



**POLITECNICO**  
MILANO 1863

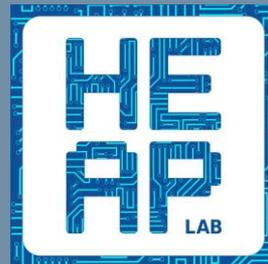
# INFORMATICA A

A.A. 2018-19

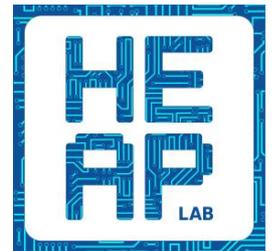
Laboratorio n°1

Dott. Michele Zanella

Ing. Gian Enrico Conti



- Contatti:
  - [michele.zanella@polimi.it](mailto:michele.zanella@polimi.it) (Squadra B)
    - [HEAP Lab](#) – Campus Leonardo, via Golgi 39, Edificio 21, Piano 1, Ufficio 4, +39 02 2399 **9613**  
(mandatemi una mail per accordarci su giorno e ora)
  - [gianenrico.conti@mail.polimi.it](mailto:gianenrico.conti@mail.polimi.it) (Squadra A e C)
- Sito web del laboratorio:
  - <http://zanella.faculty.polimi.it/teaching/informatica-a>
- **Nota per le mail:**  
Oggetto: *[INFO-A] il vostro oggetto*



# Obiettivi e organizzazione del corso

- **Obiettivi**
  - Sviluppo di capacità analitiche, di comprensione e di valutazione del codice
  - Progettazione e implementazione di soluzioni algoritmiche a problemi proposti
- **Prima parte**
  - Introduzione all'argomento giornaliero e presentazione strumenti di sviluppo
  - Acquisire autonomia nella gestione dei problemi
  - Realizzazione di primi programmi semplici e di esercizi ispirati ai temi d'esame
- **Seconda parte**
  - Applicazione delle nozioni apprese in un contesto più complesso
  - Verranno lasciati esercizi da completare a casa, privi di soluzione

## Vi è l'obbligo di frequenza:

- a) Al termine del laboratorio **si firma la presenza**
- b) 4 presenze minime su 6 laboratori
- c) In caso di mancata frequenza il docente ne terrà conto in sede di valutazione dell'esame scritto

# Obiettivi e organizzazione del corso

- **Argomenti**

- Introduzione sull'ambiente di sviluppo (IDE), compilazione di un programma, debugging
- Buone norme di programmazione in C
- I/O, semplici algoritmi e programmi di esempio
- Array e matrici
- Strutture dati (Struct)
- Puntatori
- File I/O
- Memoria dinamica e liste
- *Ricorsione*

- Autenticarsi usando il proprio codice persona e password all'indirizzo <https://virtualdesktop.polimi.it>

In *Utility* esiste lo strumento *Esplora Risorse*

- Cliccare per scaricare il file *launch.ica*
- Aprire il file *launch.ica* usando il client *Citrix*
- C: è il disco della macchina locale
- Y: è una cartella personale remota
- Usando *Esplora risorse* è possibile trasferire file da C: a Y: e viceversa.

N.B.: Se ci sono problemi copia/incolla dovete dare permessi di scrittura/lettura tramite il client Citrix.

