **caso particolare ossia** che $a^0 = 1$ **(condizione di terminazione);**
- abbiamo una **relazione funzionale** $a^n = a * a^{(n-1)}$ che lega il problema principale (a^n) al sottoproblema simile ma di complessità minore ($a^{(n-1)}$)

A1. Esempio di RICORSIONE DIRETTA: la potenza di un numero

Soluzione non ricorsiva (algoritmo iterativo) della potenza di un numero

```
FUNZIONE Potenza (VAL base: INT, VAL esp: INT) : INT
i, pot : INT
INIZIO
pot ← 1
PER i←1 A esp ESEGUI
   pot ← pot * base
   i ← i + 1
FINE PER
RITORNA (pot)
FINE
```

Soluzione ricorsiva (algoritmo ricorsivo) della potenza di un numero

```
FUNZIONE Potenza (VAL base: INT, VAL esp: INT) : INT
pot: INT
INIZIO
SE (esp = 0)
    ALLORA
        pot ← 1        <-- /* caso particolare: condizione di arresto */
    ALTRIMENTI
        pot ← base * Potenza (base, (esp -1))   <-- /* caso generale: relazione funzionale*/
FINE SE
RITORNA (pot)
FINE
```

Chiamata ricorsiva DIRETTA
N.B. Chiamata **UNA VOLTA SOLA** allo stesso sottoprogramma

A1. Esempio di RICORSIONE DIRETTA: la potenza di un numero

Schematizzazione della chiamata ricorsiva Potenza (2, 5)

Esempio: Supponiamo di avere una chiamata ricorsiva del tipo:

pot ← Potenza(2, 5) *Ecco cosa accade nella PILA delle ATTIVAZIONI:*

1° call ricorsiva pot ← *Potenza (2, 5)*

2° call ricorsiva pot ← **2** * *Potenza(2, 4)*

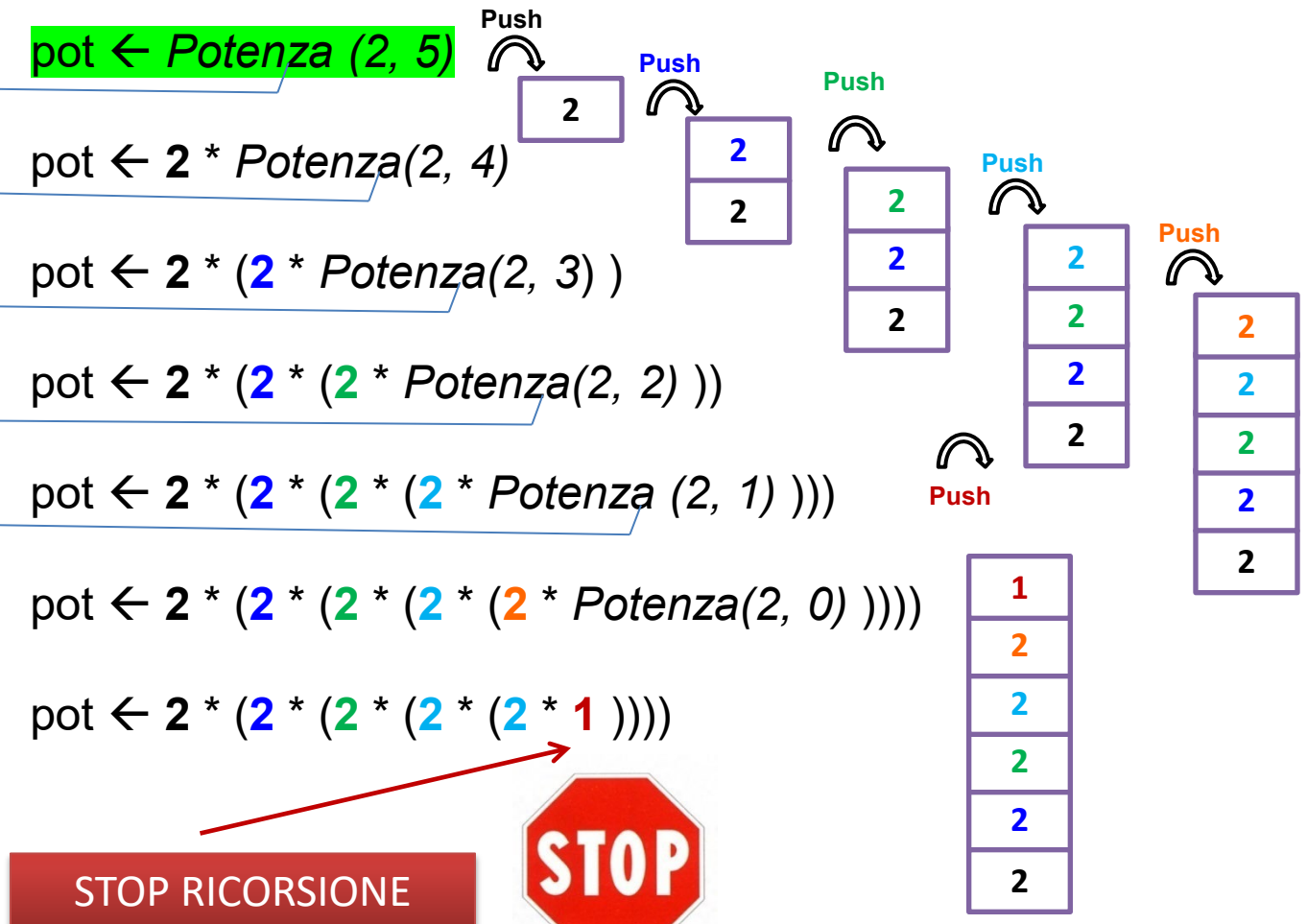
3° call ricorsiva pot ← **2** * (**2** * *Potenza(2, 3)*)

