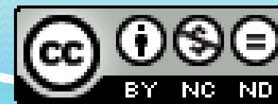


Fondamenti di Informatica

AA 2018/2019

Eng. Ph.D. Michela Paolucci

DISIT Lab http://www.disit.dinfo.unifi.it
Department of Information Engineering, DINFO
University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570
michela.paolucci@unifi.it

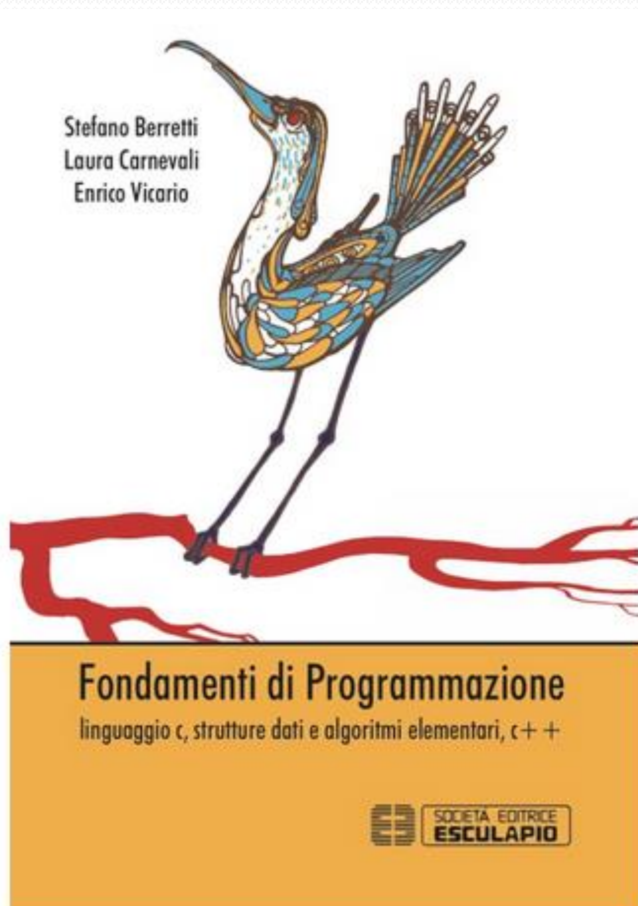
UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

DISIT Lab
http://www.disit.dinfo.unifi.it

Department of Information Engineering, DINFO
University of Florence
Via S. Marta 3, 50139, Firenze, Italy

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Libro di testo



- Stefano Berretti, Laura Carnevali, Enrico Vicario
- Fondamenti di programmazione. Linguaggio C, strutture dati, algoritmi elementari, C++
- Editore: Esculapio
- ISBN: 9788893850513
- NOTA: ***Queste slide integrano e non sostituiscono quanto scritto sul libro di testo.***

Pagina del Corso

http://www.disit.org/drupal/?q=node/7020

Qui trovate:
- AVVISI
- Slide del corso
- Approfondimenti

Orario del Corso e Ricevimento

Insegnamento: FONDAMENTI DI INFORMATICA (A-L) (FonDiInf(A-L))

Ingegneria FIRENZE - A.A. 2018/2019 - 2° periodo

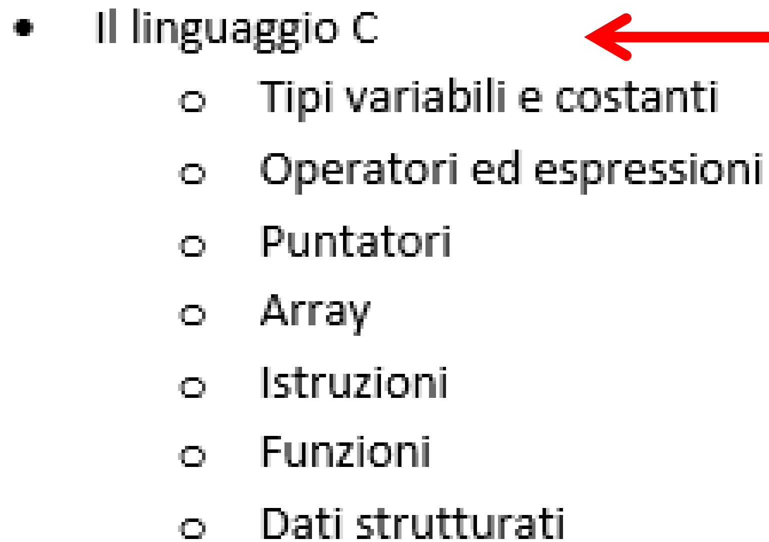
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
08:15						
09:15						
10:15						
11:15	(Auditorium A - C.D.M.)					
12:15	(Auditorium A - C.D.M.)					
14:00					(Auditorium A - C.D.M.)	
15:00					(Auditorium A - C.D.M.)	
16:00					(Auditorium A - C.D.M.)	
17:00						
18:00						

Docenti: PAOLUCCI MICHELA

Legenda: C.D.M.: Centro Didattico Morgagni

- Il ricevimento si svolge su appuntamento contattando la docente via email:
  - michela.paolucci@unifi.it

Outline



Le strutture informative (1)

- Spesso le informazioni elaborate dai programmi non sono dati singoli ma appartengono ad insiemi di dati che possono essere:
  - Omogenei
  - Eterogenei
- Un insieme di dati è:
  - eterogeneo quando in esso sono contenuti dati di diverso tipo
    - Esempio: numeri, nomi, etc.
  - Omogeneo quando contiene solo dati dello stesso tipo
    - Esempio: array di N interi

Le strutture informative (2)

- Le strutture informative possono essere:
  - Astratte
  - Fisiche

- Le strutture **Astratte** vengono usate per descrivere le relazioni fra componenti in modo astratto definendo le proprietà di tali componenti

- Le strutture **Fisiche** descrivono come le componenti sono organizzate nella memoria del computer/elaboratore

Le strutture informative (3)

- Le strutture fisiche possono essere:
  - Statiche
  - Dinamiche
- Le strutture statiche sono quelle la cui organizzazione in termini di memoria di base viene decisa direttamente prima della esecuzione del programma
- Le strutture dinamiche (che vedremo) sono quelle la cui realizzazione in termini di memoria di base viene effettuata solo al momento della esecuzione del programma
- NOTA: le strutture statiche NON possono cambiare le proprie dimensioni durante il programma al contrario di quelle dinamiche

Le strutture informative (4)

- Esempi di struttura dinamica:
  - Elenco telefonico
    - E' una struttura dinamica perché il numero degli elementi nell'elenco può cambiare nel corso di vari anni
  - Catalogo di una biblioteca
    - E' una struttura dinamica perché i libri in un biblioteca varieranno in base ai nuovi arrivi, etc.

Operazioni sulle strutture informative (1)

- Sulle strutture informative si posso effettuare varie operazioni
- Tipicamente tali operazioni si dividono in:
  - Operazioni Locali
  - Operazioni Globali
- Le **Operazioni Locali** interessano un solo elemento della struttura:
  - La modifica di un elemento modificando le componenti del singolo elemento
  - La cancellazione di un elemento della struttura dati, preservando la struttura di base
  - L'inserimento di un elemento nella struttura dati che passa ad avere un elemento in più
  - La visualizzazione delle componenti del singolo elemento della struttura dati

Operazioni sulle strutture informative (2)

- Tipicamente tali operazioni si dividono in:
  - Operazioni Locali
  - Operazioni Globali
- Le **Operazioni Globali** interessano tutta la struttura dati:
  - La concatenazione di due strutture dati compatibili
  - La visita della struttura dati per stampa o ricerca
  - L'ordinamento degli elementi della struttura dati
  - La compattazione di una struttura dati che consiste nel rimuovere le componenti non più utili (ad esempio quelle cancellate) e che quindi porta ad ottenere una struttura dati ridotta rispetto a quella iniziale
  - Il salvataggio su memoria dell'intera struttura dati
  - Il caricamento da memoria dell'intera struttura dati

Strutture Astratte

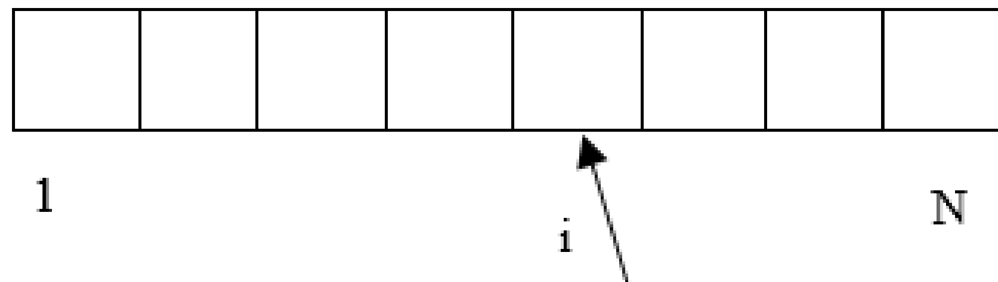
- Le strutture Astratte più importanti sono:
  - La stringa
  - Il vettore
  - La matrice
  - Il record
  - La tabella
  - La lista
  - La pila
  - La coda
  - La coda circolare
  - L'albero
  - Il grafo

La stringa

- La stringa è un insieme ordinato di simboli estratti da un certo alfabeto
- La stringa ha simboli ordinati, nel senso che è possibile riferirsi ai singoli simboli indicandoli per mezzo della loro posizione nella stringa a partire dal primo simbolo all’ultimo secondo un criterio di ordinamento
- Nel caso degli elaboratori, l’alfabeto è quello dei caratteri rappresentabili ovvero quelli codificati tramite la tabella ASCII
- Esempi:
  - monica
  - sdfliufhwebla
  - 214324nps9
  - :{(/<#ib6)HAA*

Il vettore (1)

- Il vettore è un insieme ordinato di elementi dello stesso tipo
- Il vettore ha elementi ordinati nel senso che è possibile riferirsi alle singole componenti indicandole per mezzo di un indice (indice i in figura)
- L'indice identifica in modo ordinato gli elementi del vettore, dal primo all'ultimo secondo un criterio di ordinamento (per esempio in ordine crescente)



Il vettore (2)

- Il vettore ha una dimensione pari al numero di componenti/elementi di cui è composto
- La dimensione è FISSA, ovvero NON può essere variata dopo la sua realizzazione
- I vettori sono tipicamente strutture statiche
- Gli elementi sono tutti dello stesso tipo, quindi il vettore è una struttura omogenea
- Ad esempio si possono avere vettori di:
  - numeri interi
  - reali
  - Booleani
  - etc.

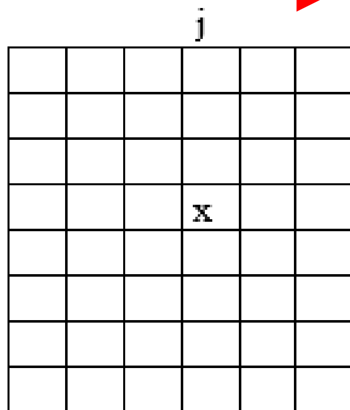
Il vettore (3)

- Il concetto di vettore è di fondamentale importanza applicativa in ogni disciplina scientifica
- L'idea di **aggregare informazioni omogenee** in una unica entità e di poterle reperire successivamente in base alla loro posizione, porta spesso a **formulare problemi in modo compatto ed elegante**
- Il vettore in ambito informatico risulta una struttura ideale per rappresentare tabelle contenenti informazioni, soprattutto quando queste sono di **natura statica**
- Un vettore V che ha dimensione N, viene rappresentato indicando i suoi elementi con V_i o V(i) ma viene anche considerato nella sua interezza tramite V
- V_i o V(i) rappresenta l'elemento i-esimo del vettore V

La matrice (1)

- La matrice è la naturale estensione bidimensionale del vettore
- La matrice ha elementi ordinati, ovvero è possibile riferirsi alle singole componenti indicandole per mezzo di due indici partendo dal primo all'ultimo secondo un criterio di ordinamento che tiene conto di entrambe le dimensioni

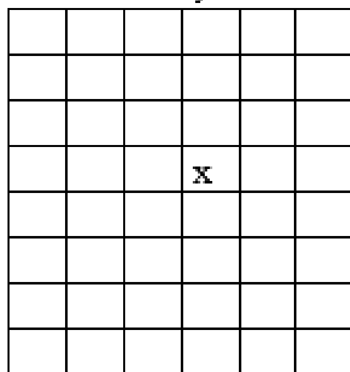
RIGHE



- Sia A(i,j) l'elemento della matrice A con:
  - i, indica la riga partendo dall'alto
  - j, indica la colonna a partire da sinistra

La matrice (2)

- Le matrici devono essere composte da **elementi dello stesso tipo**, pertanto sono strutture **omogenee**
- Le matrici, come i vettori, sono **strutture statiche**, ovvero non è possibile cambiare le dimensioni della matrice dopo la sua creazione iniziale
- La matrice può avere un numero di colonne diverso dal numero di righe (matrice rettangolare)



- Si chiamano matrici quadrate quelle che hanno il numero di colonne uguale al numero di righe

Il record (1)

- Un record è un insieme **ordinato** ed **eterogeneo** di elementi
- Il singolo elemento del record può essere chiamato campo o componente
- In un record si possono avere componenti di vario tipo: interi, numeri naturali, vettori di interi, etc.
- Il record è tipicamente una struttura statica nel senso che non è possibile variarne il numero di componenti dopo la sua creazione iniziale
- Esempio:
  - cognome: Mario
  - nome: Rossi
  - via: Via Ponte Verde
  - civico: 45

Il record (2)

- Tenendo presente l’esempio:
  - cognome: Mario
  - nome: Rossi
  - via: Via Ponte Verde
  - civico: 45
- Ogni campo del record viene identificato dal nome del campo:
  - cognome è il nome del primo campo, e *Rossi* è il relativo contenuto informativo (valore)
  - cognome è una stringa mentre civico è un intero
- Il record si può rappresentare nel modo seguente:
  - {cognome, nome, via, civico}
- Tipicamente il record stesso o meglio la struttura che lo definisce, ha un nome

La tabella (1)

- Le tabelle sono una generalizzazione del concetto di matrice
- La tabella, a differenza della matrice, può contenere dati di tipo diverso. Ad esempio: numeri, commenti, descrizioni, etc.
- Tipicamente ad ogni colonna della tabella è associato un tipo e quindi un significato
- La tabella è **un insieme ordinato ed omogeneo di record**
- Tipicamente la tabella è una struttura dati statica, ovvero le sue dimensioni NON variano dopo la sua realizzazione iniziale

La tabella (2)

- Un record della tabella può essere identificato in due modi:
  - Per indice posizionale
  - Per chiave
- L'**indice posizionale** è l'indice che identifica il singolo record partendo dal primo fino all'ultimo. In questo senso la tabella può essere vista come un insieme di record

Nome	Cognome	Età	Telefono	Indice
Carlo	Zolli	14	0123413412	1
Pino	Polli	24	0234242444	2
Gino	Arbi	4	0345543535	3
Lino	Corbi	56	0455436266	4
….	….	….	….	…

Tabella con acceso per chiave (1)

- Per **chiave** si intende che il singolo record viene identificato per mezzo del contenuto stesso di un campo del record
- Ad esempio nella tabella sottostante potrebbe essere usato il numero di telefono come chiave, oppure l'età, il nome, etc.
- Individuare un record per chiave significa identificare la posizione del record nella tabella
- Esempio: il record di Gino è in posizione di indice 3, l'indice di ordinamento va da 1 a N

Nome	Cognome	Età	Telefono	Indice
Carlo	Zolli	14	0123413412	1
Pino	Polli	24	0234242444	2
Gino	Arbi	4	0345543535	3
Lino	Corbi	56	0455436266	4
….	….	….	….	…

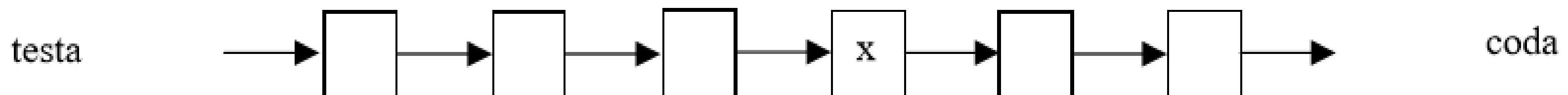
Tabella con accesso per chiave (2)

- La chiave per la quale si effettua una ricerca al fine di individuare il singolo record dovrebbe essere sufficientemente significativa da identificare in modo univoco il record
- Questo potrebbe NON essere possibile (es: se si ordina per cognome, potrebbero esserci due o più cognomi uguali)
- Per risolvere il problema si possono assegnare delle chiavi univoche oppure comporre campi dei record in modo da definire delle chiavi più significative (es: nome + cognome)

Nome	Cognome	Età	Telefono	Indice
Carlo	Zolli	14	0123413412	1
Pino	Polli	24	0234242444	2
Gino	Arbi	4	0345543535	3
Lino	Corbi	56	0455436266	4
….	….	….	….	…

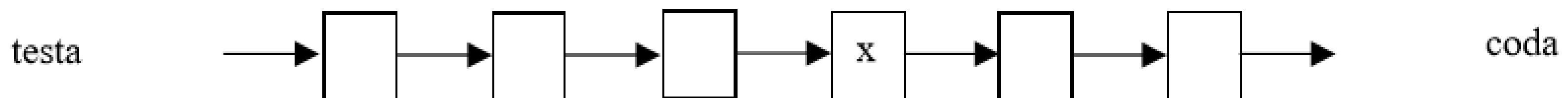
La Lista (1)

- Le liste sono strutture dati che possono essere omogenee e non omogenee, lineari e non lineari
- Si dicono **liste lineari** quelle organizzate come un insieme ordinato di elementi
- Ogni elemento ha un successivo e un predecessore ad esclusione:
  - del primo elemento (**testa**) che NON ha un predecessore
  - dell'ultimo elemento (**coda**) che non ha un successivo
- Per accedere all'elemento finale è necessario partire dal primo elemento della lista (testa) e scorrere su tutti gli elementi successivi fino all'ultimo (coda)
- Questo tipo di organizzazione per l'accesso alle componenti viene detto **sequenziale**



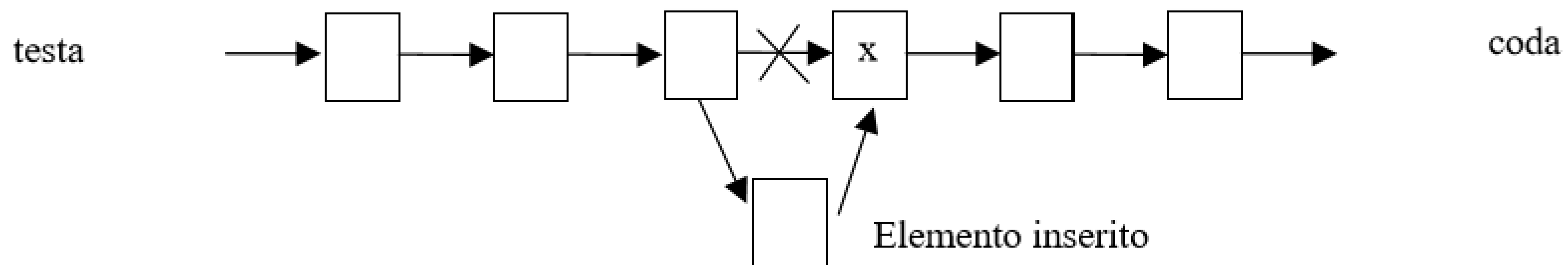
La Lista (2)

- Le liste sono strutture dati ad accesso **sequenziale**
- La lista può contenere tipicamente un **numero variabile di elementi nel tempo**, ovvero la sua dimensione in termini di elementi può variare dopo la sua creazione iniziale, quindi deve essere considerata una struttura **dinamica**
- Al momento della sua creazione, la lista è tipicamente **vuota**, in seguito vengono inseriti gli elementi connettendoli secondo regole di ordinamento o convenienza
- Su una lista si possono effettuare operazioni di:
  - Inserimento, cancellazione, visita, modifica, creazione, concatenazione di due liste, etc.
  - Una lista si crea inserendo il **primo elemento**



Inserimento in una lista (1)

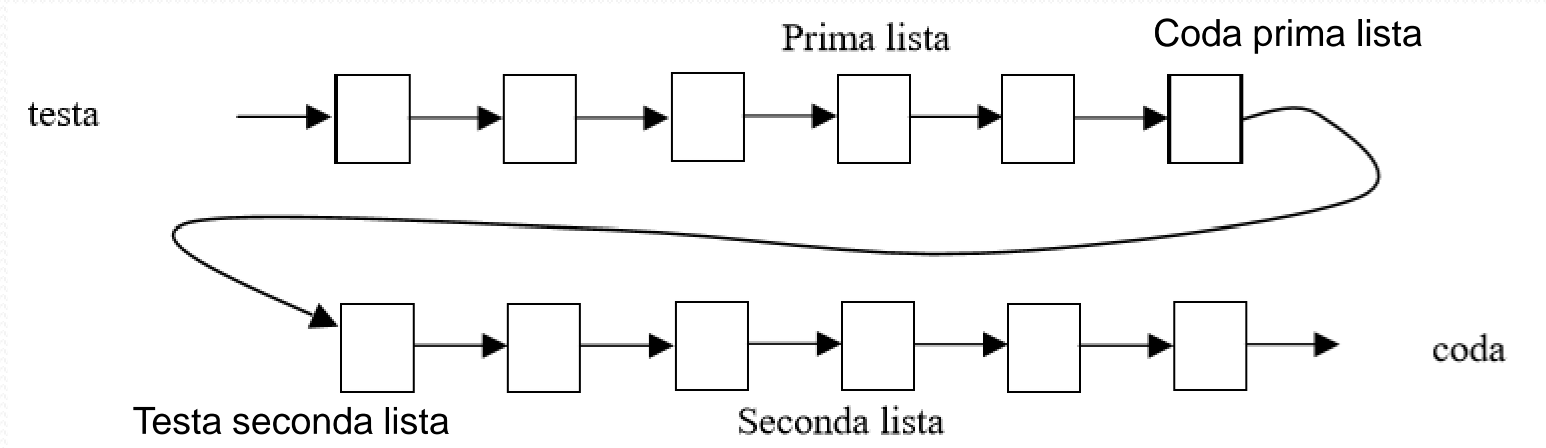
- L'inserimento in testa e in coda è semplice da comprendere: si aggiunge un ulteriore elemento e lo si collega alla lista
- L'inserimento di un elemento all'interno di una lista deve essere effettuato 'aprendo' la lista e 'ristabilendo' i legami tra gli elementi
- Si possono inserire anche elementi in testa e in coda con altre modalità…
- Se NON vi sono elementi, allora la lista è vuota
- Una lista con un solo elemento può diventare vuota se questo unico elemento viene cancellato



Concatenazione di liste

- Due liste possono essere concatenate nel senso che una lista può essere congiunta con un’altra in modo da formare un’unica lista con tutti gli elementi

- Questo avviene effettuando il collegamento tra la coda della prima lista e la testa della seconda lista



Differenza tra lista e vettore

- Le maggiori differenze tra liste e vettori sono le seguenti:
  - **Il vettore ha un accesso diretto ai singoli elementi MENTRE la lista ha un accesso sequenziale**
- La lista può avere un numero variabile di elementi nel tempo ed è quindi una struttura **dinamica**, al contrario del vettore che è una struttura **statica**
- La lista può anche contenere elementi di tipo diverso
- La lista può anche essere NON lineare

Descrizione di liste lineari e non lineari

- Gli elementi di una lista B possono essere elencati nel modo seguente:

B (3,5,6,77,88,92) -> lista lineare

- In questo caso la lista ha 6 elementi
- Su una lista sono possibili le operazioni di: ricerca, cancellazione, concatenazione, visita, etc.

