

# **CSI605**

**Introduction to gdb**

# Compiling for gdb Execution

- I often compile as follows  
`gcc -g -Wall -omyprog myprog.c`  
`g++ -g -Wall -omyprog myprog.c`
- It suffices to use  
`gdb -g -omyprogram myprogram.c`
- We note that `-g` and `-O` are incompatible

# Launching gdb

- We launch via

**gdb program**

**gdb program core-dump**

```
[jsolka@ginseng ~/CPP]$ gdb lab1
```

```
GNU gdb 4.17.0.11 with Linux support
```

```
Copyright 1998 Free Software Foundation, Inc.
```

```
GDB is free software, covered by the GNU General Public License, and you are  
welcome to change it and/or distribute copies of it under certain conditions.
```

```
Type "show copying" to see the conditions.
```

```
There is absolutely no warranty for GDB. Type "show warranty" for details.
```

```
This GDB was configured as "i386-redhat-linux"...
```

- Note that the core file may be inserted later with the **core** command
- **gdb launch options**
  - **-d dir** (tells gdb where to look for the source file)
  - **-x file** (tells gdb to execute the commands in file during startup, can include multiple files)
  - **-nx** (don't execute commands from the initialization file (normally .gdbinit))
  - **-q** (don't print the introductory messages during startup)
  - **-help** (print a message showing all of the command line options and then quit)
  - **-batch** (run in batch mode; execute any startup files (unless they are suppressed), followed by any files specified with the -x options, and then exit with status 0)

# Basic gdb Commands

- **gdb like virtually all programs supports a set of internal help files**
- **help aliases**
- **help breakpoints**
- **help data**
- **help files**
- **help internals**
- **help obscure**
- **help running**
- **help stack**
- **help status**
- **help support**
- **help users**

# Listing a File

- We use the command list command

- **listing a file**

```
gdb lab4
```

```
GNU gdb 4.18
```

```
Copyright 1998 Free Software Foundation, Inc.
```

```
GDB is free software, covered by the GNU General Public License, and  
you are
```

```
welcome to change it and/or distribute copies of it under certain  
conditions.
```

```
Type "show copying" to see the conditions.
```

```
There is absolutely no warranty for GDB. Type "show warranty" for  
details.
```

```
This GDB was configured as "i386-mandrake-linux"...
```

```
(gdb) list
```

```
30 void print_information (char name[], char ID[],char  
address[], char city[],char state[], char zip_code[],double  
total_balance);
```

```
31
```

```
32 void exit(void);
```

```
33
```

```
34
```

```
35 int main()
```

```
36 {
```

```
37 //Variable Declarations
```

```
38
```

```
39 double total_balance=0; //
```

- **listing by line numbers**

- list 1,30

- **listing a routine**

- list myroutine

- **listing quirks**

- **gdb lists 10 lines each time list is called**
- **gdb can't take illustrate conditional compilation**
  - they will appear verbatim

- **Non existent function file message**

Function name not defined

# Executing a Program

- We execute our program within the gdb environment using the run command
- Here is a typical session of running a program called myprogram with an input file of input values and an output file of outputfile

gdb myprogram

(gdb) run < inputvalues > outputfile

Program exited with code n.

(gdb)

- Repeating the run command directs gdb to repeat the run with the same parameters
  - set args (allows us to change the args command)
  - show args (allows us to show the current argument settings)
- To terminate a running program one executes CTRL-C
  - This returns control to the gdb environment
  - quit quits the gdb environment
- backtrace is used to backtrap the steps in an abnormally terminated program

# Printing Data

- **`gdb` provides us with a rich set of printing capabilities**
  - We effectively can print any value that is valid in the programming language that we are debugging in
  - function calls
  - data structures
  - value history
  - artificial arrays
- **`whatis x`**
  - This tells us what type `x` is
- **`print x`**
  - This prints the variable `x`
- If `p` is a pointer it makes sense to execute the following to see the value that `p` points to
  - **`print *p`**

