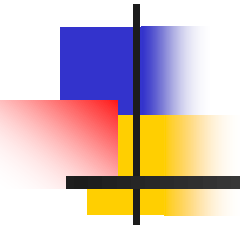
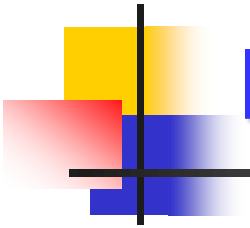


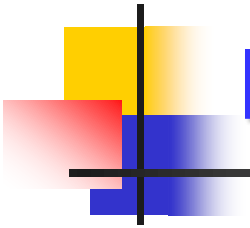
Concetti di Base sul Linguaggio C

Prof.Ing.S.Cavalieri



Linguaggio C

- 1972: Dennis Ritchie sviluppa la prima versione del linguaggio C
  - Molte idee basilari del C derivano dal linguaggio BCDL e linguaggio B
- 1977: Kernighan e Ritchie pubblicano "The C Programming Language"
- 1983: l'American National Standards Institute (ANSI) ha iniziato i lavori per la definizione dell'ANSI C
- 1989: nasce l'ANSI C, noto come C89



Linguaggio C

- 1990: la definizione di C passa a ISO/IEC (Instruments Society of America / International Electrotechnical Commission), che pubblica il **C90**
- 1994-1998: vengono pubblicate integrazioni/correzioni
- 1999: ISO/IEC definisce il **C99**
- 2000-2010: vengono pubblicate integrazioni/correzioni
- 2011: ISO/IEC definisce il **C11**, utilizzato adesso

Header File e Funzioni di Libreria in C

- Nel linguaggio C, come in altri linguaggi, molte operazioni vengono delegate a delle librerie
- Le funzioni di libreria sono divise in gruppi quali: I/O, gestione della memoria, operazioni matematiche e manipolazione di stringhe
- Per ogni gruppo di funzioni esiste un file di testo, chiamato *file header*, contenente le informazioni necessarie per utilizzare le funzioni (dichiarazione costanti, funzioni, parametri delle funzioni, etc.)
- Il codice implementativo delle funzioni di libreria è posto in altri file precompilati (file oggetto) che vengono collegati al file oggetto del programma, dal Linker.
- I nomi dei file header terminano, per convenzione, con l'estensione ".h"
  - ad esempio, *stdio.h* è il file header dello standard I/O nel linguaggio C

Header File e Funzioni di Libreria in C

- Per includere un file header in un programma in linguaggio C, occorre inserire nel codice sorgente:

  #include <nomefile.h>

- La direttiva *#include* è rivolta al preprocessore
- Si chiama: Direttiva di Precompilazione
- Esempio: Per utilizzare *printf()*, che permette di visualizzare dati su terminale, è necessario inserire nel sorgente la direttiva di precompilazione:

  #include <stdio.h>

Il Primo programma in Linguaggio C !

- La funzione main rappresenta l'entry point del programma. Deve essere sempre presente
- Lo standard C11 definisce solo due modi di definire il main

```
1) int main(void) {

                    istruzioni;

   }
2) int main(int argc, char *argv[]) {

                    istruzioni;
   }
```

- Noi useremo SOLO l'opzione 1)
- **Nota: chi ha libri vecchi deve stare attento !**

Il Primo programma in Linguaggio C !

```
#include<stdio.h>
int main(void)
{
    printf("Salve, mondo");
}
```

Include la libreria standard di I/O

Definisce una funzione main che non riceve alcun valore come argomento

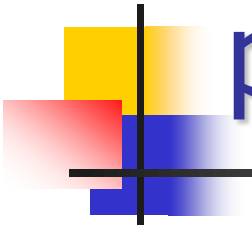
main richiama la funzione di libreria printf per stampare la sequenza di caratteri specificata

Ogni istruzione termina con il ;

Il Primo programma in Linguaggio C !

```
#include<stdio.h>
int main(void)
{
    printf("Salve, mondo");
    getchar();
}
```