- Si dicono liste non lineari quelle liste che contengono fra i loro elementi altre liste:

G (B, A, F, (2, 5,7), G, (G,T))

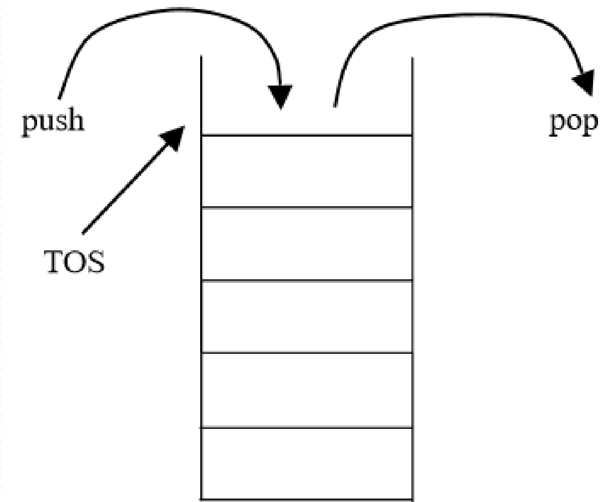
- La lista G contiene alcuni elementi singoli ma anche altre liste

- Alberi e Grafi sono liste NON lineari

La pila (o stack) (1)

- La pila (o stack) è una struttura lineare dove le operazioni di inserimento ed estrazione dei dati avvengono sempre dalla stessa parte

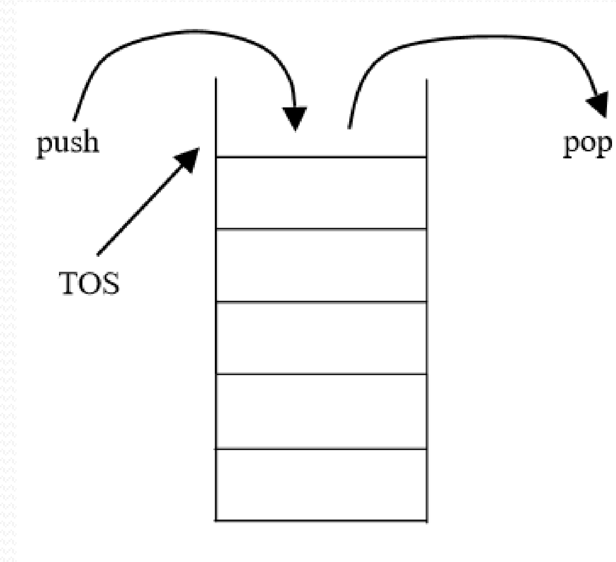
- La pila è una struttura informativa di tipo:
  - **FILO (First Input Last Output**), se il primo a entrare è l'ultimo ad uscire
  - **LIFO** (**Last Input First Output**) se l'ultimo a entrare è anche il primo a uscire.



- Esempio: tubetto di pasticche delle vitamine: per prendere l'ultima pasticca (ovvero la prima che è stata inserita) è necessario togliere tutte le pasticche precedenti

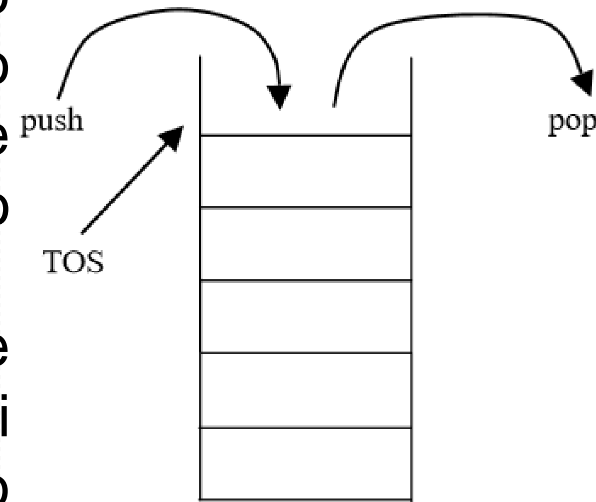
La pila (o stack) (2)

- L'operazione di inserimento è detta PUSH
- L'operazione di estrazione è detta POP
- Per la gestione di una pila è necessario conoscere solo il punto in cui viene fatta l'inserzione o l'estrazione, detto Top Of the Stack, TOS



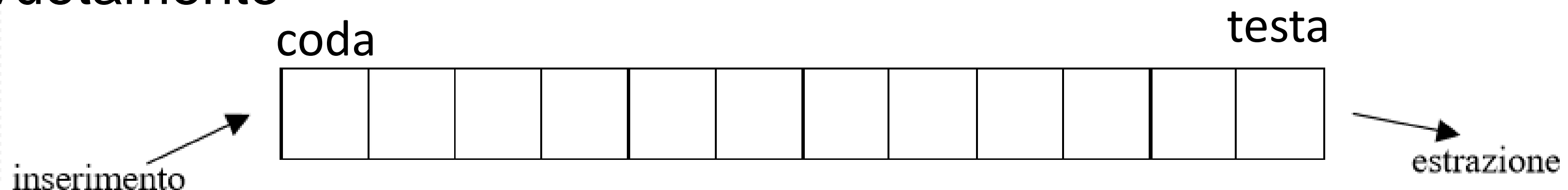
La pila (o stack) (3)

- Lo **stack** è tipicamente una **struttura dinamica** perché le sue dimensioni possono non essere note al momento della sua realizzazione
- Le operazioni sulla pila fanno tutte riferimento al TOS (Top Of the Stack), quindi è necessario monitorare tale elemento, in pratica è necessario tenere sotto controllo quando lo stack si svuota
- Se NON si effettua tale controllo, potrebbe infatti accadere di effettuare una operazione di POP (estrazione) senza che vi sia un elemento nello stack
- Le operazioni di base sono pertanto:
  - Creazione della pila, inserimento (push), estrazione (pop), verifica di stack vuoto PRIMA di fare ogni pop



La coda (o buffer) (1)

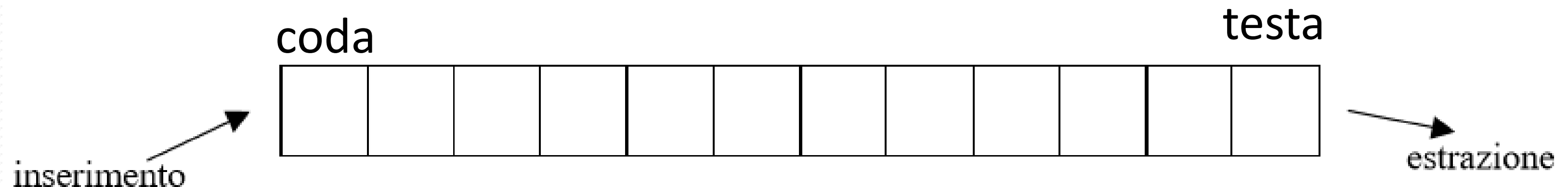
- La **coda** (o buffer) è una struttura lineare dove le operazioni di inserimento vengono effettuate da una parte, mentre quelle di estrazione da un’altra
- Tipicamente la parte in cui si fanno le estrazioni viene chiamata **testa** mentre quella in cui si effettuano gli inserimenti viene chiamata **coda** (si dice infatti ‘metti in coda’)
- La coda è tipicamente una struttura dinamica, ovvero le sue dimensioni possono non essere note al momento della sua realizzazione
- La creazione della coda si effettua inserendo il primo elemento, mentre la coda si svuota togliendo l’ultimo elemento rimasto
- Le operazioni di base sono:
  - Creazione, inserimento, estrazione, verifica di coda vuota, svuotamento



La coda (o buffer) (2)

- Il meccanismo di gestione della coda si basa sulla politica:
  - FIFO (First Input First Output), il primo ad entrare è il primo ad 'essere servito' cioè ad uscire dalla coda
- Lo stesso meccanismo può essere espresso anche con la politica LILO (Last Input Last Output), l'ultimo ad entrare è anche l'ultimo ad uscire

- Le operazioni di base per una coda sono:
  - creazione, inserimento, estrazione, verifica di coda vuota, svuotamento



Le strutture interne

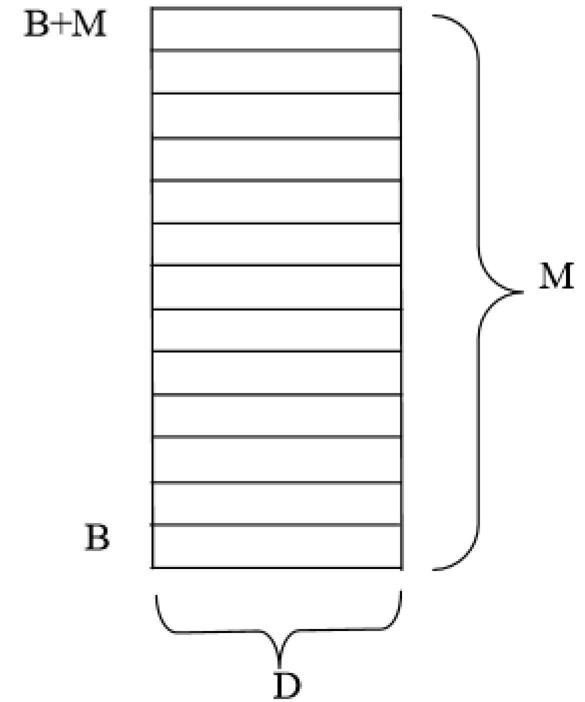
- Le strutture interne alla memoria sono:
  - Il vettore di memoria
  - La lista concatenata semplice
  - La lista concatenata doppia

Il vettore di memoria, il puntatore (1)

- Il vettore di memoria è una struttura fisica interna
- Il vettore di memoria NON deve essere confuso con il vettore come struttura astratta (o array)
- Il vettore di memoria è la struttura astratta di memorizzazione più semplice ed è costituita da elementi memorizzati in celle contigue di memoria identificate da un indirizzo di memoria iniziando da un indirizzo di partenza (o di base) B che può essere zero, 0
- L'indirizzo della memoria viene detto puntatore
- Il Puntatore è l'indice all'interno del vettore di memoria che rappresenta la memoria stessa dell'elaboratore
- La memoria ha una dimensione M che viene espressa in numero di celle
- Ogni cella può avere una dimensione D, da 1 a più byte:2,4,8

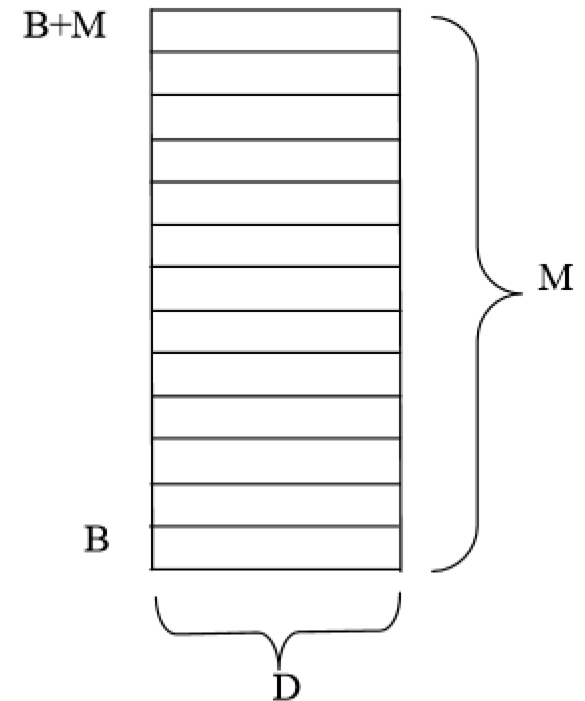
Il vettore di memoria, il puntatore (2)

- Il Puntatore è l'indice all'interno del vettore di memoria che rappresenta la memoria stessa dell'elaboratore
- La memoria ha una dimensione M che viene espressa in numero di celle
- Ogni cella può avere una dimensione D, da 1 a più byte:2,4,8
- Se si considera un insieme di numeri interi di 2 byte che sono inseriti nel vettore di memoria a partire dal suo inizio, l'elemento i-esimo del vettore si troverà ad avere il seguente indirizzo di cella:
  - Indirizzo di cella = indirizzo di base B + 2byte*i



Il vettore di memoria, il puntatore (3)

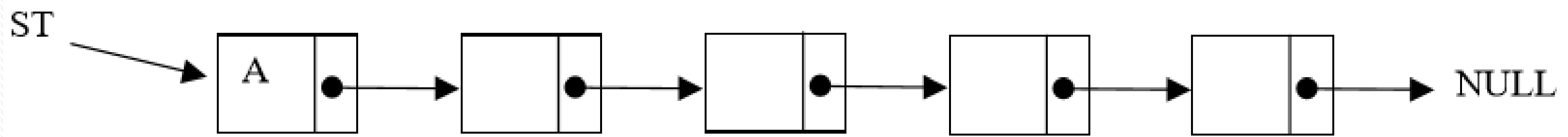
- La dimensione in byte della memoria viene stimata considerando:
  - Dimensione della cella e numero delle celle
  - Dimensione della memoria in byte = M(numero di celle)*D
- L'indirizzo di cella NON deve mai superare la dimensione in byte della memoria aumentata dell'indirizzo di partenza:
  - Indirizzo cella < B + M*D
- Il vettore di memoria può essere usato per contenere tutte le strutture astratte che sono state discusse fino ad ora



- B, indirizzo di Base
- M, indirizzo della Memoria, in numero di celle
- D, dimensione in byte di una cella di memoria

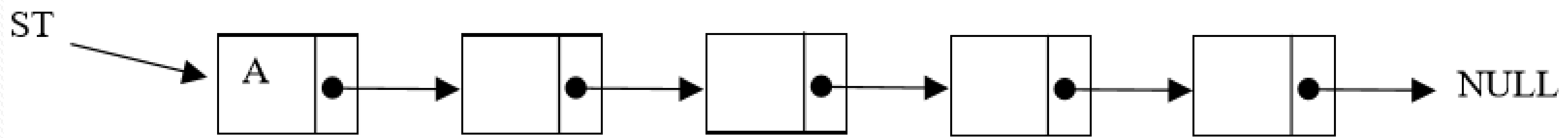
Lista concatenata semplice, catena semplice 1

- La **lista concatenata** o catena semplice è una struttura interna ordinata per la quale elementi consecutivi NON sono disposti in memorie fisiche in celle consecutive
- La sequenza, e quindi l'ordine degli elementi in una lista, viene data in base ai legami che vengono definiti tra gli elementi stessi della lista
- I singoli elementi della lista possono essere dei record
- Ogni record contiene: un contenuto informativo (Non obbligatorio), un campo chiave che distingue un record dagli altri e un campo che contiene il riferimento all'elemento successivo



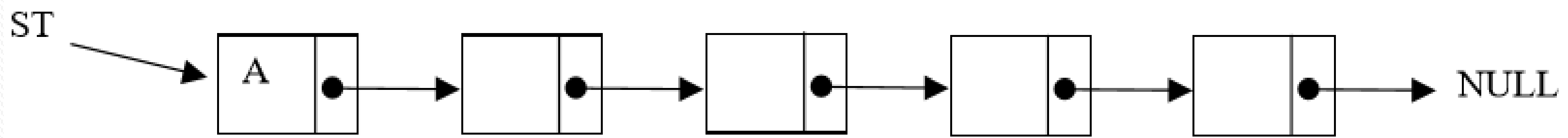
Lista concatenata semplice, catena semplice 2

- Questa struttura dati è molto più flessibile rispetto al vettore, poiché:
  - è possibile inserire elementi anche all'interno della lista (senza dover riorganizzare la lista) e non solo alle estremità come nella coda
- Ogni elemento della lista contiene una parte dati e una parte per referenziare l'elemento successivo (**Puntatore**)
- Se si conosce un elemento (es: A in figura), si può fare riferimento all'elemento successivo nella lista (seguendo il puntatore)



Lista concatenata semplice, catena semplice 3

- Se si conosce un elemento (es: A in figura), si può fare riferimento all'elemento successivo nella lista (seguendo il puntatore)
- Il primo elemento viene identificato da un puntatore (in figura ST)
- L'ultimo elemento non riferisce a nessun elemento, pertanto il suo puntatore viene impostato ad un valore prefissato e nullo, pari a NULL



Lista concatenata semplice, catena semplice 4

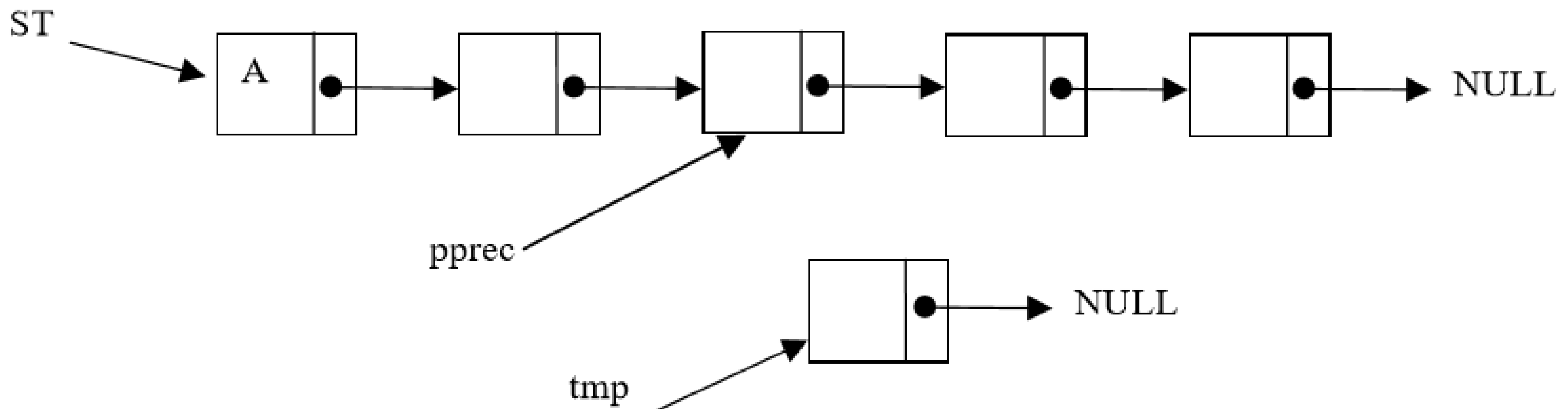
- La **lista** è una struttura dati sequenziale, pertanto non è possibile raggiungere il singolo elemento se non si conosce il suo INDIRIZZO di memoria tramite un PUNTATORE
- Confronto con il vettore:
  - La lista necessita di un maggior numero di byte di memoria per la presenza dei puntatori
  - La lista è una struttura dati più flessibile poichè le seguenti operazioni sono di più semplice realizzazione:
    - Cancellazione: cancellaz. di un elemento e aggiornamento del puntatore
    - Inserimento: inserimento di un nuovo elemento riorganizzando i puntatori degli elementi in cui il nuovo è inserito

Inserimento in lista concatenata semplice, catena semplice (1)

- Le operazioni di inserimento in testa e in coda alla lista sono semplici
- L'inserimento di un elemento lungo la lista è invece più complesso, è necessario:
  - Interrompere la catena
  - Aggiungere un ulteriore elemento
  - Ristabilire i legami
  - Fare attenzione a non perdere parte della lista durante il procedimento
- Si procede in tre fasi

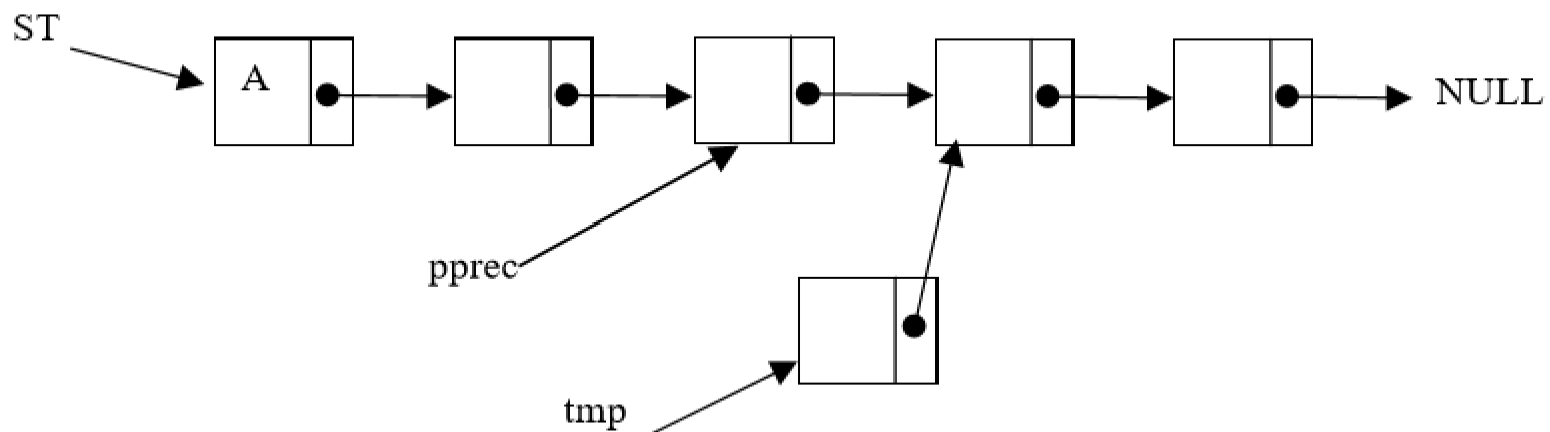
Inserimento in lista concatenata semplice, catena semplice – fase1

- Supponiamo di avere una lista e un nuovo elemento da inserire al suo interno (il uovo elemento è una lista di un solo elemento, puntata da tmp)
- Si deve avere un riferimento (puntatore) all'elemento successivamente al quale si vuole effettuare l'inserimento (pprec in figura)



