Programmazione

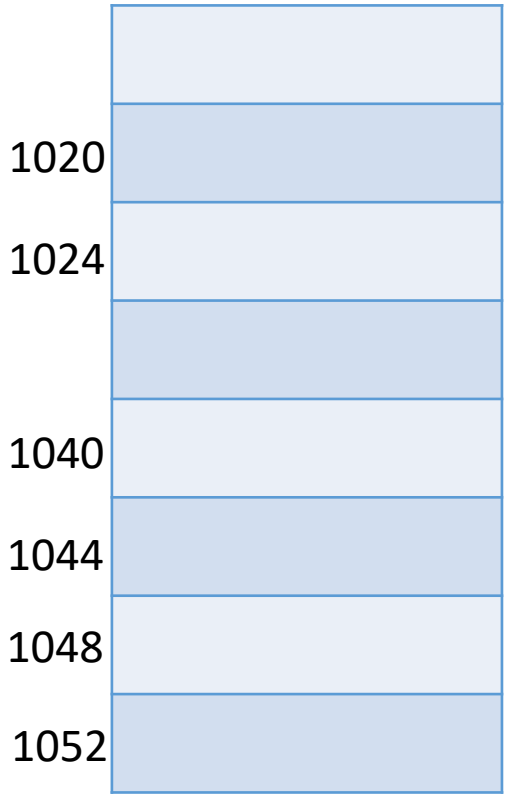
Giovanni Da San Martino

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

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

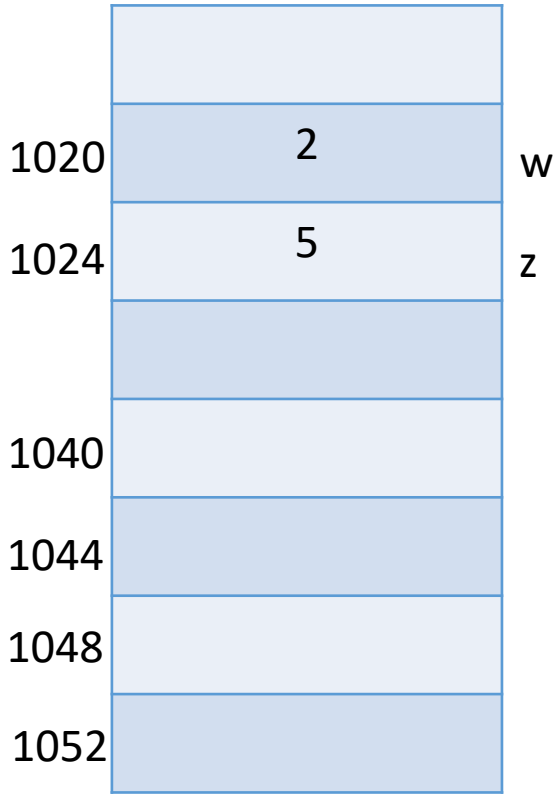
Previuosly on Programmazione


```
void scambio(int *x, int *y) {// int *x = &w; int *y = &z
        int temp = *x;
        *x = *y;
        *y = temp;
}
int main() {
        int w=2, z=5;
        scambio(&w, &z);
        printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



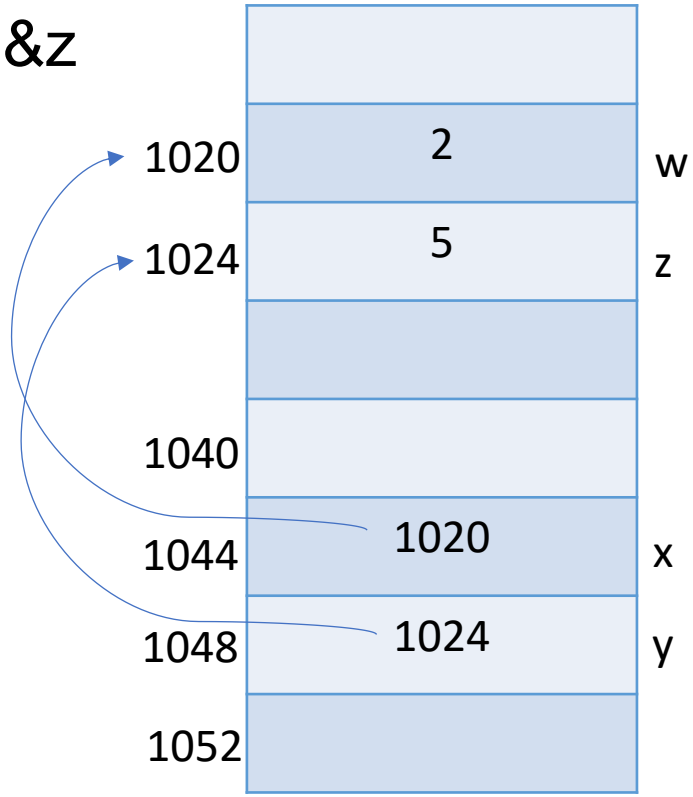
Passaggio di Parametri in C


