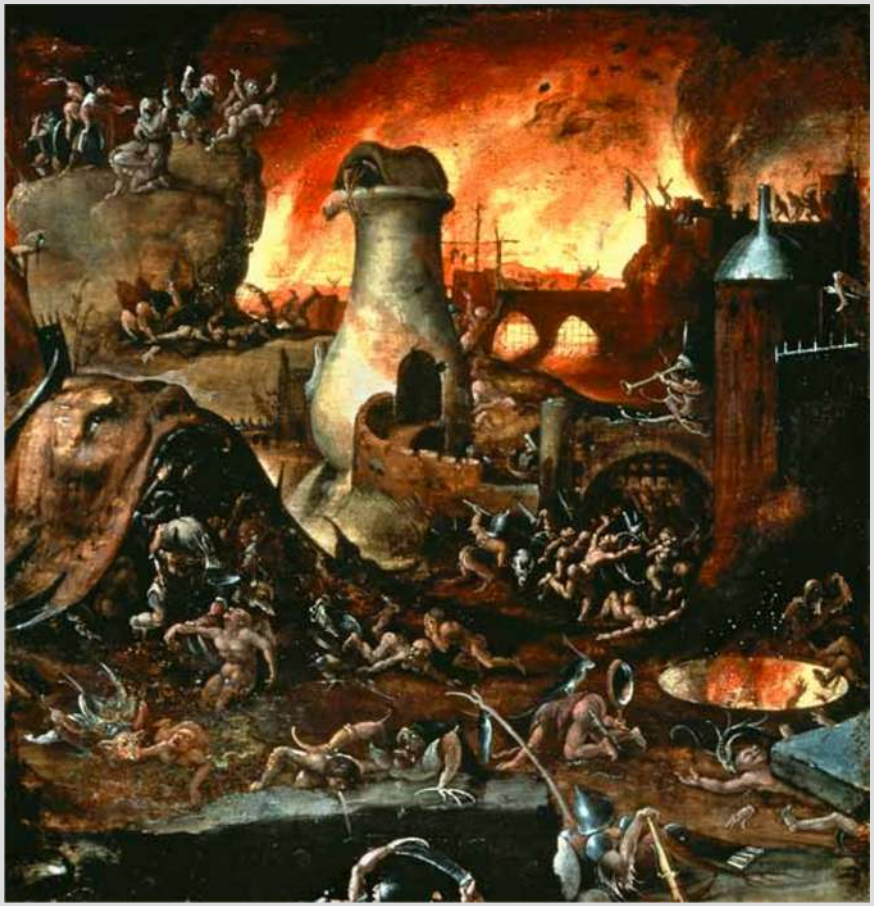
Parte 8 – Sviluppo e testing

Debugging



[Hieronymus Bosch – Hell, ~1500]

Software bug*

* http://en.wikipedia.org/wiki/Software_bug

- **Bug:** Errore o guasto che porta al malfunzionamento del software (per esempio producendo un risultato inatteso o errato)
- La **causa** del maggior numero di bug è spesso un errore nel codice sorgente scritto da un programmatore
- Un programma che contiene un gran numero di bachi che interferiscono con la sua funzionalità è detto **bacato** (in inglese *buggy*).

Bug report: relazione che dettaglia i bug di un programma

Tipologie di bug

Syntax error

- Errori che si commettono nella fase di **scrittura del programma**
- Causati dal fatto che non si rispetta la **sintassi** del linguaggio
  - Ad esempio dimentico il ; al termine dell'istruzione
- Impediscono la compilazione del programma
- Segnalati direttamente dall'editor dell'IDE e risultando dunque di facile individuazione e soluzione

Logic error

- Errori che si commettono nella fase di **progettazione dell'algoritmo** o nella sua **codifica**
- Causati, ad esempio, da una mancata comprensione del problema
- L'algoritmo non fornisce l'output richiesto nell'ambito di una o più istanze del problema da risolvere
- Sono difficili da individuare e spesso si deve ricorrere al **debugging**

Tipologie di bug (cont.)

Runtime error

- Errori che si verificano nella fase di **esecuzione del programma**, anche se l'algoritmo è corretto e il codice viene compilato correttamente
- Spesso sono relativi all'utilizzo della memoria da parte del programma stesso, che tenta ad esempio di scrivere ad una locazione di memoria alla quale non ha accesso
- Questi errori si verificano principalmente nell'utilizzo dei puntatori e in programmazione dinamica

Effetti dei bug

- Alcuni bug hanno solo un **effetto sottile** sulla funzionalità del programma e possono quindi non essere rilevati per molto tempo
- Bug più gravi possono causare il **crash** o il blocco (**freeze**) del programma
- Altri bug si qualificano come **bug di sicurezza** e potrebbero, ad esempio, consentire a un utente malintenzionato di ignorare i controlli di accesso per ottenere privilegi non autorizzati

Crash

- Condizione in cui un programma interrompe l'esecuzione delle funzioni previste (**si chiude in modo anomalo**)
- Ad esempio, a causa di una divisione per 0 o per l’accesso ad un’area di memoria non destinata al programma
- Se questo programma è una parte critica del kernel del sistema operativo, l'intero computer potrebbe bloccarsi

Freeze

- Condizione in cui il programma non risponde agli input
- Causato, ad esempio, dalla presenza di un loop infinito nel programma
- La finestra interessata dal programma o l'intero schermo del computer diventa statico
- Quando nessun altro comando di interruzione funziona, è necessario spegnere e riaccendere!

Debugging

Processo metodico per trovare e ridurre il numero di bug in un programma

- Attività che richiede molto tempo
- Spesso i programmatori dedicano più tempo e fatica a trovare e correggere bug che a scrivere un nuovo codice
- Di solito, la parte più difficile del debug è trovare il bug nel codice sorgente. Una volta trovato, correggerlo è relativamente facile

Un task difficile

LOCALIZZARE UN BUG E' UN'ARTE!

- Un bug in una sezione di un programma può causare errori in una sezione completamente diversa
- A volte, un bug non è un difetto isolato, ma rappresenta un errore di pensiero / pianificazione (vedi errori logici)
- Alcune classi di bug non hanno nulla a che fare con il codice stesso (ad esempio, relativo all'hardware)

Il processo di debugging

- Il primo passo per individuare un bug è riprodurlo in modo affidabile ricostruendo la situazione che ha dato origine al bug (ad esempio i valori in input che sono stati forniti)
- Quindi, solitamente si utilizza un **debugger** per monitorare l'esecuzione del programma nella regione difettosa e trovare il punto in cui il programma è andato fuori strada
- Tuttavia, alcuni bug possono scomparire quando il programma viene eseguito con un debugger!

