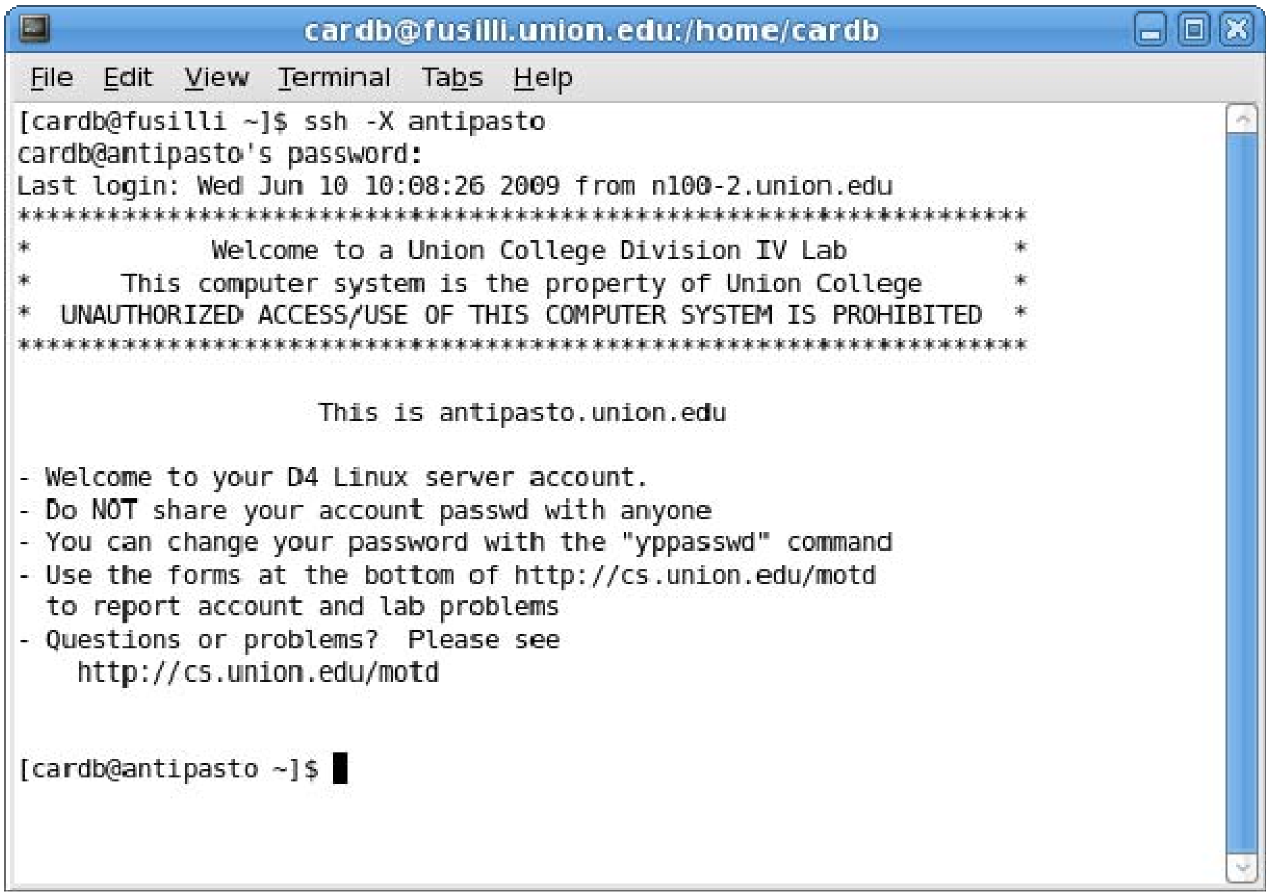
# A Quick Guide to using GBD + Emacs

Brian Card
CSC 335 – Spring 2009
6/11/09

Developing C++ programs through the terminal can be a painful experience if you are used to all of the development tools that Eclipse provides for Java programming. The closest thing I could find to a real development environment was using Emacs, which can be used in a graphical and command line forms. Of course, as true with Linux in general, the worst thing about Emacs is learning Emacs, so I recommend that you do the (very helpful) tutorial if you don't know how to use it. It will take a good 3 or so hours to complete, but it is well worth it because you will be lost without it.

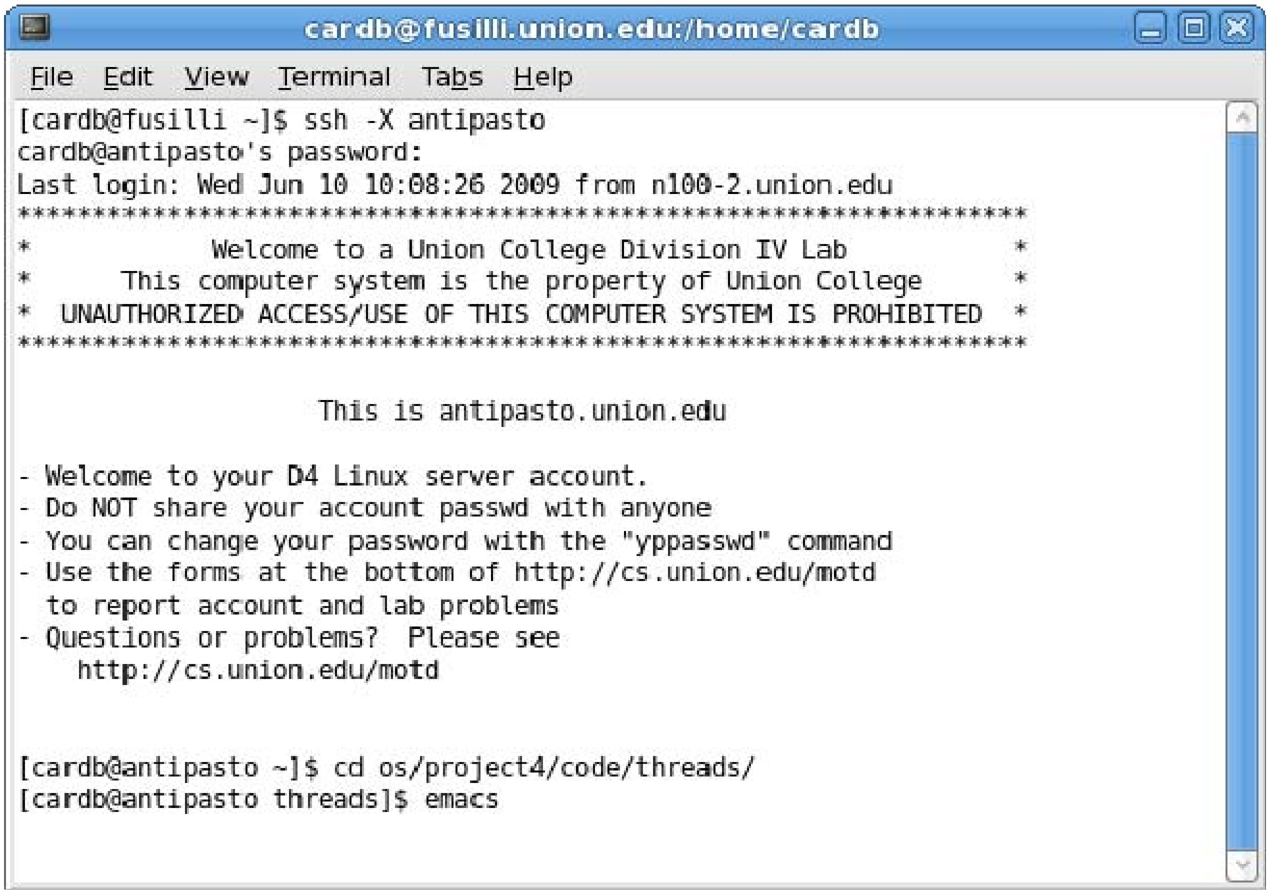
That being said Emacs integrates nicely with GDB, which is a command line debugger. Through Emacs you can set breakpoints in your code by clicking on a line in the source code, and even step through the source code while debugging. While primitive compared to the Eclipse debugger, its the best I could find. I have set up a tutorial to using this feature with the nachos project. It's a little picky, but if you stay close to this guide it should work.

