# CSE 351 Summer 2025, Midterm Exam II
# July 31, 2025

| | |
|---|---|
| **Full Name:** | |
| **UW NetID:** | @uw.edu |
| **Seat Number:** | |
| I certify that all work is my own. I had no prior knowledge of exam contents nor will I share the contents with any student in CSE 351 who has not yet taken the exam. Violation of these terms may result in a failing grade. **Signature:** | |

## Instructions:

- **Do not turn the page until you are instructed to begin.**
- This exam is closed book (no smartphones, calculators, or other electronic devices). Turn off any mobile devices, remove hats, headphones, and smartwatches, and put them away.
- You are allowed one page (U.S. letter, double-sided) of *handwritten* notes. **Write your name and NetID on your notes page and turn it in with your exam.**
- This exam contains 4 problems and 1 bonus question spread across 6 pages. The last page is a reference sheet. You may detach it from the rest of the exam.
- When a box or line is provided, please write your answer in the box or on the line.
- If a question involves bubbling in a ◯, please fill the shape in completely.
- You have 60 minutes to complete this exam. **Please stop writing when the clock stops.**

## Advice:

- Read questions carefully before starting. Make sure you understand what they're asking!
- Don't spend too much time on any one problem. If you find yourself getting stuck, skip around. Make sure you get to all the questions.
- **Relax! You are here to learn :)**

| **Question** | **1** | **2** | **3** | **4** | **5** | **Total** |
|---|---|---|---|---|---|---|
| **Points** | 12 | 14 | 8 | 16 | 1 | **51** |
| **Page** | 2 | 3 | 4 | 5 | 6 | |

Name: _____ UW NetID: _____

**1. (12 points) Arrays, Structs, & Buffer Overflow**

We define the following struct representing a cat:

```
typedef struct {
  int age;
  char name[11];
  Cat* best_friend;
  short fluffiness;
} Cat;
```

(a) (6 points) How large is an instance of a **Cat** in bytes? How many bytes of internal and external fragmentation are there?

**Size:** [ ] **Internal:** [ ] **External:** [ ]

The next two questions ask about an instance of **Cat** called **lal**. Assume that we have allocated **lal** somewhere on the stack.

(b) (2 points) Suppose we allocate a **char[] buf** on the stack, such that **buf** is 24 bytes below **lal**. That is, **&buf + 24 = &lal**.

Fill in the blank in the following C code so that it correctly sets **lal.best_friend** equal to the **NULL** pointer, without changing **lal.fluffiness**. (Other fields in **lal** may be modified.)

```
for (int i = 0; i < _________; i++) {
  buf[i] = 0;
}
```

(c) (4 points) Now suppose we want to set **lal.best_friend** to point to the address **0x1234 5678 CAFE F00D**. Fill in the contents of a **char[] new_friend** with the minimum-length exploit string so that after the following code runs, **lal.best_friend = 0x1234 5678 CAFE F00D**.

```
char* friend_ptr = (char*)&(lal.best_friend);
for (int i = 0; i < sizeof(lal.best_friend); i++) {
  friend_ptr[i] = new_friend[i];
}
```

Write each element of **new_friend** as a hex byte, e.g., **FF** or **30**, one per box. No need to include prefixes. You may not need to use all of the boxes. Leave any unused boxes blank.

```
char new_friend[] = {
```

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |

```
};
```

Name: _____ UW NetID: _____

**2. (14 points) Assembly & C**

Consider the following x86 assembly, which implements a mysterious function:

```
    mystery:
1           movl    $0, (%rdi)
2           movl    $0, %edx
3           jmp     .L2
    .L3:
4           movsbl  %sil, %eax
5           orl     (%rdi), %eax
6           movl    %eax, (%rdi)
7           shll    $8, %eax
8           movl    %eax, (%rdi)
9           addl    $1, %edx
    .L2:
10          cmpl    $3, %edx
11          jle     .L3
12          ret
```

(a) (8 points) Fill in the C code so that the assembly above correctly implements **`mystery()`**:

```
void mystery(________ a, char b) {

  ________ = 0;

  for (int i = 0; i ________; i++) {

    *a = *a | ________;

    *a = *a << 8;
  }
}
```

(b) (4 points) Could we replace the **`cmp`** on line 10 with a **`test`** instruction, without adding or changing any other instructions? If so, give an example. Explain in 1 – 2 sentences.