# Virtual Desktop (cont'd)

Utilizzeremo un IDE durante il laboratorio: *CodeBlocks*

<http://www.codeblocks.org/>

- Cliccare su *CodeBlocks* ed aprire il file *launch.ica* con il client *Citrix*

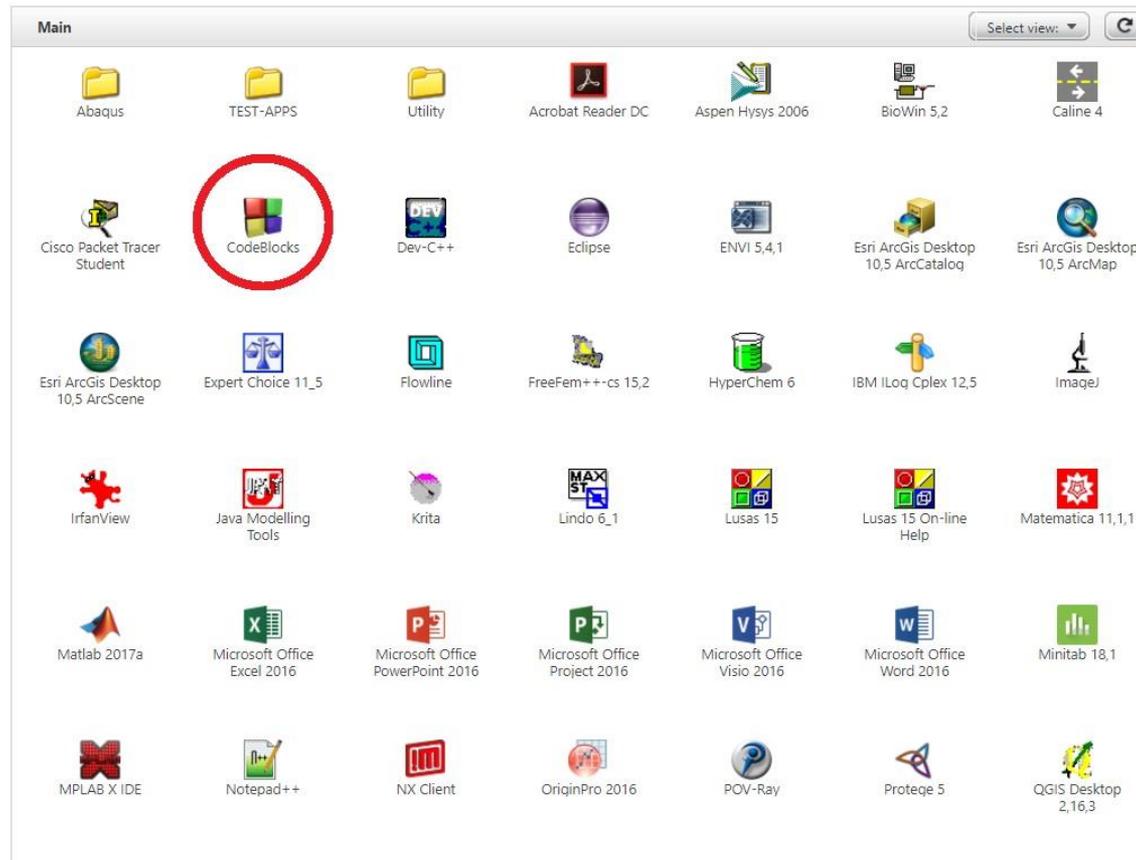
N.B.: salvare i progetti all'interno del disco di rete *Y:* in altre locazioni eseguire correttamente il compilatore e l'eseguibile stesso.

È tuttavia possibile utilizzare anche altri IDE a piacere (chiedere a noi per dettagli sui settings), ad esempio:

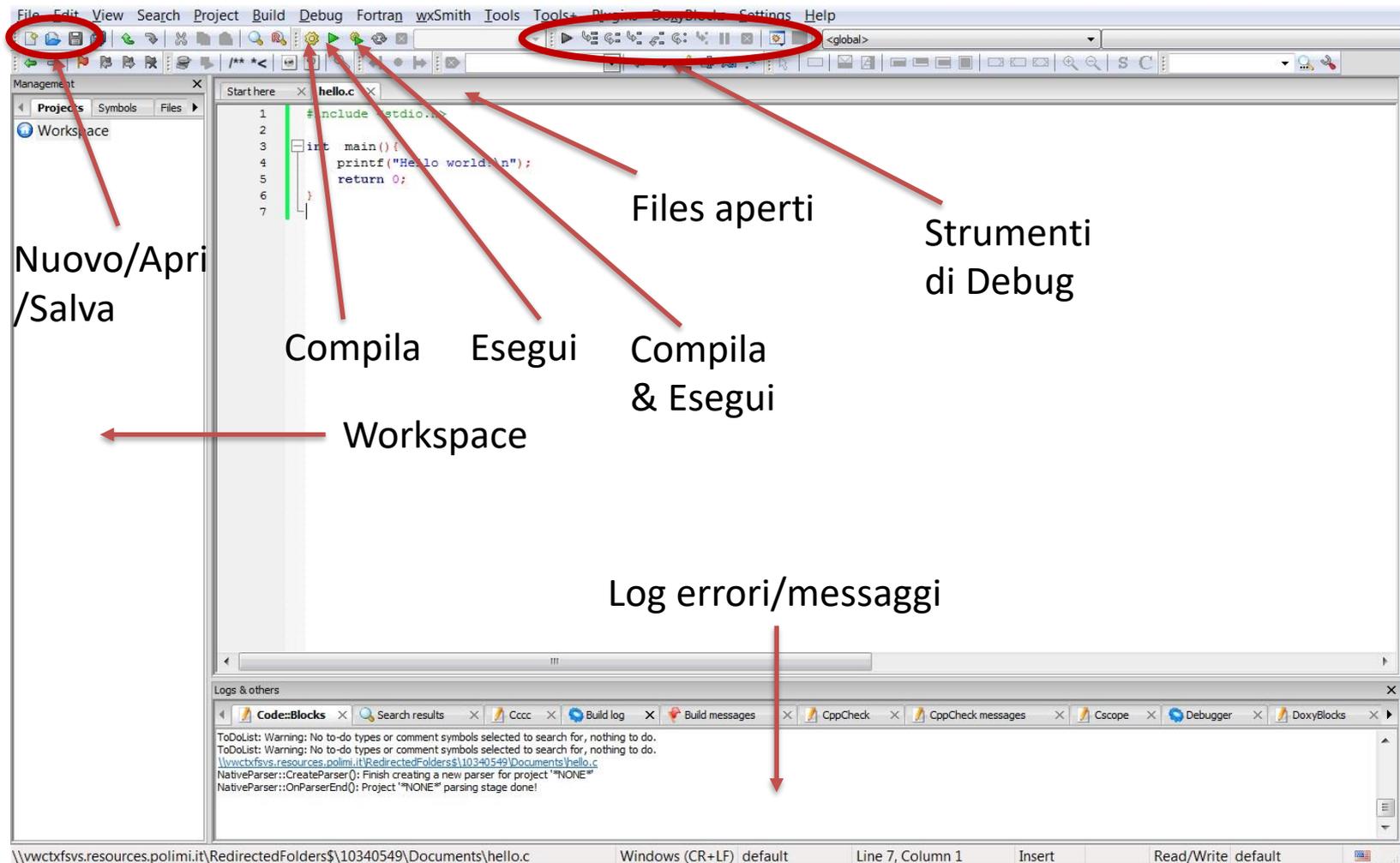
- [Dev-C++](#) (non più aggiornato)
- [Xcode](#) (per chi usa Mac OS)
- [Netbeans](#) (più complesso)
- [CLion](#)

