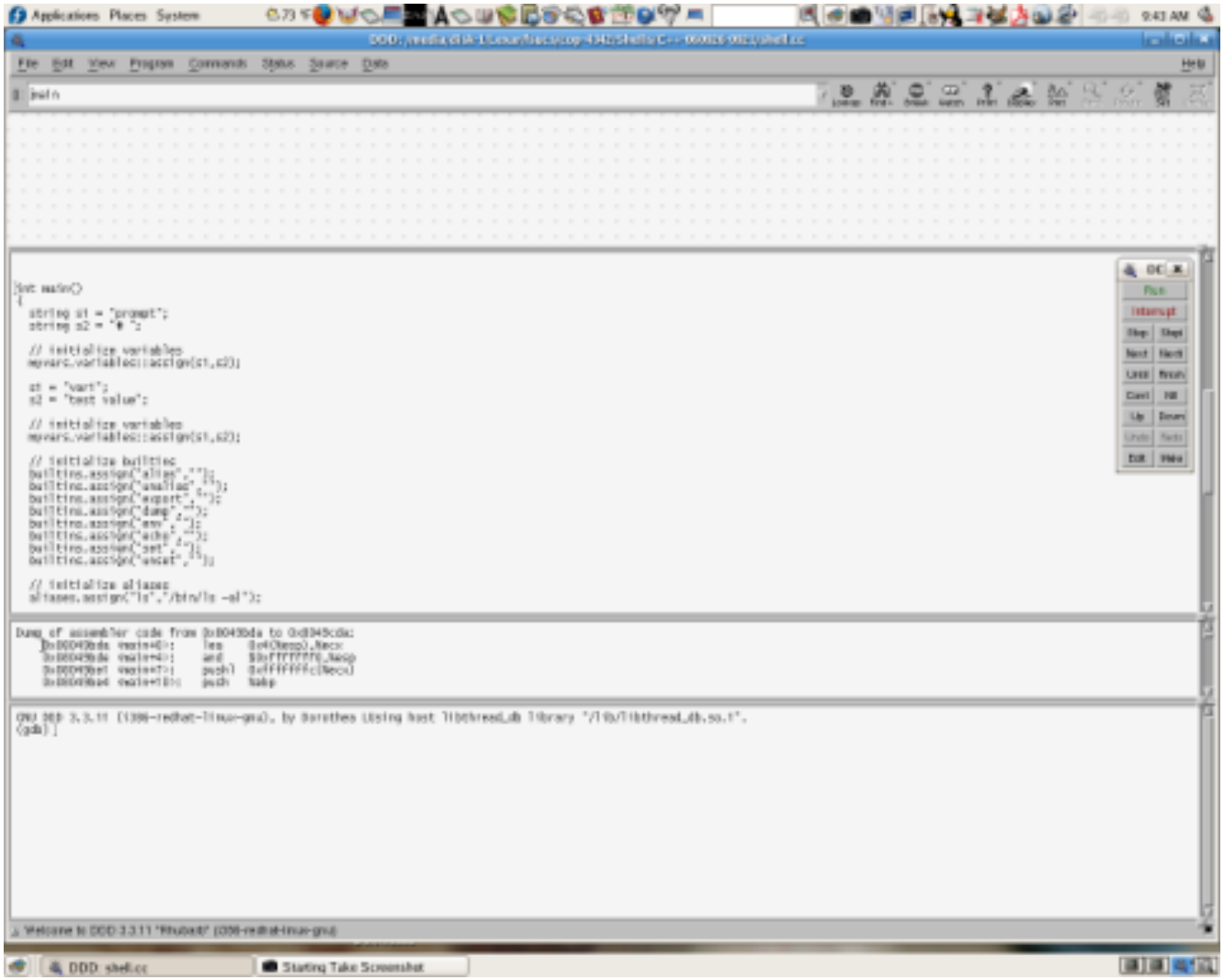
More tools: DDD

The Data Display Debugger (DDD) is a graphical front-end for GDB and other command line debuggers.
From DDD you can execute all of the GDB commands.
It also has a graphical interface which displays GDB commands, shows source code, shows executions, and allows to choose common options for commands.