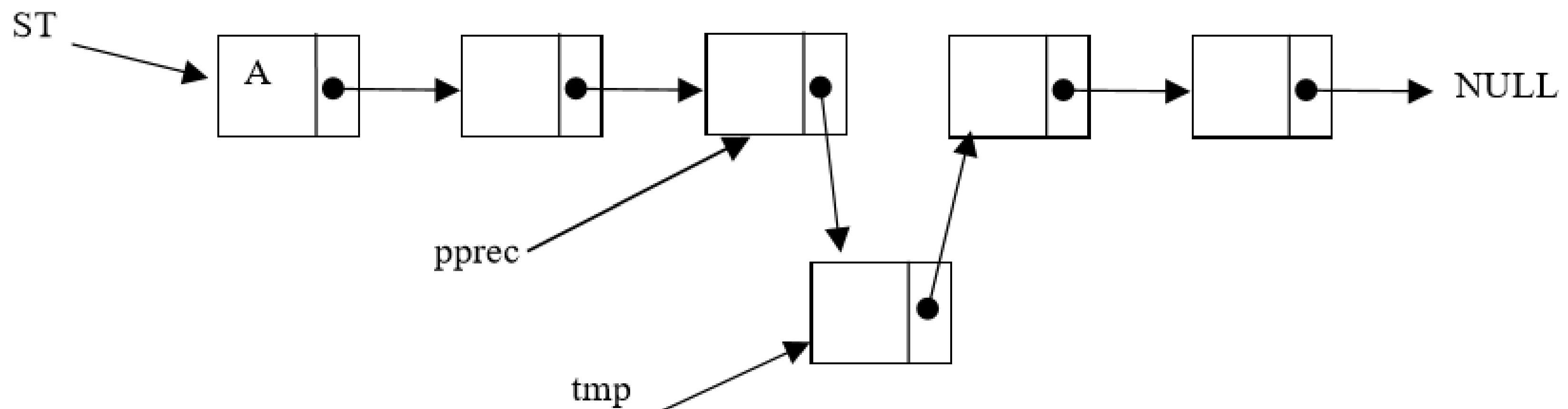
Inserimento in lista concatenata semplice, catena semplice – fase2

- L'elemento da inserire viene congiunto alla seconda parte della lista



Inserimento in lista concatenata semplice, catena semplice – fase3

- L'elemento precedente viene congiunto all'elemento da inserire assegnando il valore di tmp a valore del puntatore al successivo di pprec
- A questo punto 'operazione è conclusa e i due puntatori pprec e tmp non sono più necessari



La struttura dei programmi

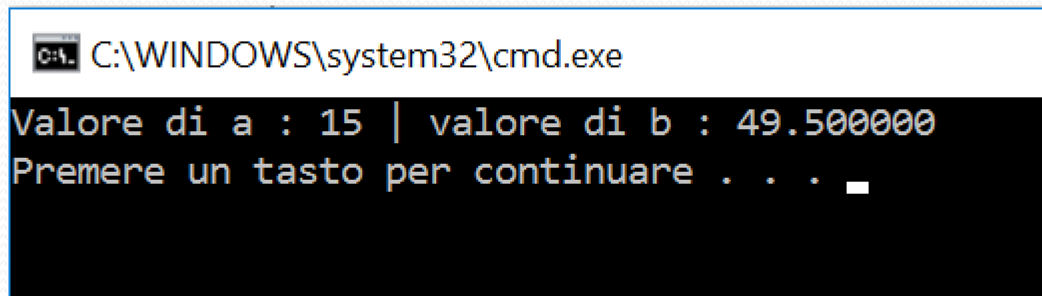
Programma (1)

```
...
main(){
      int a;
      float b;
      a = 15;
      b = a*3.3;
      printf(“Valore di a: %d | valore di b: %f”, a,b);
}
```

- Il costrutto void main (void) marca l’inizio del programma
- Il programma vero e proprio inizia con la prima parentesi graffa e termina con la corrispondente parentesi graffa chiusa
- All’interno del programma vi sono le varie istruzioni che lo compongono

Programma (2)

- Il programma può essere compilato ed eseguito, provocando la seguente stampa a schermo:



- Nel C alla stampa di una variabile a schermo per visualizzarne il valore, corrisponde sempre una conversione di formato
- Esiste una notazione per definire la conversione di formato quando vengono usate funzioni del tipo printf(), fprintf()

Programma (3)

- Il simbolo di percentuale identifica il convertitore di formato nella stringa di stampa
- E' possibile stampare numeri in varie forme e formati ma anche caratteri
- E' infatti fondamentale stampare a schermo non solo numeri ma anche commenti

Descrittori di formato

Tabella dei descrittori di formato

T	Argomento	convertito in
`d, i`	`int`	Decimale con segno
`O`	`int`	Ottale senza segno, senza `O`
`x, X`	`int`	Esadecimale senza segno (senza `0x` o `0X`), con `X` si ha `ABCDEF` anziché `abcdef`
`U`	`int`	Decimale senza segno
`C`	`int`	carattere (conversione implicita con `unsigned char`)
`C`	`unsigned char`	carattere
`S`	`char*`	stringa di caratteri fino al prossimo `\0`
`F`	`double`	numero reale nella forma *mmm.ddd*
`F`	`float`	numero reale nella forma *mmm.ddd*
`e, E`	`double`	numero reale nella forma scientifica *m.ddde±xx*, *m.dddE±xx*
`e, E`	`float`	numero reale nella forma scientifica *m.ddde±xx, m.dddE±xx*
`g, G`	`double`	come `%e,%E` se l'esponente e' minore di -4 altrimenti come `%f`
`g, G`	`float`	come `%e,%E` se l'esponente e' minore di -4 altrimenti come `%f`
`P`	`void*`	puntatore `far`
`N`	`int*`	puntatore `near` (senza segmento)
`%`		carattere `%`

Commenti multilinea e monolinea (1)

- Durante la stesura dei programmi è buona regola inserire commenti in linguaggio naturale per descrivere le istruzioni stesse del programma
- I commenti possono essere inseriti direttamente nel codice MA devono essere delimitati da opportuni marcatori in modo che il compilatore possa distinguere i commenti dalle istruzioni
- I commenti NON hanno nessun effetto sul programma, ovvero non portano a nessuna traduzione in linguaggio macchina, vengono direttamente IGNORATI dal compilatore
- I commenti possono essere:
  - Monolinea
  - Multilinea

Commenti multilinea e monolinea (2)

- I commenti monolinea iniziano con un marcatore ('//') e finiscono con la fine della linea di testo nel programma

```
int a; //Questo è un commento su una linea
```

- I commenti multilinea possono iniziare in un qualsiasi punto del programma con un marcatore di inizio commento ('/*') e terminano con un marcatore di fine commento ('*/')
- Questi commenti possono racchiudere anche commenti di linea e quindi hanno una maggiore PRIORITA' nei loro riguardi

```
/* calcolo il nuovo valore di a
a = a*3.4; //moltiplicazione per…
*/
```

Le funzioni di libreria (1)

- Per poter usare le funzioni di libreria vi è spesso la necessità di includere nel proprio programma un file di definizioni correlate alla libreria che si intende usare
- Tale inclusione deve essere dichiarata in modo esplicito all’interno del programma con una istruzione (‘#include’) che viene interpretata dal preprocessore del compilatore C
- I file contenenti le definizioni della libreria sono detti file header (intestazione) e possono avere estensione h, hxx, hpp

Le funzioni di libreria (2)

- I file di libreria tipicamente contengono:
  - Definizione di tipi
  - Prototipi di funzioni (sottoprogrammi)
  - Costanti simboliche per gestire le funzioni della libreria
- Tali file possono contenere anche sottoprogrammi o comunque istruzioni (da evitare per rendere il programma il più comprensibile possibile e le procedure usabili)

Includere funzioni di libreria (1)

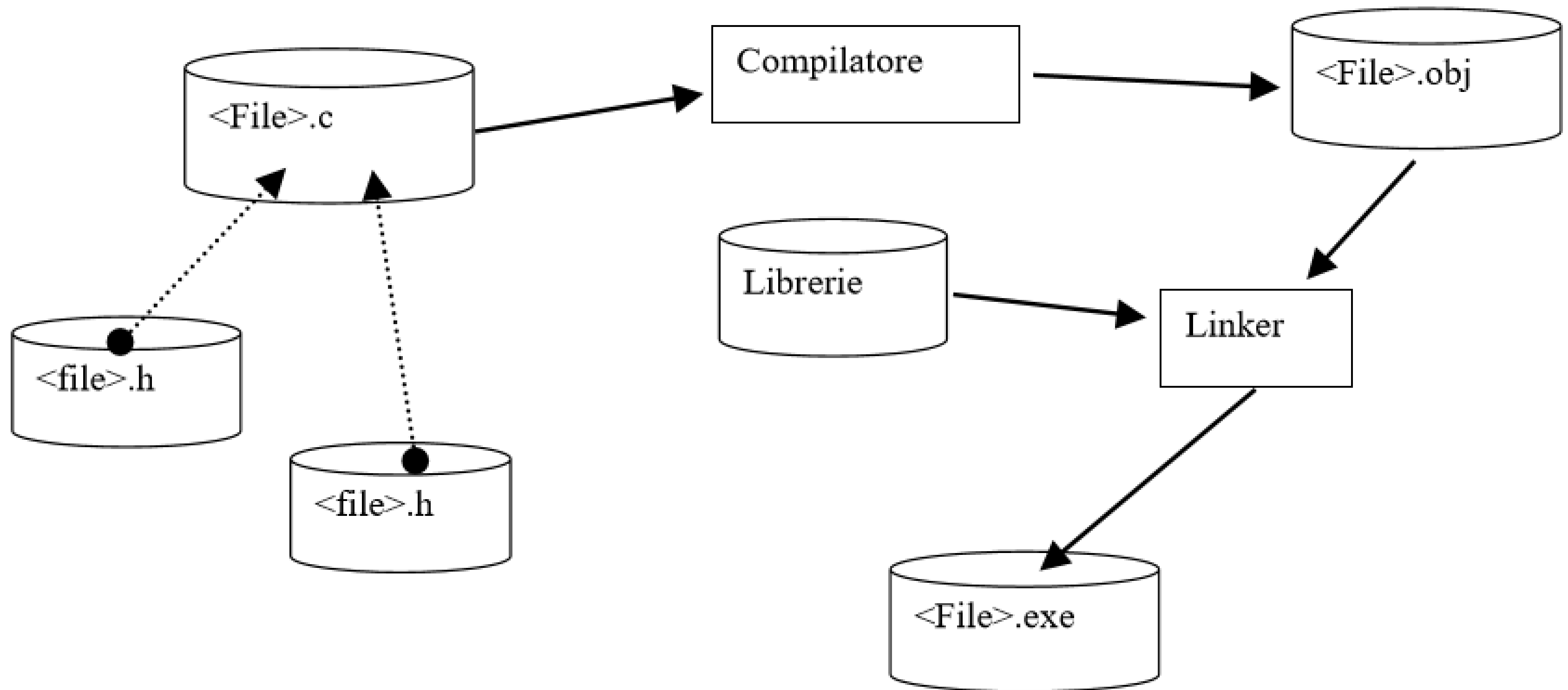
- Con le seguenti istruzioni si possono includere i file di dichiarazioni (header file) delle librerie più importanti:
- #include <stdlib.h> //libreria standard
- #include <stdio.h> //libreria di ingresso/uscita
- #include <string.h> //libreria manipolazione stringhe
- #include <math.h> //libreria funzioni matematiche
- La loro inclusione si può effettuare in qualunque punto del programma MA solitamente si inserisce nella parte iniziale, in modo da evitare conflitti e confusione nella stesura del programma stesso
- L'inclusione dei file dichiarati viene effettuata durante la fase di preprocessazione del programma

Includere funzioni di libreria (2)

- La fase di preprocessazione è una fase preliminare alla compilazione ed è direttamente effettuata dal compilatore stesso
- In seguito il programmatore lancia la compilazione
- Il preprocessore include i file dichiarati con '#include' direttamente nel file che processa
- Il file con gli header inclusi, viene in seguito compilato dal compilatore vero e proprio
- Dopo la fase di compilazione si passa alla fase di collegamento/link, durante la quale i vari moduli del programma e le librerie vengono collegati per formare il programma eseguibile. In questa fase vengono collegate le librerie dichiarate al programma realizzato

Includere funzioni di libreria (3)

- Per il programmatore è possibile creare delle librerie e dei corrispondenti file header da usare in altri programmi



Funzioni di libreria (1)

- Nella tabella che segue sono riportate le principali funzioni di libreria tipicamente messe a disposizione dalla maggior parte dei compilatori C per:
  - Effettuare conversioni di tipo
  - Calcolare funzioni matematiche
  - Manipolare stringhe

- Le funzioni di libreria sono disponibili per essere usate solo DOPO aver incluso la loro dichiarazione di prototipo

Funzioni di libreria (2)

Funzione	*Descrizione*
double asin(double)	calcolo della funzione seno
double atan(double)	calcolo dell'arcotangente
double cos(double)	calcolo della funzione coseno
double exp(double)	calcolo dell'esponenziale
double log(double)	calcolo del logaritmo
double pow(double)	calcolo del quadrato di un numero
double sqrt(double)	calcolo della radice quadrata
double tan(double)	calcolo della tangente
float atof(char *)	conversione da stringa a float
float fabs(float)	calcolo del valore assoluto
float rand (float)	generazione di un numero casuale
int atoi(char *)	conversione da stringa a int
int sizeof(<type>)	stima in byte delle dimensioni di un tipo
int strlen(char *)	stima in byte della lunghezza di una stringa
long atol(char *)	conversione da stringa a long
int rand(void)	Generazione di numeri casuali
char * strcpy(char *, char *)	Copia di stringhe
char * strcmp(char *, char *)	Confronto di stringhe
char * strcat(char *, char *)	Concatenazione di stringhe
char * sprintf()	Conversione da dato numerico a stringa

La struttura dei programmi (1)

- La struttura dei programmi in C può essere di vario tipo
- Generalmente i programmi sono composti da un certo numero di file distinti contenenti: istruzioni, dichiarazioni, definizioni, etc.
- La seguente struttura per i programmi ne facilita la lettura ma NON è l'unica possibile:

```
Include di file header di librerie
Definizioni di tipi
Prototipi di funzioni
Dichiarazioni di variabili globali //da evitare
Dichiarazione di variabili di altri moduli
main (){ … //programma principale
}
Procedure e metodi
```

La struttura dei programmi (2)

- Si noti che SE i programmi sono di una certa dimensione (anche se di pochi Kbytes) è consigliabile dividere il programma sorgente in vari file

Esercizi

- Realizzare un programma che permetta di calcolare e stampare a schermo il risultato delle seguenti operazioni, dati:
  - a = 34.4 , b = 19, c =10, d =9
- Operazioni:

$$a^b \sqrt{c + 23 - d}$$

$$2^c \sqrt{c - d} + e^{(a+b)}$$

Sin(a+b) + b c

Array monodimensionali

- Quando un array è monodimensionale si parla di vettore
- Esempi in C:

```
#define DIMENSIONE 3
…
int a[4];
float b[7];
double ef[DIMENSIONE], d;
char dop[DIMENSIONE];
```

Inizializzazione degli array (1)

- Anche gli array possono essere inizializzati come variabili singole, in tal caso si deve usare la sintassi:

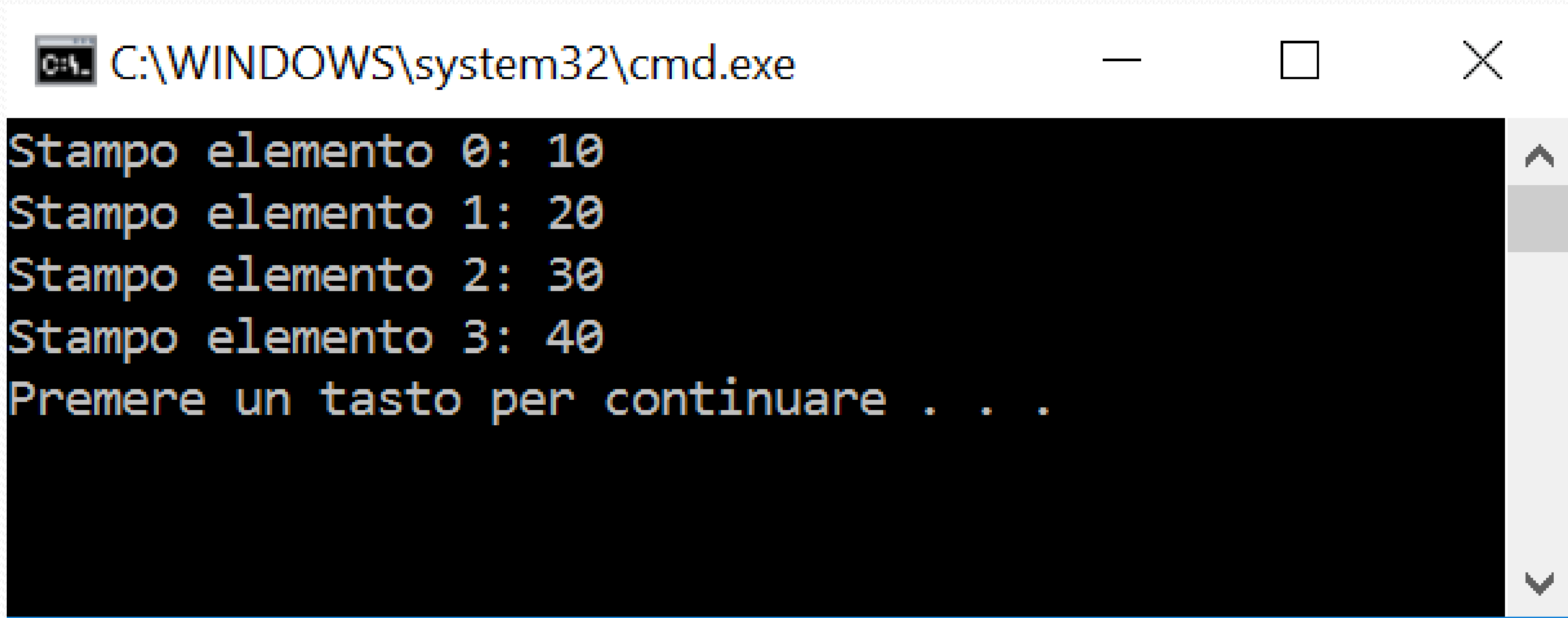
```
int a[4] = {1, 2, 3, 5};
float b[7] = { 34.0, 67.0, 45.4, 56E-2, 45.0, 34, 78 };
double ef[DIMENSIONE] = {10.3, 45.5, 55333332222.4 } , d;
```

- ESEMPIO:

```
main() {
    int i, sum;
    int A[4] = {10,20,30,40};
    for (i = 0; i < 4; i++) {//metto i primi 10 numeri pari
        printf("Stampo elemento %d: %d\n", i, A[i]);
    }
}
```

Inizializzazione degli array (2)

- Si ottiene:



Inizializzazione degli array (2)

- Le inizializzazioni viste prima, devono avere un numero congruente di elementi rispetto alla dichiarazione della dimensione della varabile array (esempio: a[2] ={12,46})
- E' anche possibile definire array in modo che acquistino direttamente la dimensione in base al numero di componenti forniti durante la loro inizializzazione

```
int a[] = {1, 2, 3, 5};
float b[] = { 34.0, 67.0, 45.4, 56E-2, 45.0, 34, 78 };
double ef[]= {10.3, 45.5, 553333332222.4 } , d;
```

- Si effettua semplicemente non definendo la dimensione del vettore, ovvero lasciando vuota la dimensione del vettore [ ]
- Tale modalità di dichiarazione non può essere usata quando il vettore NON ha i valori di inizializzazione

Accesso ai singoli elementi degli array

- Per accedere ai singoli elementi dell'array si usa un indice che identifica ogni elemento
- Con A[3] si identifica l'elemento di indice 3 dell'array

```
int k;
……….
a[3] = k * 23 + 7;
b[2] = a[0]*b[1]+23.5;
printf("il secondo elemento di a = %d", a[1]);
```

La stringa come vettore di caratteri

- In C non esiste il tipo elementare stringa, per questo le stringhe vengono realizzate come array monodimensionali di caratteri

```
char buf[20] = "Saluti a tutti";
printf("la stringa %s\n",buf);
printf("il terzo carattere %c\n", buf[2]);
```

- Nell'esempio la stringa buf (vettore di caratteri) è composta da 20 caratteri (con indice da 0 a 19)
- L'inizializzazione della stringa deve essere effettuata con i doppi apici
- L'inizializzazione dei singoli caratteri con i singoli apici
- Il primo carattere è stato inizializzato a 'S'

Terminazione di stringhe, carattere di fine stringa

- In C, l'inizializzazione di una stringa aggiunge dopo i caratteri della stringa assegnata un carattere specifico di fine stringa, il carattere di codice ASCII decimale 0 (spesso specificato con '\0')
- Le due seguenti inizializzazioni sono equivalenti:

```
char buf[ ] = "Saluti";
char buf[ ] = { 'S', 'a', 'l', 'u', 't', 'i', '\0'} ;
```

Usare i debugger per vedere i singoli caratteri…

Esempi_base1 (In debug) - Microsoft Visual Studio — Avvio veloce

File · Modifica · Visualizza · Progetto · Compilazione · Debug · Team · Strumenti · Test · Analizza · Finestra

Debug · x86 · Continua

Processo: [1644] Esempi_base1.exe · Eventi ciclo di vita · Thread: [7800] Thread principale

Main.c

Esempi_base1 · (Ambito globale) · main()

```
5    #define N 10
6    main() {
7
8        char buf[10] = "Saluti !";
9        char buff2[100];
10       printf("la stringa %s\n", buf);
11       printf("il terzo carattere %c\n", buf[2]);
12       scanf("%s", buff2);
13       printf("stampo cosa mi hai scritto: %s\n", buff2);
14
```

99 %

Variabili locali

Nome	Valore	Tipo
buf	0x006ff730 "Saluti !"	char[10]
[0]	83 'S'	char
[1]	97 'a'	char
[2]	108 'l'	char
[3]	117 'u'	char
[4]	116 't'	char
[5]	105 'i'	char
[6]	32 ' '	char
[7]	33 '!'	char
[8]	0 '\0'	char

Caratteri speciali nelle stringhe

Simbolo	Descrizione
\n	Nuova linea con ritorno carrello (a capo)
\t	Tabulazione orizzontale
\v	Tabulazione verticale
\b	Spazio indietro
\r	Ritorno carrello
\f	Cambio linea senza ritorno carrello (a capo)
\a	Beep (suono prodotto dal computer)
\\	La barra \
\?	Il carattere ?
\'	Il carattere '
\"	Il carattere "
\0II	Il carattere con codifica ASCII ottale II
\xHH	Il carattere con codifica ASCII esadecimale HH
\0	Il carattere NULL

Array multidimensionali

- In C si possono definire degli array multidimensionali
- La loro definizione è semplice, con il seguente esempio viene definito un array multidimensionale di interi di dimensioni M*N*O:

int mat[M][N][O];

- Dove M,N,O sono variabili simboliche specificate per mezzo di istruzioni del preprocessore #define
- Dato un array tridimensionale mat si puo' accedere alle sue componenti semplicemente conoscendone le sue coordinate
- Quando gli array sono bidimensionali vengono chiamati matrici

Matrici

- Esempio

```
float Mat[2][4] = { { 1.0,  2.0,  3.0,  5.0},
                    {11.0, 12.0, 13.0, 15.0} };
```

- Mat[0][1] identifica l'elemento della prima riga e della seconda colonna (anche per le matrici gli indici partono da 0)

Costrutto di selezione semplice

- In C, la selezione semplice con o senza alternativa si esprime sintatticamente nel modo seguente:

```
if(<guardia>) {<corpo1>}
[else {<corpo2>}]
```

