Programmazione

Giovanni Da San Martino

Dipartimento of Matematica, Università degli Studi di Padova
giovanni.dasanmartino@unipd.it
A.A. 2021-2022

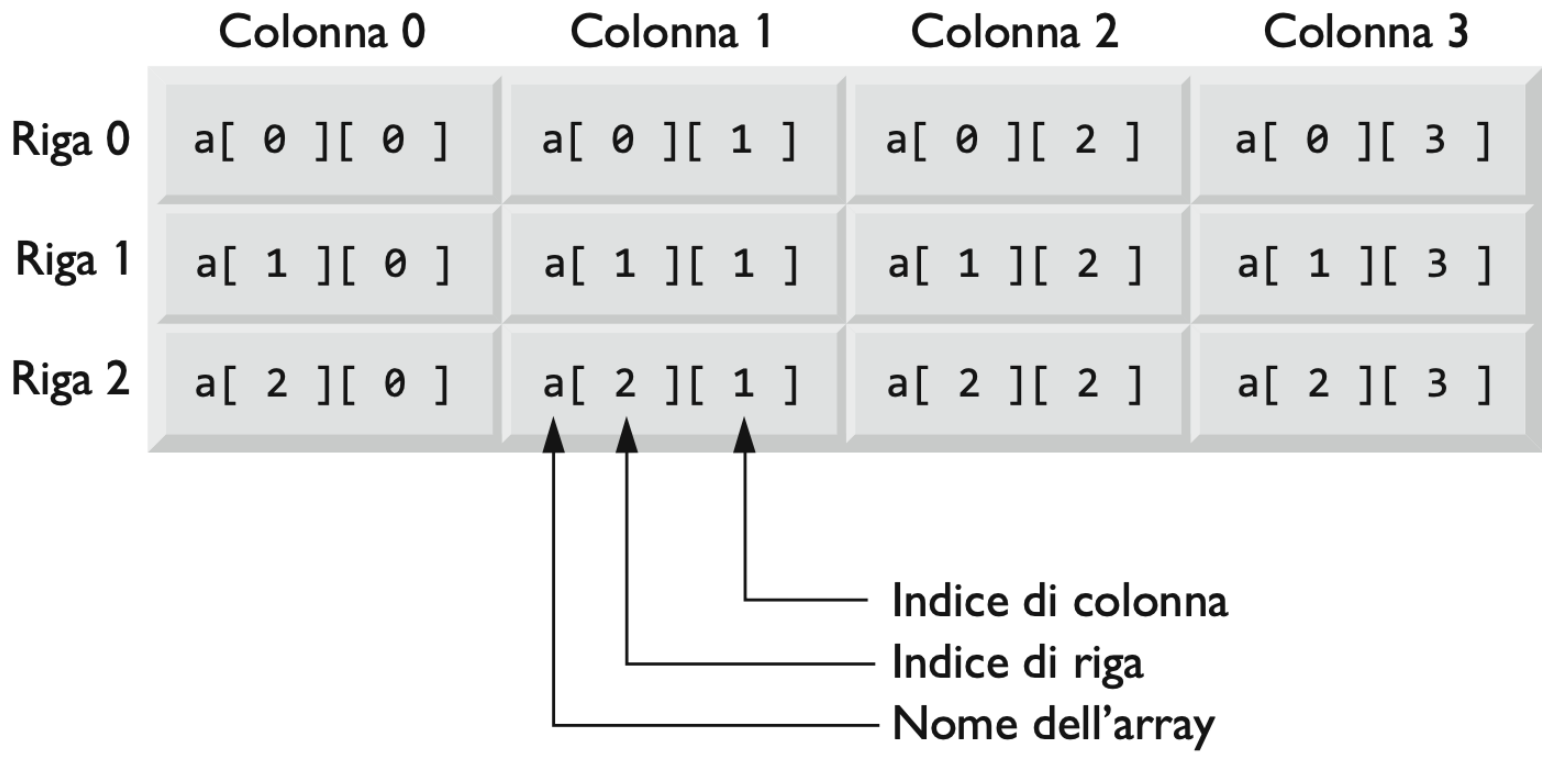
UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Array Multidimensionali

- Gli array possono avere più di una dimensione
  - Es. per rappresentare tabelle, matrici, od oggetti multidimensionali

