Debugger

Marco Alberti

Programmazione e Laboratorio, A.A. 2016-2017

Dipartimento di Matematica e Informatica - Università di Ferrara
Ultima modifica: 18 ottobre 2016

Introduzione

Che cos’è un debugger

Il debugger è un programma che consente di eseguire in modo controllato un altro programma, facilitando la ricerca delle cause di malfunzionamento (*bug*).

Funzionalità tipiche:

- Esecuzione passo-passo (stepping)
- Ispezione del valore di variabili ed espressioni
- Interruzione in punti predefiniti (breakpoint)
- Interruzione in caso di modifica del valore di determinate variabili (watchpoint)

GDB

Il debugger per programmi in C più comune in ambiente Unix è il comando GDB (Gnu DeBugger).

Documentazione:

- Guida in linea
- `man gdb`
- `info gdb`
- *cheat sheet*: `https://people.eecs.berkeley.edu/~mavam/teaching/cs161-sp11/gdb-refcard.pdf`

Compilazione con informazioni di debug

L'opzione `-g` del comando `gcc`

(eventualmente combinata con altre opzioni) inserisce nel codice generato le informazioni di debug, utili per esaminare il programma quando lo si esegue nel debugger.

Esempio

`gcc -g __nome__.c` compila con informazioni di debug il file `__nome.c__`, generando l'eseguibile `a.out`.

Comandi

Avvio e chiusura del debugger

`gdb __eseguibile__`

apre il debugger caricando il programma `__eseguibile__`.

Ora il debugger è in attesa di comandi.

`file __eseguibile__`

carica il programma `__eseguibile__` (utile ad esempio se si è invocato `gdb` senza argomenti).

`quit`

chiude il debugger.

Abbreviazioni ed help in linea

Abbreviazioni

In generale tutti i comandi di GDB ammettono qualsiasi abbreviazione non ambigua (ad esempio `quit` può essere abbreviato con `q`, `qu` o `qui`).

`help`

visualizza la guida in linea.

Avvio

`run`

avvia il programma.

Breakpoint

`break __nome_funzione__`

imposta un breakpoint all'inizio della funzione `__nome_funzione__` (cioè l'esecuzione sarà interrotta all'inizio dell'esecuzioe di `__nome_funzione__`).

Esempio

`b main` imposta un breakpoint all'inizio del programma

`break __numero_riga__`

imposta un breakpoint alla riga numero `__numero_riga__`.

Esempio

`b 5` imposta un breakpoint alla riga `5`.

Breakpoint condizionali

`if __condizione__`

dove `__condizione__` è un'espressione logica, aggiunto all'impostazione di un breakpoint fa sì che l'esecuzione si interrompa solo se `__condizione__` ha valore vero.

Esempio

`b 25 if i == 5` interrompe l'esecuzione alla riga `25` se `i == 5` è vero.

Visualizzazione ed eliminazione breakpoint

info breakpoints

mostra i breakpoint impostati, numerati.

delete breakpoint __n__

elimina il breakpoint numero __n__.

Continuazione

`continue`

riprende l’esecuzione interrotta di un programma, fino al successivo breakpoint.

Esecuzione passo-passo

`step`

esegue una singola istruzione (eventualmente entrando nel codice delle funzioni chiamate).

`next`

esegue una singola istruzione (completando in un passo unico un'eventuale chiamata di funzione).

Visualizzazione codice

`list`

visualizza le 10 righe di codice attorno alla prossima che verrà eseguita.

`list __numero_riga__`

visualizza le 10 righe di codice attorno a `__numero_riga__`.

Visualizzazione espressioni

`print __espressione__`

visualizza il risultato della valutazione di `__espressione__` (che può essere anche solo l'identificatore di una variabile).

Esempio

`p a` visualizza il contenuto della variabile `a`.

`p a + 10` visualizza il risultato dell'espressione `a + 10`.

`info locals`

elenca, con il loro valore, tutte le variabili locali.

Watchpoints

`watch __espressione__`

interrompe l'esecuzione ogni volta che `__espressione__` cambia di valore.

Esempio

`watch a` interrompe l'esecuzione quando la variabile `a` cambia di valore.

`watch a + b` interrompe quando l'espressione `a + b` cambia di valore.

