

Fondamenti di Informatica
Prof. Mauro Negri

Laboratorio A.A. 2019-2020

Ing. Paolo Visentini

Contatti e Materiale

- Responsabile di laboratorio:
 - Ing. Paolo Visentini
 - e-mail: paolo.visentini@polimi.it
- Materiale del laboratorio scaricabile da “Beep”:
 - <https://beep.metid.polimi.it/>

Programma di oggi

- Organizzazione del laboratorio
- Presentazione degli strumenti per la programmazione
- Primi esercizi

Organizzazione

- aula B8 2.2
- 8 sessioni di laboratorio da 2 ore
- 2 squadre
 - SQ1: cognomi BRA – CASTAZ
 - SQ2: cognomi CASTABA – COLZ
- 2 turni
 - I turno martedì ore 9:15 – 11:15
 - II turno martedì ore 11:15 – 13:15
 - eccezione ultima sessione: unico turno a squadre riunite ore 9:15 - 11:15

Organizzazione

- Ogni settimana le squadre cambiano turno (controllare il calendario del corso)

	24 settembre	8 ottobre	15 ottobre	...
--	--------------	-----------	------------	-----

- | | | | | |
|------------|-------|---|---|-----|
| • Turno I | Sq. 1 | 2 | 1 | ... |
| • Turno II | Sq. 2 | 1 | 2 | ... |

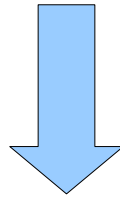
Regole

- Non è obbligatoria la frequenza ma in aula si ha la possibilità di potersi esercitare confrontandosi direttamente con i colleghi e ponendo domande al responsabile e al tutor
- Si possono utilizzare calcolatore, libri, carta e penna per sviluppare i propri elaborati - scritti in linguaggio C

Strumenti per la programmazione

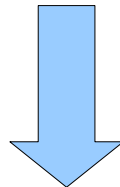
Scrivere ed eseguire un programma

- Problema da risolvere descritto in linguaggio naturale



- Algoritmo descritto in un linguaggio simbolico (linguaggio C) e salvato in un file di testo

CODICE SORGENTE



- Algoritmo rappresentato in codice macchina eseguibile dal calcolatore

PROGRAMMA ESEGUIBILE



GNU Compiler Collection GCC

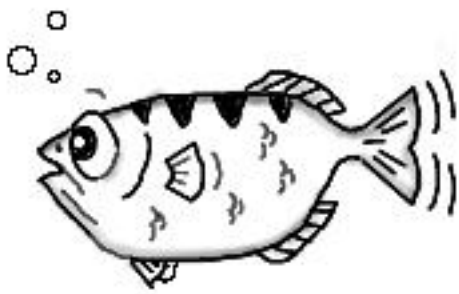
- Occorre uno strumento che riesca a ricavare dal codice facilmente leggibile dall'uomo un programma eseguibile dalla macchina. Questo strumento viene chiamato **compilatore**
- Compilatore “libero” gcc
 - <http://gcc.gnu.org/>
 - nasce come compilatore C (“GNU C Compiler”) ma poi diviene una collezione di compilatori per diversi linguaggi

Il processo di Compilazione

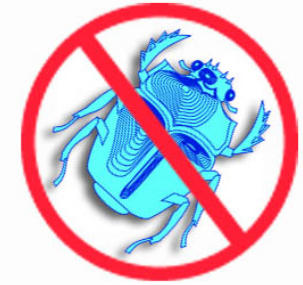


La compilazione avviene in tre fasi:

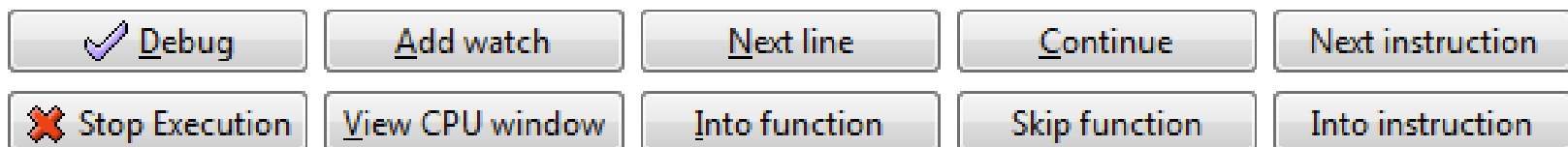
1. Il file di testo con il codice C da compilare viene trattato da un pre-processore che analizza le direttive (cioè `#define` e `#include` all'inizio del programma) e toglie i commenti sostituendoli con degli spazi
2. Viene creato un file in formato oggetto (binario)
3. Entra in funzione il linker che si occupa di assemblare tutti i “pezzi” (prendendoli anche da delle librerie) e creare un unico file eseguibile



Debugging



- Un particolare strumento detto “debugger” (ad es: gdb - <http://www.gnu.org/software/gdb/>) permette l'esecuzione controllata di un programma per testarne il comportamento e scovarne gli errori (=debugging).
- È possibile seguire l'esecuzione passo dopo passo, istruzione dopo istruzione
- È possibile interrompere l'esecuzione in punti “strategici” (detti “breakpoint”)
- E' possibile osservare (“watch”) e modificare il valore delle variabili in un certo istante di esecuzione...



IDE

IDE (Integrated Development Environment): applicazioni che forniscono in un unico pacchetto tutto quello che serve per la programmazione (gestione del progetto, compilazione, esecuzione, debugging, ...) in uno o più linguaggi

- Code::Blocks, Orwell Dev C++, Apple Xcode, Microsoft Visual Studio, Eclipse, Netbeans, ...

NB: parentesi graffe “{ }” con la tastiera italiana:
Alt Gr + Shift + []

IDE

- La cartella che rappresenta lo spazio di lavoro di un IDE viene solitamente chiamata **“Workspace”**
- All'interno di un “Workspace” si creano i progetti **“Projects”**
- I progetti possono avere delle dipendenze tra loro. Ad esempio un progetto può essere una libreria che fornisce delle funzioni utili ad un progetto dove viene implementato l'algoritmo principale.

IDE CONSIGLIATI



- Code::Blocks
 - Per Windows, Mac OS X e Linux



- (Orwell) Dev C++
 - Solo per Windows



- Xcode
 - Solo per Mac OS X

IDE Code::Blocks

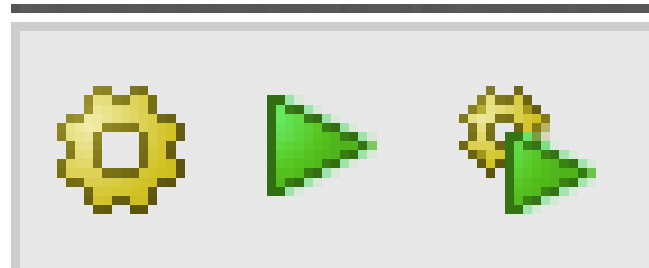


- Per Windows, Mac OS X e Linux
- Scaricabile gratuitamente dal sito SourceForge:
<https://sourceforge.net/projects/codeblocks/files/Binaries/17.12/>
- Windows → codeblocks-17.12mingw-setup.exe
 - Include “mingw” (Minimalist GNU for Windows) che è una versione del compilatore GCC per Windows
- Mac → codeblocks-17.12_OSX64.dmg
 - Occorre comunque prima installare Xcode dall’App Store per avere a disposizione il pacchetto “Command Line Tools” e il compilatore GCC

