Fondamenti di Informatica e Laboratorio T-AB
Ingegneria Elettronica e Telecomunicazioni

Unix e comandi di base

Unix - Shell dei comandi – Es. 1

- Utilizzando il comando “ls”…
- Elencare a video il contenuto della directory padre (rispetto alla directory corrente) in formato “lungo” e mostrando pure i files “nascosti” (files che cominciano con “.”
- Esistono specifiche opzioni per ottenere il comportamento desiderato (per capire quali, potete usare “man”)

Unix - Shell dei comandi – Es. 2

- Il comando “echo <stringa>” scrive semplicemente la stringa sul display
- Utilizzare il comando “echo” e salvarne l’output direttamente in un file di nome “prova.txt” tramite la redirezione dell’output
- Usare il comando “cat” per verificare il contenuto del file

Unix - Shell dei comandi – Es. 3

- Cambiare i diritti al file prova.txt in modo da negare la possibilità di lettura all'owner (proprietario)
- A tal scopo, usare il comando chmod una o più volte, se necessario)
- Usare poi il comando ls con la specifica opzione per verificare la modifica dei diritti
- Provate ad utilizzare "cat" per visualizzare il contenuto del file

Unix - Shell dei comandi – Es. 4

- Stampare a video il contenuto della directory corrente, in ordine alfabetico inverso
- Utilizzando i comandi ls e sort in pipe
- Verificare tramite “man sort” l’ opzione per ordinare in senso inverso

Unix - Shell dei comandi – Es. 5

- Elencare a video tutti i processi attualmente in esecuzione
- Individuare il PID del processo di bash
- Uccidere tale processo tramite il comando “kill -9 <PID>”
- Solo per questa volta! È molto meglio chiudere i programmi utilizzando l’apposito pulsante (o mediante gli appositi comandi)

Fondamenti di Informatica e Laboratorio T-AB
Ingegneria Elettronica e Telecomunicazioni

Lab 01
Introduzione a Codelite

Costruzione di un'Applicazione

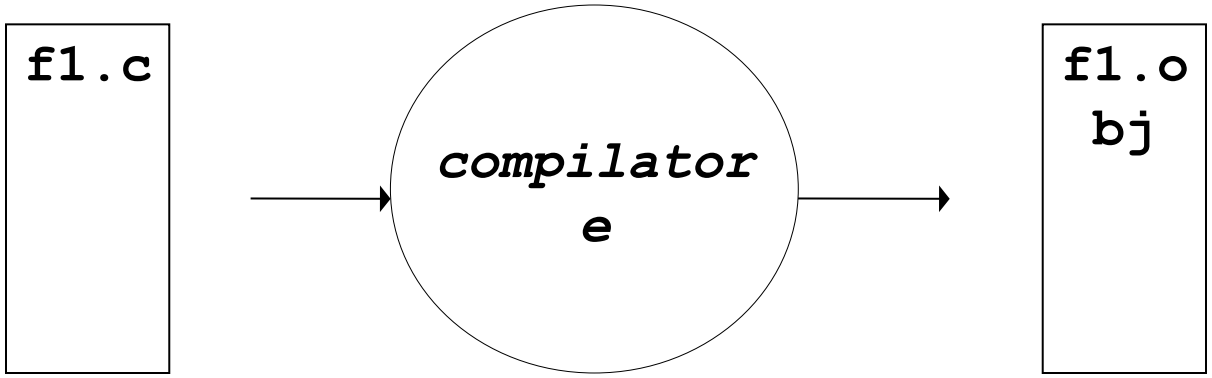
Per costruire un'applicazione occorre:

- **compilare il file (o *i* file se più d'uno) che contengono il testo del programma (file *sorgente*)**
  Il risultato sono uno o più file *oggetto.*
- **collegare i file oggetto l'uno con l'altro *e con le librerie di sistema.***

Compilazione di un'Applicazione

1) Compilare il file (o *i* file se più d'uno) che contengono il testo del programma

- **File *sorgente*:** **estensione `.c`**
- **File *oggetto*:** **estensione `.o` o `.obj`**

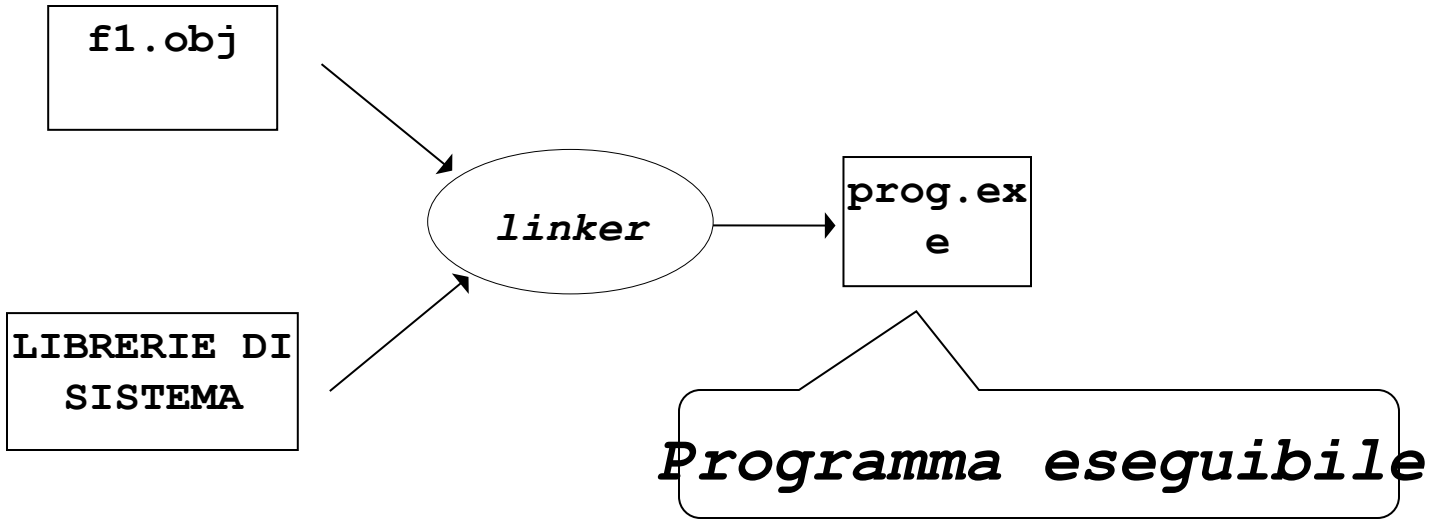


f1.obj: Una versione tradotta che però non è autonoma (e, quindi, non è direttamente eseguibile).

Collegamento (Linking) di un'Applicazione

2) Collegare il file (o *i* file) oggetto fra loro *e con le librerie di sistema*

- **File *oggetto*:** **estensione `.o` o `.obj`**
- **File *eseguibile*:** **estensione `.exe` o nessuna**



Collegamento (Linking) di un'Applicazione

LIBRERIE DI SISTEMA:

insieme di componenti software che consentono di interfacciarsi col sistema operativo, usare le risorse da esso gestite, e realizzare alcune "istruzioni complesse" del linguaggio

Ambienti Integrati

Oggi, gli *ambienti di lavoro integrati* automatizzano la procedura:

- **compilano i file sorgente (*se* e *quando necessario*)**
- **invocano il linker per costruire l'eseguibile**

ma per farlo devono sapere:

- ***quali file sorgente* costituiscono l'applicazione**
- ***il nome dell'eseguibile* da produrre.**

Progetti

È da queste esigenze che nasce il concetto di *PROGETTO*

- **un *contenitore concettuale (e fisico)***
- **che *elenca i file sorgente* in cui l'applicazione è strutturata**
- **ed eventualmente altre informazioni utili.**

Oggi, *tutti* gli ambienti di sviluppo integrati, *per qualunque linguaggio*, forniscono questo concetto e lo supportano con idonei strumenti.

