CS240 Lab 6
GDB Practice

You should use your Linux environment (VSCode) for this assignment, and enter the commands given in the Terminal.

If you did the practice problems for the Pointers assignment, you should already have the *cmemory* repository.

This exercise is similar to part 8 from the practice problems (but even if you did that problem, please repeat here for the lab assignment).

If you do not already have the repository, get it with:

$ ***cs240 start cmemory***

The file *strings3.c* from the *cmemory* repository for the pointers assignment contains the following code:

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char** argv) {
  char** commandA = (char**)malloc( 3 * sizeof(char*) );

  commandA[0] = "emacs";
  commandA[1] = "strings.c";
  commandA[2] = NULL;

  free(commandA);

  char** commandB = (char**)malloc( 3 * sizeof(char*) );

  commandB[0] = "ls";
  commandB[1] = "cs240-pointers";
  commandB[2] = NULL;

  commandA[1] = "uh oh";
  printf("A: %s %s\n", commandA[0], commandA[1]);
  printf("B: %s %s\n", commandB[0], commandB[1]);

  free(commandB);

  return 0;
}
```

Compile the program with the following command:

$ **gcc -Wall --std=c99 -g -O -o strings3 strings3.c**

Note: Using the **–g** option creates debugging information for use in **gdb**.

Use **gdb** to examine memory and understand the connection between the diagram and real memory.

$ **gdb ./strings3**

Set a breakpoint at the beginning of the program:

(gdb) **break main**

1. What address in memory does ***main*** begin?

Run the program:

```
(gdb) run
```

It will hit the breakpoint, at the beginning of *main*.

Execute a single step of the program:

```
(gdb) step
```

```
char** commandA = (char**)malloc( 3 * sizeof(char*) );
```

It displays the next instruction to be executed after taking the step, which will be to allocate memory for commandA, an array of 3 pointers.

Take another step to perform the allocation, and then examine (x) the value of **commandA**

```
(gdb) step
```

```
(gdb) x commandA
```

2. What address in memory does **commandA** refer to?

Take three more steps to execute the statements that initialize the **commandA** array:

```
commandA[0] = "emacs";
```

```
(gdb) step
```

```
commandA[1] = "strings.c";
```

```
(gdb) step
```

```
commandA[2] = NULL;
```

```
(gdb) step
```

Display the contents of the **commandA** array (**/3a** means display 3 addresses or pointers to strings):

```
(gdb) x /3a commandA
```

3.. Why is the last address displayed **0x0**?

Display the strings:

```
(gdb) x  /s *commandA

(gdb) x /s *(commandA + 1)
```

Take a step to free (deallocate) **commandA:**

```
free(commandA)
```

```
(gdb) step
```

```
char** commandB = (char**)malloc( 3 * sizeof(char*) );
```

The next instruction will allocate space on the heap for another array of 3 pointers.

Take another step to perform the allocation, and then examine the value of **commandB:**

```
(gdb) step

(gdb) x commandB
```

Also, examine **commandA** again:

```
(gdb) x commandA
```

4. What do you notice about the two values?

5. Why do you think this happened?

Take three more steps to execute the statements that initialize the **commandB** array:

```
commandB[0] = "ls";
```

```
(gdb) step
```

```
commandB[1] = "cs240-pointers";
```

```
(gdb) step
```

```
commandB[2] = NULL;
```

```
(gdb) step
```

Display the contents of the **commandB** array:

```
(gdb) x /3a commandB
```

Now execute the next instruction:

```
        commandA[1] = "uh oh";
```

```
(gdb) step
```

And again display the contents of the **command** array:

```
(gdb) x /3a commandB
```

6. Explain why **commandB** has changed.

Complete execution of the program:

```
        printf("A: %s %s\n", commandA[0], commandA[1])
```

```
(gdb) step
```

```
        printf("B: %s %s\n", commandB[0], commandB[1]);
```

```
(gdb) step
```

7. Explain the results of the print statements.

8. How could you modify the program to prevent this incorrect output?

Quit out of **gdb:**

```
(gdb) quit
```