In alcuni ambienti di compilazione/esecuzione, se non ci fosse non si vedrebbe nulla

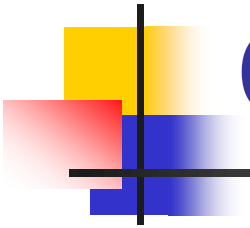


printf

- Esistono diversi modi di usarlo
- Più semplice: printf(controllo)
  - Controllo racchiuso da “ “, contiene testo e/o sequenze di escape:
    - Carattere \ “backslash”
    - \n linea nuova
    - \t salta di una tabulazione
    - \b ritorna un carattere indietro
    - \\ stampa carattere \
    - \” stampa carattere “

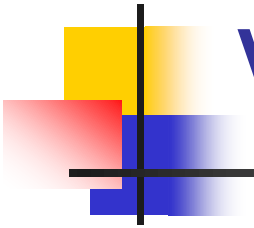
printf

```
#include<stdio.h>

int main(void){

    printf("\nIngegneria  ");
    printf("\tIngegneria  ");
    printf("\n\tIngegneria  ");
    printf("\b\bIngegneria  ");
    printf("\n\tIngegneria  ");
    printf("Ingegneria  ");
    printf("\t\nIngegneria ");
    printf("\n  \\   \" \n\n");

}
```



Commenti

- Si usa /* commento */
  - Il commento può essere lungo più righe
- Oppure //
  - Il commento deve essere lungo solo una riga

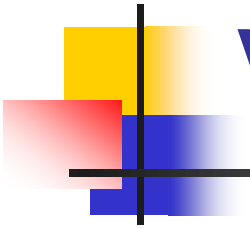


Variabili

- Corrispondono a locazioni di memoria dove memorizzare tutti i dati
- Ogni variabile deve essere Dichiarata prima di poterla usare
- La dichiarazione permette di poter riservare spazio ed associare allo spazio un nome e un tipo (codifica dati)
- Ogni variabile è caratterizzata da: nome e tipo (ossia tipologia di codifica, lunghezza e range)

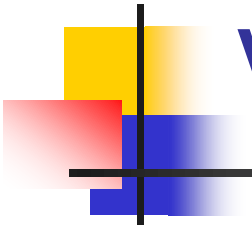
Variabili

- Ogni variabile:
  - Ha un nome al quale viene fatto corrispondere (in fase di esecuzione) un indirizzo di memoria (assegnato ad ogni esecuzione del programma); **L-value**
  - Assume un determinato valore (**R-value**) di un determinato tipo; il tipo è associato alla variabile
  - Il valore della variabile può essere inizializzato e può essere modificato nel corso del programma (tranne per le variabili const)



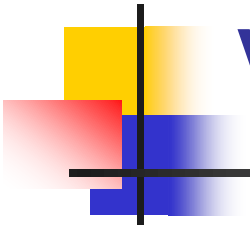
Variabili

- Ogni variabile ha un nome che:
  - Può essere composto da cifre e numeri, senza spazi, e carattere "_" ammesso
  - Non può iniziare con un numero, ma solo con carattere e/o "_" (non consigliato!)
  - Non si possono usare "parole chiavi"
  - Caratteri Maiuscoli e Minuscoli (case sensitive)
- Valore iniziale di default (non per tutte le variabili), dipendente dal tipo
- Ad ogni variabile viene associato un indirizzo di memoria (puntatore all'area della variabile)



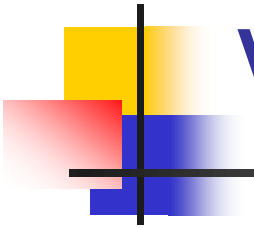
Variabili

- Possono essere dichiarate in differenti parti del programma, ma sempre prima del loro utilizzo (errore di compilazione)
- Per il momento consideriamo solo quelle ESTERNE (o GLOBALI)
  - dopo le direttive di precompilazione
  - prima del **int main(void)**
  - Vengono "allocate" in memoria RAM alla partenza e assumono un valore certo (predefinito o definito dall'utente)
  - Spariscono dalla memoria insieme alla conclusione del programma
  - File Scope: visibilità in tutto il file .c



