Laboratorio di Programmazione

Lezione 6 – Funzioni e Array Multidimensionali

Marco Anisetti (teoria)
Dipartimento di Informatica
marco.anisetti@unimi.it

Matteo Luperto (lab. turno A)
Dipartimento di Informatica
matteo.luperto@unimi.it

Nicola Bena (lab. turno B)
Dipartimento di Informatica
nicola.bena@unimi.it

Funzioni

- Una funzione è un sottoprogramma che svolge uno specifico compito

- Quando il codice diventa complesso, ~~è utile~~ occorre spezzarlo in diversi sottoprogrammi
  - Supporta il paradigma divide-et-impera
  - Migliore manutenzione (modularità e riuso)

- Ogni programma `C` è composta da almeno una funzione: `int main()`

Funzioni

- **Prototipo**: nome della funzione, i dati in input (parametri formali), e il tipo del risultato in output

```
void my_func()
```

- **Definizione**: prototipo + corpo (istruzioni)

```
void my_func() { /* codice */ }
```

- **Chiamata**: il codice della funzione viene eseguito sugli specifici dati in ingresso (parametri attuali)

```
int main() {
   my_func();
}
```

Funzioni

- Step 1: **dichiarazione**: informa il compilatore sulla presenza di una funzione con quel prototipo la cui implementazione sarà specificata in altre parti del codice/librerie

```
void my_func();
```

- Step 2: **definizione**

```
void my_func(){
   /* codice */
  }
```

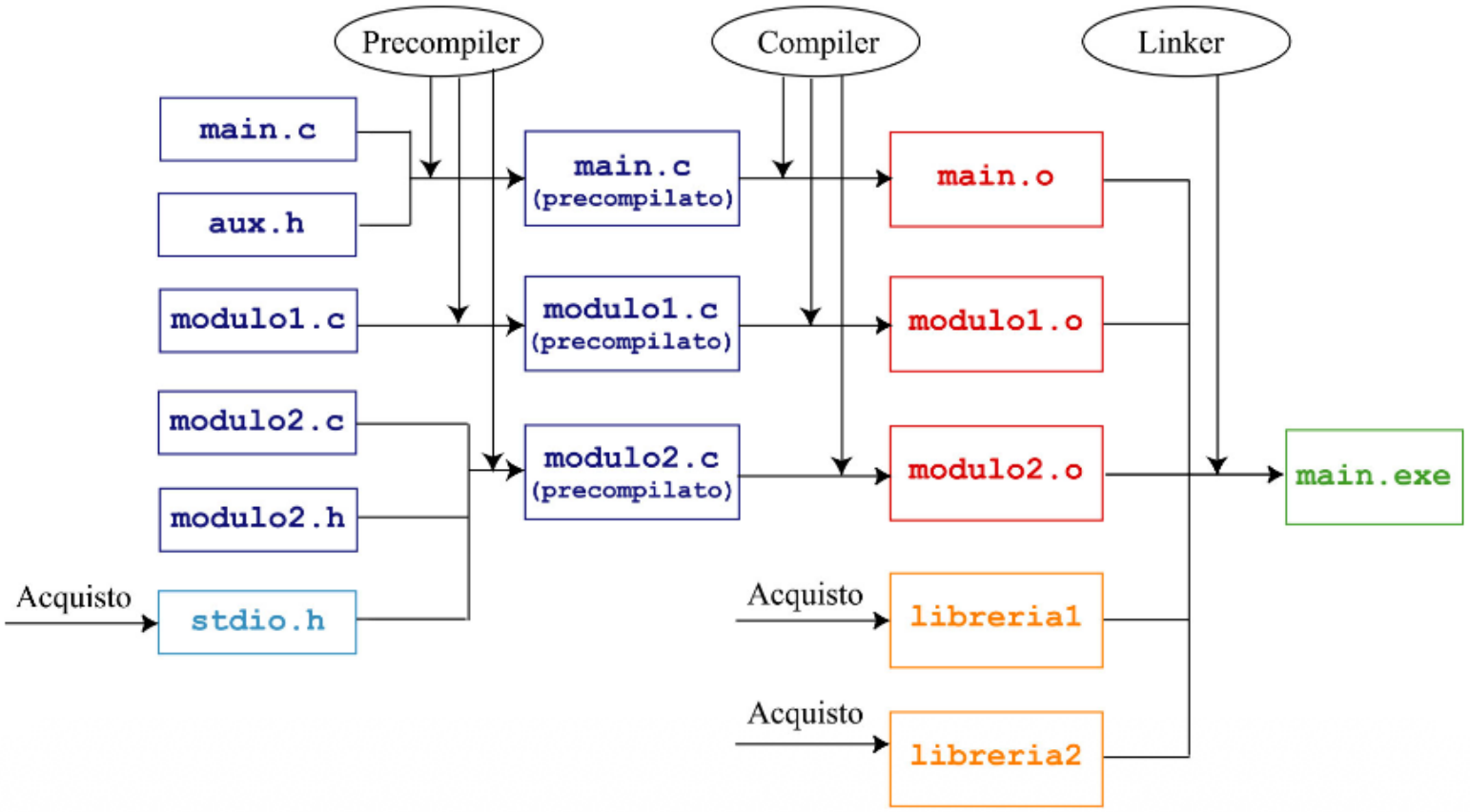
- Step 3: **chiamata**

```
int main() {
   my_func();
}
```

