

CSI605

Introduction to ddd

ddd

- **ddd stands for the Data Display Debugger**
- **ddd is a graphical environment that resides on top of gdb**
- **We recall that gdb is the GNU Source Level Debugger**
- **Like gdb ddd supports debugging of several languages including C, C++, JAVA, and PERL**
- **Our focus here is on C and C++ and in fact some of the features that we may mention are not fully supported in JAVA**
- **Developed in Germany**
- **These notes are current as of version 3.1.2**

On-line Resources for ddd

- `http://www.cs-tu-bs.de/softtech/ddd/`
- **ddd supports a guest book for you to provide humorous stories on there Web Page**
- **Downloadable executables**
- **Tutorials**
- **ps copy of the manual**

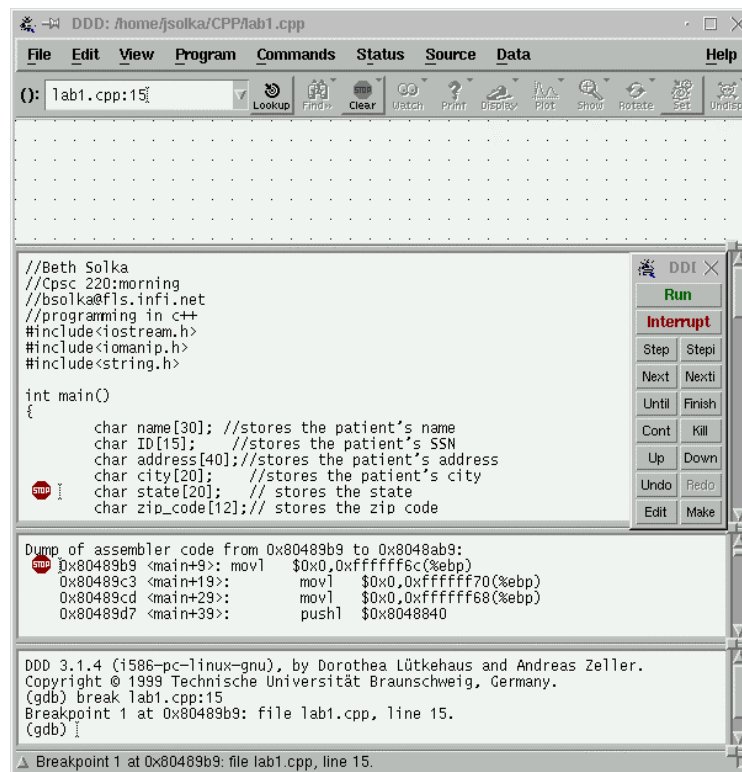
Invoking ddd

- We have to compile our program with the -g option
- ddd
 - Here we invoke just ddd
- ddd myprogram
 - Here we load our executable in
- ddd myprogram core
 - Here we load our executable and the associated core file
- ddd myprogram 666
 - Here we load our program and the associated process id

ddd and the Inferior Debugger

- **gdb is the implicit inferior debugger on our vanilla ddd launch we can launch with other explicit inferior debuggers**
- ddd --gdb myprogram
- ddd --dbx myprogram
- ddd --xdb myprogram
- ddd --jdb myclass

The ddd Main Windows



- **The Data Window** where the current state of the debugged program is shown
- **The Source Window** where the current source code of the debugged program is shown
- **The Debugger Console** accepts the debugger commands and shows debugger messages

ddd Optional Windows

- **Command Tool**
 - **Buttons for frequently used commands**
 - **Usually placed on the source window**
- **Machine Code Window**
 - **Shows the machine code**
 - **Usually placed beneath the current source**
- **Execution Window**
 - **Shows the input and output of the debugged program**

The Command Tool



Tool Bar



- **Setting arguments**
 - **Key in the arguments manually**
 - **Paste current selection into the field using mouse button 2**
 - **Clear by clicking on () label**
 - **Select an item form the source or data window**
 - **Select a previously used argument from the drop-down menu at the right of the argument field**