1. ssh to antipasto using the -X option. This will allow you to use the graphical Emacs instead of the command line one. This will only work if you are running Linux.


cardb@fusilli.union.edu:/home/cardb
File Edit View Terminal Tabs Help

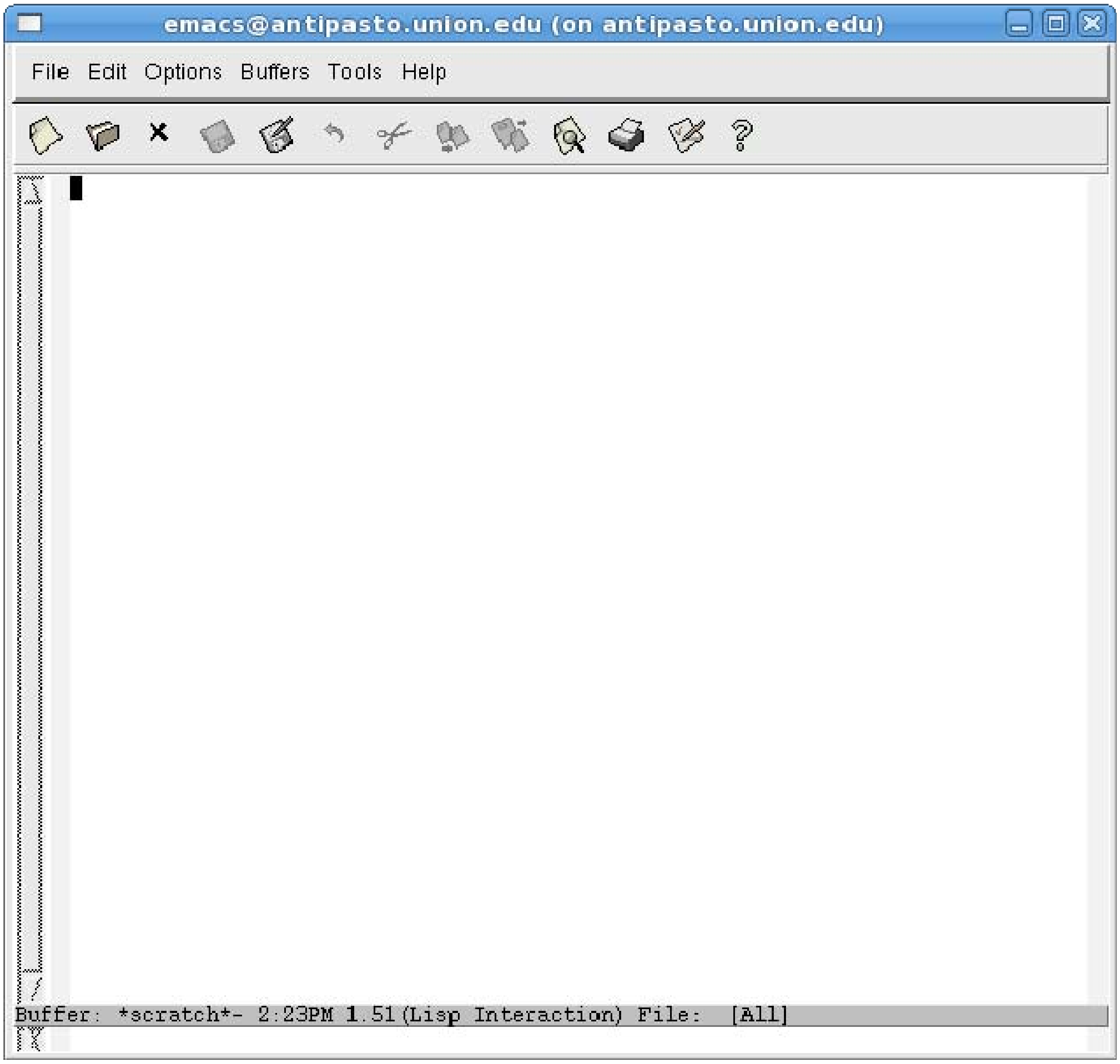
```
[cardb@fusilli ~]$ ssh -X antipasto
cardb@antipasto's password:
Last login: Wed Jun 10 10:08:26 2009 from n100-2.union.edu
**********************************************************************
*              Welcome to a Union College Division IV Lab            *
*       This computer system is the property of Union College        *
*  UNAUTHORIZED ACCESS/USE OF THIS COMPUTER SYSTEM IS PROHIBITED     *
**********************************************************************

                     This is antipasto.union.edu

- Welcome to your D4 Linux server account.
- Do NOT share your account passwd with anyone
- You can change your password with the "yppasswd" command
- Use the forms at the bottom of http://cs.union.edu/motd
  to report account and lab problems
- Questions or problems?  Please see
    http://cs.union.edu/motd


[cardb@antipasto ~]$
```

2. Navigate to the folder that you will be working in and open Emacs by typing emacs.


cardb@fusilli.union.edu:/home/cardb
File Edit View Terminal Tabs Help

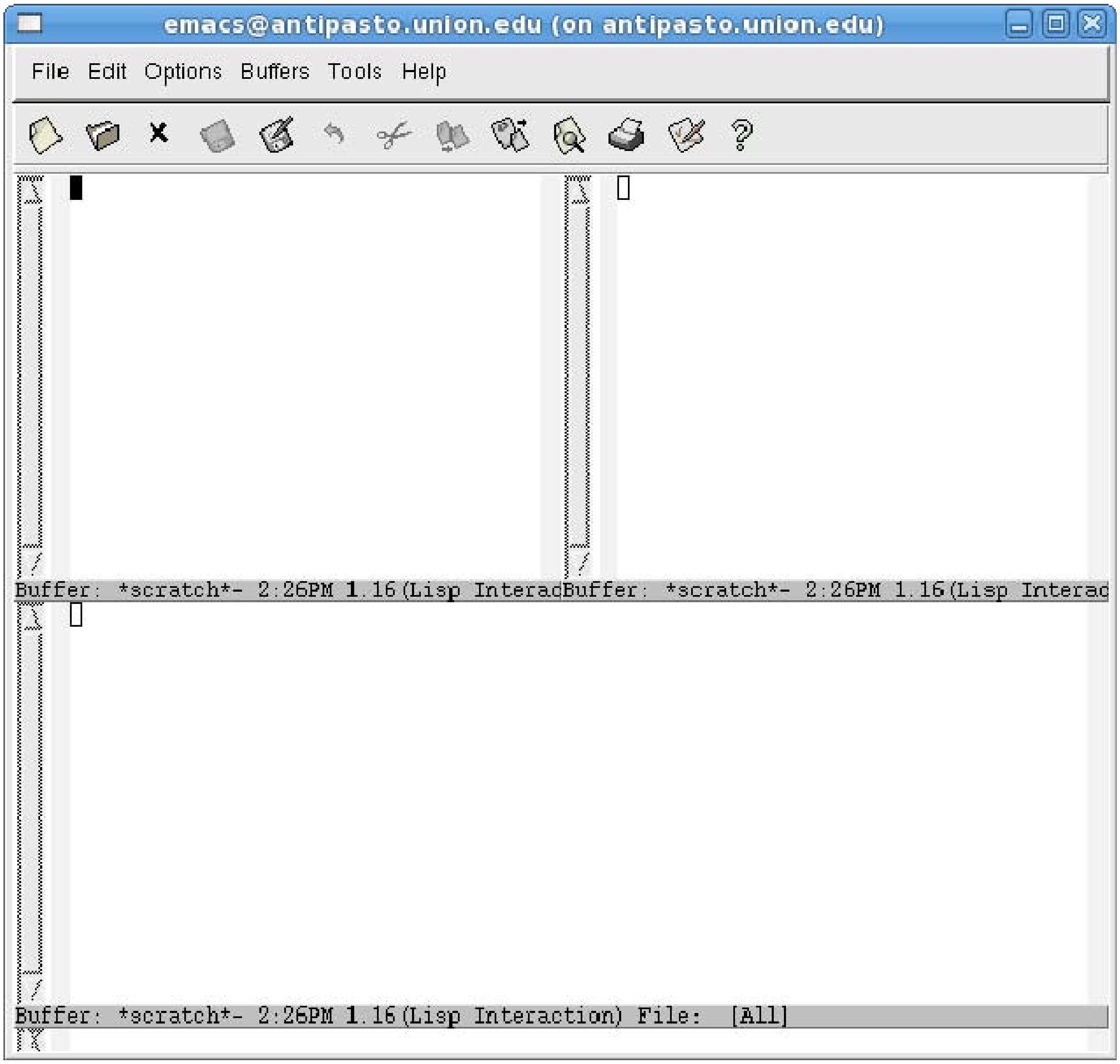
```
[cardb@fusilli ~]$ ssh -X antipasto
cardb@antipasto's password:
Last login: Wed Jun 10 10:08:26 2009 from n100-2.union.edu
*******************************************************************
*              Welcome to a Union College Division IV Lab         *
*        This computer system is the property of Union College    *
*  UNAUTHORIZED ACCESS/USE OF THIS COMPUTER SYSTEM IS PROHIBITED  *
*******************************************************************

                    This is antipasto.union.edu

- Welcome to your D4 Linux server account.
- Do NOT share your account passwd with anyone
- You can change your password with the "yppasswd" command
- Use the forms at the bottom of http://cs.union.edu/motd
  to report account and lab problems
- Questions or problems?  Please see
    http://cs.union.edu/motd


[cardb@antipasto ~]$ cd os/project4/code/threads/
[cardb@antipasto threads]$ emacs
```

This window will appear:


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Help

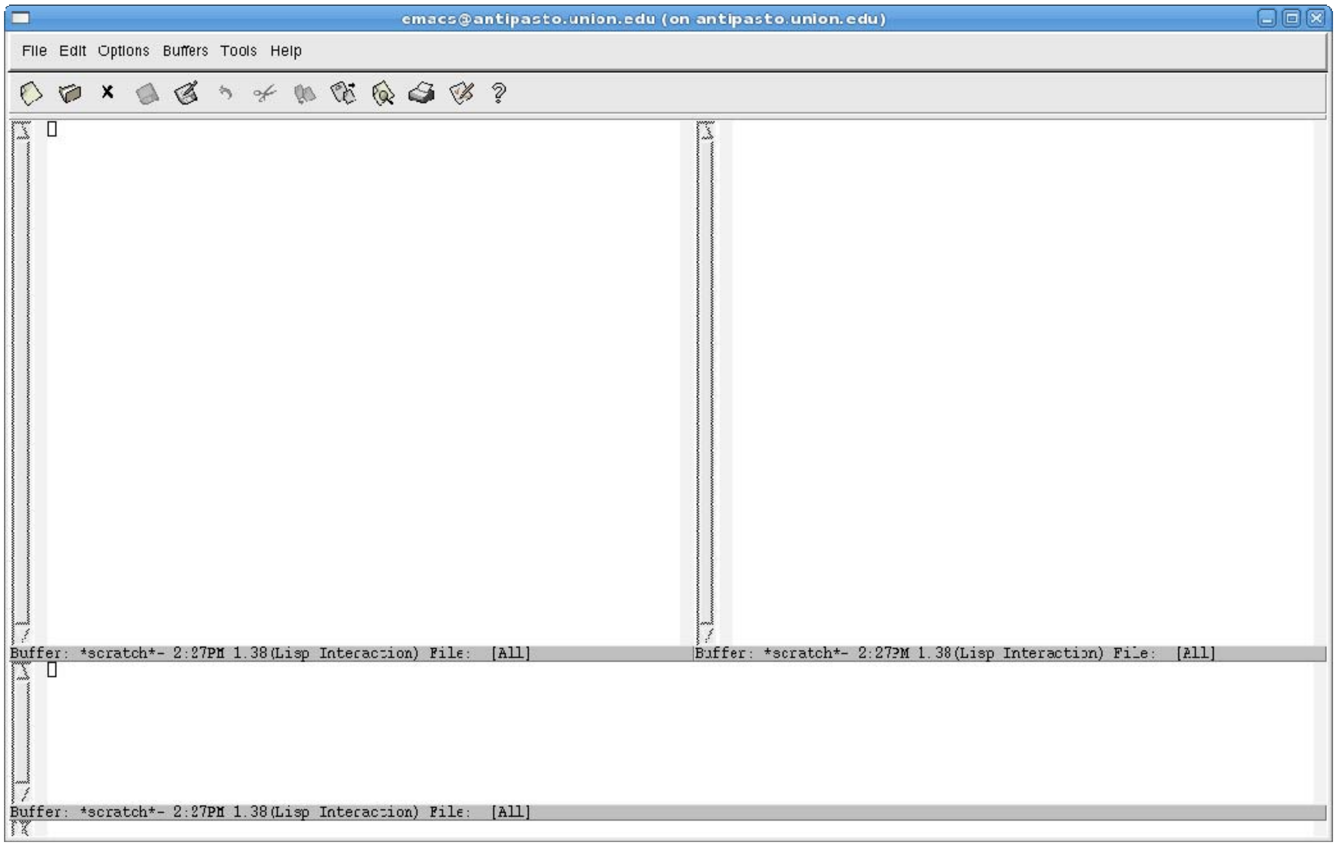
```
Buffer: *scratch*- 2:23PM 1.51 (Lisp Interaction) File:  [All]
```

I like to have two vertical windows at the top for code and one wide horizontal one for the debugger, but it's up to you.  Press the Control key and the “x” key at the same time, then press “2” by itself (notated C-x 2) to split the window in half, then press (C-x 3) to half it again.



```
Buffer: *scratch*- 2:26PM 1.16(Lisp InteracBuffer: *scratch*- 2:26PM 1.16(Lisp Interac
Buffer: *scratch*- 2:26PM 1.16(Lisp Interaction) File:  [All]
```

After some resizing...

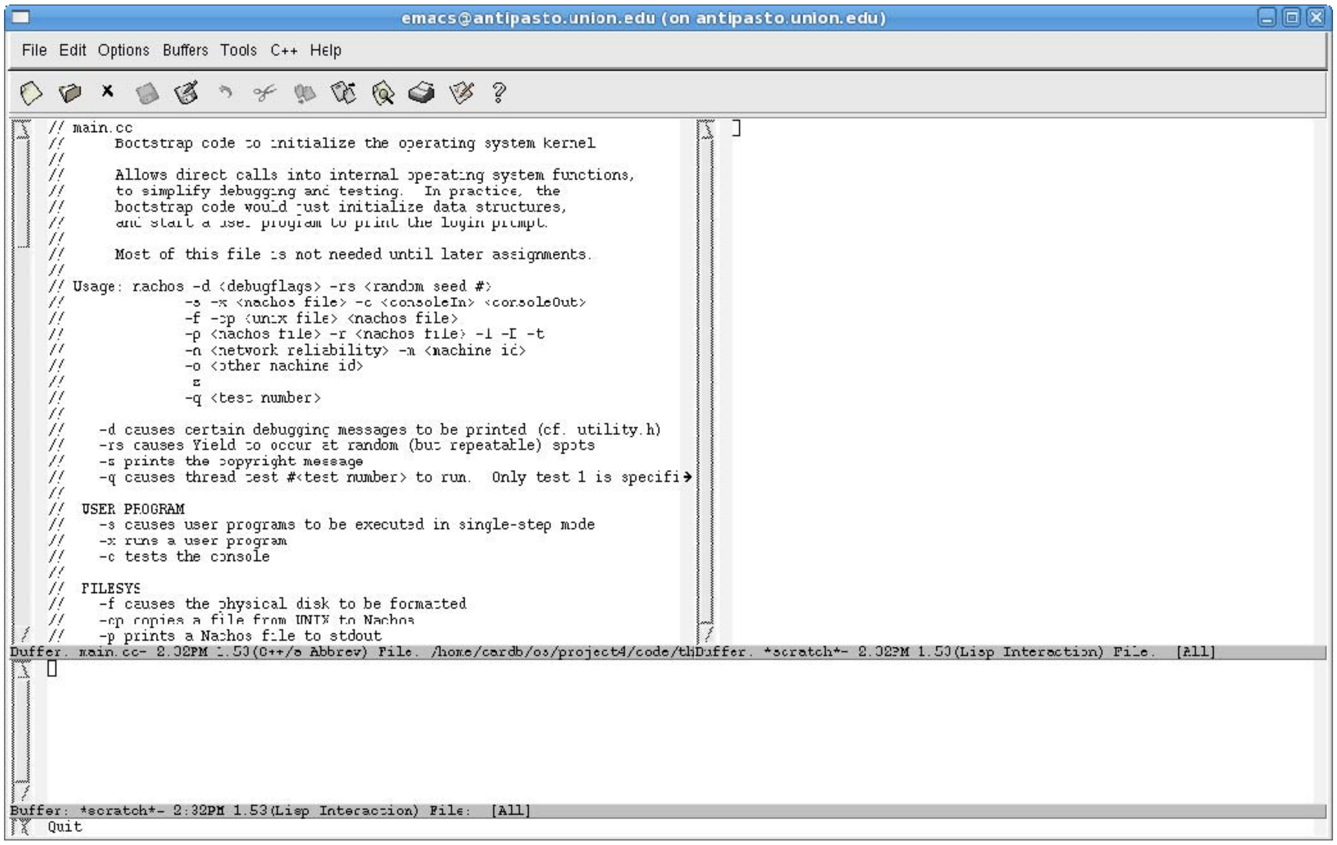

emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Help

```
Buffer: *scratch*- 2:27PM 1.38(Lisp Interaction) File:  [All]
```

```
Buffer: *scratch*- 2:27PM 1.38(Lisp Interaction) File:  [All]
```

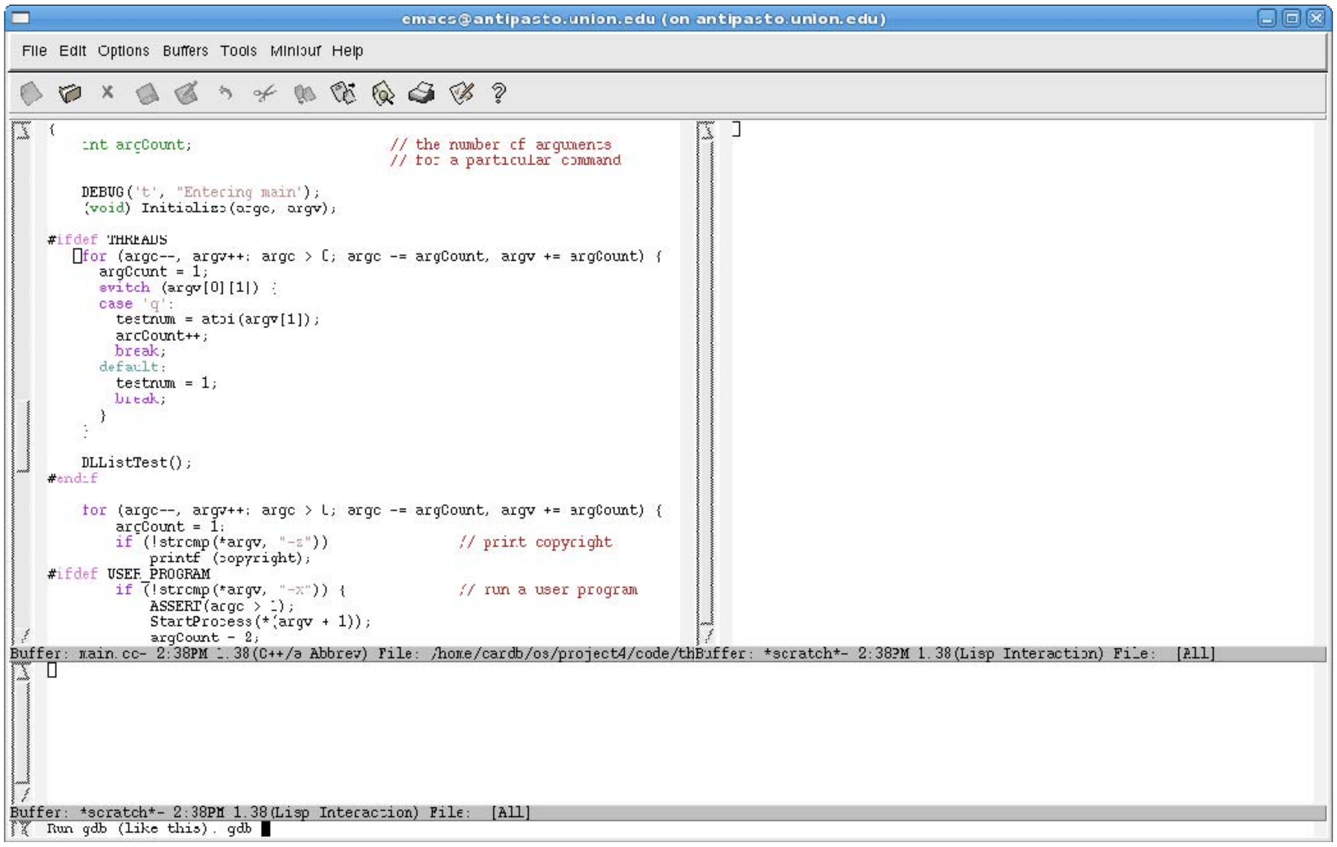
```
Buffer: *scratch*- 2:27PM 1.38(Lisp Interaction) File:  [All]
```

Open up the file you want to debug.  For this example we will look at the main.cc file and watch it run through the debugging printouts.  Click in the top left window and type (C-x C-f) then “main.cc” to open the file (if you mess up (C-g) cancels).