◯ Yes ◯ No

Explain:

**(Question continues on the next page.)**

(c) (2 points) In 1 – 2 sentences, describe what **`mystery()`** does at a high level (*not* line-by-line).

**3. (8 points) ISA Design**

Your course staff decide that x86 is far too complicated, and they set out to design a much simpler ISA to replace it: Simple86. Instead of 16 registers that can all contain values of 8 bytes or less, Simple86 will have two sets of registers: 10 registers that can only contain 8-byte values, and 12 registers that can contain values of 4 bytes or less.

(a) (4 points) Name one disadvantage of the new register design in Simple86. Explain your answer in 1 – 2 sentences.

The CSE 351 team decides that Simple86 needs to be even simpler. Now it will only have four general-purpose registers, all of which can hold 8-byte values or less: **`%rax`**, **`%rsp`**, **`%rbp`**, and **`%rdi`**. Assume Simple86 also still has the program counter, **`%rip`**.

(b) (4 points) Can we reimplement all our existing x86 programs with just the registers above? Explain why or why not in 1 – 2 sentences.

◯ Yes ◯ No

Explain:

Name: _____ UW NetID: _____

**4. (16 points) Stack & Procedures**

Consider the recursive function **foo()**:

```
int foo(int x, int y) {
  if (x > y) {
    return 0;
  } else {
    return x + foo(x << 1, y);
  }
}
```

Here is some disassembly implementing **foo()** (all addresses are in hex):

```
0000000000401129 <foo>:
  401129:    39 f7              cmp    %esi,%edi
  40112b:    7e 06              jle    401133 <foo+0xa>
  40112d:    b8 00 00 00 00     mov    $0x0,%eax
  401132:    c3                 retq
  401133:    53                 push   %rcx
  401134:    89 fb              mov    %edi,%ecx
  401136:    8d 3c 3f           lea    (%rdi,%rdi,1),%edi
  401139:    e8 eb ff ff ff     callq  401129 <foo>
  40113e:    01 d8              add    %ecx,%eax
  401140:    5b                 pop    %rcx
  401141:    c3                 retq
```

(a) (6 points) Suppose we call **foo(3, 10)** from **main()**. Using the disassembly above, fill in the contents of the stack at its deepest point (i.e., when the largest number of items have been pushed to the stack, and none of them have been popped yet).

If you do not know the value of an entry, write "unknown". Leave unused entries (where nothing has been pushed to the stack) blank. Write all values in hex, and omit leading zeros.

| Stack address | Contents |
|---|---|
| 0x7fffffffdff8 | <return address to main> |
| 0x7fffffffdff0 | 0x |
| 0x7fffffffdfe8 | 0x |
| 0x7fffffffdfe0 | 0x |
| 0x7fffffffdfd8 | 0x |
| 0x7fffffffdfd0 | 0x |
| 0x7fffffffdfc8 | 0x |

(b) (2 points) Referring to the same execution of **`foo(3, 10)`** as above, what are the contents of the registers **`%rdi`** and **`%rcx`** just before the first call to **`foo(3, 10)`** restores the original value of **`%rcx`** by popping it off the stack? Write all values in hex. Be sure to use the correct bitwidth, including leading zeros.

| | |
|---|---|
| **`%rdi`** | **`0x`** |
| **`%rcx`** | **`0x`** |

(c) (2 points) Notice that in the disassembly above, the **`callq`** instruction calls **`foo`** by the function's address, **`0x401129`**, rather than with a label. At what point in building this executable were we able to determine **`foo`**'s final address?

◯ Compilation ◯ Assembling ◯ Linking ◯ Loading

(d) (4 points) This disassembly violates one of our conventions! Which convention is being violated? Explain your answer in 1 – 2 sentences.

◯ Stack discipline ◯ Register-saving ◯ Argument-passing

Explain:

(e) (2 points) There are multiple ways we can fix this problem, but one way would be to replace a register we're currently using with a different one. Which register should we replace, and with what other register? Answer with the names for the full 64-bit versions of each register.

**Old register:** _____ **New register:** _____

**5. (1 point) What are the names of Alexandra's other two cats? *Hint*: They share a theme with Lal's name!**

_____ and _____