**Università di Pisa**

**Corso di Laurea in Ingegneria Informatica**

# Guida al Debugging di programmi C, C++ e Assembler utilizzando il Data Display Debugger (DDD)

a cura di
**Marco Cococcioni**

# Cos'è DDD

- DDD sta per Data Display Debugger
- DDD è una intefaccia grafica (GUI) per lo GNU debugger, un programma di debug disponibile alla riga di comando.

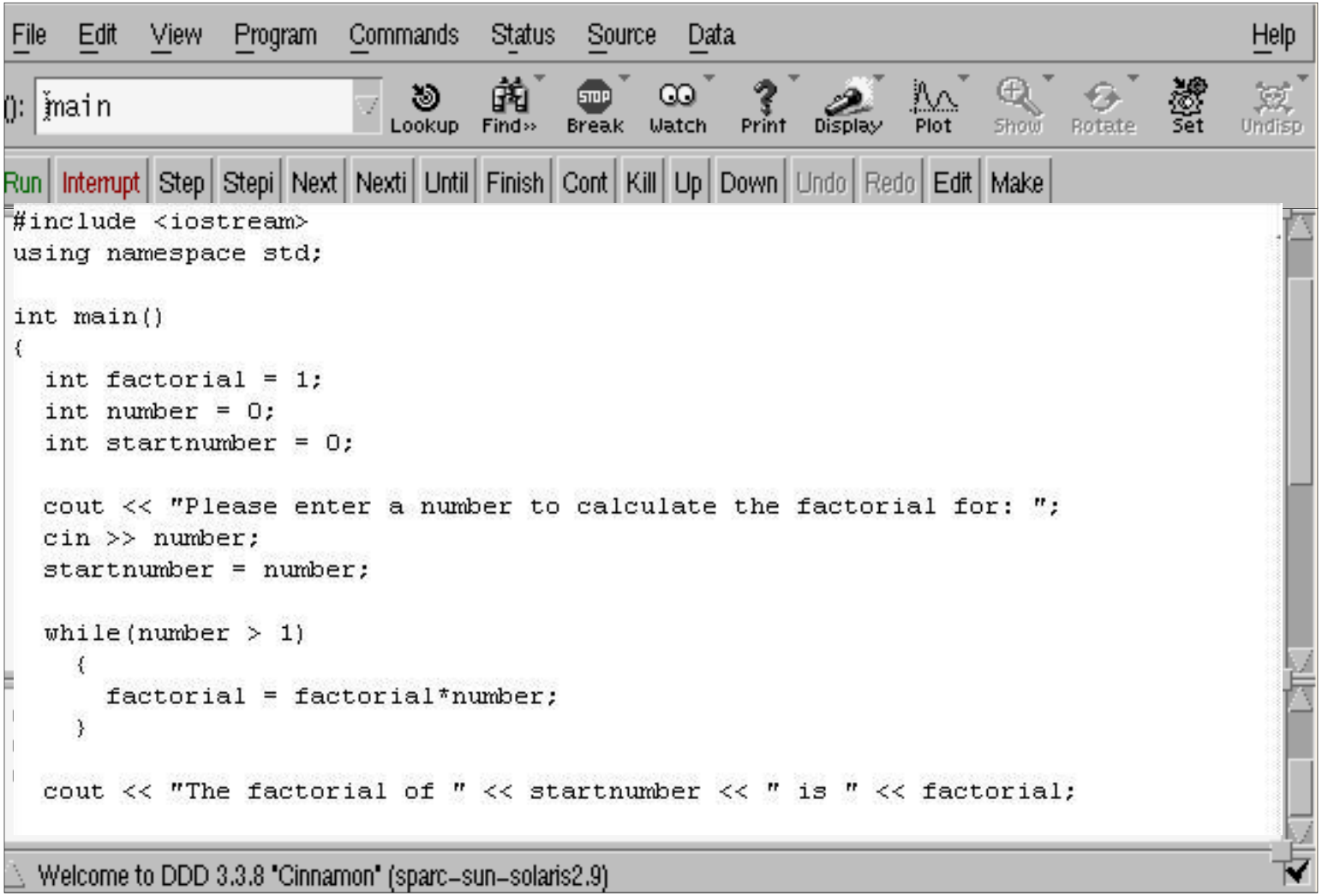
- Come si avvia DDD alla diga di comando?
- Passo 1: compilare e linkare inserendo le informazioni per il debugger (opzione -g):

```
$ g++ -g es1.cpp -o es1.exe
```

