CS 3100 Lab 4: Memory Allocation
Diagnosing Memory Errors

In Lab4, you will write a handful of very small C programs, each with a specific error purposely included. You will then execute one or more tools that will assist you in diagnosing such (unintentional) errors in the future. The tools you will be introduced to are `valgrind` and `gdb`.

Learning Objectives

- Install valgrind
- Write five small program, each with an intentional bug
- Run valgrind and gdb to see how these tools help you identify each type of bug

Assignment

Begin by installing the tools you will need for this assignment:

```
mkdir -p ~/cs3100/lab4
cd ~/cs3100/lab4
sudo apt-get update # refresh the apt-get database
sudo apt-get install valgrind
scp USER@icarus.cs.weber.edu:/var/classes/cs3100/lab4/Makefile .
```

The next step is to create the source files needed for this assignment. These are the files you will create from scratch:

```
null.c nofree.c bounds.c datagone.c freebad.c
```

The `Makefile` you downloaded from icarus is to assist you in building, testing and generating the files needed to upload to Canvas. This particular makefile will also create a zip file for uploading to Canvas, to avoid having to upload a dozen files one at a time. You are welcome.

The `Makefile` has these commands that facilitate your workflow:

`make:` Prints a list of possible `make` commands.

`make null, make nofree, make bounds, make datagone or make freebad:` compiles the requested executable from the associated `.c` file and generates an associated `.log` file containing the output from `valgrind`. Suggestion: build each executable one at time and get it working before moving on to the next. The `Makefile` also displays the output from `valgrind` for your personal inspection.

`make nullgdb:` runs the null program under gdb, the GNU debugger, demonstrating what happens when a program crashes and what information is displayed. The output from gdb is saved in `nullgdb.log`. Of course, null.c must compile and crash before this step is meaningful.

`make canvas:` runs all of the make commands above, then creates **`lab4.zip`** file suitable for submission to Canvas. This zip file contains the 5 `.c` files You would then upload JUST THIS .zip file to Canvas to submit the assignment for grading.

`make clean:` deletes all of the `.log` files, all of the executable files. Useful if you want to clean up and start over. Does not delete any `.c` files.

Here are the files you are to create:

null.c
Write a small program called `null` that creates a pointer to an integer, sets the pointer to NULL, and then attempts to set the integer to zero through the pointer. This program should crash when you run it. Compile it with `make null`, which will compile `null.c`, creating `null` and running `valgrind`, a memory utility that will identify potential memory bugs in your code. Notice what it tells you about `null`. Run the program manually to see if it crashes.

After you have successfully written a crashing `null` program, type `gdb ./null`, then `run`, then `quit`, then `y`. These commands will execute `null` under the GNU debugger. The debugger will identify the line in your program that caused the abnormal termination of `null`.

nofree.c
Write a small program called `nofree` that allocates 1024 bytes of memory using `malloc()`, then exits the program without calling `free()`. Compile it with `make nofree`, which will compile `nofree.c`, creating `nofree` and running `valgrind`. Review the `valgrind` report. Run the program manually to see if it crashes.

bounds.c
Write a small program called `bounds` that allocates memory for 100 integers (`sizeof(int) * 100`) using `malloc()` and saves the pointer from `malloc()` in a pointer variable called `ptr`. Then set `ptr[100] = 0` and see what happens when you run this program. `make bounds` will compile your code and run `valgrind`. Review the `valgrind` report. Run the program manually to see if it crashes.

datagone.c
Write a small program called `datagone` that allocates an array of integers with `malloc()` as in `bounds`, saves the pointer in a pointer variable called `ptr`, frees the storage with `free()`, then prints the 51st integer with `printf("%d\n", ptr[50]);`. `make datagone` will compile your code and run `valgrind`. Review the `valgrind` report. Run the program manually to see if it crashes.

freebad.c
Write a small program called `freebad` that allocates an array of integers with `malloc()` as in `bounds`, saves the pointer in a pointer variable called `ptr`, then frees the storage with `free(ptr+50)`, which instructs `free()` to free memory from the middle of the allocated region. `make freebad` will compile your code and run `valgrind`. Review the `valgrind` report. Run the program manually to see if it crashes.

Hints and Suggestions

- Please write these programs as simply as possible. Do not add any prompts, output or any file I/O except as specified for each program. Actually, you may add whatever embellishments you wish, but you will receive a zero for the assignment.
- You will need `#include <stdlib.h>` for any programs that call `malloc()` or `free()`. You will need `#include <stdio.h>` if you call `printf()`.
- Execute the command `make canvas` to create `lab4.zip`, and upload JUST THIS FILE to Canvas for grading as you turn in this assignment.

Grading

I use Cucumber scripts to help grade programs. Please note the following:

- The cucumber tests will fail if you do not follow the naming instructions, so please be sure that the programs are named correctly. As long as the source file is named correctly, and you have to upload it to Canvas more than once, the displayed name may have a number after it. That is just for display purposes and your file should still be named correctly internally.
- Upload the `lab4.zip` file for this assignment to Canvas when you are ready for me to grade your work.
- The cucumber scripts will run `valgrind` and create the associated log files. You will not upload your log files to Canvas.
- It is not necessary to upload any files to icarus for this assignment.

NOTE: Just because cucumber reports a certain number of points for your submission of the assignment, the instructor reserves the right to inspect your code, run your submission with additional tools and manually grade your assignment. The decision of the instructor is final. Test your code thoroughly to ensure all requirements are fully met.

Here is how you earn points for this assignment:

FEATURES	POINTS
Must-Have Features	
Only the following file is uploaded to Canvas: `lab4.zip`	20
`lab4.zip` contains only the following files: `null.c nofree.c bounds.c datagone.c freebad.c`	20
`null.c nofree.c bounds.c datagone.c freebad.c` compile	20
`null` crashes with a segmentation fault when run	20
`null.log` contains "Invalid write of size 4"	20
`nofree.log` contains "1,024 bytes in 1 blocks are definitely lost"	20

`bounds.log` contains "Invalid write of size 4"	20
`datagone.log` contains "Invalid read of size 4"	20
`freebad.log` contains "Invalid free()"	20
`nullgdb.log` contains "Program received signal SIGSEV"	20
Grand Total	200