"**Heisenbug**" https://it.wikipedia.org/wiki/Heisenbug

Debugger

Programma che aiuta i programmatori a localizzare bug tramite funzionalità specifiche quali:

- esecuzione del codice riga per riga (**step-by-step**)
- monitoraggio dei valori variabili
- altre funzionalità per osservare il comportamento del programma

Funzionamento del debugger

Un debugger tipicamente prende in ingresso **soltanto il file eseguibile** da testare

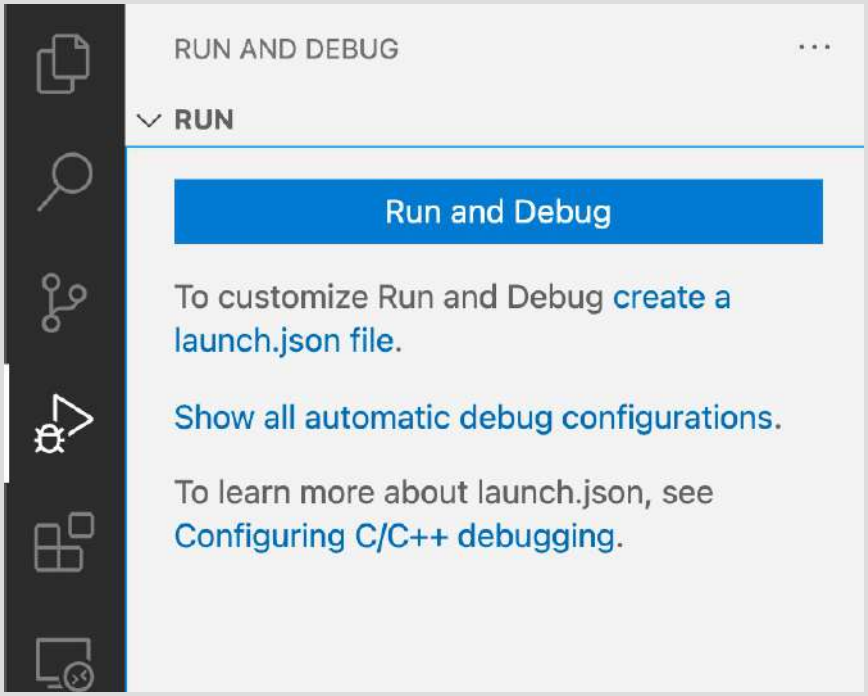
- E' necessario che l'eseguibile contenga informazioni che permettano di mettere in relazione ciascun dato o istruzione nell'eseguibile con la corrispondente entità nel sorgente (variabile, costante, istruzione, …)
- Ad esempio il debugger **gdb:** http://www.gnu.org/software/gdb/
- Per usare gdb su un eseguibile, è necessario che l'eseguibile sia stato compilato aggiungendo l'**opzione -g** (con gcc/g++)

VEDIAMO IL FUNZIONAMENTO DEL **DEBUGGER** DI **Visual Studio Code** INIZIANDO DAL PROGRAMMA HelloWorld2.cpp

