Courtney Maynard
Part One

1) After compiling the hello world program, called 1.c, and running the 1.out file, the size of the executable is 16697. The sizes of the different segments are as follows:

   Text: 1569
   Data: 600
   Bss: 8

```
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ ls -l 1.out
 -rwxr-xr-x 1 cbmaynard24 temp 16697 Oct 30 19:54 1.out
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ size 1.out
   text    data     bss     dec     hex filename
   1569     600       8    2177     881 1.out
```

2) I copied the file from the previous step, renamed it to 2.c, and added a declaration of an array with 1000 ints. After recompiling and running 2.out, the size of the executable is now 16728. The sizes of the different segments are as follows:

   Text: 1569
   Data: 600
   Bss: 432

   The difference is in the bss segment.

```
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ ls -l 2.out
 -rwxr-xr-x 1 cbmaynard24 temp 16728 Oct 30 20:40 2.out
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ size 2.out
   text    data     bss     dec     hex filename
   1569     600     432    2601     a29 2.out
```

3) I copied the file from the previous step, renamed it to 3.c, and added a value to the declaration of the array. As predicted in the textbook, the bss segment decreased to its original size and the size of the data segment increased. The size of the executable is now 17144. The sizes of the different segments are as follows:

   Text: 1569
   Data: 1016
   Bss: 8

```
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ ls -l 3.out
 -rwxr-xr-x 1 cbmaynard24 temp 17144 Oct 30 20:52 3.out
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ size 3.out
   text    data     bss     dec     hex filename
   1569    1016       8    2593     a21 3.out
```

4) I copied the file from the previous step, renamed it to 4.c, and created a function with a big array declared locally to it. Then, I declared a second big array with an initial value.The size of the executable is now 17232. The sizes of the different segments are as follows:

   Text: 1846
   Data: 1024
   Bss: 8

```
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ ls -l 4.out
 -rwxr-xr-x 1 cbmaynard24 temp 17232 Oct 30 21:09 4.out
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ size 4.out
   text    data     bss     dec     hex filename
   1846    1024       8    2878     b3e 4.out
```

Yes, the data defined locally is stored inside the executable. This is evident because when I added the two arrays inside of the practice_func function, the size of the data segment increased from 1016 to 1024. Inside of a local function, it does not make a difference in the data segment size whether the array is initialized or not. When running the size command on the a.out of 4.c with the initialized array commented out, the data segment size was still 1024, however, the size of the text segment decreased.

```
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ ls
 1.c  1.out  2.c  2.out  3.c  3.out  4.c  4.out  a.out
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ size a.out
    text    data     bss     dec     hex filename
    1814    1024       8    2846     b1e a.out
```

5a) When compiling for debugging, the size of the executable is now 19968. The sizes of the different segments are as follows:

Text: 1846

Data: 1024

Bss: 8

```
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ ls -l 5d.out
 -rwxr-xr-x 1 cbmaynard24 temp 19968 Oct 30 21:21 5d.out
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ size 5d.out
   text    data     bss     dec     hex filename
   1846    1024       8    2878     b3e 5d.out
```

There are no changes that occur in terms of file size and segment sizes.

5b) When compiling for maximum optimization, the size of the executable is now 17232. The sizes of the different segments are as follows:

Text: 1846

Data: 1024

Bss: 8

```
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ size 5o.out
    text    data     bss     dec     hex filename
    1846    1024       8    2878     b3e 5o.out
[cbmaynard24@th121-3:~/lbp/life/ASSIGNMENT_08/Part_I$ ls -l 5o.out
 -rwxr-xr-x 1 cbmaynard24 temp 17232 Oct 30 21:27 5o.out
```

The change occurred in the file size; the file size decreased when compiling for optimization, even though the segment sizes stayed the same.