- Passo 2: avviare ddd:

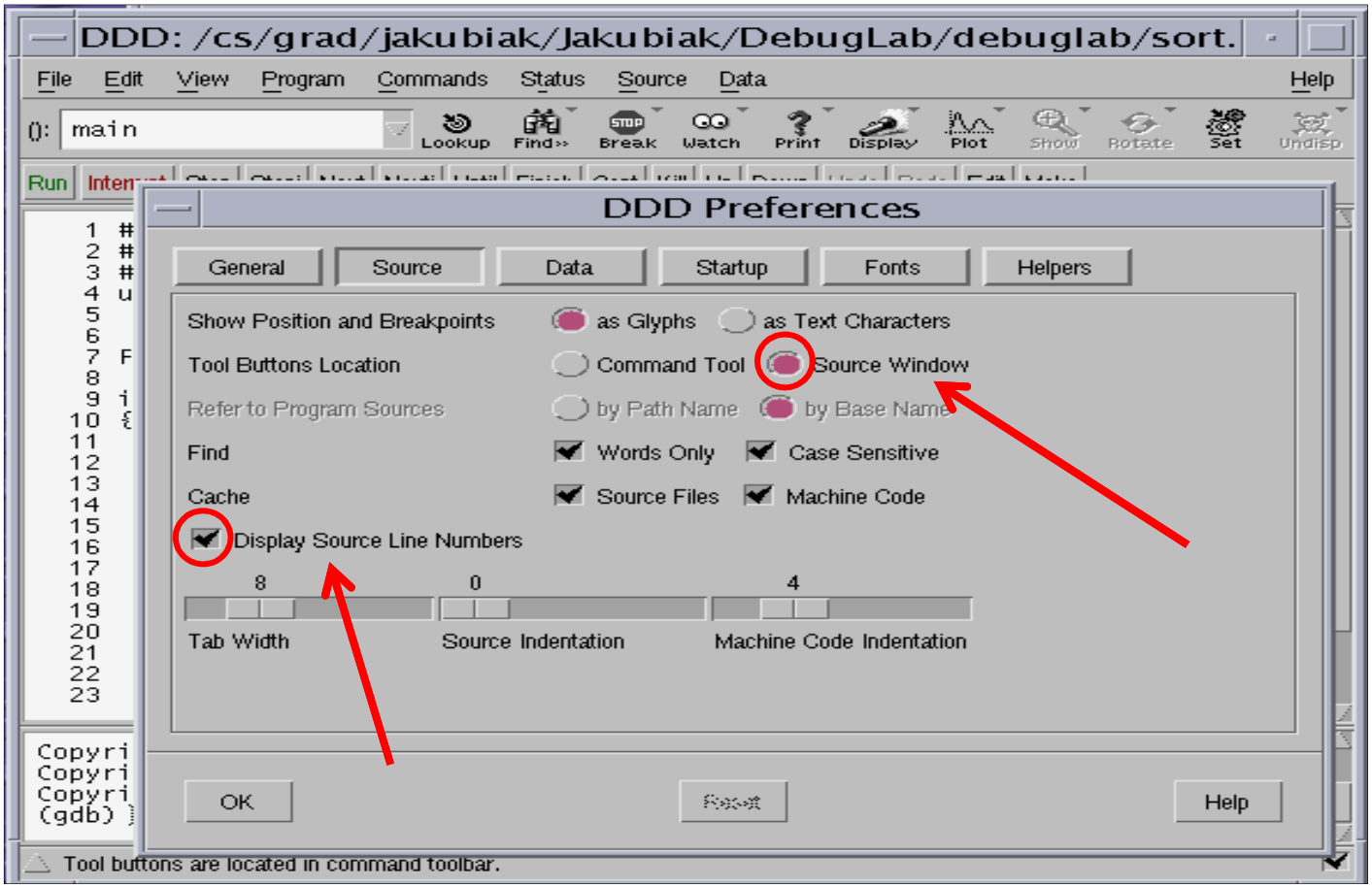
```
$ ddd es1.exe &
```

# Come appare DDD



# Customizzazione di DDD

- Numero di riga: *Source → Display Line Numbers*
- Ancoraggio della barra degli strumenti: *Edit → Preferences*