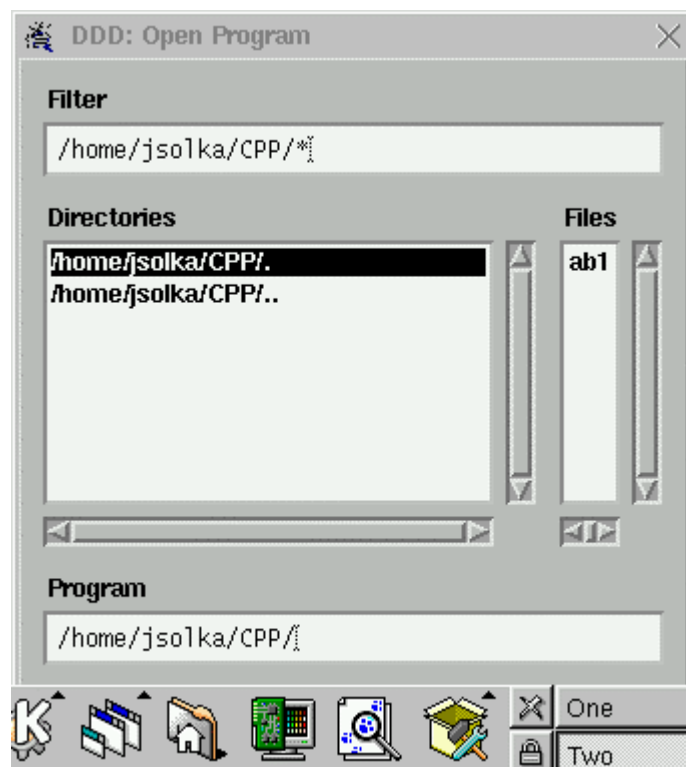
Getting Help

- **Button Tips**
 - **These work just like in MS Windows**
- **Status Line**
 - **Displays information about the currently selected item**
 - **Clicking here displays the most recently selected item**
- **Context-Sensitive Help**
 - **Help available on any visible ddd item**
 - **Point at item and press F1**
- **ddd dialogs**
 - **All of these contain help buttons**
- **Help on debugger commands**
 - **Enter help at the debugger prompt**
- **If you are stuck**
 - **Execute commands Help-->What Now?**
 - **What Now is command in the Help menu**
 - **Ctrl+F1**

Undoing Commands

- **Virtually all ddd commands can be undone**
- **Edit--->Undo**
- **Undo button**
- **Edit--->Redo**

Opening Files



- **JDB**
 - **File--->Open Class**
- **Re-open a recently debugged program**
 - **File--->Open Recent**
- **Core Dump**
 - **File--->Open Program**
 - **File--->Open Core Program**
- **Note if the program is recompiled then the source is reloaded**

Looking Up Items

- **Method1**
 - **Click with mouse button 1 on the function or variable name**
 - **Name is copied to the argument field**
 - **Click Lookup() button**
- **Method2**
 - **Press mouse button 3 on the function name and select Lookup from the source popup menu**
- **Method 3**
 - **Double-click on a function call to lookup the function definition**

Source Popup Menu

Print address

Display address

Print *address

Display *address

What is address

Lookup address

Break at address

Clear at address

Textual Search

- **Method 1**
 - Click with mouse button 1 on the item that you wish to search for in the source window ---> Click on Find>>() button to find next occurrence and Find<<() button to find the previous occurrences
- **Method2**
 - Enter the item in the argument field and click on one of the Find buttons

A Closer Look at the Source Window

The screenshot shows the DDD (Data Display Debugger) interface for the file `lab1.cpp`. The window title is `DDD: /home/jsoika/CPP/lab1.cpp`. The menu bar includes `File`, `Edit`, `View`, `Program`, `Commands`, `Status`, `Source`, `Data`, and `Help`. The toolbar contains icons for `Lookup`, `Find`, `Clear`, `Watch`, `Print`, `Display`, `Plot`, `Show`, `Rotate`, `Set`, and `Undisp`.