IDE Code::Blocks

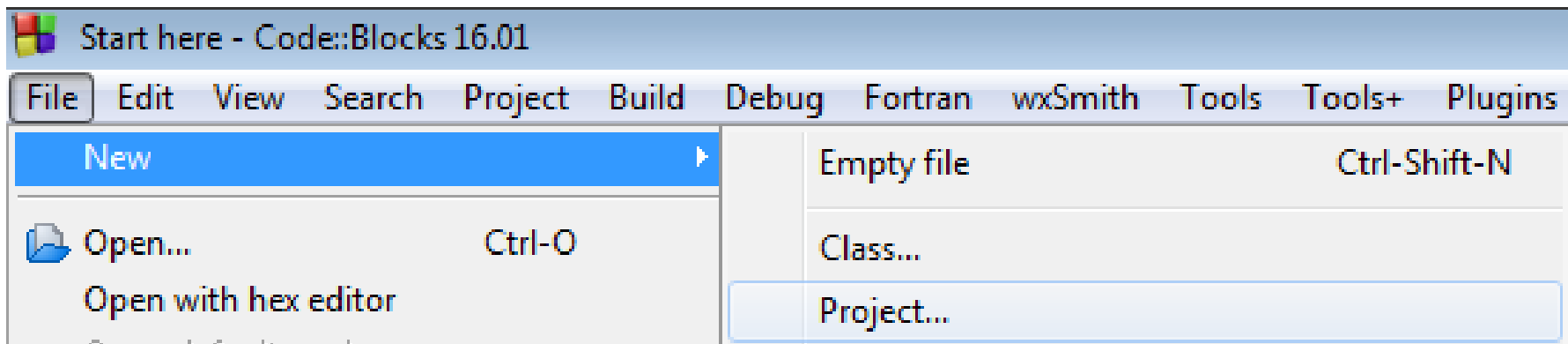


- Pulsanti importanti:
 - Compila
 - Esegui
 - Compila ed esegui



Creazione/Apertura di un Progetto

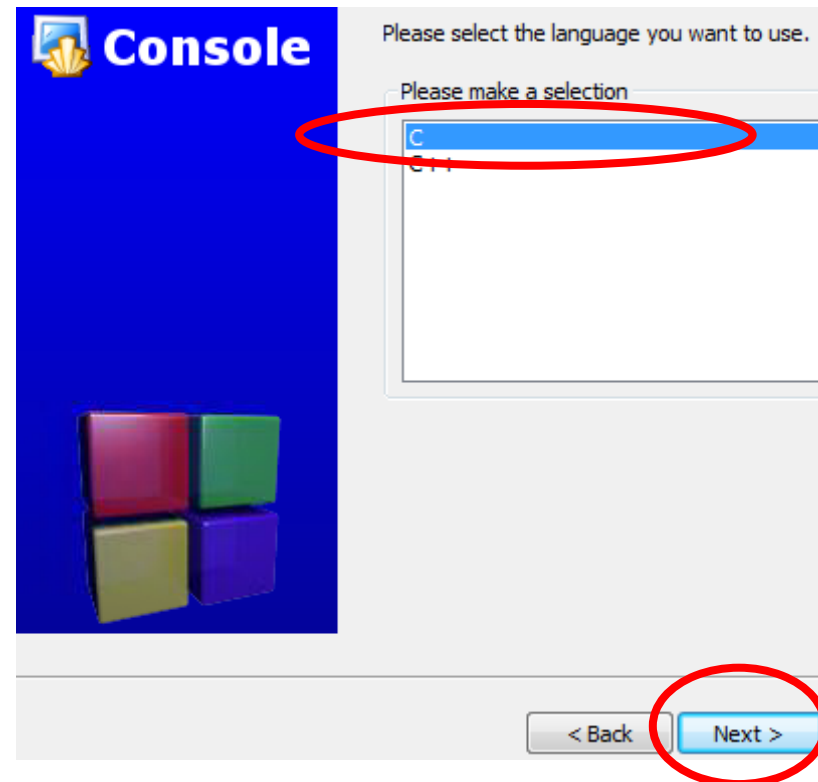
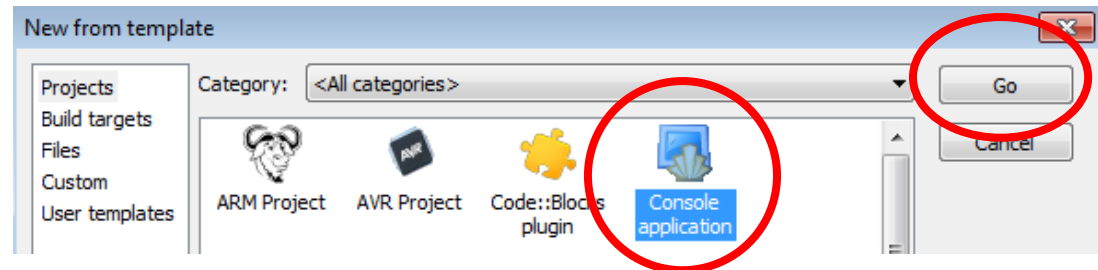
- File → New → Project
- (se già esistente) File → Open...



- Attenzione: meglio NON inserire spazi e NON usare caratteri accentati tipo “è, é, ù” nei nomi dei Progetti, delle Cartelle e dei File. In un progetto può esistere un solo “main”.

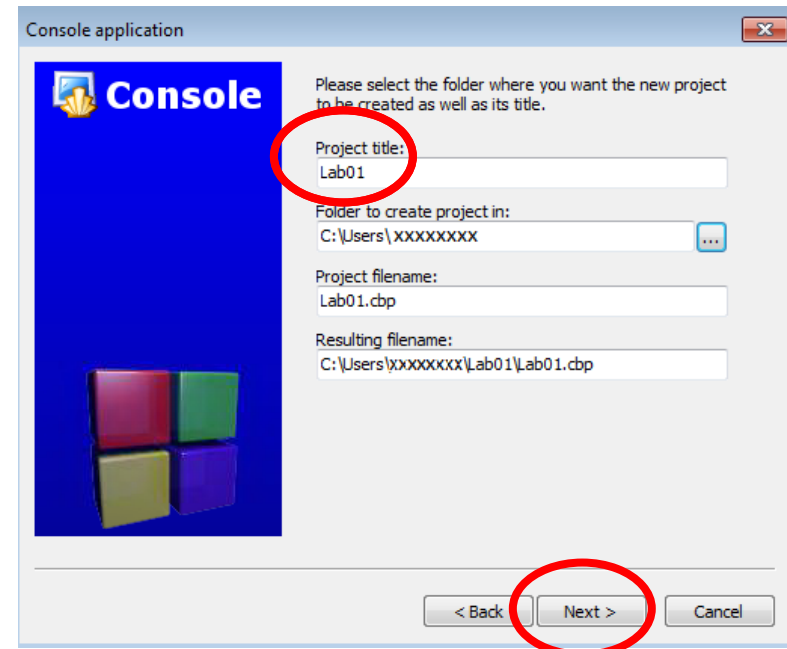
Creazione di un Progetto

- Console Application
- Go
- Next >
- C
- Next >



Creazione di un progetto

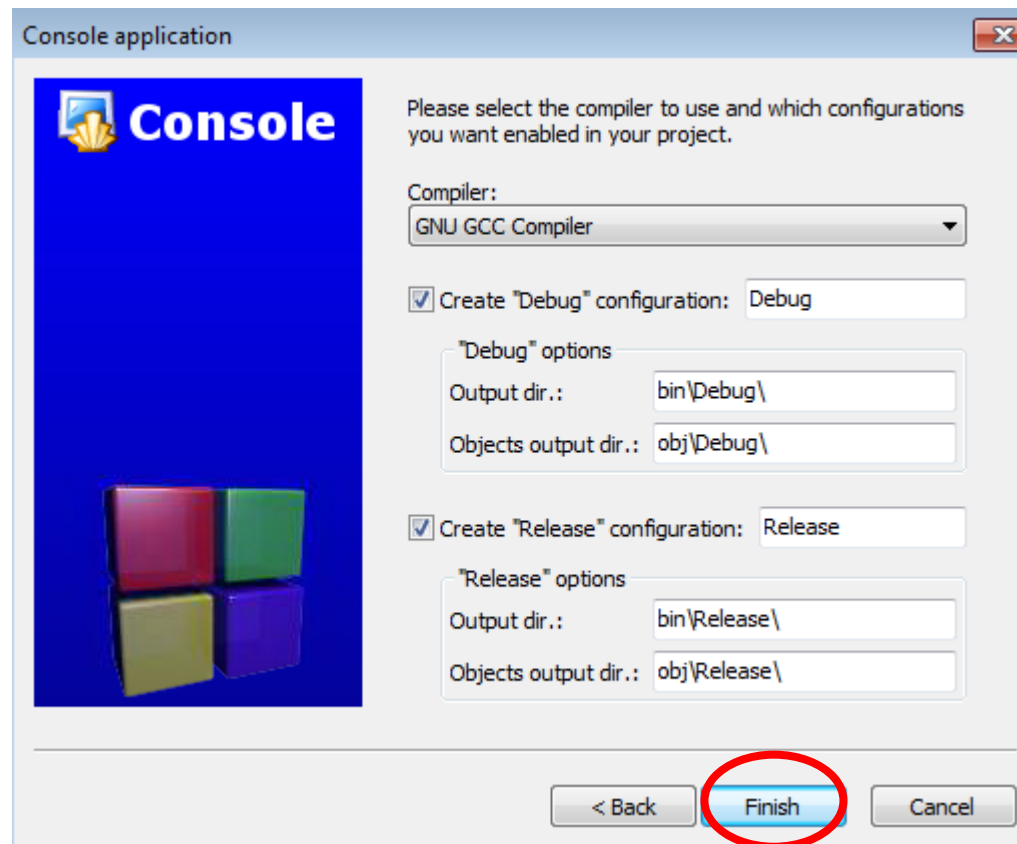
- Project title
 - Per esempio: “Lab01”



- Folder ...
- Next >

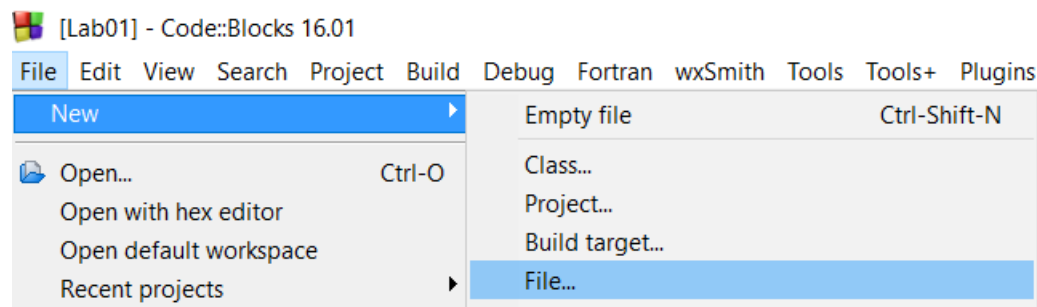
Creazione di un progetto

- Finish

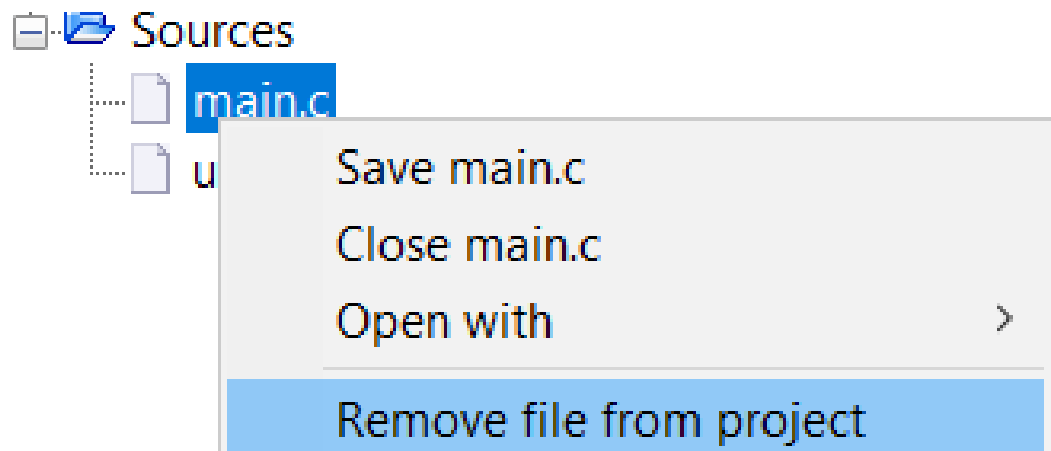


Aggiunta/Rimozione file dal progetto

- File → New → File...