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools C++ Help

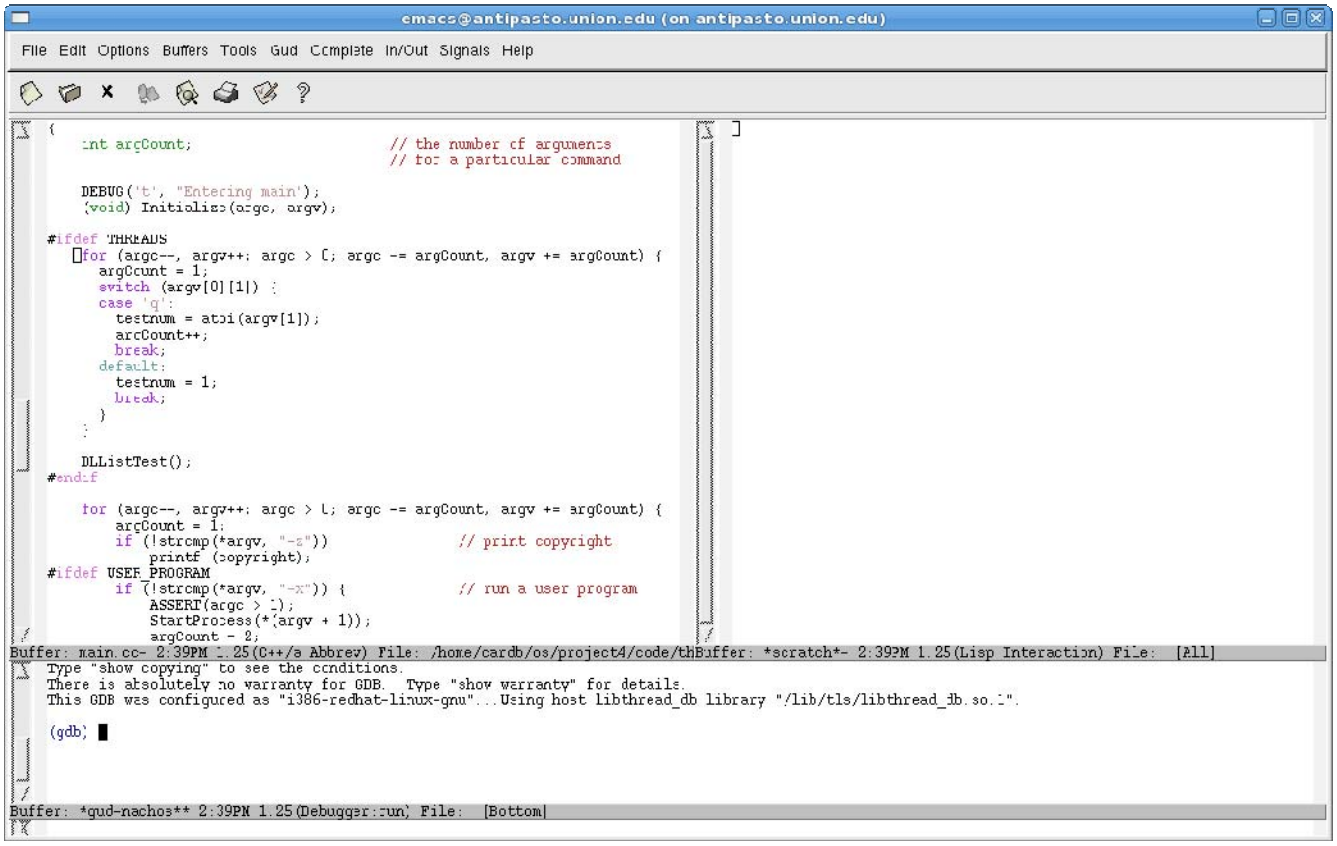
```
// main.cc
//      Bootstrap code to initialize the operating system kernel
//
//      Allows direct calls into internal operating system functions,
//      to simplify debugging and testing.  In practice, the
//      bootstrap code would just initialize data structures,
//      and start a user program to print the login prompt.
//
//      Most of this file is not needed until later assignments.
//
// Usage: nachos -d <debugflags> -rs <random seed #>
//              -s -x <nachos file> -c <consoleIn> <consoleOut>
//              -f -cp <unix file> <nachos file>
//              -p <nachos file> -r <nachos file> -l -D -t
//              -n <network reliability> -m <machine id>
//              -o <other machine id>
//               -z
//              -q <test number>
//
//    -d causes certain debugging messages to be printed (cf. utility.h)
//    -rs causes Yield to occur at random (but repeatable) spots
//    -z prints the copyright message
//    -q causes thread test #<test number> to run.  Only test 1 is specifi
//
//  USER PROGRAM
//    -s causes user programs to be executed in single-step mode
//    -x runs a user program
//    -c tests the console
//
//  FILESYS
//    -f causes the physical disk to be formatted
//    -cp copies a file from UNIX to Nachos
//    -p prints a Nachos file to stdout
Buffer: main.cc- 2:32PM 1.53(C++/a Abbrev) File: /home/cardb/os/project4/code/thBuffer: *scratch*- 2:32PM 1.53(Lisp Interaction) File:  [All]
Buffer: *scratch*- 2:32PM 1.53(Lisp Interaction) File:  [All]
Quit
```

Go to the Options menu and click Syntax Highlight to get some color.
For this example we want to set a breakpoint at the first line in the main() function (line 90, use (alt-x) then type “goto-line” then press enter and type 90 to jump to that line).
First we need to start the debugger. Click on the bottom frame (or whatever window you want the debugger to occupy) and click the tools menu and select “Debugger (GUD)...”.


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Minibuf Help

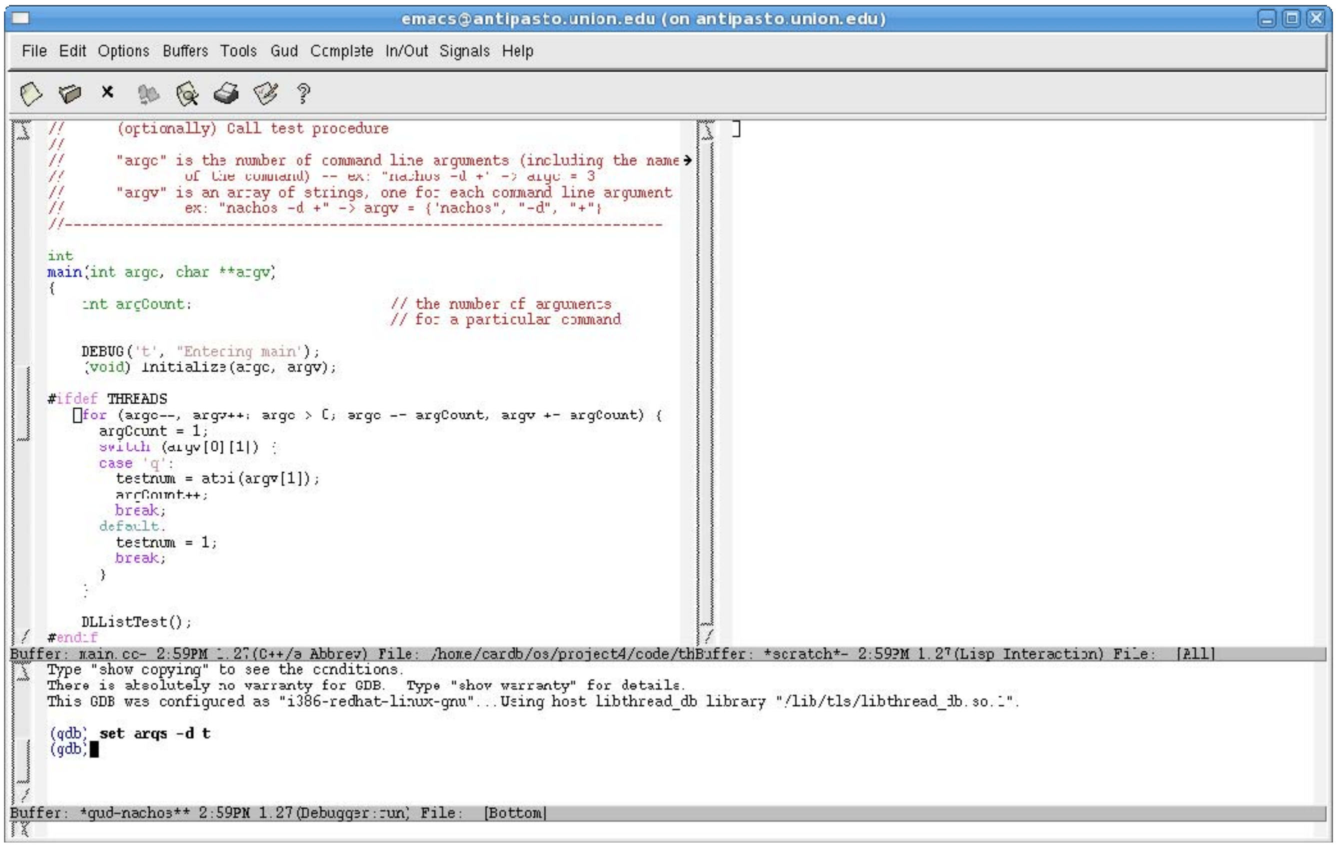
```
{
    int argCount;                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main");
    (void) Initialize(argc, argv);

#ifdef THREADS
   for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif

    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
        argCount = 1;
        if (!strcmp(*argv, "-z"))               // print copyright
            printf (copyright);
#ifdef USER_PROGRAM
        if (!strcmp(*argv, "-x")) {             // run a user program
            ASSERT(argc > 1);
            StartProcess(*(argv + 1));
            argCount = 2;
Buffer: main.cc- 2:38PM 1.38(C++/a Abbrev) File: /home/cardb/os/project4/code/thBuffer: *scratch*- 2:38PM 1.38(Lisp Interaction) File:  [All]

Buffer: *scratch*- 2:38PM 1.38(Lisp Interaction) File:  [All]
Run gdb (like this): gdb
```