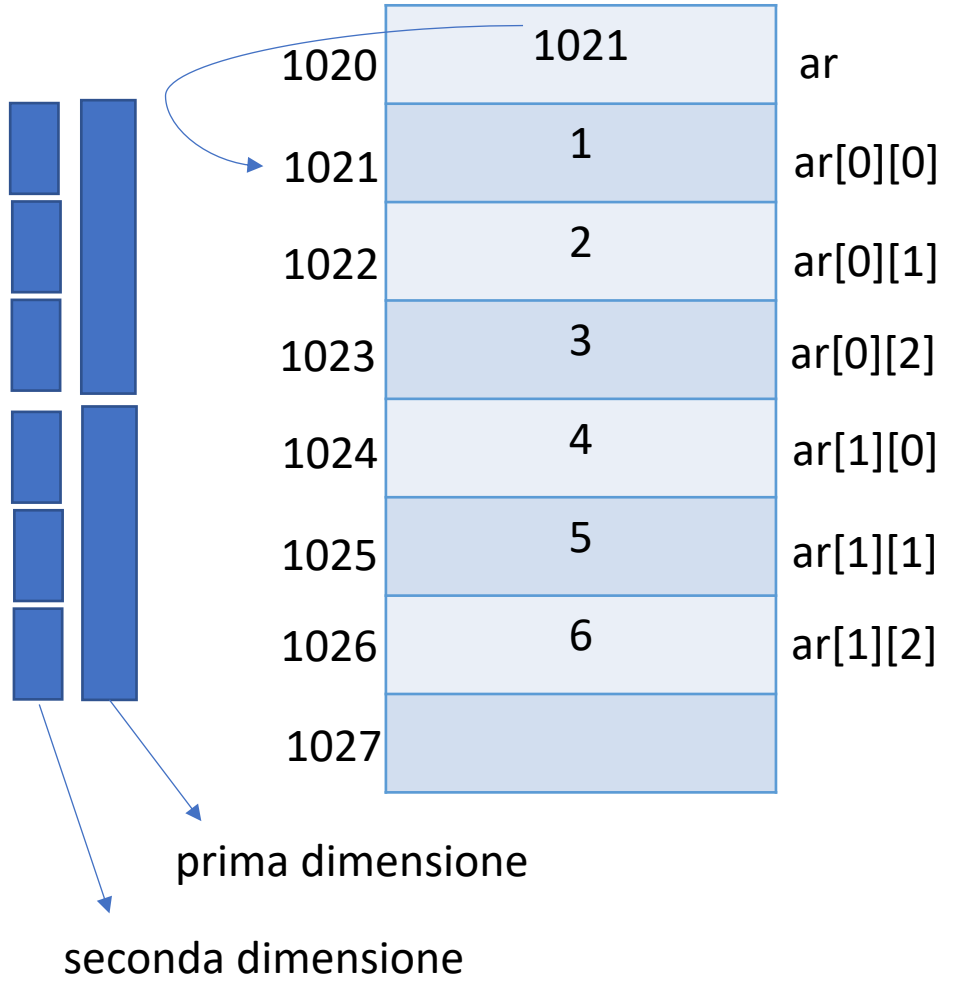
- tipo nome[indice1][indice2]...

- Es. int x[3][4];



Array Multidimensionali in Memoria

- int ar[][3] = { {1, 2, 3}, {4, 5, 6}}; // solo la prima dimensione può essere lasciata in bianco
- Perché le altre devono esserci? Se accedo a ar[1][0] devo sapere che devo saltare 3 elementi
- a[0] punta alla prima riga, a[1] alla seconda



Array Multidimensionali e Puntatori

- int ar[2][3] = { {1, 2, 3}, {4, 5, 6} };

- void stampa0(int a[][3], int righe);
- void stampa1(int righe, int colonne, int a[righe][colonne]);
- void stampa2(int* a[], int righe, int colonne);
- void stampa3(int* a, int righe, int colonne)