Prototipi

- La dichiarazione viene usata per istruire il compilatore sulla presenza di funzioni che non si trovano nel codice sorgente che sta compilando
- Librerie
  - File `.h` contengono i prototipi e vengono importati nel codice dal preprocessore
  - File `.o` contengono la definizione delle funzioni e vengono aggiunte all'eseguibile finale dal linker
- Astrazione: il programmatore conosce solo il nome della funzione, i dati che servono in ingresso, i dati che restituisce. Non serve sapere l'implementazione.

Librerie e Compilazione



Credits: Prof. Roberto Cordone

Funzioni

La dichiarazione viene usata per istruire il compilatore sulla presenza di funzioni che non si trovano nel codice sorgente che sta compilando

- Step 1: definizione della funzione

```
void my_func(){
   /* codice */
}
```

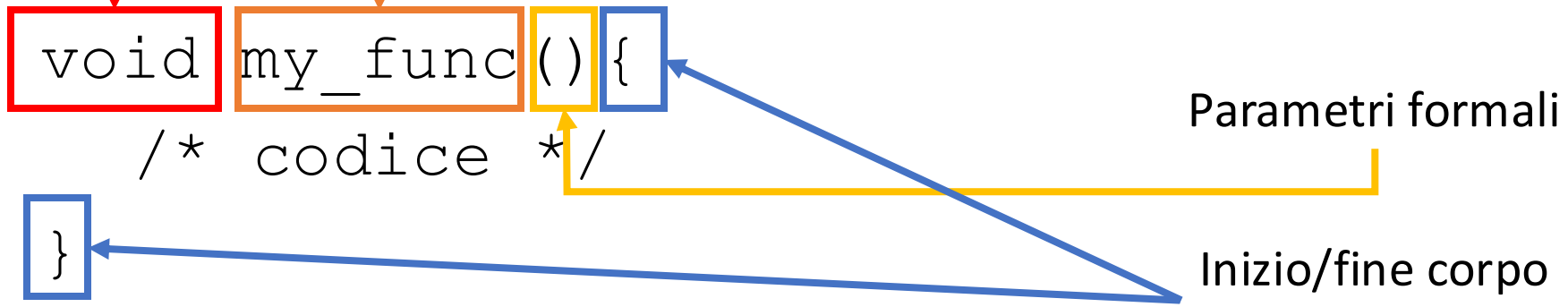
- Step 2: utilizzo della funzione:

```
int main() {
   my_func();
}
```