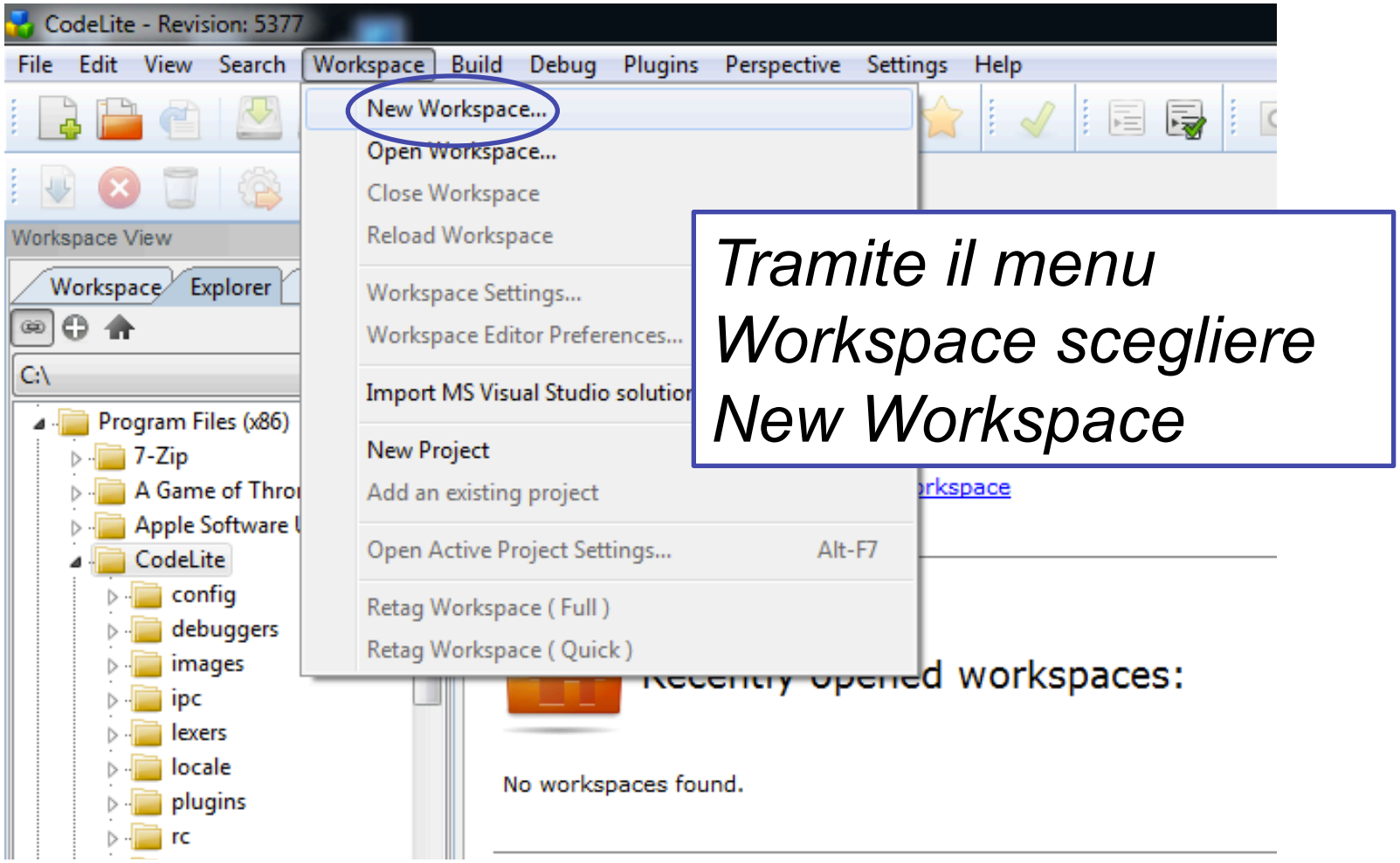
Progetti in Codelite

Download

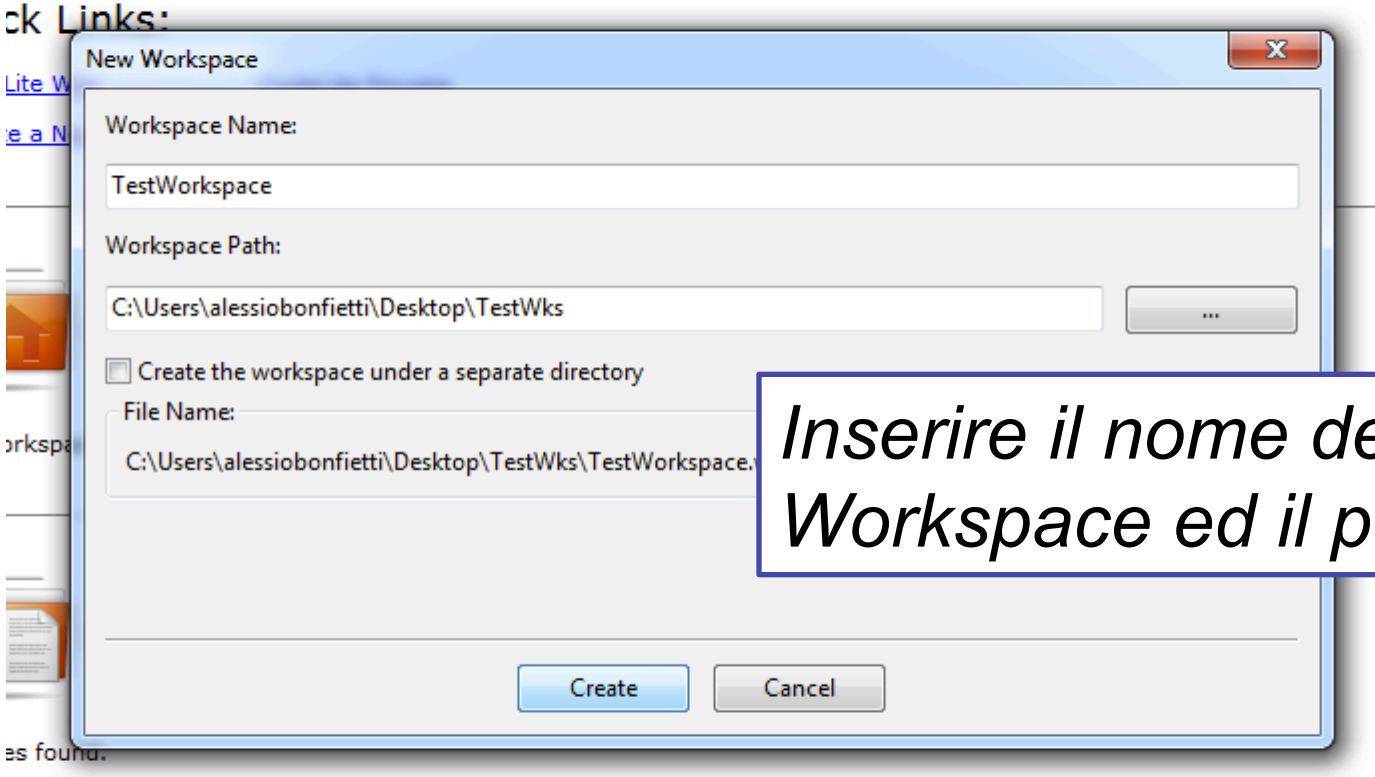
an installer which includes

codelite IDE + MinGW suite (GNU toolchain + WinAPI)