# Some More Esoteric Prints

- When we print something we may see a \$1 or \$2 or \$3 next to the outputted value
  - We may identify these values in future expressions using these identifiers
    - print \$1 - 1
- Examining arrays
  - To examine the next 10 entries after h execute
    - print h@10
  - Given an identifier on this array of \$14 we may then examine its sixth entry with
    - print \$14[5]
- Printing Pretty
  - Sometimes vertical columns may be preferable
    - This are obtainable with
      - set print array

# Breakpoints

- **This allows one to temporarily stop execution of the program to allow examination or modification of the variables**
- **Some of the ways that it can be called include**
  - **break line-number**
    - **Stop the program just before executing the given line.**
  - **break function-name**
    - **Stop the program just before entering the named function.**
  - **break line-or-function if condition**
    - **If the following condition is true, stop the program when it reaches the given line or function.**
  - **break routine-name**
    - **Set a breakpoint at the entrance to the specified routine. When the program is executing gdb temporarily halts the program at the first executable line of the given function.**

# More About Breakpoints

- **Programs built from multiple source files**
  - **break filename:line-number**
  - **break filename:function-name**
- **Conditional break points**
  - **break line-or-function if expr**
    - **Good for stopping in the middle of loops**
- **watch points (like conditional breakpoints)**
  - **watch x <500**

# More About Watch Points

- **On virtually all workstations watchpoints slow execution down by a factor of around 100**
- **Here is an effective strategy for watchpoint use**
  - **Use regular breakpoints to get the program as close as possible to the point of failure**
  - **Set a watchpoint**
  - **Let the program continue execution with the continue command**
  - **Let your program run overnight**

# Continuing to Run After a Breakpoint

- We continue execution after a breakpoint with the **continue** command
- Program execution will continue until another breakpoint or error is hit
- One can abbreviate continue as **c**
- One can tell gdb to continue execution through a number of breakpoints using **c n**
  - **c 3**

# Managing Breakpoints

- **info breakpoints**
  - Shows us what breakpoints are currently active
- **delete #**
  - This deletes the breakpoint whose identification number is given by #
- **clear # (clear function name)**
  - Deletes all breakpoints on line # of the code or deletes all breakpoints at the entry to function\_name

# Enabling and Disabling Breakpoints

- **disable #**
  - **disable breakpoint identified by #**
- **enable #**
  - **enable breakpoint identified by #**
- **enable once #**
  - **enable breakpoint identified by # only once**

# Examining and Modifying Variable Values

- **whatis**
  - Identifies the type of a variable or array
- **set variable**
  - Assigns a value to a variable
- **print**
  - This is used for assigning and printing values
  - `print myfunction(x,y,z)` works

# Example of Examination and Modification -I

```
[jsolka@ginseng ~/CPP]$ gdb lab1
```

```
GNU gdb 4.17.0.11 with Linux support
```

```
Copyright 1998 Free Software Foundation, Inc.
```

```
GDB is free software, covered by the GNU General Public License, and you are  
welcome to change it and/or distribute copies of it under certain conditions.
```

```
Type "show copying" to see the conditions.
```

```
There is absolutely no warranty for GDB. Type "show warranty" for details.
```

```
This GDB was configured as "i386-redhat-linux"...
```

```
(gdb) break 44
```

```
Breakpoint 1 at 0x8048b07: file lab1.cpp, line 44.
```

```
(gdb) run
```

```
Starting program: /home/jsolka/CPP/lab1
```

```
Welcome to Sister Joan Hospital Database
```

```
Written by Beth Solka.
```

```
This database will store patient information.
```

```
It will bill the patient and print out information.
```

```
Press enter to continue.
```

```
Menu Screen.
```

```
1. Enter Patient Information.
```

```
2. Bill Patient.
```

```
3. Print patient Information.
```

```
4. Exit.
```

```
Enter choice:
```

```
1
```



# Single-Step Execution

- next
  - Takes a single step but executes an entire function if it encounters a call
- step
  - Takes a single step

# Function Debugging

- **call function name**
  - Call and execute a function
  - Will not print the function return value if it is void
- **finish**
  - Finish executing the current function and print its return value if any.
- **return value**
  - Cancel execution of the current function, returning value to the caller.