Funzioni

- Step 1: definizione della funzione

```
void my_func(){
    /* codice */
}
```



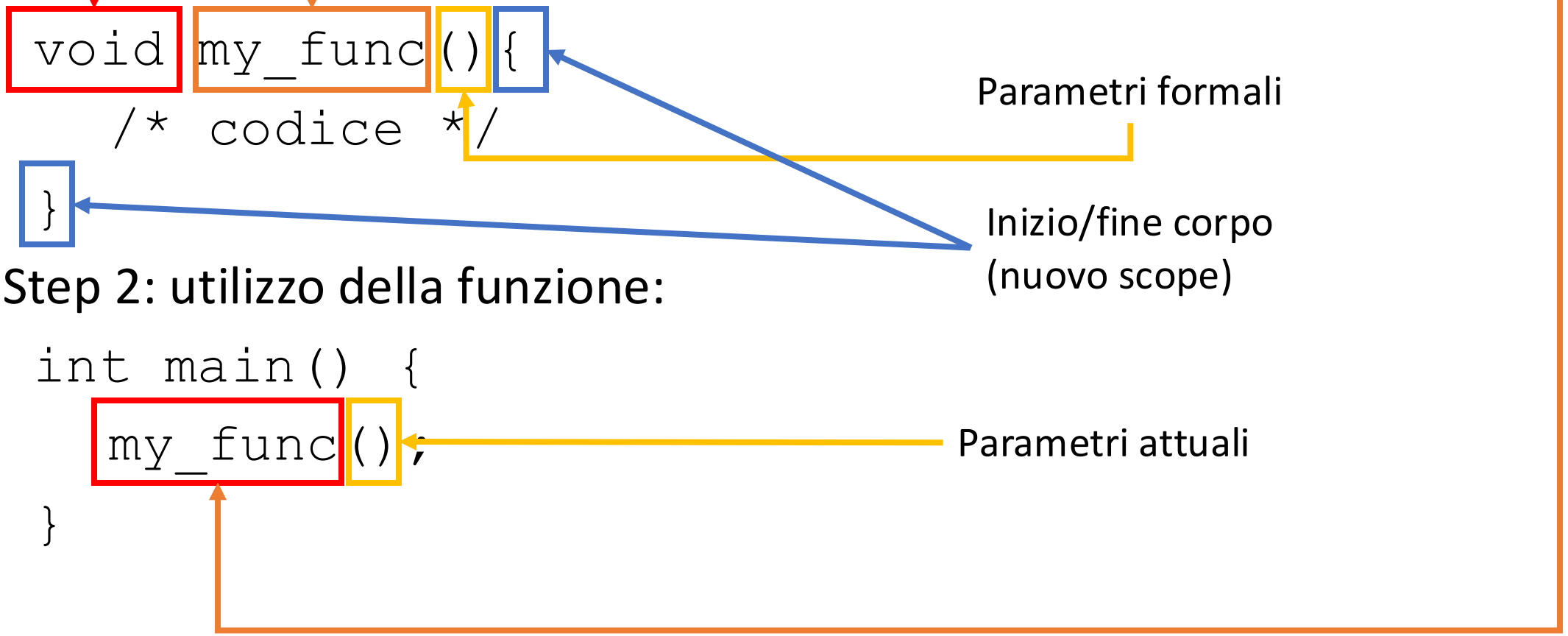
- Step 2: utilizzo della funzione:

```
int main() {
   my_func();
}
```

Funzioni

- Step 1: definizione della funzione

```
void my_func(){
    /* codice */
}
```



- Step 2: utilizzo della funzione:

```
int main() {
    my_func();
}
```

Funzioni

- Step 1: definizione della funzione

```
int my_func(){
   int result = ...;
   return result;
}
```

- Step 2: utilizzo della funzione:

```
int main() {
   int result = my_func();
}
```

Funzioni

- Step 1: definizione della funzione

```
int my_func(){
    int result = ...;
    return result;
}
```

- Step 2: utilizzo della funzione:

```
int main() {
    int result = my_func();
}
```

La funzione deve ritornare un valore di tipo `int`

Una funzione chiamata è un'espressione, il cui valore corrisponde al valore specificato dopo `return` → `return` è l'ultima istruzione eseguita, specifica di tornare alla funzione chiamante