Type “nachos” and press enter.


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

```
{
    int argCount;                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main");
    (void) Initialize(argc, argv);

#ifdef THREADS
   for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif

    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
        argCount = 1;
        if (!strcmp(*argv, "-z"))               // print copyright
            printf (copyright);
#ifdef USER_PROGRAM
        if (!strcmp(*argv, "-x")) {             // run a user program
            ASSERT(argc > 1);
            StartProcess(*(argv + 1));
            argCount = 2;
Buffer: main.cc- 2:39PM 1.25(C++/a Abbrev) File: /home/cardb/os/project4/code/thBuffer: *scratch*- 2:39PM 1.25(Lisp Interaction) File:  [All]
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-gnu"...Using host libthread_db library "/lib/tls/libthread_db.so.1".

(gdb) █
Buffer: *gud-nachos** 2:39PM 1.25(Debugger:run) File:  [Bottom]
```

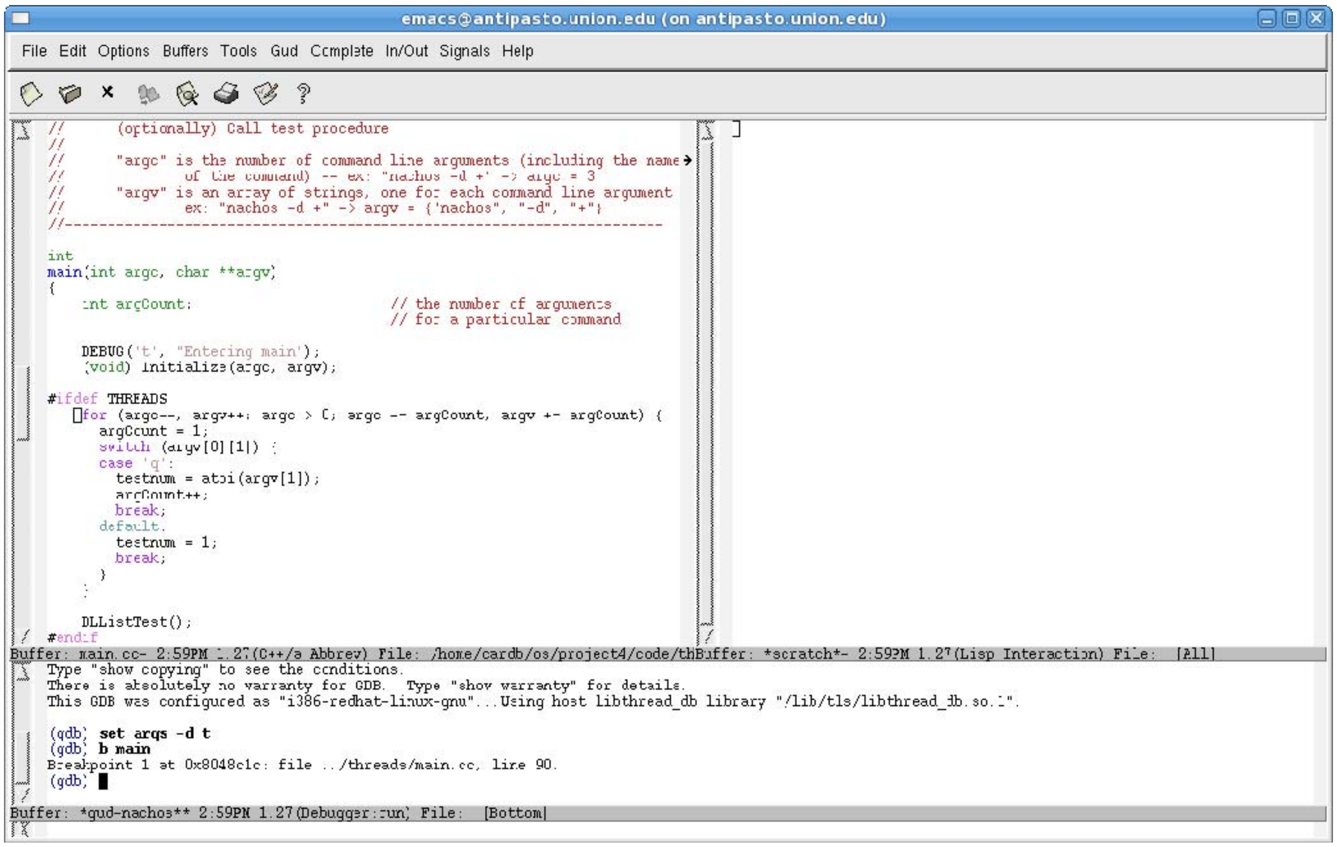
If you wanted to run nachos with some options (like -d to debug), you can type “set args ...” into the debugger where the ... represents your arguments.  We will use the -d t enable debugging.


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

```
//      (optionally) Call test procedure
//
//      "argc" is the number of command line arguments (including the name
//              of the command) -- ex: "nachos -d +" -> argc = 3
//      "argv" is an array of strings, one for each command line argument
//              ex: "nachos -d +" -> argv = {"nachos", "-d", "+"}
//----------------------------------------------------------------------

int
main(int argc, char **argv)
{
    int argCount;                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main");
    (void) Initialize(argc, argv);

#ifdef THREADS
   for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif
Buffer: main.cc- 2:59PM 1.27(C++/a Abbrev) File: /home/cardb/os/project4/code/thBuffer: *scratch*- 2:59PM 1.27(Lisp Interaction) File:  [All]
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-gnu"...Using host libthread_db library "/lib/tls/libthread_db.so.1".

(gdb) set args -d t
(gdb) 

Buffer: *gud-nachos** 2:59PM 1.27(Debugger:run) File:  [Bottom]
```

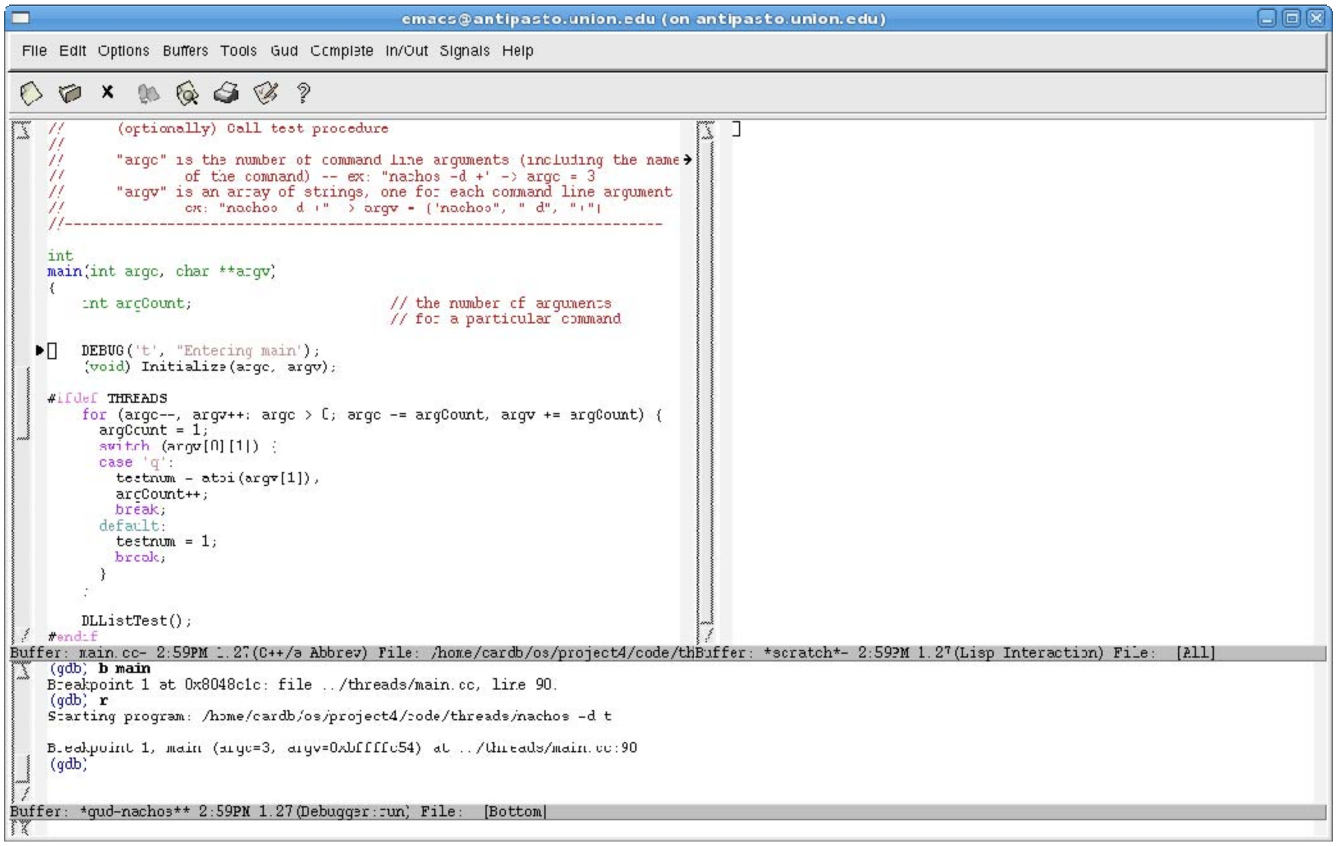
You'll notice that when the cursor is in the bottom window the menu bar changes and you get Gud, Complete, In/Out, and Signals menus.  The Gud is the only important one and has commands and shortcuts for the debugger.

