Laboratorio II

Corso A

Lezione 2

Funzioni e IO

Scoping

Puntatori

Passaggio per riferimento

Array, puntatori e funzioni

Parametri dalla linea di comando

Funzioni ***variadic***

Funzioni

- Astrazioni di blocchi di codice con un nome - come in JS/TS
  - Implementano una certa funzionalità per essere riutilizzata
- Sintassi C:
  - ***Dichiarazione*** - tipicamente all'inizio del programma - parametri e tipo di ritorno - prototipo
  - ***Definizione*** - include il corpo della funzione con il codice da eseguire.
  - Posso dichiarare e definire allo stesso tempo prima della main - di solito si separano (anche in file diversi e.g. .h, .c)
  - Se manca la definizione => linker error
  - Se manca la dichiarazione => compiler error

```c
#include <stdio.h>

int max(int, int);

int main(void) {

  int a = 5, b=10;

  printf("%d\n", max(a,b));

}

int max(int a, int b){
  if (a>b)
    return a;
  return b;
}
```

Qualche funzione per I/O

- Libreria standard per I/O: `stdio`
- Per mostrare a video (stdout) dei valori - funzione `printf()`
  - `printf(stringa_formattazione)` - scrive la stringa passata come argomento.
  - la stringa di formattazione può contenere dei segnaposti (`%...`) per includere dei valori di variabili - le variabili vanno incluse dopo la stringa, in ordine

```c
#include <stdio.h>

#define LEN 20

struct Studente {
  char nome[LEN];
  int eta;
};

int main(void) {

  struct Studente s= {  "Antonio",15};

  printf("Nome: %s, Età: %d\n",s.nome, s.eta);
  printf("Dimensione in memoria: %lu\n", sizeof(s));

  return 0;
}
```

`printf`

- il segnaposto dipende dal tipo della variabile da scrivere
  - interi: `%d, %u` (`unsigned`). Si antepone `h` per `short` e `l` per `long`
  - reali: `%f, %e` (notazione scientifica), `%g` la piu' breve tra notazione standard e scientifica. Per i `long double` si antepone `L`
  - caratteri: `%c`
  - stringhe: `%s`
  - si possono includere opzioni di formattazione
    - ampiezza minima del campo “%5d”
    - numero di decimali dopo la virgola “%.5f”

Qualche funzione per I/O

- Per leggere da stdin - funzione `scanf()`
- Sintassi simile alla `printf`, prende una stringa di formattazione più gli **indirizzi (&)** delle variabili dove salvare i valori letti.
  - `scanf("%d %d", &a, &b)` - legge due interi e li salva nelle variabili `a` e `b`
- Gli segnaposti sono gli stessi di `printf` tranne quello per `double: "%lf"`

```c
#include <stdio.h>

int main(void) {
  int a=0;
  double b=0;
  scanf("%d %lf",&a,&b);
  printf("a=%d\nb=%f\n", a,b);
  return 0;
}
```

```
> ./main
7
9.8
a=7
b=9.800000
>
```

Scoping

- I nomi sono utilizzabili dal momento in cui sono dichiarati fino alla fine dello scope in cui sono dichiarati
  - Variabili locali (interne) ad un blocco/funzione - visibili fino alla fine del blocco/funzione
  - Variabili globali (esterne) - definite fuori dalle funzioni - visibili fino alla fine del file corrente.
    - Funzioni sono sempre globali - non possiamo definire funzioni dentro ad altre funzioni
  - Variabili `extern` - possiamo utilizzare variabili globali definite in file diversi dichiarandole con `extern` – vanno definite solo una volta in tutto il programma (un solo file)
  - Variabili `static`
    - Possiamo impedire ad altri file di utilizzare variabili (anche funzioni) globali con `extern`, utilizzando `static`
    - Variabili `static` locali a funzioni - visibili solo nella funzione, persistenti tra una chiamata della funzione ad altra
- Shadowing possibile simile a JS

File header `.h`

Per modularità e riutilizzo del codice può essere utile dividere il programma in vari file (librerie)

