# CSE 351 Spring 2021 – Unit Summary #2 – Task 3
# Due Fri 5/07/21 11:59pm to Gradescope

**Your Name:**________________________

**UWNet ID (email):**________________________

**Academic Integrity Statement**________________________

All work on these questions in 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
- 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 study guide 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 the Ed Board. 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/cse351/21sp/unit_summaries/

Good Luck!

**1. C and Assembly (13 points total)**

Consider the following function given in x86-64 assembly:

```
sun:                                  # line 1
        movl    $0, %eax              # line 2
        movl    $0, %r8d              # line 3
        jmp     .L2                   # line 4
.L3:                                  # line 5
        addl    $1, %eax              # line 6
.L2:                                  # line 7
        cmpl    %esi, %eax            # line 8
        jge     .L5                   # line 9
        movslq  %eax, %rcx            # line 10
        cmpl    %edx, (%rdi,%rcx,4)   # line 11
        jne     .L3                   # line 12
        addl    $1, %r8d              # line 13
        jmp     .L3                   # line 14
.L5:                                  # line 15
        movl    %r8d, %eax            # line 16
        ret                           # line 17
```

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

```
__________ sun(__________ 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 11 was changed into:

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

## 2. C and Assembly (11 points total)

Consider the following function given in x86-64 assembly:

```
00000000004005f7 <yowza>:
  4005f7: 89 f8                mov    %edi,%eax
  4005f9: 83 ff 01             cmp    $0x1,%edi
  4005fc: 7f 02                jg     400600 <yowza+0x9>
  4005fe: f3 c3                repz retq
  400600: 53                   push   %rbx
  400601: 89 f3                mov    %esi,%ebx
  400603: c1 fe 02             sar    $0x2,%esi
  400606: 8d 7f ff             lea    -0x1(%rdi),%edi
  400609: e8 e9 ff ff ff       callq  4005f7 <yowza>
  40060e: 01 d8                add    %ebx,%eax
  400610: 5b                   pop    %rbx
  400611: 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 **`yowza()`** 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 **`yowza`**, use variable names that correspond to the register names (e.g. `eax` for `%eax`):

```
    if ( ____________________________ )

        return ______________________________
```