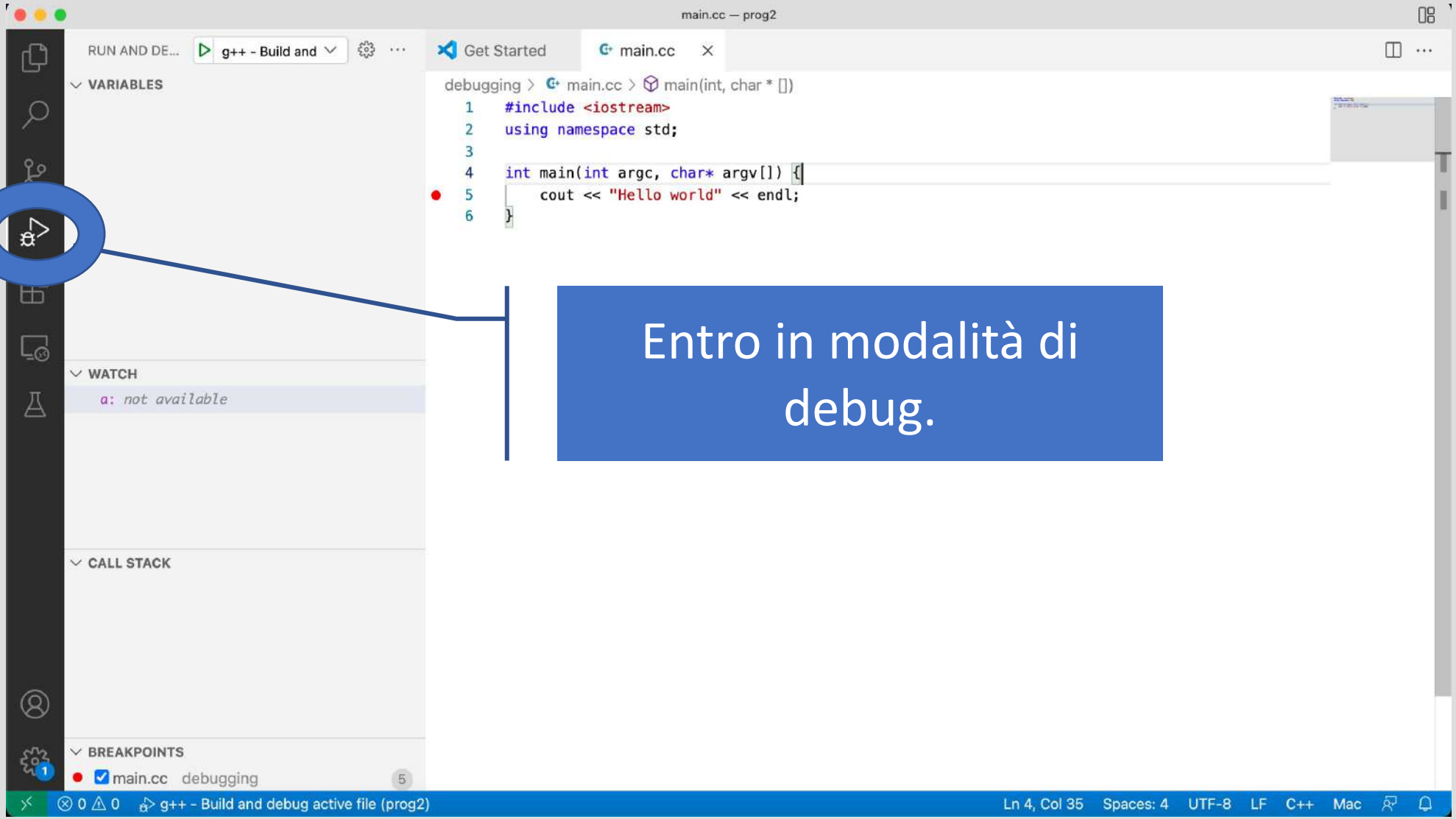
Prima esecuzione del debugger

Nella schermata del debug viene presentata questa finestra

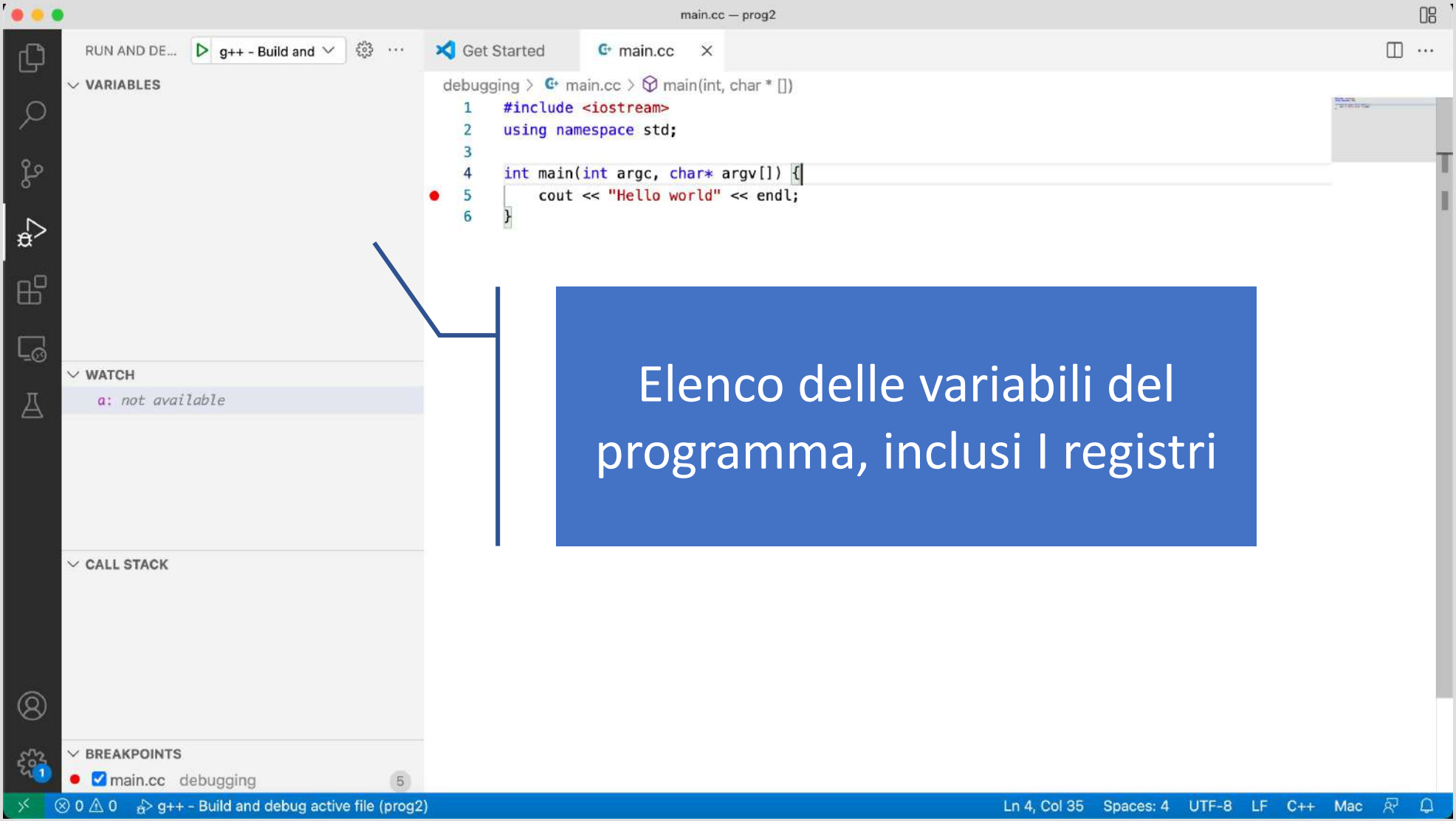
Cliccando su ‘run and debug’ viene creata una configurazione di base per lanciare il debug, che viene memorizzata in **launch.json**



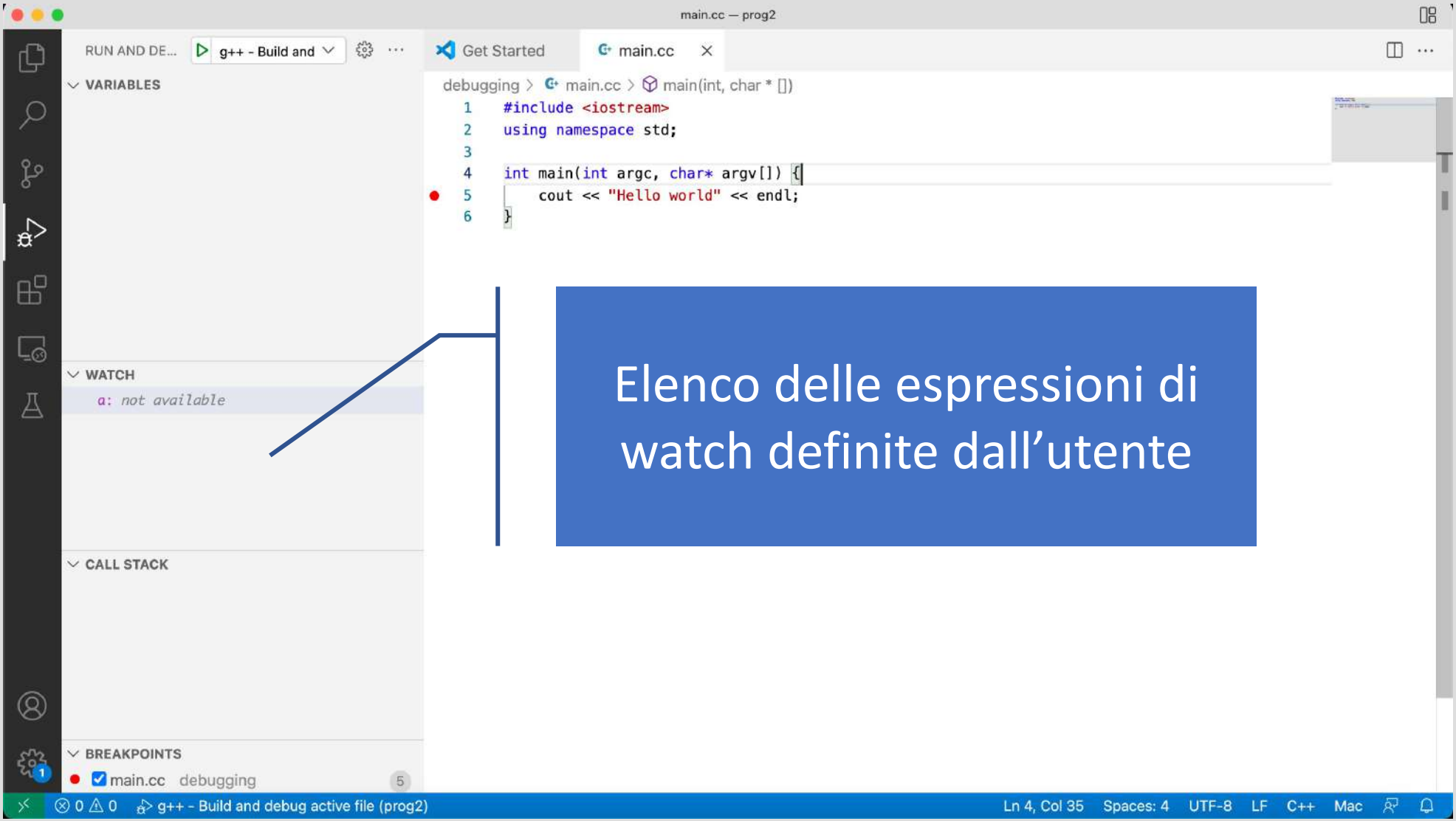
Proviamo il debugger di VSCode
sul programma HelloWorld.cc