4° call ricorsiva pot ← **2** * (**2** * (**2** * *Potenza(2, 2)*))

5° call ricorsiva pot ← **2** * (**2** * (**2** * (**2** * *Potenza (2, 1)*)))

6° call ricorsiva pot ← **2** * (**2** * (**2** * (**2** * (**2** * *Potenza(2, 0)*))))

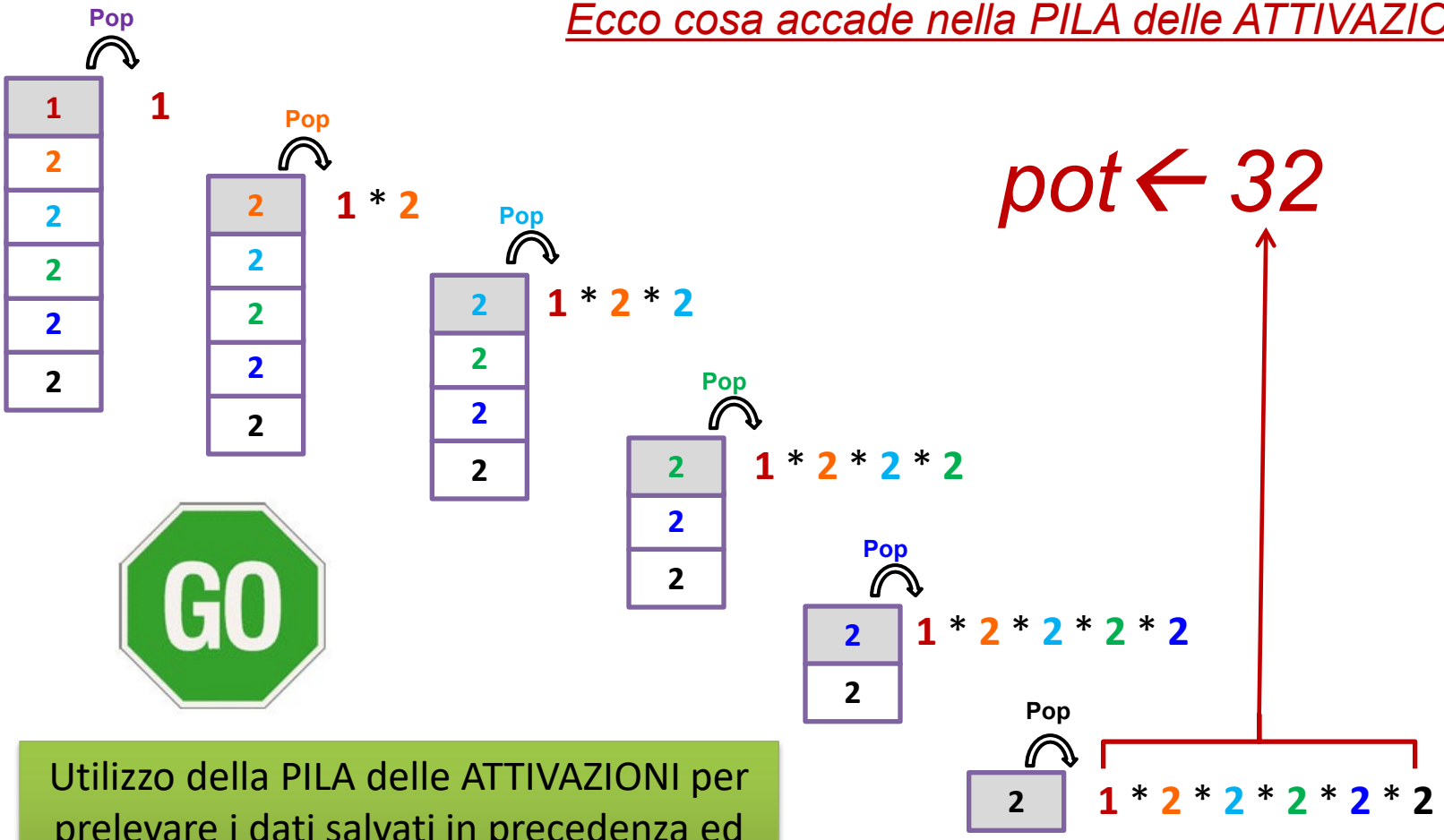
calcolo pot ← **2** * (**2** * (**2** * (**2** * **1**))))