# Come impostare i punti di interruzione

- Prima di mettere in esecuzione il programma occorre impostare uno o più **punti di interruzione** (*breakpoints*), altrimenti una volta avviato andrebbe immediatamente alla fine
- Per impostare un breakpoint:
  - **click-pulsante-destro** alla linea in cui si desidera impostare l'interruzione
  - Selezione dell'opzione **Set Breakpoint**
  - (ora dovrebbe comparire un segnale di stop all'inizio di quella linea)

# Avvio del programma da debuggare

- In cima allo schermo, selezionare il bottone “Run” per avviare il programma
  - Run può anche essere trovato nel menu ‘program’
- Il programma si arresterà in corrispondenza della riga in cui è posizionato il brackpoint
  - (La linea nera indica la prossima riga che verrà eseguita)

# Next, Hover, e Step

- Usare *Next* per muoversi nel programma una riga alla volta
- Dopo ogni *Next* è possibile posizionare il cursore sopra una variabile per vederne il valore
  - Questa operazione è chiamata *hovering*
- *Step* può essere usata in alternativa a *Next*
  - *Step* fa andare il programma una riga avanti, ma nel caso di chiamata di funzioni ci fa eseguire passo passo anche la funzione
  - *Next* fa andare il programma una riga avanti. Nel caso di chiamata di funzione, salta la chiamata e ci porta alla riga successiva alla chiamata stessa.

# Correzione del programma (*Bug Fixing*)

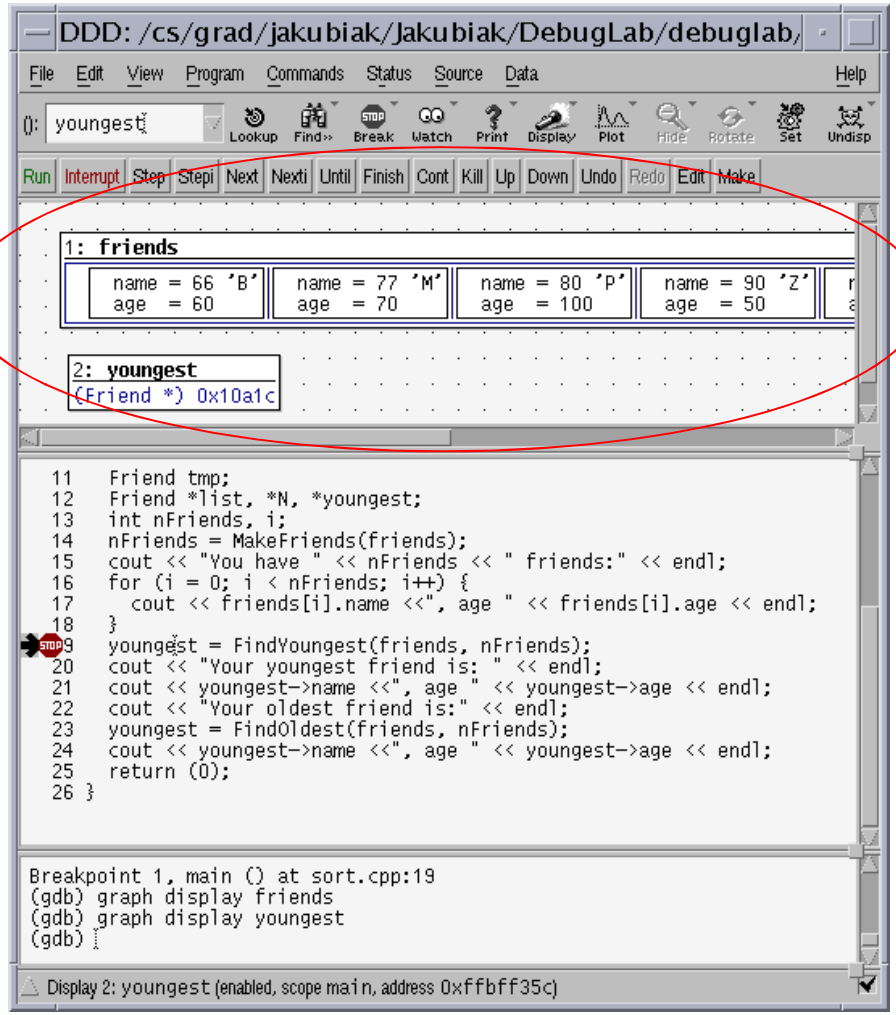
- Per correggere il programma, una volta rilevato l'errore (baco):
  - chiudere DDD
  - aprire il file sorgente usando l'editor (gedit es1.cpp)
  - correggere il programma
  - ricompilare e rilinkare con opzione -g
  - riaprire l'eseguibile con DDD