Salviamo il valore restituito in una variabile nella funzione chiamante dello stesso tipo del valore restituito (se serve)

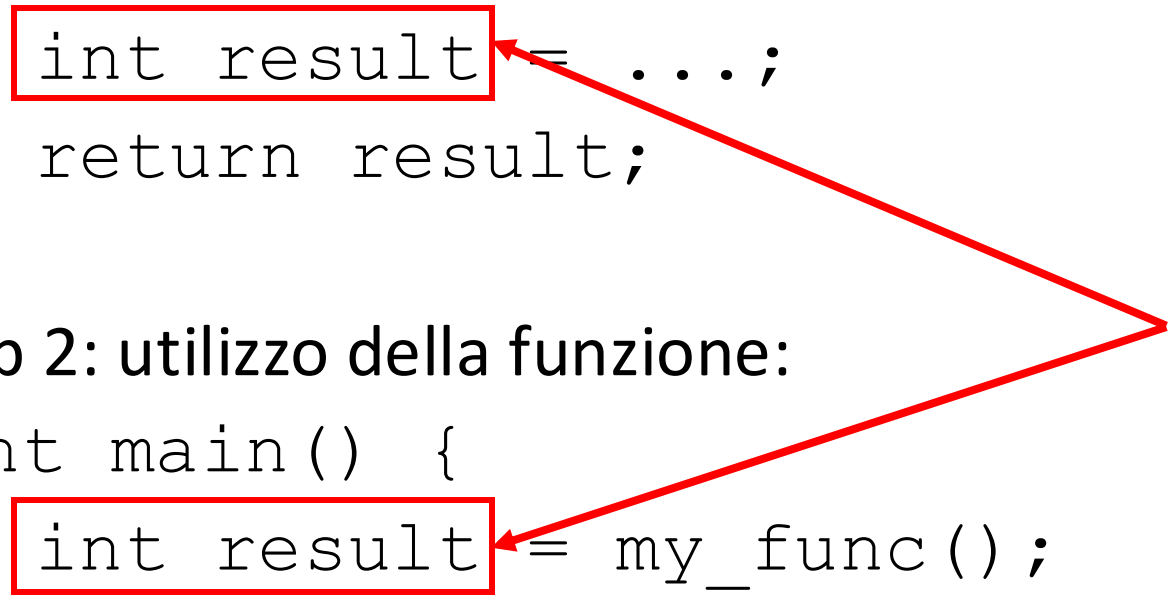
Funzioni

- Step 1: definizione della funzione

```
int my_func(){
   int result = ...;
   return result;
}
```

- Step 2: utilizzo della funzione:

```
int main() {
   int result = my_func();
}
```



Le variabili dichiarate all'interno del corpo di una funzione hanno visibilità all'interno dello scope della funzione

Sono due variabili diverse!

Funzioni

- Passaggio di parametri in ingresso
  - **Passaggio per valore**: il *record di attivazione* della funzione contiene una **copia** dei dati passati in ingresso → eventuali modifiche sono locali allo scope della funzione
  - **Passaggio per indirizzo**: il record di attivazione della funzione contiene **i dati** passati in ingresso → eventuali modifiche si riflettono anche all'esterno
- C supporta solo il passaggio per valore, e «simula» il passaggio per indirizzo tramite i puntatori
  - Attenzione: eventuali modifiche hanno effetto anche al di fuori del record di attivazione della funzione!
  - Se una variabile è passata per indirizzo, non si possono impedire operazioni non consentite sui dati. Come proteggere i dati? Lo vedremo in Java.