Part Two

1) The approximate location of the stack on my system is at 0x7fff1e9d924.

```
cbmaynard24@th121-13:~/lbp/life/ASSIGNMENT_08/Part_II$ ./stack_hack_1.out
The stack top is near 0x7ffff1e9d924
```

2) I declared several variables that will be placed in the data segment. Additionally, I created a pointer to allocate heapspace so that I could determine the address of the heap as well. Lastly, main is part of the text segment so I was able to find the address for that as well. The results are as follows:

   The stack top is now at the location 0x7ffc5b7241d0.
   The location of an integer variable, x, is 0x7ffc5b7241d4
   The location of a second integer variable, y, is 0x7ffc5b7241d8
   The location of a third integer variable, z, is 0x7ffc5b7241dc

   It is clear just from looking at the locations of the variables that the stack grows downwards. Additionally,

   The location of main is 0x5581230zz189
   The location of the heap is 0x5581237b62a0

```
cbmaynard24@th121-13:~/lbp/life/ASSIGNMENT_08/Part_II$ ./stack_hack_2.out
The stack top is near 0x7ffc5b7241d0

The value of z is 60

The location of x is 0x7ffc5b7241d4

The location of y is 0x7ffc5b7241d8

The location of z is 0x7ffc5b7241dc

The location of main, part of the text segment, is 0x5581230aa189

The location of w, which provides heap space, is 0x5581237b62a0
```

3) After adding local arrays inside of a function and calling that function in order to make the stack grow, the top of the stack is now near/at the location 0x7ffd28bdb42c.

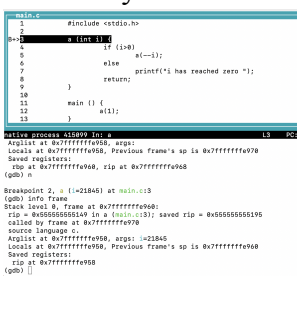
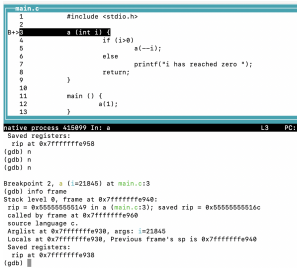
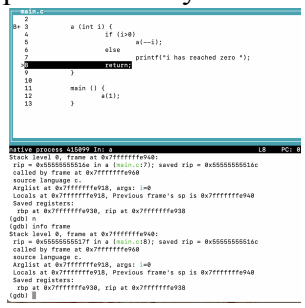
```
cbmaynard24@th121-13:~/lbp/life/ASSIGNMENT_08/Part_II$ ./stack_hack_3.out
The stack top is near 0x7ffd28bdb42c

The value of z is 60

The location of x is 0x7ffd28bdb430

The location of y is 0x7ffd28bdb434

The location of z is 0x7ffd28bdb438

The location of main, part of the text segment, is 0x561369983189

The location of w, which provides heap space, is 0x56136b2722a0

The value outputted from the function array_func_sh is 120
```

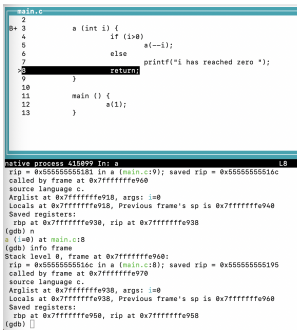
Part Three

1)

a called by main	local vars - none arguments - int i, with a value of 1 prev frame - main return address - line 10
a called by a	local vars - none arguments - int i, with a value of 0 prev frame - first call to a return address - line 3
printf called by a	local vars - none arguments - string “i has reached zero” prev frame - 2nd call too a return address - line 5
return called by a	local vars - none arguments - none prev frame - printf return address - line 10

2) I compiled the program for debugging with the **gcc -g** command to add breakpoints for debugging. Then, I added breakpoints at the beginning of the a function so I could see what the frame is like whenever a is called. After doing this, I stepped through the program using the **n** command for next, and executing **info frame** when I wanted to see where the different parts of the program were on the stack frame. The resulting table below is the table from problem one, but with specific addresses for the location of each element according to the output of **info frame** at the corresponding step.

a called by main 	local vars - 0x7fffffffe950 arguments - 0x7fffffffe950 prev frame -0x7fffffffe960 return address (aka saved rip) - 0x7fffffffe958
a called by a 	local vars - 0x7fffffffe930 arguments - 0x7fffffffe930 prev frame - 0x7fffffffe940 return address -0x7fffffffe938
printf called by a 	local vars - 0x7fffffffe918 arguments - 0x7fffffffe918, i = 0 prev frame - 0x7fffffffe940 return address - 0x7fffffffe938
return called by a	local vars - 0x7fffffffe938 arguments - 0x7fffffffe938, i =0 prev frame - 0x7fffffffe960



return address - 0x7fffffffe958