# Automatic Execution of Commands

- Here is the syntax of this type of beast  
commands #  
... list\_of\_commands  
... list\_of\_commands  
end
- # is the breakpoint number that we would like to link our commands to
- ex.  
commands 23  
echo value of x \n  
print x  
end

# Convenience Variables

- **gdb lets you define your own local variables**
- **They are typically of the form \$name**
- **They are assigned using  
set variable = expression**
- **They can be initialized and printed at the same time using  
print  
print \$x=12**

# Working with Source Files in gdb

- **search text**
  - Searches forward for the next occurrence of text
- **reverse-search text**
  - Searches backwards for the next occurrence of text
- We may add directories **directory1** and **directory2** to the gdb directory search path at launch time using  
**gdb -d directory1 -d directory2 myprogram**
- After launch the directory search path may be modified to add the directory **directory1** using  
**directory directory1**
- To revert the search path back to the default, execute]  
**directory**

# User Defined Command Sequences

## Using the define Command

- Here is the standard format of a user defined command sequence using define

```
define my_command  
... command1 ...  
... command2 ...  
... command3 ...  
end
```
- One can also use hooks and script files

# Execution of UNIX Commands Within gdb

- To execute a UNIX command within gdb execute

shell command name

- For example

shell pwd

- \* There is also a nice relationship between gdb and emacs \*

# Command Abbreviations in gdb

- Here are most of the gdb commands along with there associated abbreviations

<b>break</b>	<b>b</b>
<b>continue</b>	<b>c</b>
<b>delete</b>	<b>d</b>
<b>frame</b>	<b>f</b>
<b>help</b>	<b>h</b>
<b>info</b>	<b>i</b>
<b>list</b>	<b>l</b>
<b>next</b>	<b>n</b>
<b>print</b>	<b>p</b>
<b>quit</b>	<b>q</b>
<b>run</b>	<b>r</b>
<b>step</b>	<b>s</b>

# Attaching to an Existing Process

- **gdb allows one to attach to a program that is already running**
- **gdb my\_program my\_program pid**
- **Note that gdb will initially inform you that there is no file by the name of your process id. This is just gdb checking to see if there is a core file available.**
- **Alternatively one can start gdb and then use the attach command to attach to a running process**
- **The process is then restarted with detach**

# **gdb Flash Card**

- **backtrace** Prints location and stack trace.
- **breakpoint** Sets breakpoint.
- **cd**
- **clear** Delete the breakpoint where you just stopped.
- **commands** Provide commands to be executed at breakpoint
- **delete** Delete breakpoint or watchpoint.
- **display** Display variables or expressions at program stop.
- **down** Move down the stack frame.
- **frame** Select a frame for the next continue.
- **info** Display program information.
- **jump** Start execution at another point in the program.
- **kill** Abort the process under gdb's control.
- **list** List the source file for the executing process.
- **next** Execute the next source line executing a function in its entirety.
- **print** Print the value of a variable or an expression.
- **pwd**
- **ptype** Show the contents of a data type.
- **quit** Exit gdb.
- **reverse-search** Search backwards.
- **run** Execute the program.
- **search** Search forwards.
- **set variable** Assign a value to a variable.
- **signal** Send a signal to the running process.
- **step** Execute the next source line stepping into the function if necessary.
- **undisplay** Reverse a display command.
- **until** Finish the current loop.
- **up** Move up the stack frame to make another function the current one.
- **watch** Set a watchpoint in the program.
- **whatis** Print the type of a variable or function.