Progetti in Codelite



Progetti in Codelite

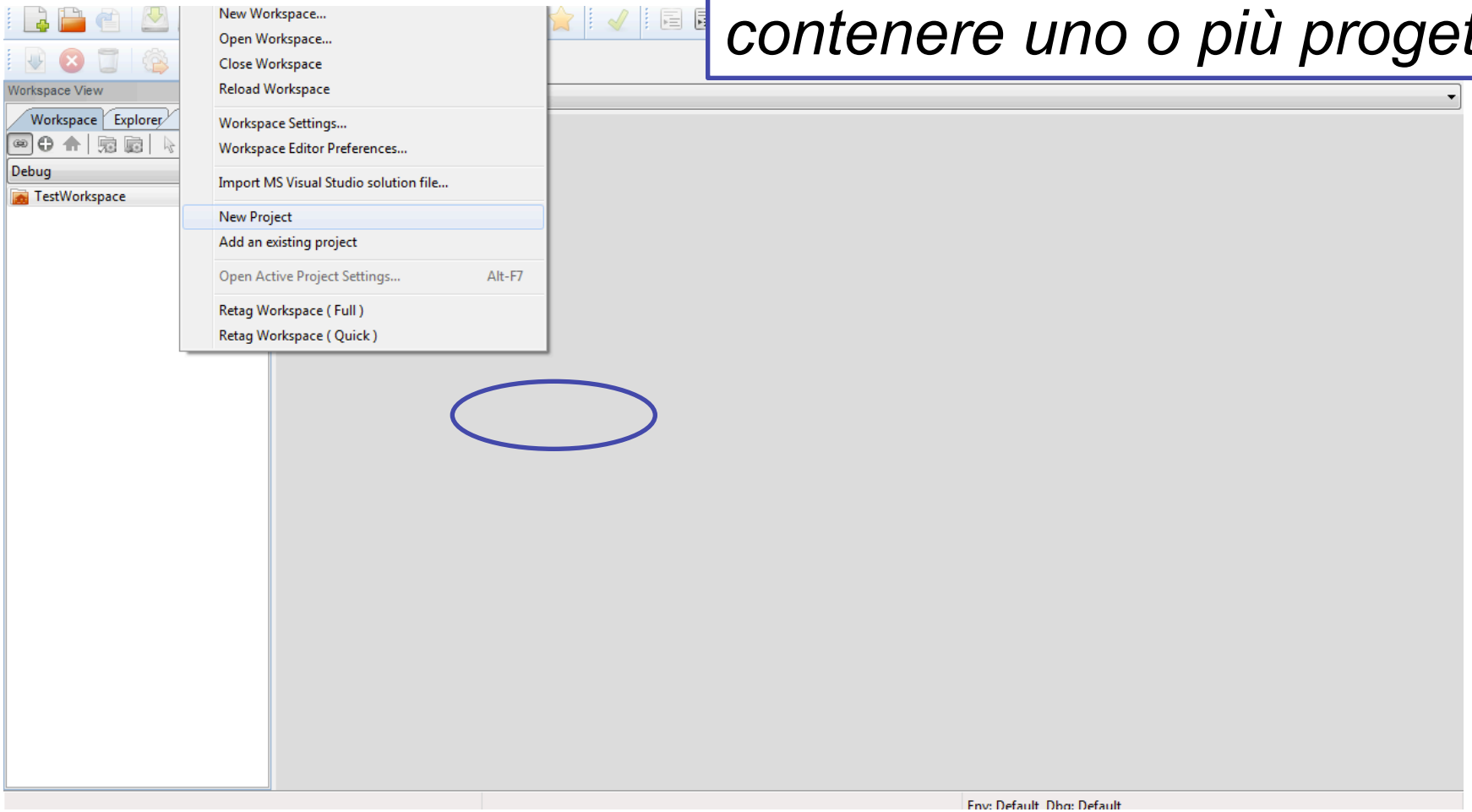


Inserire il nome del Workspace ed il percorso

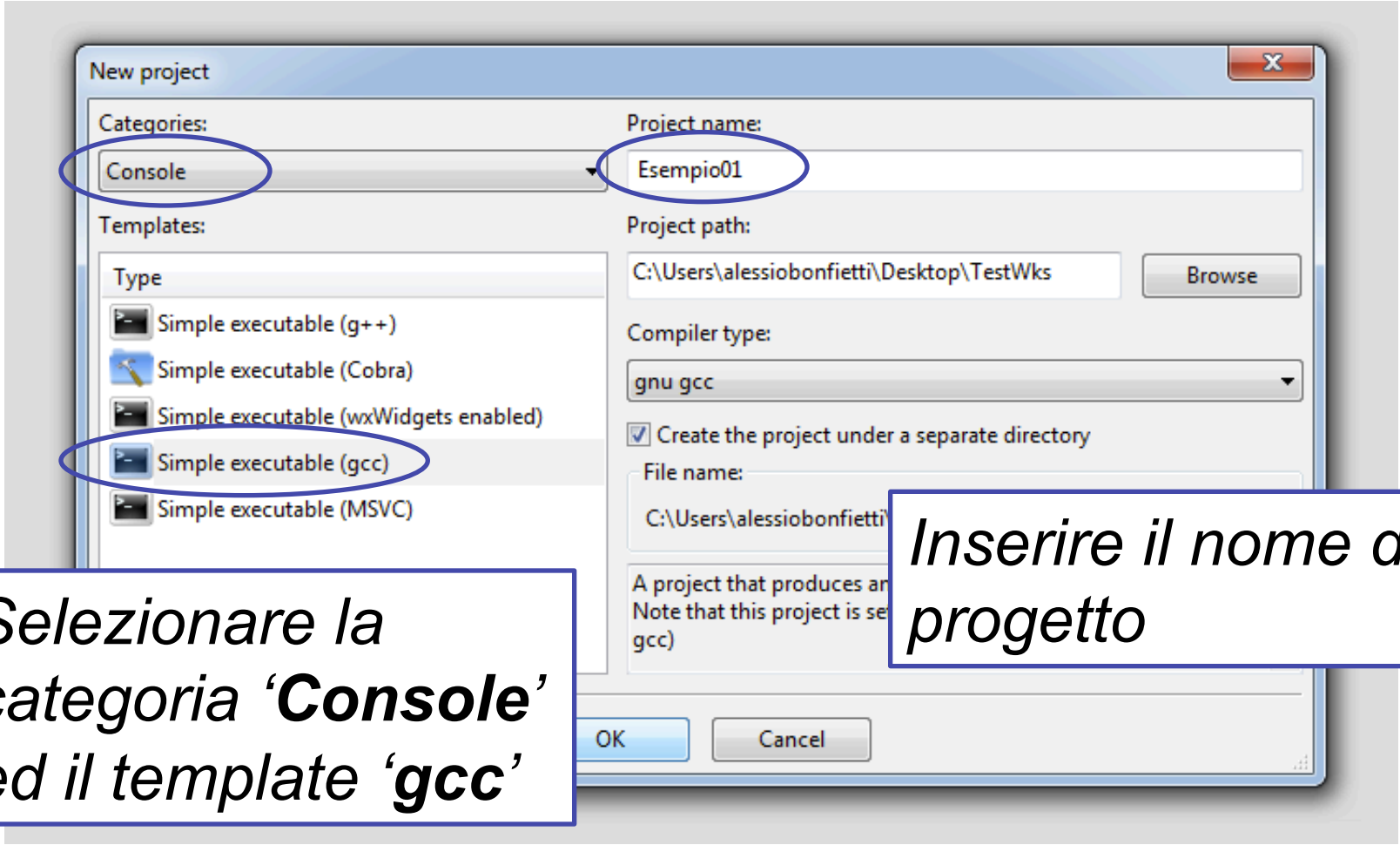
Si consiglia di lavorare sempre in c:\temp

Progetti in Codelite

Ogni workspace può contenere uno o più progetti



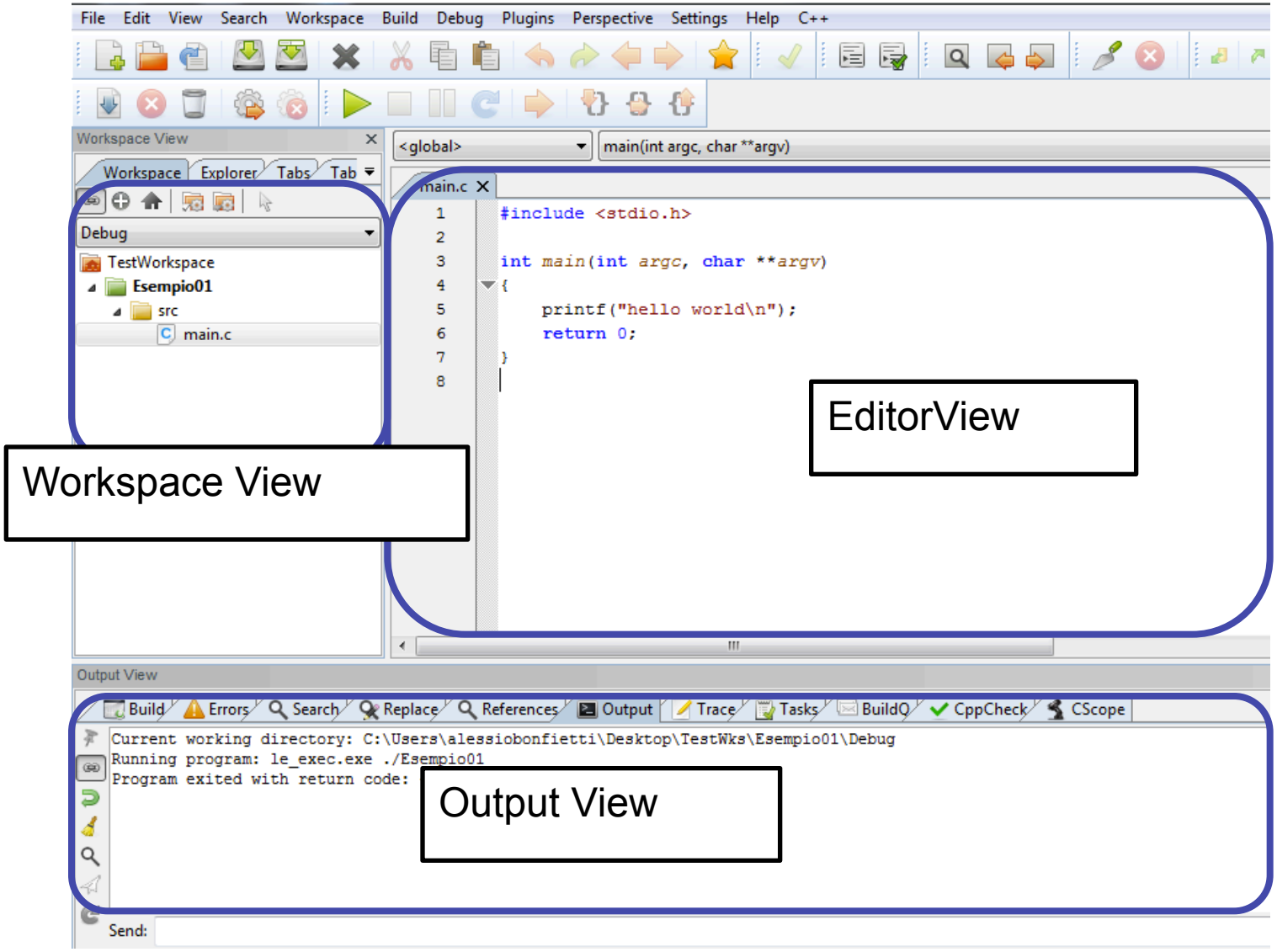
Progetti in Codelite



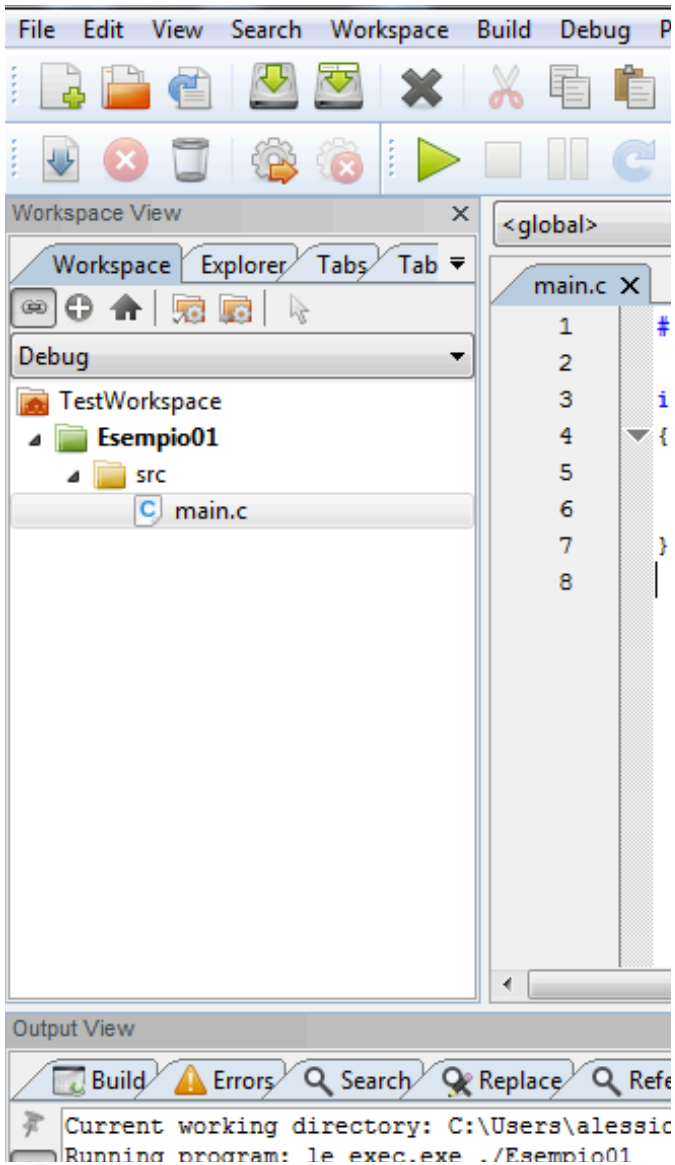
*Selezionare la categoria '**Console**' ed il template '**gcc**'*