```
void scambio(int *x, int *y) {// int *x = &w; int *y = &z
        int temp = *x;
        *x = *y;
        *y = temp;
}
int main() {
        int w=2, z=5;
        scambio(&w, &z);
        printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



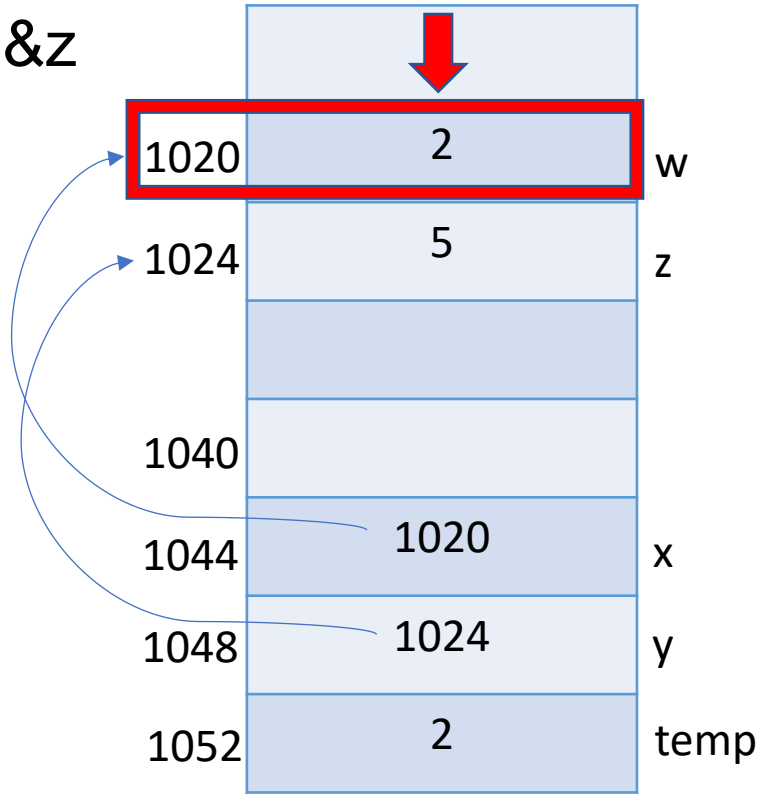
Passaggio di Parametri in C

```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
        int temp = *x;
        *x = *y;
        *y = temp;
  }


int main() {
        int w=2, z=5;
        scambio(&w, &z);
        printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



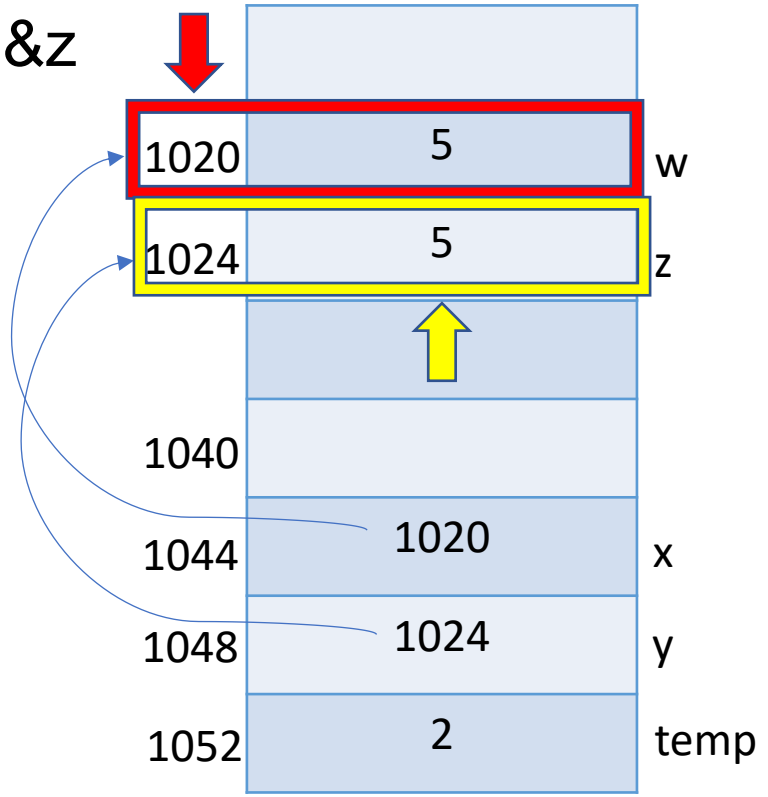
Passaggio di Parametri in C

```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
        int temp = *x;
        *x = *y;
        *y = temp;
}


int main() {
        int w=2, z=5;
        scambio(&w, &z);
        printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