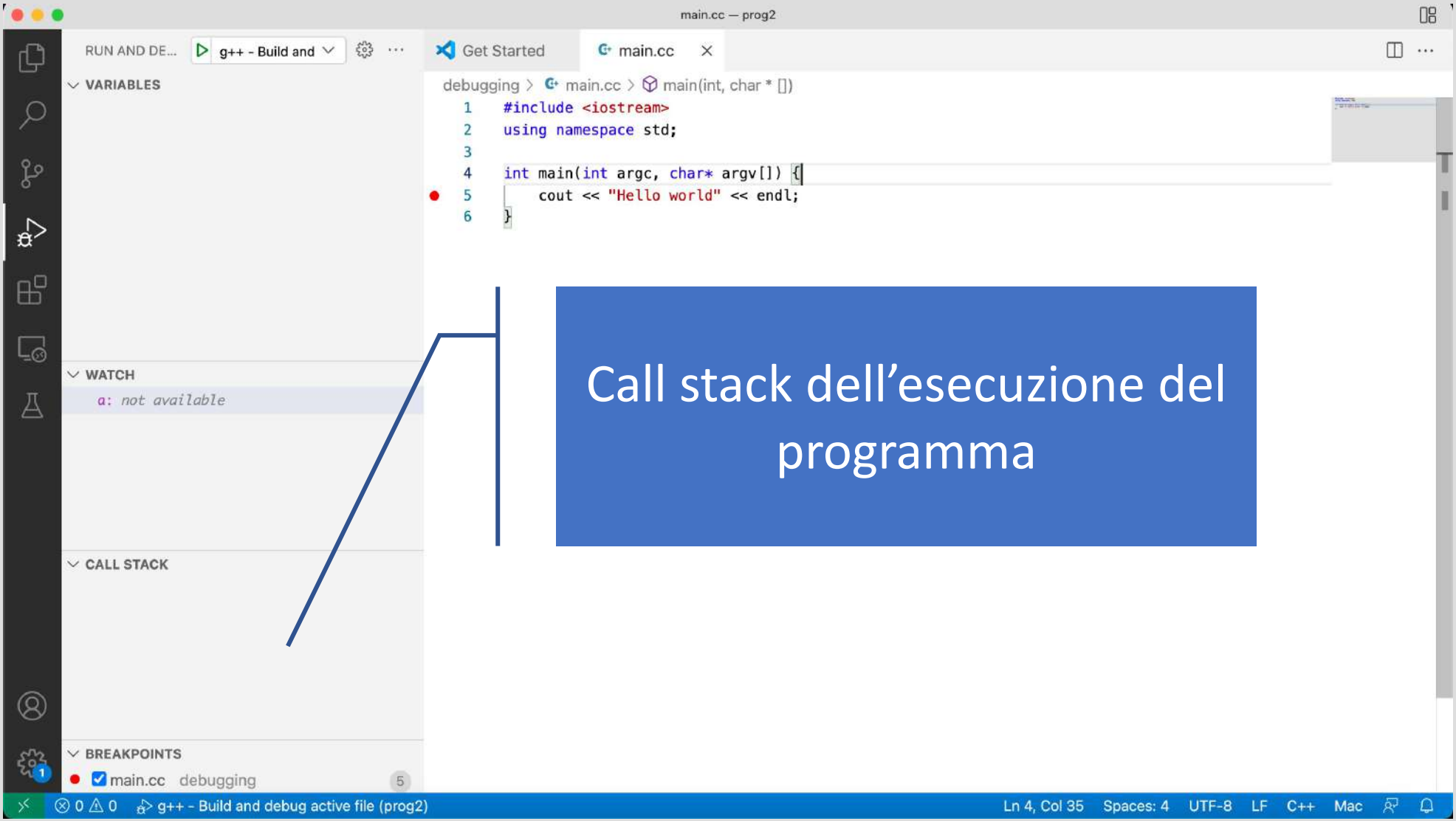
Proviamo il debugger di VSCode sul programma HelloWorld.cc



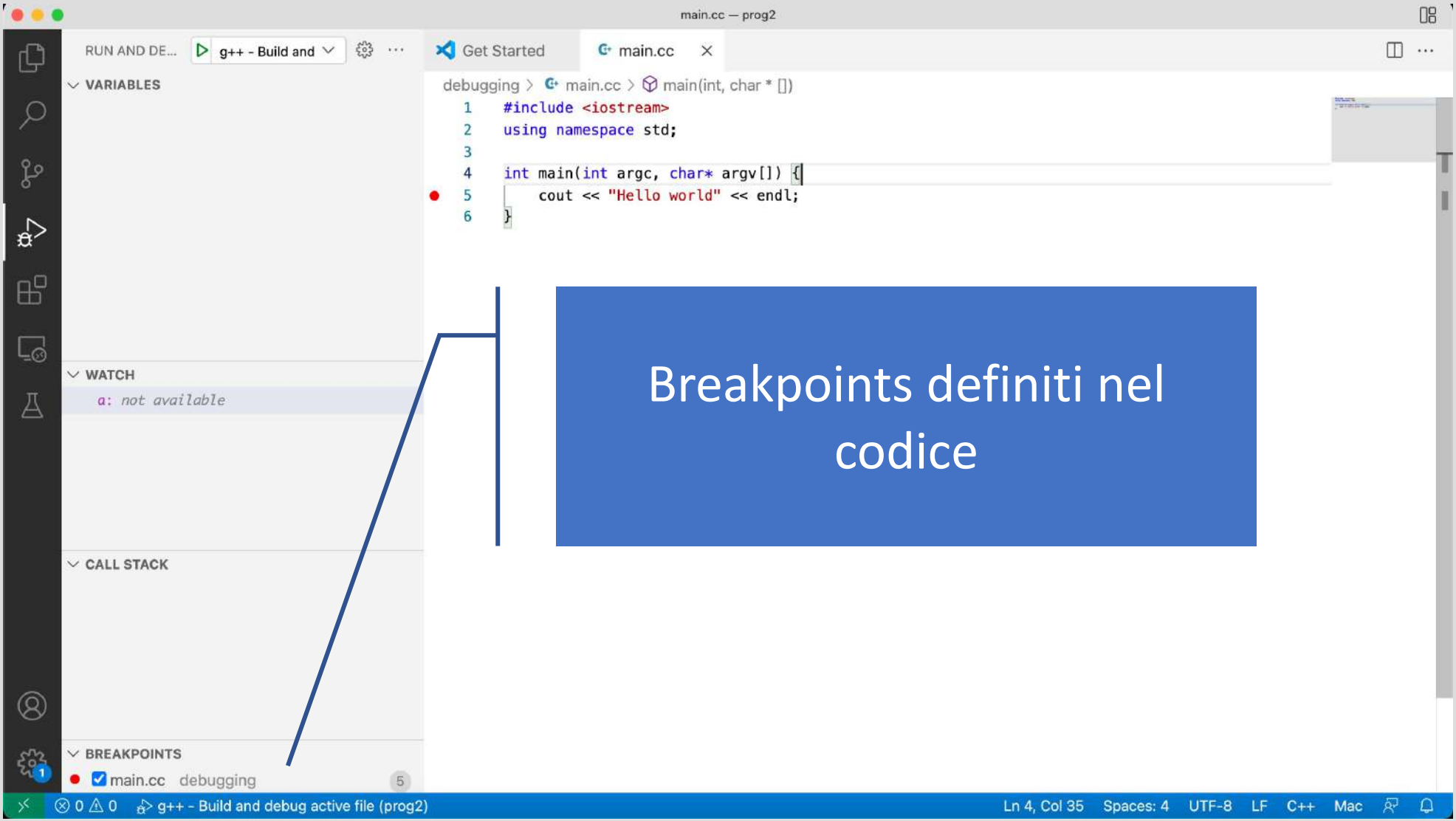
Proviamo il debugger di VSCode sul programma HelloWorld.cc