A1. Esempio di RICORSIONE DIRETTA: la potenza di un numero

Schematizzazione della chiamata ricorsiva Fattoriale(5)

Ecco cosa accade nella PILA delle ATTIVAZIONI:



Utilizzo della PILA delle ATTIVAZIONI per prelevare i dati salvati in precedenza ed effettuare i calcoli necessari al competamento della chiamata ricorsiva

A2. Esempio di RICORSIONE DIRETTA: il fattoriale di un numero

In matematica sono possibili due definizioni generali di **fattoriale** di un numero **n** intero positivo.

1) definizione **non ricorsiva**

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

2) definizione **ricorsiva**

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

Anche questo problema è stato espresso in termini ricorsivi, in quanto risponde ai tre requisiti enunciati poco fa. Infatti:

- sappiamo che calcolare **n!** dipende esclusivamente dal calcolo di **(n-1)!** **(scala gerarchica);**
- conosciamo la soluzione del **caso particolare** che **0! = 1** **(condizione di terminazione);**
- abbiamo una **relazione funzionale** **n! = n * (n-1)!** che lega il problema principale (**n!**) ad un sottoproblema simile ma di complessità minore (**(n-1)!**).

A2. Esempio di RICORSIONE DIRETTA: il fattoriale di un numero

Soluzione non ricorsiva (algoritmo iterativo) del fattoriale di un numero

```
FUNZIONE Fattoriale (VAL num: INT) : INT
i, fatt : INT
INIZIO
fatt← 1
PER i←num INDIETRO A 1 ESEGUI
   fatt ← i * fatt
   i ← i - 1
FINE PER
RITORNA (fatt)
FINE
```

Soluzione ricorsiva (algoritmo ricorsivo) del fattoriale di un numero

```
FUNZIONE Fattoriale (VAL num: INT) : INT
fatt: INT
INIZIO
SE (num = 0)
    ALLORA
       fatt← 1        <-- /* caso particolare:  condizione di arresto */
    ALTRIMENTI
       fatt ← num* Fattoriale (num - 1)   <-- /* caso generale: relazione funzionale*/
FINE SE
RITORNA (pot)
FINE
```

Chiamata ricorsiva DIRETTA
N.B. Chiamata **UNA VOLTA SOLA** allo stesso sottoprogramma

A2. Esempio di RICORSIONE DIRETTA: il fattoriale di un numero

Schematizzazione della chiamata ricorsiva Fattoriale(5)

Esempio: Supponiamo di avere una chiamata ricorsiva del tipo:

fatt ← Fattoriale (5) *Ecco cosa accade nella PILA delle ATTIVAZIONI:*

1° call ricorsiva fatt ← *Fattoriale (5)*

2° call ricorsiva fatt ← **5** * *Fattoriale (4)*

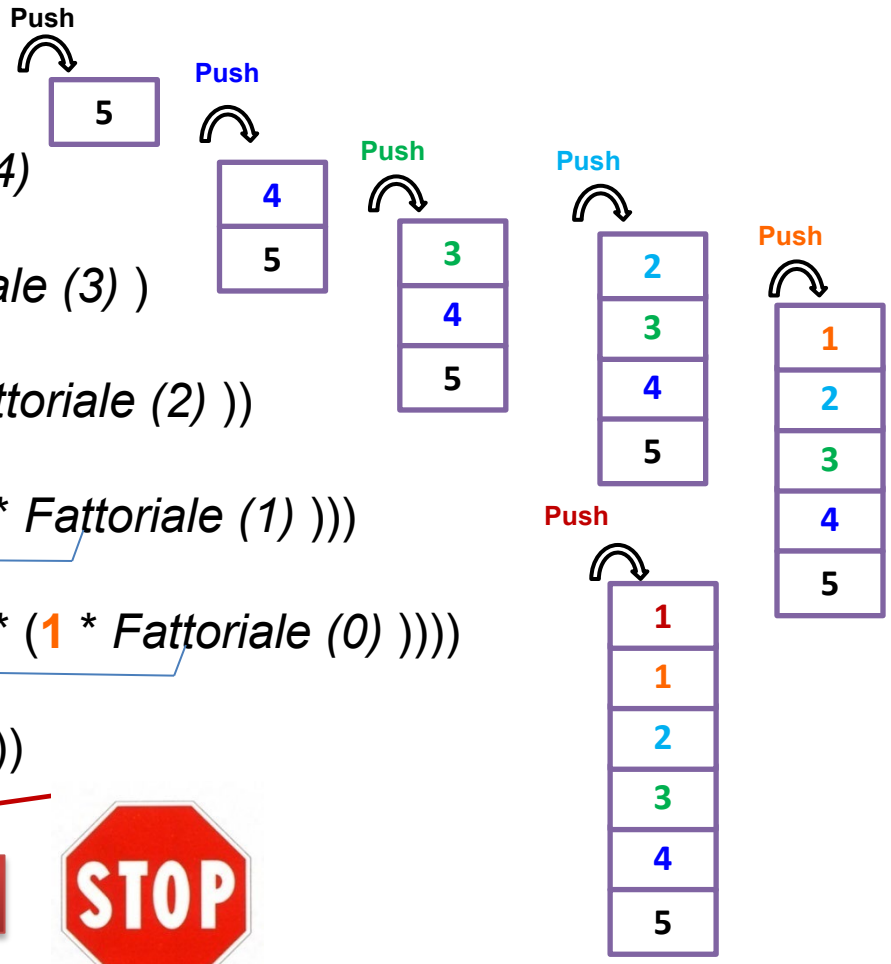
3° call ricorsiva fatt ← **5** * (**4** * *Fattoriale (3)*)

4° call ricorsiva fatt ← **5** * (**4** * (**3** * *Fattoriale (2)*))

5° call ricorsiva fatt ← **5** * (**4** * (**3** * (**2** * *Fattoriale (1)*)))

6° call ricorsiva fatt ← **5** * (**4** * (**3** * (**2** * (**1** * *Fattoriale (0)*))))