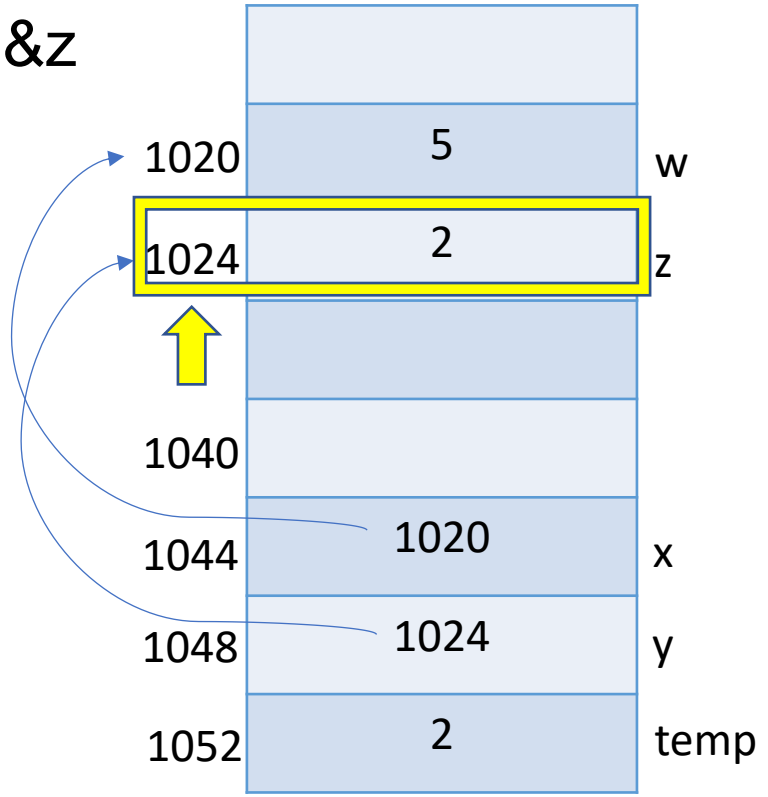
Passaggio di Parametri in C

```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
        int temp = *x;
        *x = *y;
        *y = temp;
}


int main() {
        int w=2, z=5;
        scambio(&w, &z);
        printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



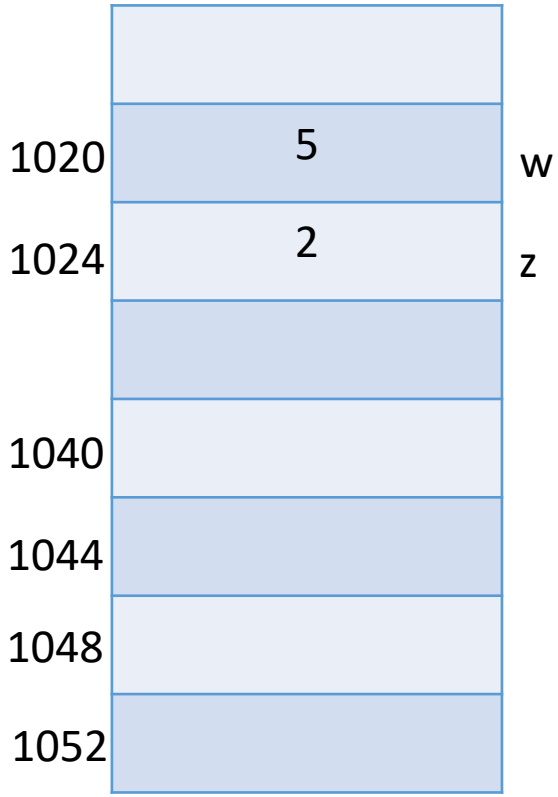
Passaggio di Parametri in C

```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
        int temp = *x;
        *x = *y;
        *y = temp;
}


int main() {
        int w=2, z=5;
        scambio(&w, &z);
        printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



Passaggio di Parametri in C


```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
        int temp = *x;
        *x = *y;
        *y = temp;
}


int main() {
        int w=2, z=5;
        scambio(&w, &z);
        printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



Passaggio di Parametri in C: Array

- I due prototipi seguenti sono equivalenti ed intercambiabili:

```
void f(int *array);
void f(int array[]);
```

- int x[5];
- x == &x[0] // puntatore al primo elemento dell'array
- int *p = x; // corretto, equivale a &x[0]
- f(x); f(p); // sono corrette per entrambi i prototipi
- Un array viene sempre modificato all'interno di una funzione! Perché questa scelta?

Puntatori

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Dangling Reference

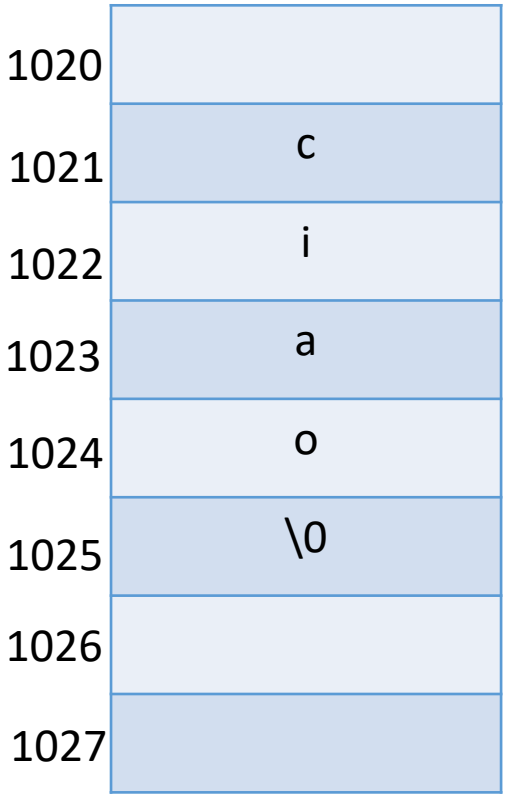
- Una funzione può restituire un puntatore
- Bisogna stare però attenti: le variabili create all'interno della funzione vengono deallocate (distrutte) al termine dell'esecuzione
- anche min viene deallocata (il contenuto della cella di memoria potrebbe essere utilizzato per creare altre variabili)