Integrazione con Visual Studio Code

Utilizzo di GDB dentro Visual Studio Code

Visual Studio Code consente il debugging di un programma con i principali comandi di GDB dalla stessa finestra di editing del codice sorgente.

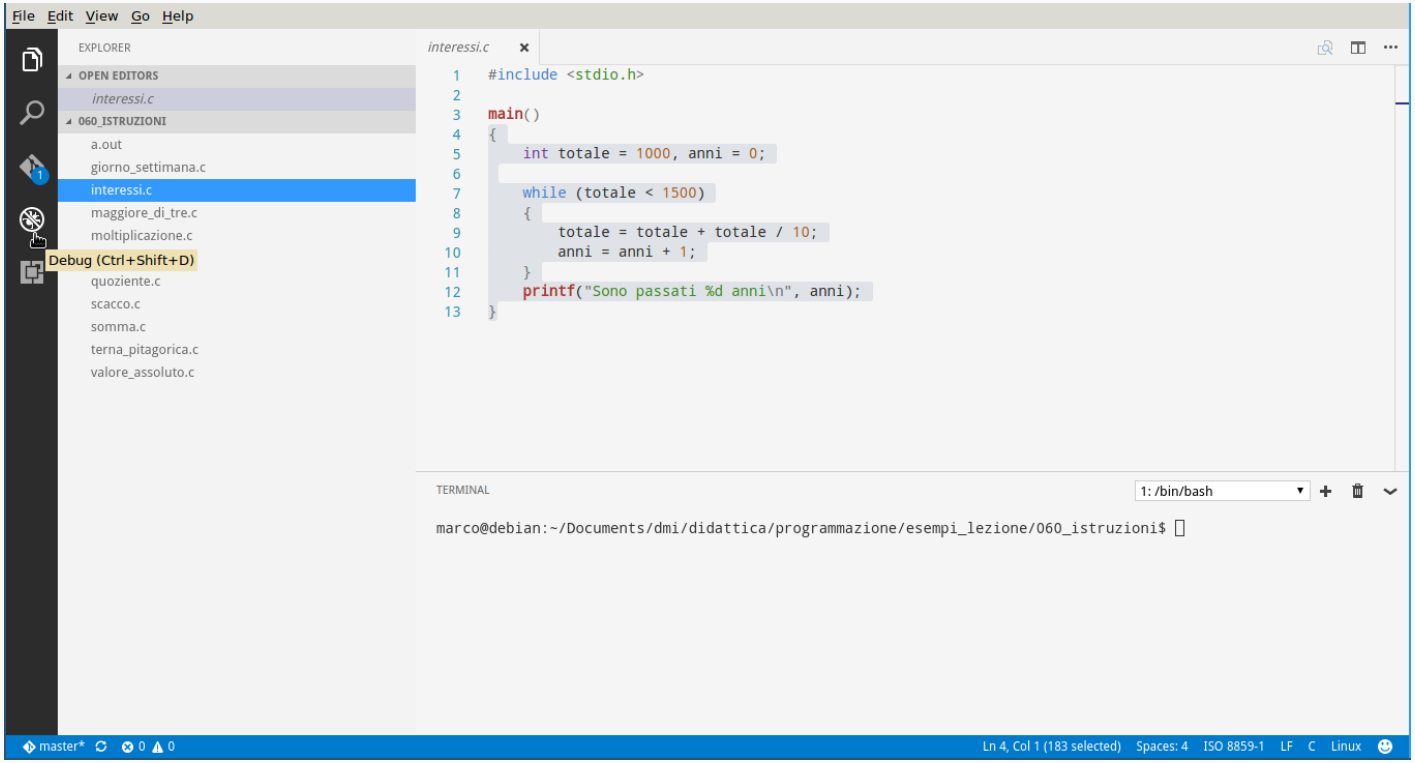
In particolare è possibile:

- Eseguire passo-passo vedendo nel codice sorgente l'istruzione eseguita
- Impostare breakpoint nel codice sorgente
- Visualizzare i valori delle variabili locali di una funzione

Informazioni dettagliate:
`https://code.visualstudio.com/docs/languages/cpp`.

