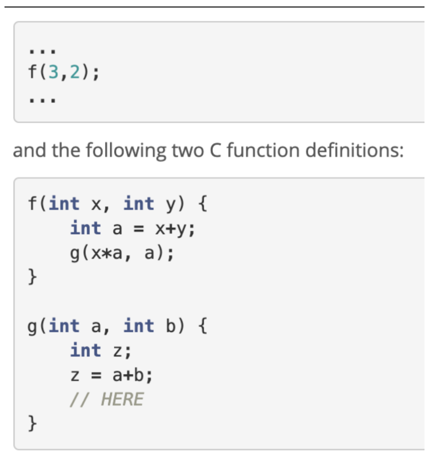
Exercise 1 "Draw" a stack

Draw a picture of the stack at the point where the program reaches the comment "// HERE". You don't have to show any stack content pushed before the arguments to f().



```
...
f(3,2);
...
```

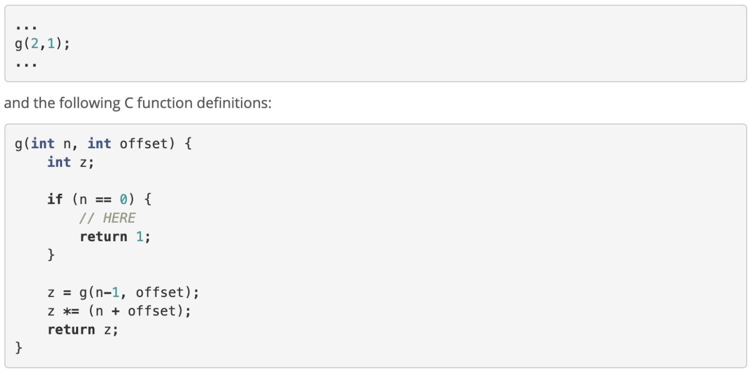
and the following two C function definitions:

```
f(int x, int y) {
    int a = x+y;
    g(x*a, a);
}

g(int a, int b) {
    int z;
    z = a+b;
    // HERE
}
```

y = 2
x = 3
return @ to main
saved EBP
a = 5
b = 5
a = 15
return @ to f
saved EBP
z = ~~?~~ 20

Exercise 2: “Draw” another stack

Draw a picture of the stack at the point where the program reaches the comment “// HERE”. You don’t have to show any stack content pushed before the arguments to g().



```
...
g(2,1);
...
```

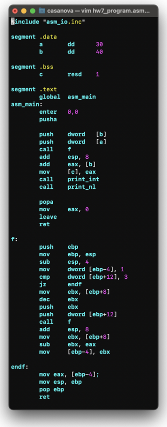
and the following C function definitions:

```
g(int n, int offset) {
    int z;

    if (n == 0) {
        // HERE
        return 1;
    }

    z = g(n-1, offset);
    z *= (n + offset);
    return z;
}
```

offset = 1
n = 2
return @ to main
saved EBP
z = ?
offset = 1
n = 1
return @ to g
saved EBP
z = ?
offset = 1
n = 0
return @ to g
saved EBP
z = ?

Exercise 3: Reverse-engineering

Write a C translation of the NASM program below, sticking to the assembly code as much as possible. Use single-letter variable names for function parameters (e.g., int foo(int x, int y)) and for local variables within function (e.g., int z) instead of using x86 register names (in fact registers should never appear in your translation). It is expected that your C code is much shorter than the assembly code.
Hint: This program outputs the number -217 (but it is not necessary to know this to do the translation). -217


casanova — vim hw7_program.asm...

```
%include "asm_io.inc"

segment .data
        a       dd      30
        b       dd      40

segment .bss
        c       resd    1

segment .text
        global  asm_main
asm_main:
        enter   0,0
        pusha

        push    dword   [b]
        push    dword   [a]
        call    f
        add     esp, 8
        add     eax, [b]
        mov     [c], eax
        call    print_int
        call    print_nl

        popa
        mov     eax, 0
        leave
        ret

f:
        push    ebp
        mov     ebp, esp
        sub     esp, 4
        mov     dword [ebp-4], 1
        cmp     dword [ebp+12], 3
        jz      endf
        mov     ebx, [ebp+8]
        dec     ebx
        push    ebx
        push    dword [ebp+12]
        call    f
        add     esp, 8
        mov     ebx, [ebp+8]
        sub     ebx, eax
        mov     [ebp-4], ebx

endf:
        mov eax, [ebp-4];
        mov esp, ebp
        pop ebp
        ret
```

```
#include <stdio.h>
    //helper function f
    //takes in @param a and b
    //returns variable x
    int f(int a, int b) {
    int x = 1;
    if (b - 3 == 0) {
        return x;
        }
        x = a - f(b,a-1);
        return x;
    }
    // int main function
    int main(void) {
        int a = 30;
        int b = 40;
        int c = f(a,b) + b;
        printf("%d\n", c);
        return 0;
    }
```