Funzioni

```
int sum(int a, int b){
     return a+b;
}

int main(){
     int n1 = 10;
     int n2 = 20;
     int result;

     result = sum(n1, n2);
}
```

Funzioni

```
int sum(int a, int b){
      return a+b;
}

int main(){
      int n1 = 10;
      int n2 = 20;
      int result;

      result = sum(n1, n2);
}
```

Elenco dei parametri in ingresso nella forma
`<tipo> <identificatore>`

I parametri sono utilizzabili (solo)
nel corpo della funzione

Parametri attuali nella forma
`<nome-funzione>(<espressione>, …)`

in questo caso `<espressione>` è
l'identificatore di una variabile

Funzioni

```
int sum(int a, int b){
     return a+b;
}

int main(){
     int n1 = 10;
     int n2 = 20;
     int result;

     result = sum(n1, n2);
}
```

Quando la funzione `main` chiama la funzione `sum`

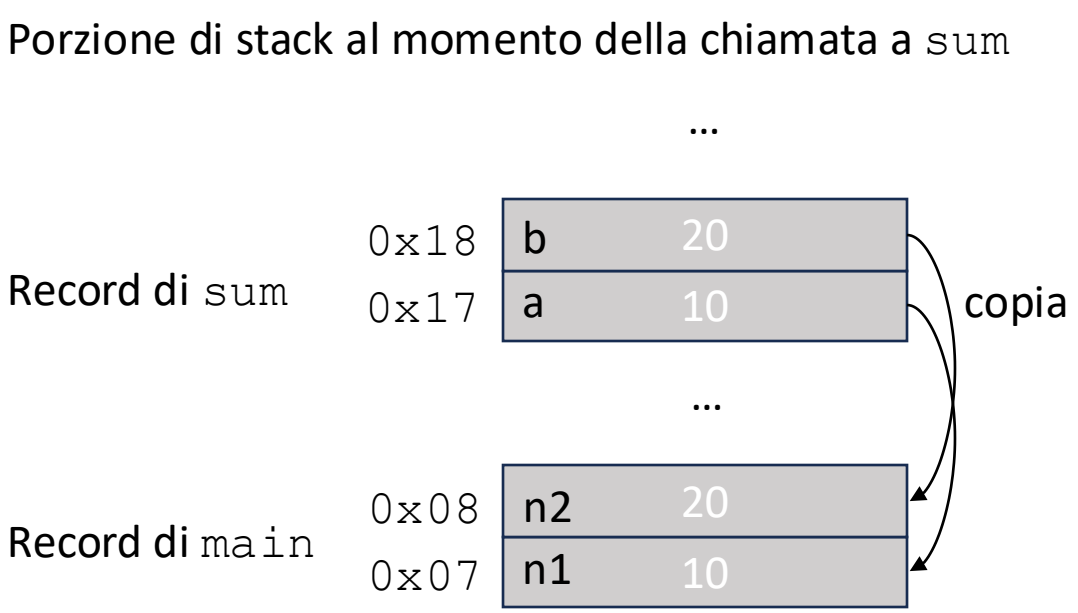
1. Valutazione dei parametri attuali (valore di `n1` e `n2`)
2. Creazione e allocazione dello spazio per il nuovo record di attivazione
3. Assegnamento del valore dei parametri attuali ai parametri formali (copia)
   - Il valore di `n1` viene copiato nella cella di memoria del record di attivazione con identificatore `a`
   - Il valore di `n2` viene copiato nella cella di memoria del record di attivazione con identificatore `b`
4. Le istruzioni contenute in `sum` vengono eseguite, tenendo traccia del valore dell'espressione che segue `return`
5. Il valore al punto 4. viene assegnato alla variabile `result`

Funzioni

```
int sum(int a, int b){
     return a+b;
}

int main(){
     int n1 = 10;
     int n2 = 20;
     int result;

     result = sum(n1, n2);
}
```