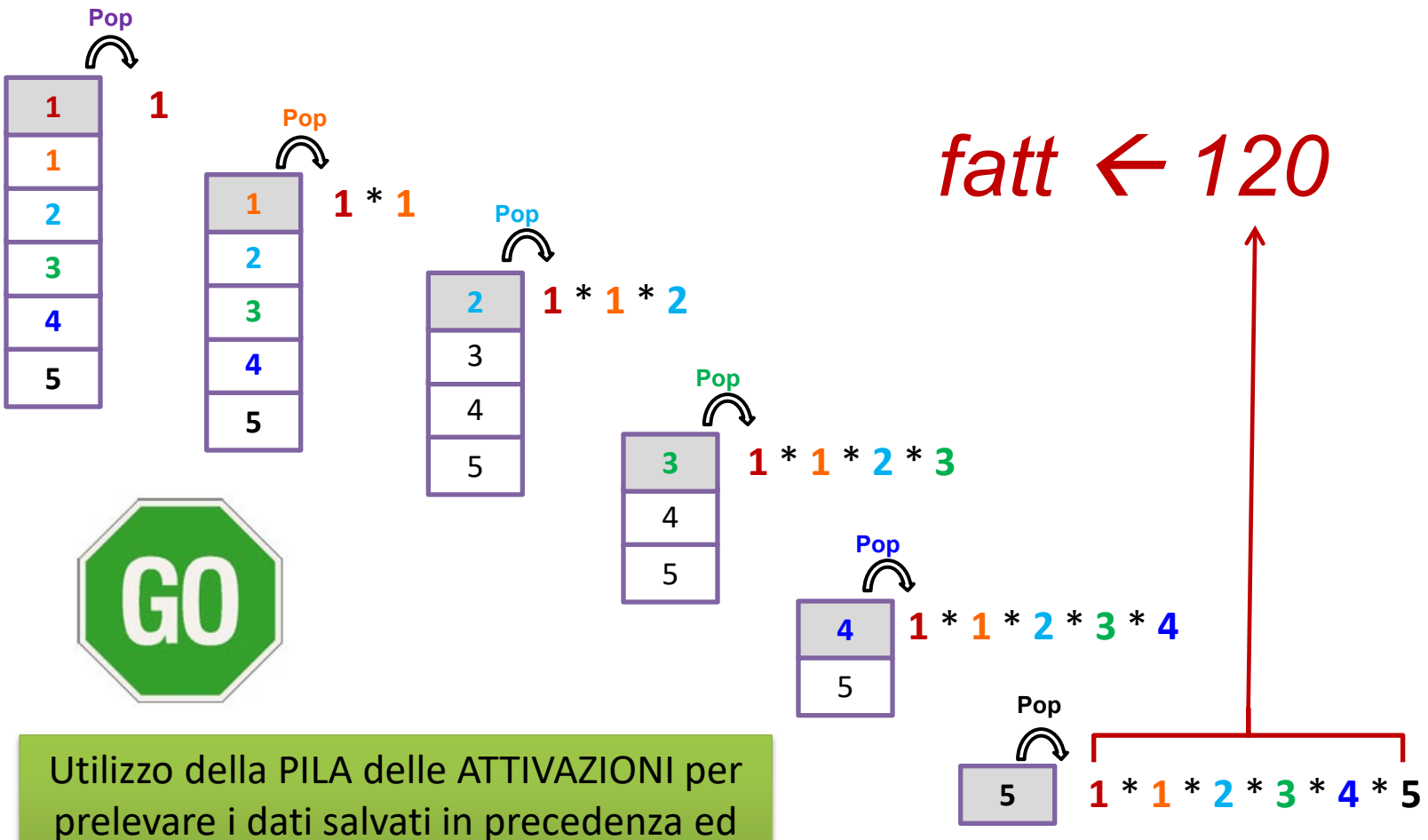
calcolo fatt ← **5** * (**4** * (**3** * (2 * (**1** * **1**))))



STOP RICORSIONE

A2. Esempio di RICORSIONE DIRETTA: il fattoriale di un numero

Schematizzazione della chiamata ricorsiva Fattoriale(5)



Utilizzo della PILA delle ATTIVAZIONI per prelevare i dati salvati in precedenza ed effettuare i calcoli necessari al competamento della chiamata ricorsiva

Le tre tipologie di ricorsione: la RICORSIONE DIRETTA

Esistono tre tipologie di ricorsione:

A. Ricorsione **DIRETTA**
B. Ricorsione **MULTIPLA** 
C. Ricorsione **INDIRETTA**

B. DEF: Un sottoprogramma implementa la **ricorsione MULTIPLA** quando nella sua definizione compaiono **ALMENO DUE CHIAMATE** al sottoprogramma stesso.

Esempio classico *di problema che ammette una soluzione* ***ricorsiva MULTIPLA****:*

- ***Serie di Fibonacci***

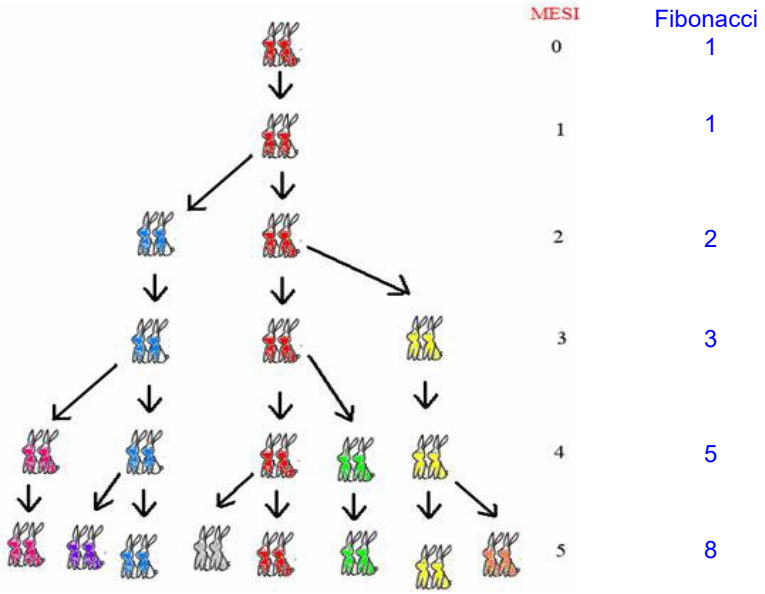
B1. Esempio di RICORSIONE MULTIPLA: la serie di FIBONACCI