# Inspezione delle variabili (1/2)

- Fare click sul pulsante destro sopra una variabile e poi scegliere **Display *nomeVariabile***
  - questo provoca l'apertura del *display editor*
- Fare ancora click sul pulsante destro sopra al nome della variable e scegliere

  **Display *nomeVariabile**

  (*notare l'asterisco*)

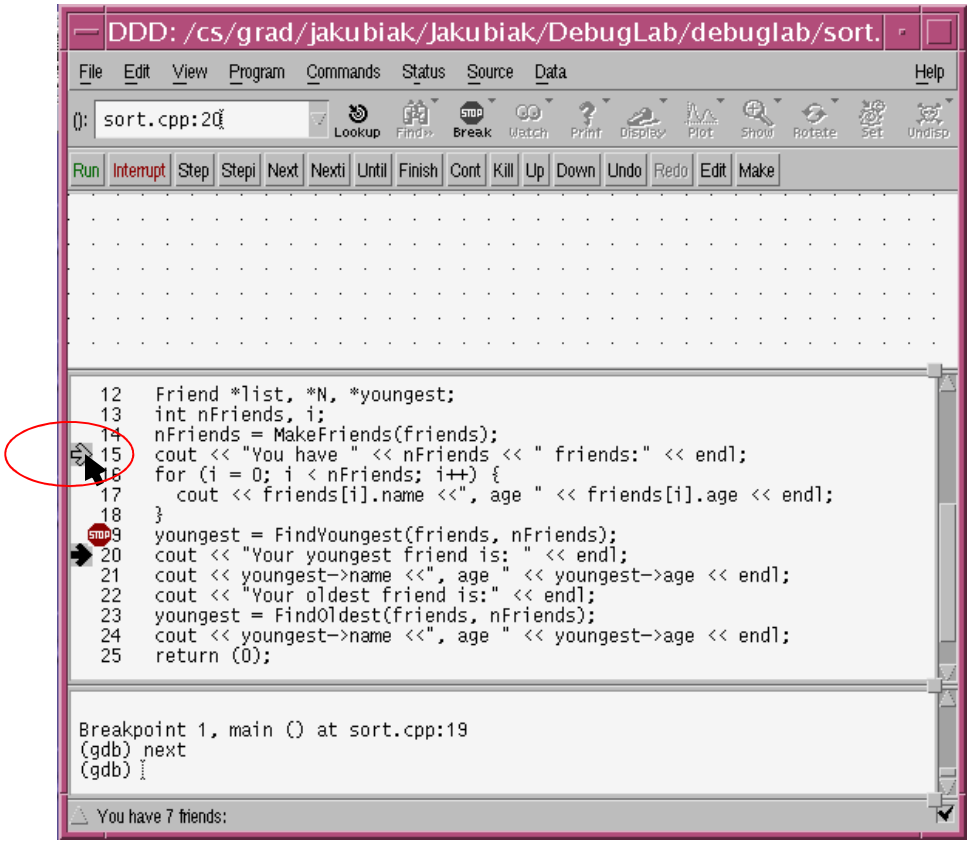


# Inspezione delle variabili (2/2)

- Le variabili possono essere visualizzate nell'area di display in diverse rappresentazioni:
  - **/t *nomeVar*** : binario
  - **/d *nomeVar*** : decimale
  - **/h *nomeVar*** : esadecimale
  - **/o *nomeVar*** : ottale
- Nel caso di programmi assembler si può visualizzare nella finestra di display (quella in alto) il contenuto dei registri come fossero variabili:
  - **/t $eax**: mostra eax in bin
  - **/d $ebx**: mostra ebx in decimale, ecc…
- Il contenuto dei registri può essere visualizzato anche nella finestra console di dello GNU debugger (GDB), nel seguente modo:
  - **i r eax** (che sta per info register eax)
- NB1: qui non serve $, né %, prima del nome del registro
- NB2: esiste una limitazione: il contenuto dei registri a 8 e 16 bit (al, ah, ax, bl, …) non può essere visualizzato