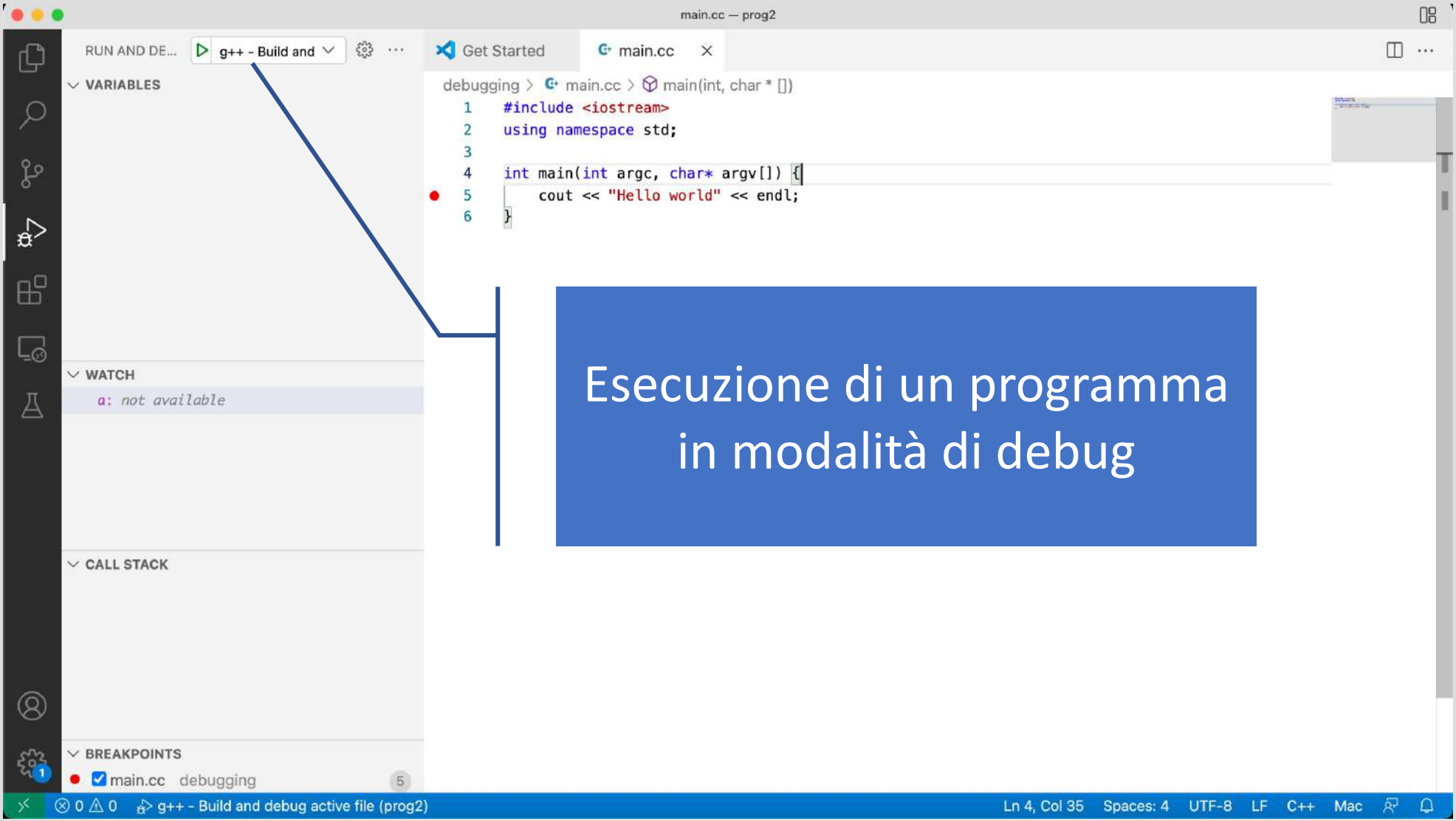
Proviamo il debugger di VSCode sul programma HelloWorld.cc



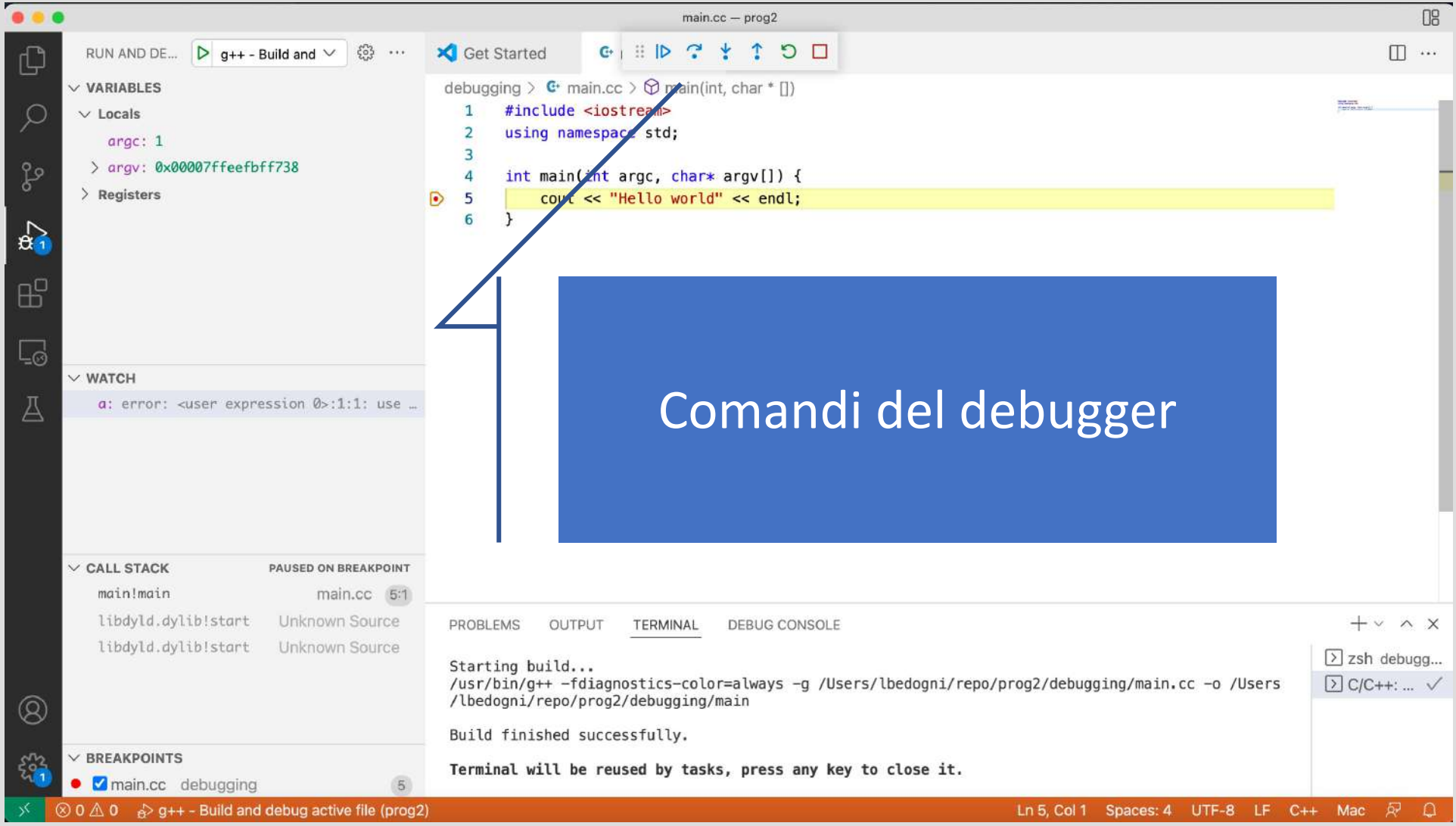
Proviamo il debugger di VSCode sul programma HelloWorld.cc



