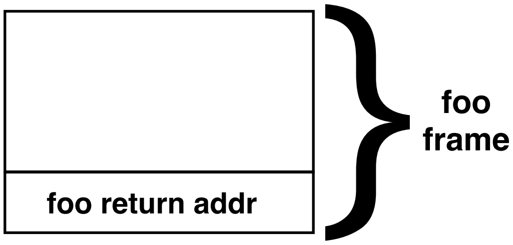
CSCI 2330 – x86-64 Procedures Exercises

1. Consider the following functions and the functions they call (**f4** and **f5** and any lines other than function calls not shown):

```
main() {        f1() {          f2() {          f3() {
    f1();           f3();           f4();           f5();
    f2();       }               }               }
}
```

Suppose the program is executing and is inside **f4**. Draw a picture of the current stack as a series of stack frames, labeled with their function names. If the stack frame includes a return address, mark the return address in the frame. An example frame is shown to the right.



2. Consider the following two functions **foo** and **bar**. Suppose the program is executing and is paused at the point indicated in the **bar** function. Draw a picture of the stack showing the stack frames of **foo** and **bar**. Label each frame along with the components of each frame, each specified as a variable name or return address along with its size in bytes. Assume that an **int** is 4 bytes and that all variables other than those that *must* be stored in memory are stored only in registers.

```
                            // 8 args, all of type int except a1
void foo() {                void bar(int* a1, int a2, ..., int a8) {
  int x = ...;                int z = ...;
  int y = ...;                int* p = &z;
  bar(&x, 2, 3, ..., 8);      ... // program paused here
}                           }
```

3. Write a snippet of x86-64 assembly that implements the following C function. Don't use **push** or **pop** instructions; instead, work with **%rsp** directly. Assume that an **int** is 4 bytes and that **foo** is some function that takes two **int*** arguments and returns an **int**. The **leaq** instruction will be useful here.

```
int cfun() {
    int x = 3;
    int y = 7;
    int z = foo(&x, &y); // note: could modify x or y
    return x + z;
}
```