Inserire il nome del progetto

Progetti in Codelite



Progetti in Codelite



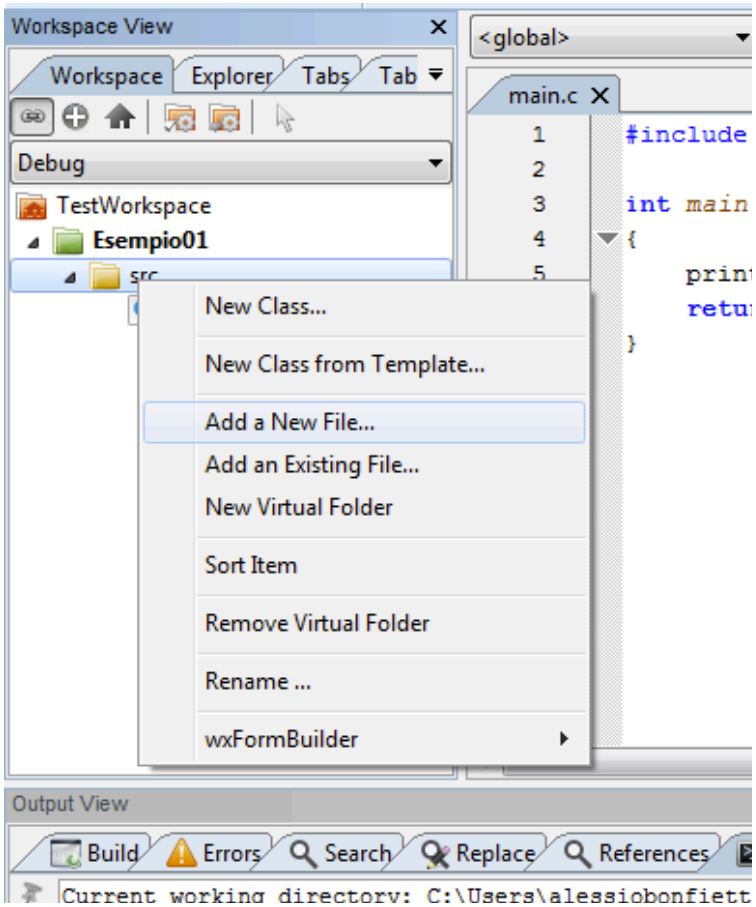
Workspace View

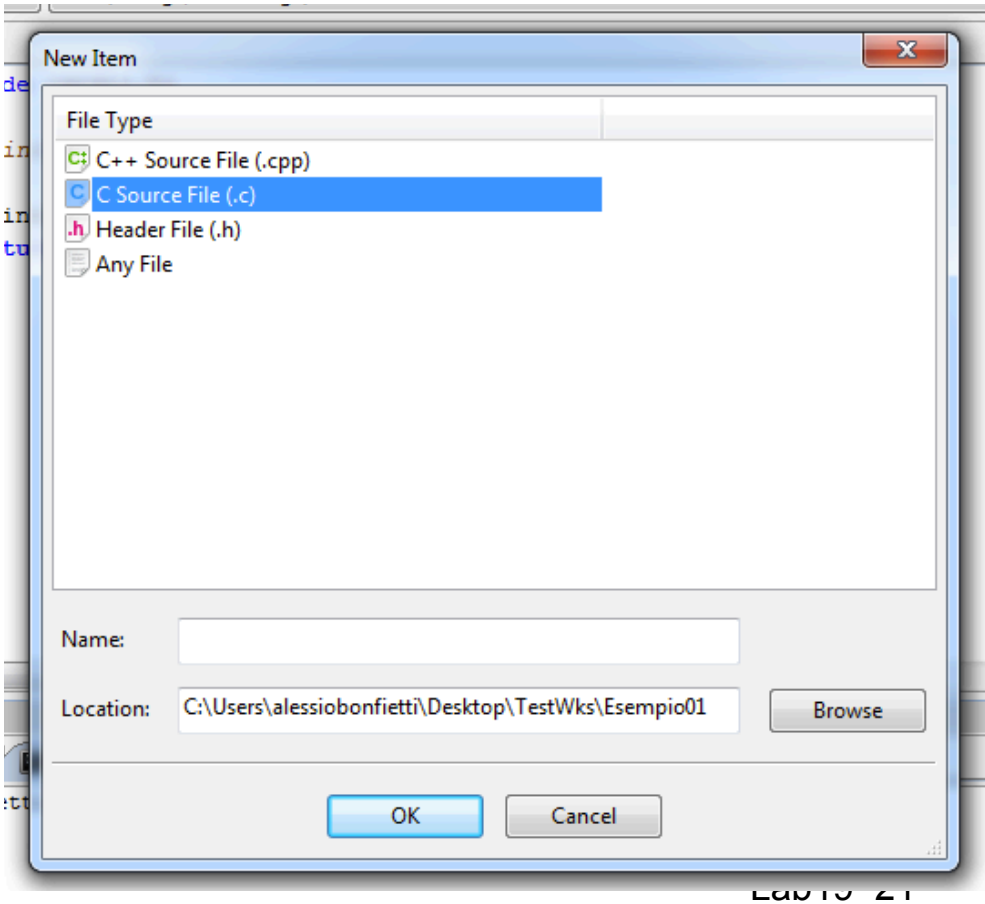
Alla creazione di un progetto, l'IDE **Codelite** crea automaticamente il file principale contenente la funzione main del programma.