# Virtual Desktop (cont'd)

- Schermata principale *Virtual Desktop*



# Integrated Development Environment: CodeBlocks



# CodeBlocks: creare un progetto

- *File -> New -> Project...*
- Selezionare *Console application -> Go*
- Seguire la procedura guidata:
  - Selezionare *C* nella schermata linguaggio
  - Inserire il nome del progetto
  - Scegliere la cartella di destinazione (*Y:*)
  - Lasciare le opzioni di default -> *Finish*

N.B.1: è consigliabile usare solo [A-Za-z0-9\_-.] per i nomi dei file e cartelle -> **Non usare spazi**

N.B.2: in un progetto possono esserci più file (.c e .h), tuttavia non è possibile avere più di una variabile globale/funzione con un dato nome.

- Processo con il quale una serie di istruzioni scritte un determinato linguaggio di programmazione (*codice sorgente*, e.g. *.c* o *.h*) viene tradotto in istruzioni di un altro linguaggio (*codice oggetto*, e.g. *.exe* o *.o*), quest'ultimo comprensibile ed eseguibile dalla macchina.
- Tramite interfaccia grafica (IDE)
- Tramite interfaccia a riga di comando (CLI)

# Compilazione (IDE)

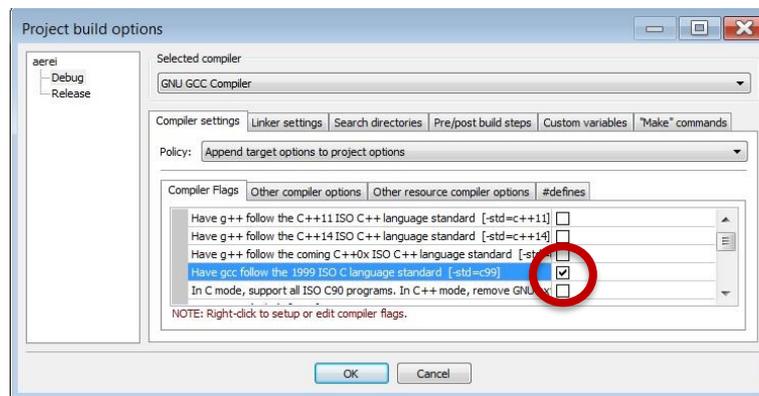
- *Build* -> *Build* (o Ctrl+F9) oppure:



- **In assenza di errori**, viene prodotto un file.exe nella stessa cartella in cui è stato salvato il codice sorgente
  - **In presenza di errori**, il codice non viene compilato e vengono mostrati dei messaggi relativi agli errori
- 
- **I messaggi di errore sono importanti!**
  - **Correggere il primo errore incontrato e ricompilare** (gli errori successivi potrebbero dipendere dal primo)

# Compilazione (IDE)

- Opzioni compilatore: *Project->Build options...*
  - Standard linguaggio: *Compiler settings->General->Have gcc follow the 1999 ISO...*



- Produrre *Messaggi di Warning*: *Warnings->Enable all common compiler warnings [-Wall]*
- Se si utilizzano librerie esterne non auto *linked*: *Linker settings->Link libraries->Add (select the file)*

# Compilazione (CLI - Linux)

- Recarsi tramite *Shell* nella cartella dove si trova il sorgente

```
$ gcc file_sorgente.c -o file_eseguibile
```

- Se si utilizzano librerie esterne (e.g., *libm* per *math.h...*)

```
$ gcc file_sorgente.c -o file_eseguibile -lm
```

- Con file sorgenti multipli

```
$ gcc file_1.c file_2.c -o file_eseguibile
```

- Generare *Messaggi di Warning*

```
$ gcc -Wall file.c -o file_eseguibile
```

# Esecuzione (IDE)

- *Build -> Run* (o Ctrl+F10) oppure:



- Solo dopo che il programma è stato compilato
- Il programma può essere eseguito anche aprendo il file .exe prodotto, ma in quel caso la finestra si chiuderà subito dopo l'esecuzione
- *Build -> Build and Run* (o F9) oppure:



- Per compilare ed eseguire con un solo comando

# Esecuzione (CLI - Linux)

- Nella cartella dove è presente il file eseguibile

```
$ ./file_eseguibile
```

## Errori comuni...

- Controllare di aver salvato il file sorgente come `.c` e non `.C++`
- Controllare di aver salvato il file nella cartella remota (`Y:`) e non nel disco locale (`C:`)
- Controllare i `';` e le *parentesi graffe*



# Debugging (IDE cont'd)

- Per proseguire con l'esecuzione del programma:
  - *Start/Continue*: Prosegue l'esecuzione fino al successivo *breakpoint*.



- *Next line* (o F7): Esegue la linea successiva



- *Step into*: Entra nella funzione



# Primi passi: Hello world!

Creare un nuovo progetto come indicato in precedenza e scrivere il seguente codice:

```
#include <stdio.h>

int main(){
    printf("Hello World!\n");
    return 0;
}
```

Compilare ed eseguire

## Primi passi: Hello world! (cont'd)

Partendo dal file scritto in precedenza, inseriamo degli errori e prendiamo familiarità con il debugging.

```
#include <stdio.h>

int main(){
    printf("Hello world!");
    return 0;
}
```

- Inserire un *breakpoint* alla riga 4 ed avviare il debug.
- Inserire alcuni errori non sintattici e analizzare tramite debugging.

# I/O: printf(...)

- *int printf(...)*: permette di mostrare dei messaggi a schermo, la sintassi:

```
printf("Hello world!");
```

Testo del messaggio

- Posso formattare il testo inserendo anche dei caratteri speciali (a capo, tab...)

```
printf("Hello world\n");
```

Simbolo  
'A capo'

- Posso inserire anche delle variabili da mostrare nel messaggio:

```
printf("Messaggio numero %i", num);
```

Specificatore di formato

Variabile

- **Caratteri speciali** (considerati come singoli caratteri)
  - \n A capo
  - \t Tabulazione
  - \' Apostrofo
  - \” Doppi apici
  - \\ \
- **Specificatori di formato:**
  - %d Interi
  - %f float, double
  - %e decimali, in notazione esponenziale
  - %c caratteri
  - %s stringa

# I/O: scanf(...)

- *int scanf(...)*: permette di leggere un insieme di caratteri da tastiera:

```
scanf("%d", &var);
```

Specificatore di formato

Variabile in cui  
memorizzare i  
caratteri letti

- I caratteri vuoti (spazio, tab...) sono ignorati
- La funzione ritorna il numero di caratteri letti
- **L'argomento di scanf deve essere sempre un puntatore!!**  
Nel caso si utilizzano variabili non puntatori si antepone il simbolo **&**

- Quando leggo più di un carattere e nel frattempo stampo:

```
scanf("%c", &ch); // a\nprintf("%c", ch); // ch = 'a'\nscanf("%c", &ch); // b\nprintf("%c", ch); // '\n'
```

- La seconda scanf legge il carattere "a capo"
- Risolvere inserendo uno spazio bianco o un '\n' prima di '%c'

```
scanf(" %c", &ch); // a\nprintf("%c", ch); // ch = 'a'\nscanf("\n%c", &ch); // b\nprintf("%c", ch); // ch = 'b'
```

# Esercizio 1.1: Calcolo IVA

Scrivere un programma che, dato un costo ivato di un articolo (in float), calcoli il costo senza IVA (IVA al 22%).