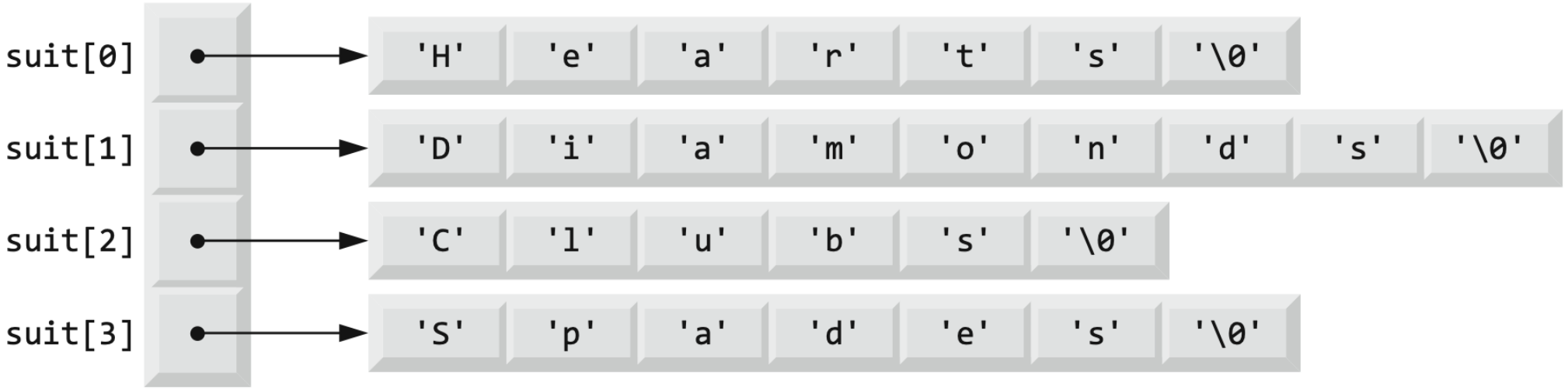
(vedi file stampa_matrice.c)

```
int *p[] = {a[0], a[1]};
int *pp = a[0];

stampa0(a,2);
stampa1(2,3,a);
stampa2(p,2,3);
stampa3(pp,2,3);
```

Array Multidimensionali e Puntatori

- I puntatori sono più generali, permettono di avere righe di dimensione diversa:
- char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spades"};



Array di Caratteri e Puntatori a Stringhe

Una stringa può essere rappresentata da

- un array di caratteri, se si aggiunge ‘\0’ alla fine.
  - La dichiarazione riserva un certo numero di celle di memoria. Quindi Si riesce a modificare un singolo carattere (s[0]=‘K’)
- un puntatore a char, a cui può essere assegnata una stringa costante, della quale non si possono modificare i caratteri. Si può riassegnare una seconda stringa al puntatore.

```
char s[8] = "Hearts";
char *ps = "Hearts";
printf("%s - %s\n\n", s, ps);
s[0] = 'K';
ps = "Ke";
printf("%s - %s\n", s, ps);
--Output--
Hearts - Hearts

Kearts - Ke
```

Esercizi

- Vero – Falso: una variabile puntatore ad intero occupa la stessa quantità di memoria di una variabile di tipo puntatore a double.

Esercizi

- Cosa stampa il codice seguente?

```
#include <stdio.h>
void quadrato(int x) {
  x = x*x;
}

int main (void) {
  int x[3] = {1,2,3};
  for(int i=0; i<3; i+=1) {
    quadrato(x[i]);
    printf(" %d", x[i]);
  }
}
```

Esercizi

- Cosa stampa il codice seguente?

```
void fun(int* a){
   a[1]=a[1]*2;
   a[2]=a[2]*2;
}
int main(void) {
   int x[]={0,1,2,3,4};
   fun(x+2);
   for(int i=0; i<5; i+=1) {
      printf(" %d", x[i]);
   }
   printf("\n");
}
```

Correttezza

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

PRE - POST Esempio


```
int radice_quadrata (float x) {
        //PRE: x>=0
        ....
        //POST: restituisce la radice quadrata di x
}
```

- La PRE e la POST sono le informazioni che servono a chi deve usare la funzione
  - La POST ci dice cosa calcola
  - La PRE, assieme al prototipo, come dobbiamo invocarla
  - Per chi usa la funzione, l'implementazione è irrelevante se la POST è dimostrata

PRE – POST e Correttezza

- Siamo liberi di scegliere la PRE come vogliamo, ma dovrebbe essere più generale possibile, affinché la nostra funzione sia utilizzabile in più scenari possibili.
  - Chi usa la funzione deve rispettare la PRE; chi la crea la assume vera
- La POST è una proprietà della funzione che indica ciò che calcola
- La correttezza di un frammento di codice (una funzione) si dimostra
  - assumendo vera la PRE
  - deducendo la POST dalle istruzioni del programma
- Se il programma non è corretto, basta fornire un esempio di input/output che non sia corretto

PRE - POST Esempio


```
void minimo(int x, int y, int z) {
   /* PRE: */
   printf("Il minore dei tre valori è ");
   if (x < y) { // so che x<y
      if (x < z) { // x < y e x < z
         printf("%d\n", x);
      } else { // z <= x < y
         printf("%d\n", z);
      }
   } else { // y <= x
      if (y < z) { // y < z e y <= x
         printf("%d\n", y);
      } else { // z <= y <= x
         printf("%d\n", z);
      }
   }
   /* POST: stampa "Il minore dei tre valori è a\n" dove a
è il valore minore tra x,y,z */
```