Variabili

- Tipo: si dichiarano utilizzando parole chiavi: short, unsigned, int, long, float, double …….
- In particolare considereremo i seguenti tipi di dato:
  - il tipo intero corto short int (o short);
  - il tipo intero int;
  - Il tipo intero long int (o long)
  - Il tipo intero long long int (o long long)
  - il tipo intero senza segno (unsigned short, unsigned int, unsigned long, unsigned long long);
  - il tipo reale a singola precisione (float);
  - il tipo reale a doppia precisione (double);
  - il tipo reale ad alta precisione (long double);
  - il tipo carattere (char);

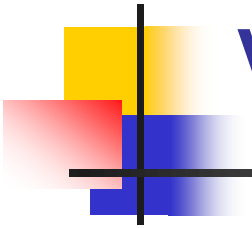


Variabili

- Le codifiche della regola di conversione è sempre la stessa per qualunque compilatore
- Le lunghezze in bit possono variare da compilatore a compilatore
  - Lo standard ha permesso delle scelte opzionali
  - Attenzione agli intervalli
- Per ogni tipo, esistono degli intervalli ammessi

Variabili Intere

Type	Numero di Bits	Codifica	Valore Minimo	Valore Massimo
short	16	Complemento a 2	–32.768	32.767
unsigned short	16	Senza Segno	0	65.535
int	32	Complemento a 2	–2.147.483.648	2.147.483.647
unsigned int	32	Senza Segno	0	4.294.967.295
long	32	Complemento a 2	–2.147.483.648	2.147.483.647
unsigned long	32	Senza Segno	0	4.294.967.295
long long int	64	Complemento a 2	-9223372036854775808	-9223372036854775807
unsigned long long	64	Senza Segno	0	18446744073709551615

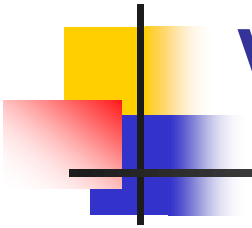


Variabili

- Nel file limits.h sono contenute le definizioni delle costanti SHRT_MIN, SHRT_MAX, INT_MIN, INT_MAX, LONG_MIN, LONG_MAX, USHRT_MAX, UINT_MAX e ULONG_MAX
- Assumono i valori massimi e minimi dei tipi interi.
- Valori iniziali 0

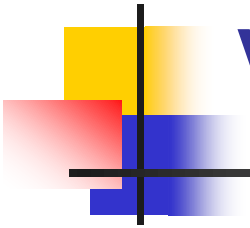
Variabili

Type	Codifica	Valore Minimo	Valore Massimo
float	IEEE 754 32 bit	$\pm 1.17 \times 10^{-38}$	$\pm 3.4 \times 10^{+38}$
double	IEEE 754 64 bit	$\pm 2.2 \times 10^{-308}$	$\pm 1.7 \times 10^{+308}$
long double	Estensione IEEE 754	$\pm 3.3 \times 10^{-4932}$	$\pm 1.1 \times 10^{+4932}$



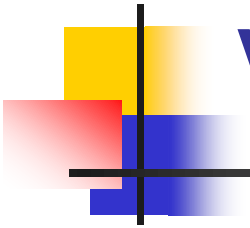
Variabili

- Nel file float.h sono contenute le definizioni delle costanti FLT_MIN, FLT_MAX, DBL_MIN, DBL_MAX, LDBL_MIN, LDBL_MAX
- Assumono i valori massimi e minini dei tipi float, double e long double.
- Valori inziali 0.0