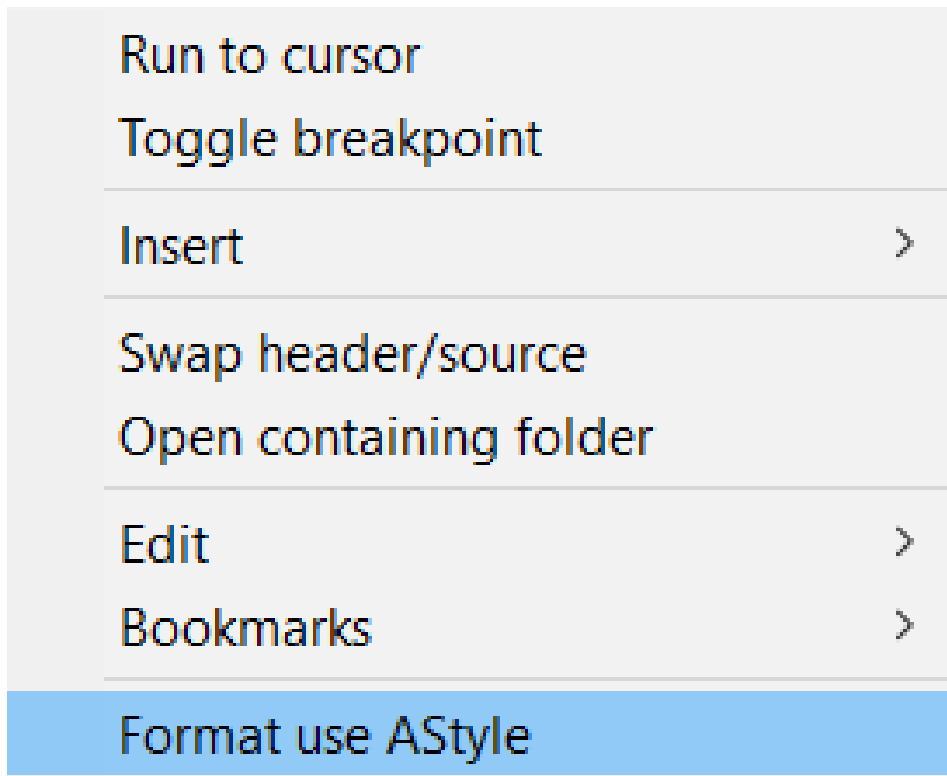


- Tasto destro → “Remove file from project”

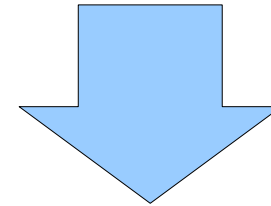


Formattazione automatica del codice

- Tasto destro sul codice → “Format use AStyle”



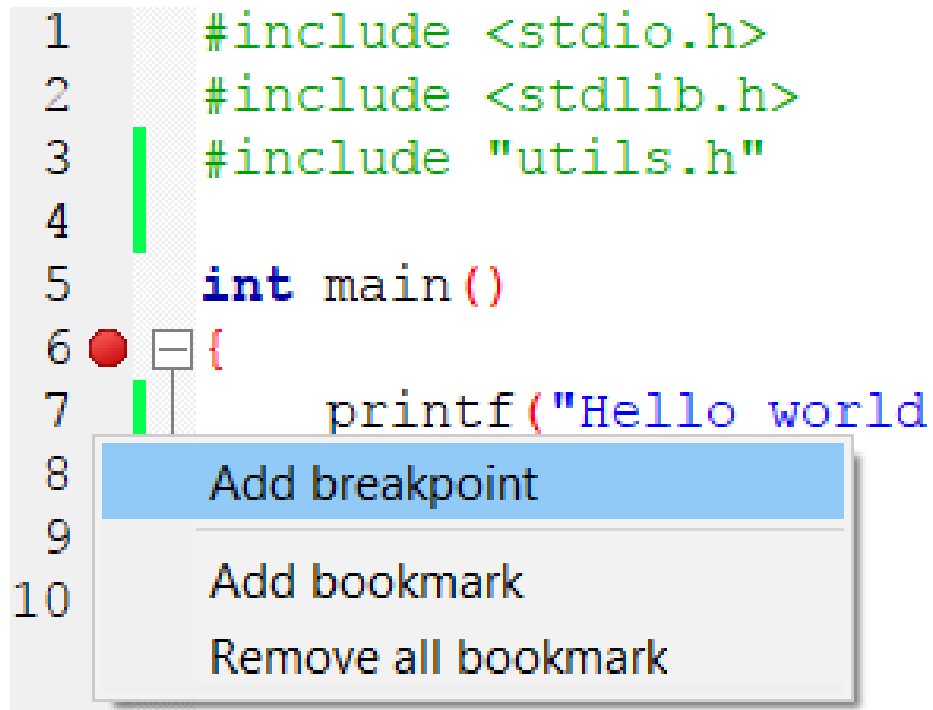
```
int a; scanf("%d",&a); if (
    a < 0) {a= - a;}
else{    a++;    }
```



```
int a;
scanf("%d", &a);
if (a < 0)
{
    a= - a;
}
else
{
    a++;
}
```

Debugging

- I breakpoint si aggiungono/tolgono facendo click vicino al numero della riga (oppure tasto destro → “Add breakpoint”). Compare un pallino rosso che segnala la presenza del breakpoint.



The screenshot shows a code editor with a C program. The code is as follows:

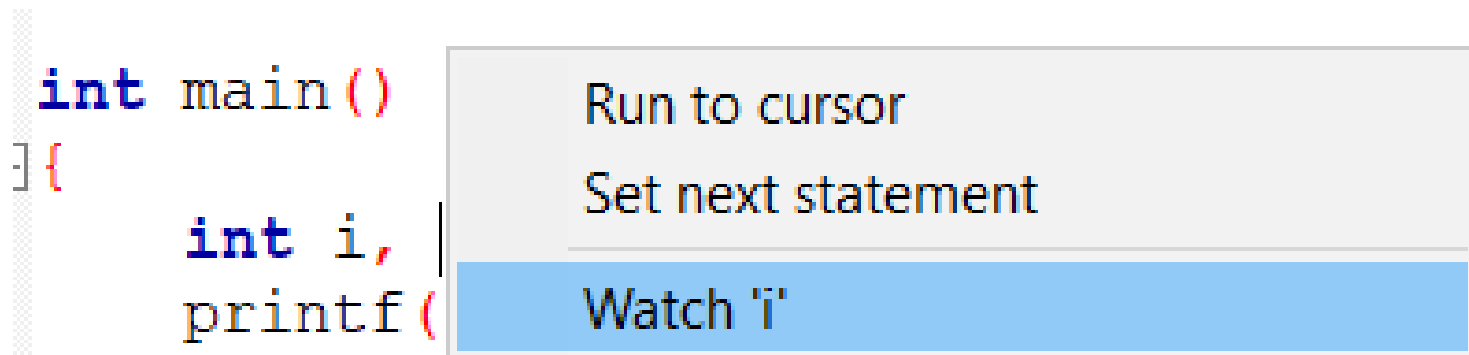
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "utils.h"
4
5  int main()
6  {
7      printf("Hello world\n");
8  }
9
10
```

A red dot, representing a breakpoint, is placed on the left margin next to line 6. A context menu is open, showing the following options:

- Add breakpoint
- Add bookmark
- Remove all bookmark

Debugging

- I watch si aggiungono facendo click col tasto destro sulla variabile da osservare e selezionando la voce “watch”



- Poi i valori della variabile si possono osservare nella finestra “Watches”

Watches (new)	
Function arguments	
<input type="checkbox"/> Locals	
<code>i</code>	<code>0</code>

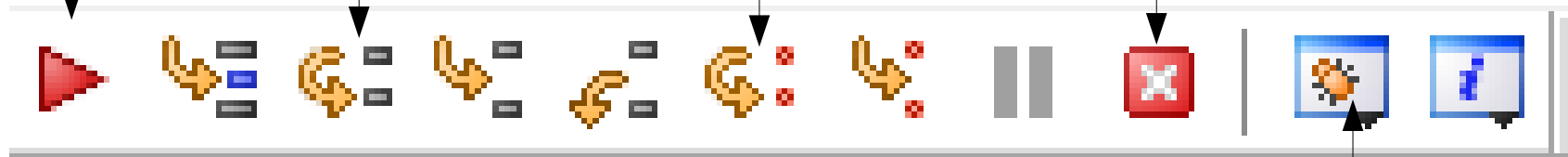
Debugging

Debug/Continue

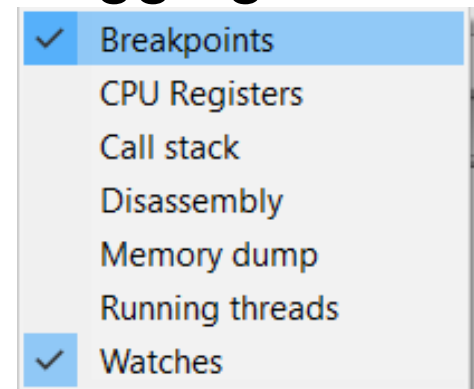
Next line

Next instruction

Stop debugger

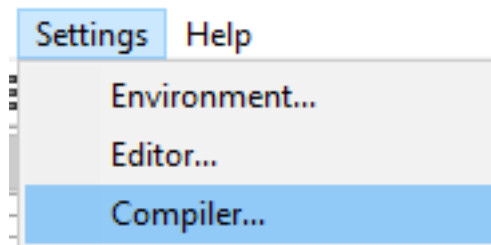


Debugging Windows

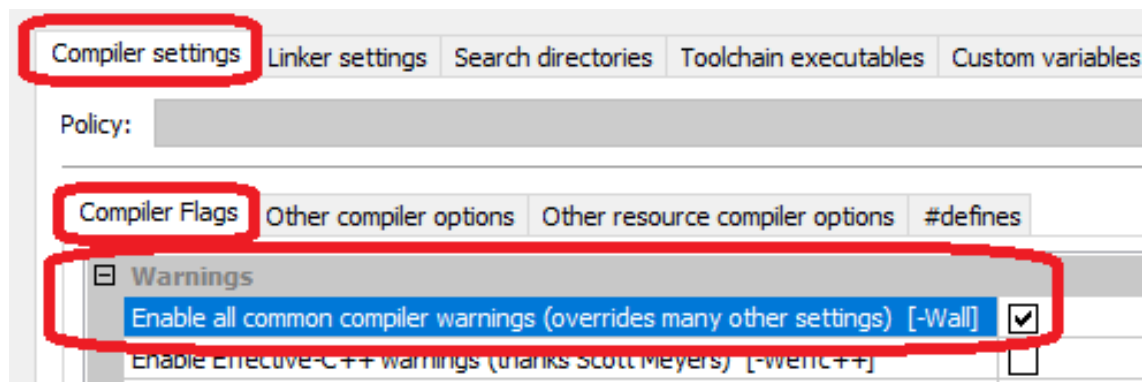


Mostrare tutti i Warning (-Wall)

- Menù “Settings” → Voce “Compiler...”



- Tab “Compiler settings” → Tab “Compiler Flags” → Sezione “Warnings” → Spunta “Enable all common compiler warnings (overrides many other settings [-Wall])”



IDE Dev C++

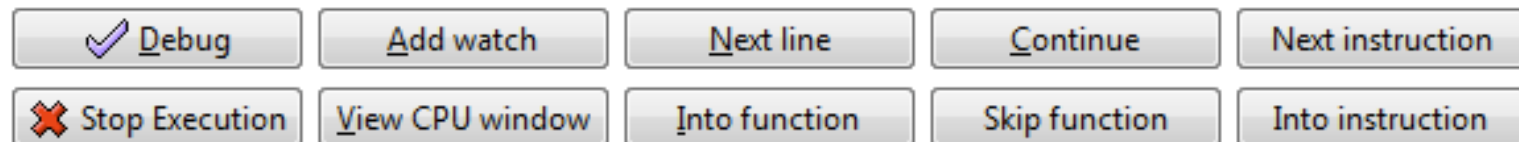


- Solo per Windows
- Scaricabile gratuitamente dal sito SourceForge:

[https://sourceforge.net/projects/orwellddevcpp/files/Setup Releases/](https://sourceforge.net/projects/orwellddevcpp/files/Setup%20Releases/)

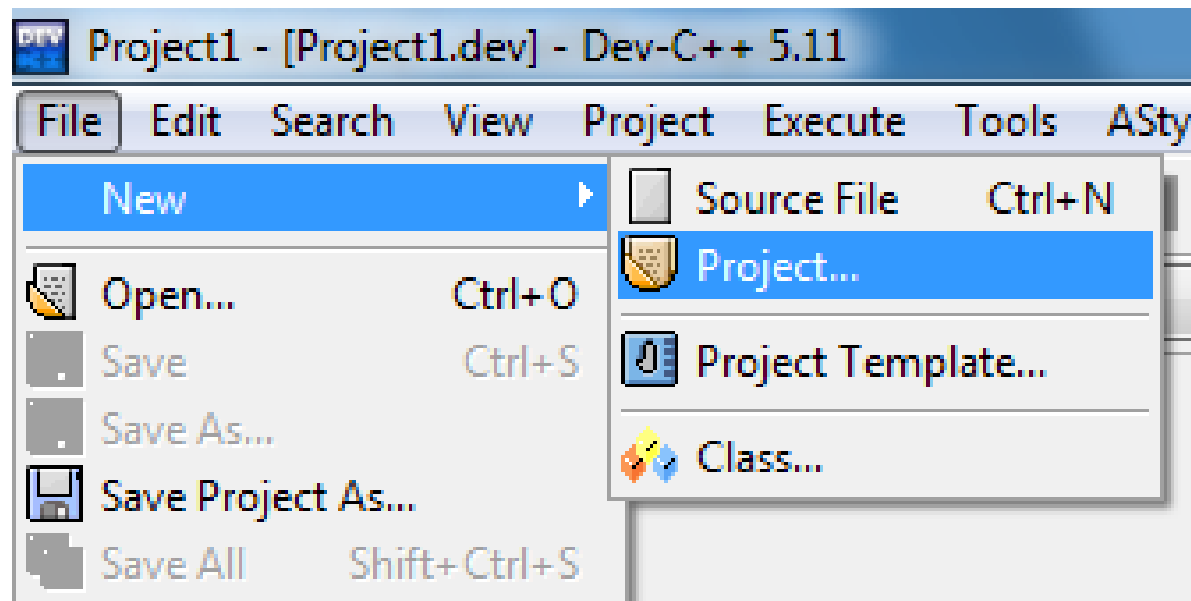
- Dev-Cpp 5.11 TDM-GCC 4.9.2 Setup.exe
- Pulsanti importanti:

- compila
 - esegui
 - compila ed esegui
 - debug



Creazione/Apertura di un Progetto

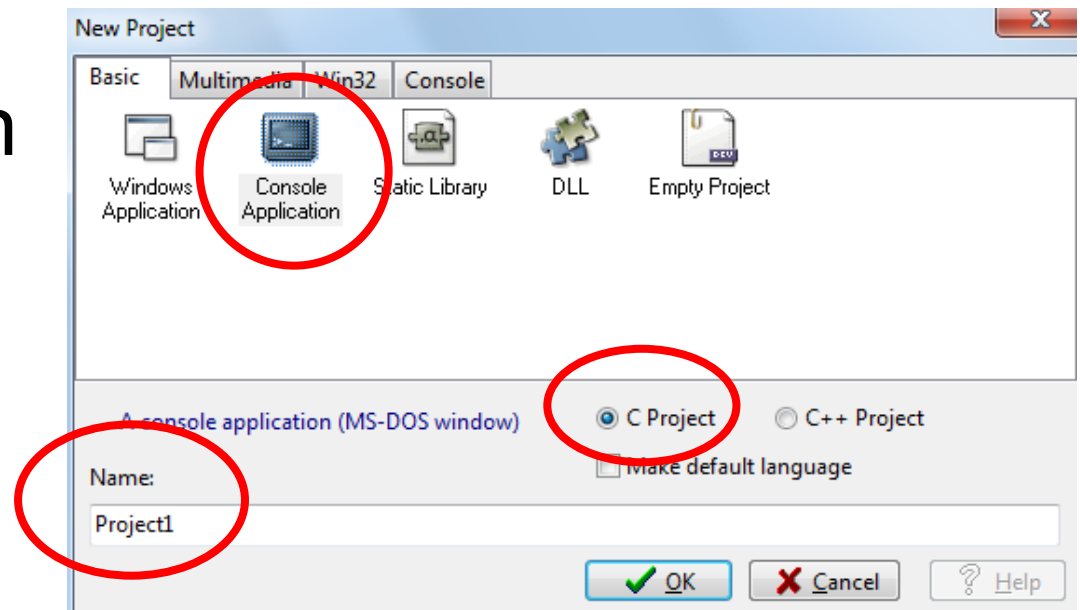
- File → New → Project
- (se già esistente) File → Open



- Attenzione: meglio NON inserire spazi nei nomi dei Progetti, delle Cartelle e dei File.

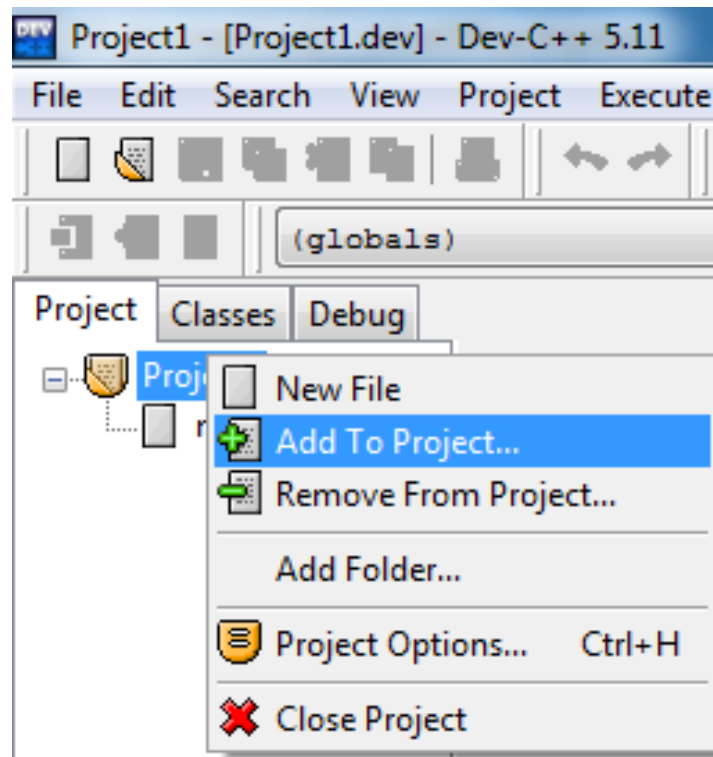
Creazione di un Progetto

- Console Application
- C Project
- Nome
- Ok



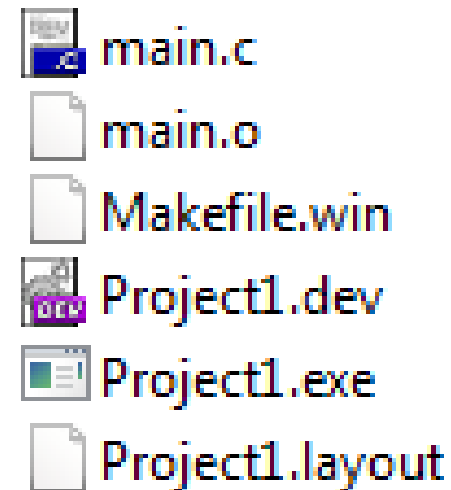
Aggiunta di file al progetto

- Tasto destro sul progetto -> “New File” per nuovo o “Add to Project” per file esistente
- Attenzione: aggiungere solo file con estensione .h e .c



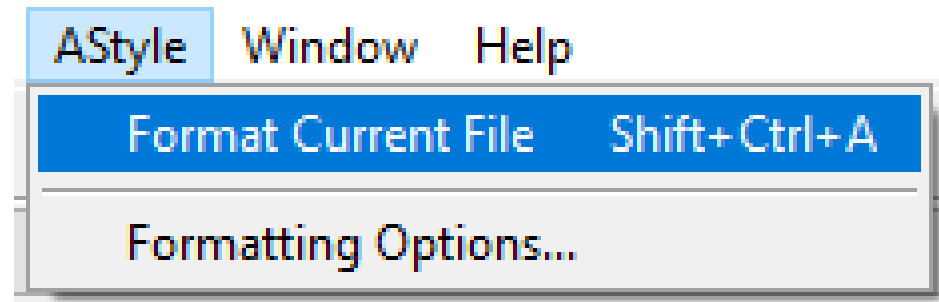
Tipi di File presenti nella cartella del progetto

- “.dev” “.layout” → Progetto Dev C++
- “.c” → Codice sorgente