```
int a[] = {1,2,3,4}
p = min(a, 4);
printf("\n%d\n", *p);
```

```
int * min(int*X, int dimX) {
    int i=1, min=X[0];
    while(i < dimX) {
        if(X[i] < min)
            min=X[i];
        i=i+1;
    }
    return &min;
}
```

Stringhe (Sequenze di Caratteri)

- Una sequenza di caratteri viene anche chiamata stringa. Es.
- char string1[] = ”ciao";
- In memoria viene memorizzato come “ciao\0”, per cui serve un array con n+1 elementi, dove n è la lunghezza della stringa (5 nell’esempio)
- il carattere \0 indica la fine della stringa
- il primo elemento di string1 è un char *



Direttive al Preprocessore

- La prima fase della traduzione da codice C a linguaggio macchine è eseguita dal preprocessore che elimina i commenti e
- crea costanti simboliche:
- #define identificatore testo_di_sostituzione
- c'è solo uno spazio tra l'identificatore e il testo sostituito (non c'è ; dopo!)

```
#define PI 3.1415
int main(void) {
        float x = 2*PI;
}
```


```
int main(void) {
        float x = 2*3.1415;
}
```

Operatori Logici

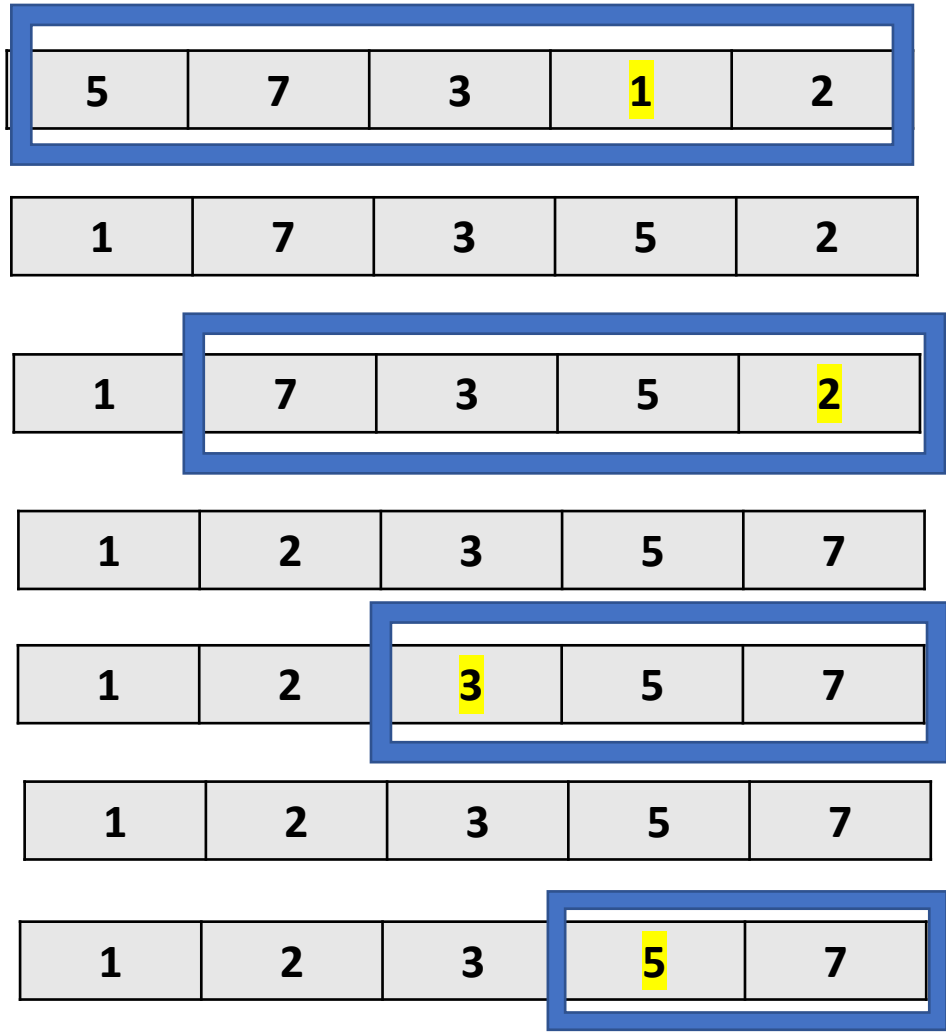
Operatore	Significato	Esempio
&& (binario)	AND logico. Vero solo se entrambi gli operandi sono Veri	((5==5) && (5>2)) è Falso.
|| (binario)	OR logico. Vero se almeno uno degli argomenti è Vero	((5==5) || (5>2)) è Vero

- && e || sono valutati in modo lazy (pigro).
  - Se A e B sono due condizioni per (A && B) e (A || B) si valuta sempre prima A:
  - se A è falsa, A && B restuisce Falso senza valutare B
  - se A è vera, A || B restituisce Vero senza valutare B

Esempio: Ordinamento

- Consegna: Ordinare un array di interi
- Idea:
  1. trovare il minimo tra gli elementi dell'array X[i:5] (dalla posizione i inclusa a 5 esclusa)
  2. scambiarlo con l'elemento in posizione i (adesso l'array X[0:i] è ordinato)
  3. Ripetere per 0<=i<=3

(vedere ordina_array.c)

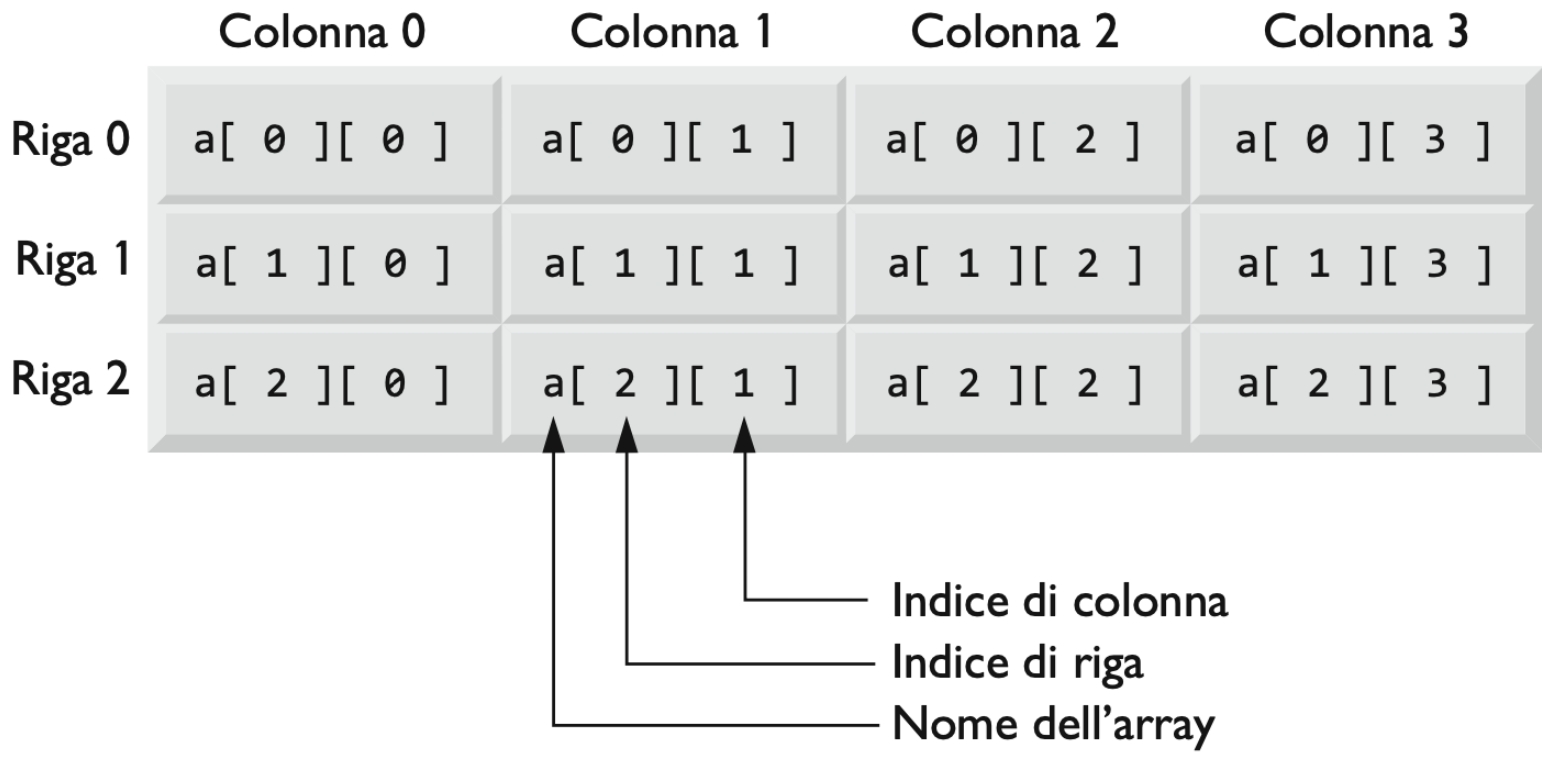


Array Multidimensionali