The source code window displays the following code:

```
char state[20]; // stores the state
char zip_code[12]; // stores the zip code
double amount_due=0; //stores the patient's bill
int choice = 0; //stores the menu choice
char enter[1]; //responce to continue

//*****Greeting Header*****//
cout << endl;
cout << "Welcome to Sister Joan Hospital Database\n";
cout << "Written by Beth Solka.\n";
cout << "This database will store patient information.\n";
cout << "It will bill the patient and print out information.\n";
cout << "Press enter to continue.\n";
cin.getline(enter,1,'\n');
//*****Menu Header*****//
```

The assembly window shows the following instructions:

```
0x8048a40 <main+144>:  addl  $0x8,%esp
0x8048a43 <main+147>:  pushl $0xa
0x8048a45 <main+149>:  pushl $0x1
0x8048a47 <main+151>:  leal  0xfffff67(%ebp),%eax
0x8048a4d <main+157>:  pushl %eax
0x8048a4e <main+158>:  pushl $0x804a228
0x8048a53 <main+163>:  call  0x80488b0 <istream::getline(char *, int,
```

The breakpoint window shows:

```
Breakpoint 2, main () at lab1.cpp:28
(gdb) next
Press enter to continue.
(gdb) !
```

The status bar at the bottom indicates: `▲ Press enter to continue.`

Breakpoints

- **This is one of the most important features of debugging**
- **It of course allows us to stop execution of a program at a particular location**
- **We will discuss numerous ways to set these breakpoints**

Setting Breakpoints by Location

- **Method 1**
 - Click with mouse button 1 on the left of the source line and then on the Break at () button
- **Method 2**
 - Press mouse button 3 on the left of the source line and select the Set Breakpoint item from the popup menu
- **Method 3**
 - Double-click on the left of the source line to set a breakpoint
- **Method 4**
 - Select Source--->Edit Breakpoint
 - Click on the Break button and enter the location

A Closer Look at the Line Popup Menu



- **Breakpoints are indicated as a plain stop sign or as #n#**
- **A grayed out stop sign or _n_ indicates a disabled breakpoint**
- **?n? indicates a conditional breakpoint or a breakpoint with an ignore count set**

Bugtraq on Stop Signs

- **Some Motif versions fail to display the stop signs correctly**
 - **Edit \$HOME/.ddd/init**
 - **Insert**
 - **Ddd*cacheGlyphImages:off**
 - **Restart ddd**

Setting Breakpoints By Name

- **Method 1**
 - Click with mouse button 1 on the function name
 - Function name is copied to the argument field
 - Click on the Break at () button to set the breakpoint here
- **Method 2**
 - Press mouse button 3 on the function name and select break
- **Method 3**
 - Click New from the Breakpoint editor
 - Invoked from Source --->Breakpoints
 - Enter the function name

Setting Regexp Breakpoints

- This allows us to set breakpoints on all functions that match a given string
 - Break at () --->Set Breakpoints at Regexp()
 - Sets breakpoint on all functions whose name matches the regular expression given in ()
- **Examples**
 - To break at functions that starts with Xm
 - Set () to ^Xm
 - To break on every member of class Date
 - Set () to ^Date::
 - To break on every function whose name contains _fun
 - Set () to _fun
 - To break on every function that ends in _test
 - Set () to _test\$

Disabling Breakpoints

- **Method 1**
 - Press mouse button 3 on the breakpoint symbol and select Disable Breakpoint from the breakpoint popup menu. To enable it again, select Enable Breakpoint
- **Method 2**
 - Select breakpoint
 - Click on Disable or Enable in the Breakpoint editor
 - Invoked through Source--->Edit Breakpoints
- The Disable Breakpoint item is also accessible via the Clear at () button
 - Press and hold mouse button 1 on the button to get a popup menu