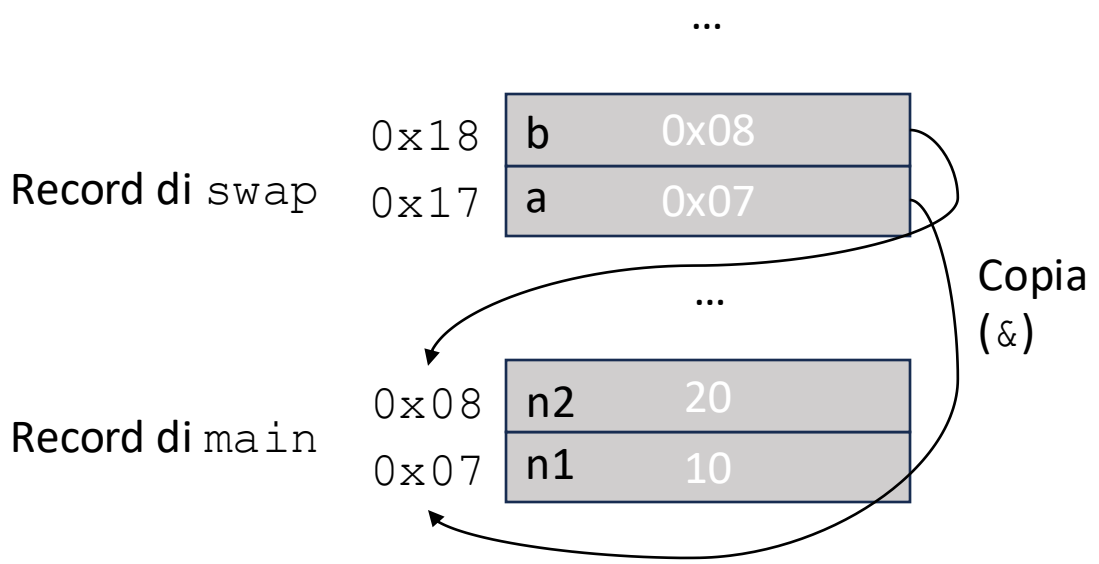
Funzioni

```
void swap(int *a, int *b){
      int temp = *a;
      *a = *b;
      *b = temp;
}

int main(){
      int n1 = 10;
      int n2 = 20;

      swap(&n1, &n2);
}
```

Porzione di stack al momento della chiamata a `swap`



L'indirizzo di `n1` e `n2` viene copiato nel record di attivazione di `swap`

→ È ancora un passaggio per valore!

Funzione `main`

- La funzione `int main()` è la funzione «principale» del nostro programma, viene chiamata dal terminale tramite il sistema operativo quando digitiamo il nome del programma compilato
- Senza argomenti, non riceve in ingresso alcun dato (vedremo più avanti come fare)
- Restituisce al chiamante (sistema operativo/terminale) un intero (`0` per convenzione indica che l'esecuzione è andata a buon fine, un codice diverso indica l'errore che ha causato la terminazione del programma)

```
int main(){
      /* codice */
      return 0;
}
```

Come strutturare un programma in funzioni

- Evitare la ripetizione del codice
- Se vi accorgete che state duplicando il codice più volte, è buona norma rendere il codice ripetuto una funzione una funzione

```
for (int i = 0; i < r1; i++)
        for (int j = 0; j < c1; j++)
                scanf("%d", &mat1[i][j]);

for (int i = 0; i < r2; i++)
        for (int j = 0; j < c2; j++)
                scanf("%d", &mat2[i][j]);

for (int i = 0; i < r3; i++)
        for (int j = 0; j < c3; j++)
                scanf("%d", &mat3[i][j]);
```

```
leggiMatrice(r1, c1, mat1);
leggiMatrice(r2, c2, mat2);
leggiMatrice(r3, c3, mat3);

. . .
void leggiMatrice(int righe, int colonne, int mat[N][M]) {
        for (int i = 0; i < righe; i++)
                for (int j = 0; j < colonne; j++)
                        scanf("%d", &mat[i][j]);
        }
}
```

Come strutturare un programma in funzioni

Tipicamente una funzione deve fare **una singola operazione** ben definita.

- Leggere un input o
- Fare dei calcoli / processing dei dati o
  - Fare uno dei passaggi di un algoritmo più complesso
- Stampare un output a schermo

Se una funzione fa più cose assieme, è meglio dividerla se possibile in più funzioni distinte.