In matematica La **successione di Fibonacci** (detta anche **successione aurea**), indicata con **Fib** indica una successione di numeri interi in cui ciascun numero è la somma dei due precedenti, eccetto i primi due che sono, per definizione: **Fib (0) = 1** e **Fib (1) = 1**

The Fibonacci Sequence

1,1,2,3,5,8,13,21,34,55,89,144,233,377…

1+1=2	13+21=34
1+2=3	21+34=55
2+3=5	34+55=89
3+5=8	55+89=144
5+8=13	89+144=233
8+13=21	144+233=377



La **definizione ricorsiva** della serie di Fibonacci relativa ad un numero **n intero positivo**

Fib (n) = 1	**se n = 0**
Fib (n) = 1	**se n = 1**
Fib (n) = Fib (n – 2) + Fib (n – 1)	**se n ≥ 2**

B1. Esempio di RICORSIONE MULTIPLA: la serie di Fibonacci

Soluzione ricorsiva (algoritmo ricorsivo) della serie di Fibonacci

```
FUNZIONE Fibonacci (VAL num : INT) : INT
fib : INT
INIZIO
SE ((num = 0) OR (num = 1))
    ALLORA
        fib ← 1          /* caso particolare: condizione di arresto*/
    ALTRIMENTI
        fib ← Fibonacci (num - 2) + Fibonacci (num - 1)     /* caso generale: relazione funzionale*/
FINE SE
RITORNA (fib)
FINE
```

Chiamata ricorsiva MULTIPLA
N.B. PIU'di UNA CHIAMATA allo stesso sottoprogramma (in questo caso DUE)

B1. Esempio di RICORSIONE MULTIPLA: la serie di Fibonacci

Schematizzazione della chiamata ricorsiva **Fibonacci (4)**

Esempio: Supponiamo di avere una chiamata ricorsiva del tipo:

fib ← Fibonacci (4) *Ecco cosa accade nella PILA delle ATTIVAZIONI:*

1° call ricorsiva fib ← *Fibonacci (4)*

2° call ricorsiva fib ← *Fibonacci (2)* + *Fibonacci (3)*

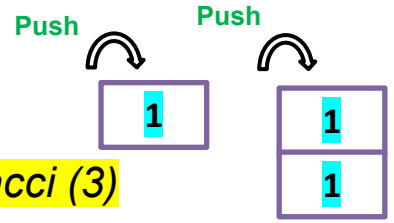
3° e 4° call ricorsiva fib ← *Fibonacci* (0) + Fibonacci (1) + *Fibonacci (3)*

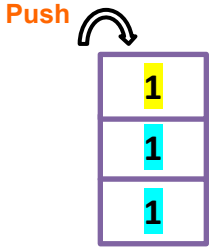
5° call ricorsiva fib ← **1** + **1** + *Fibonacci (3)*

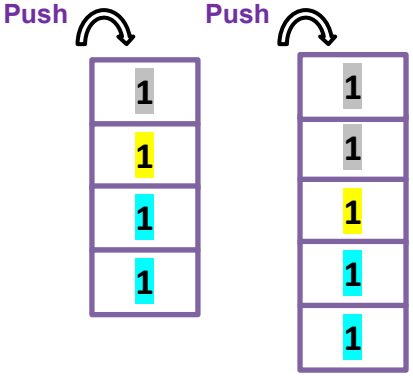
6° e 7° call ricorsiva fib ← **1** + **1** + *Fibonacci (1)* + *Fibonacci (2)*

8° e 9° call ricorsiva fib ← **1** + **1** + **1** + *Fibonacci (0)* + *Fibonacci (1)*