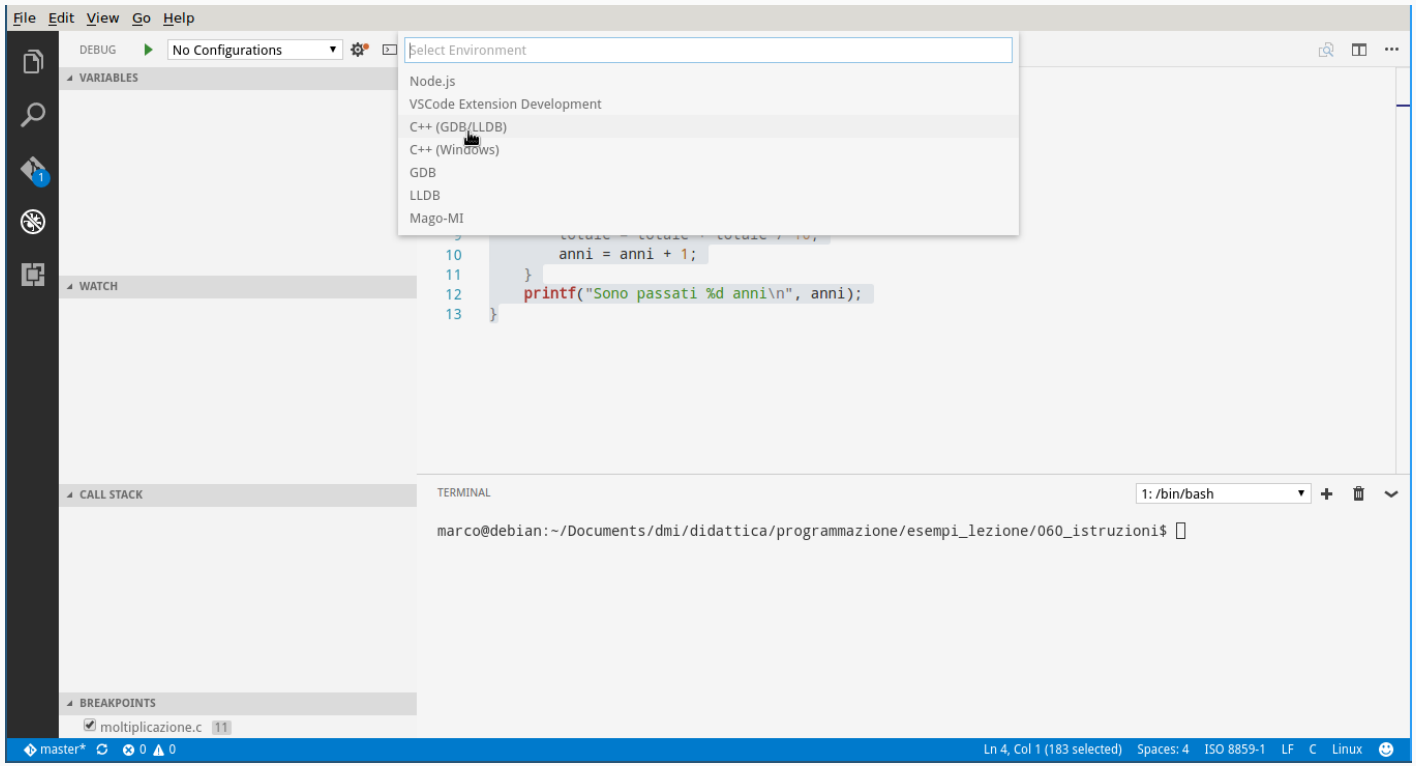
Finestra di debug

La finestra di debug si richiama con la combinazione CTRL+SHIFT+D o con l'apposita icona.



Impostazione di Visual Studio Code per debugging

E’ necessario impostare la comunicazione fra Visual Studio Code e GDB nel file `__cartella__/vscode/launch.json`, dove `__cartella__` è la cartella di lavoro (da selezionare con il comando `Open folder` del menu `file`). Per modificare il file cliccare sull’icona dell’ingranaggio; se non esiste cliccare sull’icona stessa e poi su C++(GDB/LLDB).



Tipico contenuto del file launch.json

Il seguente contenuto è sufficiente per iniziare:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "C++ Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceRoot}/a.out",
            "stopAtEntry": false,
            "cwd": "${workspaceRoot}"
        }
    ]
}
```

(eventualmente sostituire a.out con il nome dell'eseguibile)