Temporary Breakpoints

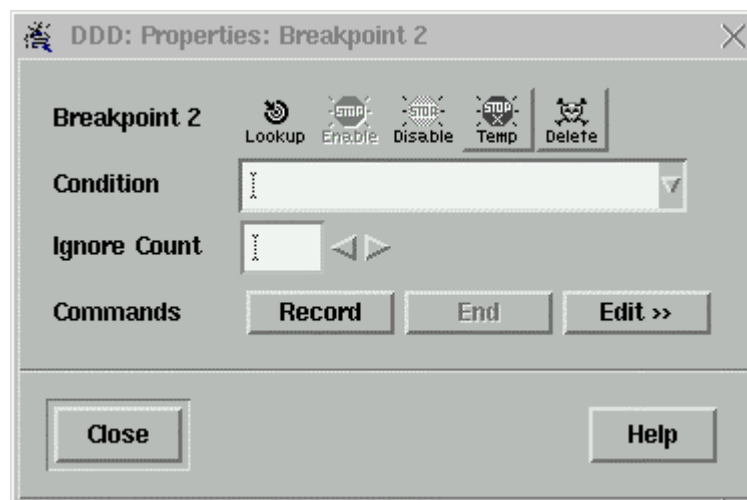
- A temporary breakpoint is immediately deleted as soon as it is reached
- Setting temporary breakpoints
 - Method 1
 - Press mouse button 3 on the left of the source line and select Set Temporary Breakpoint item from the popup menu
 - Method 2
 - Double click to the left of the source line while holding Ctrl
- The Continue Until Here item from the popup menu sets a temporary breakpoint on the left of the source line and immediately continues execution. Execution stops when the temporary breakpoint is reached.
- The Set Temporary Breakpoint and Continue Until Here items are also accessible via the Break at () button

Deleting Breakpoints

- **Method 1**
 - Click with mouse button 1 on the breakpoint
 - The breakpoint location is copied to the argument field
 - Click on the Clear at () button
- **Method 2**
 - Click with mouse button 1 on the function name.
 - The function name is copied to the argument field.
 - Click on the Clear at () button
- **Method 3**
 - Press mouse button 3 on the breakpoint
 - Select Delete Break point from the popup menu
- **Method 4**
 - Select the breakpoint
 - Click on the Delete in the Breakpoint editor
 - Invoked through Source--->Edit Breakpoints
- **Method 5**
 - Double-click on the breakpoint while holding Ctrl

Editing Breakpoint Properties

- **Press mouse button 3 on the breakpoint symbol**
 - **Select properties from the breakpoint popup menu**
- **Alternatively double click on the breakpoint**



Some Important Breakpoint Properties

- **Breakpoint Conditions**
 - Set in the field Condition of the Breakpoint Properties panel
 - Breakpoint executes if stated condition is true
- **Breakpoint Ignore Counts**
 - Set the field Ignore Count in the Breakpoint Properties panel
 - Give an value N then next N crossings of the breakpoint will be ignored
 - Each pass decrements the ignore count
 - ignore count = 0 causes the program to stop
- **Breakpoint Commands**
 - Available only under gdb
 - Uses Commands buttons of the Breakpoint Properties panel
 - One may specify a sequence of commands to be executed when the breakpoint is hit

Recording a Command Sequence

- Click on Record
- Interact with ddd (commands are recorded not executed)
- To stop recording, click on End or enter end at the GDB prompt
- Recording may be canceled by clicking on Interrupt or Esc
- Edit >> edits the recorded commands
- Edit << cancels the editing process

Watchpoints

- **These cause the program to stop as soon as some variable changes, or when some variable is read or written**
- **Execution can be two orders of magnitude slower on those architectures without special watchpoint support**
- **Virtually all of the options previously discussed for breakpoints are available for watchpoints**

Running the Program

- **Select Program-->Run**
 - **Select from list of previously supplied arguments or type in your own arguments**
 - **Press Run**
- **To reexecute program again with the same arguments type**
 - **Program-->Run Again**
- **Program-->Run in Execution Window**
 - **By default input and output of your program go to the debugger console**
 - **The above command causes the input output to go to an execution window**

Attaching to a Running Process

- We can actually begin debugging by attaching to a running process
- To attach the process
 - File-->Attach to Process
 - Attach
 - One typically wants to use Open Program to specify the program running in the process and load its symbol table
- To detach the process
 - File-->Detach Process

Resuming Execution

- Click the Continue button
 - Resumes execution where the program last stopped
- Click the Step button
 - Execution continues until we reach a new source line
 - Can lead to a step into a new function
- Click the Next button
 - Continue to the next line in the current function
 - Similar to step but any function calls appearing within the line of code are executed without stopping
- Click on the Until Button
 - Continue until a greater line in the current function is reached
 - Useful to avoid single stepping through a loop more than once
- Click on the Finish button
 - Continues running until the current function returns