Il nome della funzione deve essere significativo del duo ruolo, esempio
`somma maxArray minValore stampaArray leggiInt`

Come strutturare un programma in funzioni

```
int maxArray(int a[N]){
        int max = -9999;
        printf("Inserisci i valori dell'array\n");
        for(int i=0; i<N; i++)
                scanf("%d",&a[i]);
        for(int i=0; i<N; i++)
                if (max < a[i])
                        max = a[i];
        return max;
}
```

Cosa fa questa funzione?

Come strutturare un programma in funzioni

```
int maxArray(int a[N]){
        int max = -9999;
        printf("Inserisci i valori dell'array\n");
        for(int i=0; i<N; i++)
                scanf("%d",&a[i]);
        for(int i=0; i<N; i++)
                if (max < a[i])
                        max = a[i];
        return max;
}
```

Cosa fa questa funzione?

Inizializza i valori di un array e ne calcola il massimo

Errore: fa due cose! ~~è utile~~ occorre dividerla in due funzioni

Come strutturare un programma in funzioni

```
// nel main o
// o in una funzione
printf("Inserisci i valori dell'array\n");
for(int i=0; i<N; i++)
        scanf("%d",&array[i]);
max = maxArray(array)

// ...
int maxArray(int a[N]){
        int max = -9999;
        for(int i=0; i<N; i++)
                if (max < a[i])
                max = a[i];
        return max;
}
```

Cosa fa questa funzione? Calcola il massimo di un array

Restituire più di un valore con una funzione

La funzione termina quando il primo `return` viene eseguito, e viene restituito la variabile corrispondente.

Come posso fare se voglio restituire più di un parametro?

- Passare i valori per riferimento
- Restituire una `struct` (se i dati sono di tipo eterogeneo)
- Restituire un array (se i dati sono omogenei)

Come fare? Lo vediamo nelle prossime lezioni…

Array Multidimensionali

- Un array è una sequenza contigua di `N` elementi dello stesso tipo

10	20	30	40
`a[0]`	`a[1]`	`a[2]`	`a[3]`

- Una matrice è una sequenza multi-dimensionale di `M` x `N` elementi dello stesso tipo

10	20	30	40
11	21	31	41
12	22	32	42

`M=3` righe da `0` a `M-1`
`N=4` colonne da `0` a `N-1`

Array Multidimensionali

- Il numero di dimensioni è specificato a tempo di compilazione
  - Usare `#define`!
- Per dichiarare un array multidimensionale ripetiamo più volte le parentesi quadre
  - `tipo identificatore[VAL1][VAL2][VALn];`
  - `[VALi]` specifica che l'i-esima dimensione della matrice è composta da `vali` elementi
- Come per il resto delle variabili, in `C` va inizializzato prima dell'uso

```
#define M 3
#define N 4
int matrice[M][N];
```

Array Multidimensionali

```
#define M 3
#define N 4

int matrice[M][N] = {
      {10, 20, 30, 40},
      {11, 21, 31, 41},
      {12, 22, 32, 42}
};
```

- Il numero di dimensioni è specificato a tempo di compilazione
  - Usare `#define`!
- Per dichiarare un array multidimensionale ripetiamo più volte le parentesi quadre
  - `tipo identificatore[VAL1][VAL2][VALn];`
  - `[VALi]` specifica che l'i-esima dimensione della matrice è composta da `VALi` elementi

Array Multidimensionali

- Il numero di dimensioni è specificato a tempo di compilazione
  - Usare `#define`!
- Per dichiarare un array multidimensionale ripetiamo più volte le parentesi quadre
  - `tipo identificatore[VAL1][VAL2][VALn];`
  - `[VALi]` specifica che l'i-esima dimensione della matrice è composta da `VALi` elementi
- Specularmente, per accedere al valore di una cella specifichiamo l'indice per ogni dimensione
  - `matrice[i][j]` → elemento nella riga `i` e colonna `j`