Da questa interfaccia è dedicata alla gestione dei file sorgente

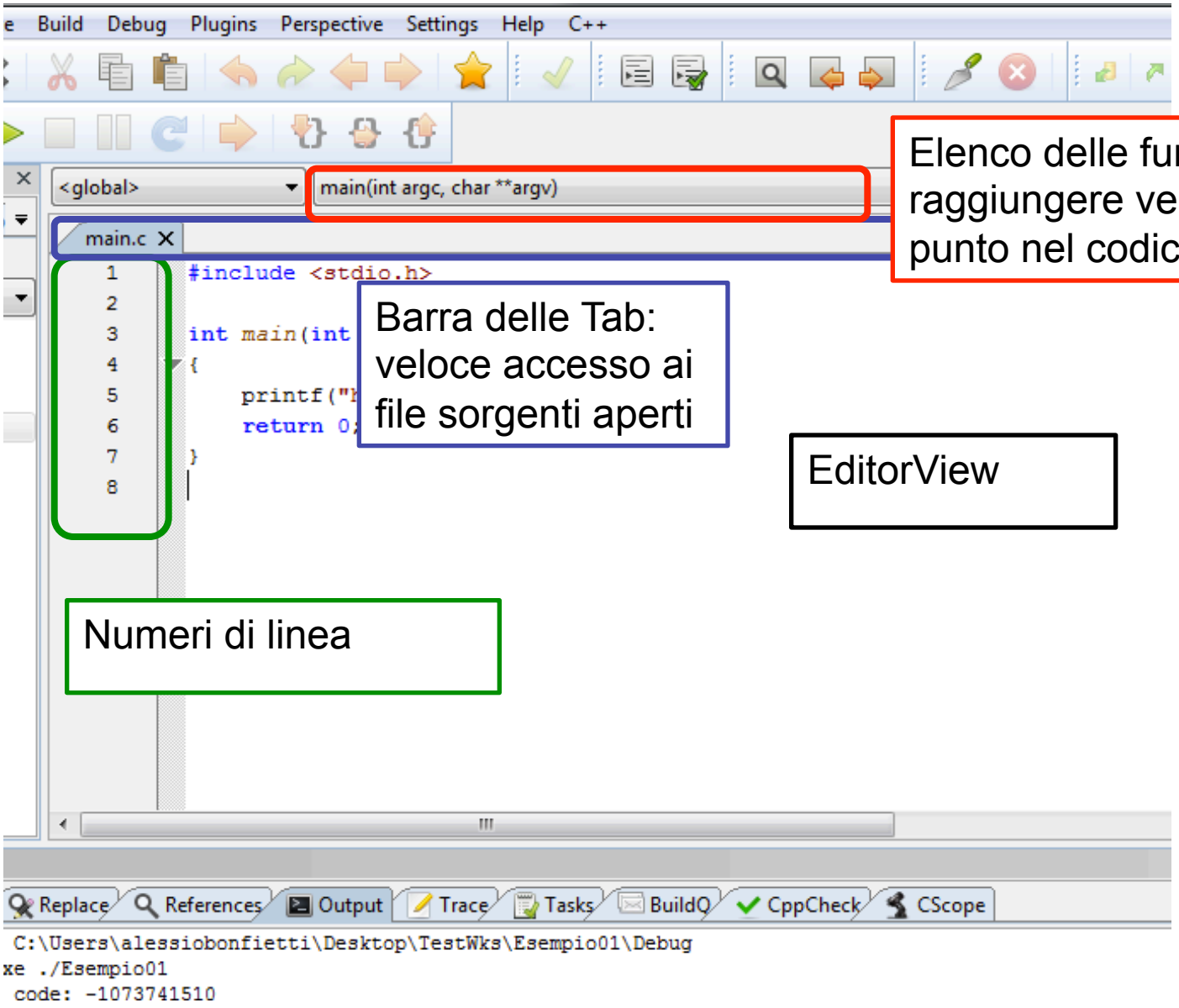
Progetti in Codelite

Click destro sulla directoy ‘src’ per aggiungere un file sorgente



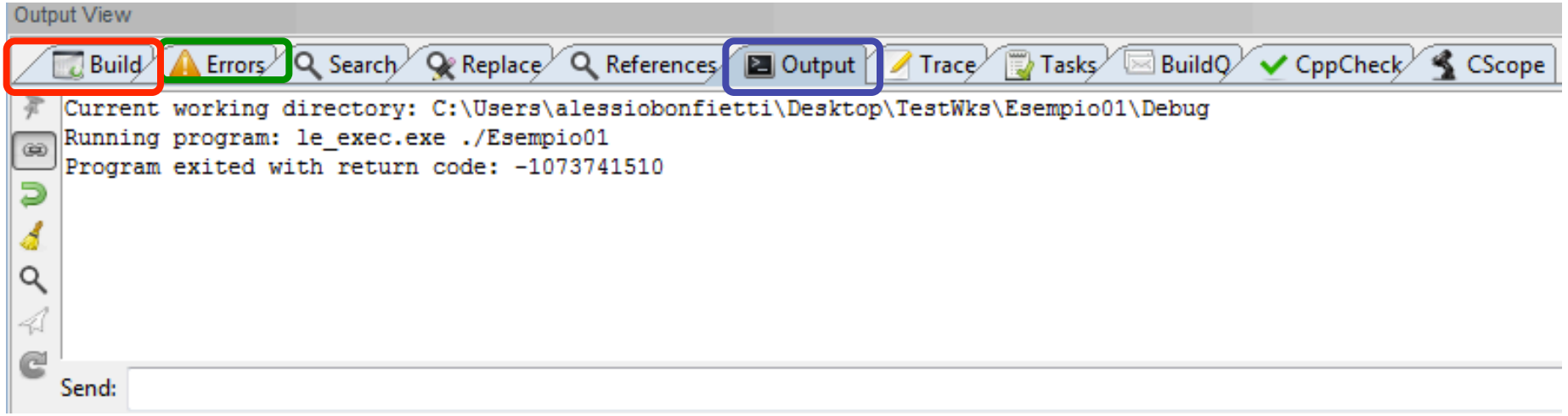


Progetti in Codelite



Progetti in Codelite

Output View

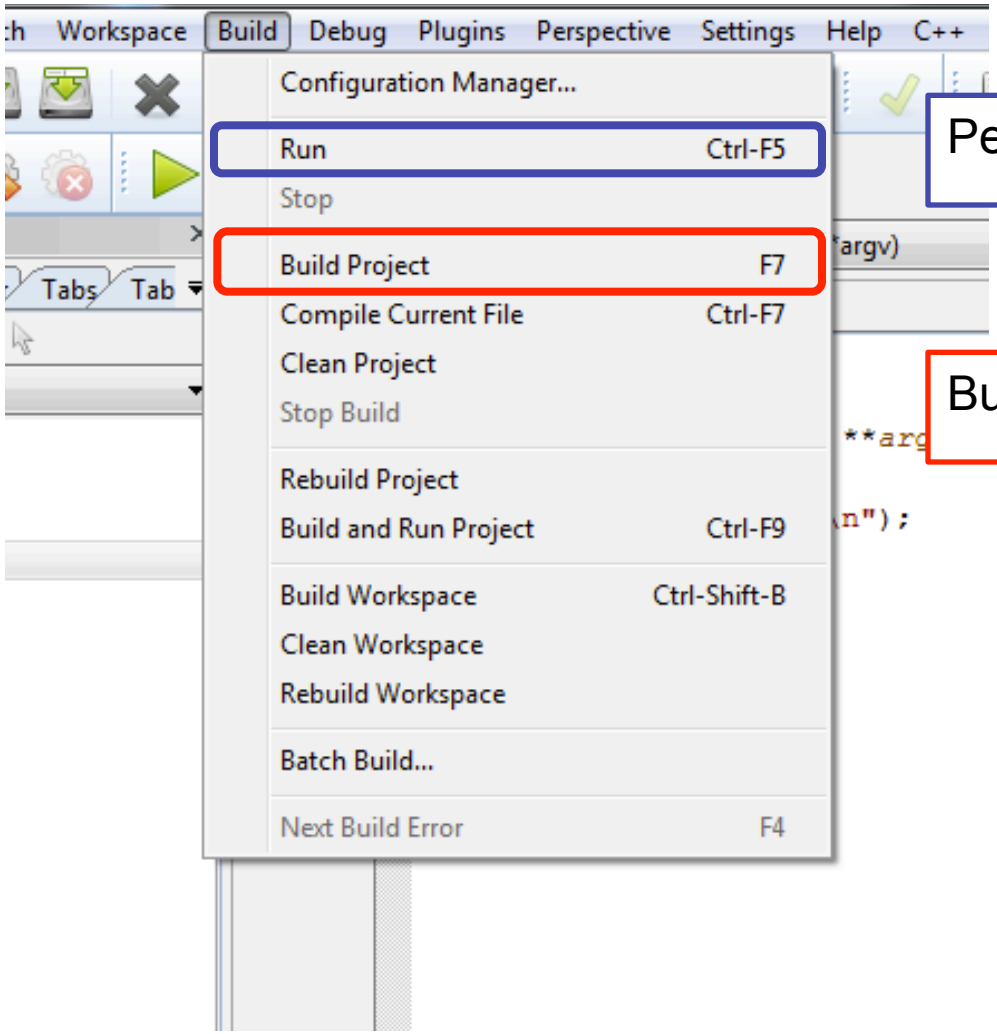


Controllo degli errori con visualizzazione dei comandi di compiling e di linking

Visualizzazione dell'output di compilazione su console

Visualizzazione intuitiva degli errori e dei warning di compilazione

Progetti in Codelite



Per Eseguire il programma

Build = Compile + Link

Esempio di programma

```
#include <stdio.h>

int main(int argc, char **argv) {
    int a, b;
    printf(“Inizio\n”);
    b = 5;
    a = b + 2;
    printf(“a = %d\n”, a);
    return 0;
}
```

Il Debugger

Una volta scritto, compilato e collegato il programma (ossia, costruito l'eseguibile)

occorre uno strumento che consenta di

- **eseguire il programma *passo per passo***
- ***vedendo le variabili* e la loro evoluzione**
- **e *seguendo le funzioni* via via chiamate.**

Debugger

Debugger

Sia Codelite sia altri ambienti di sviluppo incorporano un *debugger* con cui eseguire il programma,

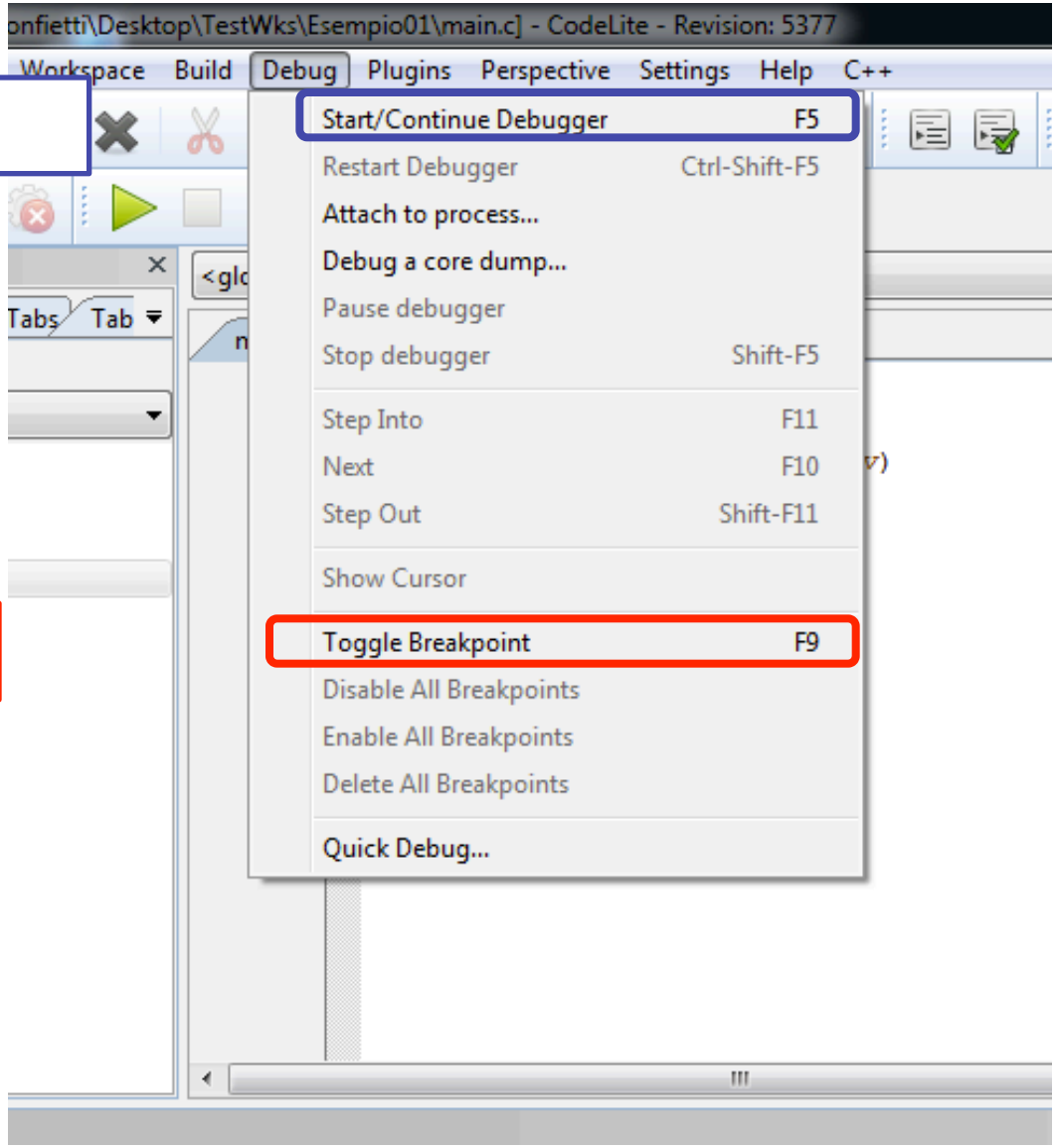
- ***riga per riga***
  - **entrando anche dentro alle funzioni chiamate**
  - **oppure considerando le chiamate di funzione come una singola operazione**