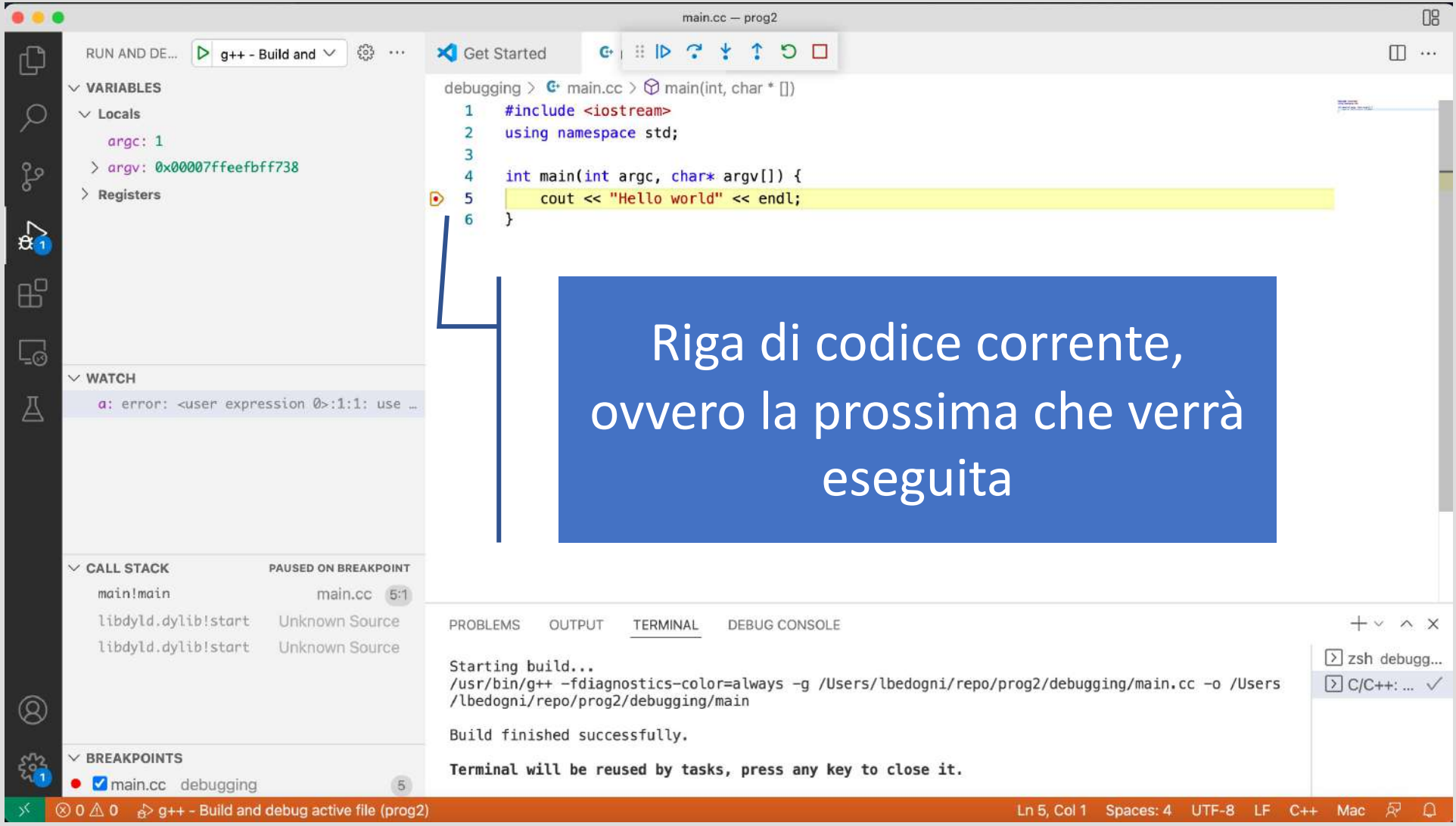
Proviamo il debugger di VSCode sul programma HelloWorld.cc



Proviamo il debugger di VSCode sul programma HelloWorld.cc



Proviamo il debugger di VSCode sul programma HelloWorld.cc



Uso dei breakpoint

- L'importanza del debugger entra ovviamente in gioco quando si esegue il programma **step-by-step**
- Come facciamo a interrompere l'esecuzione di un programma in un punto del codice per poterlo poi eseguire passo-passo?

  Mettiamo un **BREAKPOINT**

Breakpoint in VSCode

Per **aggiungere/eliminare** un breakpoint in VSCode:

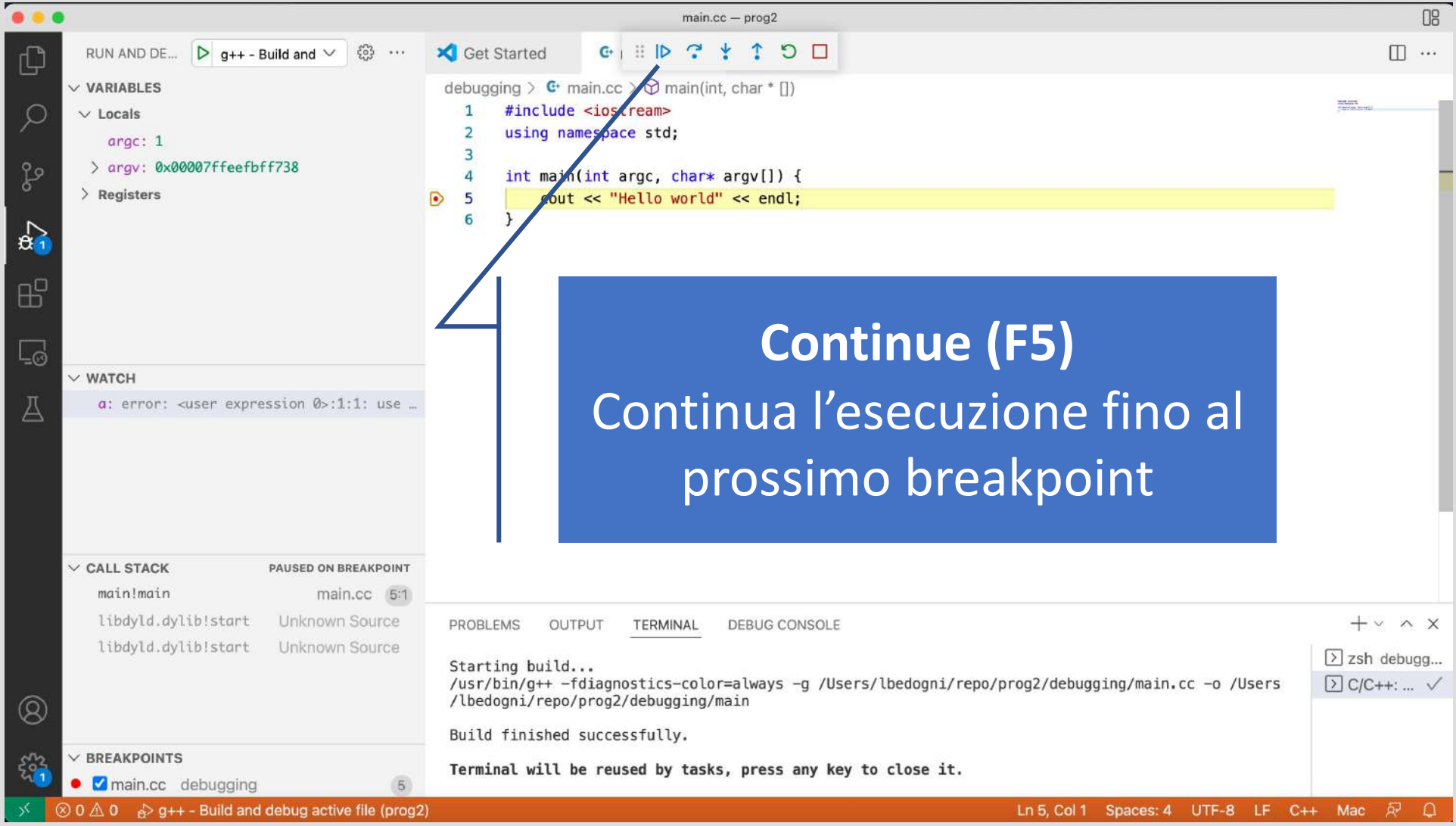
- Puntare all'inizio della riga del codice sorgente in cui vogliamo aggiungere/eliminare un breakpoint
- Cliccare 2 volte sulla riga