- “.h” → File header
- “.o” → File oggetto
- “.exe” → Eseguitibile
- “Makefile” → “Ricetta” per la compilazione



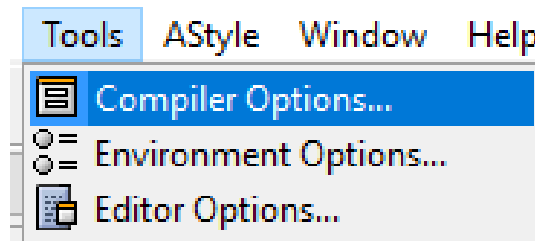
Formattazione automatica del codice

- Menù “Astyle” → Voce “Format Current File”

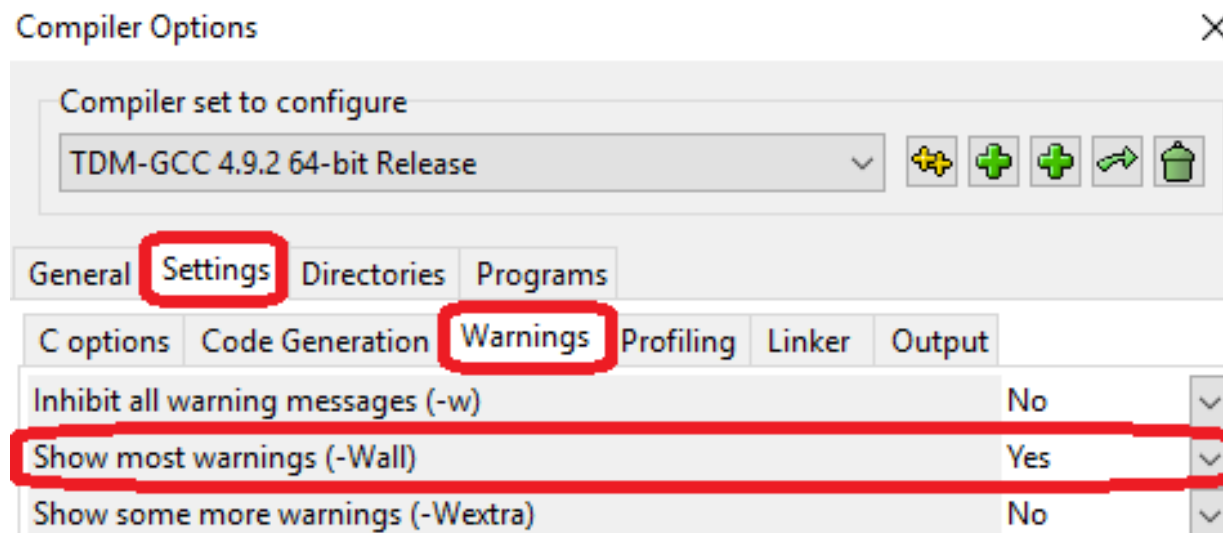


Mostrare tutti i Warning (-Wall)

- Menù “Tools” → Voce “Compiler Options...”



- Tab “Settings” → Tab “Warnings” → Voce “Show most warnings (-Wall)” = Yes



Xcode

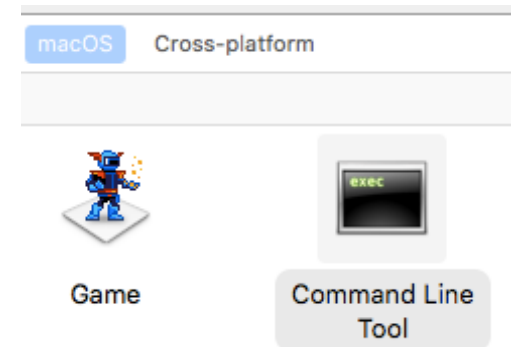


- Solo per Mac OS X
- Scaricabile gratuitamente dall'App Store (lungo download di circa 4.5 GB!)
- Create a new Xcode project



Create a new Xcode project
Create an app for iPhone, iPad, TV

- MacOS → Command Line Tool



- Language: C

Language:

Xcode



- Comandi importanti nel menu “Product”:

- ⌘B → Compila (Build)
- ⌘R → Esegui (Run)

Product	Debug	Source
Run		⌘R
Test		⌘U
Profile		⌘I
Analyze		⇧⌘B
Archive		
Build For		▶
Perform Action		▶
Build		⌘B
Clean		⇧⌘K
Stop		⌘.

- In Mac OS X il compilatore può anche essere lanciato (comando gcc) da Terminale (Applicazioni → Utility → Terminale)

Appendice 1

Compilazione/Debugging in Linux tramite Terminale

Il terminale



Alcuni comandi utili:

- **ls** mostra i file nella directory corrente
- **ls *.c** mostra tutti i file con estensione .c
- **cd directory** cambia la cartella corrente
- **pwd** mostra il path della cartella corrente
- **rm nomefile** elimina nomefile
- **man printf** mostra il manuale di printf
- **(freccia in su)** riprende il comando precedente
- **cp filesorg filecopia** copia un file
- **mv filedaspostare destinazione** sposta un file

Compilazione ed Esecuzione dal Terminale

CON DUE COMANDI:

- **gcc -c file.c**
 - Crea il file oggetto file.o
- **gcc -o file_eseguibile file.o**
 - Crea il file eseguibile file_eseguibile

CON UN UNICO COMANDO:

- **gcc -o file_eseguibile file.c**
 - Crea il file eseguibile file_eseguibile

ESECUZIONE:

- **./file_eseguibile parametro1 parametro2 ...**
 - (il punto “.” sta per “la directory corrente”)

Primo programma – Hello World!



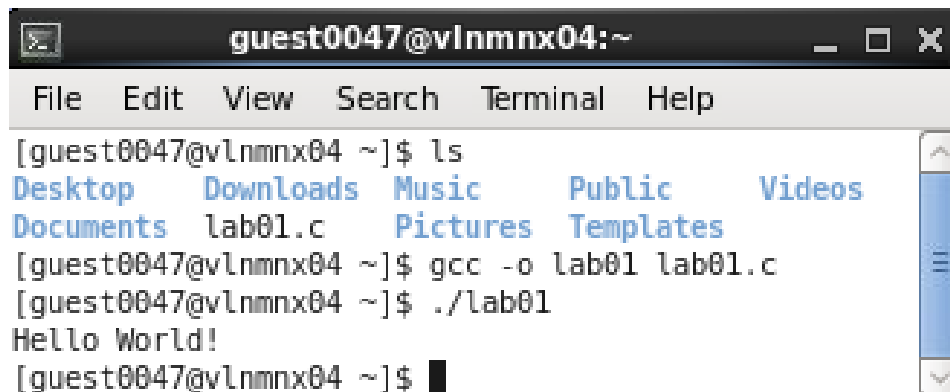
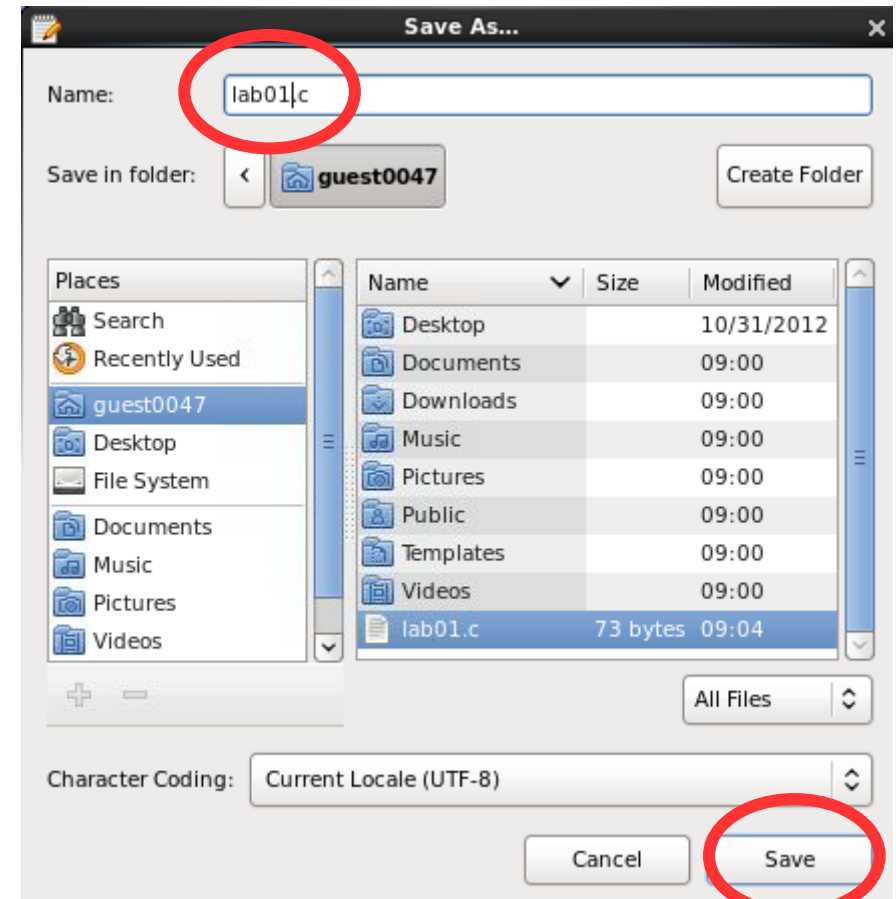
lab01.c (~) - gedit

File Edit View Search Tools Documents Help

Open Save Undo

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!\n");
5     return 0;
6 }
```

