Question 1.

In this question, you will be practicing using postmortem debugging in VS Code as well as practicing using `valgrind`.

Part (a)

Write code that accesses a memory location it is not supposed to be accessing ("segmentation fault"). If the program does not crash, modify it until it does crash. Run the code in VS Code, and make sure that you can see in VS Code which line caused the segmentation fault, and that you see in VS Code the values of all the local variables just before the segmentation fault.

Part (b)

On ECF, use `valgrind` on the program that produces the seg fault in order to accomplish the same thing as before.

Part (c)

Write a program that reads text from a file named `a.txt`. Do not use `fclose(fp);` at the end. On ECF, run `valgrind` on the program to observe that there is a memory leak. Show that this can be fixed by using `fclose` after you've finished using `FILE *fp`.

Part (d)

If `fopen` cannot open the file when you use `FILE *fp = fopen(filename)`, `fopen` will return `NULL`. Modify your program from before to display an error message if the file cannot be opened.

Question 2.

The function `atof` from `stdlib.h` converts an input string to a `double`, and the function `atoi` converts an input string to an `int`.

Here is a sample implementation of `atoi`:

```
int my_atoi(const char *str)
{
  int i = 0;
  int sign = 1;
  if(str[i] == "-"){
    sign = -1;
    i++;
  }
  int result = 0;
  while(str[i] >= '0' && str[i] <= '9'){
    result = result * 10 + (str[i] - '0');
    i++;
  }
  return result * sign;
}
```

In C, **without** using `atof` or any of the `scanf` family of functions, write a program that reads a text file that lists several (not necessarily 3) constants formatted similarly to the following:

```
PI = 3.14
G = 0.0000000000667
g = 9.8
```

and outputs the sum of all the constants in the file. For example, the output in this example would be (3.14+0.0000000000667+9.8) 12.9400000000667.

The following will be useful:

`https://stackoverflow.com/questions/16839658/printf-width-specifier-to-maintain-precision-of-f`

Create a text file to test your program and then test it.

Question 3.

In class, we used the following C code:

```
typedef struct student1{
  char name[3];
  int age;
} student1;


void change_name1_wrong(student1 s)
{
  s.name[0] = 'b';
}

void change_name1_right_a(student1 *p_s)
{
  p_s->name[0] = 'b';
}

void change_name1_right_b(student1 *p_s)
{
  strcpy(p_s->name, "b");
}
```

Suppose that in Python, we store a student with the name `"ab"` and the age `20` as `["ab", 20]`.

Explain why each of the C functions above does not have an equivalent in Python (without using `ctypes`).

Write a Python function that changes the name of a student to a new name that's passed to the function.

Now, write a C function that works the same way as the Python function you wrote.

For both the C function and the Python function, write out the memory diagrams.