Variabili

- Char
- Il tipo carattere (char) in linguaggio C occupa 1 byte, e utilizza la codifica ASCII (pronunciato "aschi").
- Valore iniziale: carattere vuoto ‘\0’ (codice ASCII 0)
- Valori tra singoli apici (‘a’)



Variabili

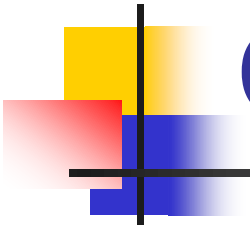
- Dichiarazione di Variabili Globali (Esterne)

```
#include <stdio.h>

tipobase nome_variabile;
tipobase nome_variabile1, nome_variabile2;

int main(void){


}
```

```
#include <stdio.h>

int x;
float y, z;
unsigned int k;

int main(void){


}
```

Inizializzazione di Variabili

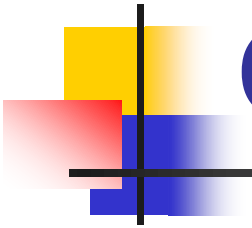
- Nella dichiarazione di una variabile, è possibile impostare il valore iniziale
- Operatore di assegnazione =
- Esempio:
  - int x=3; float y=3.4; char z='a';



Costanti

- Si usa la direttiva #define
- #define Simbolo Valore
- Senza ;
- Esempio: #define N 10
- In alternativa si può usare il costrutto

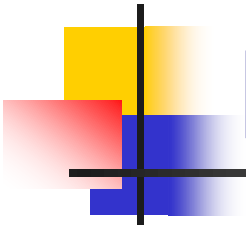
  const tipobase nome=valore;



Costanti

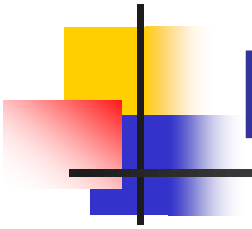
```
#include <stdio.h>
#define N 10


int main(void){

}
```

```
#include <stdio.h>
const unsigned int N=10;


int main(void){

}
```



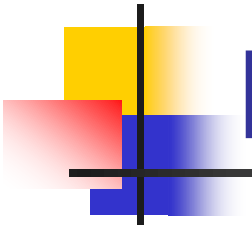
printf

- Uso di printf per visualizzare valori di variabile/costante
- printf(controllo, argomenti)
- Argomenti: elenco di variabili con uso della virgola
- Controllo contiene anche le regole di formattazione in uscita (simbolo %) per ogni variabile (corrispondenza posizionale): vedi lucido successivo



printf

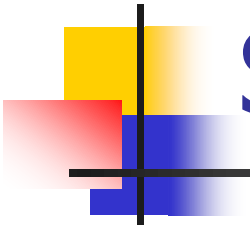
- Controllo contiene per ogni variabile da visualizzare:
  - %d, intero con segno
  - %u, intero senza segno
  - %c, carattere
  - %s, stringa
  - %f, reale virgola mobile (float e double)
  - %e,%E, reale virgola mobile (esponenziale: e+10, E+10)
  - %p, puntatore (indirizzo)
  - %zu, visualizza il valore di sizeof
  - %%, visualizza %



printf

Simboli da inserire prima dei caratteri di controllo:

- Simbolo **l**, long int (%ld), unsigned long (%lu)
- Simbolo **ll**, long long int (%lld), unsigned long long (%llu)
- Simbolo **L**, long double (%Lf, %Le, %LE)
- Simbolo **h**, per gli short (%hd), (%hu)



Sizeof

- E' possibile "scoprire" la dimensione in byte di un tipo su un certo sistema
- Esiste una funzione di libreria: sizeof(tipo);
- Funzione di <stdio.h>
- Esempio: sizeof(int)
- Formato per printf: %zu (da C99 in poi)