calcolo fib ← **1** + **1** + **1** + **1**+ **1**

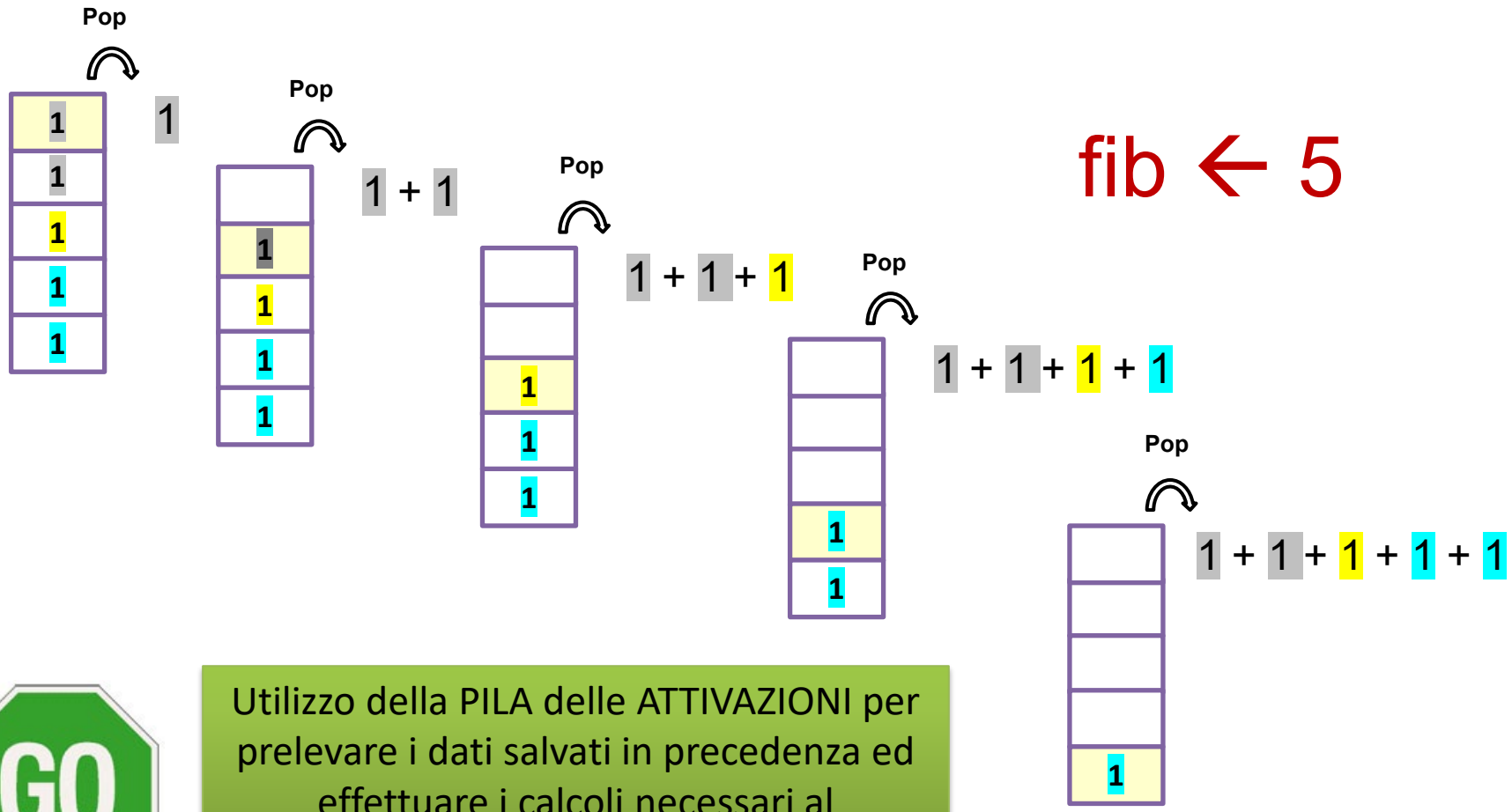






B1. Esempio di RICORSIONE MULTIPLA: la serie di Fibonacci

Schematizzazione della chiamata ricorsiva **Fibonacci (4)**



Utilizzo della PILA delle ATTIVAZIONI per prelevare i dati salvati in precedenza ed effettuare i calcoli necessari al competamento della chiamata ricorsiva

Le tre tipologie di ricorsione: la RICORSIONE DIRETTA

Esistono tre tipologie di ricorsione:

A. Ricorsione **DIRETTA**

B. Ricorsione **MULTIPLA**

C. Ricorsione **INDIRETTA**

C. DEF: Si parla invece di ricorsione **INDIRETTA** quando nella definizione di un sottoprogramma compare la chiamata ad un altro sottoprogramma il quale, **direttamente o indirettamente**, chiama il sottoprogramma iniziale.

