INFORMATICA (PER AEROSPAZIALI)

A.A. 2018-19

Laboratorio n°1

Prof. Christian Pilato

Dott. Michele Zanella

HEAP LAB

Info Logistiche

- Contatti:
  - michele.zanella@polimi.it
    - 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)
- Sito web del laboratorio:
  - Pagina del corso BeeP (Cartella Laboratori)
  - http://zanella.faculty.polimi.it/teaching/informatica-aero
- **Nota per le mail:**

  Oggetto: *[INFO-AERO] il vostro oggetto*

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

IDE

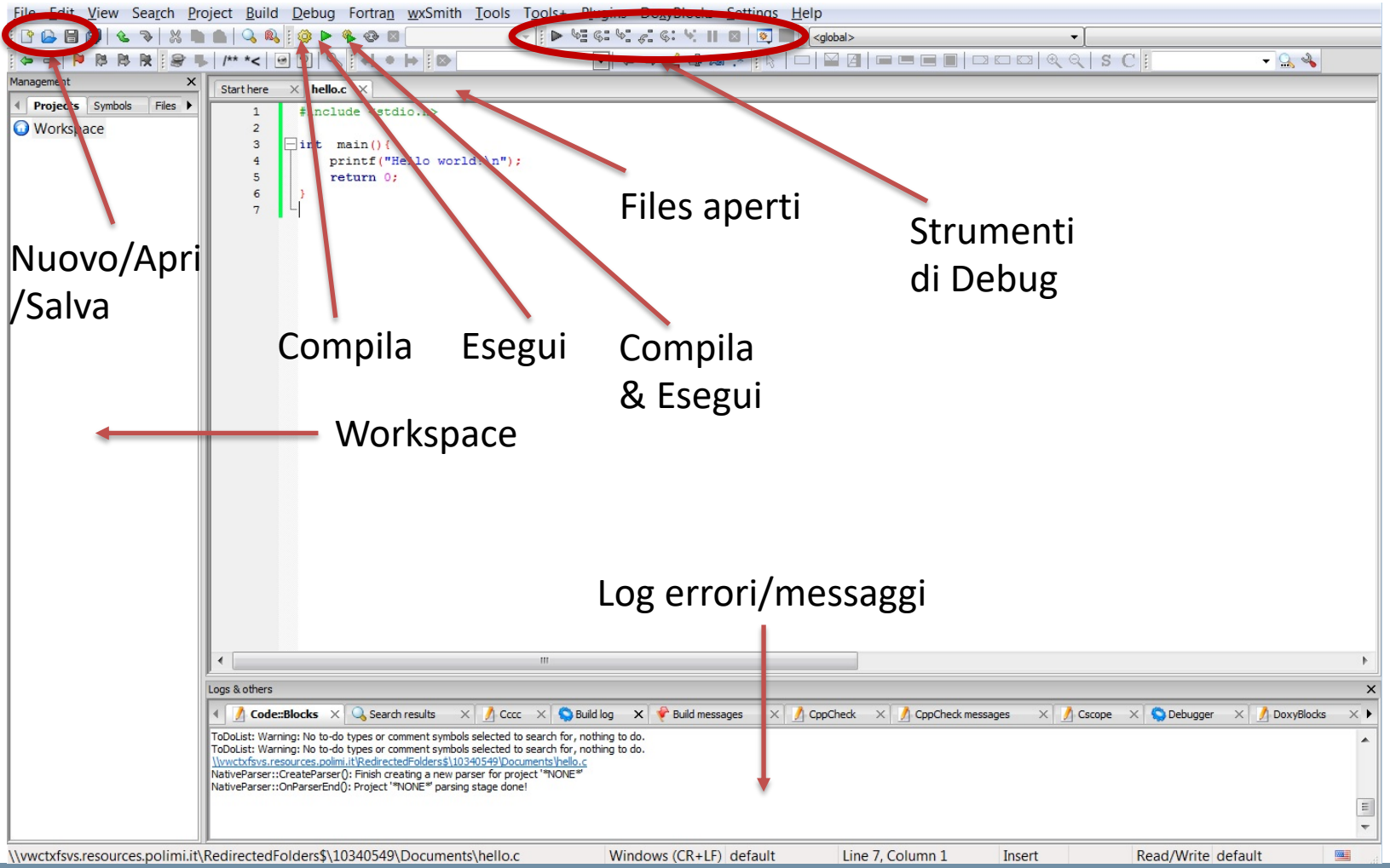
Utilizzeremo un IDE durante il laboratori: *CodeBlocks*

http://www.codeblocks.org/

È tuttavia possibile utilizzare anche altri IDE a piacere (non si fornirà supporto), ad esempio:

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

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.