Applications Places System
File Edit View Program Commands Status Source Data
int main()
{
string s1 = "prompt";
string s2 = "$ ";
// initialize variables
myvars.variables::assign(s1,s2);
s1 = "var";
s2 = "test value";
// initialize variables
myvars.variables::assign(s1,s2);
// initialize builtins
builtins.assign("alias","");
builtins.assign("unalias","");
builtins.assign("export","");
builtins.assign("dump","");
builtins.assign("env","");
builtins.assign("echo","");
builtins.assign("set","");
builtins.assign("uncat","");
// initialize aliases
aliases.assign("ls","/bin/ls -al");
Run
Interrupt
Step
Next
Until
Cont
Up
Undo
Edit
Make
Dump of assembler code from 0x8049bda to 0x8049cda:
0x8049bda <main+0>: lea 0x4(%esp),%ecx
0x8049bde <main+4>: and $0xfffffff0,%esp
0x8049be1 <main+7>: pushl 0xfffffffc(%ecx)
0x8049be4 <main+10>: push %ebp
(gdb)
Welcome to DDD 3.3.11 "Rhubarb" (i386-redhat-linux-gnu)
DDD: shell.cc
Starting Take Screenshot


FLORIDA STATE UNIVERSITY
VIRES
ARTES
MORES
1851

DDD features

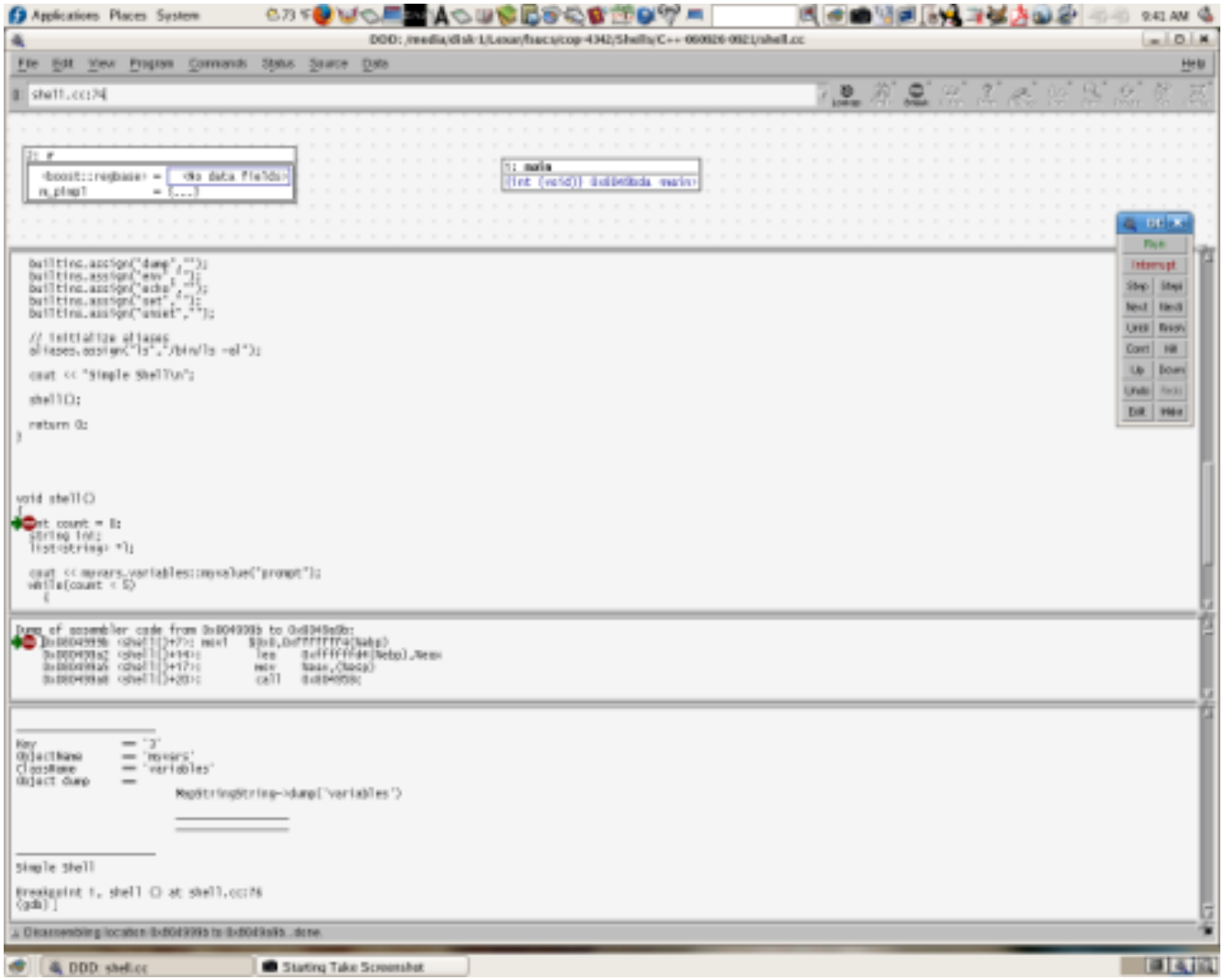
DDD shows four different windows:

- A data window to display variables.
- Source window to display source code.
- Machine code window to display disassembled machine code

DDD features

- GDB console where conventional gdb commands can be typed.