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

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

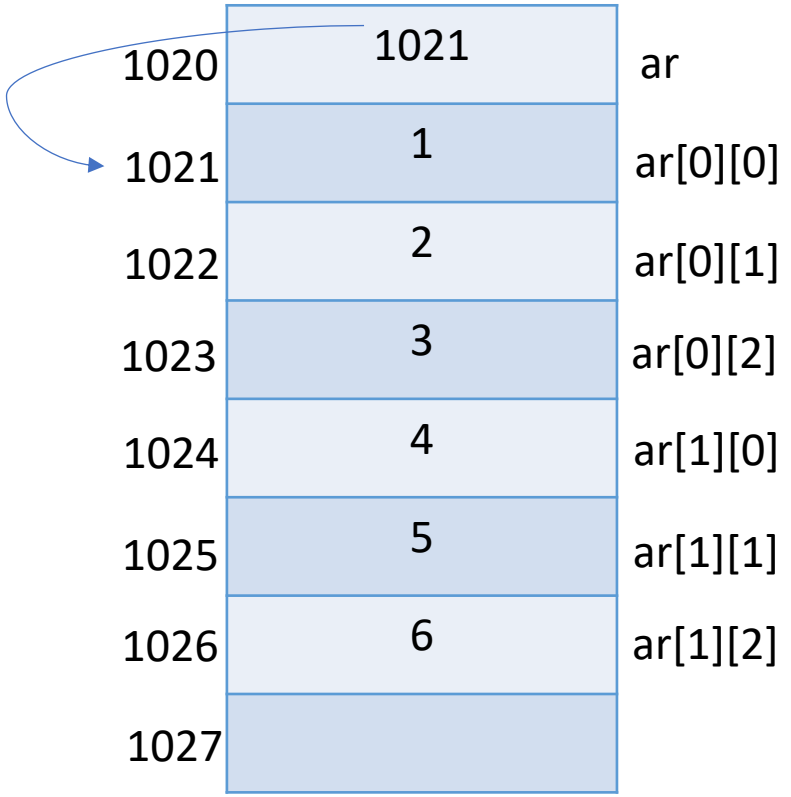
- Es. int x[3][4];



Array Multidimensionali

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

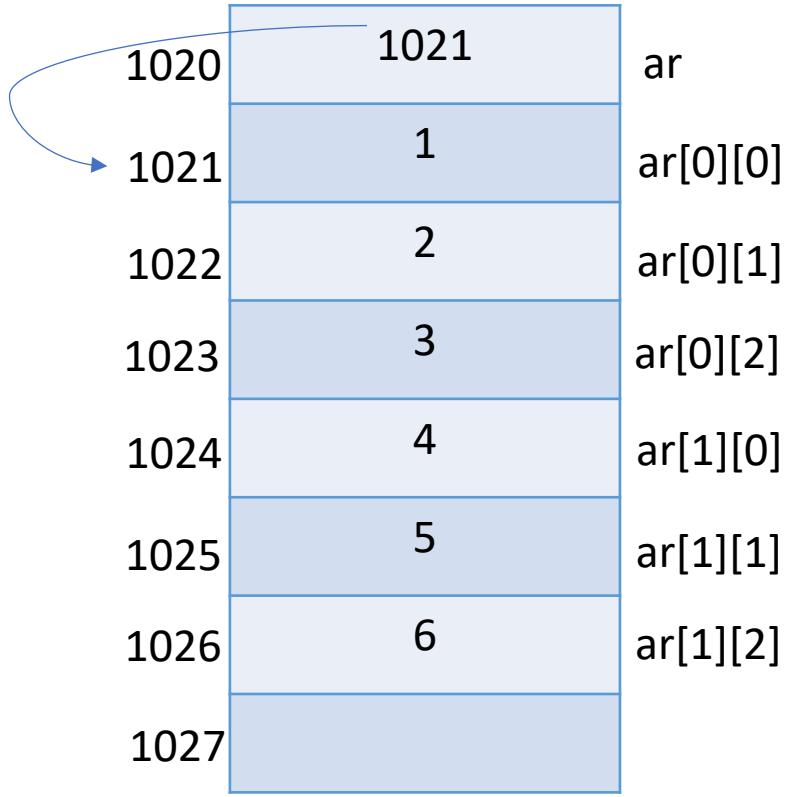
  es. a[1][0] {0 [ ] 1 [ ] }



Array Multidimensionali

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

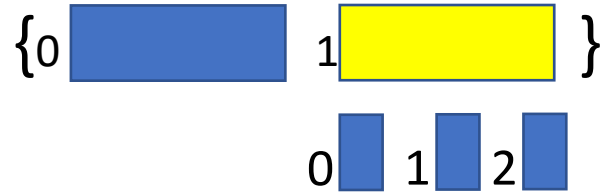
  es. a[1][0] {0  1  }

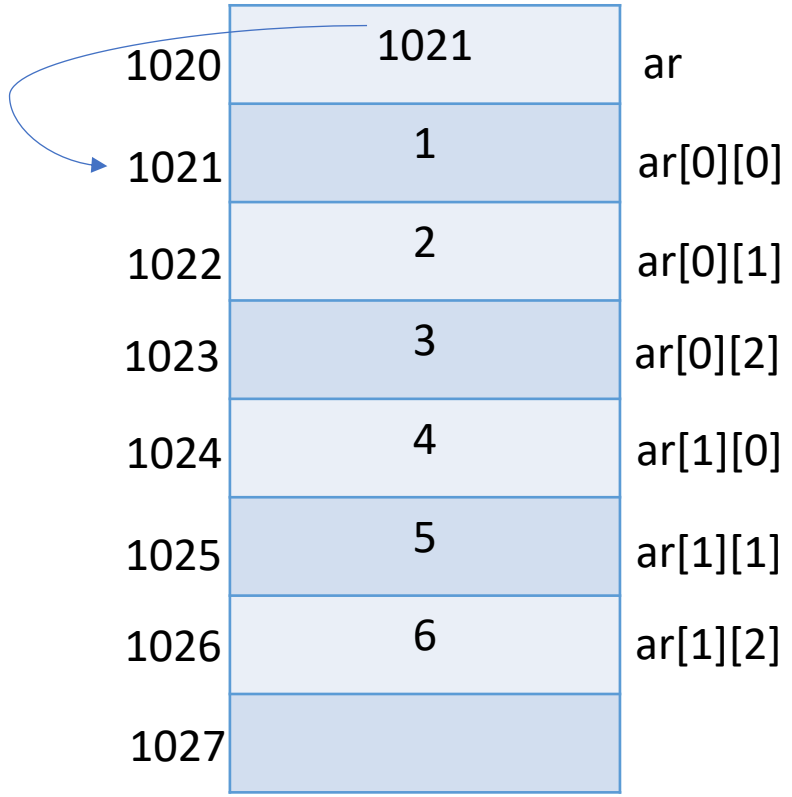


Array Multidimensionali

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

  es. a[1][0]





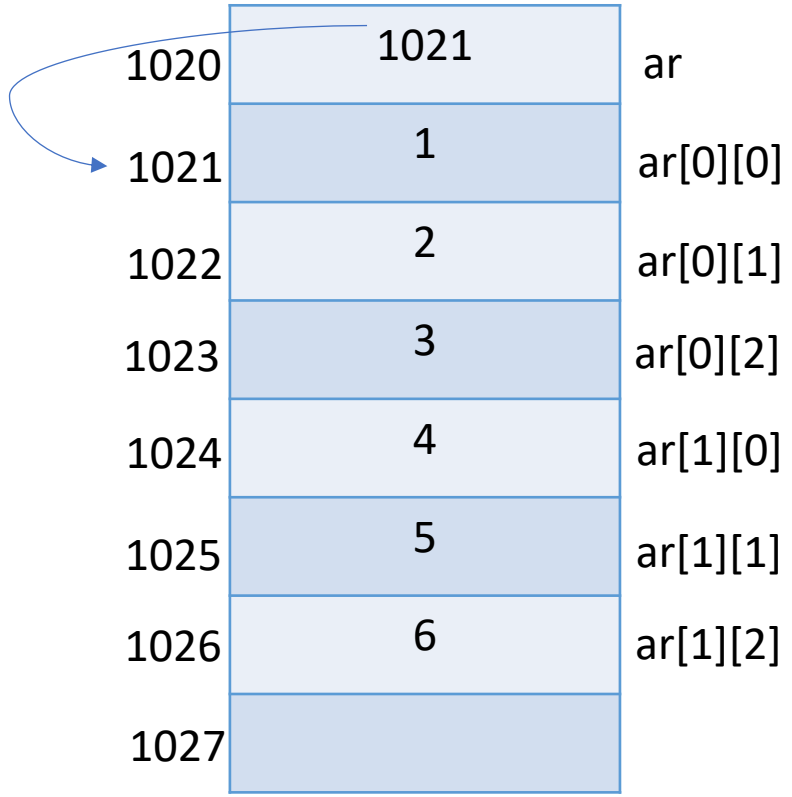
Array Multidimensionali

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

  es. a[1][0] {0  1  }

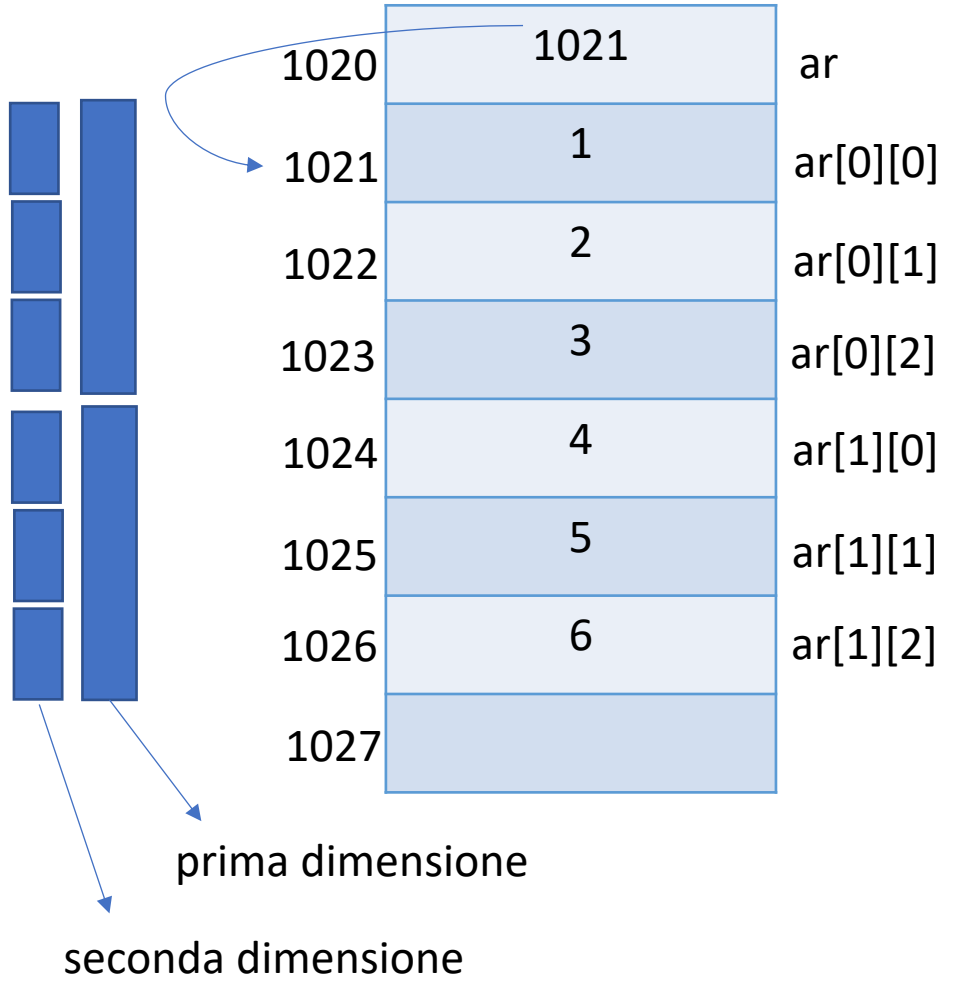
  0  1  2 

- Anche gli array multidimensionali sono salvati come una sequenza contigua di variabili, riga per riga. Le seguenti inizializzazioni sono equivalenti:
- int ar[2][3] = {1, 2, 3, 4, 5, 6};
- int ar[][3] = {1, 2, 3, 4, 5, 6}; // solo la prima dimensione può essere lasciata in bianco



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 (ar è solo un puntatore al primo elemento dell'array)!
- Gli stessi ragionamenti si applicano ad array di più dimensioni: int *X[2][3][4];



Correttezza

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

PRE and POST Condizioni

- Fino ad ora abbiamo detto che un programma è corretto se realizza la consegna.
- Cerchiamo di essere più formali. Anche se si può dimostrare la correttezza di frammenti di codice o interi programmi, generalmente di dimostra la correttezza di una funzione
- La Precondizione (PRE) indica ciò che si assume vero prima della chiamata della funzione
- La Postcondizione (POST) indica ciò che sarà vero dopo che la funzione sarà eseguita. La POST deve essere utile per dimostrare la consegna

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