Compilazione

- 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)

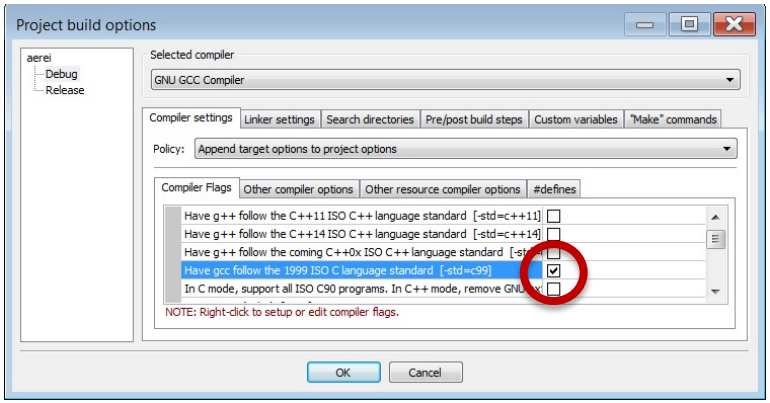
- *Build -> Build* (o Ctrl+F9) oppure:

  - 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)*

Virtual Desktop (in alternativa)

- 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 laboratori: *CodeBlocks*

http://www.codeblocks.org/

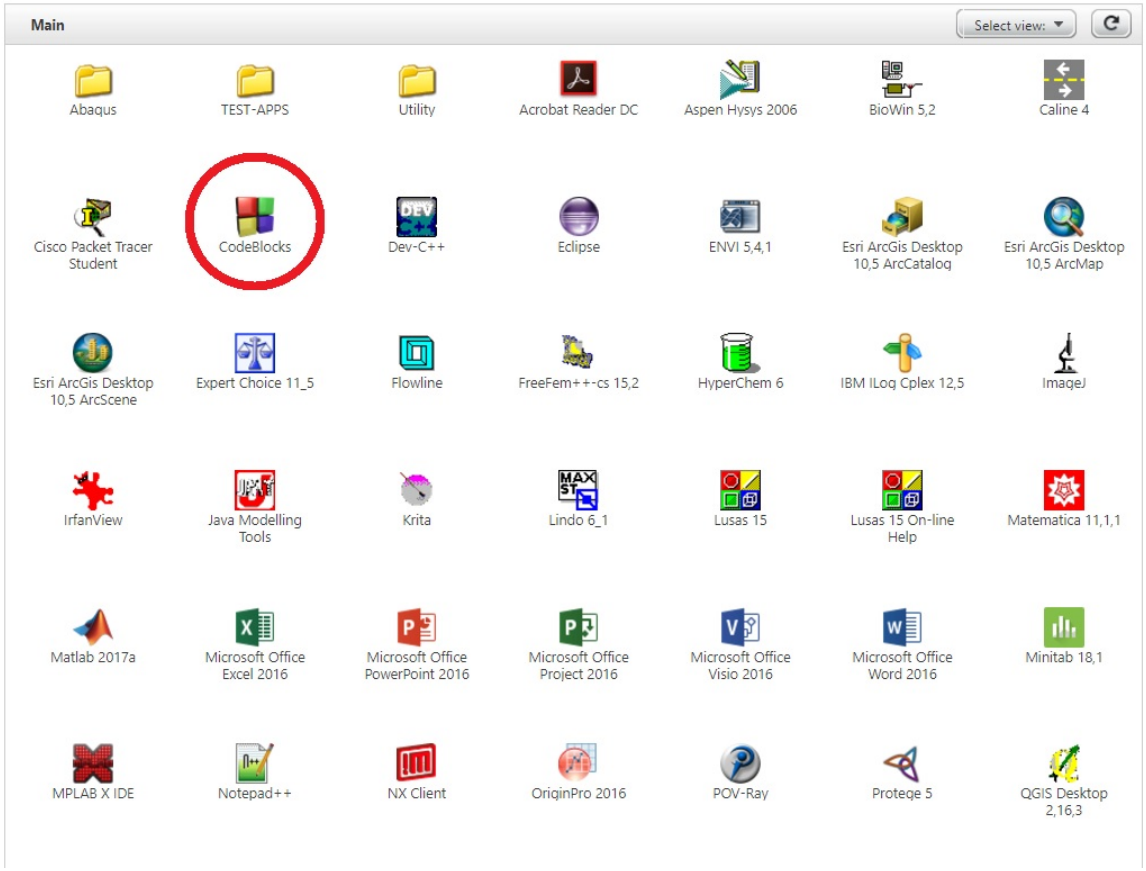
- Cliccare su *CodeBlocks* ed aprire il file *launch.ica* con il client *Citrix*

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

N.B.2: Oppure caricare i file fatti durante il laboratorio nella propria area riservata di BeeP.