C Tab Width: 8 Ln 5, Col 18 INS



guest0047@vlnmnx04:~

File Edit View Search Terminal Help

```
[guest0047@vlnmnx04 ~]$ ls
Desktop  Downloads  Music      Public     Videos
Documents lab01.c    Pictures   Templates
[guest0047@vlnmnx04 ~]$ gcc -o lab01 lab01.c
[guest0047@vlnmnx04 ~]$ ./lab01
Hello World!
[guest0047@vlnmnx04 ~]$
```

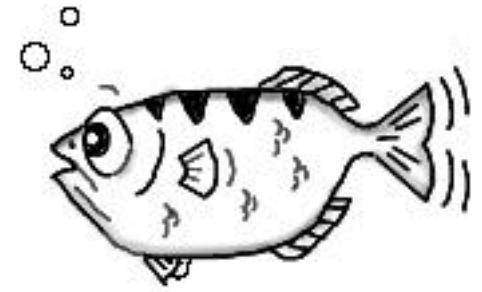
Curiosità: Codice Assembly

gcc -S nomefile.c

- Produce il file “nomefile.s” contenente il codice assembly derivante dalla compilazione

```
somma :  
.LFB0 :  
    .cfi_startproc  
    pushl    %ebp  
    .cfi_def_cfa_offset 8  
    .cfi_offset 5, -8  
    movl     %esp, %ebp  
    .cfi_def_cfa_register 5  
    movl     12(%ebp), %eax  
    movl     8(%ebp), %edx  
    addl     %edx, %eax
```


GDB - GNU Debugger



- <http://www.gnu.org/software/gdb/>
- Occorre compilare i programmi in una maniera particolare per aggiungere le informazioni utili al debugger... occorre usare l'opzione **-g**
 - gcc **-g** -o file_eseguibile file.c
- Per far partire gdb segnalandogli che dovrà controllare l'esecuzione di file_eseguibile occorre scrivere:
 - gdb file_eseguibile

Comandi GDB

- Quando GDB è partito compare il prompt (**gdb**)
- A questo punto è possibile usare i seguenti comandi:
 - **run lista_parametri** esegue il programma
 - **step** esegue una singola istruzione
 - **break posizione** esegue il programma fino ad incontrare posizione (ad esempio break main per fermarsi all'inizio o break 8 per fermarsi alla riga 8)
 - **continue** riprende l'esecuzione
 - **print espressione** stampa il valore di espressione (ad esempio la variabile i o anche i+i)
 - **quit** termina gdb

Esempio di debugging

- `gcc -g -o lab01 lab01.c` ← compila il progetto predisponendolo al debugging
- `gdb lab01` ← avvia il debugging su lab01
- `break main` ← aggiunge un breakpoint all'inizio di main
- `run 5 4` ← avvia il programma passando i parametri "5" e "4"; il programma si fermerà al primo breakpoint "main".
- `step` ← passa all'istruzione successiva
- `step` ← idem. Supponiamo che in questo punto la variabile "a" assuma un valore, ad es. `a = 5`;
- `print a` ← stampa il valore della variabile a (5)
- `continue` ← riprende l'esecuzione fino alla fine
- `quit` ← uscita dal debugger

Appendice 2

Passaggio di Parametri a un Programma

Passaggio di Parametri

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

- `argc` è un numero intero corrispondente al numero di parametri ricevuti.
- `argv` è il vettore dei parametri ricevuti.

I parametri sono tutte stringhe di testo → per ottenere dei tipi numerici vanno applicate delle conversioni (ad es. tramite la funzione `atoi`)

- Attenzione: il primo parametro c'è sempre e corrisponde al nome (con eventuale path) del programma eseguibile

Programma di Esempio

```
#include <stdio.h>

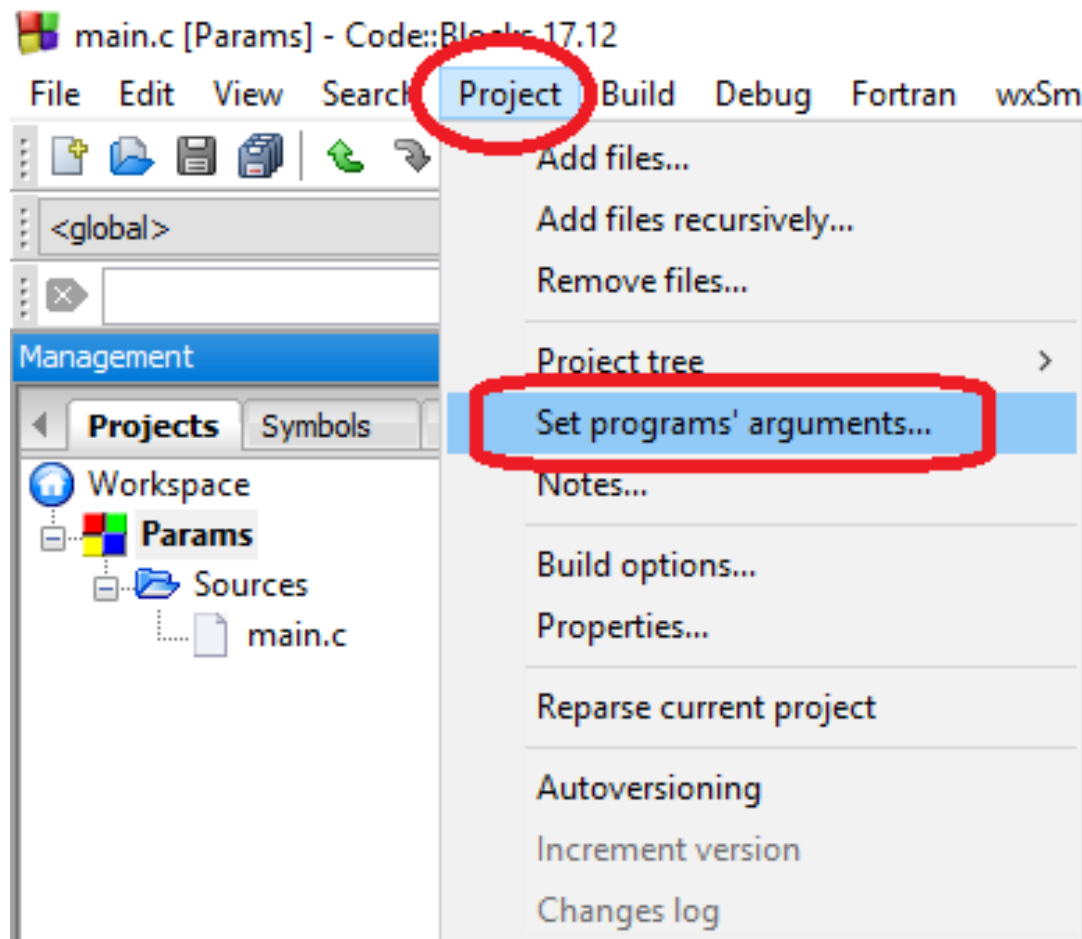
//stampa i parametri ricevuti

int main(int argc, char* argv[]) {
    int i;
    printf("num parametri=%d\n", argc);
    for (i = 0; i < argc; i++)
        printf("%d = %s\n", i, argv[i]);
    return 0;
}
```

Parametri da Code::Blocks



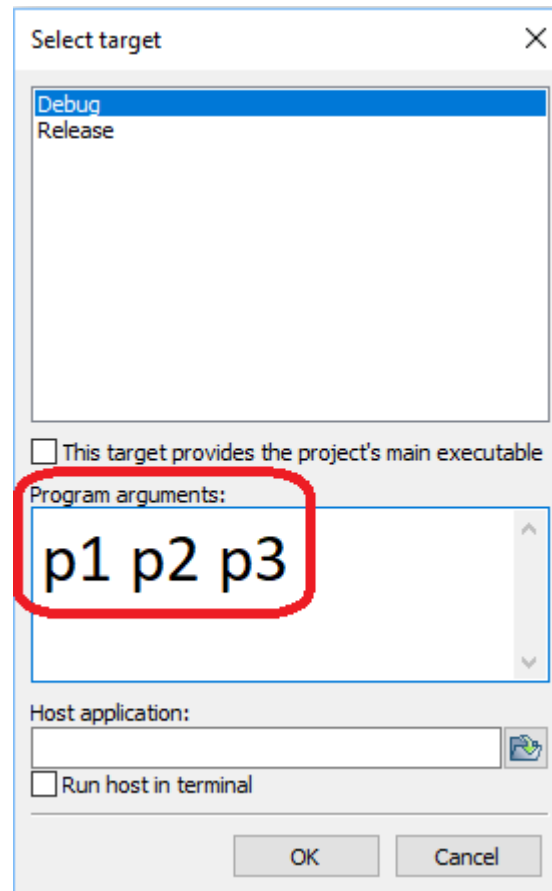
- Project → Set programs' arguments...