# Per tornare indietro nell'esecuzione

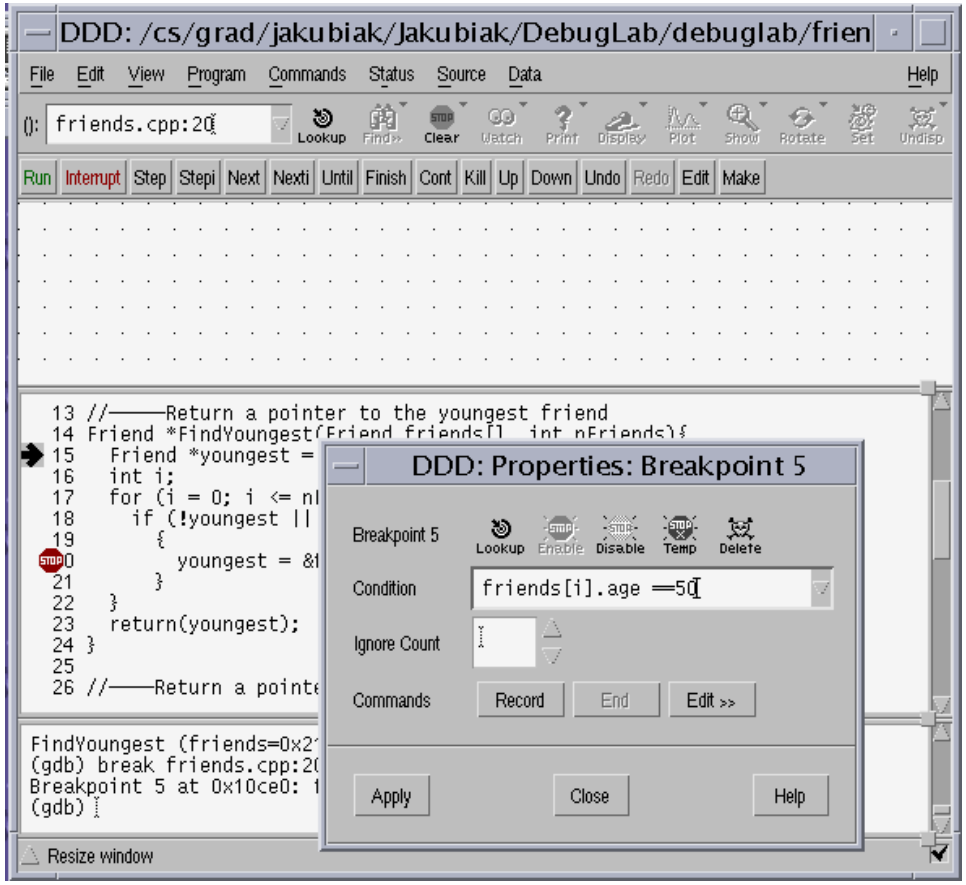
- Per andare indietro rispetto alla linea di interruzione
  - clickare e trascinare la freccia in alto, alla linea desiderata



# Materiale addizionale

# Conditional Breakpoints

- What if you had 10,000 friends? It would take a long to check each assignment.
- Instead, first make sure that the loop reaches the youngest friend.
- Use a conditional breakpoint
  - Insert a breakpoint
  - Right-click the break point and choose properties
  - Enter the condition (in C code) when, if true, the code should pause.



# Watchpoints

- Often you have a program and, at some point, an important value gets set incorrectly
  - e.g., using (x = 0) instead of (x == 0)
  - Suppose you know that
    - A line 6, x is set to 7
    - At line 9,234  x = 0 (but x should still be 7).
  - How can you quickly find the line where x was incorrectly set.
    - A watchpoint allows you to watch for a variable to change