DDD also has other panels which include common commands that can be selected with the mouse.


Applications Places System
DDD: /media/disk-1/Lexar/fsucs/cop-4342/Shells/C++ 000820-0821/shell.cc
File Edit View Program Commands Status Source Data Help
shell.cc:74
builtins.assign("dump","");
builtins.assign("env","");
builtins.assign("echo","");
builtins.assign("set","");
builtins.assign("unset","");
// Initialize aliases
aliases.assign("ls","/bin/ls -al");
cout << "Simple Shell\n";
shell();
return 0;
}
void shell()
{
int count = 0;
string in;
list<string> *l;
cout << myvars.variables::myvalue("prompt");
while(count < 5)
{
Dump of assembler code from 0x804930b to 0x8049a0b:
0x08049996 <shell()+7>: mov %ebp,0xfffffff4(%ebp)
0x08049a3a <shell()+14>: lea 0xfffffff4(%ebp),%eax
0x08049a3d <shell()+17>: mov %eax,(%esp)
0x08049a40 <shell()+20>: call 0x8049930
Key = "2"
ObjectName = "myvars"
ClassName = "variables"
Object dump =
MapStringString->dump("variables")
Simple Shell
Breakpoint 1, shell () at shell.cc:74
(gdb)
DDD shell.cc
Starting Take Screenshot

Using the DDD Source Window

- Can set a breakpoint by using the right mouse button and positioning the cursor to the left of a source code line.
- Can instantly view the value of a variable by placing the mouse over it (look at the very bottom of the display.)
- Can highlight a variable and select to print or display its value by using the options at the top.

Using the DDD Data Window

To have a variable with its value appear in the data window as a display:

- A user can highlight a variable in the source window and then click on the display button.
- A user can double click on a variable in the source window.

diff

The diff Unix utility compares two files and displays the differences between the two files. The differences are displayed with an ed-like notation indicating what changes are needed to modify the first file to make it similar to the second.
`diff` is very useful in shell scripts to detect differences between expected output and actual output.

`diff` Output (UPT 11.1)

- Diff output consists of a list of changes.
- General form consists of a sequence of:

```
commands
lines
```

`diff` Output (UPT 11.1)

- Commands are of the form (a for append, c for change, and d for delete):

  ```
  linenums [acd] linenums
  ```

- Lines from the first file are preceded by "<". Lines from the second file are preceded by ">".
- `diff -r` can be recursively to compare whole directories trees.

diff Example

tmp1.txt:

```
cat
dog
mouse
```

tmp2.txt:

```
cat
mouse
```

tmp3.txt:

```
dog
mouse
cow
```

```
% diff tmp1.txt tmp2.txt
2d1
< dog
% diff tmp2.txt tmp3.txt
1d0
< cat
3a3
> cow
```

```
% diff tmp2.txt tmp3.txt
1c1
< cat
---
> dog
2a3
> cow
```

Patch (UPT 20.9)

Patch is a Unix utility that takes diff output and applies the commands to make the first file have the same contents as the second file. Updates to free software are often accomplished using patch. Often the differences between versions of files is much smaller than the files themselves.

The cmp Unix utility just returns a status indicating if the files differ.

Exit status	Meaning
0	Files are identical
1	Files are different
2	An error occurred

The cmp utility is often used when comparing two binary files since it is typically quicker than diff.
You can also specify `-s` to make `cmp` silent when it finds a difference (by default, it displays the byte and line number where the first difference was found.)

Configuration Management Systems

Definitely not the same as a Content Management System!
Configuration Management Systems are however quite similar to Content Management Systems (CMSs):

- Configuration Management Systems always provide a history mechanism, as do most CMSs.
- Provides controlled access by different users to shared files.

Configuration Management Systems

- SCCS – Source Code Control System. This is now deprecated. It kept the original files, and the deltas to get to the current version(s) of code.
- RCS – Revision Control System. Still popular. It keeps the most recent version(s) of files, and the deltas to take you back to older version(s).
- CVS – Concurrent Version System. Quite popular. Actually uses RCS underneath.
- subversion – Also quite popular, and is a strong competitor with CVS. Directories and file meta-data are also kept under version control. Commits are also truly atomic.

gprof

The gprof Unix utility produces an execution profile of the call graph of a program.
The profile is useful for finding out where most of the time is spent during the execution of your program.
A gmon.out file will be produced as a side effect A developer can use this information to tune the time-consuming portions of a long-running program.

gprof

You can have a program instrumented to collect data that can be processed by gprof by using the `-pg` option when compiling with gcc:

```
gcc -pg -c XYZ.c
```

A gmon.out file will be produced as a side effect of running your program.
You can obtain the profile from the gmon.out file by running the following command:

```
gprof -b
```