COS 217, Fall 2022
Midterm Exam

This exam has 4 questions, some with several parts. You have 50 minutes, so budget your time.
Do all of your work on these pages and give the answer in the space provided.
Assume the `armlab`/Linux/C/`gcc217` environment unless otherwise stated.

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

Name: ______________________ **NetID:** ______________

Precept (circle one):

1: MW 1:30 X. Li	2: MW 3:30 X. Li	3: TTh 12:30 M. Perroni-Scharf
4: TTh 1:30 Q. Dang	5: TTh 1:30 D. Gabai	6: TTh 2:30 J. Chung
7: TTh 3:30 D. Xu	8 : TTh 3:30 D. Gabai	9: TTh 7:30 W. Yang

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.

Write out and sign the Honor Code pledge before turning in the test:

"I pledge my honor that I have not violated the Honor Code during this examination."

Pledge:

Signature:

Question # and Title	Available points	Points earned
1 The Gilded Six Bits	**8**	
2 Peeling Onions	**4**	
3 l(a	**12**	
4 Filling Station	**8**	
TOTAL	**32**	

(The exam questions begin on page 3. This page may be used for scratch work or to complete a question for which you found insufficient space. In the latter case, please clearly label which question you are continuing.)

1. The Gilded Six Bits (8 points)

This problem deals with a computer on which bytes are **6** bits long, rather than 8 bits. When describing 6-bit byte ("*6yte*") values and addresses, one can use two octal symbols as shorthand for the 6 bits (similar to how we use two hexits as shorthand for 8 bits). Despite the difference in byte size, this computer represents integer types the same way as armlab: `char` is 1 *6yte*, `short` is 2 *6ytes*, `int` is 4 *6ytes*, `long` is 8 *6ytes* (48 bits), and signed types use two's complement.

(1a – 2 points) What is the maximum possible value, in decimal, of a (`signed`) `long` on this computer? Represent your answer in terms of a power of 2, for example $2^8 - 39$. Show your work for potential partial credit but circle your final answer.

(1b – 6 points) Now consider the following variable declarations and initializations:

```
char c = 0xC;
short s = (short) -7;
unsigned int i = 51U;
```

Write the memory contents of each variable immediately after these initializations have occurred, using the octal symbol shorthand described above. For example, if a variable's value is represented as the bits `000010001111`, you would write `0217`. Show your work for potential partial credit but circle your final answer for each variable (2 points each):

`c`:

`s`:

`i`:

2. Peeling Onions (4 points)

Consider the following program, which is intended to use `chars` to count down from 100 to 0:

```
1     #include <stdio.h>
2     enum {LIMIT = 100};
3     void countdown(char c) {
4        if(LIMIT < c) return;
5        printf("%d\n", c);
6        countdown(c-1);
7     }
8
9     int main(void) {
10       countdown('d');
11       return 0;
12    }
```

For each bolded line in the program above, considering only that line, does the line:

A exhibit a portability issue because it assumes a property of ASCII to accomplish this
B exhibit a portability issue even if we can assume ASCII
C both A and B
D neither A nor B

Circle exactly one letter for each line (1 point each):

Line 2:	A	B	C	D
Line 4:	A	B	C	D
Line 10:	A	B	C	D
Line 11:	A	B	C	D

3. l(a (12 points)

For each part in this problem you are given a brief function specification similar to what you would find in a COS 217-compliant function comment, along with a restriction on how you may implement it in order to earn full credit. You may assume these function declarations are in the API (`.h` file) for a module. In the space below each specification, write your function definition.

Your functions on this problem do **not** have to validate that their parameters are not `NULL` with `asserts`, do **not** have to contain comments, and may use either array or pointer notation. If any of your responses is much longer than 7-9 lines, you may be off on the wrong track.

(3a – 4 points) `putChars` takes a string `s`, whose contents it may not change, prints all the characters of the string (not including the trailing nullbyte) to `stdout`, and returns nothing. *Restriction for full credit*: must use `putchar` (not `printf` or others) to print each character.

(3b – 4 points) `upAndPrint` takes a string `s`, changes each character in the string to be the upper-case version of itself, if applicable, then prints the newly modified string to `stdout` and also returns the modified string. *Restriction for full credit*: must use a single call to `printf` to print the string (i.e., not character-by-character like part a). (Reminder – you may use this function from `ctype.h`, which will return the upper-case version of `c` or `c` itself if there is no upper-case version: `int toupper(int c)`)

(3c – 4 points) `findEscape` takes a string `s`, whose contents it may not change, and returns a pointer to the location in the string that contains the first backslash character, or `NULL` if there is no backslash character in the string. The pointer returned is not restricted against changing the contents to which it points. *Restriction for full credit*: must declare no variables other than `s`.

4. Filling Station (8 points)

Consider the following (**very** buggy) program to read in a series of `int`s from `stdin` and report their sum on `stdout`:

```
1     #include <stdio.h>
2     #include <stdlib.h>
3
4     enum {LENGTH=10};
5
6     static void printSum(const int aiNums[LENGTH], size_t ulLen) {
7        int iSum = 0;
8        int *piIndex = malloc(sizeof(*piIndex));
9
10       while(*piIndex < ulLen)
11          iSum += aiNums[ulLen--];
12       printf(iSum);
13       free(aiNums);
14    }
15
16    int main(void) {
17       int iScanfReturn;
18       int *piSomeInts, *piThisInt;
19
20       piSomeInts = calloc(LENGTH, sizeof(4));
21       piThisInt = piSomeInts;
22       while((iScanfReturn = scanf("%d", piThisInt)) == 1)
23          piThisInt++;
24       printSum(piSomeInts, piThisInt-piSomeInts);
25       free(piSomeInts);
26    }
```

(4a – 2 points) `gcc217` will show a warning on four of these lines. List **two** such line numbers:

(4b – 2 points) After fixing the four lines mentioned in part a (you don't have to do so!), there remain two logic bugs dealing with loop control in `printSum`. Describe **one** of them, in no more than two sentences:

(4c – 2 points) After fixing the four lines mentioned in part a, there remain at least two dynamic memory management issues in `printSum`. One is that there is no check that `malloc` did not return `NULL`. Describe another such error, in no more than two sentences:

(4d – 2 points) There is a major problem in the code in `main` for reading the integers into `piSomeInts`. Describe the issue, in no more than two sentences:

(There are no more questions beyond this point. This page may be used for scratch work or to complete a question for which you found insufficient space. In the latter case, please clearly label which question you are continuing.)

Because some of you will have been thinking about this: the problem titles are notable works from 20th Century American literary giants, and each fits the theme of its problem: *The Gilded Six Bits*, by Zora Neale Hurston (6-bit bytes); *Peeling Onions*, by Adrienne Rich (a recursive function); *l(a*, by e e cummings (a very short poem with a very short title for some very short functions); *Filling Station*, by Elizabeth Bishop (filling up an array).