Individual Lab 0 - GNU Debugger (GDB)

Outcomes:
The goal of this lab is for you to become familiar with the basics of examining a program through the GNU Debugger (GDB).
The lab will cover:
Compiling a program so that GDB can examine it
Loading a compiled program into GDB
Executing the program in GDB
Exiting GDB
Use this file: simple.cpp
Proper Program Compilation:
In order for GDB to have access to your program's source code, variables, data structures, and functions it requires that program's symbol table to be constructed. The symbol table is constructed during compilation by using the –g flag when compiling. Try compiling simple.cppusing the –g flag like this:

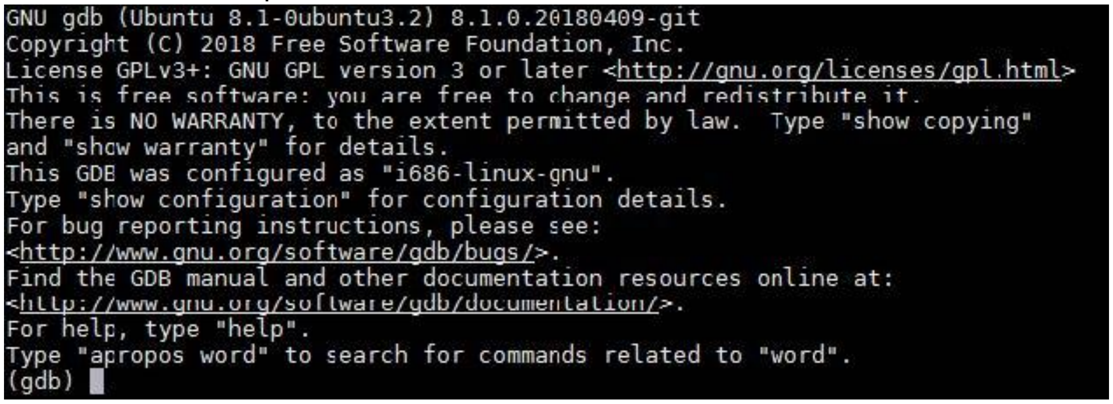
```
g++ -g simple.cpp
```

You should now have an a.out file just like normal. This file can now be loaded into GDB.

Loading a Program into GDB:
Once you have a properly compiled program, you can examine that program in GDB. To run GDB you will type:

```
gdb
```

You should see output similar to this:



```
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) 
```

However, you program is still not loaded into GDB. To load your executable into GDB you will use the file command.
Once GDB is running type file and then the name of your executable like this:

```
file a.out
```

If you have not created the symbol table using the –g command, GDB will tell you "no debugging symbols
found". If you have created the symbol table correctly, then it will not provide any warning.
Alternatively, you can pass your executable into GDB at runtime by specifying the executable name as a runtime argument like this:

```
gdb a.out
```
Both methods provide the same outcome of having your executable be ready to run in GDB.
Executing a Program in GDB:
To execute a program in GDB, you will use the run command. The program will execute from start to finish just as it would outside of GDB. Any output is displayed inside GDB, and required text for any prompts can be typed into GDB. Once the program has finished, GDB will tell you that the inferior process has exited. Go ahead and execute your currently loaded program in GDB and interact with it as requested by typing:
```
run
```
You can repeatedly execute a program as many times as you need in GDB without restarting GDB.

Exiting GDB:
Once you are finished executing your programs, you can exit GDB using the quit command. Go ahead and exit GDB now by typing:
```
quit
```
You should explore using various gdb commands we discussed in class. You might even intentionally introduce some errors into the given code and then use gdb to debug them so you can see what the look like in the tool.
When you have completed your explorations, please create a 1-2 paragraph report for submission as a pdf document. You should address the following topics and questions.
What are your general impressions of gdb as a tool? Do you think it will be helpful to you in debugging more complex programs that you will be working on?
What did you like best?
What did you like least?
Provide a short narrative describing your explorations and your experiments with introducing and debugging errors.