

Debugging


Paul Russo

art/ LArSoft course

August 6, 2015



Fermi National Accelerator Laboratory

 Office of Science / U.S. Department of Energy

Managed by Fermi Research Alliance, LLC

Introduction

Introduction

We will be learning the basics of using the **gdb** debugger to find and fix errors in code.

A debugger cooperates with the operating system to run your program under manual control.

You can:

- Stop execution at any time.
- Examine and change the values of variables.
- Display source code.
- Run the code one line at a time.

The Lectures and Exercises

- The text of this lecture is available in the same directory as the exercises.
- The exercises come in broken and fixed versions, and some have reference output for the properly working version.
- There is a gdb reference card included in the materials.

Debugging Symbols

- In order for the debugger to show you source code and to be able to associate variable names with the memory locations in use by your program the compiler must help the debugger by writing a detailed description of your code to the object file during compilation. This detailed description is commonly referred to as debugging symbols.

Dealing with Compiler Optimization

- Compilers normally rearrange your code during compilation to increase the execution speed, this is called optimization.
- This optimization step makes things very difficult for the debugger because it cannot match the source lines to the compiled code anymore.
- To prevent this it is necessary when compiling your code for debugging purposes to disable the compiler optimizations.
- If you are using buildtool it will do this for you when making a debug build.

Debugging with buildtool

When creating a build area for a project using buildtool you request that the build area be setup for debugging using the "-d" flag to the `setup_for_development` script:

```
$ mkdir <MyBuildArea>  
$ cd <MyBuildArea>  
$ source <MyProject>/ups/setup_for_development -d
```

Basics

Running gdb

- `run` – Start the program running.
- `start` – Start the program and stop at `main()`.
- `next` – Run one line of code, single-stepping.

Basics

- `print` – Displaying variable values.
- `whatis` – Displaying the type of a variable.
- `ptype` – Dumping a type declaration.
- `frame` – Show the current stack frame and line of code.
- `list` – List source code.

Function Calls

- **step** – Step into a function call.
- **backtrace** – Display the function call stack.
- **frame** – Switching stack frames.
- **up** – Up one stack frame.
- **down** – Down one stack frame.

Breakpoints and Watchpoints

- `break` – Stop running at a line of code.
- `info break` – Displaying breakpoints.
- `continue` – Continue running code after a break.
- `delete` – Deleting breakpoints.

Exceptions

- catch throw – Stopping at any exception throw.
- catch catch – Stopping at any exception catch.