- Esempio:

```
int a, b;
a=45;
…
if(a<60){
	b= 60*a;
}
else{
	b=0;
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Concatenazione di costrutti di selezione semplice

```
If(<guardia1>) {<corpo1>}
[else if(<guardia2>){<corpo2>}]
[else if(<guardia3>){<corpo3>}]
…
[else {<corpoN>}]
ESEMPIO:
..
if(a==1){
	b=1;
}
else if (a>1){
	b= 5;
}
else
	b=0;
```

UNIVERSITÀ DEGLI STUDI FIRENZE DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Costrutti iterativi

```
for(condizione_inizio_ciclo; GUARDIA; Incremento){
        CORPO
}

while (GUARDIA){
CORPO
}

do {
        CORPO
}
while (GUARDIA)
```

I costrutti for annidati

- I costrutti for possono essere annidati
- Il massimo numero di annidamenti possibili dipende dal compilatore
- In certi casi di cicli annidati l'esecuzione di una istruzione break comporta l'uscita dal ciclo corrente e NON da tutti

- Cicli for annidati possono ad esempio servire per effettuare calcoli sulle matrici

Esempio: cicli for annidati e matrici – stampa elementi

```
#define NR 3
#define NC 4
main() {
    int i, j;
    //siano: NR= numero righe e NC=numero colonne
    int matrix[NR][NC] = { { 23,54,6,3 },{ 2, 0,13, 44 },
                                    { 91,539,0,12 } };
    for (i = 0; i < NR; i++) {//ciclo sulle righe
      for (j = 0; j < NC; j++) {//ciclo sulle colonne
          printf("Elemento in riga %d e colonna %d, che vale: %d -
             d\n", i+1, j+1, matrix[i][j]);
    }
}
printf("Fine visita della matrice");
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Esempio: cicli for annidati e matrici (1) calcolo somma elementi

```
...
#define NR 3
#define NC 4
main() {
    int i, j, sum;
    sum = 0;
    //siano: NR= numero righe e NC=numero colonne
    int matrix[NR][NC] = { { 23,54,6,3 },{ 2, 0,13, 44 },
                                          { 91,539,0,12 } };
    for (i = 0; i < NR; i++) {//ciclo sulle righe
      for (j = 0; j < NC; j++) {//ciclo sulle colonne
          sum = sum + matrix[i][j];
          printf("Elemento in riga %d e colonna %d, che vale: %d -
               Somma parziale: %d\n", i+1, j+1, matrix[i][j], sum);
    }
}
printf("Somma totale %d \n", sum);
```

Esempio: cicli for annidati e matrici (2)

C:\WINDOWS\system32\cmd.exe

```
Elemento in riga 1 e colonna 1, che vale: 23 - Somma parziale: 23
Elemento in riga 1 e colonna 2, che vale: 54 - Somma parziale: 77
Elemento in riga 1 e colonna 3, che vale: 6 - Somma parziale: 83
Elemento in riga 1 e colonna 4, che vale: 3 - Somma parziale: 86
Elemento in riga 2 e colonna 1, che vale: 2 - Somma parziale: 88
Elemento in riga 2 e colonna 2, che vale: 0 - Somma parziale: 88
Elemento in riga 2 e colonna 3, che vale: 13 - Somma parziale: 101
Elemento in riga 2 e colonna 4, che vale: 44 - Somma parziale: 145
Elemento in riga 3 e colonna 1, che vale: 91 - Somma parziale: 236
Elemento in riga 3 e colonna 2, che vale: 539 - Somma parziale: 775
Elemento in riga 3 e colonna 3, che vale: 0 - Somma parziale: 775
Elemento in riga 3 e colonna 4, che vale: 12 - Somma parziale: 787
Somma totale 787
Premere un tasto per continuare . . .
```

Esempio: ricerca del valore massimo degli elementi di una matrice (1)

```
…
#define NR 3
#define NC 4
main() {
    //ricerca valore max elementi di una matrice
    int i, j, max =0, imax=0, jmax=0;
    //siano: NR= numero righe e NC=numero colonne
    int matrix[NR][NC] = { { 23,54,6,3 },{ 2, 0,13, 44 },{ 91,539,0,12 } };
    for (i = 0; i < NR; i++) {//ciclo sulle righe
      for (j = 0; j < NC; j++) {//ciclo sulle colonne
          if (max < matrix[i][j]) {
              max = matrix[i][j];
              imax = i+1;
              jmax = j+1;
      }
    }
}
printf("L'elemento della matrice con valore massimo,\nsi trova alla riga
%d e alla colonna %d,\ned ha valore: %d \n", imax, jmax, max);
}
```

Esempio: ricerca del valore massimo degli elementi di una matrice (2)

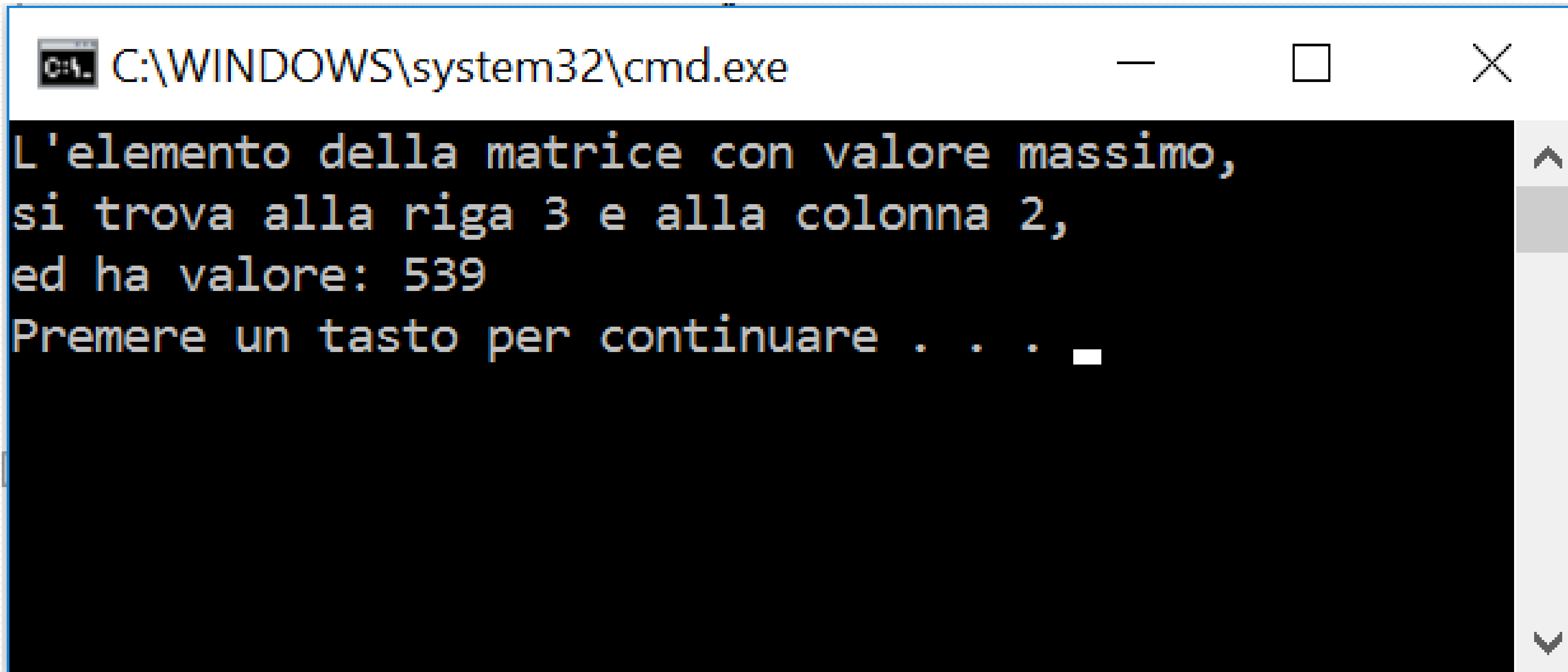
- Il programma itera su tutti gli elementi della matrice
- Per ogni elemento effettua il confronto con il valore della variabile max
- La variabile max viene inizializzata a 0 (in base anche al tipo di variabili usate e alla inizializzazione della matrice)
- Ogni volta che: max<matrix[i][j] il valore di max viene aggiornato e vengono memorizzate la riga e la colonna del max (si potrebbe fare la stessa anche con gli indici)

Esempio: ricerca del valore massimo degli elementi di una matrice ()

- Avendo inizializzato la matrice nel modo seguente:

```
int matrix[NR][NC] = { {23,54,6,3},{2,0,13,44},{91,539,0,12} };
```

- In console si vede il seguente risultato:



Calcolo del valor medio degli elementi in una matrice (1)

- Il valor medio degli elementi di un insieme di dati viene calcolato facendo la somma degli elementi e dividendo il risultato per il numero stesso degli elementi

- Il seguente programma effettua il calcolo del valor medio degli elementi di una matrice

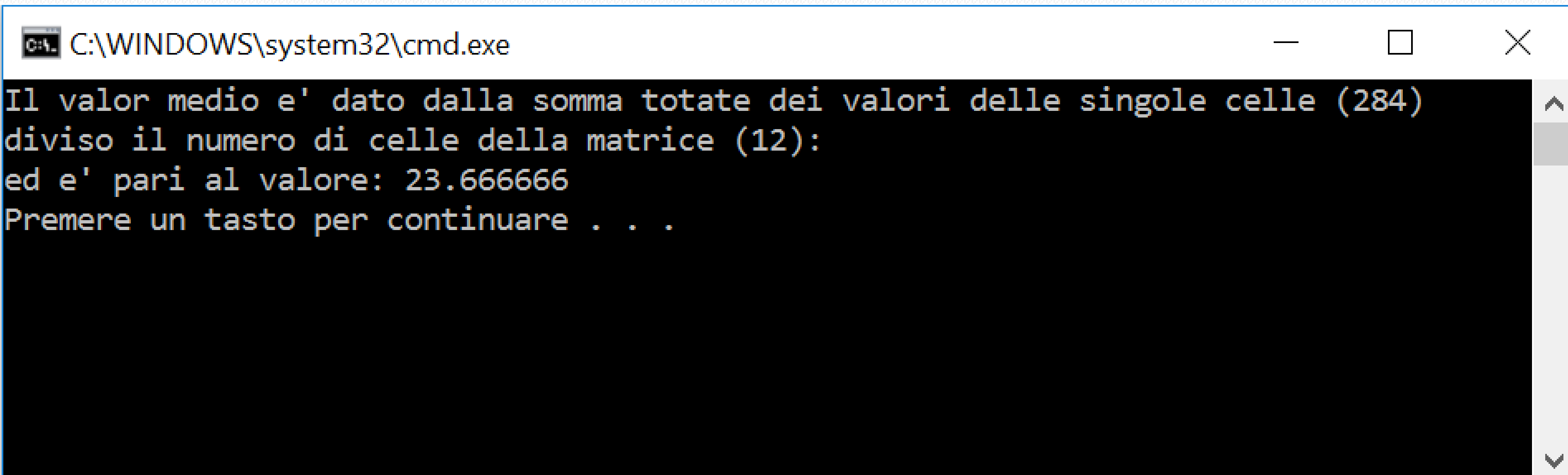
- Il programma itera su tutti gli elementi della matrice per calcolare la somma degli elementi

Valor medio degli elementi in una matrice (2)

```
...
main() {   //con #define NR 3   e #define NC 4
    int i, j, sum=0;
    float med=0;
    //siano: NR= numero righe e NC=numero colonne
    int matrix[NR][NC]={ { 23, 53,6,3 },{ 2, 0,13, 44 },{ 91,37,0,12 } };
    for (i = 0; i < NR; i++) {//ciclo sulle righe
       for (j = 0; j < NC; j++) {//ciclo sulle colonne
           sum = sum + matrix[i][j];
       }
    }
    med = (float)sum/(NR*NC);
    printf("Il valor medio e' dato dalla somma totate dei valori delle
    singole celle (%d)\ndiviso il numero di celle della matrice
    (%d):\ned e' pari al valore: %f\n", sum, NR*NC, med);
}
```

Valor medio degli elementi in una matrice (3)

- Si noti che per ottenere un valor medio preciso è stata usata una variabile di tipo float



Costrutti di selezione composta: Switch, default, break (1)

- Quando ci sono varie possibili alternative si può ricorrere alla selezione composta con e senza alternativa
- Questa struttura di controllo permette di individuare le istruzioni da eseguire in base a differenti valori assunti da una espressione
- La selezione composta viene introdotta dalla parola chiave **switch** seguita dalla **espressione di guardia** da valutare tra parentesi e da altre parole chiave quali **case, break, default**

Costrutti di selezione composta: Switch, default, break (2)

- L’istruzione **break** può apparire solo all’interno del corpo di una ***istruzione di iterazione*** o di uno **switch**
- L’effetto del **break** è quello di trasferire il controllo alla prima istruzione successiva alla istruzione di iterazione o di switch nella quale è contenuta

Costrutti di selezione composta: Switch, default, break (2)

```
switch(espressione di guardia){
        case valore1:
                corpo1;
                break;
        case valore2:
                corpo2;
                break;
        …
        default :
                corpo di default;
                break;
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Costrutti di selezione composta: switch, default, break (3)

```
….
switch(a+5){
        case 6:
                b=10;
                break;
        case 9:
                b=20;
                break;
        default:
                b=30;
}
after_switch_body;
```

Costrutto goto (salto)

- Il goto, serve per effettuare un ‘salto’ (poco usato in C)

```
….
switch(a+5){
        case 6:
                b=10;
                goto after_switch_body;
        case 9:
                b=20;
                goto after_switch_body;
        default:
                b=30;
}
after_switch_body;
```

Le istruzioni break e continue per i costrutti iterativi (1)

- In C è possibile uscire dal ciclo for a prescindere dalla condizione di ciclo grazie alla istruzione **break**
- Esiste inoltre la possibilità di interrompere l'esecuzione dell'istruzione interna (se è una istruzione composta) e di saltare direttamente al punto in cui viene fatto l'aggiornamento del valore dell'indice dell'iterazione e del successivo controllo della iterazione, tramite l'istruzione **continue**

Istruzioni break e continue nei costrutti iterativi (2)

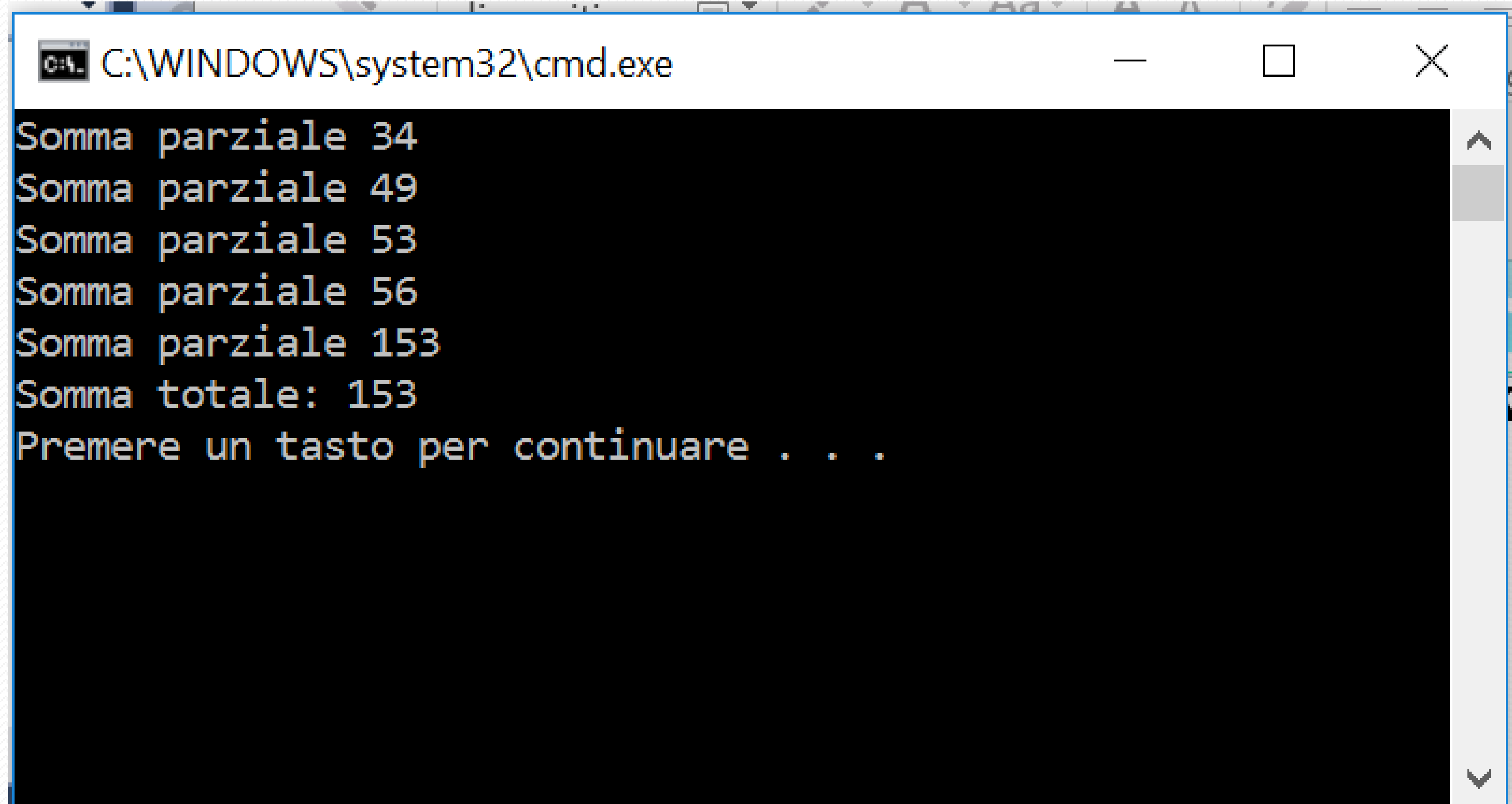
- Ogni ciclo può essere interrotto anche nel corso di esecuzione con due modalità: break e continue
- **break**:
  - Interrompe prematuramente il ciclo in esecuzione
  - Se si hanno cicli annidati comporta l'uscita dal ciclo in cui si esegue ma non da tutti i cicli
- **continue**:
  - Nel ciclo for interrompe l'esecuzione del ciclo e salta al punto in cui il valore dell'indice dell'iterazione viene aggiornato
  - Nel ciclo while interrompe l'esecuzione del ciclo e salta direttamente in testa, dove viene valutata l'espressione
  - Nel ciclo do-while interrompe l'esecuzione del ciclo e passa a valutare l'espressione
  - SE si hanno più cicli annidati, viene applicato SOLO al ciclo in cui viene eseguito

Uso del costrutto break in un ciclo for (1)

```
void main(void){
    int av[10] = { 34, 15, 4, 3, 97, 42, 4, 7, 12, 0 };
    int sum, i;
    sum = 0;
    for (i = 0; i < 10; i++) {
      sum = sum + av[i];
      printf("Somma parziale %d\n", sum);
      if (sum > 100)
          break;
    }
    printf("Somma totale: %d\n", sum);
}
```

Uso del costrutto break in un ciclo for (2)

Da console…

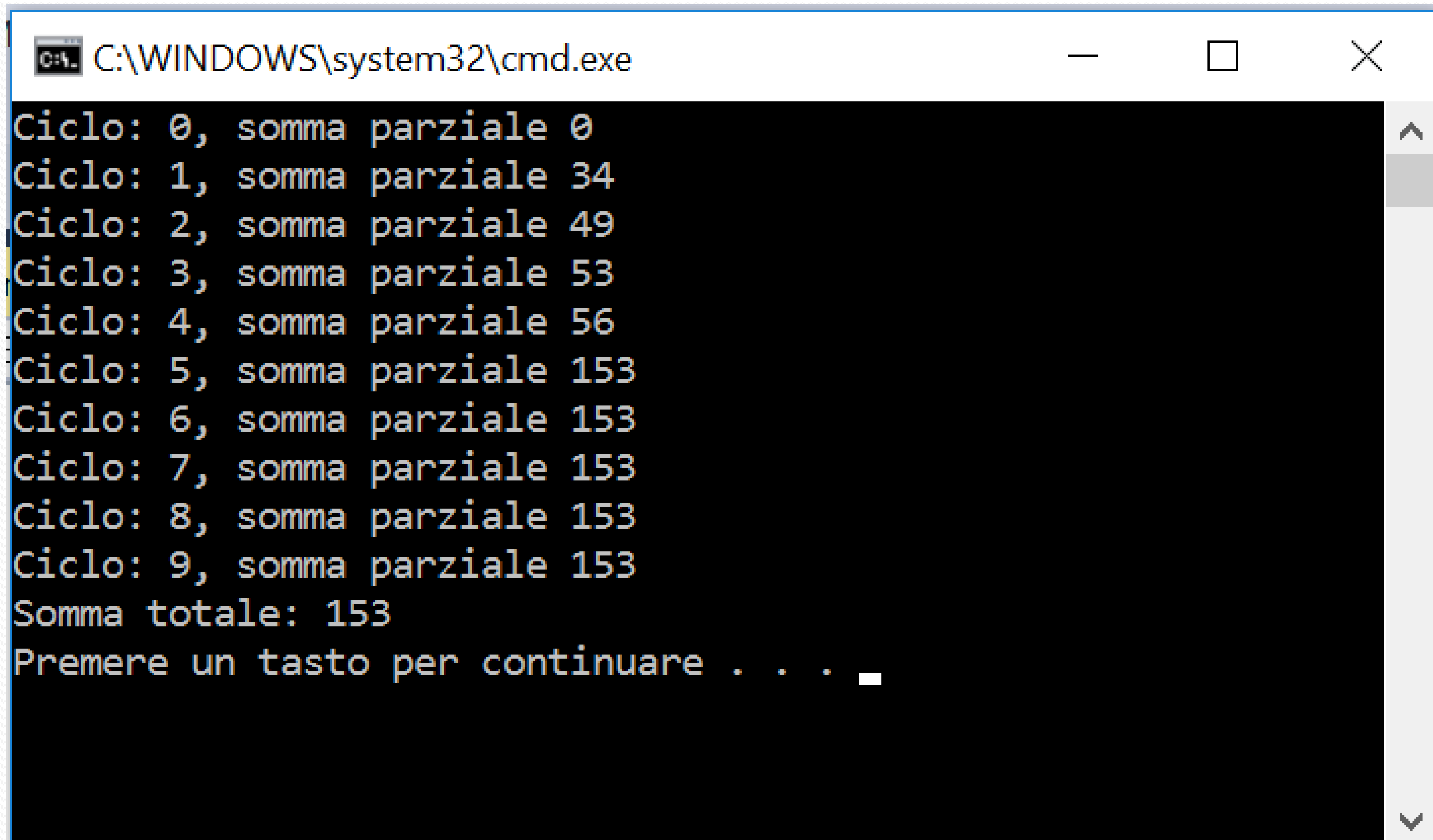


Uso del costrutto continue in un ciclo for (1)