Parametri da Code::Blocks



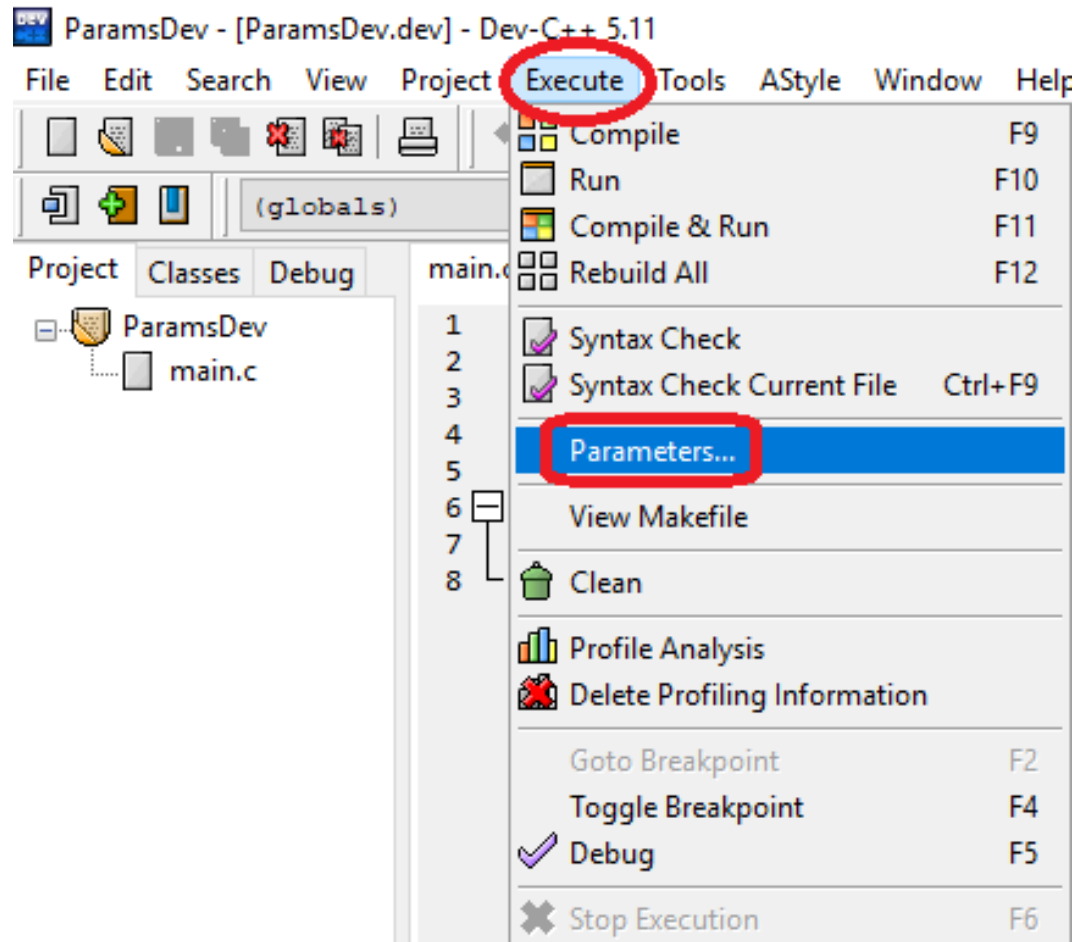
- Inserire i parametri, separati da uno spazio, nella casella “Program arguments:”



Parametri da Dev C++



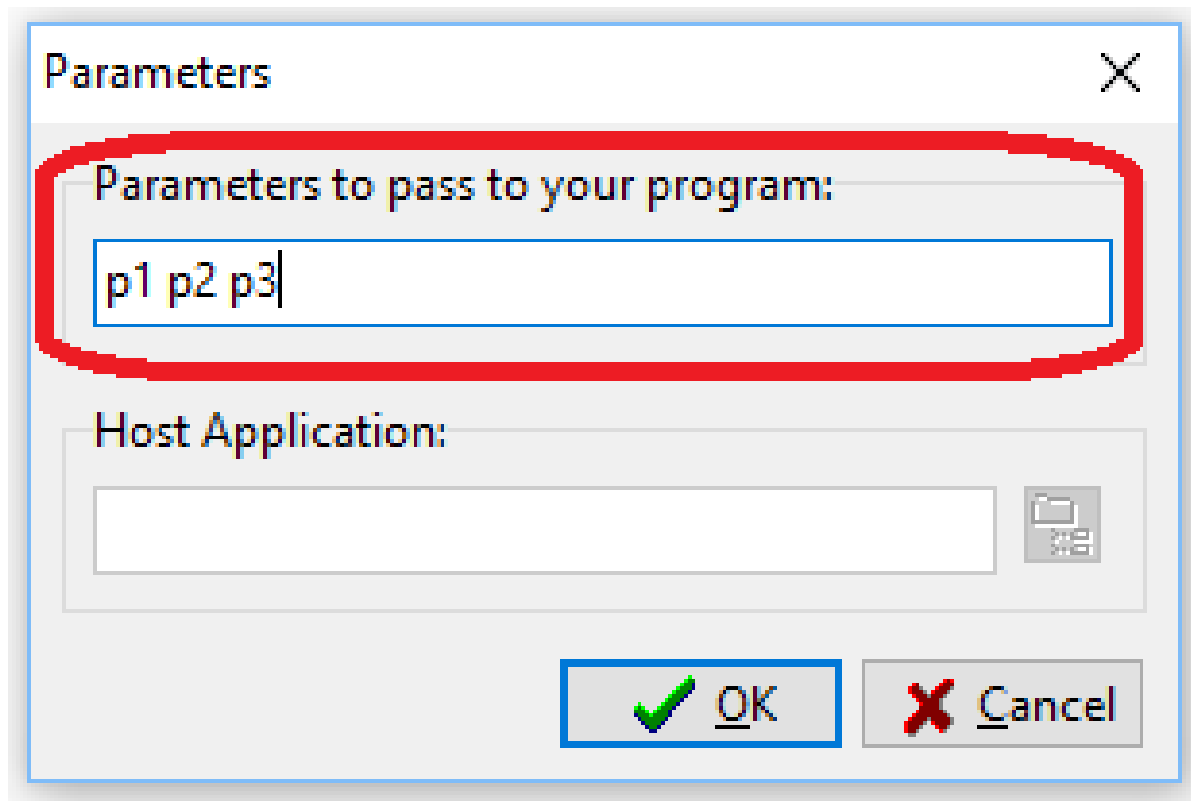
- Execute → Parameters...



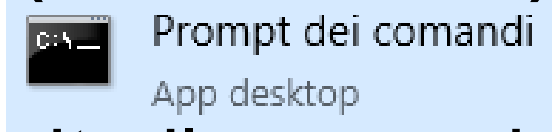
Parametri da Dev C++



- Inserire i parametri, separati da uno spazio, nella casella "Parameters to pass to your program:"



Parametri da Terminale (Windows)



- Aprire il terminale, spostarsi tramite il comando `cd` nella cartella dove è stato creato il programma eseguibile
- In Windows, scrivere il nome dell'eseguibile `.exe` seguito dai parametri separati da spazio e premere invio

```
C:\Users>cd C:\Laboratorio\params\bin\Debug  
  
C:\Laboratorio\params\bin\Debug>params.exe p1 p2 p3  
Questo programma ha ricevuto un numero di parametri pari a 4.  
0 = params.exe  
1 = p1  
2 = p2  
3 = p3
```

Parametri da Terminale

(GNU/Linux / Unix / Mac Os X)

- Aprire il terminale, spostarsi tramite il comando `cd` nella cartella dove è stato creato il programma eseguibile
- Scrivere il nome dell'eseguibile, preceduto da `./` e seguito dai parametri separati da spazio e premere invio

```
~/Laboratorio$ cd /home/paolo/Laboratorio
~/Laboratorio$ gcc -o params params.c
~/Laboratorio$ ./params p1 p2 p3
Questo programma ha ricevuto un numero di parametri pari a 4
0 = ./params
1 = p1
2 = p2
3 = p3
~/Laboratorio$
```

Esempio con conversione a int

```
#include <stdio.h>

#include <stdlib.h>

//raddoppia il numero passato

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

    int n;

    if (argc < 2) {

        printf("Passare almeno un numero.\n");

        return -1;

    }

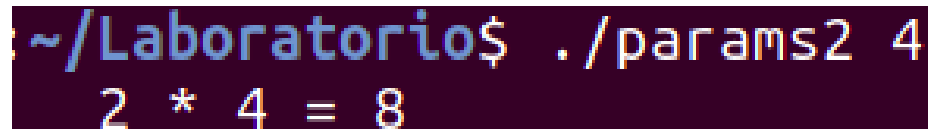
    //per float usare atof invece di atoi

    n = atoi(argv[1]);

    printf("2 * %d = %d\n", n, n * 2);

    return 0;

}
```

A terminal window with a dark purple background. The prompt is ~/Laboratorio\$. The command ./params2 4 has been executed, and the output is 2 * 4 = 8.

```
~/Laboratorio$ ./params2 4
2 * 4 = 8
```

Appendice 3

Altri Strumenti

Installazione GNU/Linux

- Per chi desiderasse sperimentare e provare a utilizzare gcc dal terminale, è possibile scaricare qualsiasi distribuzione di GNU/Linux e installarla sul proprio PC, o usarla in modalità “Live” (cioè senza installare nulla) o installarla su una macchina “virtuale”
 - <https://www.ubuntu.com>
 - <https://getfedora.org/>
 - ...