Esempio: *di problema che ammette una soluzione* ***ricorsiva INDIRETTA*** *(funzioni cooperanti)*

```
FUNZIONE F1 (VAL n : INT) : INT
esito: INT
INIZIO
SE (n < 1)
   ALLORA
      esito ← 1
   ALTRIMENTI
      esito ← F2 (n-1)
FINE SE
RITORNA (esito)
FINE
```

```
FUNZIONE F2 (VAL n : INT) : INT
esito: INT
INIZIO
SE (n < 2)
   ALLORA
      esito ← 0
   ALTRIMENTI
      esito ← F1 (n-2)
FINE SE
RITORNA (esito)
FINE
```

Algoritmi ricorsivi VS Algoritmi iterativi

PRO NELL'USO DI ALGORITMI RICORSIVI RISPETTO A QUELLI ITERATIVI EQUIVALENTI

a) Un algoritmo ricorsivo **è più facile da realizzare** perché risolve un problema complesso riducendolo a problemi dello stesso tipo ma su dati di ingresso più semplici
b) Un algoritmo ricorsivo è **più facile da capire da parte di altri programmatori** che non siano autori di quel software

CONTRO NELL'USO DI ALGORITMI RICORSIVI RISPETTO A QUELLI ITERATIVI EQUIVALENTI

a) Per poter utilizzare un algoritmo ricorsivo occorre **un grosso sforzo iniziale** da parte del progettista/programmatore per poter acquisire una *visione ricorsiva* del problema posto
b) E' facilmente possibile avere ricorsioni che non terminano mai (**ricorsione infinita**) ossia un sottoprogramma che richiama se stesso infinite volte o perché è stata omessa la clausola di terminazione o perché i valori del parametro ricorsivo non si semplificano mai. Dopo un certo numero di chiamate la memoria disponibile per quel sottoprogramma si esaurisce ed esso viene terminato automaticamente con segnalazione **di errore di overflow nello stack** conseguente **crash** del programma in esecuzione
c) **Eccessiva occupazione dello spazio di memoria** attraverso l'utilizzo di variabili locali non necessarie
d) **Aumento dei tempi di esecuzione (soluzione meno efficiente)** soprattutto degli algoritmi ricorsivi che presentano ricorsioni multiple dovute al proliferare delle chiamate ricorsive con conseguente aggravio dei costi di chiamata di un sottoprogramma

Problemi con la ricorsione

(A) Secondo la matematica la definizione ricorsiva della somma dei primi **n** numeri interi strettamente positivi può essere formalizzata nel seguente modo:

$$\begin{cases} \text{Somma}(n) = 1 & \text{se } n = 1 \\ \text{Somma}(n) = n + \text{Somma}(n-1) & \text{se } n > 1 \end{cases}$$

Possiamo quindi scrivere il seguente pseudocodice della funzione ricorsiva Somma

```
FUNZIONE Somma (VAL n : INT) : INT
s: INT
INIZIO
SE (n =1)
   ALLORA
      s ← 1
   ALTRIMENTI
      s ← n + Somma(n-1)
FINE SE
RITORNA (s)
FINE
```

Clausola di terminazione

Il valore di n si semplifica

Problemi con la ricorsione

Dettagliamo la seguente chiamata ricorsiva

….

s ← Somma(5)

…..

Il valore del parametro attuale (costante intera) trasferito al parametro formale ***n*** è **5**.

Attivata la funzione viene eseguita l'istruzione

s ← 5 + Somma (4)

che non può essere eseguita e risolta direttamente in quanto contiene una chiamata allo stesso sottoprogramma (**procedura ricorsiva diretta**) che conduce alla seguente successione

s ← 5 + (4 +Somma (3))

s ← 5 + (4 +(3 + Somma (2)))

s ← 5 + (4 +(3 + (2 + Somma (1))))

s ← 5 + (4 +(3 + (2 + (1)))) che è uguale a **15**

Fin qui TUTTO OK....ma che succede se compio qualche piccolo errore nella progettazione dell'algoritmo ricorsivo?????

Problemi con la ricorsione

RICORSIONE INFINITA...i valori del parametro **NON SI SEMPLIFICANO**

```
FUNZIONE Somma (VAL n : INT) : INT
s: INT
INIZIO
SE (n =1)
    ALLORA
        s ← 1
    ALTRIMENTI
        s ← n + Somma(n)
FINE SE
RITORNA (s)
FINE
```

Condizione di arresto ***presente***

Il valore di ***n*** ***NON si semplifica***

RICORSIONE INFINITA...manca la **CONDIZIONE DI ARRESTO**

```
FUNZIONE Somma (VAL n : INT) : INT
s: INT
INIZIO
    s ← n + Somma(n-1)
RITORNA (s)
FINE
```

Il valore di ***n*** ***si semplifica***