CpSc 1111 Lab 6
Debugging With GDB

Goals

This lab will introduce you to gdb, a tool that can be used to debug programs that have run-time errors. You may find this to be a useful tool as you begin writing more complicated programs than the ones you have seen thus far and encounter run-time errors such as "segmentation faults" or "bus errors". After you complete this lab, you should continue to experiment with gdb so that you can become more familiar with it - especially if you have programs that have run-time errors.

gdb Debugger

The gdb debugger is an effective interactive tool that will allow you to run a program, stopping at predefined break points that you set; print the value of variables, lines of code, etc.; continue executing to the next break point, or line by line; and find the statement at which a program suffered a fatal error.

In order to use gdb (or any other debugger) on your programs, you must instruct the compiler to include *debugging symbols* in the executable. Otherwise, the compiler leaves out these symbols to reduce the size of the executable files. With the gcc compiler, the -g switch turns on debugging symbols. In other words, if you have a program called `prog1.c`, you would type the following when compiling:

```
gcc -g -Wall prog1.c
```

Some of the commonly used gdb commands are shown in the table below:

gdb ./a.out	load the program "a.out" in the current working directory and start the debugger
gdb –tui ./a.out	load the program "a.out" in the current working directory and start the debugger; using the **-tui** splits the screen showing the code in the upper half and the gdb prompt in the lower half
break main (or: **b main**)	cause execution to pause at the start of the function "main"
break 32 (or: **b 32**)	cause execution to stop at line 32 (or whatever line number you specify)
run (or: **r**)	start execution of the currently loaded program
r > outputFile.txt	start execution of the currently loaded program, redirecting the output to the file specified (could be useful for debugging programs that produce an image file, e.g a .ppm image)
n (or **next**)	execute the next line of source code
c (or **continue**)	continue without stopping to the next breakpoint, program termination, or error
p x (or **print x**)	print (to the screen) the current value of the variable x (or whatever variable name is specified)
d x (or **display x**)	display the current value of x at each gdb command prompt
q (or **quit**)	quit gdb
r	restart the currently running program using the previous command line

A more in-depth list of commands may be found here:
http://people.cs.clemson.edu/~chochri/Courses/Documents/gdb-basics.pdf or by searching online.

Assignment

In this lab, you will be given a program that compiles but does not run successfully. When you try to run it, you should get the message "Segmentation fault". This and "Bus Error" are the two most common run-time errors. Note that no indication is given as to where the program failed. You can imagine that in a large program, the message is pretty useless in trying to locate the problem.

Once you log in to your account, navigate to a directory where you will do this week's lab work. Then type the following command:

```
cp  /group/course/cpsc111/public_html/F18Labs/lab6/* .
```

Don't forget the dot at the end – this says to copy all the files from that specified path into your current working directory.

Once you copy the files, type `ls` and you should see the following files: `example.c`, `broken.c` and `questions.txt`. Your lab assignment is to do the following:

1. compile and try to run `example.c`
2. follow the steps given on the next page where it says "Trying Out gdb" to try to find what is causing the seg fault in `example.c`
3. compile and try to run `broken.c`
4. use the gdb debugger to try to find where the seg fault occurs
5. after a bit of detective work with the use of gdb, answer the questions in the file `questions.txt`
6. fix the error(s) in `broken.c`
7. Show your TA that you completed the assignment. Then submit both the fixed `broken.c` program and the completed/updated `questions.txt` files to the handin page: http://handin.cs.clemson.edu ***Don't forget to always check on the handin page that your submission worked. You can go to your bucket to see what is there.***

Grading Rubric

If your program does not compile on our Unix machines or your assignment was not submitted on time, then you will receive a grade of zero for this assignment. Otherwise, points for this lab assignment will be earned based on the following criteria:

Fixed `broken.c` file	55	(functionality: fixed the problem and now the program works)
Formatting	10	(5 for added header information at top: name, course & semester, lab #; 5 for other formatting)
Answers in `questions.txt`	35	(5 points per question)

Trying Out gdb

The following is the code contained in `example.c`:

```
/*  program 17.4 page 398  */

#include <stdio.h>

int main(void)
{
   const int data[5] = {1, 2, 3, 4, 5};
   int i, sum;

   for (i = 0; i >= 0; ++i) {
      sum += data[i];
   }

   printf("sum = %i \n", sum);

   return 0;
}
```

1. compile: `gcc -g example.c`
2. try to run the program the regular way without gdb: `a.out`
3. it seg faults
4. start it with gdb: `gdb -tui ./a.out`
5. `list main`
6. `break main`
7. `run` (this starts running the program up to the break point at `main`)
8. `p data` (the debugger stops at the line preceding the current line of code shown, so when you try to print `data`, at that point, the array called `data` has not been initialized yet; in other words, the line of code that is highlighted is the line that *will be executed next*).
9. `next`
10. `p data` (now the values for `data` are there)
11. `next`
12. `next`
13. `next`
14. `p i`
15. `next`
16. `p i`
17. `next`
18. `p i` (see what's happening?)
19. `continue` the program continues until it gets to the next break, or, as in this case, the seg fault since no other break was set
20. this is what it shows:
    ```
    Program received signal SIGSEGV, Segmentation fault.
    0x000000000040052d in main () at example.c:11
    ```
    and the line of code that is highlighted in the upper box is:
    ```
    > 11                    sum += data[i];
    ```
21. `list` this will list 10 lines near where the seg fault occurred (which is already in the upper box)
22. `p i` this will show what value `i` has at this point (what is `i` on your screen and why?)
23. `quit` will quit the gdb debugger and give you back a regular prompt