1.1
Since variables are copied when they are passed to functions (passed by value), so only the copy is modified in the function call to increment_with_value. The original will remain unchanged and main therefore returns 23.

1.2
Unless they are static, local variables will not show up when listing object file symbols. This is because they are defined on the stack at runtime.

1.3
a.
13106

b.
foo is an integer and takes up 4 bytes (on my machine, at least). foo is a local variable and is assigned to the stack at the start of the program. Then s is allocated with 12 bytes on the stack, one for each character.
t is then initialized with 15 bytes and copied to s. Since s can only hold 12 of these bytes, we get an overwrite of several bytes in foo, leading to the value above.

	Value
foo	00000000
foo	00000000
foo	00000000
foo	00000000
s[11]	-
s[10]	-
s[9]	-
s[8]	-
s[7]	-
s[6]	-
s[5]	-
s[4]	-
s[3]	-
s[2]	-
s[1]	-
s[0]	-

	Value	
foo	00000000	
foo	00000000	‘NUL’
foo	00110011	‘3’
foo	00110010	‘2’
s[11]	00110001	‘1’
s[10]	00110000	‘0’
s[9]	00111001	‘9’
s[8]	00111000	‘8’
s[7]	00110111	‘7’
s[6]	00110110	‘6’
s[5]	00110101	‘5’
s[4]	00110100	‘4’
s[3]	00110011	‘3’
s[2]	00110010	‘2’
s[1]	00110001	‘1’
s[0]	00110000	‘0’

Address
0xFF
0xFE
0xFD
0xFC
0xFB
0xFA
0xF9
0xF8
0xF7
0xF6
0xF5
0xF4
0xF3
0xF2
0xF1
0xF0

c.
In gcc version 9.3.0, you have to provide the compiler option “-fno-stack-protector” to even be able to cause a stack buffer overflow.
Maybe the compiler option in question is “-fstack-protector”? It does not seem to exist in newer versions of gcc.

d.
If foo was static, foo would not be stored on the stack and hence would not be overwritten. So in this specific instance, it would solve our problem. However, t would still be too large to fit in the array s and we will still get a stack smashing error, unless we disable stack protector.

1.4

a.
rec() is stored in the text segment.
c and counter is initialized to 0 at compile time and is located in the bss.
d is initialized at compile time and static, it is therefore located in the data segment.
If rec had returned, a would be on the stack and so would parameter a.

b.
We get a stack overflow because of infinite recursion and hence a segmentation fault when the program tries to write to unavailable memory. If you add local variable char array[1000] to function rec, this would happen faster.