CSCI 2330 – x86-64 Procedures Exercises

1. Consider the following two functions **foo** and **bar**. Assume that a program executes the **foo** function 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** and labeling each component within the frames. Assume there are no caller-saved registers or any other pieces of data not implied by the code snippets shown. How many bytes are in each frame?

```
void foo() {

  int x = ...;

  bar(&x, 2, 3, ..., 8);

}
```

```
// 8 args, all of type int except a1
void bar(int* a1, int a2, ..., int a8) {

  int y = ...;

  int* p = &y;

  ... // program paused here

}
```

2. Write a snippet of x86-64 assembly that implements the following C function. Assume that an **int** is 4 bytes and that **foo** is some function that takes two **int*** arguments and returns an **int**. Remember that **x** and **y** must be stored in memory (<u>not</u> in registers)! The **leaq** instruction will be useful here.

```
int cfun() {

      int x = 3;

      int y = 7;

      int z = foo(&x, &y);

      return x + z;

}
```

3. Consider the C function **compute** given below, and assume that the compiler uses registers to store local variables **x** and **y**. Assuming the compiler makes the most sensible choices, what kind of register (**caller-saved** or **callee-saved**) will be chosen for each variable? Then, given your choices, which registers (**just x**, **just y**, **both**, or **neither**) will need to be saved by the **compute** function itself? Do not make any assumptions about which registers are used by **do_work**.

```
void compute() {

      int x = 5;

      int y = do_work(x);

      do_work(y);

      do_work(y);

      do_work(y);

}
```