- 1. What is loop unrolling? How can it make your program more efficient? How can it make your program less efficient?
- 2. What affects the data stored on the stack? In the context of the attack lab, what instructions should be paid careful attention to?

3. The provided function func_one takes as input two pointers, that are actually each individually pointing to the first element in a N by M array of integers.

```
int func_one(char* one, char* two, int N, int M) {
               int i, j, k;
               int sum = 0;
               char* ptr1 = one;
               char* ptr2 = two;
               for (k = 0; k < 4; k++) {
                  for (j = 0; j < M; j++) {
                        for (i = 0; i < N; i++) {
                              char one = *(ptr1 + k + j*4 + i*4*M);
                               int masked = one & 0xFF;
                               int shift = k << 3;
                              int shifted = masked << shift;</pre>
                               *(ptr2 + k + j*4 + i*4*M) = masked;
                              sum += shifted;
                        }
                  }
               }
               return sum;
}
```

In what ways can we optimize the above function?

4. The provided code below is an optimization of the previous problem. Fill in the blanks.

```
int func_two(char* one, char* two, int N, int M) {
             int i, j, k;
             int sum = 0;
             int temp = 0;
             char* ptr1 = one;
             char* ptr2 = two;
             for (i = 0; i < N; i++) {
                for (j = 0; j < M; j++) {
                     temp = (0xFF \& *ptr1);
                     sum += _____;
                     *ptr2 = _____;
                     ptr1++;
                     ptr2++;
                     temp = _____;
                     sum += _____;
                     *ptr2 = _____;
                     ptr1++;
                     ptr2++;
                }
             }
             return sum;
}
```

```
🁚 jjm3105 — ssh myong@lnxsrv09.seas.ucla.edu — 80×24
Flat profile:
Each sample counts as 0.01 seconds.
                                   self
                                           total
 % cumulative self
time seconds seconds
                         calls s/call s/call name
71.85
          23.48
                23.48 1 23.48
                                          23.48 func_one
16.20
          28.78
                   5.29
                                    5.29
                                            5.29 func_one_opt
                               1
 6.46
          30.89
                   2.11
                                                   main
                               1
                                    1.95
 5.96
          32.83
                   1.95
                                             1.95 func_two
          the percentage of the total running time of the
time
          program used by this function.
cumulative a running sum of the number of seconds accounted
         for by this function and those listed above it.
seconds
self
          the number of seconds accounted for by this
seconds
          function alone. This is the major sort for this
          listing.
calls
          the number of times this function was invoked, if
          this function is profiled, else blank.
"gprof.output" 157L, 6493C
                                                           2,0-1
                                                                        Top
```

The above shows the running time of the functions discussed in problems 5 and 6. func_one is the code from problem 5 as is, func_one_opt is one optimization of func_one, and func_two is the completed code from problem 6.

The results were generated using gprof