Midterm Exam — COS 217 — Spring 2023

This exam consists of 6 questions. You have 50 minutes – budget your time wisely. Assume the ArmLab/Linux/gcc217 environment unless otherwise stated in a problem.

Do all of your work on these pages. You may use pages 2 and 8 as scratch space, however this exam is preprocessed by computer, so for your final answers to be scored you must write them inside the designated boxes and fill in selected circles and boxes completely (⚫ and ⬛, not ✔ or ✘). Please make text answers dark and neat.

Name: ______________________ NetID: ____________

Precept:

◯ P01 - MW 1:30 Donna Gabai	◯ P04 TTh 1:30 Wei Luo	◯ P06 TTh 3:30 Ashwini Raina
◯ P02 - MW 3:30 Donna Gabai	◯ P04A TTh 1:30 Samuel Ginzburg	◯ P07 TTh 7:30 Wei Tang
◯ P03 - TTh 12:30 Guðni Nathan Gunnarsson	◯ P05 TTh 2:30 Jianan Lu	

This is a closed-book, closed-note exam, except you are allowed one single-sided study sheet. Please place items that you will not need out of view in your bag or under your working space at this time. Electronic devices such as cell phones, laptops, etc. may not be used during this exam.

This examination is administered under the Princeton University Honor Code. Students should sit one seat apart from each other, and refrain from talking to other students during the exam. All suspected violations of the Honor Code must be reported to: `honor@princeton.edu`.

In the box below, **copy and sign** the Honor Code pledge before turning in your exam:
"I pledge my honor that I have not violated the Honor Code during this examination."

Exam statistics:
Maximum: 54 / 55
Mean: 38.45 / 55 (70%)
Median: 40 / 55 (73%)
Mode: 44 / 55 (80%)
Standard Deviation: 8.32 / 55

X ______________________________

(*... Ready for It?* – The exam questions begin on page 3. This page may be used for scratch work, however any answers given on this page will not be graded.)

Question 1: *You Belong with Me* 6 points

In precepts 4 and 5, you transitioned from a single-file program using functions to compute the greatest common divisor and the least common multiple into a reusable module with an interface (intmath.h), an implementation (intmath.c), and a sample client (testintmath.c). Imagine that we have expanded this module with a new function that returns the larger of its two integer parameters' values. For each line of code from the expanded module given below, identify whether it is **most** likely to be found in the client, in the interface, or in the implementation. Fill in exactly one circle per line.

	testintmath.c	intmath.h	intmath.c
a. `int IntMath_max(int i, int j);`	○	●	○
The function *declaration* appears in the **interface** file, where a client can see it			
b. `int IntMath_max(int i, int j) {`	○	○	●
The *definitions* for the interface's functions appear in the **implementation** file			
c. `int main(void) {`	●	○	○
The **client** file is the (only!) one with the `main` function for the program			
d. `#define INTMATH_INCLUDED`	○	●	○
This is the 2nd part of the pattern to guard against multiple inclusion of **interface**s			
e. `if(i >= j) return i; else return j;`	○	○	●
This is the logic in the body of `IntMath_max` in the **implementation**			
f. `assert(IntMath_max(i, j) >= i);`	●	○	○
This checks the value from a *call* to `IntMath_max`, most likely in the **client**			

Question 2: *Anti-Hero* 5 points

For each expression, write its result **in decimal (base 10)** in the corresponding box.
Hint: recall that the hex literal `0xF` is of type `signed int`.

Expression	Result
a. `~(0xF << 2)`	-61
b. `(-~0xF) >> 3`	2
c. `-(0xF + ~0xF)`	1
d. `0xF & ~(1 << 3)`	7
e. `~(0xF >> !0xF)`	-16

Question 3: *I Know Places | Bigger than the Whole Sky* 12 points

For each numbered expression in the program below, indicate the section in memory and the number of bytes that are allocated (statically or dynamically) by the **bolded** portion. If no memory is allocated, write "NONE" and 0 in the two boxes.

```
unsigned int ai2ui(① int ai[], size_t len) {
  size_t i;
② unsigned int ui = 0;
  for(i = 0; i < len; i++) ui = 10*ui + ai[i];
  return ui;
}
int main(void) {
③ int aiDigits[] = {1, 7, 4, 6};
④ unsigned int *pui;
  pui = ⑤ malloc(sizeof(*pui));
  *pui = ai2ui(aiDigits,
         sizeof(aiDigits)/sizeof(aiDigits[0]));
  printf(⑥ "%u\n", *pui);
  free(pui);
  return 0;
}
```

	SECTION		NUMBER OF BYTES
①	STACK		8
②	STACK		4
③	STACK		16
④	STACK		8
⑤	HEAP	Assume `malloc` succeeds.	4
⑥	RODATA		4

Question 4: *Forever & Always*

12 points

Consider these C definitions:

```
char a0[3] = {'2','1','7'};
char *p1 = a0;
const char *p2 = a0;
char *const p3 = a0;
const char *const p4 = a0;
```

For each expression using the variables defined above, indicate whether the given increment operation is legal or would produce a compiler error from `gcc217`. Fill in exactly one circle per line.