La dichiarazione delle funzioni e costanti in un file `.h` da includere (`#include`) negli altri file che utilizzano le funzioni

La definizione delle funzioni in un file `.c` - può essere compilato separatamente

max.c

```c
int max(int a, int b){
  if (a>b)
    return a;
  return b;
}
```

max.h

```c
int max(int, int);

```

main.c

```c
#include <stdio.h>
#include "max.h"

int main(void) {
  int a=5, b=10;
  printf("%d\n", max(a,b));
}

```

Esempio

max.h

```c
int max(int, int);
```

main.c

```c
#include <stdio.h>
#include "max.h"

int main(void) {

  int a=0,b=0;

  do{
    int a=0;
    scanf("%d %d",&a,&b);
    printf("Max(%d,%d)=%d\n",a,b,max(a,b));
  } while(a!=b);

  printf("Hai inserito a>b %d volte\n",countA);

}
```

max.c

```c
int countA=0;
int countB=0;

int max(int a, int b){
  if (a>b){
    countA++;
    return a;
  }
  countB++;
  return b;
}
```

Esempio

max.h

```
1   int max(int, int);
```

main.c

```
1   #include <std
2   #include "max
3
4   int main(void
5
6     int a=0,b=0
7
8     do{
9       int a=0;
10      scanf("%d
11      printf("M
12    } while(a!=
13
14    printf("Hai
15
16  }
```

max.c

```
                                    int b){
11  }
```

```
main.c:14:40: error: use of undeclared
identifier
        'countA'
  printf("Hai inserito a>b %d volte\n",
countA);

^
1 error generated.
exit status 1
```

Esempio

max.h

```c
int max(int, int);
```

main.c

```c
#include <stdio.h>
#include "max.h"

extern int countA;

int main(void) {

  int a=0,b=0;

  do{
    int a=0;
    scanf("%d %d",&a,&b);
    printf("Max(%d,%d)=%d\n",a,b,max(a,b));
  } while(a!=b);

  printf("Hai inserito a>b %d volte\n",countA);

}
```

Quando ci fermiamo?

max.c

```c
int countA=0;
int countB=0;

int max(int a, int b){
  if (a>b){
    countA++;
    return a;
  }
  countB++;
  return b;
}
```

Esempio

max.h

```
int max(int, int);
```

main.c

```
#include <stdio.h>
#include "max.h"

extern int countA;

int main(void) {

  int a=0,b=0;

  do{
    int a=0;
    scanf("%d %d",&a,&b);
    printf("Max(%d,%d)=%d\n",a,
  } while(a!=b);

  printf("Hai inserito a>b %d v

}
```

```
> ./main
4 7
Max(4,7)=7
7 2
Max(7,2)=7
8 1
Max(8,1)=8
6 8
Max(6,8)=8
3
3
Max(3,3)=3
0
0
Max(0,0)=0
Hai inserito a>b 2 volte
```

```
int countA=0;
int countB=0;

int max(int a, int b){
  if (a>b){
    countA++;
    return a;
  }
  countB++;
  return b;
}
```

Esempio

max.h

```c
int max(int, int);
```

main.c

```c
#include <stdio.h>
#include "max.h"

extern int countA;

int main(void) {

  int a=0,b=0;

  do{
    int a=0;
    scanf("%d %d",&a,&b);
    printf("Max(%d,%d)=%d\n",a,b,max(a,b));
  } while(a!=b);

  printf("Hai inserito a>b %d volte\n",countA);

}
```

max.c

```c
static int countA=0;
static int countB=0;

int max(int a, int b){
  if (a>b){
    countA++;
    return a;
  }
  countB++;
  return b;
}
```

Esempio

max.h

```
1   int max(int, int);
```

main.c

```
1   #includ
2   #includ
3
4   extern
5
6   int mai
7
8     int a
9
10    do{
11      int
12      sca
13      pri
14    } whi
15
16    printf("Hai inserito a>b %d volte\n",countA);
17
18  }
19
```

max.c

```
10    return b;
11  }
```

```
/tmp/main-df0f92.o: In function `main':
main.c:(.text+0x84): undefined referenc
e to `countA'
clang-7: error: linker command failed w
ith exit code 1 (use -v to see invocati
on)
exit status 1
```