Corretto?


```
void minimo(int x, int y, int z) {
   /* PRE */
   if ( ( x <= y) && (x <= z) )
      printf("%d", x);

   if ( ( y <= x) && (y <= z) )
      printf("%d", y);

   if ( ( z <= x) && (z <= y) )
      printf("%d", z);
/* POST stampa un numero che è Il valore minimo tra le 3 variabili
```

Invariante di Ciclo

- L'invariante di un ciclo è una proprietà che è vera
  - prima,
  - durante e
  - dopo un ciclo.
- Si usa per dimostrare la POST quando il codice ha un ciclo

```
i=0
// invariante vera qua
while (i<N) {
   // invariante vera qua
   printf("%d\n", i)
}
// invariante vera qua
```

Invariante di Ciclo: Esempi


```
int somma(int *X, int size) {
    //PRE l'array X ha dimensione size
    int sum = 0;
    for (int i=0; i<size; i=i+1) {
        //INV sum=0+X[0]+...+X[i-1]
        sum += X[i];
    }
    return sum;
    //POST i=size. sum=X[0]+...+X[size-1] (restituisce la somma dei valori dell'array)
}
```

Invariante di Ciclo

- L'invariante di un ciclo è una proprietà che è vera prima, durante e dopo un ciclo. Si usa per dimostrare la POST quando il codice ha un ciclo

```
int minimo(int *X, int size) {
    //PRE l'array X ha dimensione size
    int min = X[0];
    for (int i=1; i<size; i=i+1) {
        //INV per ogni 0<=j<i. min<=X[j] (min è il minimo tra X[0] e X[i])
        if(X[i]<min)
            min = X[i];
    }
    return min;
    //POST per ogni 0<=j<size. min<=X[j] (min è il minimo dell'array)
}
```

Esercitazione 1


```
* Implementare un'applicazione per cifrare/decifrare messaggi.
*
* - Cifratura -
*
* La cifratura avviene modificando solamente i caratteri
* alfanumerici (ovvero le cifre 0-9, le lettere minuscole
* a-z, le lettere maiuscole A-Z) del messaggio nel
* modo seguente: dato il valore numerico
* corrispondente ad un carattere (es. a=97, A=65), e dato un
* intero k!=0, la cifratura di un carattere avviene sommando k
* al valore numerico del carattere.
* Ad esempio se k=2 'a' viene trasformata in 'c', 'B' in 'D', '1'
* in '3', '!' in '!' (perché non è un carattere alfanumerico).
```

Esercitazione 1

* Inoltre si deve assicurare che ogni cifra
* sia trasformata in una cifra, ogni lettera minuscola in
* una lettera minuscola, ogni lettera maiuscola in una
* lettera maiuscola. Ciò si ottiene considerando 'a' il
* carattere seguente di 'z', 'A' quello di 'Z', '0' quello di '9'.
* Ad esempio se k=3 'y' viene trasformata in 'b', 'Z' in 'C',
* '7' in '0'.

Esercitazione 1: Schermate di Esempio


```
--------------------
Benvenuto in CrittApp
--------------------

Menu Principale:

1) specificare la chiave di cifratura
2) codifica un messaggio
3) decodifica un messaggio
4) esci

Indicare il carattere corrispondente all'opzione scelta e premere Invio: 1
Inserisci un intero diverso da 0: 2
nuova chiave: 2

Menu Principale:
```

Esercitazione 1: Schermate di Esempio

Menu Principale:

1) specificare la chiave di cifratura
2) codifica un messaggio
3) decodifica un messaggio
4) esci

Indicare il carattere corrispondente all'opzione scelta e premere Invio: 2

Inserisci un messaggio non più lungo di 20 caratteri, seguito da Invio: ciao

Messaggio codificato: ekcq

Menu Principale:

Esercitazione 1: Schermate di Esempio

Menu Principale:

1) specificare la chiave di cifratura
2) codifica un messaggio
3) decodifica un messaggio
4) esci

Indicare il carattere corrispondente all'opzione scelta e premere Invio: 4

Criteri di Valutazione

Dovrete valuater il codice dei vostri compagni.

Leggete attentamente la consegna e cercate di pensare il codice rispetto ai seguenti criteri (quando applicabili)

- Correttezza (le funzioni devono essere commentate con PRE POST)
- Efficienza: evitare codice inutile, cercare di trovare l'algoritmo più efficiente per risolvere i sottoproblemi
- Organizzazione del codice: divisione logica del codice in funzioni. Evitare di risolvere una seconda volta problemi già risolti
- Stile: il codice deve essere leggibile, evitare istruzioni non spiegate da me, commentare i frammenti di codice che non siano ovvi, usare nomi significativi per le variabili