CSE 351 Summer 2021 – Unit Summary #2 – Task 3
Due Monday, August 2nd 8pm to Gradescope

Your Name:_____

UWNet ID (email):_____

Academic Integrity Statement_____

All work on these questions is my own. I have not shared or discussed my answers with anyone else. (please sign) (1 point)

- To complete Task 3, please either:
  - print these THREE pages, fill them out and then scan and convert into a pdf
  - use digital ink or otherwise annotate the pdf electronically
- Gradescope requires you to upload a pdf that needs to be 3 pages long – annotate the PDF directly and submit as a 3-page document.
- Fill in your name and UW NetID above, then read the Academic Integrity Statement and sign your name indicating that you understand and will comply with the statement. If you are not printing this out or do not have access to digital ink, just type your full name.
- You may show scratch work for potential partial credit but showing work is not required. Be sure your final answer is placed in the blanks, boxes, or spaces provided.
- You may use your floorplan from Task 1, course lecture slides and Ed Lessons, and course textbooks while completing this task.
- Use of reference materials external to those listed above is not allowed (e.g., Stack Overflow, web searches, communicating with anyone other than the course staff, etc.)
- If you have questions, please ask on Ed! A private post is fine! Questions about the unit summaries will not be answered in office hours.
- Refer to the Unit Summary webpage for additional information: https://courses.cs.washington.edu/courses/cse351/21su/unit_summaries/

Good Luck!

1. C and Assembly (13 points total)

Consider the following function given in x86-64 assembly:

```
fun_fn:
     movl $-1, %eax            #line 1
     testl     %esi, %esi      #line 2
     jle  .L5                  #line 3
     movl %esi, %esi           #line 4
     xorl %ecx, %ecx           #line 5
.L2:
     cmpl %edx, (%rdi,%rcx,4) #line 6
     je   .L4                  #line 7
     incq %rcx                 #line 8
     cmpq %rcx, %rsi           #line 9
     jne  .L2                  #line 10
.L5:
     retq                      #line 11
.L4:
     movl %ecx, %eax           #line 12
     retq                      #line 13
```

a) (4 pts) Fill in the function's C signature with the correct **C types**:

`_____ fun_fn(_____ arg1, _____ arg2, _____ arg3)`

b) (4 pts) This function contains a `for` loop. Fill in the corresponding parts below, use variable names that correspond to the register names used (e.g. use `eax` for `%eax`):

`for (_____; _____; _____)`

c) (3 pts) Describe at a high level what you think this function accomplishes. (not line-by-line)

d) (2 pts) Describe at a high level what change if any would it make to what this function accomplishes if the `cmpl` on line 6 was changed into:

```
subl    %edx, (%rdi,%rcx,4)
```

2. C and Assembly (11 points total)

Consider the following function given in x86-64 assembly:

```
13f20 <_cool_fn>:
13f20: 89 f8                movl    %edi, %eax
13f22: 85 ff                testl   %edi, %edi
13f24: 78 22                js      0x13f48 <_cool_fn+0x28>
13f26: 55                   pushq   %rbp
13f27: 48 89 e5             movq    %rsp, %rbp
13f2a: 53                   pushq   %rbx
13f2b: 50                   pushq   %rax
13f2c: 89 f3                movl    %esi, %ebx
13f2e: ff c8                decl    %eax
13f30: c1 fe 02             sarl    $2, %esi
13f33: 81 f6 5f 01 00 00    xorl    $351, %esi
13f39: 89 c7                movl    %eax, %edi
13f3b: e8 e0 ff ff ff       callq   0x13f20 <_cool_fn>
13f40: 01 d8                addl    %ebx, %eax
13f42: 48 83 c4 08          addq    $8, %rsp
13f46: 5b                   popq    %rbx
13f47: 5d                   popq    %rbp
13f48: c3                   retq
```

a) (2 pts) How much space (in bytes) does this function take up in our final executable?

b) (2 pts) What callee-saved registers (if any) are used? Answer with the 64-bit register names.

c) (2 pts) What caller-saved registers (if any) are used? Answer with the 64-bit register names.

d) (2 pts) What is the return address to **`cool_fn()`** that gets stored on the stack during the recursive calls? (provide your answer in hex)

e) (3 pts) Fill in the blanks for the C code for the base case of **`cool_fn`**, use variable names that correspond to the register names (e.g. `eax` for `%eax`):

```
if ( _____________________________ )

    return _____________________________
```