COA1 Exam 2 – Fall 2019

Name: ________________________________ Computing ID: __________

Write Letters clearly: if we are unsure of what you wrote you will get a zero on that problem.
Bubble and Pledge the exam or you will lose points.
Assume unless otherwise specified:

- all necessary `#include`s have been used
- `char`, `short`, `int`, and `long` are 8-, 16-, 32-, and 64-bits long, respectively
- the compiler pads pointers where it is allowed to do so such that
  - ▷ an `X`-pointer is a multiple of `sizeof(X)` for all types `X`
  - ▷ `sizeof(struct X)`
    - – an even multiple of the size of its largest field
    - – the smallest such multiple big enough to store all its fields
- compilation happens using `clang` on a Linux system

Single-select by default: Multiple select are all clearly marked; answer them by putting 1 or more letters in the box, or writing "`none`" if none should be selected.
Page-at-a-time Grading: We scan your exam and grade each page separately. Do not refer to other pages, scrratch paper, etc., in your answer.
Mark clarifications: If you need to clarify an answer, do so, and also add a `*` to the top right corner of your answer box.

. . . .

Information for questions 1–2
Suppose the assembly given in each subquestion was inserted at random between two instructions of a function, with all jump targets and other code addresses updated accordingly. Either state that this has no functional impact by writing "`nop`" or describe a scenario where such an insertion could change the behavior of the function.

Question 1 [2 pt]: (see above) What if we insert `leaq (%rbx), %rbx`?

Answer: ____

____

Question 2 [2 pt]: (see above) What if we insert `xorq $0, %r9`?

Answer: ____

____

Information for questions 3–6
For each of the following questions, assume the first eight registers have the following values prior to the assembly being run:

Register	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
Value (hex)	1234	11111111	0	FF	30	3	FFFF	FFFFFFFF

The questions below are independent. Do not use the result of one as the input for the next.

Answer by writing a changed register and its new value, like "RDI = 24F2", leaving one or more lines blank if fewer registers change than there are lines.

Question 3 [2 pt]: (see above) Which program registers are modified, and to what values, by `leaq 0x4(%rdi,%rbp,2), %rdx`?

____

____

Question 4 [2 pt]: (see above) Which program registers are modified, and to what values, by `pushq %rcx`?

____

____

Question 5 [2 pt]: (see above) Which program registers are modified, and to what values, by `cmp %rsi, %rbx`?

____

____

Question 6 [2 pt]: (see above) Which program registers are modified, and to what values, by `addw %cx, %si`?

____

____

Question 7 [2 pt]: Consider the following assembly:

```
quux:
    movq flub, %rsp
    retq
flub:
```

Functionally (ignoring time taken to execute), what would `callq quux` do?

A it depends on the contents of `%rsp` before the `callq`
B it depends on the contents of `(%rsp)` before the `callq`
C it depends on what bytes follow `flub:`
D nothing; it's a no-op
E overwrite the top of the stack with 8 bytes of function `flub`
F push 8 bytes of function `flub`
G the same thing as `retq`, except `%rsp` is different
H the same thing as `jmp flub`, except `%rsp` is different
I the same thing as `call flub`, except `%rsp` is different

Answer:

Question 8 [2 pt]: What value is placed in `x`?

```
#define THING(x) 2 * x
int y = THING(1 + 2);
```

Answer:

Question 9 [2 pt]: Assume we have defined `xyxxy` as
`typedef struct { int x; char[2] y; } xyxxy;`.
What is `sizeof(xyxxy[2])`?
See the assumptions on page 1 to compute an exact number.

Answer:

Question 10 [2 pt]: What does the following code print? Recall that `puts` prints a string argument.

```
const char *s = "four";
const char *t = s + 1;
puts(t);
```

Answer:

If it has an error, write "error"

Information for questions 11–13

Consider the following code, shown with line numbers which are not part of the code itself:

```
 1.  int numbers[5] = {2, 3, 5, 7, 0};
 2.
 3.  /// determine if two numbers are co-prime
 4.  int coprime(int a, int b) {
 5.      while(b > 0) { int tmp = a % b; a = b; b = tmp; }
 6.      return a == 1;
 7.  }
 8.
 9.  /// Replace the first 0 in the array with a number
10.  /// coprime to all other numbers in the array
11.  int *add_coprime(int *array) {
12.      int *ans = malloc(sizeof(array));
13.      for(int i=0; array[i]; i+=1) ans[i] = array[i];
14.      while(*array) array += 1;
15.      int found = 0;
16.      for(int i=1; !found; i+=1) {
17.          found = 1;
18.          for(int j=0; ans[j]; j+=1)
19.              if (!coprime(i, ans[j]))
20.                  found = 0;
21.          if (found) *ans = i;
22.      }
23.      return array;
24.  }
```

Question 11 [2 pt]: (see above) The code has one memory leak. After which line should we add a `free`? For example, if a `free` should be added between `return a == 1;` and the subsequent `}`, answer "10".

Answer:

Question 12 [2 pt]: (see above) The code has one memory leak. What should be `free`ed? For example an answer "`i`" means we need to insert `free(i)` into the code.

Answer:

Question 13 [6 pt]: (see above) For each of the following memory error types, enter either a line number exhibiting the error, or "none" if the error does not occur. If there is more than one line with a given error, pick just one in your answer.

Line ______________ accesses uninitialized memory

Line ______________ accidentally casts to a pointer

Line ______________ could overflow a buffer

Line ______________ uses after `free`

Line ______________ uses after `return`

Line ______________ fails to use `sizeof`/uses `sizeof` incorrectly

Question 14 [8 pt]: Convert this C code into equivalent code using `goto` and `if`, but no `else`, loops, or `switch`es. Your code should work the same as the C code for all values of `n` (including negative values).

```
for(int i=0; i<n; i+=1) {
    if (i % x == 0) y *= i;
    else z += 1;
}
```

Question 15 [2 pt]: In the following code, comment out the `free`s which should not be present by adding `//` in front of those lines

```
int a[5];
int x;
int *f(int b[3]) {
    int *c = (int *)calloc(7, sizeof(int));
    int d[4] = {1, 2, 4, 8};
    a[0] = b[0]; b[1] = c[1]; c[2] = d[2]; d[3] = a[3];
    x = a[0] + b[1] + c[2] + d[3];

    free(a);

    free(b);

    free(c);

    free(d);

    return &x;
}
```

Information for questions 16–18
For each of the following, answer "C" if there's a compile-time error, "R" if there's a run-time error, and "L" if there's a logic error (runs but does the wrong thing).

Question 16 [2 pt]: (see above) `int y = x[6];` when `x` is defined as `int x[2] = {1,2};`

Answer:

Question 17 [2 pt]: (see above) `int y = *x;` when `x` is defined as `int *x = NULL;`

Answer:

Question 18 [2 pt]: (see above) `int y = *x;` when `x` is defined as `int x = 2501;`

Answer:

Pledge:

On my honor as a student, I have neither given nor received aid on this exam. I will not discuss the content of this exam, even in vague terms, with *anyone* other than current course staff, until Friday 8 November 2019.

____
Your signature here