Virtual Desktop (cont'd)

- Schermata principale *Virtual Desktop*



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

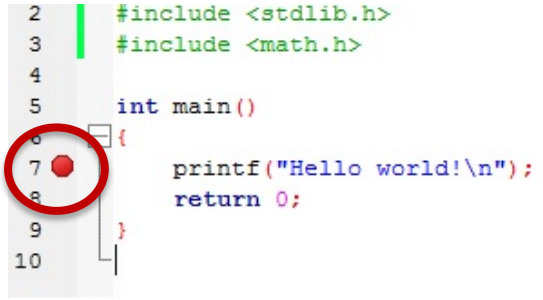
Errori comuni…

- 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)

- *Debug -> Start/Continue* (o F8) oppure:

  - Serve per analizzare il flusso del programma ed individuare gli errori in esecuzione (e.g., valori non corretti nelle variabili, condizioni non soddisfatte, etc…)
  - È necessario indicare le righe alle quali si vuole fermare il programma tramite dei *breakpoints,* ogni volta che l'esecuzione passa per quelle righe di codice il programma viene messo in pausa.

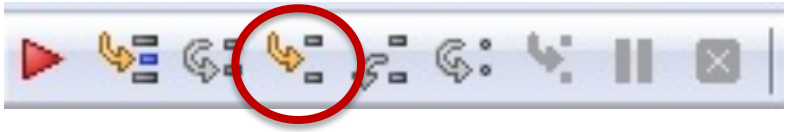


Debugging (IDE cont'd)

- 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 una variabile e prendiamo familiarità con il debugging.

```
#include <stdio.h>

int main(){
        printf("Hello world!");
        int i = 1;
        i++;
        return 0;
}
```

- Inserire un *breakpoint* alla riga 5 ed avviare il debug.
- Analizzare tramite debugging il valore della variabile "i".

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…)

Simbolo 'A capo'

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

- Posso inserire anche delle variabili da mostrare nel messaggio:

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

Specificatore di formato

Variabile

I/O: printf(…) (cont'd)

- **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 **&**

I/O: Problemi comuni

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

```
scanf("%c", &ch); // a\n
printf("%c", ch); // ch = ‘a’
scanf("%c", &ch); // b\n
printf("%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\n
printf("%c", ch);   // ch = ‘a’
scanf("\n%c", &ch); // b\n
printf("%c", ch);   // ch = ‘b’
```

Cicli

- **For**

```
for(init-expr; test-expr; increment-expr) {
    statement
}
```

  - *Init-expr:* dichiara ed inizializza uno o più contatori
  - *est-expr:* se viene soddisfatta (*true*) allora viene eseguita la *statement*, altrimenti il ciclo termina
  - *Increment-expr:* viene eseguita alla fine di un'iterazione e incrementa uno o più contatori.

```
for(int i=0; i<2; i++)
        printf("%d\t",i);
```

  - **Risultato: *0* *1***

Cicli (cont'd)

- **While**

```
while(condition) {
    statement
};
```

La *statement* viene eseguita finchè la condizione è verificata

- **Do….while**

```
do {
    statement
}while(condition);
```

La condizione viene verificata alla fine, la *statement* è eseguita almeno una volta.

Condizioni

- **If**

```
if(boolean_expression){
    statement
}
```

Lo *statement* viene eseguito se l'espressione è verificata

- **If…else**

```
if(boolean_expression) {
    statement1
} else {
    statement2
}
```

Lo *statement1* viene eseguito se l'espressione è verificata **altrimenti** viene eseguito lo *statement2*

Condizioni

- **Switch**

```
switch(variable) {
    case value:
        statement1;
        …
    default:
        default_statement;
}
```

La variabile viene verificata nei diversi casi, se per uno è vero ne viene eseguito il relativo *statement*. Altrimenti viene eseguito il caso di *default*.

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: 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.

Hint

- Vedere Flowchart dell'esercizio 2.1 tra gli esercizi per casa dell'esercitazione 1.

Esercizio 1.3: 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 1.4: 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 calcoalre il resto della divisione intera
- Combinare le varie condizioni in costrutti condizionali annidati o mediante operatori AND e OR logici

Esercizio 1.5: 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

Hints:

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

Esercizio 1.5: 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 1.6: Numeri primi di Mersenne

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

Hints:

- Vedere Flowchart dell'esercizio 2.5 tra gli esercizi per casa dell'esercitazione 1.