Esempio

max.h

```c
int max(int, int);
```

main.c

```c
#include <stdio.h>
#include "max.h"


int main(void) {

  int a=0,b=0;

  do{
    int a=0;
    scanf("%d %d",&a,&b);
    printf("Max(%d,%d)=%d\n",a,b,max(a,b));
  } while(a!=b);

}
```

max.c

```c
#include <stdio.h>

int max(int a, int b){
  static int countA=0;
  static int countB=0;

  if (a>b){
    countA++;
    printf("countA=%d, countB=%d\n",countA,countB);
    return a;
  }
  countB++;
  printf("countA=%d, countB=%d\n",countA,countB);
  return b;
}
```

Esempio

max.h

```
int max(int, int);
```

main.c

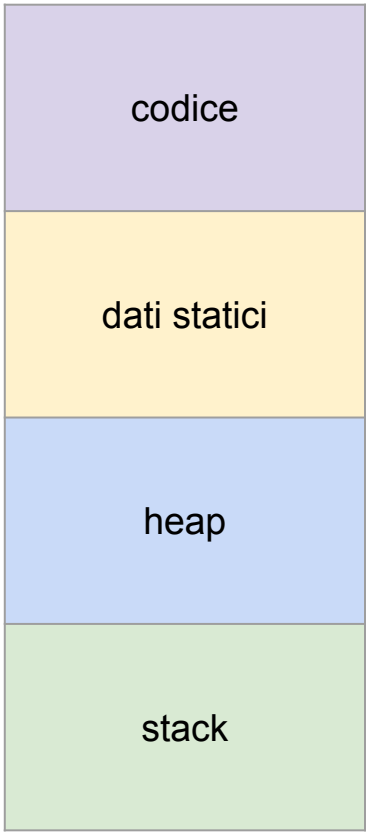
```
#include <stdio.h>
#include "max.h"


int main(void) {

  int a=0,b=0;

  do{
    int a=0;
    scanf("%d %d",&a,&b);
    printf("Max(%d,%d)=%d\n",a,b,max
  } while(a!=b);

}
```

```
3 5
countA=0, countB=1
Max(3,5)=5
6 2
countA=1, countB=1
Max(6,2)=6
3 8
countA=1, countB=2
Max(3,8)=8
2 2
countA=1, countB=3
Max(2,2)=2
0 0
countA=1, countB=4
Max(0,0)=0
```

```
de <stdio.h>

x(int a, int b){
ic int countA=0;
ic int countB=0;

a>b){
untA++;
intf("countA=%d, countB=%d\n",countA,countB);
turn a;

tB++;
tf("countA=%d, countB=%d\n",countA,countB);
rn b;
```

Gestione della memoria

- Programma e dati - memoria
- Zona dati
  - Dati statici - possono essere inferiti a tempo di compilazione
    - Variabili globali, statici
  - Stack - record di attivazione
    - 1 per ogni blocco di codice (incluso chiamate di funzioni)
    - spazio per variabili locali
  - Heap - dati dinamici (runtime), frammentati
    - Memoria allocata dinamicamente - tramite puntatori



Puntatori

- Qualsiasi dato/istruzione che sta in memoria ha **un valore** e un **indirizzo di memoria**
- Un puntatore è **una variabile** che contiene un indirizzo di memoria
  - Ha al suo turno un indirizzo di memoria
  - Il valore è un indirizzo di memoria - ci indica l'inizio di uno spazio di memoria che può essere nello stack, heap, anche istruzioni (codice)
  - Il tipo ***indica*** il tipo di dato che mi aspetto di vedere a quel indirizzo di memoria.
- Possiamo leggere/scrivere il valore memorizzato ad un certo indirizzo di memoria (operatore *)
- Possiamo leggere l'indirizzo di memoria di una variabile (operatore &)

```
int main(void) {

  int a = 5;

  int *aPtr = &a;

}
```

valore: 5
indirizzo: 13

valore: 13
indirizzo: 36

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	**5**	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	**13**	38	39	40	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65