```
void main(){
    int av[10] = { 34,15,4,3,97,42, 4, 7, 12 };
    int sum, i;
    sum = 0;
    for (i = 0; i < 10; i++) {
      printf("Ciclo: %d, somma parziale %d\n", i+1, sum);
      if (sum < 100) {
          sum = sum + av[i];
          continue;
      }
    }
    printf("Somma totale: %d\n", sum);
}
```

Uso del costrutto continue in un ciclo for (2)

Da console…



Uso di break e continue in un ciclo for (1)

```
…
int main(){
        int sum =0;
        int A[4]={10,20, 30,40};
        for(int i=0; i<4;i++) {
          sum +=  A[i];
          printf("Incremento la somma che ora vale: %d \n\n", sum);
          if(sum<35)
                continue;
          else
                break;
         }
         printf("Alla fine del ciclo for, sum avra' valore: %d\n\n", sum);
}
```

Uso di break e continue in un ciclo for (2)

```
…
int main(){
            int sum =0;
            int A[4]={10,20, 30,40};
            for(int i=0; i<4;i++) {
              sum +=  A[i];
              printf("Incremento la somma che ora vale: %d \n\n", sum);
              if(sum<35)
                      continue;
              else
                      break;
             }
           printf("Alla fine del ciclo for, sum avra' valore: %d\n\n", sum);
}
```

```
Incremento la somma che ora vale: 10

Incremento la somma che ora vale: 30

Incremento la somma che ora vale: 60

Alla fine del ciclo for, sum avra' valore: 60
```

Uso di break e continue in un ciclo for (2)

```
…
int main(){
        int sum =0;
        int A[4]={10,20, 30,40};
        for(int i=0; i<4;i++) {
          sum +=  A[i];
          printf("Incremento la somma che ora vale: %d \n\n", sum);
          if(sum<35)
                break;   //invertiamo break e continue
          else
                continue;
         }
        printf("Alla fine del ciclo for, sum avra' valore: %d\n\n", sum);
}
```

Uso di break e continue in un ciclo for (4)

```
…
int main(){
        int sum =0;
        int A[4]={10,20, 30,40};
        for(int i=0; i<4;i++) {
          sum +=  A[i];
          printf("Incremento la somma che ora vale: %d \n\n", sum);
          if(sum<35)
                break;    //invertiamo break e cotinue
          else
                continue;
         }
        printf("Alla fine del ciclo for, sum avra' valore: %d\n\n", sum);
}
```

```
Incremento la somma che ora vale: 10

Alla fine del ciclo for, sum avra' valore: 10
```

I puntatori e vettori (1)

- I puntatori sono delle variabili che mantengono il valore dell'indirizzo di altre variabili o di altri puntatori
- Le dimensioni di una variabile puntatore e quindi la sua rappresentazione in memoria dipendono dal sistema operativo e da altri aspetti (microprocessore, linguaggio usato, compilatore)
- In C i puntatori sono di fondamentale importanza
- Esistono anche linguaggi che non hanno puntatori e che risultano però meno flessibili rispetto a quelli che ne permettono l'utilizzo

I puntatori e operatori relativi: &, * (1)

- I puntatori sono tipizzati, questo permette di identificare il tipo di variabile ai quali puntano
- La loro rappresentazione in memoria non dipende dal tipo di variabile alla quale fanno riferimento
- La tipizzazione dei puntatori in C è utile quando si effettuano dei calcoli con il loro valore
- In C è possibile dichiarare dei puntatori ‘neutri’, cioè NON tipizzati
- La **dichiarazione** dei **puntatori** si effettua semplicemente anteponendo un asterisco al nome della variabile in fase di dichiarazione:

```
int *a, b=5;    //dichiarazione del puntatore a e della
                //variabile b
```

I puntatori e operatori relativi: &, * (2)

- Data la dichiarazione:

```
int *a, b=5;
```

- Le due variabili a e b NON sono dello stesso tipo:
  - a è un puntatore ad un intero
  - b è un intero
- Pertanto NON è possibile assegnare il valore di b ad a e viceversa
- Dopo la dichiarazione di un puntatore il suo valore non ha molto senso perché non indica/referenzia una specifica variabile, quindi i puntatori NON devono essere usati senza aver INIZIALIZZATO in modo opportuno il loro valore
- Per convenzione il valore iniziale di un puntatore è NULL. NULL è una costante simbolica pari a 0

I puntatori e operatori relativi: &, * (3)

- Data la dichiarazione:

  int *a, b=5;

- Per convenzione il valore iniziale di un puntatore è NULL. NULL è una costante simbolica pari a 0

- Data una variabile è possibile risalire al suo indirizzo tramite l'operatore unario **&.** Per esempio data la variabile intera b SE si scrive **&b**, si intende fare riferimento all'indirizzo della variabile b. Quindi si può anche scrivere la seguente assegnazione:

  a = &b;

I puntatori e operatori relativi: &, * (4)

- Dato:

```
int *a, b=5;
a = &b;
```

- Dopo l'assegnazione il puntatore a punta alla cella b
- Il puntatore è rappresentato con una freccia
- Se un puntatore punta ad una cella, può essere usato per manipolare il contenuto di tale cella
- Per esempio con *a si fa riferimento alla cella puntata da a (sarebbe: 'ciò a cui punta a')

```
int *a, b = 5, c = 2;
a = &b;
c = *a * c;
printf("%d, %d, %d\n", *a, b, c);
```

C:\WINDOWS\system32\cmd.exe

```
5, 5, 10
Premere un tasto per continuare . . .
```

I puntatori e operatori relativi: &, * (5)

```
int *a, b = 5, c = 2;
a = &b;
c = *a * c;
printf("Valore a cui punta il puntatore (a interi) a: %d\nValore della variabile b: %d\nValore di c: %d\n", *a, b, c);
printf("Valore di a: %X, Indirizzo di b (&b): %X\n\n",a, &b);
```

C:\WINDOWS\system32\cmd.exe

```
Valore a cui punta il puntatore (a interi) a: 5
Valore della variabile b: 5
Valore di c: 10

Valore di a: C4F9C0 Indirizzo di b (&b): C4F9C0
```

C:\WINDOWS\system32\cmd.exe

```
Valore a cui punta il puntatore (a interi) a: 5
Valore della variabile b: 5
Valore di c: 10

Valore di a: 12FFCA8 Indirizzo di b (&b): 12FFCA8
```

Array di puntatori (1)

- Si possono definire anche array di puntatori sia monodimensionali che multidimensionali
- Anche in questo caso, nelle espressioni i vari operatori relativi ai puntatori (* e &) vengono applicati in base a delle regole specifiche di precedenza
- Quando NON è possibile specificare esattamente il tipo di puntatore si può usare il tipo void che rappresenta un tipo generico di puntatore. Al momento dell'uso, questo puntatore può essere convertito nel tipo corretto mediante una operazione di casting esplicito
- Definizione di vettori di puntatori:

  ```
  int  *p[10]; //vettore di 10 puntatori ad interi
  ```
- NON confondere con:

  ```
  int (*p)[10]; //puntatore p ad un vettore di 10 interi
  ```

Puntatori e stringhe

- Gli operatori * e & per la gestione di puntatori, sono molto utili per la gestione di vettori e quindi anche di stringhe che sono vettori di caratteri
- Le seguenti dichiarazioni di stringhe sono equivalenti:

```
char buf[] = "Giovanni";
char *buf = "Giovanni";
```

- In entrambi i casi:
  - la stringa buf viene realizzata come un vettore di 8 caratteri+ il carattere di fine stringa
  - Si può fare riferimento ai singoli caratteri per mezzo dell’indice
  - Il puntatore al primo carattere, quello di indice 0, può essere indicato con: &buf[0] o con buf (più usata). In C buf rappresenta anche l’intera stringa e non solo il suo puntatore

Il concetto di sottoprogramma (1)

- Comunemente la realizzazione di un algoritmo prevede l'uso di algoritmi più semplici
- Gli algoritmi più semplici spesso costituiscono porzioni che possono essere riutilizzate più volte nello stesso o in altri contesti
- Questo meccanismo può essere usato per suddividere l'algoritmo iniziale in un insieme di algoritmi più semplici che, una volta risolti, possono portare alla soluzione del problema completo iniziale
- E' quindi necessario un metodo per la definizione e l'aggregazione delle istruzioni: blocchi o sequenze, procedure e funzioni
- Questo meccanismo si realizza con la formalizzazione di sottoprogrammi

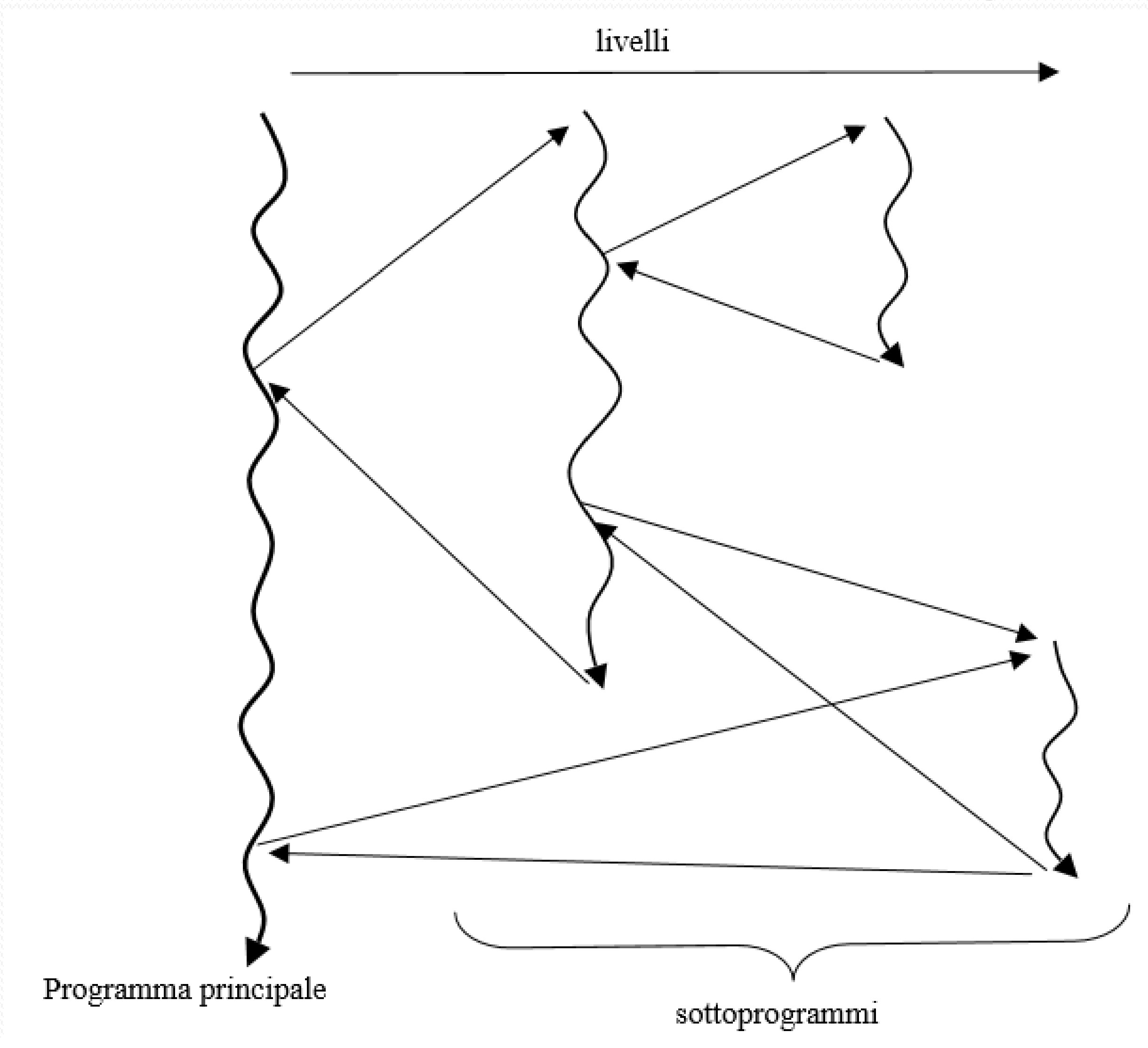
Il concetto di sottoprogramma (2)

- La forma elementare per la suddivisione/raggruppamento di istruzioni è il costrutto sequenza per la realizzazione di un blocco di istruzioni
- Il blocco delimita un gruppo di istruzioni, ma non ha un identificativo che lo contraddistingue
- In C i delimitatori di blocco di istruzioni sono le parentesi graffe {} (concetto di compound)
- A livello intuitivo quindi un programma è un blocco/sequenza di istruzioni identificato da un nome, che viene evocato al momento della chiamata
- A tal punto il programma chiamante trasferisce al sottoprogramma i dati necessari all'elaborazione secondo una convenzione posizionale

Il concetto di sottoprogramma (3)

- I sottoprogrammi si possono classificare in procedure e funzioni
- Sono particolari strutture di controllo che, in seguito alla loro invocazione, **alterano il flusso del programma**
- Il programma principale può usare un sottoprogramma per effettuare determinate operazioni
- Il sottoprogramma a sua volta può richiamare altri sottoprogrammi
- I sottoprogrammi possono essere usati in più punti di un programma
- E' importante che i singoli sottoprogrammi lavorino su variabili proprie o su copie, comunque solo in modo controllato

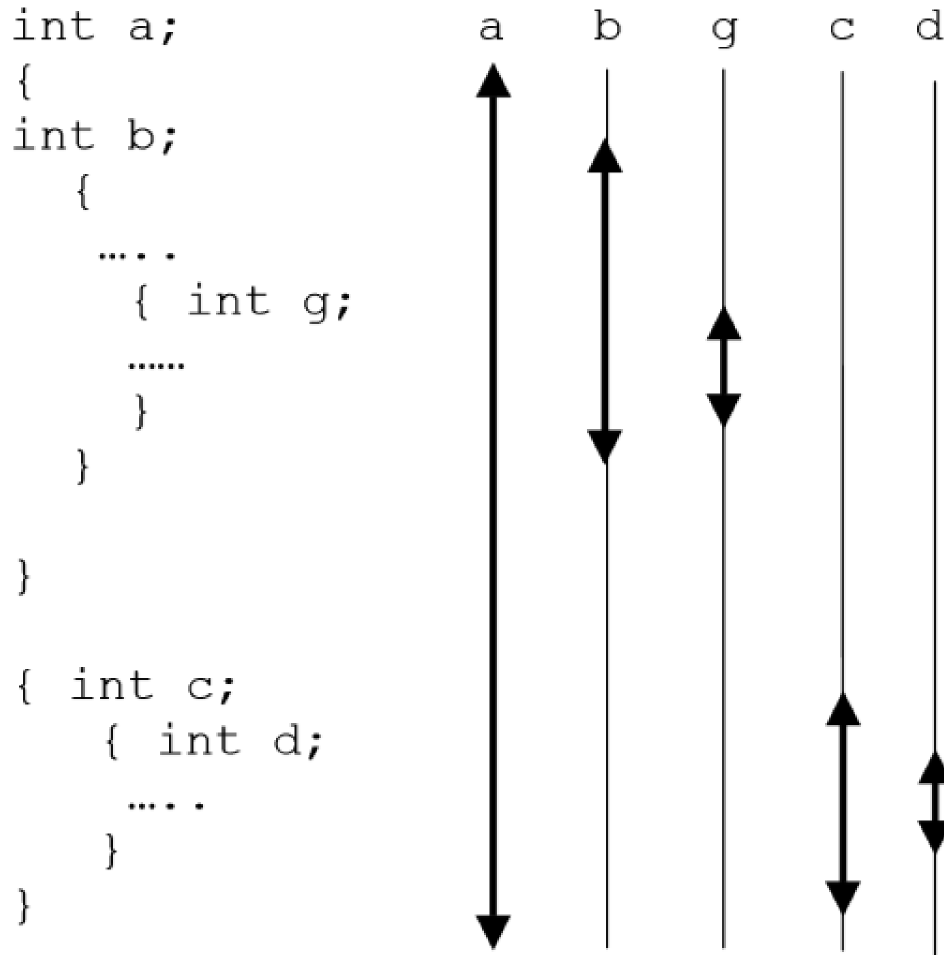
Il concetto di sottoprogramma (4)



Il dominio di validità delle variabili, lo scope (1)

- Ogni sottoprogramma può avere delle variabili e queste variabili hanno validità SOLO nel blocco in cui sono definite e nei blocchi sottostanti SE in questi non sono definite variabili omonime
- Il campo di validità di una variabile è detto scope
- Le variabili sono dichiarate in un certo contesto che è determinato dal costrutto sequenza (ovvero dalla coppia di parentesi che troviamo sia nei sottoprogrammi che nei blocchi annidati di istruzioni…if, while, for, do-while, etc.)

Il dominio di validità delle variabili, lo scope (1)



- La variabile a è visibile per tutto il campo di validità (**variabile globale**)
- Le altre variabili hanno uno scope limitato (**variabili locali**)

Struttura dei sottoprogrammi (1)

- Comunemente un sottoprogramma in C ha le seguente struttura (espressa in EBNF):

<intestazione del sottoprogramma>
{
[<definizione dei tipi locali>]
…
[<dichiarazione di variabili locali>]
[<Istruzioni>]
}

- Un sottoprogramma è costituito da una intestazione seguita dal simbolo { che marca l'inizio del sottoprogramma
- Il sottoprogramma termina con la }
- Nota: in EBNF ciò che è dentro le [ ] può essere omesso

Struttura dei sottoprogrammi (2)

- Un sottoprogramma è costituito da una intestazione seguita dal simbolo { che marca l’inizio del sottoprogramma

- Il sottoprogramma termina con la }

- Dentro il corpo del sottoprogramma vi possono essere definizioni di nuovi tipi, e quindi dichiarazioni di variabili locali e le istruzioni stesse del sottoprogramma

Intestazione dei sottoprogrammi (1)

- L'intestazione contiene la dichiarazione del tipo RESTITUITO dal sottoprogramma (può essere omessa SE coincide con il tipo int)
- Esistono due modalità per realizzare le intestazioni:
  - Caso 1:

  <tipo> <nome funzione> (<nome argomento1>, …, <nome argomentoN>)
  <tipo1> <nome argomento1>; …
  <tipoN> <nome argomentoN>;
  - **Caso2 (la più usata):**

  <tipo> <nome funzione> (<tipo1> <nome argomento1>, …, <tipoN> <nome argomentoN>)

Intestazione dei sottoprogrammi (2)

- Esistono due modalità per realizzare le intestazioni:
  - Caso 1 (**MENO USATO**):

<tipo> <nome funzione> (<nome argomento1>, …, <nome argomentoN>)
<tipo1> <nome argomento1>; …
<tipoN> <nome argomentoN>;

  - Esempio:

```
int n;
int DividiPerDue(n);
```

Intestazione dei sottoprogrammi (3)

- Esistono due modalità per realizzare le intestazioni:
  - **Caso2 (la più usata):**

  <tipo> <nome funzione> (<tipo1> <nome argomento1>, …, <tipoN> <nome argomentoN>)
  - Esempio:

    int DividiPerDue(float a);

Sottoprogrammi: procedure e funzioni

- Un sottoprogramma può essere una procedura o una funzione. Le procedure sono sottoprogrammi che NON hanno un tipo
- In C il nome dato ad un sottoprogramma NON può essere assegnato ad altri sottoprogrammi
- Il sottoprogramma può essere privo di parametri; in tal caso si puo' scrivere la parola chiave del linguaggio void. Questa parola chiave significa vuoto e nella dichiarazione dei sottoprogrammi può essere usata per specificare che:
  - Non ha parametri (in **input**):

    int NomeSottoprogramma(){…}

    int NomeSottoprogramma(void){…}
  - Non ha un tipo (in **output**):

  void NomeSottoprogramma(int a){…}

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Sottoprogrammi: funzioni (1)

- Le funzioni sono particolari sottoprogrammi per i quali è richiesta l'indicazione del **tipo di dato restituito in uscita**, ovvero il tipo di dato che caratterizza la funzione;
  - **int** Somma (int X, int Y); //**dichiarazione della funzione**
  - **void** StampaHelloSuConsole(void){}
- **Le funzioni, essendo tipizzate, possono essere usate come funzioni matematiche in espressioni e quindi a destra del simbolo di assegnazione (=)** o anche in condizioni di costrutti di selezione o iterazione:
  - C = Somma(X,Y); //**chiamata della funzione**
- Le funzioni, come tutti i sottoprogrammi possono essere prive di parametri, ad esempio la funzione rand che produce un numero casuale tra 0 e il massimo della dinamica fra interi ha come **prototipo** (o dichiarazione della funzione): int rand(**void**);

Sottoprogrammi: definizione

- Esempi di **definizione di funzioni**:

```
float DividiPerDue(int a){
        return (float)a/2;
}

void Stampa(){
        printf(“Hello World!”);
}

int Somma (int X, int Y){
        return X+Y;
}
```

Sottoprogrammi: funzioni (2)

- Dato un sottoprogramma, per restituire il valore si usa la parola chiave **return** che chiude la funzione e rimanda il controllo alla istruzione successiva alla chiamata

- Alla fine della esecuzione della funzione il valore assunto dalla funzione è pari al valore usato come parametro della parola chiave **return**

- Nel corpo del sottoprogramma non vi sono limitazioni al numero di chiamate alla funzione **return**()

Sottoprogrammi: funzioni (3)

- Si noti che **prima dell'uso della funzione (o chiamata a funzione) o di un sottoprogramma è necessario riportare il prototipo** della funzione stessa

- Il prototipo di un sottoprogramma è una dichiarazione che specifica i tipi coinvolti nella chiamata e il loro ORDINE:
  - int Somma (int A, int B); //prototipo o dichiarazione di //funzione

- Questa dichiarazione permette al compilatore di effettuare un **controllo di consistenza** fra i tipi delle variabili coinvolte nella chiamata e il sottoprogramma stesso

Sottoprogrammi: funzioni (3)

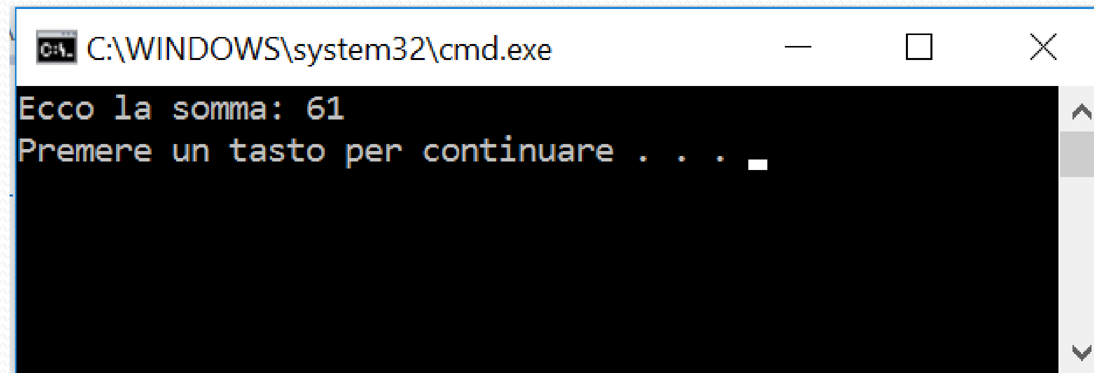
- La porzione di codice che include la chiamata al sottoprogramma si chiama programma chiamante:
  - C = Somma(X,Y); //**chiamata della funzione**

- Al momento della esecuzione del programma, ovvero quando viene invocato il sottoprogramma, l'esecuzione passa a tale porzione di codice e viene effettuato un cambio di contesto

- Nel nuovo contesto vengono assegnate alle variabili temporanee (le variabili che rappresentano i parametri delle funzioni) i valori attuali delle variabili usate nella chiamata

Sottoprogrammi: funzioni (4)