	LEGAL	COMPILER ERROR
a. `a0++;`	○	●
b. `(*a0)++;`	●	○
c. `p1++;`	●	○
d. `(*p1)++;`	●	○
e. `(&p1)++;`	○	●
f. `(*(&p1))++;`	●	○
g. `p2++;`	●	○
h. `(*p2)++;`	○	●
i. `*(p3++);`	○	●
j. `(*p3)++;`	●	○
k. `p4++;`	○	●
l. `(*p4)++;`	○	●

Question 5: *I Did Something Bad*

12 points

Consider the following code, which appears at the beginning of a function, and assume that all `malloc` and `calloc` invocations are successful (i.e., they do not return `NULL`).

```
int i;
int ai[3] = {1, 2, 3};
int *pi1 = (int *) malloc(3 * sizeof(int));
int *pi2 = (int *) calloc(3, sizeof(int));
for(i = 0; i < 3; i++)
   pi1[i] = ai[i];
```

For each of these snippets that could individually appear later in the same function, identify **all** the memory management errors resulting from execution of that snippet.

- **A**: accesses unallocated memory
- **B**: accesses freed memory (dangling pointer)
- **C**: leaks memory
- **D**: frees unallocated memory
- **E**: double frees allocated memory
- **None**: none of the memory management errors above results

	A	**B**	**C**	**D**	**E**	**None**

a. `free(pi2);`
`free(pi1);`
`free(ai);`
	☐	☐	☐	■	☐	☐

b. `pi1 = ai;`	☐	☐	■	☐	☐	☐

c. `free(pi1);`
`pi1 = pi2;`
`free(pi1);`
	☐	☐	☐	☐	☐	■

d. `pi2 = pi1;`
`free(pi2);`
`*pi1 = ai[0];`
	☐	■	■	☐	☐	☐

e. `pi2[3] = ai[0];`	■	☐	☐	☐	☐	☐

f. `free(pi1);`
`pi2 = pi1;`
`*pi2 = ai[2];`
`free(pi2);`
	☐	■	■	☐	■	☐

Question 6: *Glitch*

8 points

Consider the following extension to the string library that you implemented in Assignment 2. You may assume that this function's file has `#included` all potentially required `.h` files. The function `Str_reverse` is intended to reverse a string's contents (not including the trailing nullbyte). For example, the contents `{'S','w','i','f','t','\0'}` would become `{'t','f','i','w','S','\0'}`.

```
1    void Str_reverse(char *pcSrc) {
2       char *pcTemp;
3       char *pcTempStart;
4       char *pcSrcStart = pcSrc;

5       assert(pcSrc != NULL);

6       pcTemp = (char *)malloc(sizeof(pcSrc));
7       if (pcTemp == NULL) {
8          fprintf(stderr, "Insufficient memory\n");
9          exit(EXIT_FAILURE);
10      }
11      pcTempStart = pcTemp;

12      while (*pcSrc != '\0')
13         *pcTemp++ = *pcSrc++;

14      pcSrc--;
15      pcTemp = pcTempStart;

16      while (&pcSrc >= &pcSrcStart)
17         *pcSrc-- = *pcTemp++;

18      free(pcTemp);
19      return;
20   }
```

This implementation has at least three major bugs. For any two bugs, identify the line number on which the bug occurs and write the correctly debugged line in its entirety.

a. Bug 1 Line Number

b. Bug 1 Correction

SEE NEXT PAGE

c. Bug 2 Line Number

d. Bug 2 Correction

(*Blank Space* – that was the last question. The space below is intentionally left blank. You may use it for scratch work, however any answers given below will not be graded.)

The three bugs in Question 6 were:

Line 6: `sizeof(pcSrc)` calculates the number of bytes of the pointer, which will not vary based on the string's length. The argument to `malloc` should have been `Str_getLength(pcSrc)` (or `strlen`). It would be fine to allocate 1 more byte than that for the `'\0'`, which fits our general pattern, however it is not necessary in this case since the extra byte will never be filled by the loop on lines 12-13.

Line 16: this compares the addresses of these two pointer variables, not their values (i.e., where they point). The correct loop sustaining condition should be:
`(pcSrc >= pcSrcStart)`

Line 18: this attempts to `free` memory at an address that was not returned by `malloc`, because we have advanced `pcTemp` to the end of the string. Thus, we should use the variable meant to keep the place of the beginning of the string instead:
`free(pcTempStart);`