Puntatori - esempio

```c
#include <stdio.h>

int main(void) {

  int a=0, b=0;

  int *aPtr=&a;

  printf("Il valore di a è %d, l'indirizzo di a è %p\n", a, &a);
  printf("Il valore di aPtr è %p, l'indirizzo di aPtr è %p\n", aPtr, &aPtr);
}
```

```
Il valore di a è 0, l'indirizzo di a è 0x7ffc743dbd1c
Il valore di aPtr è 0x7ffc743dbd1c, l'indirizzo di aPtr è 0x7ffc743dbd10
```

Puntatori

```c
#include <stdio.h>

int main(void) {

  double a=0;
  double *aPtr=&a, *aPtr1=&a;

  printf("Il valore di a è %f, l'indirizzo di a è %p\n", a, &a);

  (*aPtr)=10;
  printf("Il valore di a è %f, l'indirizzo di a è %p\n", a, &a);

  (*aPtr1)=20;
  printf("Il valore di a è %f, l'indirizzo di a è %p\n", a, &a);
}
```

```
Il valore di a è 0.000000, l'indirizzo di a è 0x7ffdb8ce4fe8
Il valore di a è 10.000000, l'indirizzo di a è 0x7ffdb8ce4fe8
Il valore di a è 20.000000, l'indirizzo di a è 0x7ffdb8ce4fe8
```

Puntatori

```c
#include <stdio.h>

int main(void) {

  int a=15;
  double *aPtr=&a;

  printf("Il valore di a è %d, l'indirizzo di a è %p\n", a, &a);

  printf("Il valore di aPtr è %p\n", aPtr);

  printf("Il valore memorizzato a %p è %f", aPtr, *aPtr);
}
```

```
Il valore di a è 15, l'indirizzo di a è 0x7ffcf87e0a3c
Il valore di aPtr è 0x7ffcf87e0a3c
Il valore memorizzato a 0x7ffcf87e0a3c è 0.000000
```

Puntatori

```c
#include <stdio.h>

int main(void) {

  int a=15;
  double *aPtr=&a;
```

```
main.c:6:11: warning: incompatible pointer types initializing 'double *' with an
      expression of type 'int *' [-Wincompatible-pointer-types]
  double *aPtr=&a;
          ^    ~~
1 warning generated.
```

```c
a, &a);

;

}
```

```
Il valore di a è 15, l'indirizzo di a è 0x7ffcf87e0a3c
Il valore di aPtr è 0x7ffcf87e0a3c
Il valore memorizzato a 0x7ffcf87e0a3c è 0.000000
```

Puntatori

```c
#include <stdio.h>

int main(void) {

  double a=15.6;
  int *aPtr=&a;

  printf("Il valore di a è %f, l'indirizzo di a è %p\n", a, &a);

  printf("Il valore di aPtr è %p\n", aPtr);

  printf("Il valore memorizzato a %p è %d", aPtr, *aPtr);
}
```

```
Il valore di a è 15.600000, l'indirizzo di a è 0x7ffee72ceed8
Il valore di aPtr è 0x7ffee72ceed8
Il valore memorizzato a 0x7ffee72ceed8 è 858993459
```

Passaggio parametri a funzioni

- Il passaggio di parametri viene fatto sempre per valore - si passa una copia del valore della variabile
- Per poter cambiare il valore di una variabile in una funzione si utilizzano i puntatori
  - Passando l'indirizzo di memoria di una variabile, quella variabile può essere modificata -

```
a=5,b=10
a=10,b=5
```

```c
#include <stdio.h>

void swap(int*, int* );

int main(void) {
  int a=5, b=10;

  printf("a=%d,b=%d\n", a,b);

  swap(&a,&b);

  printf("a=%d,b=%d\n", a,b);
}

void swap(int* x, int* y){
  int var=*x;
  *x=*y;
  *y=var;
}
```