Esempio

- Il seguente programma in C permette di visualizzare sullo schermo i limiti per gli interi.

```
#include <stdio.h>
#include <limits.h>
int main(void) {
  printf("Dimensione in byte del tipo short = %zu \n", sizeof(short));
  printf ("\nRange del tipo short \n");
  printf ("MIN=%d \t",SHRT_MIN);
  printf ("MAX=%d \n",SHRT_MAX);
  printf("\nRange del tipo unsigned short \n");
  printf("MIN=0 \t");
  printf("MAX=%u \n", USHRT_MAX);
  printf("Dimensione in byte del tipo int = %zu \n", sizeof(int));
  printf("\nRange del tipo int \n");
  printf ("MIN=%d \t",INT_MIN);
  printf ("MAX=%d \n",INT_MAX);
  printf("\nRange del tipo unsigned int \n");
  printf("MIN=0 \t");
  printf("MAX=%u \n", UINT_MAX);
  printf("Dimensione in byte del tipo long int = %zu \n", sizeof(long int));
  printf("\nRange del tipo long int \n");
  printf ("MIN=%ld \t",LONG_MIN);
  printf ("MAX=%ld \n",LONG_MAX);
  printf("\nRange del tipo unsigned long int \n");
  printf("MIN=0 \t");
  printf("MAX=%lu \n", ULONG_MAX);
  printf("Dimensione in byte del tipo long long int = %zu \n", sizeof(long long int));
  printf("\nRange del tipo long long int \n");
  printf("MIN=%lld \t", LLONG_MIN);
  printf("MAX=%lld \n", LLONG_MAX);
  printf("\nRange del tipo unsigned long long int \n");
  printf("MIN=0 \t");
  printf("MAX=%llu \n", ULLONG_MAX); }
```

Esempio

- Il seguente programma in C permette di visualizzare sullo schermo i limiti dei numeri reali.
- Il programma fa uso della libreria float.h che contiene le definizioni delle costanti FLT_MIN, FLT_MAX, DBL_MIN, DBL_MAX, LDBL_MIN, LDBL_MAX relative rispettivamente ai valori massimi e minini dei tipi float, double e long double.

```
#include <stdio.h>
#include <float.h>

int main(void) {

  printf("Dimensione in byte del tipo float = %zu ", sizeof(float));
  printf("\nRange del tipo float ");
  printf ("MIN = %e \t",FLT_MIN);
  printf ("MAX = %e ",FLT_MAX);

  printf("\n\nDimensione in byte del tipo double = %zu ", sizeof(double));
  printf("\nRange del tipo double ");
  printf ("MIN = %e \t",DBL_MIN);
  printf ("MAX = %e ",DBL_MAX);

  printf("\n\nDimensione in byte del tipo long double = %zu ", sizeof(long double));
  printf("\nRange del tipo long double ");
  printf ("MIN = %Le \t",LDBL_MIN);
  printf ("MAX = %LE \n",LDBL_MAX);
}
```

Esempio

- Il seguente programma permette di visualizzare il codice ASCII di un carattere ('a`):

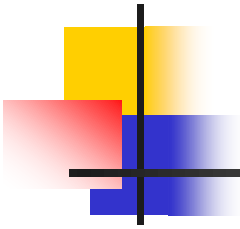
```
#include<stdio.h>

int main(void){
   printf("\nCarattere ASCII di %d ",'a');
}
```

Esempio

```
#include<stdio.h>

char c='a';

int main(void){
    printf("\nIl carattere e' %c ",c);
    printf("\nIl suo codice ASCII e' %d ",c);

}
```

```
#include<stdio.h>

int x;

int main(void){
    printf("\nLa variabile e' = %d ",x);

}
```

```
#include<stdio.h>

int x=3;

int main(void){
    printf("\nLa variabile e' = %d ",x);

}
```



scanf

- Sintassi: scanf("%x",&variabile)
- %x = %d, %f, %u ……. come printf
- & (ampersand) = indirizzo RAM della variabile
- Differenze con printf:
  - l = long int o unsigned long int o **double**
    - %ld, %lu, **%lf, %le, %lE**
  - L = long double
    - %Lf, %Le, %LE