The first breakpoint has to be inserted manually.  Type “b main” into the debugger and press enter (b is short for break).


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

```
//      (optionally) Call test procedure
//
//      "argc" is the number of command line arguments (including the name
//              of the command) -- ex: "nachos -d +" -> argc = 3
//      "argv" is an array of strings, one for each command line argument
//              ex: "nachos -d +" -> argv = {"nachos", "-d", "+"}
//----------------------------------------------------------------------

int
main(int argc, char **argv)
{
    int argCount;                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main");
    (void) Initialize(argc, argv);

#ifdef THREADS
    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif
Buffer: main.cc- 2:59PM 1.27(C++/a Abbrev) File: /home/cardb/os/project4/code/thBuffer: *scratch*- 2:59PM 1.27(Lisp Interaction) File:  [All]
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-gnu"...Using host libthread_db library "/lib/tls/libthread_db.so.1".

(gdb) set args -d t
(gdb) b main
Breakpoint 1 at 0x8048c1c: file ../threads/main.cc, line 90.
(gdb) 
Buffer: *gud-nachos** 2:59PM 1.27(Debugger:run) File:  [Bottom]
```

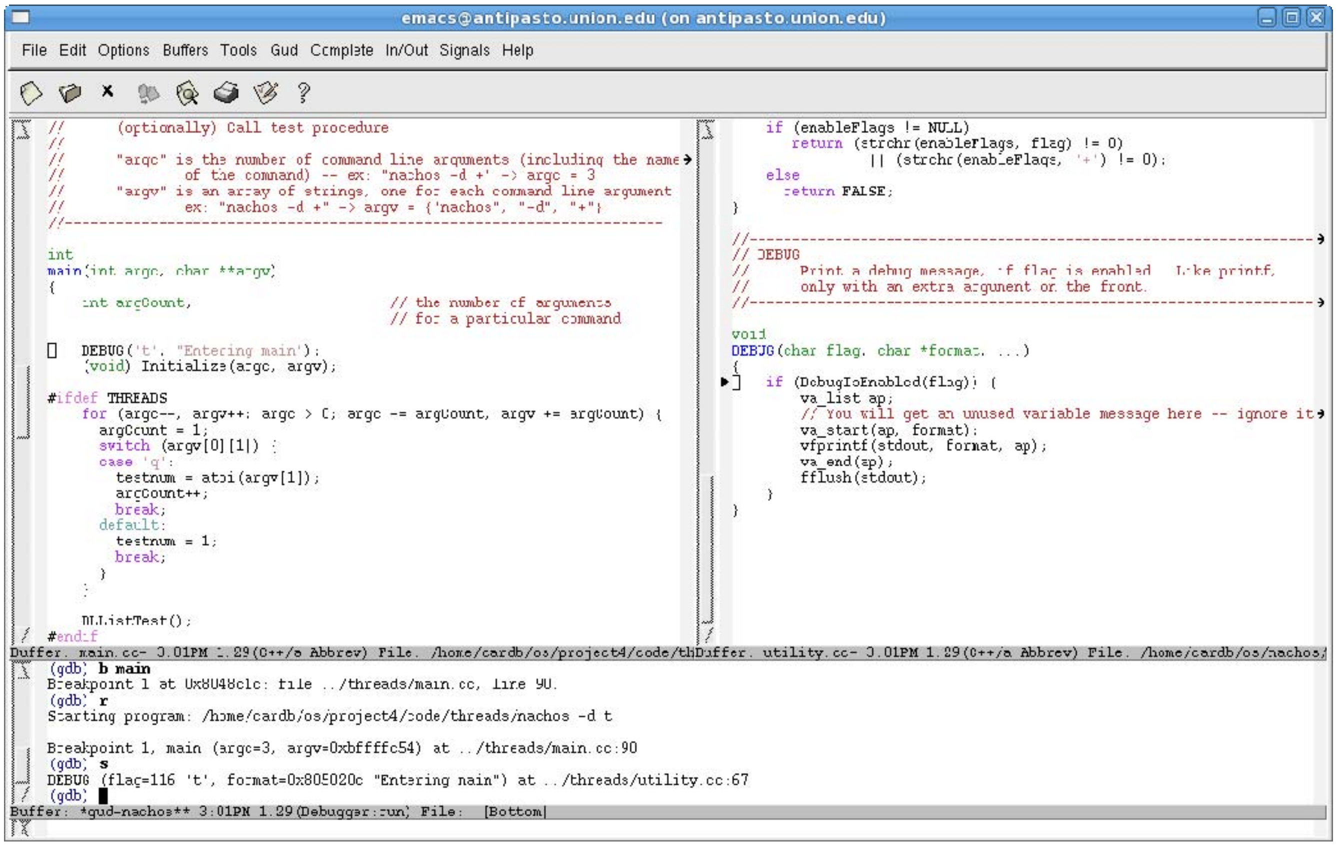
Type “r” and press enter to run.


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

```
//      (optionally) Call test procedure
//
//      "argc" is the number of command line arguments (including the name
//              of the command) -- ex: "nachos -d +" -> argc = 3
//      "argv" is an array of strings, one for each command line argument
//              ex: "nachos -d +" -> argv = {"nachos", "-d", "+"}
//----------------------------------------------------------------------

int
main(int argc, char **argv)
{
    int argCount;                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main");
    (void) Initialize(argc, argv);

#ifdef THREADS
    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif
Buffer: main.cc- 2:59PM 1.27(C++/a Abbrev) File: /home/cardb/os/project4/code/thBuffer: *scratch*- 2:59PM 1.27(Lisp Interaction) File:  [All]
```

```
(gdb) b main
Breakpoint 1 at 0x8048c1c: file ../threads/main.cc, line 90.
(gdb) r
Starting program: /home/cardb/os/project4/code/threads/nachos -d t

Breakpoint 1, main (argc=3, argv=0xbffffc54) at ../threads/main.cc:90
(gdb)

Buffer: *gud-nachos** 2:59PM 1.27(Debugger:run) File:  [Bottom]
```

The program has started running and has now stopped at line 90, at the beginning of the main() function (actually a little after, but that's ok). You'll notice in your code that there is an arrow at the DEBUG statement. This is the next line the debugger will execute. Type “s” into the debugger to step into the function at the highlighted line.