- **oppure *inserendo breakpoints***

Progetti in Codelite

Eseguire in modalità debug

Inserire un Breakpoint

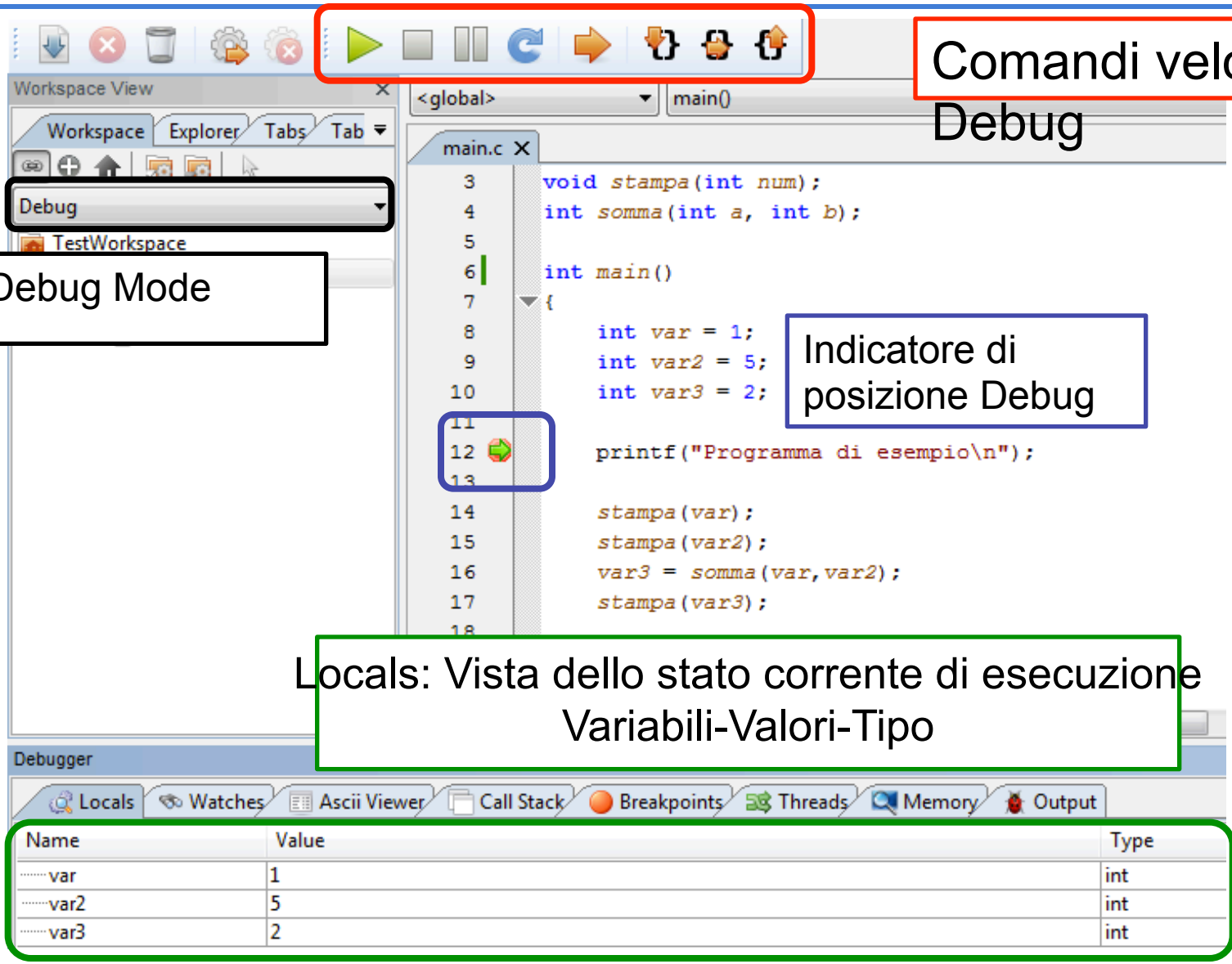


Fase di Debugging

- **Prima di iniziare la sessione di debugging e' possibile inserire i cosiddetti *breakpoints***
  - ***punti di interruzione nell'esecuzione del programma in cui il debugger fornisce una "fotografia" dello stato delle variabili***

- **Per inserire un breakpoint posizionare il cursore nel punto in cui si vuole fermare il debug e (alternative):**
  - ***Utilizzare il comando da Menù***
  - ***Premere F9***
  - ***Singolo click a fianco del numero di riga***

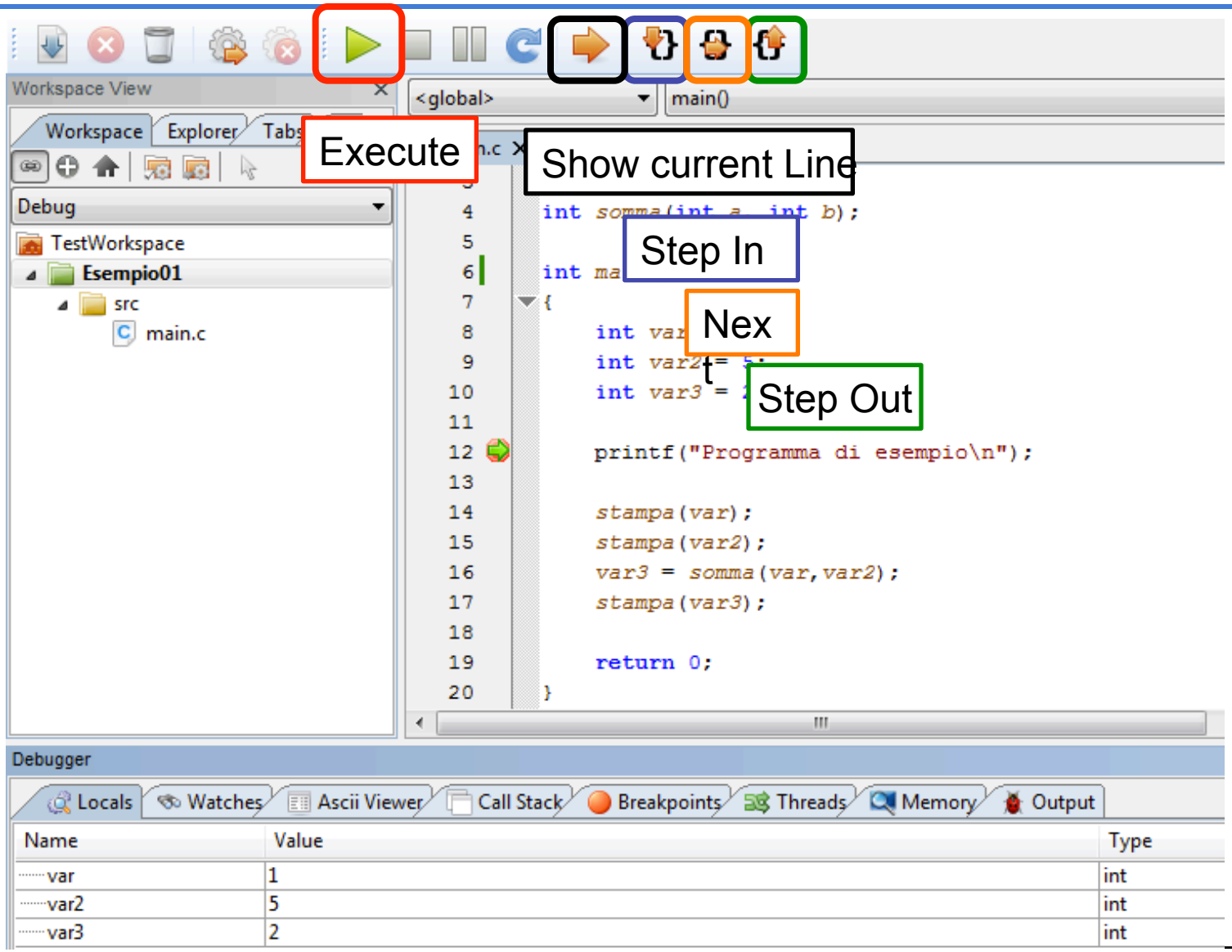
Debugger


Comandi veloci Debug
Debug Mode
Indicatore di posizione Debug
Locals: Vista dello stato corrente di esecuzione
Variabili-Valori-Tipo
Workspace View
Workspace
Explorer
Tabs
Tab
Debug
TestWorkspace
<global>
main()
main.c
3 void stampa(int num);
4 int somma(int a, int b);
5
6 int main()
7 {
8 int var = 1;
9 int var2 = 5;
10 int var3 = 2;
12 printf("Programma di esempio\n");
14 stampa(var);
15 stampa(var2);
16 var3 = somma(var,var2);
17 stampa(var3);
Debugger
Locals
Watches
Ascii Viewer
Call Stack
Breakpoints
Threads
Memory
Output
Name
Value
Type
var
1
int
var2
5
int
var3
2
int