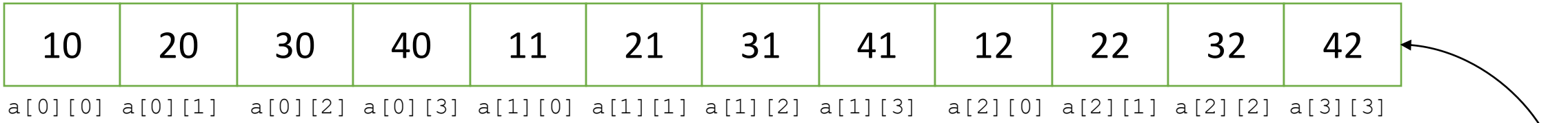
```
#define M 3
#define N 4

int matrice[M][N] = {
        {10, 20, 30, 40},
        {11, 21, 31, 41},
        {12, 22, 32, 42}
};

int i, j;
for(i=0; i<M; i++){
  for(j=0; j<N; j++){
    printf("%d ", matrice[i][j]);
  }
  printf("\n");
}
```

Array Multidimensionali

- Gli array multidimensionali sono memorizzati in una sequenza di memoria contigua



- Gli indici ci consentono di operare su di esso come se fosse realmente diviso in più righe e colonne

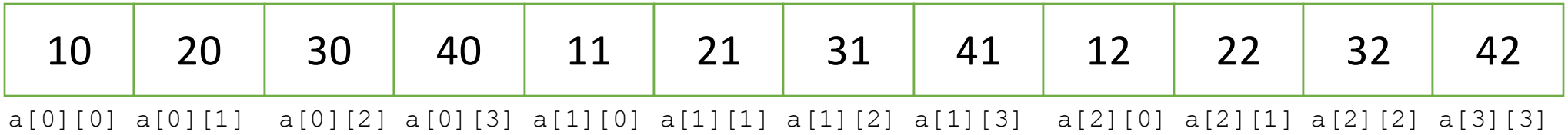
10	20	30	40
11	21	31	41
12	22	32	42

Rappresentazione in memoria

Rappresentazione logica nel codice

Array Multidimensionali

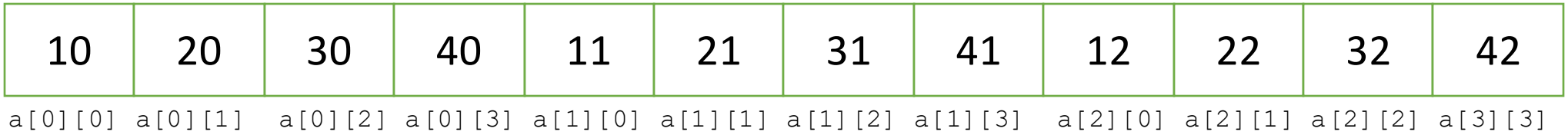
- Gli array multidimensionali sono memorizzati in una sequenza di memoria contigua



- Se sforo l'indice di una colonna, leggo automaticamente un valore situato riga successiva
- Qual è il valore di `a[0][4]`?

Array Multidimensionali

- Gli array multidimensionali sono memorizzati in una sequenza di memoria contigua



- Se sforo l'indice di una colonna, vado automaticamente alla riga successive
- Qual è il valore di `a[0][4]`? 11

Array Multidimensionali

Come per gli array monodimensionali, non è possibile la copia mediante assegnamento, occorre copiare elemento per elemento

```
#define M 3
#define N 4
int main(){
        int a[M][N];
        int b[M][N];
        b = a;
}
```

Errore

```
#define M 3
#define N 4
int main(){
        int a[M][N];
        int b[M][N];
        int i, j;

        for(i=0; i<M; i++){
                for(j=0; j<N; j++){
                        b[i][j] = a[i][j]
                }
        }
}
```

OK!

Trova l’errore

```c
#include <stdio.h>

double avg(int n1, int n2){
    return (n1 + n2) / 2.0;
}

int main(){
    int n1, n2;
    int result;

    printf("Inserisci i due numeri separati da spazio: ");
    scanf("%d %d", &n1, &n2);

    result = avg(n1, n2);
    printf("avg(%d, %d)=%d", n1, n2, result);

    return 0;
}
```