Part I

2.

a. The size of 1.out executable is 16696 bytes, or 17K. It has 1566 bytes of text, 600 of data, and 8 of bss.

Command used:

```
gcc -o 1.out 1.c
ls -l 1.out
size 1.out
```

b. The size of 2.out executable is 16728 bytes, or 17K. It has 1566 bytes of text, 600 of data, and 4032 of bss.

Difference: the size of bss increased due to a declaration of an array. Text segment also increases due to more texts are written.

Command used:

```
gcc -o 2.out 2.c
ls -l 2.out
size 2.out
```

c. The size of 3.out executable is 20744 bytes, or 17K. It has 1566 bytes of text,4616 of data, and 8 of bss.

Difference: the size of bss decreased with data increased due to initializing an array. Text segment also increases due to more texts are written.

Command used:

```
gcc -o 3.out 3.c
ls -l 3.out
size 3.out
```

d. When an array without initialization is declared, 4.out has 1723 of text, 4624 of data. And 8 of bss. When the second array with initialization is declared, it changes to 1755 of text, but the data and bss segment stay the same. The total size of 4.out is now 20792 bytes.

The locally defined data will not be stored in the data segment, and it doesn’t make any difference if it’s initialized.

Command used:

```
gcc -o 4.out 4.c
ls -l 4.out
size 4.out
```

e. When compiled for debugging, the overall size of the executable increases to 23408 bytes. The segments don’t change.

Command used:

```
gcc 5.c -g -o 5d.out
ls -l 5d.out
size 5d.out
```

f. After the maximum optimization (using O3), the total file size grows to 24088 bytes. The text segment falls back to 1558 and data to 4616.

Command used:

```
gcc 5.c -g -O3 -o 5d.out
ls -l 5o.out
size 5o.out
```

Part II

2.

a. Command used:

```
gcc stack_hack_1.c -o stack_hack_1.out
./stack_hack_1.out
```

(output)

```
The stack top is near 0x7fffe8e1c7a4
```

b. Command used:

```
gcc stack_hack_2.c -o stack_hack_2.out
./stack_hack_2.out
```

(output)

```
The stack top is near 0x7ffdae5073fc
The location of global_data in the initialized data segment is 0x5593c48cb010
The location of text_seg in the text segment is 0x5593c48c8020
The location of p in the heap segment is 0x5593c496b2a0
```

I declared a global integer in the data segment, a const in text segment, and a pointer pointing to the result of malloc(128) in the heap segment.

c. Command used:

```
gcc stack_hack_3.c -o stack_hack_3.out
./stack_hack_3.out
```

(output)

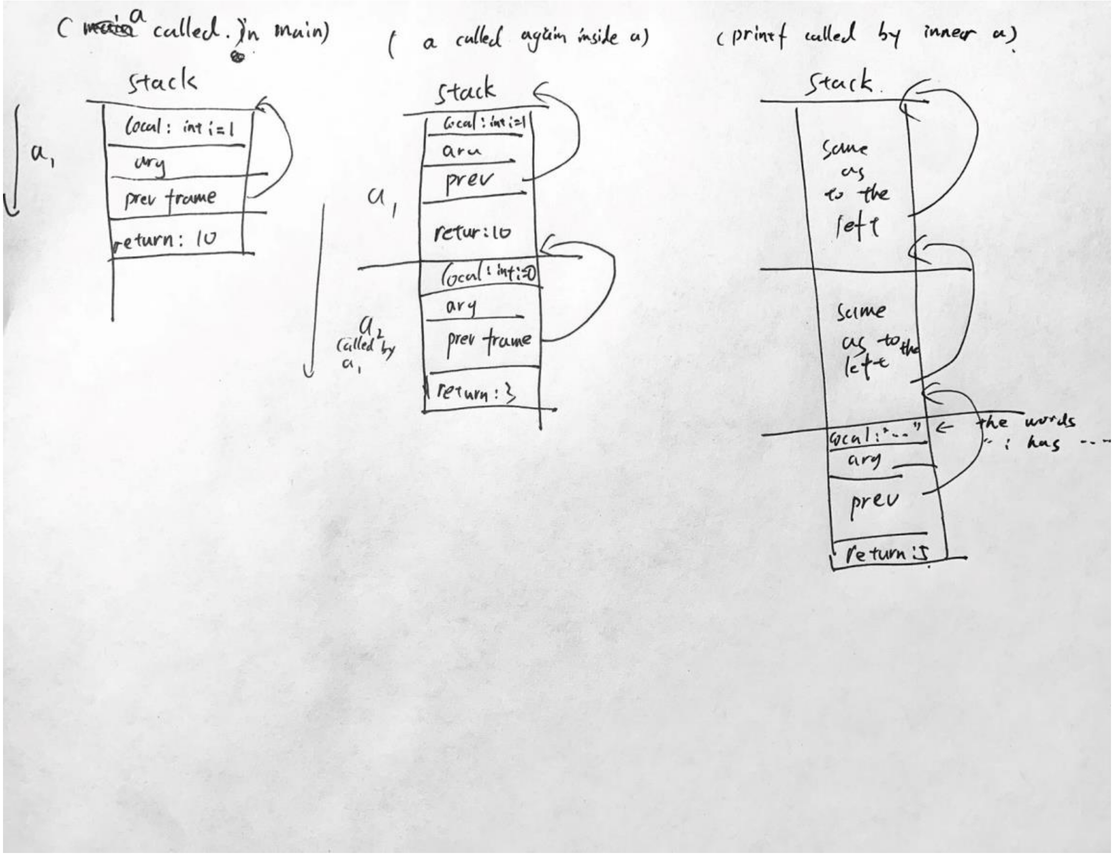
```
The stack top is now near
0x7ffdb3c74c90
The location of global_data in the initialized data segment is 0x55dc911a9010
The location of text_seg in the text segment is 0x55dc911a6020
The location of p in the heap segment is 0x55dc917612a0
```

I declared a large local array and called a function to change local variables. This should grow the stack as those local variables are stored in the stack/

Part III

2.

a.



b. Command used: gcc -g main.c

in gdb: set the end line of a as breakpoint

and do info frame

The actual frame on Linux substitutes the address with actual memory address.

For example:

```
Stack level 0, frame at 0x7fffffffe930:
rip = 0x55555555516e in a (main.c:6); saved rip = 0x55555555516c
called by frame at
0x7fffffffe950
source language c.
Arglist at 0x7fffffffe908, args: i=0
Locals at 0x7fffffffe908, Previous frame's sp is 0x7fffffffe930
Saved
registers:
rbp at 0x7fffffffe920, rip at 0x7fffffffe928
```