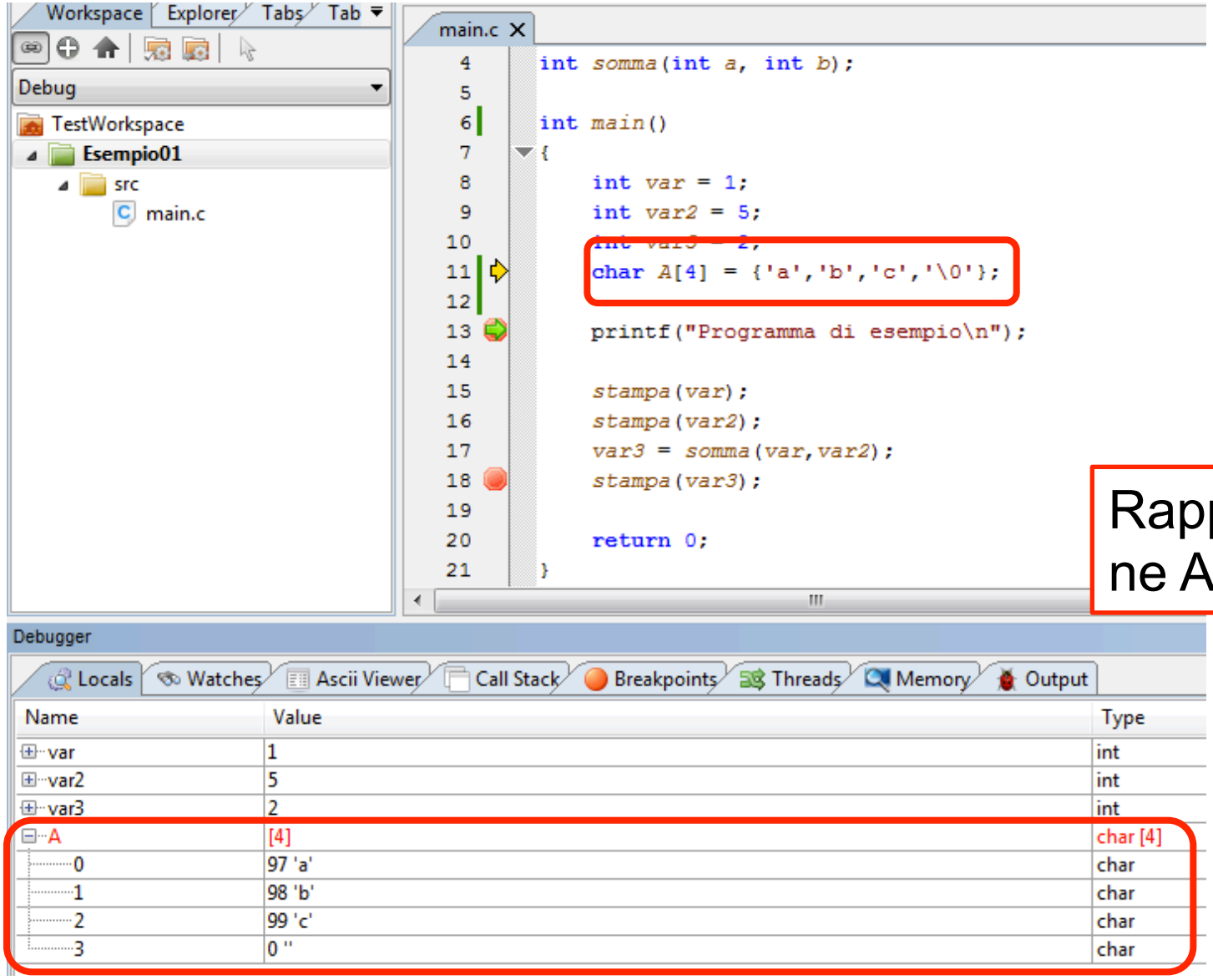
Debugger: Come Procedere

- **Nel menu Debug che compare quando il Debugger e' attivo ci sono alcune voci importanti:**
  - **Execute: esegue il programma fino al prossimo Debug**
  - **Step in: esegue passo passo le istruzioni di una funzione**
  - **Step Out: esegue l'istruzione e torna alla funziona chiamante**
  - **Next: esegue l'istruzione corrente**
  - **Show current line: permette di posizionare il cursore in una determinata posizione nel sorgente e esegue tutte le istruzioni fino ad arrestarsi al cursore.**

Debugger

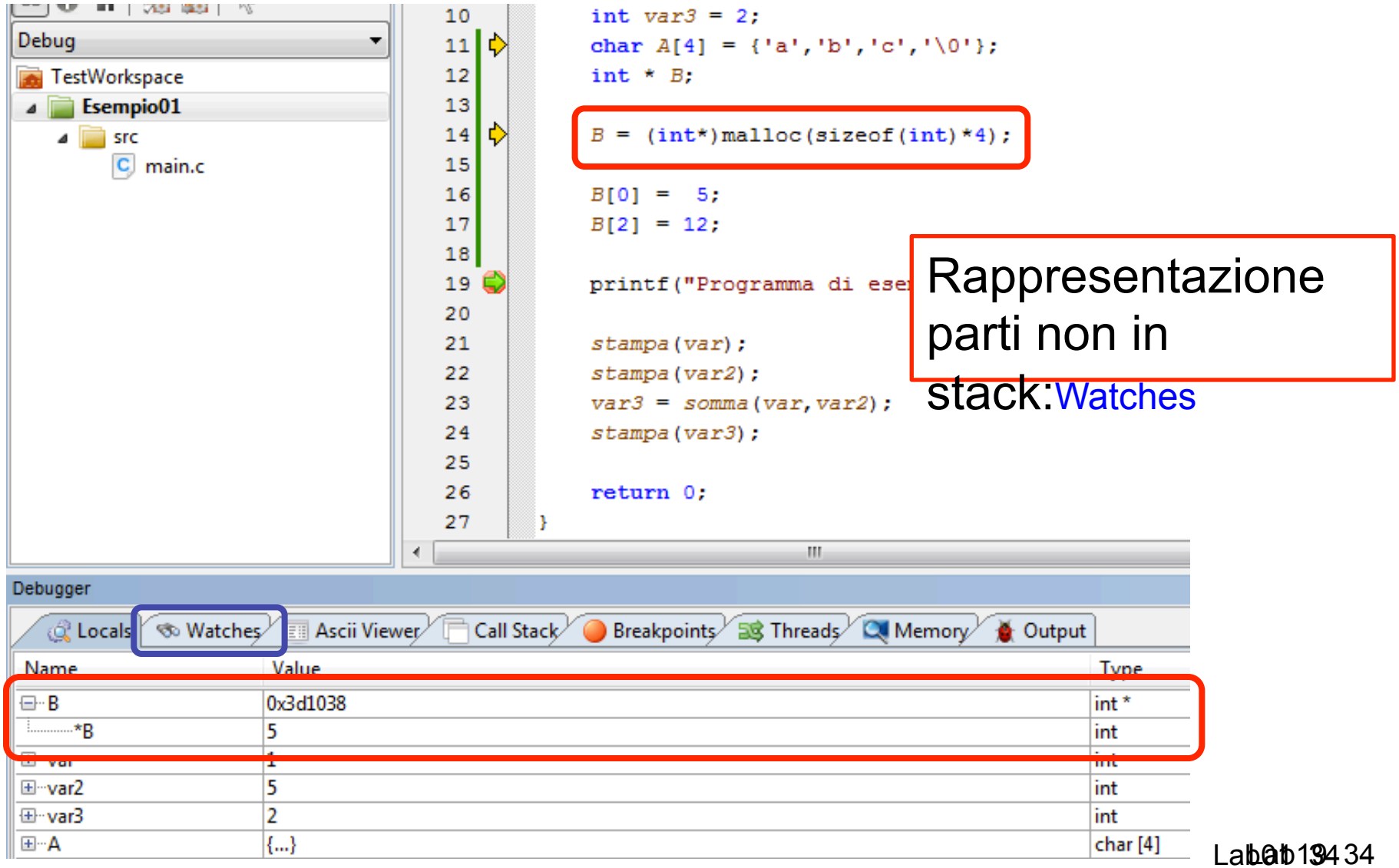

Execute
Show current Line
Step In
Nex
t
Step Out
Workspace View
Workspace
Explorer
Tabs
Debug
TestWorkspace
Esempio01
src
main.c
<global>
main()
4 int somma(int a, int b);
5
6 int ma
7 {
8 int var
9 int var2
10 int var3
11
12 printf("Programma di esempio\n");
13
14 stampa(var);
15 stampa(var2);
16 var3 = somma(var,var2);
17 stampa(var3);
18
19 return 0;
20 }
Debugger
Locals
Watches
Ascii Viewer
Call Stack
Breakpoints
Threads
Memory
Output
Name
Value
Type
var
1
int
var2
5
int
var3
2
int