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

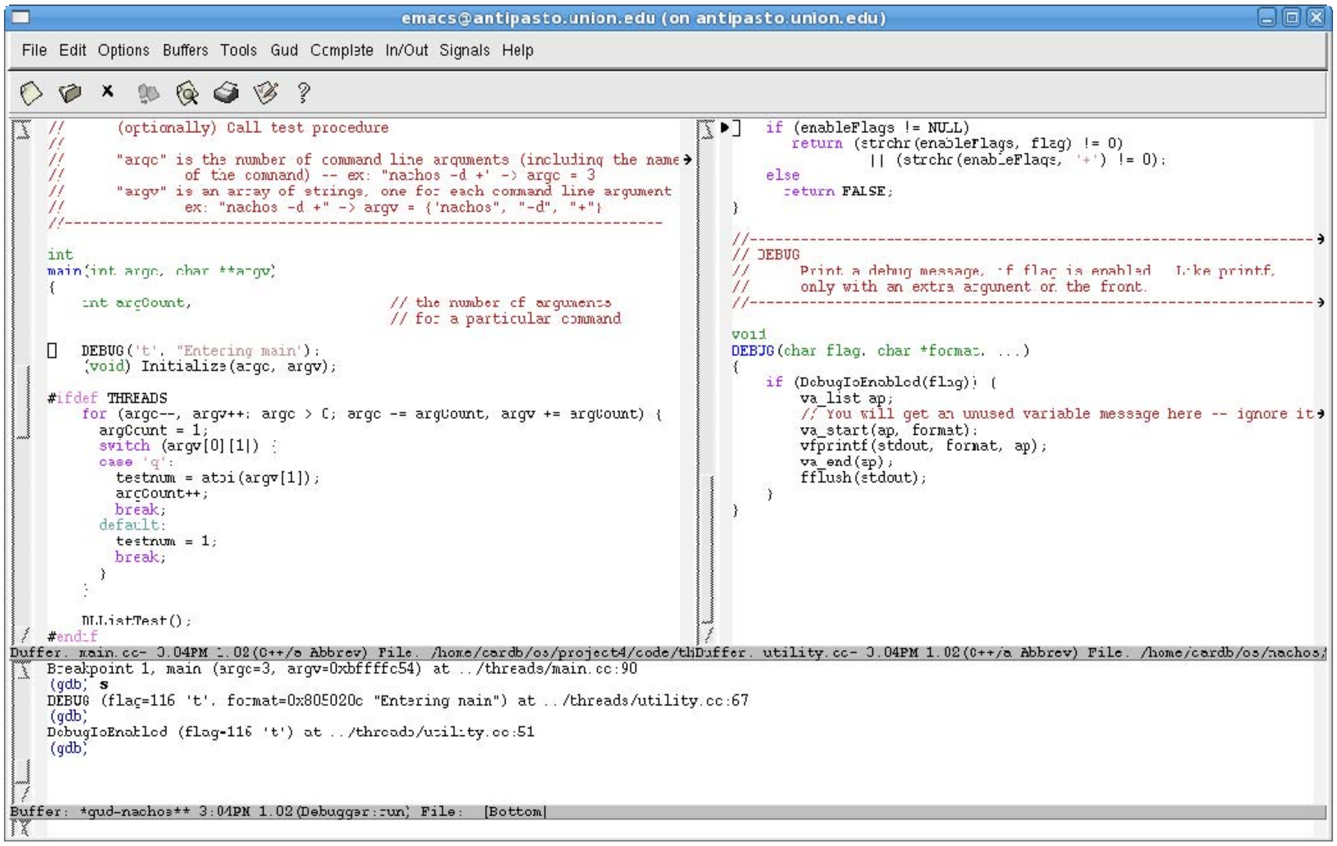
```
//      (optionally) Call test procedure
//
//      "argc" is the number of command line arguments (including the name
//              of the command) -- ex: "nachos -d +' -> argc = 3
//      "argv" is an array of strings, one for each command line argument
//              ex: "nachos -d +" -> argv = {'nachos", "-d", "+"}
//----------------------------------------------------------------------

int
main(int argc, char **argv)
{
    int argCount;                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main");
    (void) Initialize(argc, argv);

#ifdef THREADS
    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif
```

```
    if (enableFlags != NULL)
       return (strchr(enableFlags, flag) != 0)
                || (strchr(enableFlags, '+') != 0);
    else
      return FALSE;
}

//----------------------------------------------------------------------
// DEBUG
//      Print a debug message, if flag is enabled.  Like printf,
//      only with an extra argument on the front.
//----------------------------------------------------------------------

void
DEBUG(char flag, char *format, ...)
{
    if (DebugIsEnabled(flag)) {
        va_list ap;
        // You will get an unused variable message here -- ignore it
        va_start(ap, format);
        vfprintf(stdout, format, ap);
        va_end(ap);
        fflush(stdout);
    }
}
```

```
Buffer: main.cc- 3:01PM 1.29(C++/a Abbrev) File: /home/cardb/os/project4/code/th
Buffer: utility.cc- 3:01PM 1.29(C++/a Abbrev) File: /home/cardb/os/nachos/
```

```
(gdb) b main
Breakpoint 1 at 0x8048c1c: file ../threads/main.cc, line 90.
(gdb) r
Starting program: /home/cardb/os/project4/code/threads/nachos -d t

Breakpoint 1, main (argc=3, argv=0xbffffc54) at ../threads/main.cc:90
(gdb) s
DEBUG (flag=116 't', format=0x805020c "Entering main") at ../threads/utility.cc:67
(gdb) 
Buffer: *gud-nachos** 3:01PM 1.29(Debugger:run) File:  [Bottom]
```

You'll notice that the empty window was filled with the utility.cc file where the DEBUG function lives. Emacs automatically opened this for you and scrolled to the line that will be executed next. If you have more windows, you can have many different sections of source code to step through. Emacs will toggle back and forth between them as necessary. Press enter to execute the previous command again (in this case “s” for step).



emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

```
//      (optionally) Call test procedure
//
//      "argc" is the number of command line arguments (including the name
//              of the command) -- ex: "nachos -d +" -> argc = 3
//      "argv" is an array of strings, one for each command line argument
//              ex: "nachos -d +" -> argv = {"nachos", "-d", "+"}
//----------------------------------------------------------------------

int
main(int argc, char **argv)
{
    int argCount;                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main");
    (void) Initialize(argc, argv);

#ifdef THREADS
    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif
```

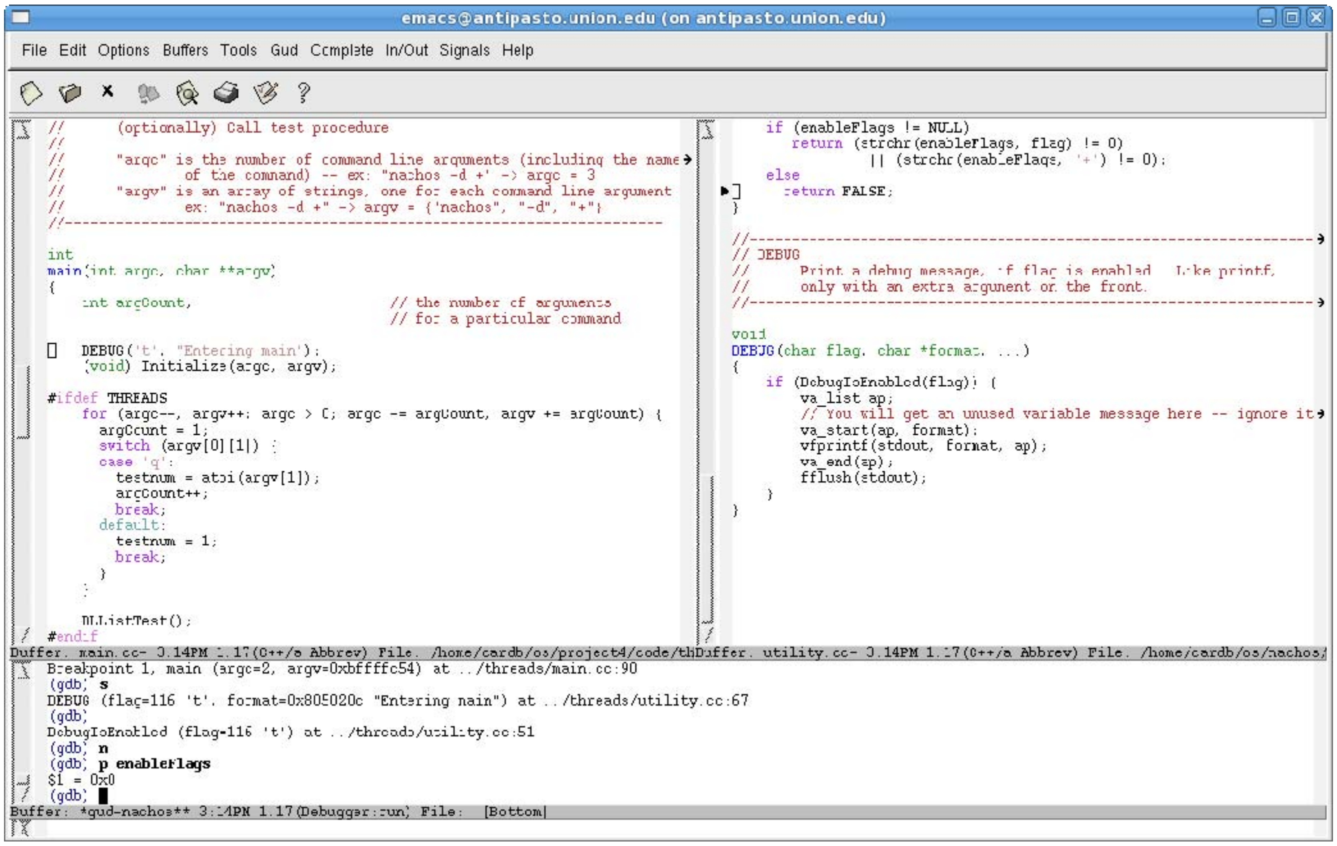
```
    if (enableFlags != NULL)
       return (strchr(enableFlags, flag) != 0)
                || (strchr(enableFlags, '+') != 0);
    else
      return FALSE;
}

//----------------------------------------------------------------------
// DEBUG
//      Print a debug message, if flag is enabled.  Like printf,
//      only with an extra argument on the front.
//----------------------------------------------------------------------

void
DEBUG(char flag, char *format, ...)
{
    if (DebugIsEnabled(flag)) {
        va_list ap;
        // You will get an unused variable message here -- ignore it
        va_start(ap, format);
        vfprintf(stdout, format, ap);
        va_end(ap);
        fflush(stdout);
    }
}
```

```
Buffer: main.cc- 3:04PM 1.02 (C++/a Abbrev) File: /home/cardb/os/project4/code/th
Buffer: utility.cc- 3:04PM 1.02 (C++/a Abbrev) File: /home/cardb/os/nachos/
```

```
Breakpoint 1, main (argc=3, argv=0xbffffc54) at ../threads/main.cc:90
(gdb) s
DEBUG (flag=116 't', format=0x805020c "Entering main") at ../threads/utility.cc:67
(gdb)
DebugIsEnabled (flag=116 't') at ../threads/utility.cc:51
(gdb)
```

```
Buffer: *gud-nachos** 3:04PM 1.02 (Debugger:run) File:  [Bottom]
```

The arrow jumped to the top of the window where the DebugIsEnabled() function is.  If you scrolled up a bit you would see the function declaration.  Type “n” for next to step through this function without entering the sub functions.  If we wanted to check the value of a variable, type “p var” what var is the name of the variable.  Lets look at enableFlags to see if it really was NULL.