Istruzioni macchina virtuale

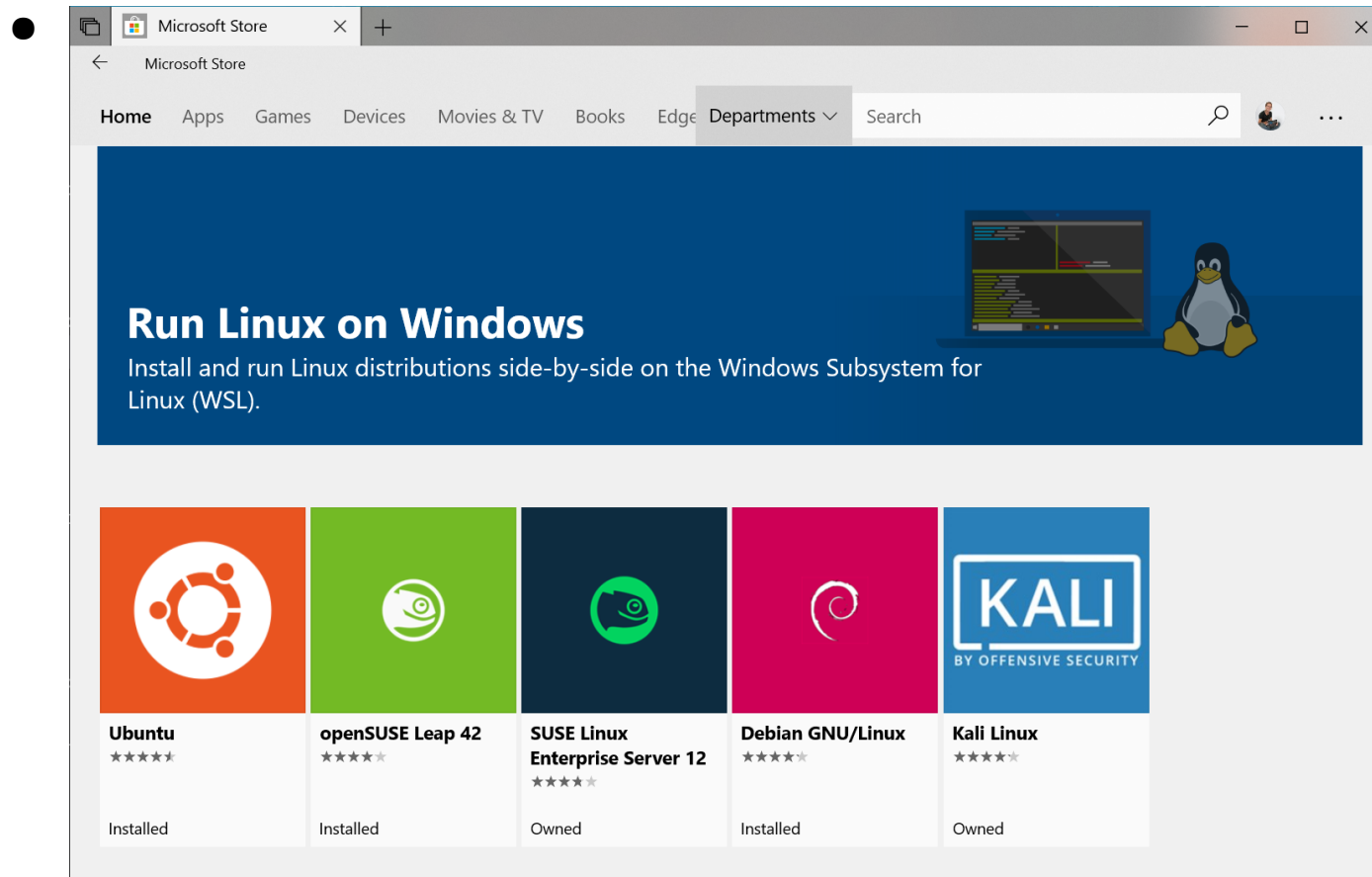
- Scaricare la distribuzione di GNU/Linux - ad esempio “Ubuntu”
 - sito <http://www.ubuntu.com>
- Scaricare e installare il software di virtualizzazione VirtualBox e creare una macchina virtuale
 - sito <https://www.virtualbox.org>

Cygwin

- Una sorta di “Distribuzione GNU/Linux” che funziona sotto Windows
- sito: <http://www.cygwin.com/>

Linux on Windows 10

- <https://docs.microsoft.com/en-us/windows/wsl/install-win10>

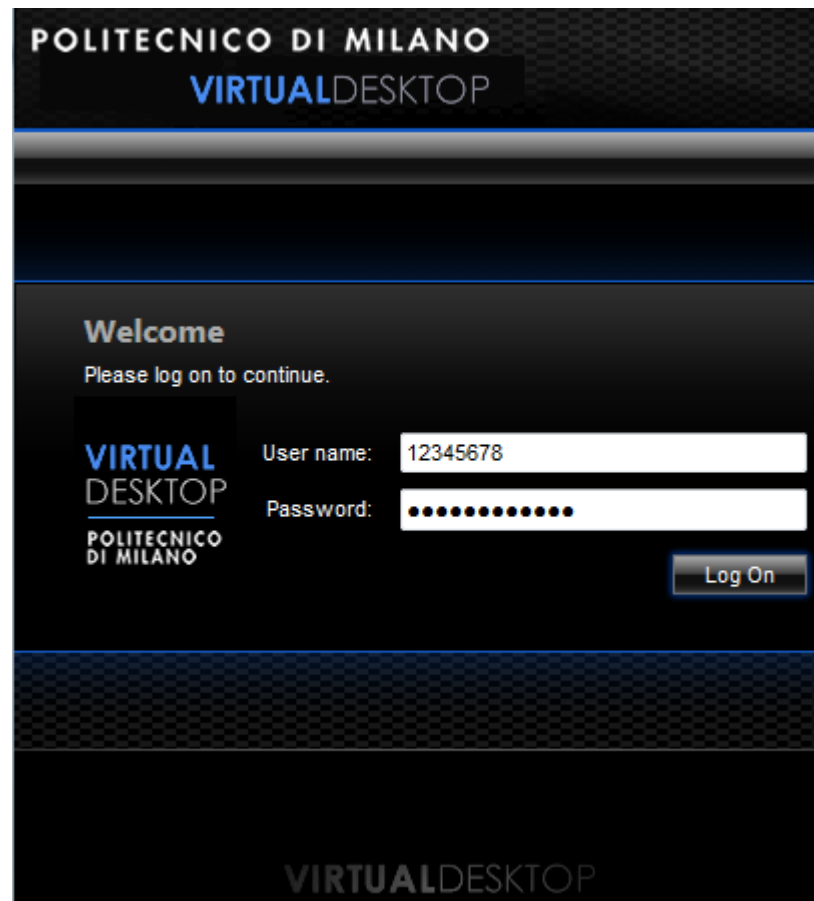


Polimi - Virtual Desktop

- Le applicazioni sono eseguite su richiesta da un server remoto: non occorre che siano installate sul PC dell'aula (o sul proprio PC).
- Occorre che sul PC in uso sia installato il software “**Citrix Receiver**”. Istruzioni su:
 - <http://www.client.polimi.it/virtual-desktop>
- Per richiedere l'esecuzione di un'applicazione è sufficiente collegarsi al seguente sito e fornire il proprio codice persona e password:
 - <https://virtualdesktop.polimi.it>
- NB: funziona anche da casa ma solo la sera e nei fine settimana

Polimi - Virtual Desktop

- <https://virtualdesktop.polimi.it>



The image shows a login interface for the Politecnico di Milano Virtual Desktop. The header features the text "POLITECNICO DI MILANO" and "VIRTUALDESKTOP" in a stylized font. Below this, a "Welcome" message is displayed, followed by the instruction "Please log on to continue." The login form includes a "VIRTUAL DESKTOP POLITECNICO DI MILANO" logo on the left. To the right of the logo, there are two input fields: "User name:" with the value "12345678" and "Password:" with a masked password represented by dots. A "Log On" button is positioned to the right of the password field. The footer of the page displays "VIRTUALDESKTOP" in a large, bold font.

POLITECNICO DI MILANO
VIRTUALDESKTOP

Welcome
Please log on to continue.

**VIRTUAL
DESKTOP**
POLITECNICO
DI MILANO

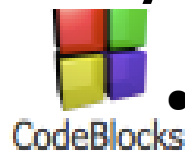
User name: 12345678
Password:

Log On

VIRTUALDESKTOP

Applicazioni Virtual Desktop

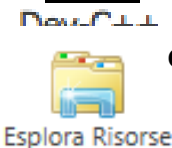
- Applicazioni utili tra quelle disponibili:



- Code::Blocks → IDE per programmare in C



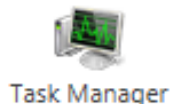
- Dev C++ → IDE per programmare in C



- Esplora Risorse → per trasferire i file sulla/dalla macchina remota




- Task Manager → per terminare processi bloccati sulla macchina remota



NX Client

- NX Client → Per avviare in una finestra una macchina virtuale con il sistema operativo Linux

Avvio della macchina virtuale Linux

- Avviare la macchina “NX Client”  NX Client
- Menù “Applications” → “Accessories” → “gedit Text Editor” per avviare l’editor di testi “gedit”