```
… int Somma (int A, int B); //dichiarazione della funzione
main() {
    int C, X=5,Y=56;
    C = Somma(X,Y); //chiamata della funzione
    printf("Ecco la somma: %d\n", C);
    }
int Somma(int A, int B) {//definizione della funzione
    int d;
    d = A + B;
    return d;
}
```



NOTA1: A assume il valore di X e B assume il valore di Y (assegnazione fatta in base alla posizione, ovvero all'ordine con il quale le variabili nella chiamata sono organizzate rispetto alla struttura stessa della intestazione della funzione)

Sottoprogrammi: funzioni (5)

```
… int Somma(int A, int B); ... //dichiarazione
main() {
    int C, X, Y;
    X = 5;
    Y = 56;
    C = Somma(X,Y);          //Somma() usata in una assegnazione
    printf("Ecco la somma: %d\n", C);
}
int Somma(int A, int B) { //definizione
    int d;
    d = A + B;
    return d;
}
```

- NOTA2: per restituire il valore si usa la parola chiave **return** che chiude la funzione e rimanda il controllo alla istruzione successiva alla chiamata

Sottoprogrammi: funzioni (6)

```
… int Somma(int A, int B);
main() {
    int C, X, Y;
    X = 5;
    Y = 56;
    C = Somma(X,Y);        //Somma() usata in una assegnazione
    printf("Ecco la somma: %d\n", C);
}
int Somma(int A, int B) {
    int d;
    d = A + B;
    return d;
}
```

- NOTA 3: Alla fine della esecuzione della funzione, il valore assunto dalla funzione è pari al valore usato come parametro della parola chiave **return (valore assegnato a C nell'esempio)**

Sottoprogrammi: funzioni (7)

- La funzione viene conclusa con l'esecuzione della istruzione **return** che rimanda il controllo al programma chiamante e permette di usare il valore di ritorno come il valore della funzione stessa

- Dicendo che la funzione viene 'conclusa' o 'chiusa', si intende che la funzione termina e la memoria usata per le eventuali variabili temporanee che sono state allocate durante la sua esecuzione viene rilasciata

Sottoprogrammi: le procedure (1)

- Le **procedure** sono sottoprogrammi che possono avere dei parametri in ingresso, ma che **non restituiscono in uscita nessun valore**
- Le procedure non sono tipizzate e quindi non possono essere usate come funzioni matematiche in espressioni a destra del simbolo di assegnazione (=)
- Esempio:

```
void Stampa(int a){
    printf(“Stampo il valore %d”, a);
    return;
}
```

- La parola chiave return può essere usata anche nelle procedure senza specificare il suo parametro e provoca l’uscita dalla procedura per tornare al chiamante

Sottoprogrammi: le procedure (2)

```
#include <stdio.h>
void Stampa(int a);//dichiarazione o prototipo della procedura

int main(){
        int d=8;
        Stampa(d); //chiamata della procedura
}
void Stampa(int a){ //definizione della procedura
        printf("Stampo il valore della variabile in ingresso
        alla funzione %d", a);
        return;
}
```

```
Stampo il valore della variabile in ingresso alla procedura: 8
```

Sottoprogrammi: legame tra parametri formali e attuali (1)

- I parametri **formali** sono i parametri che si trovano nella **definizione della funzione**
- I parametri **attuali** sono quelli usati nel momento in cui **la funzione viene chiamata**
- I parametri formali assumono una realizzazione concreta nel momento in cui la funzione riceve il controllo
- Al momento della chiamata, la funzione chiamante specifica dei parametri attuali che vengono legati ai parametri formali per attribuire a loro una identità e un valore

Sottoprogrammi: legame tra parametri formali e attuali (2)

… int Somma(**int A, int B**); // A e B sono i parametri **formali** ovvero quelli //che si trovano nella **definizione della funzione**

```
main() {
    int C, X=5, Y=56;
    C = Somma(X,Y); // X e Y sono i parametri attuali ovvero quelli
                    //usati nel momento in cui la funzione viene chiamata
    printf("Ecco la somma: %d\n", C);
}
int Somma(int A, int B) {
    int d;
    d = A + B;
    return d;
}
```

Sottoprogrammi: legame tra parametri formali e attuali (3)

- Vi sono almeno due modi diversi per trasferire i parametri:
  - Per valore
  - Per indirizzo
- Il passaggio **per valore** comporta la **duplicazione del valore del parametro attuale**
  - In questo modo si garantisce che il parametro attuale NON verrà modificato dal sottoprogramma
- **Il passaggio per indirizzo** permette al sottoprogramma di lavorare **DIRETTAMENTE sulla variabile trasmessa come parametro attuale**
  - Le modifiche apportate all'argomento diventano visibili anche dal programma chiamante

Sottoprogrammi: legame tra parametri formali e attuali (4)

```
//ESEMPIO DI PASSAGGIO PER VALORE
… int Somma(int A, int B); //A e B sono parametri formali
main() {
    int C, X=5, Y=56;
    C = Somma(X,Y); // X e Y sono i parametri attuali
    printf("Ecco la somma: %d\n", C);
}

int Somma(int A, int B) {
    int d;
    d = A + B;
    return d;
}
```

- *Il passaggio* ***per valore*** *comporta la* ***duplicazione del valore del parametro attuale.***
- *In questo modo si garantisce che il parametro attuale NON verrà modificato dal sottoprogramma*

Sottoprogrammi: legame tra parametri formali e attuali (5)

- Nel C per realizzare il **passaggio per indirizzo**, il programmatore deve ricorrere all’uso dei **puntatori** e degli operatori unari * ed &
- In questo caso si parla del passaggio del valore dell’indirizzo
- Quando è necessario passare un argomento per indirizzo, si passa il valore del puntatore all’argomento in questione
- Il sottoprogramma usa il puntatore per accedere direttamente alla variabile in questione

Passaggio per puntatore: esempio

```
void SommaP(int A, int B, int *C);
main() {
        int a, b = 5, c = 2;
        SommaP(b,c,&a);   //variabili attuali
        printf("A che valore punta a? %d", a);
}

void SommaP(int A, int B, int *C) {//variabili formali temporanee
        *C = A + B;
}
```

- **Il passaggio per indirizzo** permette al sottoprogramma di lavorare **DIRETTAMENTE sulla variabile trasmessa come parametro attuale**
- Le modifiche apportate all’argomento diventano visibili anche dal programma chiamante (si lavora direttamente sulle variabili attuali)

Riferimento a variabili array (1)

- Il nome associato ad una variabile array denota il valore (costante) dell'indirizzo a partire dal quale l'array è memorizzato
- Il tipo di tale valore è quello di un puntatore a variabili del tipo raccolte nell'array
- Le singole variabili che costituiscono l'array possono essere referenziate combinando il nome con un indice (un numero intero non negativo) che numera gli elementi dell'array a partire da 0

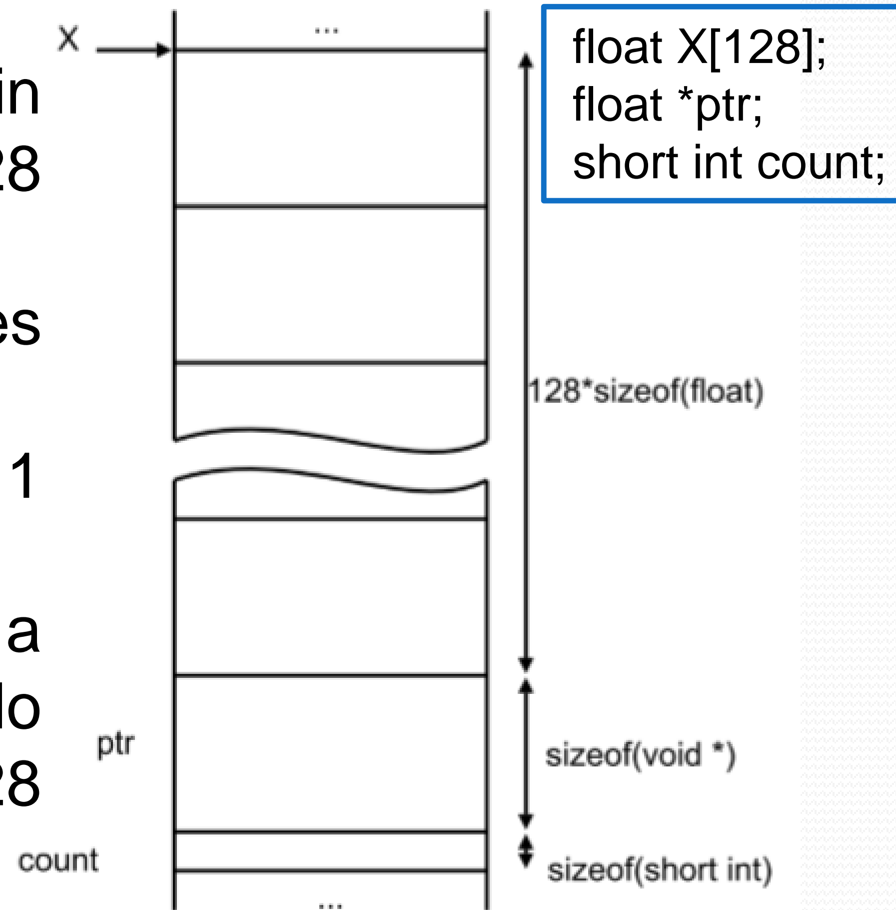
Riferimento a variabili array- esempio (1)

```
…
float X[128];
float *ptr;
short int count;
…
ptr = X;                      //ptr punta la prima locazione dell'array X
ptr[0] = 10;                  //assegna 10 alla prima variabile di X
(ptr + 1)[0] = 20;            //assegna 20 alla seconda variabile di X
count = 1;
(ptr + 1)[count + 1] = 40; //assegna 40 alla quarta variabile di X
X[200] = 0;                       //scrive uno zero in formato float alla
                                  //locazione di memoria che si trova
                                  //partendo dall'indirizzo di base
                                  //dell'array X e avanzando di 200 float !!!
                                  //'Spara a caso in memoria'
```

Riferimento a variabili array - esempio (2)

- Allocazione di memoria di X, ptr e count:
  - Per X viene riservato in memoria lo spazio di 128 variabili float (128*4 bytes)
  - Per ptr sono riservati 4 bytes (indirizzi a 32 bit)
  - Per count viene riservato 1 byte
- Il simbolo X denota l'indirizzo a partire dal quale è riservato lo spazio in memoria per le 128 variabili
- ptr e count sono nomi di variabili

```
float X[128];
float *ptr;
short int count;
```



Array multidimensionali (1)

- Consideriamo il caso in cui siano assegnate due matrici di numeri reali A e B e se ne voglia **calcolare il prodotto C**
- Sapendo che: Se A e B hanno dimensione I x H e H x J, allora (si noti che n°colonneA = n°righeB = H):
  - C ha dimensione I x J
  - L'elemento in posizione ij di C è il risultato del prodotto scalare tra: la riga i-esima di A e la colonna j-esima di B:

$$A_{I\times H}B_{H\times J} = C_{I\times J}$$
$$c_{ij} = \sum_{h=1,H} A_{ih}B_{hj}$$

Prodotto tra matrici – esempio (1)

Le tre matrici A,B e C possono essere rappresentate come tre array di array di variabili

```
int A[2][2] = { {1,0} ,{2,3} };
int B[2][3] = {{0,1,2}, {1,1,1}};
int C[2][3];
int i, j, h;
for (i = 0; i<2; i++) {        // I=2
     for (j = 0; j<3; j++) { //J=3
          C[i][j] = 0;           //H=2
          for (h = 0; h < 2; h++) {
               C[i][j] = C[i][j] + A[i][h] * B[h][j];
          }
     }
}
//stampo somma elemento C[1][2] , con indice di riga 1 e indice colonna 2
printf("Elemento C[1][2] = A[1][0]*B[0][2] + A[1][1]*B[1][2];\n
Sostituendo i valori: %d= %d*%d+%d*%d \n",
C[1][2],  A[1][0], B[0][2], A[1][1], B[1][2]);
}
```

```
//C[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0]
//C[0][1] = A[0][0]*B[0][1] + A[0][1]*B[1][1]
//C[0][2] = A[0][0]*B[0][2] + A[0][1]*B[1][2]
//C[1][0] = A[1][0]*B[0][1] + A[1][1]*B[1][0]
//C[1][1] = A[1][0]*B[0][1] + A[1][1]*B[1][1]
//C[1][2] = A[1][0]*B[0][2] + A[1][1]*B[1][2]
```

$$A_{I\times H}B_{H\times J} = C_{I\times J}$$
$$c_{ij} = \sum_{h=1,H} A_{ih}B_{hj}$$

Prodotto tra matrici – esempio (2)

```
C:\WINDOWS\system32\cmd.exe
Elemento C[1][2] = A[1][0]*B[0][2] + A[1][1]*B[1][2];
Sostituendo i valori: 7= 2*2+3*1
Premere un tasto per continuare . . .
```

```
//C[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0]
//C[0][1] = A[0][0]*B[0][1] + A[0][1]*B[1][1]
//C[0][2] = A[0][0]*B[0][2] + A[0][1]*B[1][2]
//C[1][0] = A[1][0]*B[0][1] + A[1][1]*B[1][0]
//C[1][1] = A[1][0]*B[0][1] + A[1][1]*B[1][1]
//C[1][2] = A[1][0]*B[0][2] + A[1][1]*B[1][2]
```

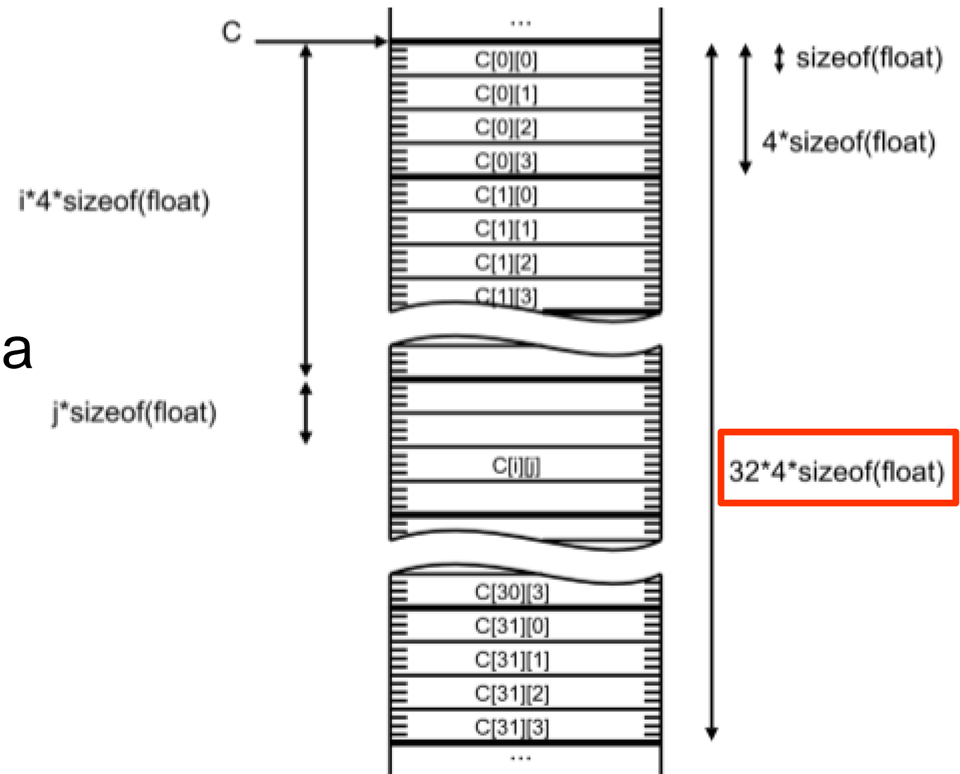
Array multidimensionali (2)

- Le tre matrici A, B e C possono essere rappresentate come tre array di array di variabili:

```
float A[32][16];
float B[16][4];
float C[32][4];
… //inserire valori nelle matrici
int i, j, k;
for (i = 0; i<32; i++){
        for (j = 0; j<4; j++) {
                C[i][j] = 0;
                for (h = 0; h < 16; h++) {
                        C[i][j] = C[i][j] + A[i][h] * B[h][j];
                        //alternativa: C[i][j]+=A[i][h]*B[h][j];
                }
        }
}
```

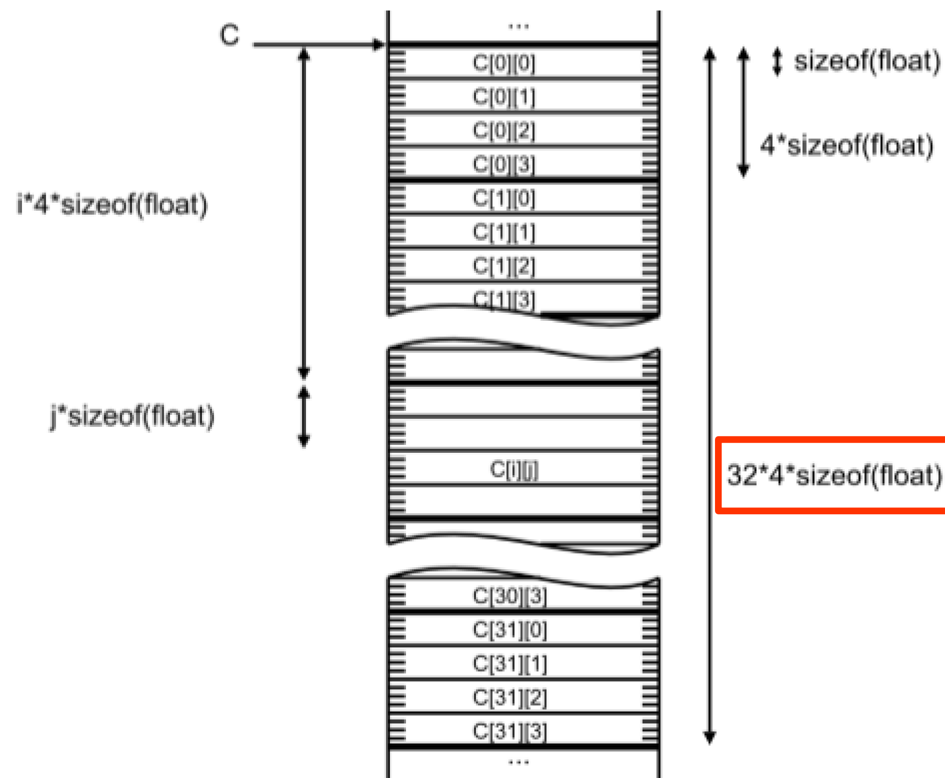
Array multidimensionali (2)

- Si noti che, le matrici A,B,C :
  - Sono oggetti di natura bi-dimensionale
  - Devono essere rappresentate nella memoria di un elaboratore che è organizzata come una sequenza lineare di locazioni



Array multidimensionali (3)

- Matrice C:
  - È dichiarata (float C[32][4];) come array di 32 array di 4 variabili float
  - C denota un valore che punta ad un segmento di memoria nel quale sono rappresentate 32 variabili
  - Ciascuna di queste 32 variabili è a sua volta un segmento di 4 variabili di tipo float
  - Le singole variabili dell'array sono serializzate in memoria secondo un ordine riga-colonna



- Nelle prime locazioni abbiamo gli elementi della prima riga e nella ultime quelli dell'ultima riga

Array multidimensionali (4)

- La dichiarazioni di matrici come array di array è del tutto naturale, però incontra grosse limitazioni nell'uso di funzioni e variabili allocate dinamicamente
- Nella pratica della programmazione del C le matrici a più dimensioni sono comunemente dichiarate **come array monodimensionali**, lasciando al programmatore la gestione della indicizzazione
- In questo caso, una matrice di dimensioni IxJ:
  - **è dichiarata come un array monodimensionale di dimensione TxJ**
  - **L'elemento [i][j] è selezionato con l'indice [i*J+j]**

Array multidimensionali (5)

- Il prodotto tra matrici può essere scritto anche come:

```
float A[32 * 16]; //A[512]
float B[16 * 4];   //B[64]
float C[32 * 4];   //C[128]
int i, j, h;
for (i = 0; i<32; i++){
      for (j = 0; j<4; j++) {
          C[i * 4 + j] = 0;
          for (h = 0; h<16; h++) {
                C[i * 4 + j] += A[i * 16 + h] * B[h * 4 + j];
          }
      }
}
```

Array multidimensionali (6)