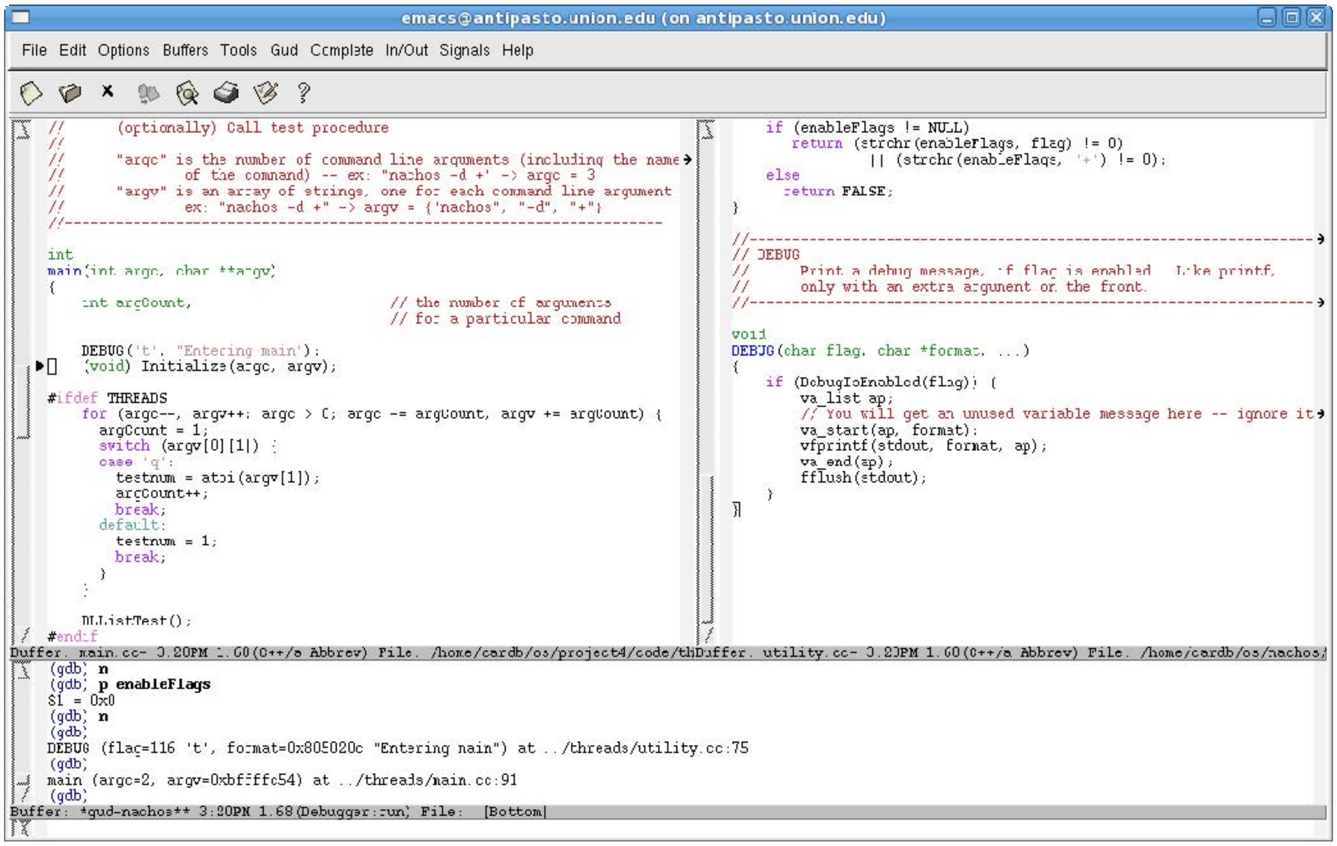

emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

```
//      (optionally) Call test procedure
//
//      "argc" is the number of command line arguments (including the name
//              of the command) -- ex: "nachos -d +' -> argc = 3
//      "argv" is an array of strings, one for each command line argument
//              ex: "nachos -d +" -> argv = {'nachos", "-d", "+"}
//----------------------------------------------------------------------

int
main(int argc, char **argv)
{
    int argCount,                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main');
    (void) Initialize(argc, argv);

#ifdef THREADS
    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif
```

```
    if (enableFlags != NULL)
       return (strchr(enableFlags, flag) != 0)
                || (strchr(enableFlags, '+') != 0);
    else
      return FALSE;
}

//----------------------------------------------------------------------
// DEBUG
//      Print a debug message, if flag is enabled.  Like printf,
//      only with an extra argument on the front.
//----------------------------------------------------------------------

void
DEBUG(char flag, char *format, ...)
{
    if (DebugIsEnabled(flag)) {
        va_list ap;
        // You will get an unused variable message here -- ignore it
        va_start(ap, format);
        vfprintf(stdout, format, ap);
        va_end(ap);
        fflush(stdout);
    }
}
```

```
Buffer: main.cc- 3:14PM 1.17(C++/a Abbrev) File: /home/cardb/os/project4/code/thBuffer: utility.cc- 3:14PM 1.17(C++/a Abbrev) File: /home/cardb/os/nachos/
Breakpoint 1, main (argc=2, argv=0xbffffc54) at ../threads/main.cc:90
(gdb) s
DEBUG (flag=116 't', format=0x805020c "Entering main") at ../threads/utility.cc:67
(gdb)
DebugIsEnabled (flag=116 't') at ../threads/utility.cc:51
(gdb) n
(gdb) p enableFlags
$1 = 0x0
(gdb) 
Buffer: *gud-nachos** 3:14PM 1.17(Debugger:run) File:  [Bottom]
```

Hmm, it is, I guess that's why this line won't print (actually, it never will, I don't even know why it's there).  GDB use C style syntax for the variable calls and is context dependent.  So, if we were **inside** a rectangle class and we wanted to see what the “width” variable contained, we would type “p width”.  However, if we were in the function that created the rectangle (say the created rectangle is called rec), we would type “p rec->width” to see the width.  Type “n” a couple more times to run through the debug code.


emacs@antipasto.union.edu (on antipasto.union.edu)
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

```
//      (optionally) Call test procedure
//
//      "argc" is the number of command line arguments (including the name
//              of the command) -- ex: "nachos -d +" -> argc = 3
//      "argv" is an array of strings, one for each command line argument
//              ex: "nachos -d +" -> argv = {"nachos", "-d", "+"}
//----------------------------------------------------------------------

int
main(int argc, char **argv)
{
    int argCount;                       // the number of arguments
                                        // for a particular command

    DEBUG('t', "Entering main");
    (void) Initialize(argc, argv);

#ifdef THREADS
    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount) {
      argCount = 1;
      switch (argv[0][1]) {
      case 'q':
        testnum = atoi(argv[1]);
        argCount++;
        break;
      default:
        testnum = 1;
        break;
      }
    }

    DLListTest();
#endif
```

```
    if (enableFlags != NULL)
       return (strchr(enableFlags, flag) != 0)
                || (strchr(enableFlags, '+') != 0);
    else
      return FALSE;
}

//----------------------------------------------------------------------
// DEBUG
//      Print a debug message, if flag is enabled.  Like printf,
//      only with an extra argument on the front.
//----------------------------------------------------------------------

void
DEBUG(char flag, char *format, ...)
{
    if (DebugIsEnabled(flag)) {
        va_list ap;
        // You will get an unused variable message here -- ignore it.
        va_start(ap, format);
        vfprintf(stdout, format, ap);
        va_end(ap);
        fflush(stdout);
    }
}
```

```
Buffer: main.cc- 3:20PM 1.60 (C++/a Abbrev) File: /home/cardb/os/project4/code/th  Buffer: utility.cc- 3:20PM 1.60 (C++/a Abbrev) File: /home/cardb/os/nachos/
(gdb) n
(gdb) p enableFlags
$1 = 0x0
(gdb) n
(gdb)
DEBUG (flag=116 't', format=0x805020c "Entering main") at ../threads/utility.cc:75
(gdb)
main (argc=2, argv=0xbffffc54) at ../threads/main.cc:91
(gdb)
Buffer: *gud-nachos** 3:20PM 1.68 (Debugger:run) File:  [Bottom]
```

The cursor jumped back to the main.cc frame and execution continues.  You'll notice now that when you click inside the main.cc or utility.cc windows you will have a “Gud” menu bar.  This only appears if you have entered the code in that window.  With this menu bar you can finally ad breakpoints at arbitrary lines by clicking on a line and selecting “Set Breakpoint”.

That's pretty much it.  There are a few good things to know, like typing “f” into the debugger will open the window where the current line is (if you get messed up somehow).  If you want to jump from breakpoint to breakpoint, type “c” for continue. Typing “info b” shows the breakpoints, to get rid of one find it's number and type “disable num”, where num is the number.  There are some nice things in the tools menu, like compile, which means you never have to leave Emacs to compile your code (in addition, typing (alt-x) then typing “shell” will open a shell in the selected window.

Emacs is complicated but powerful.  Good tools are essential to creating good software; you don't want to spend most of your time fighting the system instead of getting work done.  Hope this guide helps, good luck!