COA1 Exam 2 – Fall 2018

Name: _____ Computing ID: _____

Letters go in the boxes unless otherwise specified (e.g., for **C** 8 write "C" not "8").
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:

- the following have been declared:
```
void *malloc(size_t);    void free(void *);
int puts(const char *);  int printf(const char *, ...);
```
- `char`, `short`, `int`, and `long` are 8-, 16-, 32-, and 64-bits long, respectively; and that `float` is 32- and `double` is 64-bits long.
- 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.
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 1–4
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 `callq nothing`, where elsewhere we've defined `nothing: retq`?

Answer: _____

_____

Question 2 [2 pt]: (see above) What if we insert `andq %rdi,%rdi`?

Answer: _____

_____

Information for questions 3–11

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)	0	1C3F5678	200400800	FFFF	200	240	20	100

Note: the questions are independant. 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 (%rsp,%rbp), %rdi`?

Question 4 [2 pt]: (see above) Which program registers are modified, and to what values, by `callq 0x5345B`?

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

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

Question 7 [2 pt]: Sometimes a compiler can replace a recursive function call is replaced by a jump to the function: i.e., replace `callq this_function` with `jmp this_function`. In which of the following cases would that optimization always work? Write a minimal set of letters that, if all their corresponding options are met, will permit this optmization.

A `callq this_function` is immediately followed by `retq`
B `callq this_function` is the first instruction in `this_function`
C `this_function` does not access its arguments after the recursive call
D `this_function` does not access modify `%rax` after the recursive call
E `this_function` does not modify `%rsp` after the recursive call
F `this_function` has no arguments
G `this_function` has no return value

Answer:

Question 8 [2 pt]: Consider the following assembly:

```
wiz:
    movq waz,(%rsp)
    retq
waz:
```

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

A copy 8 bytes of function `waz` into the stack
B copy 8 bytes of function `waz` into `%rsp`
C move `%rsp` to point to `waz`
D overwrite 8 bytes before function `waz` with values from the stack
E overwrite 8 bytes of function `waz` with values from the stack
F the same thing as `callq waz`
G the same thing as `jmp waz`
H it depends on the contents of `%rsp`
I it depends on the contents of `(%rsp)`

Answer:

Information for questions 9–17
For each of the following bugs, indicate the stage of compilation that would be find it. If it would not be found until run-time, write "`none`". The stages are

- **L**exing – breaking input into words and related tokens
- **P**arsing – making an abstract syntax tree (AST)
- **T**ype-checking – annotating the AST with data types, etc
- **C**ode generation – creating assembly
- **A**ssembling – turning assembly into machine code
- **L**inking – attaching library files to code

Question 9 [2 pt]: (see above)
Naming a variable `ümłą̂t`

Answer:

Question 10 [2 pt]: (see above)
The trinary operator with a missing middle case (e.g. `a?:c`)

Answer:

Question 11 [2 pt]: (see above)
Using the `&` operator on floating-point operands (e.g. `2.3&4.5`)

Answer:

Question 12 [2 pt]: (see above)
Writing two version of the same function in different files

Answer:

Question 13 [2 pt]: (see above)
Providing too few arguments to a variadic function (e.g. `printf("%d")`)

Answer:

Question 14 [2 pt]: How many times will the loop be executed?

```
#define AGAIN 1
while(AGAIN) {
    puts("Loop run");
    #define AGAIN 0
}
```

Answer:

Question 15 [2 pt]: What is `sizeof(struct{int x; double y;})`? See the assumptions on page 1 to compute an exact number.

Answer:

Question 16 [2 pt]: What is the minimum number of bytes of read-only memory needed for the compiler to store the following set of string literals: `"her"`, `"here"`, `"other"`, `"he"`?

Answer:

Question 17 [8 pt]: The following program both (a) contains a memory error and (b) has a memory leak. Circle and describe the error, and insert any needed `free` invocations to fix the memory leak.

```
typedef struct node_s { int val; struct node_s *left, *right; } node;

node *new_tree(int root_val) {
    node root;
    root.val = root_val;
    root.left = NULL;
    root.right = NULL;

    return &root;
}

void insert(node *root, int val) {

    if (val < root->val)
        if (root->left) insert(root->left, val);
        else {
            root->left = (node *)malloc(sizeof(node));
            root->left->val = val;

        }
    else
        if (root->right) insert(root->right, val);
        else {
            root->right = (node *)malloc(sizeof(node));
            root->right->val = val;

        }

}

node *remove(node *me) {
    if (me->left) {

        me->val = me->left->val;
        me->left = remove(me->left);

        return me;
    } else if (me->right) {

        return me->right;
    } else {

        return NULL;
    }
}
```

Question 18 [6 pt]: Write a snippet of C code that could have created this assembly. Do not use `goto` in your solution.

```
     movq %rdi, %rdx;
     jmp A;
 B:
     movl %edx, (%r9,%rdx,4);
     addq $-1, %rdx;
 A:
     cmpq %rdx, -1;
     jne B;
```

Pledge:

On my honor as a student, I have neither given nor received aid on this exam.

_____

Your signature here