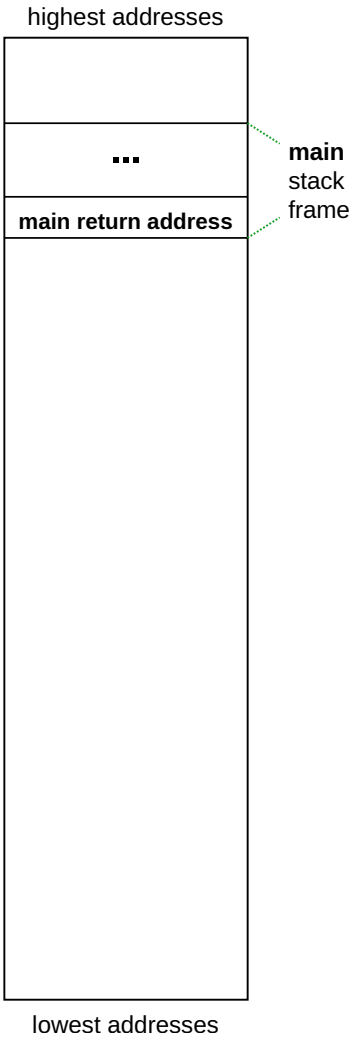
CSCI 2330 – Buffer Overflow Exercises

Consider the following two function snippets of a program that contains a buffer overflow vulnerability:

```
void foo() {
    char buf1[20];
    bar(buf1);
    ...
}
```

```
void bar(char *buf1) {
    char buf2[30];
    gets(buf2);
    ...
}
```

Assume that the only local variables used in **foo** and **bar** are **buf1** and **buf2**, respectively, and further assume that the compiler does not add any padding in the stack frames. Also assume that **foo** is called directly from the **main** function (i.e., the call chain is **main**, then **foo**, then **bar**).



1. Following the model on the right, draw a picture showing the components of the stack **just prior to executing `callq gets`**. The stack frame of **main** is already indicated for you. Clearly label the following components in memory: **stack frames** (including function names), **local variables** (including variable names), and **return addresses**. Don't worry about accurately drawing the relative size of each component or the amount of unused stack space at the top or bottom of the picture.

2. Suppose an attacker exploits this program using a code injection attack and injects an exploit string containing **15 bytes of assembly instructions** (in addition to anything else required for the exploit to work). In the same style, draw a second picture depicting the stack **just after returning from `gets`** (i.e., still within the **bar** function). Clearly label same pieces as before. Additionally, draw an **arrow** indicating where the overwritten return address points, and indicate the location of the **exploit code** and the **exploit padding**.

3. How many bytes does the entire exploit string from Q2 require?

4. Suppose an attacker exploits this program using a return-oriented programming (ROP) attack instead of a code injection attack. Assume that this attack requires three gadgets. Draw a third picture of the stack, showing the same components as in Q2 as well as clearly labeling the **gadget addresses**.

5. How many bytes does the ROP exploit string from Q4 require?