oppure

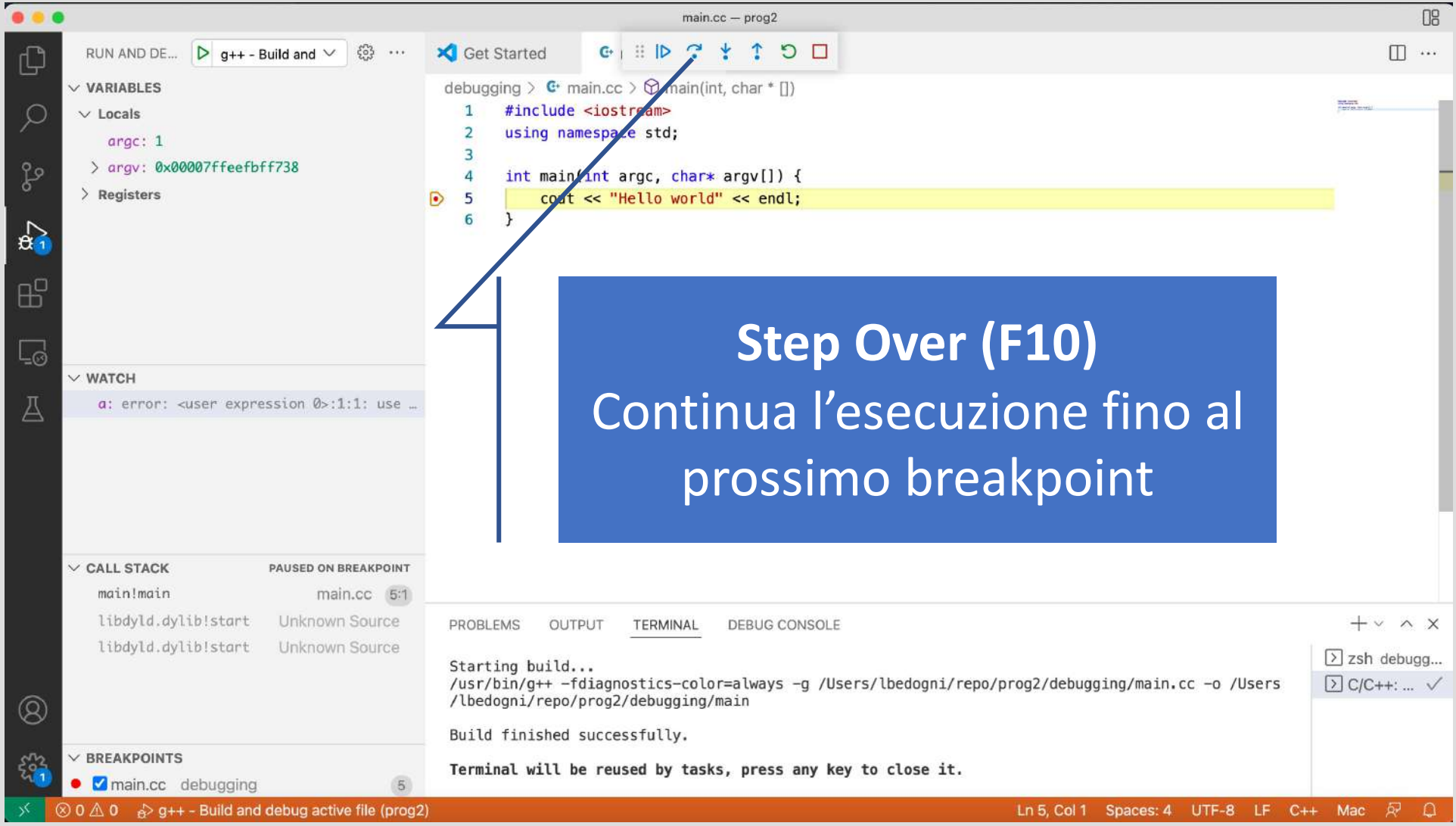
tasto destro → «Add Breakpoint…»/
«Remove Breakpoint»

- Durante la sessione di debugging, è possibile aggiungere un numero di breakpoint a piacere

Uso dei comandi

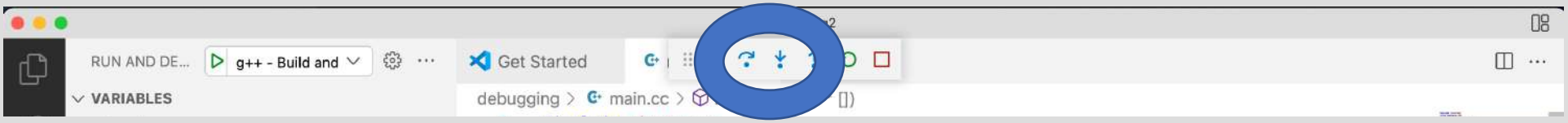


Uso dei comandi



Run → Step into (F11) e Step over (F10)

Eseguiamo il programma step-by-step



- **Step into** e **Step over** eseguono la riga corrente
- Ripetendo il comando più volte è quindi possibile eseguire il programma passo-passo
- I due comandi differiscono nell'esecuzione di funzioni (contenute nella riga da eseguire) in quanto:

  Step over: Esegue le chiamate di funzione senza 'entrarvi' dentro

  Step into: Entra nel codice delle funzioni

Run → Terminate(Shift+F5)

Eseguiamo il programma step-by-step

- **Terminate** permette di concludere l'esecuzione del programma
- E' anche possibile riavviarlo sempre in modalità debug

Esercizio

VEDIAMO IL FUNZIONAMENTO DI **STEP INTO** E **STEP OVER**:

- Mettiamo un breakpoint alla riga 16 del codice
- Eseguiamo il codice fino al prossimo breakpoint
- Proviamo ad eseguire la riga con il commando Step into
- Riapplichiamo l'intero processo di debugging usando il commando Step over