La funzione `scanf`

```
int scanf(char* format,
...)
```

- Per ogni segnaposto viene letto un valore e salvato in una variabile - richiede di modificare il valore della variabile => si passa il puntatore
- Possiamo ignorare parte dell'input: usare "`%*...`"

```
7
9
a=7,b=9
aPtr=0x7ffda64dd56c,bPtr=0x7ffda64dd568
```

```c
int main(void) {
  int a, b;
  scanf("%d %d", &a, &b);
  printf("a=%d,b=%d\n", a,b);
}
```

```c
int main(void) {
  int a, b;
  int *aPtr=&a, *bPtr=&b;
  scanf("%d %d", aPtr, bPtr);
  printf("a=%d,b=%d\n", a,b);
  printf("aPtr=%p,bPtr=%p\n", aPtr,bPtr);
}
```

La funzione `scanf`

```
int scanf(char* format, ...)
```

- La funzione restituisce il numero di valori correttamente letti
  - Possiamo controllare l'input utente tramite la stringa di formattazione
  - Se la lettura fallisce dobbiamo svuotare il buffer di `scanf` per riprovare a leggere

`%`*n*`c`	legge esattamente *n* caratteri e.g. "%10c"
`%`*n*`s`	legge una stringa fino al primo white space o fino al massimo *n* caratteri, salta gli whitespace iniziali e.g. "`%10s`"
`%`*n*`[`*regex*`]`	legge una stringa di massimo n caratteri, che soddisfa l'espressione regolare e.g. "`%10[1-9]`" ,"`%[^\n]`"
http://www.gnu.org/software/libc/manual/html_node/String-Input-Conversions.html	

Esempio: controllo dell'input con `scanf`

Leggiamo un intero da linea di commando. Cosa succede se l'utente inserisce un input sbagliato? Stampiamo un messaggio di errore e rileggiamo.

```c
int main(void) {

  int a=0;

  while(scanf("%d",&a)!=1){
    printf("Errore di input. Inserisci
    un intero corretto.\n");
    scanf("%*[^\n]\n");
  }

  printf("Grazie, hai inserito: %d\n",a);
```

```
gfhd
Errore di input. Inserisci un intero corretto.
%
Errore di input. Inserisci un intero corretto.
5
Grazie, hai inserito: 5
```

```
5.3
Grazie, hai inserito: 5
```

Esempio: controllo dell'input con `scanf`

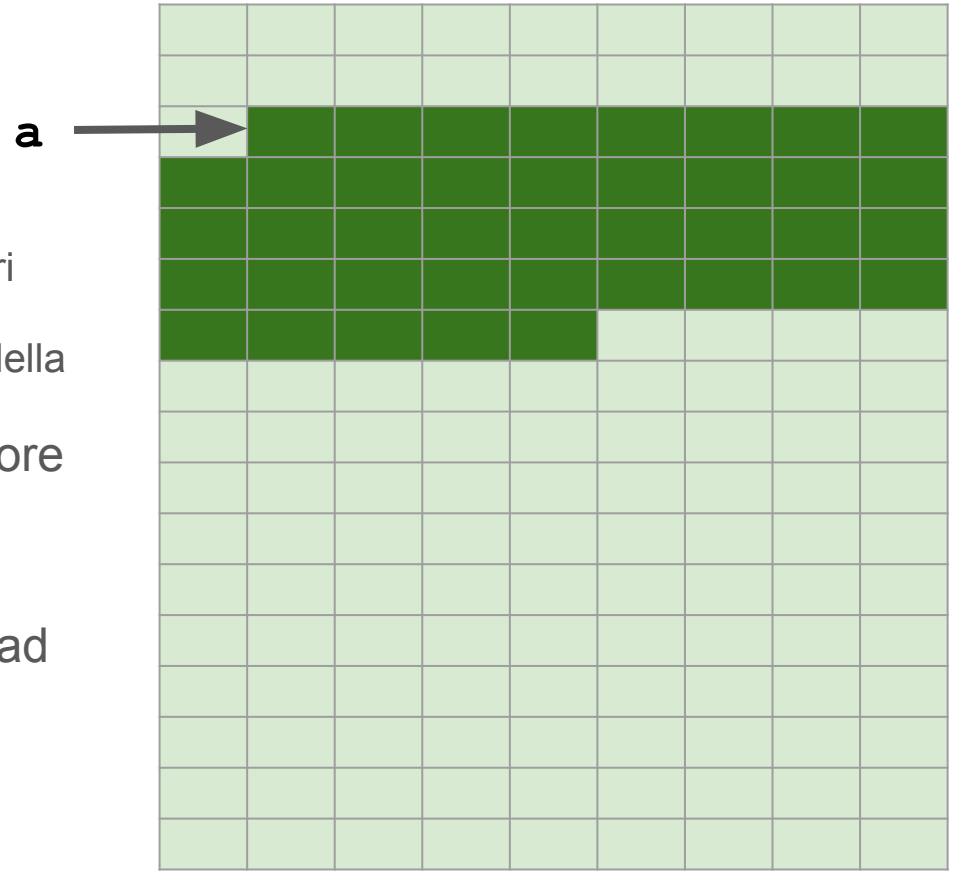
Leggiamo un intero da linea di commando. Cosa succede se l'utente inserisce un input sbagliato? Stampiamo un messaggio di errore e rileggiamo.

```
dfhd
Errore di input. Inserisci un intero corretto.
76.9
Errore di input. Inserisci un intero corretto.
8754
Grazie, hai inserito: 8754
```

```c
#include <stdio.h>


int main(void) {

  int a=0;
  char b;

  while(scanf("%d%c",&a,&b)!=2 || b=='.')
  {
    printf("Errore di input. Inserisci
    un intero corretto.\n");
    scanf("%*[^\n]\n");
  }

  printf("Grazie, hai inserito: %d\n",a);

}
```