Esempio:

Inserisci: 122 -> *100.00 + IVA 22.00*

## Hints:

- Il costo finale è determinato dal costo + IVA, dove IVA è il 22% del costo.

## Esercizio 1.2: Volume di un cilindro

Scrivere un programma che, dati il raggio e l'altezza di un cilindro, ne calcoli il volume.

Volume del cilindro:  $V = \pi r^2 h$

### Hints:

- Consideriamo  $\pi = 3,14$  per semplicità

## Esercizio 1.3: Volume di un cono

Scrivere un programma che, dati il raggio e l'altezza di un cono, ne calcoli il volume.

$$\text{Volume del cono: } V = \frac{\pi r^2 h}{3}$$

### Hints:

- Consideriamo  $\pi = 3,14$  per semplicità

## Esercizio 1.4: Quadrato perfetto

---

Dato in input un numero intero  $A > 0$ , verificare se  $A$  è un quadrato perfetto, cioè se esiste un numero  $B$  tale che  $A = B * B$ .

## Esercizio 1.5: Potenze di numeri

Scrivere un programma che, dati in ingresso due numeri interi positivi  $N$  e  $K$ , stampa potenze di  $N$  con esponenti da 1 a  $K$ .

Esempio:

Base: 4

Esponente massimo: 4

*Potenze: 4 16 64 256*

## Esercizio 1.6: Anno bisestile

Scrivere un programma che, dato un anno inserito dall'utente come numero intero, dica se è bisestile o meno.

Esempio:

Inserisci anno: 1777

*L'anno 1777 non è bisestile!*

### Hints:

- Un anno è bisestile se è multiplo di 4. Se però è multiplo di 100 non è bisestile, con l'eccezione dei multipli di 400 che sono bisestili.
- Usare `scanf` per leggere il numero
- Utilizzare l'operatore `%` per calcolare il resto della divisione intera
- Combinare le varie condizioni in costrutti condizionali annidati o mediante operatori AND e OR logici

## Esercizio 1.7: Gara di tuffi

Nelle gare di tuffi a 5 giudici il punteggio finale è doppio della somma dei voti ottenuta eliminando il più alto ed il più basso.

Scrivere un programma che legga in input 5 valori e calcoli in uscita il voto finale.

Esempio:

~~8.0~~, 7.5, 7.5, 7.5, ~~7.0~~ = 22.5 x 2.0 = 45.0

([http://www.federnuoto.it/pdf/t\\_reg\\_tec\\_09-13.pdf](http://www.federnuoto.it/pdf/t_reg_tec_09-13.pdf))

### Hints:

- Utilizzare un ciclo
- Controllare ad ogni iterazione se il voto supera o è inferiore rispettivamente al max o min trovato finora.

## Esercizio 1.7: Gara di tuffi (cont'd)

Si replichi la logica dell'es. precedente, ma si inseriscano dati per  $n$  tuffatori.

Per ciascuno si inserisca:

- Il numero di pettorale (intero positivo)
- I 5 voti dei 5 giudici
- Si calcoli il punteggio assegnato (con la regola dell'es. precedente)

Alla fine il programma stampi il tuffatore che ha vinto (pettorale e voto)

### Hints:

- Utilizzare un ciclo, ciclando su ogni tuffatore.

# Esercizio a casa 1.1: Divisori Comuni

Scrivere un programma che, dati due numeri interi positivi inseriti dall'utente, stampi a video i divisori comuni maggiori di 1.

Esempio:

Inserisci due numeri positivi: 18 72

*I divisori comuni sono: 2 3 6 9 18*

Se l'unico divisore comune è 1, stampare a video che i due numeri sono **coprimi**.

**Hints:**

- Usare l'operatore % per calcolare il resto della divisione intera

# Esercizio a casa 1.2: Numeri primi di Mersenne

Dato in input un numero primo  $P$ , verificare che il numero di Mersenne  $M = 2^P - 1$  sia anch'esso primo e nel caso stamparlo.

## Hints:

- Utilizzare il flowchart prodotto dagli esercizi di esempio [qui](#) (Es 3.5)
- Avete già visto come verificare se un numero è primo