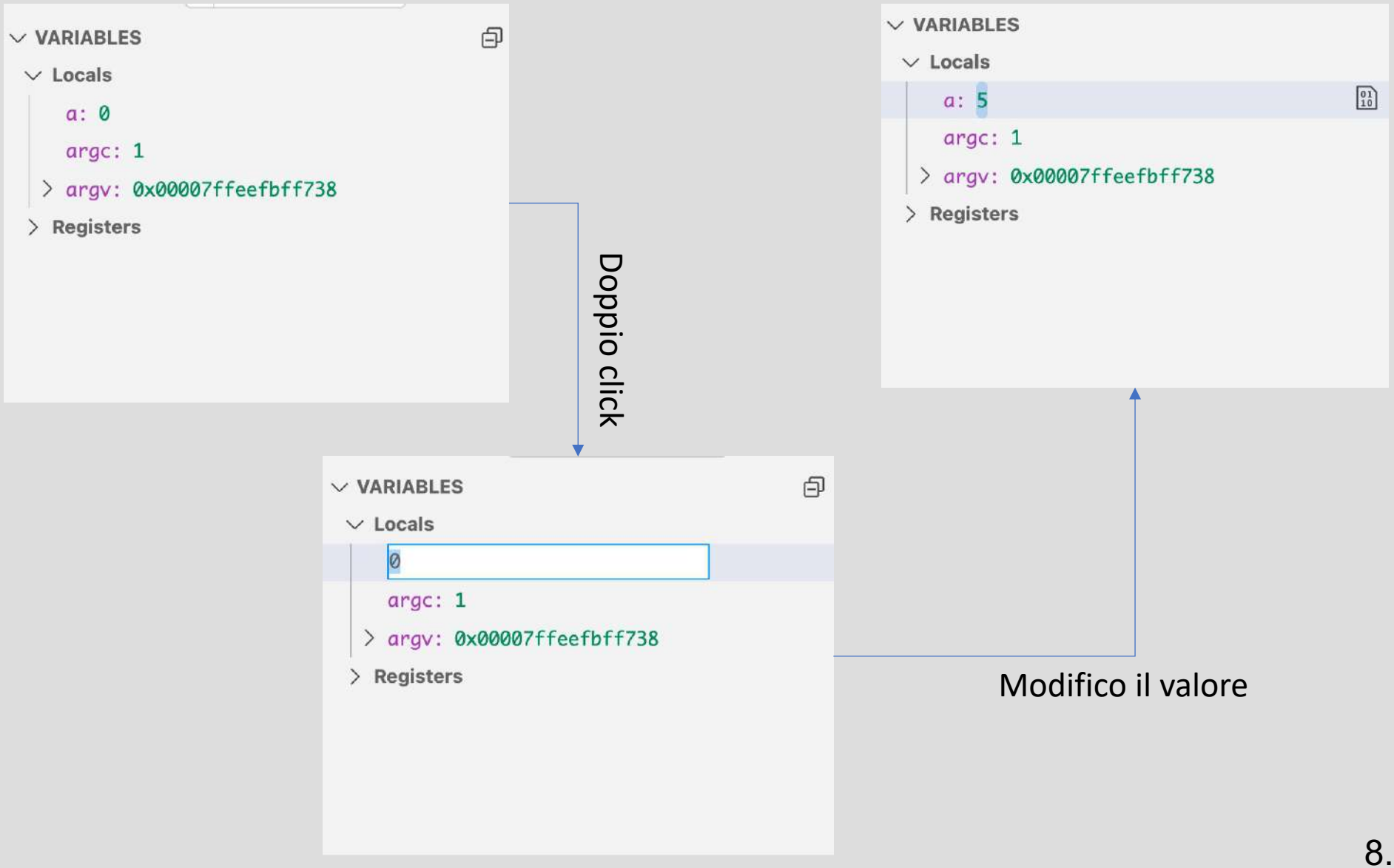
Prossimo programma

Nelle prossime slide faremo riferimento al programma *variabili_input_ciclo.cc*

Visualizzazione variabili

- L'esecuzione di un programma produce il **cambiamento di valore delle variabili**, ed è guidata dal valore delle variabili stesse
- Pertanto la **conoscenza del valore delle variabili** ci dice praticamente tutto sullo stato di un programma
- Un debugger supporta la **visualizzazione delle variabili**
- Un debugger può anche **forzare** il valore delle variabili

Forzare il valore di una variabile



Abilitiamo l'input

Possiamo modificare la configurazione di debug cliccando sull'ingranaggio nella finestra di debug

Questo ci apre il file **launch.json**, che contiene le configurazioni per il lancio del nostro programma

In particolare per abilitare l'input mettiamo a **true** il parametro "**externalConsole"**

Questo abilita un terminale esterno per l'input

Continuiamo con l'esempio

- Mediante **Step into** e **Step over** eseguiamo il programma passo-passo
- Con l'esecuzione dell'operatore di ingresso su **`cin`** il programma si blocca
- Il programma sta aspettando che immettiamo un valore intero da **stdin**
- Immettiamo il valore nella finestra che si è aperta

Esecuzione ciclo e funzione

- Premendo più volte il tasto **Step over** si può osservare l'esecuzione del ciclo for per la stampa dei valori degli elementi dell'array
- Fermarsi subito prima dell'invocazione della funzione **`fun`**
- Cambiare il valore della variabile **`b[1]`**
- Premere **Step into** per entrare nella funzione **`fun`**

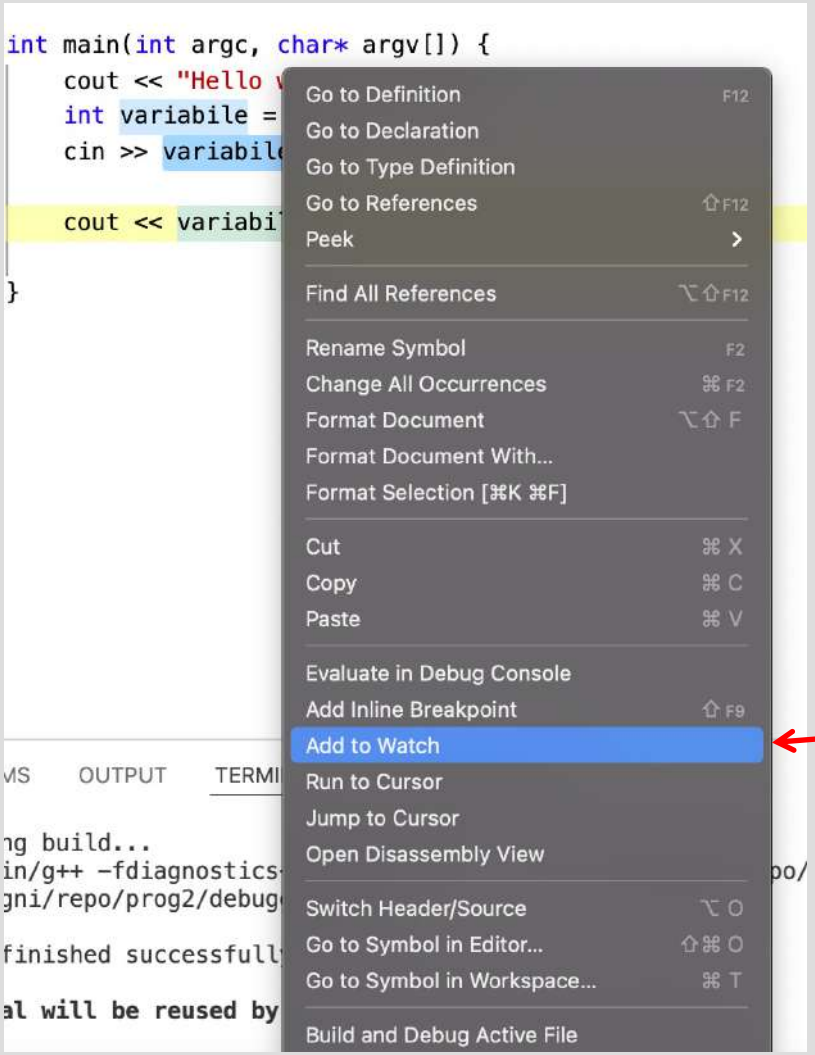
Visualizzazione veloce

Se si sposta semplicemente il puntatore su una variabile appare un piccolo riquadro con il valore corrente della variabile rispetto allo scope

DOMANDA:

- quando stai eseguendo step-by-step la funzione **`fun`** e passi sulla variabile **`a`** nel **`main`**, quale valore vedi?

Funzioni associate alle variabili



Aggiunge la variabile tra le espressioni continuamente monitorate durante l'esecuzione

Esercizio

Utilizzando **solo il debugger** trovare eventuali errori nei programmi

- *memoria.cpp*
  - *Lavorare sul tipo di dato passato*
- *potenze.cpp*
  - *Il programma deve stampare tutte le potenze cubiche dal numero di partenza fino a 1*
- *calcolaMeta.cpp*
  - *Il programma deve visualizzare la metà del numero inserito*