Debugging C Code

Compared to other programming languages, C can be a more difficult language to debug and figure out errors within code, no matter if it is logic or syntax-based. Even runtime errors hide behind a vague message. Let's see this in action.

```
:~$ ./a.out
Segmentation fault (core dumped)
```

Segmentation fault are caused by a program trying to read or write an illegal memory location and they stop the program from executing any longer. The problem with this error message is that a segmentation fault can occur for many different situations and can be caused by many different logical or syntactical errors. Let's look at ways to expand this error message and debug the program.

The first thing to do is recompile the program with these **`cflags`** in the command

```
:~$ gcc -Wall -pedantic -g test.c
test.c: In function 'main':
Test.c:5:5: warning: format '%s' expects argument of
type 'char *', but argument 2 has type 'int'
['-Wformat]
```

Recompiling with these flags, we see more descriptive warnings when errors occur. `-Wall` and `-pedantic` have been discussed in class while `-g` gives us debugging options. From looking at this and the code, a simple solution of changing the format specifier or the argument for `printf()` will fix this solution and work as intended.

Sometimes errors cannot be solved this simply. This is where using debuggers and debugging statements come in handy.

Enter GNU debugger (**`gdb`**), a debugger that is already installed on Loki. This debugger allows us to run our program and get more meaningful errors. **Here is how to start up the debugger with your program.**

```
:~$ gdb a.out
// Licensing Stuff
(gdb) run
```

COLLEGE OF INFORMATION SCIENCE & TECHNOLOGY
COMPUTER SCIENCE LEARNING CENTER

This workforce product was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The product was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The U.S. Department of Labor makes no guarantees, warranties, or assurances of any kind, express or implied, with respect to such information, including any information on linked sites and including, but not limited to accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability, or ownership.

Debugging C Code

When we run our program under the debugger, it gives us more information about why a segmentation fault occurred. Here are some useful flags and gdb commands in order to get more information.

`gdb --args`: Allows you to pass command line arguments to the program instead of gdb
`gdb --batch --ex r --ex bt --ex q --args`: This command runs the program under gdb and gives you the backtrace up to when the program crashed.

- **`break (b) [file]:[line number]`:** Set a breakpoint at [file]:[line number]. Shortcut is b.
- **`break (b) [line number]`:** Set breakpoint at [line number] in file with main.
- **`backtrace (bt)`:** Gives the backtrace on the current calling methods.
- **`print expr (p)`:** Prints the value of a variable. You can use format specifiers like this: `p/c i` will print `i` as a `char`, even if its an `int`
- **`next (n)`:** Goes to next line and steps over function calls
- **`step (s)`:** Goes to next line and steps into function calls
- **`list (l) [line number]`**: Print out the specified line of source code and 5 above and below
- **`continue (c)`:** run until crash, end of program, or next breakpoint.
- **`info break`:** will list all current breakpoints

Other commands for gdb can be searched online for tutorials and other quick references. If you are on OS X, you must use LLDB instead.

Now that we have run our program through a debugger, let's go into the code and figure out how to fix the errors. An easy way to do this is to follow these steps from industry professional, Zed A. Shaw:

1. You cannot debug code just by looking at the code
2. Repeat the bug with an automated test
    a. This can be done with a simple shell script, basically put the commands you would use into a .sh file and run that file
3. Run the program under a debugger
4. Find the backtrace and go into the code and print out all variables used in the calling functions
5. Once the code has been fixed, add the assert() macro to prevent it
    a. `assert()` makes sure that a statement is true
    b. ex: `assert(pointer != NULL);`

COLLEGE OF INFORMATION SCIENCE & TECHNOLOGY
COMPUTER SCIENCE LEARNING CENTER

This workforce product was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The product was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The U.S. Department of Labor makes no guarantees, warranties, or assurances of any kind, express or implied, with respect to such information, including any information on linked sites and including, but not limited to accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability, or ownership.