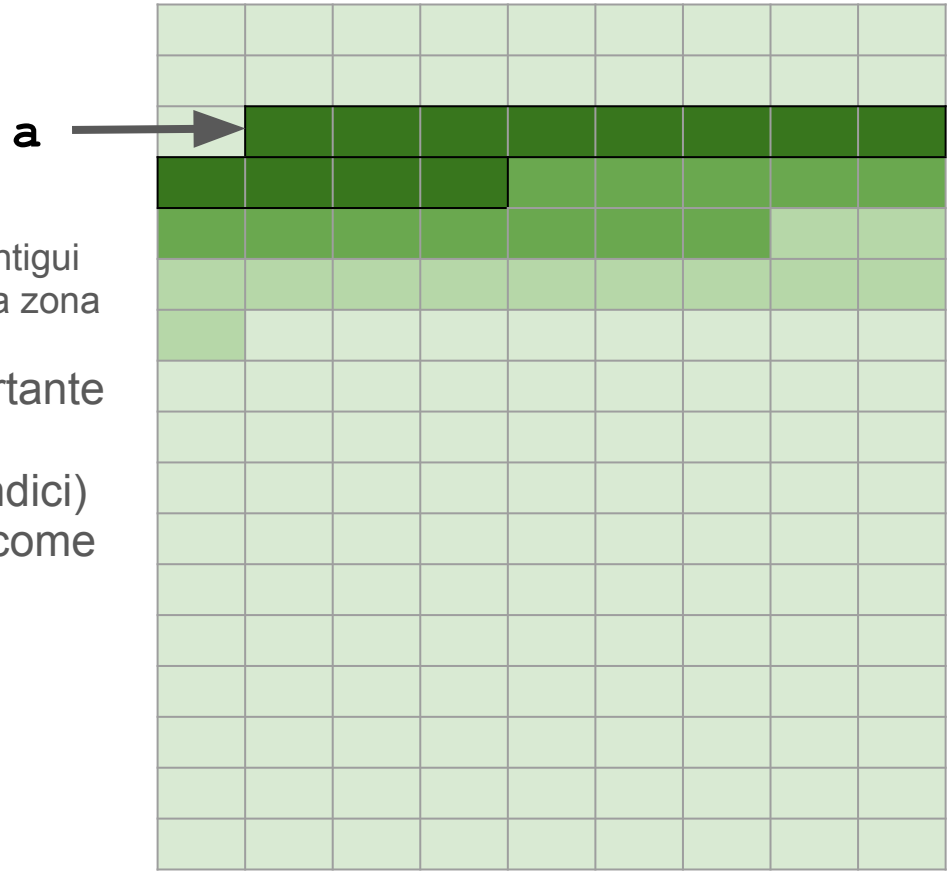
Array, puntatori e funzioni

- `int a[10];`
  - crea una variabile `a`
  - alloca in memoria sullo stack 10 interi contigui
  - memorizza in a l'indirizzo dell'inizio della zona di memoria allocata
- La variabile `a` è in realtà un puntatore
  - array statici - allocati sullo stack
  - array dinamici (vediamo la prossima settimana) - allocati sul heap
- Passare un array come parametro ad una funzione - viene passato il puntatore al primo elemento



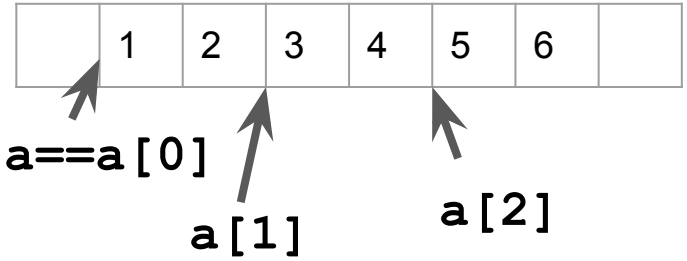
Array bidimensionali

- `int a[3][3];`
  - crea una variabile `a`
  - alloca in memoria sullo stack 9 interi contigui
  - memorizza in a l'indirizzo dell'inizio della zona di memoria allocata
- La seconda dimensione è molto importante (aiuta il compilatore a calcolare la locazione di memoria giusta, dati gli indici)
- Per passare un array bidimensionale come parametro dobbiamo specificare la seconda dimensione
  - `int f(int [][3]);`



Array bidimensionali

```c
#include <stdio.h>

int main(void) {

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

  printf("%p\n", a);
  printf("%p, %p, %p\n", a[0], a[1], a[2]);

}
```




```
0x7ffd2ba5ac10
0x7ffd2ba5ac10, 0x7ffd2ba5ac18, 0x7ffd2ba5ac20
```

Parametri dalla linea di comando

`./main` esegue il codice dalla linea di commando

Possiamo passare dei parametri e opzioni, come nel caso di comandi UNIX

Esempio: nel calcolo del max passiamo i due valori di a e b dalla linea di comando

```
