2. Memory layout (10 points)

Consider the following C function. Draw a picture of the state of the memory at the marked position. The picture should contain represent-1ations:

(a) of the **stack**: RSP, RBP of caller, current RBP, return address, function arguments, local variables
(b) of the **heap**: the allocated space(s), and the break
(c) of the register contents (symbolic name, not value)

You can assume that the C compiler will produce un-optimized machine code, and that the memory manager used by the C runtime library is similar to the one presented in class (i.e., uses a header for allocated blocks). **High addresses** are at the top of the paper, **low addresses** at the bottom – Take care of this for **all** parts of the drawing (you can use the boxes below)!

You don't have to provide specific addresses, only draw "boxes" (and arrows) and indicate what they are, but take care **where** the **arrows** point **to** (start, end, or middle of a block)! You don't have to write content (=values) into the memory locations.

You **must** assume that in this exemplary case the memory allocation succeeds and the parameter "array" was reserved on the heap by the procedure calling this function.

The drawing below conforms to the **(MARK which one you use - empty → 0 points!)**

0 C calling convention (C part of lecture)

0 AMD64 calling convention (Assembler part of lecture; use registers **only** if possible at all)

```
#define DEBUG

int play_with_array(int number, int array[20]) {
  int i, a = number + array[0];
  int b = array[19] - number;
#ifdef DEBUG
  printf("a=%d, b=%d\n", a, b);
#endif
  int *sumptr= (int *)malloc(sizeof(int));
  for(i=0;i<20;i++)
    *sumptr = *sumptr + *(array + i);
  printf("sum=%d\n", *sumptr);
    /*** What do stack and heap look like at this point? */
```