Exercise: Stack Frame for a Function Invocation

The stack frame for a function invocation contains arguments (in reverse order), return address, frame pointer of caller, and local variables. Remember the stack grows towards the low address.

```
void func(int a, char b, int c) {
   short x = 5;
   char d[] = "ABC";
   char e[6];
   ...
}
```

An `int` type requires 4 bytes of memory, a `short` type takes 2 bytes of memory, and a `char` type takes 1 byte of memory. Return address (RA) and previous frame pointer (PFP) fields take 4 bytes each. Assume that the storage for local variables is allocated in the order of declaration in the source code and the stack protector/guard is not enabled. Assume the program is run on a little-endian machine.

Show where the storage/value for `a`, `b`, `c`, `x`, `d`, `e`, RA and PFP is allocated/stored in the stack frame. **Show where the current frame pointer during the `func()` invocation points at in the stack frame.**

Important: each cell in the stack segment memory block below represents one byte (8 bits) of memory.

High address	c (MSB)
	c
	c
	c (LSB)
	b
	a (MSB)
	a
	a
	a (LSB)
	RA (MSB)
	RA
	RA
	RA (LSB)
	PFP (MSB)
	PFP
	PFP
CFP/EBP →	PFP (LSB)
	x (MSB): 00000000
	x (LSB): 00000101
	d[3]: '\0'
	d[2]: 'C'
	d[1]: 'B'
	d[0]: 'A'
	e[5]
	e[4]
	e[3]
	e[2]
	e[1]
Low address	e[0]