./main 7 5   - stampa 7
./main 2 3   - stampa 3
```

Parametri dalla linea di comando

Vengono passati al programma tramite parametri della funzione `main`:

```
int main(int argc, char* argv[])
```

- `argc` - numero di parametri
- `argv` - parametri stessi - stringhe (`char*`)
  - `argv[0]` - nome del programma

Funzioni con numero variabile di parametri

Variadic functions - libreria `stdarg.h`

Specifichiamo i parametri variabili con … (ellipsis) : `void f(char x,...)`

E' necessario avere almeno un parametro non variabile, e … deve essere incluso alla fine

Per accedere ai parametri usiamo dei tipi e macro definiti in `stdarg.h`:

- `va_list` - definire la lista di parametri : `va_list par;`
- `va_start` - inizializzazione della lista specificando l'ultimo parametro noto: `va_start(par,x);`
- `va_arg` - restituisce il prossimo parametro castandolo al tipo specificato: `va_arg(par,int);`
- `va_end` - cleanup, deve essere chiamato alla fine della funzione: `va_end(par);`

Domande?

Esercizi

- Scrivere un programma che legge ***n*** e due vettori di dimensione ***n***, e fa il prodotto scalare tra i due vettori. Il prodotto va calcolato in una funzione.
- Scrivere un programma che legge ***n*** e due matrici quadrate di dimensione ***nXn***, e fa il prodotto tra le due matrici. Il prodotto va calcolato in una funzione.
- Scrivere un programma che fa la somma dei parametri ricevuti alla linea di comando
- Scrivere una funzione variadica che fa la somma di tutti i parametri ricevuti