```
int radice_quadrata (float x) {
   //PRE: x==9
   return 3;
   //POST: restituisce la radice quadrata di x
} // corretta ma non interessante, la funzione può essere applicata solo al numero 9!
```

```
int radice_quadrata (float x) {
    //PRE:
          if (x<0) {return -1;}
          ....
   //POST: se x<0 restituisce -1, altrimenti restituisce la radice quadrata di x
}
```

PRE - POST Esempio

- Al posto dell'if possiamo usare la funzione assert, che genera un errore se la condizione al suo interno è falsa e termina il programma

```
#include <assert.h>
int radice_quadrata (float x) {
    //PRE: x>=0
        assert(x>=0) ;
        ....
   //POST: restituisce la radice quadrata di x
}
```

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 */
```

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];
    //INV per ogni 0<=j<=i. min<=X[j] (min è il minimo tra X[0] e X[i])
    for (int i=1; i<5; i=i+1) {
        if(X[i]<min)
            min = X[i];
    }
    return min;
    //POST per ogni 0<=j<=size. min<=X[j] (min è il minimo dell'array)
}
```

Invariante di Ciclo: Esempi


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

Invariante di Ciclo: Esempi


```
int trova_minimo(int *ar, int size); // PRE ar è un array di interi di dimensione size
                    //POST per ogni 0<=j<=size. min<=X[j] (restituisce il minimo dell'array)

void ordina(int *X, int size) {

  int indice_min, i=0;
  //INV per ogni 0<=j<i. ar[j]<ar[j+1]
  for(i=0; i<size-1; i+=1) {
        indice_min = trova_minimo(X+i, size-i); //per j. i<=j<size. X[indice_min]<=X[j]
        scambia(X+i, &X[indice_min+i]);       //X[i]=X[indice_min]; X[indice_min]=X[i]
   }
}
```