scanf

- Programma con molti errori
- Comandi in rosso sono errati

```
#include<stdio.h>

float x;
double y;

int main(void){
    printf("\nInserisci un numero float ");
    scanf("%f", x);
    printf("\nIl numero inserito e' %f ",x);

    printf("\nInserisci un numero float ");
    scanf("%f",&x);
    printf("\nIl numero inserito e' %f ",x);

    printf("\nInserisci un numero double ");
    scanf("%f",&y);
    printf("\nIl numero inserito e' %f ",y);

    printf("\nInserisci un numero double ");
    scanf("%lf",&y);
    printf("\nIl numero inserito e' %d ",y);

    printf("\nInserisci un numero double ");
    scanf("%lf",&y);
    printf("\nIl numero inserito e' %f \n\n",y);
}
```

scanf

65	'A'
66	'B'
67	'C'
68	'D'
69	'E'
70	'F'
71	'G'

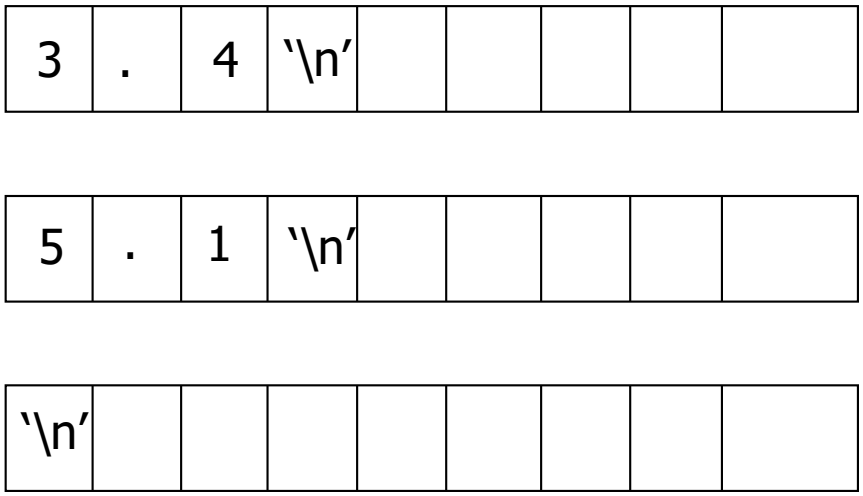
97	'a'
98	'b'
99	'c'
100	'd'
101	'e'
102	'f'
103	'g'

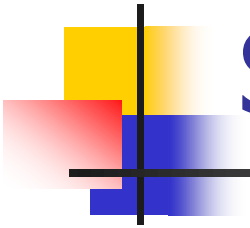
```
#include<stdio.h>

char c;

int main(void){

   printf("\nInserisci un carattere Maiuscolo ");
   scanf("%c",&c);
   printf("\nIl carattere inserito e' %c ",c);
   printf("\nIl suo codice ASCII e' %d ",c);
   printf("\nIl carattere minuscolo e' %c ",c+'a'-'A');
   printf("\nIl suo codice ASCII e' %d \n\n\n",c+'a'-'A');
}
```

Problema con scanf

```
#include <stdio.h>

float x,z;
char y;

int main(void){
   printf("\nInserisci un numero reale ");
   scanf("%f",&x);
   printf("\nIl numero reale e' = %f ",x);
   printf("\nInserisci un numero reale ");
   scanf("%f",&z);
   printf("\nIl numero reale e' = %f ",z);
   printf("\nInserisci un carattere ");
   scanf("%c",&y);
   printf("\nIl carattere  e' = %c \n\n",y);

}
```

buffer di ingresso “stdin”





Soluzione

- Inserire prima della lettura di un carattere il codice per pulire il buffer di ingresso “stdin”
  - fflush(stdin), ma non è standard
- Ignorare il carattere return '\n'
  - scanf("%*c%c",&variabile);
  - scanf("\n%c",&variabile);
  - scanf(" %c",&variabile); (spazio)
  - Il carattere '\n' viene ignorato e non assegnato alla variabile