```
#include <stdio.h>
#include <stdlib.h>
main(){
  int A[2*2] = {1,0, 2,3 };
  int B[2*3] = {0,1,2,1,1,1};
  int C[2*3];
  int i, j, h ,z;
  for (i = 0; i<4; i++) {
          for (j = 0; j<3; j++) {
                    C[i * 3 + j] = 0;
                    for (h = 0; h < 2; h++) {
                         C[i * 3 + j] += A[i * 2 + h] * B[h * 3 + j];
                    }
          }
}
for(z=0; z<6;z++ ){
 printf("C[%d] vale %d\n",z, C[z]); } }
```

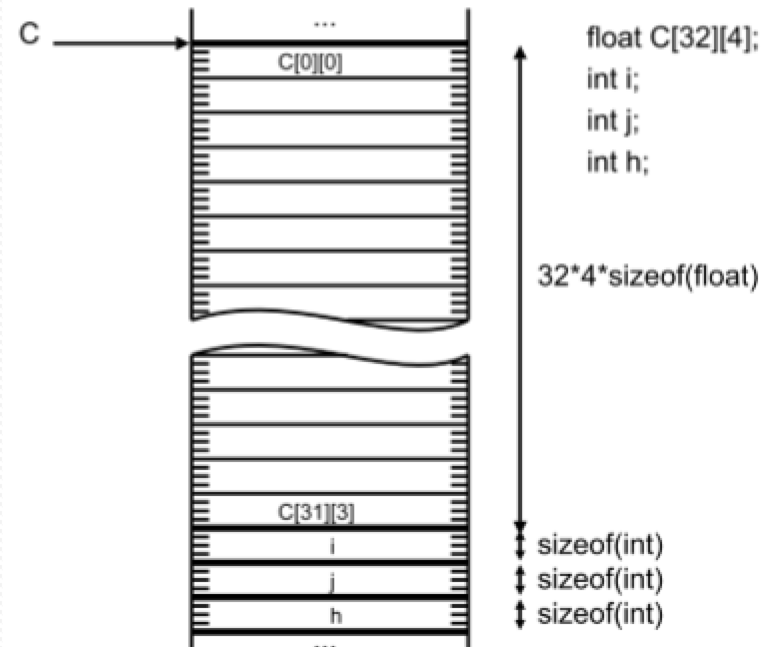
```
C[0] vale 0
C[1] vale 1
C[2] vale 2
C[3] vale 3
C[4] vale 5
C[5] vale 7
```

Array multidimensionali (6)

- Il metodo può essere applicato a matrici di 3 o più dimensioni.
- Nel caso di una matrice a tre dimensioni:
  - Matrice dichiarata con più indici:
    - Ha la forma float X3[128][16][4];
    - l'elemento di indice ijk viene referenziato come X3[i][j][k]
  - Matrice dichiarata con indice unico:
    - Ha la forma float X3[128*16*4];
    - l'elemento di indice ijk viene referenziato come: X3[i*J*K+j*K+k], dove J e K denotano il numero di valori del secondo e del terzo indice (16 e 4 nell'esempio)
- Nota: ai fini del riferimento il valore del primo indice (128) è irrilevante

Array multidimensionali (7) – Allocazione Dinamica

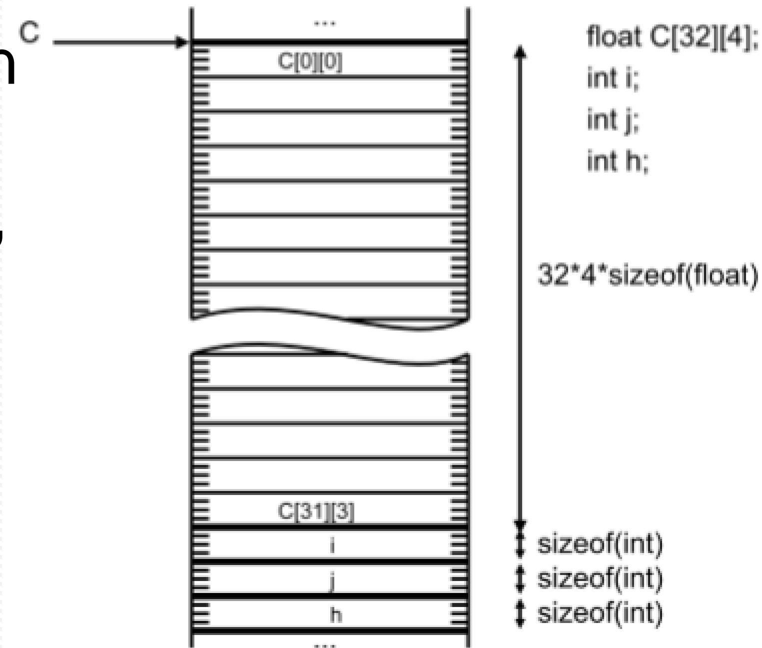
- Nella dichiarazione di un array, la specifica del numero di elementi è necessaria al compilatore per determinare l'ampiezza del segmento di memoria che deve essere riservato
- Facendo ancora riferimento all'esempio legato al prodotto tra matrici, sappiamo che il compilatore riserva in memoria per la matrice C un segmento di 32*4*sizeof(float) bytes e colloca nei bytes che seguono gli indici i, j, h



NOTA: per riservare spazio in memoria agli indici, il compilatore deve conoscere la dimensione di C

Array multidimensionali (8) – Allocazione Dinamica

- Sapendo che per riservare spazio in memoria agli indici, il compilatore deve conoscere la dimensione di C, si deduce che:
  - Non è possibile dichiarare un array di dimensione variabile, il cui valore sia determinato al tempo di esecuzione del programma (magari in base ad un dato ricevuto in input)
  - Questo può diventare un problema concreto per molti programmi



Array multidimensionali (8) – Allocazione Dinamica

- Se volessimo calcolare il prodotto di due matrici di dimensione 7 × 4 e 4 × 3, dovremmo sostituire tutte le occorrenze di 32, 16 e 4 con 7, 4 e 3.
- La difficoltà è alleviata dall’uso della direttiva di preprocessing #define che permette di denotare un valore con un simbolo
- Grazie a questa direttiva, possiamo riscrivere il calcolo del prodotto delle due matrici

Array multidimensionali (9) – Allocazione Dinamica

```
#define I 32
#define H 16
#define J 4
float A[I*H];
float B[H*J];
float C[I*J];
int i, j, h;
for(i=0;i<I;i++) {
    for(j=0;j<J;j++){
        C[i*J+j]=0;
        for(h=0;h<H;h++) {
            C[i*J+j]+=A[i*H+h]*B[h*J+j];

    }
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Array multidimensionali (10) – Allocazione Dinamica

- L'uso della direttiva #define riduce la complessità di editing del programma e la possibilità di commettere errori
- Tuttavia non permette di trattare arrays la cui dimensione viene determinata durante l'esecuzione
- Infatti i valori di I, H, J sono comunque fissati prima che il programma venga lanciato
- Per risolvere il problema occorre che lo spazio per la rappresentazione dell'array sia allocato nel corso dell'esecuzione del programma
- Questo non lo può fare il compilatore, ma può essere realizzato interfacciando il programma con il sistema operativo che gestisce le risorse della macchina, fra cui la memoria.

Array multidimensionali (10) – Allocazione Dinamica

- La funzione di libreria **malloc()** richiede al sistema operativo l'allocazione di un segmento di memoria di dimensione determinata al momento della chiamata e restituisce il puntatore alla locazione della memoria a partire dalla quale è stato riservato lo spazio.
- Usando la **malloc**() è possibile calcolare il prodotto di due matrici di dimensione determinata nel corso dell'esecuzione

Array multidimensionali (11) – Allocazione Dinamica

```
#include<stdlib.h>
#include<stdio.h> ...
int i, j, h, I, H ,J;
float *A, *B, *C; ... //puntatori a float
scanf("%d",&I); // acquisisce da tastiera il valore di I, J, e H
scanf("%d",&H);
scanf("%d",&J); ...
A=(float *)malloc(I*H*sizeof(float)); // alloca i tre array
B=(float *)malloc(H*J*sizeof(float));
C=(float *)malloc(I*J*sizeof(float));
for(i=0;i<I;i++) {
        for(j=0;j<J;j++){
                C[i*J+j]=0;
                for(h=0;h<H;h++)
                        C[i*J+j]+=A[i*H+h]*B[h*J+j];
        }
}
```

Allocazione statica:
*float A[I*H];*
*float B[H*J];*
*float C[I*J];*

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Array multidimensionali (12) – Allocazione Dinamica

- Si consideri l'istruzione:
  - A=(float *)malloc(I*H*sizeof(float));
- **malloc**() richiede al sistema operativo di riservare un segmento di memoria grande quanto è indicato nel suo **argomento**. In questo caso richiede quindi di riservare:
  - I*H*sizeof(float) bytes
  - sizeof(float) restituisce la dimensione di un float, ovvero la dimensione di ciascuna locazione in termini di byte
- La funzione **malloc**() **restituisce il puntatore al primo byte del segmento allocato**
- Lo restituisce nel tipo di puntatore a carattere
- Il cast (float *) converte il puntatore nella forma di puntatore a float che è quello che serve nel nostro esempio

Array multidimensionali (13) – Allocazione Dinamica

- Una volta allocato l'array e assegnato ad A il suo puntatore, gli elementi di A sono referenziati con l'indice entro parentesi quadre (ad esempio A[i*H+h]), nello stesso modo con cui potremmo farlo se A fosse stato dichiarato staticamente
- Un segmento di memoria allocato con la funzione malloc() resta in uso fino al termine del programma
- **Nel caso in cui il segmento cessi di essere utile prima del termine del programma, può essere rilasciato attraverso l'uso della funzione free()** (in #include<memory.h>):

```
...
float * C;
...
C=(float *)malloc(I*J*sizeof(float));
...
free(C);
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

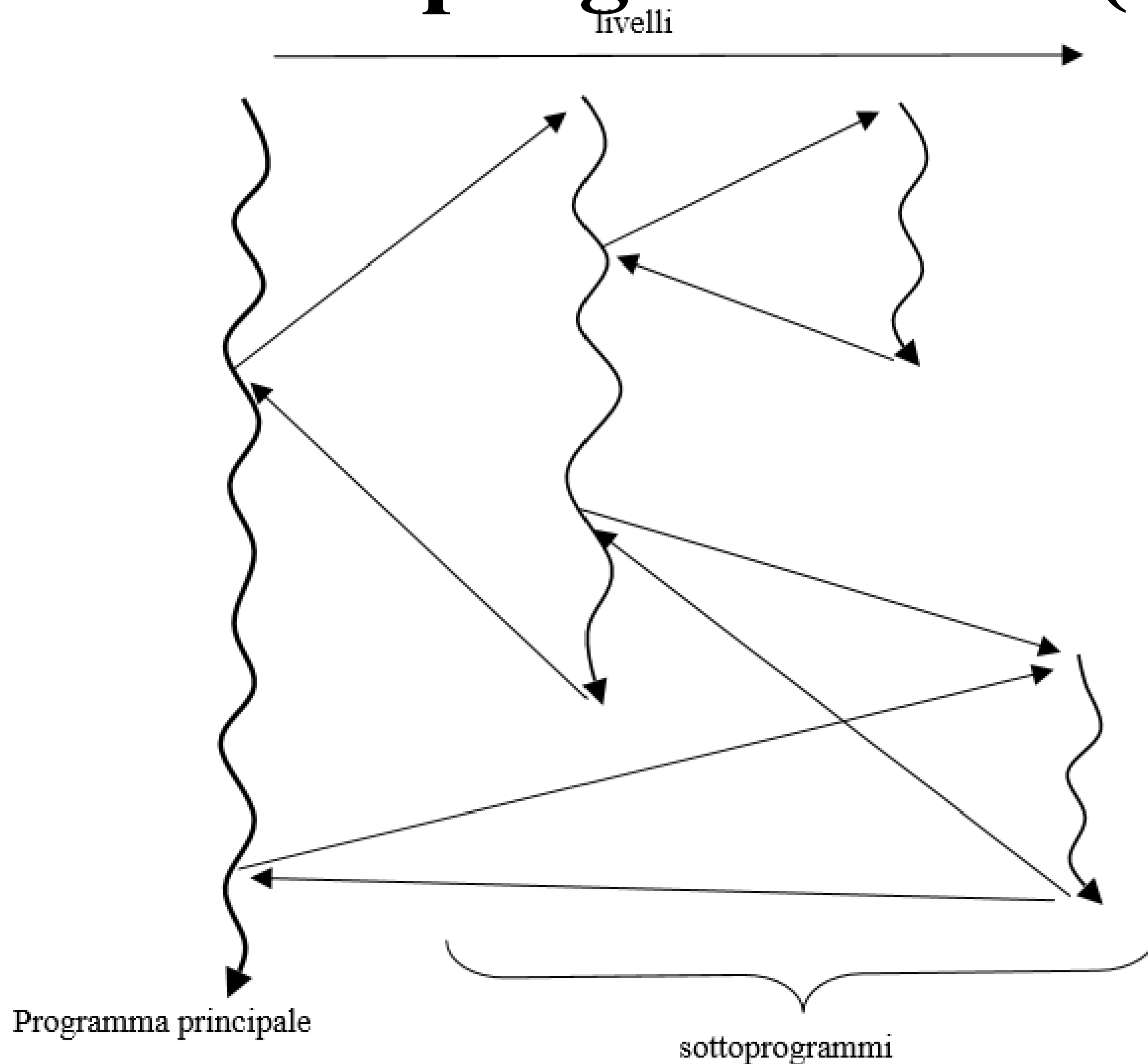
Tornando al concetto di sottoprogramma… (1)

- La realizzazione di un algoritmo prevede l'uso di algoritmi più semplici
- Gli algoritmi più semplici spesso costituiscono porzioni che possono essere riutilizzate più volte nello stesso o in altri contesti
- Questo meccanismo può essere usato per suddividere l'algoritmo inziale in un insieme di algoritmi più semplici che, una volta risolti, possono portare alla soluzione del problema completo iniziale
- E' quindi necessario un metodo per la definizione e l'aggregazione delle istruzioni: blocchi o sequenze, procedure e funzioni
- Questo meccanismo si realizza con la formalizzazione di sottoprogrammi

Tornando al concetto di sottoprogramma… (2)

- I sottoprogrammi sono particolari strutture di controllo che, in seguito alla loro invocazione, **alterano il flusso del programma**
- Il programma principale può usare un sottoprogramma per effettuare determinate operazioni
- Il sottoprogramma a sua volta può richiamare altri sottoprogrammi
- I sottoprogrammi possono essere usati in più punti di un programma
- E' importante che i singoli sottoprogrammi lavorino su variabili proprie o su copie, comunque solo in modo controllato

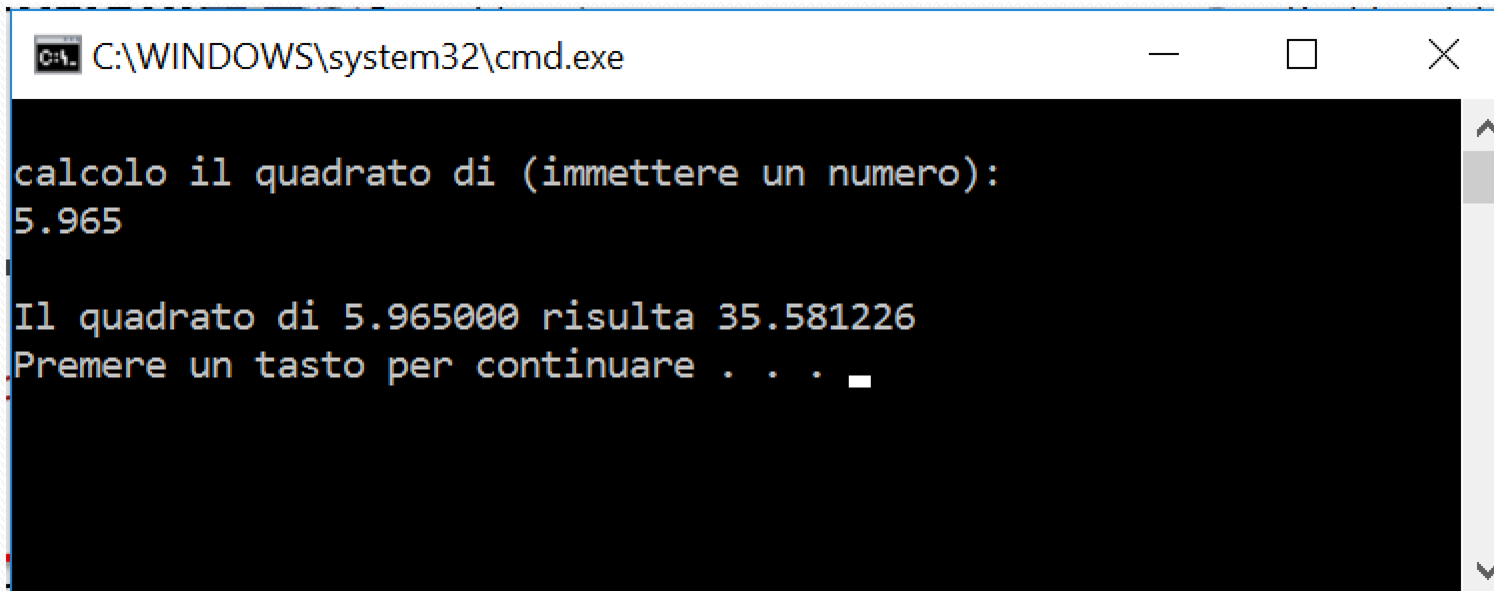
Tornando al concetto di sottoprogramma… (3)



Esempio (1)

```
#include <stdio.h>
float square ( float x ); // Prototipo o dichiarazione
float VariabileGlobale; // variabile globale
int main(){
  float m, n ;
  printf ( "\nEnter some number for finding square \n");
  scanf ( "%f", &m );
  n = square (m) ;  // Chiamata della funzione
  printf ( "\nSquare of the given number %f is %f", m, n );
}
//definizione della funzione
float square ( float x ){
  float p; p = x * x;
  return p; // Valore di ritorno
}
```

Esempio (2)



Passaggio per puntatore: esempio

```
void SommaP(int A, int B, int *C) ; //dichiarazione

main() {
        int a, b = 5, c = 2;
        SommaP(b,c,&a);    //chiamata della funzione con
                           //variabili attuali
        printf("A che valore punta a? %d", a);
}

void SommaP(int A, int B, int *C) {  //definizione funzione
                                     //variabili formali
        *C = A + B;
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

…tornando allo Stack di sistema (1)

- Quando un sottoprogramma viene eseguito, per ogni suo parametro vengono allocate delle celle nella memoria detta Stack
- Inoltre vengono allocate nello Stack anche le variabili temporanee dichiarate nel sottoprogramma ed una cella per il valore di ritorno se il sottoprogramma è di tipo function
- Lo spazio è allocato in accordo alla rappresentazione in memoria del tipo di ogni variabile
- Lo Stack è una particolare area di memoria usata per tale scopo ed è organizzata come una pila, secondo la regola FILO (First In Last Out, il primo che entra è l'ultimo ad uscire)

…tornando allo Stack di sistema (2)

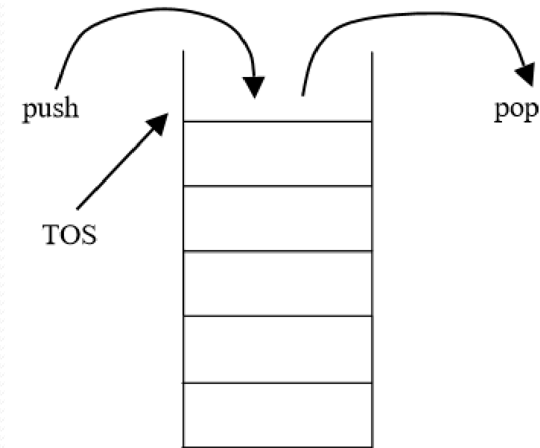
- Lo stack di sistema è un'area di memoria che supporta i meccanismi di trasferimento del controllo e la comunicazione tra le funzioni, riflettendo la possibile nidificazione delle chiamate
- Tale area di memoria rimane trasparente al programmatore ma la comprensione del suo funzionamento permette di capire le regole relative al tempo di vita delle variabili e la visibilità delle dichiarazioni
- Al momento della chiamata a funzione, sullo Stack viene memorizzato l'indirizzo dell'ultima istruzione eseguita nella funzione chiamante che SERVE al TERMINE della funzione chiamata per determinare l'INDIRIZZO della istruzione a cui deve essere restituito il controllo

…tornando allo Stack di sistema (3)

- Al momento della chiamata ad un sottoprogramma, sullo stack viene memorizzato l’indirizzo dell’ultima istruzione eseguita nella funzione chiamante, che serve al termine della funzione chiamata per determinare l’indirizzo dell’istruzione a cui deve essere restituito il controllo
- Sullo stack viene poi riservato lo spazio per “**attualizzare**” i parametri formali della funzione chiamata e tali parametri sono inizializzati con il valore restituito dai parametri attuali a loro collegati
- Infine, sempre sullo stack, sono allocate le variabili dichiarate all’interno della funzione chiamata.

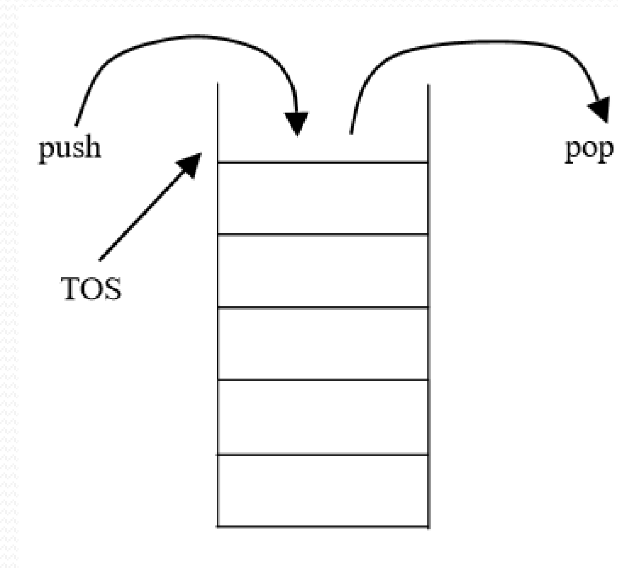
Tornando al concetto di stack… (1)

- La pila (o stack) è una **struttura lineare dove le operazioni di inserimento ed estrazione dei dati avvengono sempre dalla stessa parte**
- La pila è una struttura informativa di tipo:
  - FILO (First Input Last Output), se l'ultimo entrare è il primo ad uscire
  - LIFO (Last Input First Output) se l'ultimo entrare è anche il primo a uscire.
- Esempio: tubetto di pasticche delle vitamin per prendere l'ultima pasticca (ovvero la prima che è stata inserita) è necessario togliere tutte le pasticche precedenti



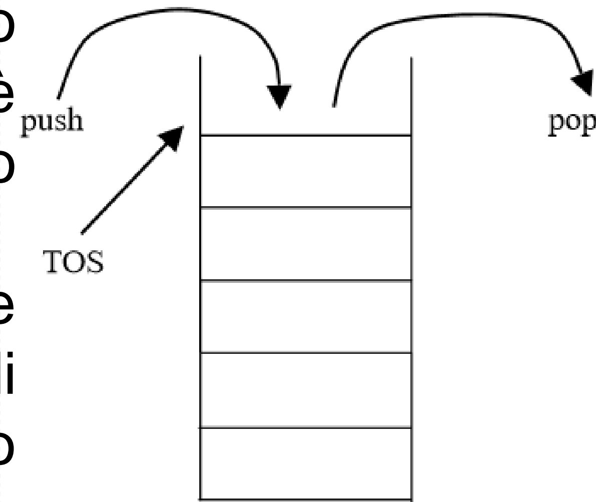
Tornando al concetto di stack… (2)

- L’operazione di inserimento è detta **PUSH**
- L’operazione di estrazione è detta **POP**
- Per la gestione di una pila è necessario conoscere solo il punto in cui viene fatta l’inserzione o l’estrazione, detto Top Of the Stack, **TOS**



Tornando al concetto di stack… (3)

- Lo stack è tipicamente una struttura **dinamica** perché le sue dimensioni possono non essere note al momento della sua realizzazione
- Le operazioni sulla pila fanno tutte riferimento al **TOS** (Top of the Stack), quindi è necessario monitorare tale elemento, in pratica è necessario tenere sotto controllo quando lo stack si svuota
- Se NON si effettua tale controllo, potrebbe infatti accadere di effettuare una operazione di POP (estrazione) senza che vi sia un elemento nello stack
- Le operazioni di base sono pertanto:
  - **Creazione della pila, inserimento (push), estrazione (pop), verifica di stack vuoto** PRIMA di fare ogni pop



Rappresentazione in C di uno stack (1)

```
… //inclusione librerie
//definizioni per l'uso di booleani
typedef unsigned short int Boolean;
#define TRUE 1
#define FALSE 0
//costante simbolica per il dimensionamento statico del buffer
#define N 128

//prototipi di funzioni  (dichiarazioni) per lo stack
void init(int *TOS_ptr);
Boolean push(float *buffer, int *TOS_ptr, float value);
Boolean pop(float *buffer, int *TOS_ptr, float *value_ptr);
void notify_push_failure(void);
void notify_pop_failure(void);
void getvalue(float *value_ptr);
void putvalue(float value);
void notify_selection_failure(char selection);
```