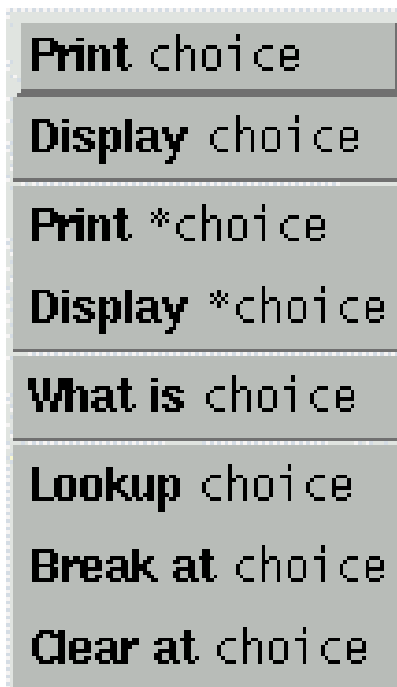
Examining the Stack

- **This allows one to determine where a program stopped and how it got there.**
- **Select Status-->Backtrace**
 - **Up button selects the function that called the current one**
 - **Down button selects the function that was called by the current one**

Examining Data

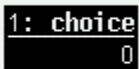
- **Value Hints**
 - Basically like a tool tip except the values displayed are the data values
 - Good for simple data types
- **Printing Values**
 - Allows for reuse of printed values
 - Good for larger data structures
- **Displaying Values**
 - Allows the display of complex data structures
 - These remain permanent until dismissal and they are undated
 - Good for large dynamic structures
- **Plotting Values**
 - Displays arrays of data values graphically
- **Memory Dumps**
 - Available using gdb only
 - Allows one to dump memory content in several formats

Printing Simple Values in the Debugger Console



- Method 1
 - Click mouse button 1 on its name
 - Variable name is copied to the argument field
 - Click the Print() button
- Method 2
 - Press mouse button 3 on the variable name and select Print

Displaying Complex Values in the Data Window



1: choice
0

- **Method 1**
 - **Click mouse button 1 on the name of the variable**
 - **Variable name is copied to the argument field**
 - **Click on the Display() button**

Showing and Hiding Values

- **Aggregate values such as structures, records, arrays and classes can be show hidden (as {...}) or expanded**
- **Showing Details**
 - **Select the aggregate by clicking mouse button 1 on its name or value and click on the Show() button**
- **Hiding Details**
 - **Select the aggregate by clicking mouse button 1 on its name or value and then click on the Hide() button**

Rotating Arrays

- **Arrays can be displayed with vertical or horizontal alignment**
- **We may change the alignment**
- **Method 1**
 - **Select the array**
 - **Click on the Rotate button**
- **Method 2**
 - **Press mouse button 3 on the array**
 - **Select the Rotate menu item**



Dereferencing Pointers



- Note that this menu provides us with the capability to dereference a pointer
- Method 1
 - Select the pointer
 - Click on the Display*() button
- Method 2
 - Press mouse button 3 on the original pointer
 - Select the Display * menu item

Displaying Multiple Array Values and Altering Variable Values

- **ddd provides us with a natural mechanism to display multiple array values**
 - **graph display argv[0..9]**
- **We may change a variable value within ddd**
 - **Use the Set() or the Set Value menu item in the data popup menu**
 - **One clicks on the Ok or Apply button to commit your change**



Plotting Arrays

- **ddd provides the capability to plot 1-d and 2-d arrays**
- **1-d arrays**
 - **These are plotted as array values vs. the data index.**
 - **Steps in Plotting**
 - **Select it by clicking mouse button 1 on an occurrence**
 - **Array name is copied to the argument field**
 - **Click the Plot button**
- **2-d arrays**
 - **These are plotted as array values vs. the double data index**
 - **Steps in Plotting**
 - **Select it by clicking mouse button 1 on an occurrence**
 - **Array name is copied to the argument field**
 - **Click the Plot button**

Examining Memory

- Choose Data--->Examine Memory
- This launches a pop-up window that allows one to display a plethora of different parameters
- Please see the ddd manual to fully explore these

Some Other ddd Features

- **Thread Examination**
- **Program Execution “Undoing”**
- **Signal Handling**