Funzioni di Libreria

- Una *funzione* è un insieme di istruzioni che realizzano un compito: a partire da uno o più valori in input, restituisce un determinato valore in output.
- Esistono funzioni di libreria STANDARD, ossia definite dallo stesso documento che definisce il linguaggio C: tutti i compilatori DEVONO implementarle
- Si dividono in gruppi

Funzioni di Libreria

- Diagnostics **<assert.h>**
- Complex **<complex.h>**
- Character handling **<ctype.h>**
- Errors **<errno.h>**
- Floating-point environment **<fenv.h>**
- Characteristics of floating types **<float.h>**
- Format conversion of integer types **<inttypes.h>**
- Alternative spellings **<iso646.h>**
- Sizes of integer types **<limits.h>**
- Localization **<locale.h>**
- Mathematics **<math.h>**
- Nonlocal jumps **<setjmp.h>**
- Signal handling **<signal.h>**
- Alignment **<stdalign.h>**
- Variable arguments **<stdarg.h>**
- Atomics **<stdatomic.h>**
- Boolean type and values **<stdbool.h>**
- Common definitions **<stddef.h>**
- Integer types **<stdint.h>**
- Input/output **<stdio.h>**
- General utilities **<stdlib.h>**
- **_Noreturn <stdnoreturn.h>**
- String handling **<string.h>**
- Type-generic math **<tgmath.h>**
- Threads **<threads.h>**
- Date and time **<time.h>**
- Unicode utilities **<uchar.h>**
- Extended multibyte/wide character utilities **<wchar.h>**
- Wide character classification and mapping utilities **<wctype.h>**

Funzioni di Libreria

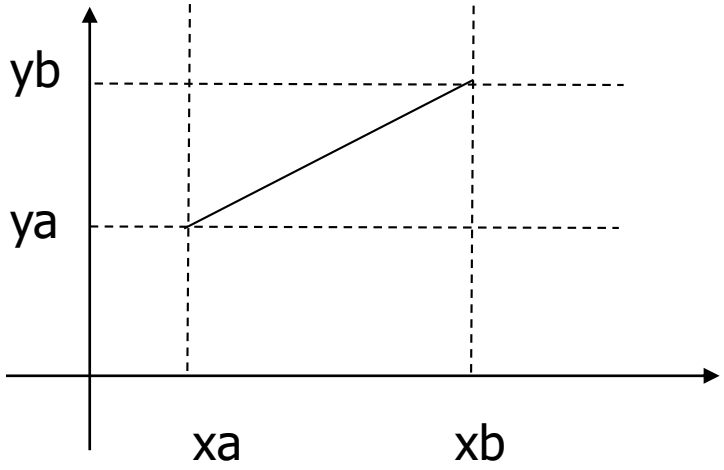
- **Ad esempio: math.h**
  - Per poter utilizzare le funzioni si deve includere il riferimento alla libreria `math.h`.

```
#include <math.h>
```

  - sqrt (intero/reale): radice quadrata
  - pow (intero/reale,intero/reale): potenza

Funzioni di Libreria

```
#include <stdio.h>
#include <math.h>
int xa, ya, xb, yb;

int main(void) {
   printf("\n\nLUNGHEZZA SEGMENTO\n");
   printf("\n\nCoordinate Primo Estremo: ");
   printf("\nCoordinata x: ");
   scanf("%d", &xa);
   printf("\nCoordinata y: ");
   scanf("%d", &ya);
   printf("\n\nCoordinate Secondo Estremo: ");
   printf("\nCoordinata x: ");
   scanf("%d", &xb);
   printf("\nCoordinata y: ");
   scanf("%d", &yb);
   printf("Lunghezza segmento: %f\n", sqrt(pow(xb - xa, 2) + pow(yb - ya, 2)));
}
```



$$l = \sqrt{(xb - xa)^2 + (yb - ya)^2}$$