Rappresentazione in C di uno stack (2)

```
void main(void) {
    int TOS;
    float buffer[N]; //per memorizzare i dati si usa un array
    char selection[10];
    float value;
    int first = 0;
    Boolean exit_required, result;
    init(&TOS); //chiamata a funzione
    printf("Digita:\n");
    printf("-i per fare un inserimento nello stack\n");
    printf("-o per estrarre un elemento dallo stack stack\n");
    printf("-x per uscire dal programma\n");
    exit_required = FALSE;
    do{…}
    while(exit_required == FALSE)
}
printf("Fine!");
```

```
//dichiarazione
void init(int *TOS_ptr);
```

Rappresentazione in C di uno stack (3)

```
...
void main(void) { ...  do {
        printf("\nChe operazione vuoi fare? ");
        scanf("%s", selection); //legge comando
        switch (selection[0]) {
            case 'i':
                getvalue(&value); //prende il valore da inserire nello stack
                result = push(buffer, &TOS, value); //inserisce value nel buffer e rende TRUE
                if (result == FALSE)                //altrimenti rende FALSE
                    notify_push_failure(); //notifica il fallimento di un push
                break;
            case 'o':
                result = pop(buffer, &TOS, &value);
                if (result == TRUE)
                    putvalue(value);          //stampa a video di un float
                else
                    notify_pop_failure();     //notifica il fallimento di un pop
                break;
            case 'x':
                exit_required = TRUE;
                break;
            default:
                notify_selection_failure(selection[0]);   //notifica il fallimento della selezione
        }
} while (exit_required == FALSE);        printf("Fine!");
```

```
int TOS;
float buffer[N];
float value;
```

```
void getvalue(float *value_ptr);
```

```
Boolean push(float *buffer, int *TOS_ptr, float value);
```

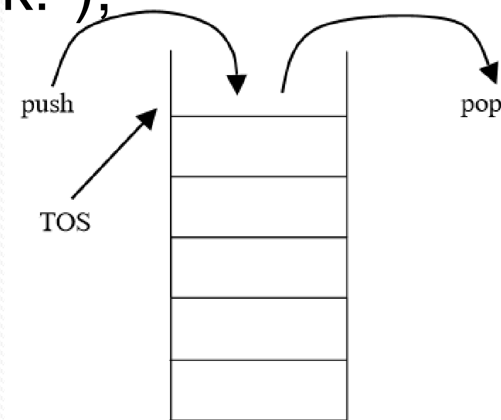
```
Boolean pop(float *buffer, int *TOS_ptr, float *value_ptr);
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Rappresentazione in C di uno stack (4)

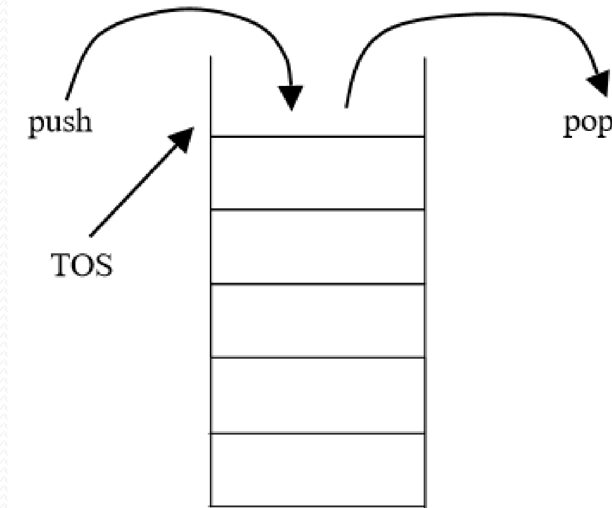
```
void init(int *TOS_ptr) {//inizializza a zero TOS
        *TOS_ptr = 0;//('ciò a cui punta TOS' ha valore 0)
}
Boolean push(float *buffer, int *TOS_ptr, float value) {
//se il buffer NON è pieno inserisce value e restituisce
//TRUE, altrimenti rende FALSE
        if (*TOS_ptr < N) {
                buffer[*TOS_ptr] = value;
                (*TOS_ptr)++;
                printf("Valore inserito nello stack!");
                        return TRUE;
        }
        else
                return FALSE;
}
```

```
int TOS;
float buffer[N];
float value;
```



Rappresentazione in C di uno stack (5)

```
Boolean pop(float *buffer, int *TOS_ptr, float *value_ptr) {
//SE il buffer NON è vuoto, ne estrae un valore in *value_ptr e
//restituisce TRUE, altrimenti rende FALSE
        if (*TOS_ptr>0) {
                (*TOS_ptr)--;
                *value_ptr = buffer[*TOS_ptr];
                return TRUE;
        }
        else {
                return FALSE;
        }
}
void getvalue(float *value_ptr) {
//acquisisce un float da tastiera (passaggio per puntatore)
        printf("\nInserisci un valore: ");
        scanf("%f", value_ptr);
}
```




```
int TOS;
float buffer[N];
float value;
```

Rappresentazione in C di uno stack (6)

```
void putvalue(float value) {
        //stampa a video di un float
        printf("\nvalore estratto: %f", value);
}
void notify_push_failure(void) {
        //notifica il fallimento di un push
        printf("\nBuffer pieno!! Inserimento cancellato");
}
void notify_pop_failure(void) {
    //notifica il falimento di un pop
    printf("\nBuffer Vuoto!! Non è possibile estrarre elementi");
}
void notify_selection_failure(char selection) {
        //notifica il fallimento della selezione
        printf("\n %c Selezione non legale!!", selection);
}
```

Rappresentazione in C di uno stack (6)

- Esempio di inserimenti ed estrazioni

C:\WINDOWS\system32\cmd.exe

```
Digita:
-I per fare un inserimento nello stack
-O per estrarre un elemento dallo stack stack
-X per uscire dal programma

Che operazione vuoi fare? i
Inserisci un valore: 7
Valore inserito nello stack!
Che operazione vuoi fare? i
Inserisci un valore: 23.97
Valore inserito nello stack!
Che operazione vuoi fare? o

valore estratto: 23.969999
Che operazione vuoi fare? i
Inserisci un valore: 9765.09
Valore inserito nello stack!
Che operazione vuoi fare? x
Fine!Premere un tasto per continuare . . .
```

Rappresentazione in C di uno stack (7)

- Supponiamo di aver definito: #define N 3

```
C:\WINDOWS\system32\cmd.exe
Digita:
-I per fare un inserimento nello stack
-O per estrarre un elemento dallo stack stack
-X per uscire dal programma

Che operazione vuoi fare? i
Inserisci un valore: 8
Valore inserito nello stack!
Che operazione vuoi fare? i
Inserisci un valore: 89
Valore inserito nello stack!
Che operazione vuoi fare? i
Inserisci un valore: 09
Valore inserito nello stack!
Che operazione vuoi fare? i
Inserisci un valore: 45

Buffer pieno!! Inserimento cancellato
Che operazione vuoi fare?
```

Rappresentazione in C di uno stack (7)

C:\WINDOWS\system32\cmd.exe

```
Digita:
-I per fare un inserimento nello stack
-O per estrarre un elemento dallo stack stack
-X per uscire dal programma

Che operazione vuoi fare? i
Inserisci un valore: 6
Valore inserito nello stack!
Che operazione vuoi fare? i
Inserisci un valore: 23
Valore inserito nello stack!
Che operazione vuoi fare? i
Inserisci un valore: 9

Buffer pieno!! Inserimento cancellato
Che operazione vuoi fare? o

valore estratto: 23.000000
Che operazione vuoi fare? o

valore estratto: 6.000000
Che operazione vuoi fare? o

Buffer Vuoto!! Impossibile estrarre elementi
Che operazione vuoi fare? _
```

- Supponiamo di aver definito: #define N 2

Tornando al concetto di record… (1)

- Un record è un insieme ordinato ed eterogeneo di elementi
- Il singolo elemento del record può essere chiamato campo o componente
- In un record si possono avere componenti di vario tipo: interi, numeri naturali, vettori di interi, etc.
- Il record è tipicamente una **struttura statica** nel senso che non è possibile variarne il numero di componenti dopo la sua creazione iniziale
- Esempio:
  - cognome: Rossi
  - nome: Mario
  - via: Via Ponte Verde
  - civico: 45

Tornando al concetto di record… (2)

- Ogni campo del record viene identificato dal nome del campo:
  - cognome è il **nome del** primo **campo**, e Rossi è il relativo contenuto informativo (**valore**)
  - Cognome è una stringa mentre civico è un intero
- Il **record** si può rappresentare nel modo seguente:
  - **{cognome, nome, via, civico}**
- Tipicamente il record stesso o meglio la struttura che lo definisce, ha un nome
- Esempio:
  - cognome: Rossi
  - nome: Mario
  - via: Via Ponte Verde
  - civico: 45

Dati strutturati (1)

- Un dato strutturato è un **insieme di variabili**, anche di tipo diverso, che possono essere dichiarate e referenziate attraverso un **nome collettivo** e un indice simbolico
- Questo fornisce un meccanismo supportato dal compilatore che garantisce l'associazione tra variabili correlate
- Si pensi ad esempio al caso del descrittore di uno studente, fatto di: un **nome**, un **cognome** e un **numero di matricola**:
  - i tre dati possono essere codificati in tre variabili, associate dalla disciplina del programmatore attraverso un buon uso dei nomi e posizionamento conveniente delle dichiarazioni

Dati strutturati (2)

- Mentre le variabili di un array sono posizionate in locazioni contigue e di uguale dimensione**, le variabili di un dato strutturato, essendo di tipi diversi, sono posizionate a distanza non regolare**
- E per ragioni di ottimizzazione della compilazione spesso non sono neppure posizionate in locazioni contigue, diversamente da quanto avveniva per gli arrays
- Occorre allora che il compilatore disponga di una **definizione di tipo dello specifico dato** che determini quali sono i campi che lo compongono, quale è la dimensione complessiva del dato, e in quali posizioni relative sono collocati i diversi campi
- La definizione di **tipo di un dato strutturato** è una nuova categoria sintattica basata sull’uso della parola chiave **struct**

Dati strutturati (3)

- **struct** serve per dichiarare un tipo di dato che definisce un gruppo di variabili di tipo diverso sotto lo stesso nome in un blocco di memoria
- L'accesso ai dati è ottenuto attraverso un **puntatore** oppure usando il nome della **struct** dichiarata che ritorna lo stesso indirizzo
- Esempio:

```
struct persona {
        char *nome;
        char *cognome;
        char *indirizzo;
        int civico;
};
```

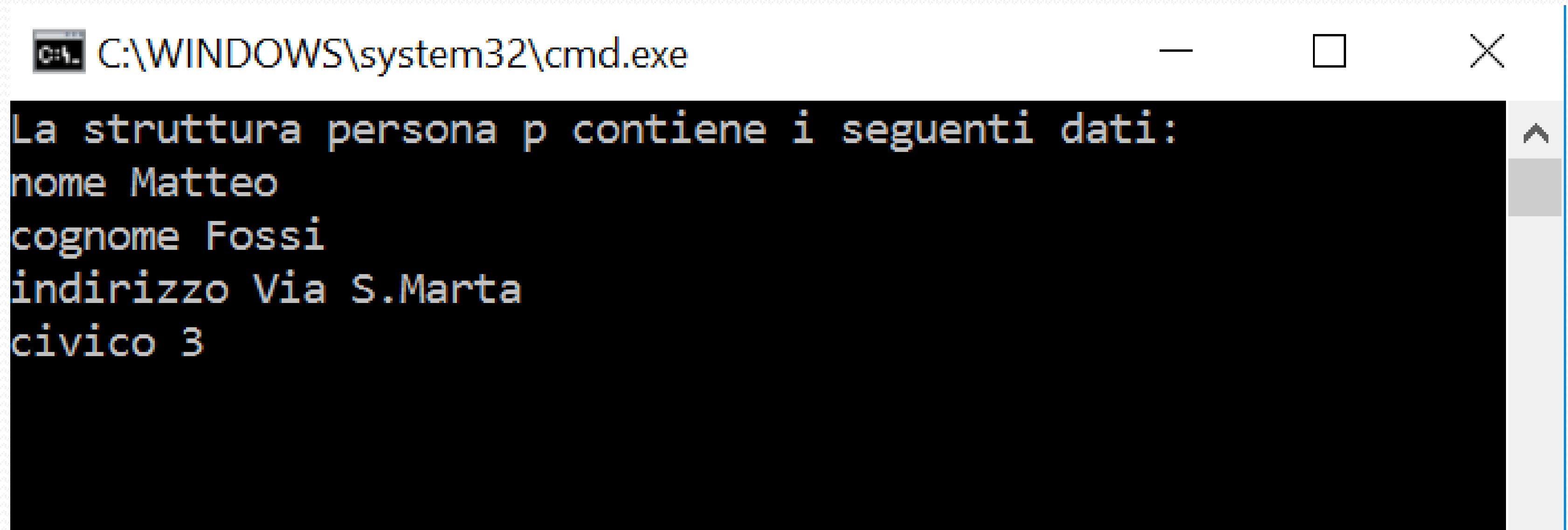
Dati strutturati (4)

```
struct persona { //definizione
        char *nome;
        char *cognome;
        char *indirizzo;
        int civico;
};
struct persona p; //Dichiarazione
//per accedere alle componenti:
p.Nome …
p.cognome  …
p.indirizzo  …
p.civico …
```

Dati strutturati (5)

```
...
struct persona { //definizione
    char *nome;
    char *cognome;
    char *indirizzo;
    int civico;
};
main(){
        struct persona p; //Dichiarazione:
        p.nome = "Matteo";
        p.cognome = "Fossi";
        p.indirizzo = "Via S.Marta";
        p.civico = 3;
        printf("La struttura persona p contiene i seguenti dati:\n",);
        printf("nome %s\ncognome %s\nindirizzo %s\ncivico
        %d\n", p.nome, p.cognome, p.indirizzo, p.civico);
}
```

Dati strutturati (6)



Rappresentazione di uno stack con struct (1)

```
struct stack {
        int size;
        int TOS;
        float *buffer;
};
//NOTA: i vari elementi saranno inseriti tramite allocazione dinamica
//protipi funzioni per la definizione dello stack
void getsize(int *size_ptr);
void init(struct stack *ptr, int size);
Boolean push(struct stack *ptr, float value);
Boolean pop(struct stack *ptr, float *value_ptr);
void getvalue(float *value_ptr);
void putvalue(float value);
void notify_push_failure(void);
void notify_pop_failure(void);
void notify_selection_failure(char selection);
```

```
//rappresentazione Stack senza struct
void init(int *TOS_ptr);
Boolean push(float *buffer, int *TOS_ptr, float value);
Boolean pop(float *buffer, int *TOS_ptr, float *value_ptr);
```

Rappresentazione di uno stack con struct (2)

```
void main(void) {
        struct stack buffer;
        int size=0;
        float value;
        Boolean result, exit_required;
        char selection[10];
        getsize(&size);
        init(&buffer, size);
        printf("Digita:\n");
        printf("-I per fare un inserimento nello stack\n");
        printf("-O per estrarre un elemento dallo stack stack\n");
        printf("-X per uscire dal programma\n");
        exit_required = FALSE;
do {…
} while();
```

```
struct stack {
        int size;
        int TOS;
        float *buffer;
};
```

```
void getsize(int *size_ptr);
void init(struct stack *ptr, int size);
```

Rappresentazione di uno stack con struct (3)

```
void main(void){…
  do {
     printf("\nChe operazione vuoi fare? ");
     scanf("%s", selection);
     switch (selection[0]) {
          case 'i':
             getvalue(&value);
             result = push(&buffer, value);
             if (result == FALSE)
                    notify_push_failure();
             break;
          case 'o':
              result = pop(&buffer, &value);
              if (result == TRUE)
                    putvalue(value);
              else
                    notify_pop_failure();
              break;
          case 'x':
              exit_required = TRUE;
              break;
          default:
              notify_selection_failure(selection[0]);
  }
} while (exit_required == FALSE);   printf("Fine!");
```

```
struct stack {
        int size;
        int TOS;
        float *buffer;
};
```

```
struct stack buffer;
```

```
Boolean push(struct stack *ptr, float value);
Boolean pop(struct stack *ptr, float *value_ptr);
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Rappresentazione di uno stack con struct (4)

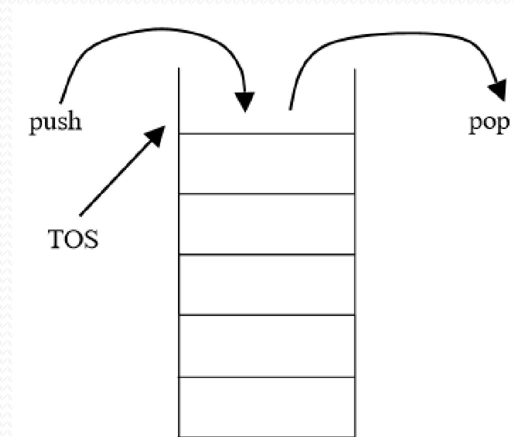
```
void getsize(int *size_ptr){
        //acquisisce dinamicamente la dimensione dello stack
        printf("\nDigita la dimensione dello stack:");
        scanf("%d", size_ptr);
}
```

```
struct stack {
        int size;
        int TOS;
        float *buffer;
};
```

```
void init(struct stack *ptr, int size) {
        //alloca il buffer e inizializza i campi TOS e size
        ptr->size = size;
        ptr->TOS = 0;
        ptr->buffer = (float *)malloc(size * sizeof(float));
}
```

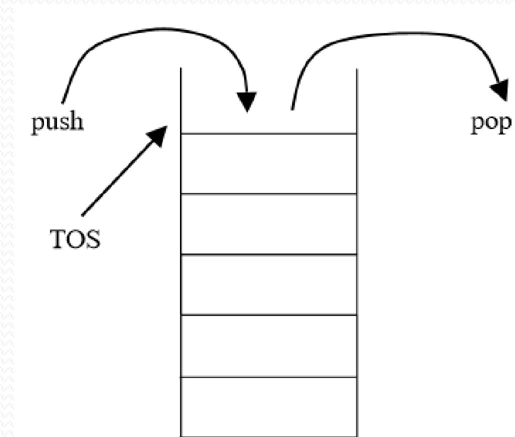
Rappresentazione di uno stack con struct (5)

```
Boolean push(struct stack *ptr, float value) {
//se il buffer NON è pieno inserisce value e restituisce TRUE,
//altrimenti rende FALSE
        if (ptr->TOS < ptr->size) {
                (ptr->buffer)[ptr->TOS] = value;
                (ptr->TOS)++;
                printf("Valore inserito nello stack!");
                return TRUE;
        }
        else
                return FALSE;
}
```



Rappresentazione di uno stack con struct (6)

```
Boolean pop(struct stack *ptr, float *value_ptr) {
//SE il buffer NON è vuoto, ne estrae un valore in
//*value_ptr e restituisce TRUE, altrimenti rende FALSE
        if (ptr->TOS>ptr->size) {
                (ptr->TOS)--;
                *value_ptr = (ptr->buffer)[ptr->TOS];
                return TRUE;
        }
        else {
                return FALSE;
        }
}
```



Rappresentazione di uno stack con struct (7)

```
 void getvalue(float *value_ptr) {
//acquisisce un float da tastiera (passaggio per puntatore)
         printf("Inserisci un valore: ");
         scanf("%f", value_ptr);
}

void putvalue(float value) {
         //stampa a video di un float
         printf("\nvalore estratto: %f", value);
}
```

Rappresentazione di uno stack con struct (8)

```
void notify_push_failure(void) {
        //notifica il fallimento di un push
        printf("\nBuffer pieno!! Inserimento cancellato");
}

void notify_pop_failure(void) {
        //notifica il falimento di un pop
        printf("\nBuffer Vuoto!! Impossibile estrarre elementi");
}

void notify_selection_failure(char selection) {
        //notifica il fallimento della selezione
        printf("\n%c Selezione non legale!!", selection);
}
```

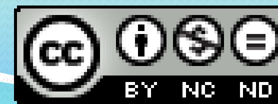
Rappresentazione di uno stack con struct (9)

C:\WINDOWS\system32\cmd.exe

```
Digita la dimensione dello stack:2
Digita:
-I per fare un inserimento nello stack
-O per estrarre un elemento dallo stack stack
-X per uscire dal programma

Che operazione vuoi fare? i
Inserisci un valore: 4
Valore inserito nello stack!
Che operazione vuoi fare? i
Inserisci un valore: 3
Valore inserito nello stack!
Che operazione vuoi fare? i
Inserisci un valore: 2

Buffer pieno!! Inserimento cancellato
Che operazione vuoi fare? _
```



Fondamenti di Informatica

Eng. Ph.D. Michela Paolucci

DISIT Lab *http://www.disit.dinfo.unifi.it/*
Department of Information Engineering, DINFO
University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570
michela.paolucci@unifi.it

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

Esempio: costrutto goto (1)

```
main() {
    int i=0;
    prima:
        printf("(i=%d)Vado ad eseguire la prima (incremento i di 1 e
        passo alla terza)\n", i);
        i++;
        goto terza;
    seconda:
        printf("(i=%d)Vado ad eseguire la seconda (incremento i di 2 e
        SE i<15 vado alla prima altrimenti vado alla fine)\n", i);
        i = i + 2;
        if (i < 15)
                goto prima;
        else
                goto fine;
    terza:
        printf("(i=%d)Vado ad eseguire la terza, poi passo alla seconda) \n",
    i);
        goto seconda;
    fine:;}
```

Esempio: costrutto goto (2)

C:\WINDOWS\system32\cmd.exe

```
(i=1)Vado ad eseguire la terza (vado alla seconda)
(i=1)Vado ad eseguire la seconda (incremento i di 2 e SE i<15 vado alla prima altrimenti vado alla fine)
(i=3)Vado ad eseguire la prima (incremento i di 1 e passo alla terza)
(i=4)Vado ad eseguire la terza (vado alla seconda)
(i=4)Vado ad eseguire la seconda (incremento i di 2 e SE i<15 vado alla prima altrimenti vado alla fine)
(i=6)Vado ad eseguire la prima (incremento i di 1 e passo alla terza)
(i=7)Vado ad eseguire la terza (vado alla seconda)
(i=7)Vado ad eseguire la seconda (incremento i di 2 e SE i<15 vado alla prima altrimenti vado alla fine)
(i=9)Vado ad eseguire la prima (incremento i di 1 e passo alla terza)
(i=10)Vado ad eseguire la terza (vado alla seconda)
(i=10)Vado ad eseguire la seconda (incremento i di 2 e SE i<15 vado alla prima altrimenti vado alla fine)
(i=12)Vado ad eseguire la prima (incremento i di 1 e passo alla terza)
(i=13)Vado ad eseguire la terza (vado alla seconda)
(i=13)Vado ad eseguire la seconda (incremento i di 2 e SE i<15 vado alla prima altrimenti vado alla fine)
Premere un tasto per continuare . . .
```