Debugger



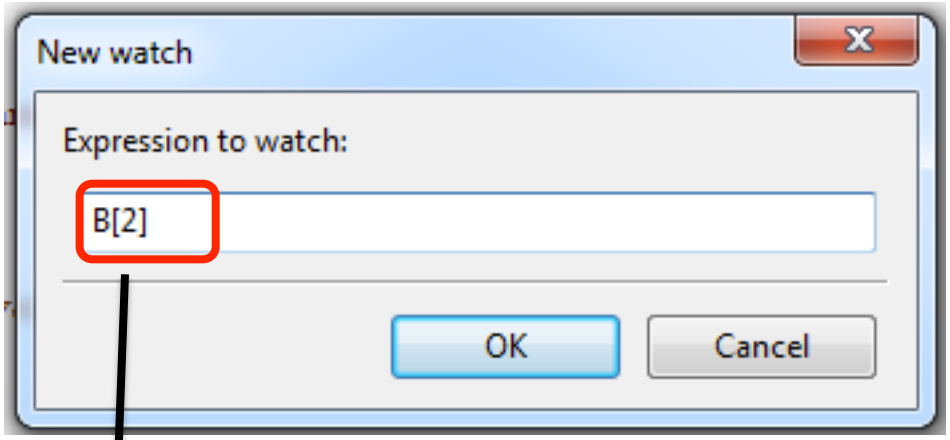
Rappresentazione Array statici

Debugger

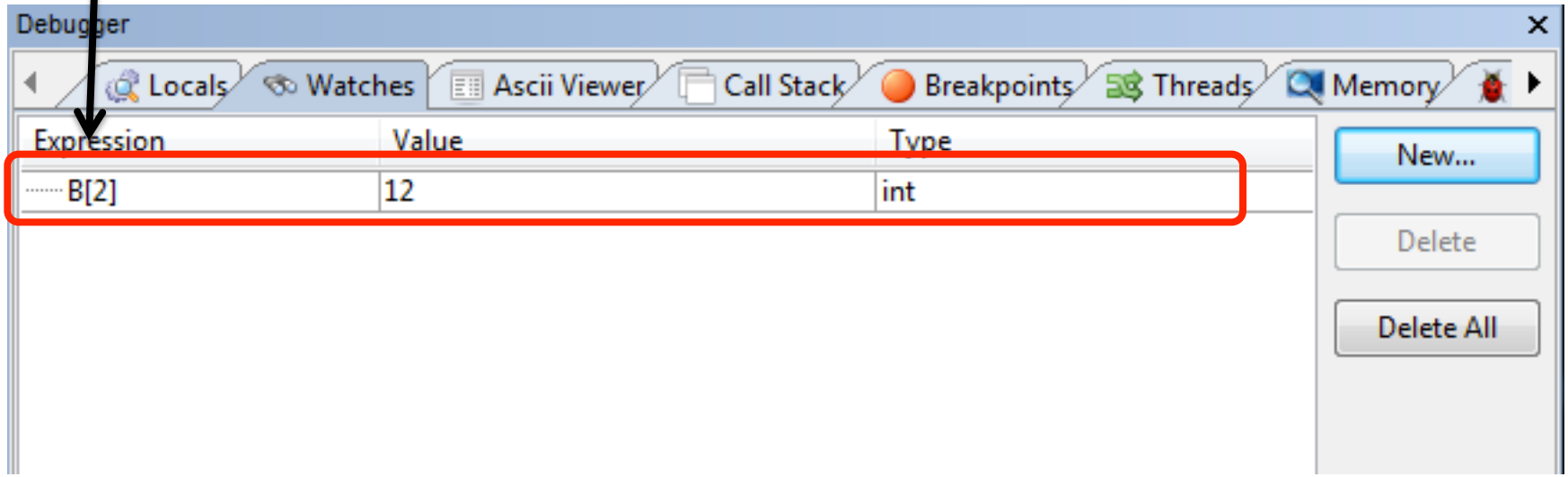


Rappresentazione parti non in stack: Watches

Debugger

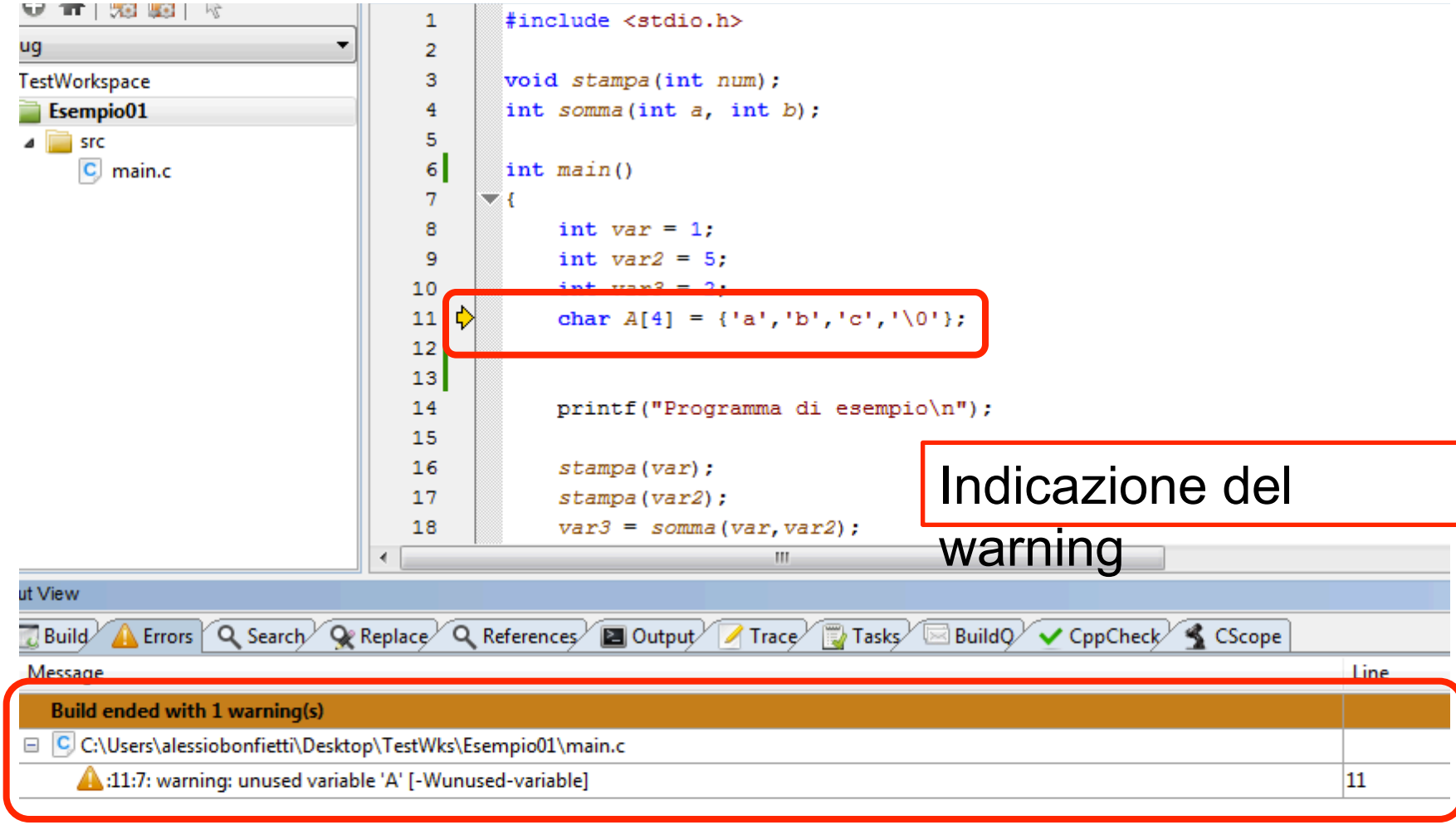


Rappresentazione parti non in stack: Watches



Progetti in Codelite

Build: Warning



Build: Errors

main.c

```
6   int main()
7   {
8       int var = 1;
9       int var2 = 5;
10      int var3 = 2;
11      char A[4] = {'a','b','c','\0'}
12
13
14      printf("Programma di esempio\n");
15
16      stampa(var);
17      stampa(var2);
18      var3 = somma(var,var2);
```

Indicazione degli errori

View

Build | Errors | Search | Replace | References | Output | Trace | Tasks | BuildQ | CppCheck | CScope

Message	Line
Build ended with 1 errors, 1 warnings	
C:\Users\alessiobonfietti\Desktop\TestWks\Esempio01\main.c	
:14:2: error: expected ',' or ';' before 'printf'	14
:11:7: warning: unused variable 'A' [-Wunused-variable]	11

Build | Errors | Search | Replace | References | Output | Trace | Tasks | BuildQ | CppCheck | CScope

```
gcc -c  "C:/Users/alessiobonfietti/Desktop/TestWks/Esempio01/main.c" -g -O0 -Wall  -o ./Debug/main.o -I. -I.
C:/Users/alessiobonfietti/Desktop/TestWks/Esempio01/main.c: In function 'main':
C:/Users/alessiobonfietti/Desktop/TestWks/Esempio01/main.c:14:2: error: expected ',' or ';' before 'printf'
C:/Users/alessiobonfietti/Desktop/TestWks/Esempio01/main.c:11:7: warning: unused variable 'A' [-Wunused-variable]
mingw32-make.exe[1]: *** [Debug/main.o] Error 1
mingw32-make.exe[1]: Leaving directory `C:/Users/alessiobonfietti/Desktop/TestWks/Esempio01'
mingw32-make.exe: *** [All] Error 2
----------Build Ended----------
1 errors, 1 warnings
```

Unix e linguaggi di programmazione: Compilazione di un programma C

- Comando **`gcc -c <file.c>`**
  - Compilatore C e C++
  - Compila **`<file.c>`** producendo il file eseguibile **`<file>.o`**

Unix e linguaggi di programmazione: Linking di un programma C

- Comando **`gcc <file.o>`**
  - Compilatore C e C++
  - Compila **`<file>`** producendo il file eseguibile **`a.out`**
  - Per dare un nome diverso al file prodotto opzione **`-o`**
- Es: **`gcc file_exec.o -o f_ex`**
- Esecuzione: **`./f_ex <parametri>`**

Unix e linguaggi di programmazione: Building di un programma C

- Comando **`gcc <file.c>`**
  - Compilatore C e C++
  - Compila **`<file>`** producendo il file eseguibile **`a.out`**
  - Per dare un nome diverso al file prodotto opzione **`-o`**
- Es: **`gcc file_exec.c -o f_ex`**
- Esecuzione: **`./f_ex <parametri>`**