Midterm Exam  Spring 2026

This exam consists of 4 substantive questions. You have 50 minutes – budget your time wisely. Assume the ArmLab/gcc217 environment unless otherwise stated in a problem.

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

Name: _____ NetID: _____

Precept:

◯ P01 - MW 1:20 Xiaoyan Li	◯ P03 - TTh 12:15 Polly Ren	◯ P06 - TTh 1:20 Lana Glisic
◯ P02 - MW 3:30 Nicholas Yap	◯ P04 - TTh 12:15 David Shustin	◯ P08 - TTh 3:30 Viola Chen

This is a closed-book, closed-note exam, except you are allowed one one-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, smartwatches except to check the time, 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."

X_____

Question 0: Prologue 0 points

Make sure you have filled out your name, NetID (not PUID, not email alias), precept and the Honor Code pledge on the front page. Sign once you have finished the exam.

Question 1: (Bug) Hunter's Mantra 9 points

Consider the following function, which separates `aiNum`'s `len` elements into `piece` data structures, allocating an array of `piece`s that is returned, with each `piece` being filled with `numPer` elements except, potentially, for the last one. The caller owns the newly allocated memory.

```
struct piece {
   int *piNums;
   size_t ulCount;
};

struct piece *breakIntoAMillionPieces(int aiNums[], size_t len, size_t numPer) {
   struct piece *psPieces;
   size_t i, j, k;

   assert(aiNums != NULL);
   assert(numPer != 0);
   assert(len != 0);

   i = len / numPer;
   psPieces = (struct piece *) calloc(i, sizeof(struct piece *));
   if(!psPieces) return NULL;
   for(j = 0; j < i; j++) {
      psPieces[j].piNums = (int *) calloc(numPer, sizeof(int));
      if(!psPieces[j].piNums) return NULL;
      psPieces[j].ulCount = numPer;
      for(k = 0; k < numPer; k++)
         psPieces[j].piNums[k] = aiNums[j*numPer + k];
   }
   free(aiNums);
   return psPieces;
}
```

The function `breakIntoAMillionPieces` has *many* problems. In each of the three boxes at the top of the next page, describe one bug. Your descriptions should each be no more than two sentences.

A bug in this problem is something that may:

- cause a compiler warning or error
- violate the behavior from this problem's desecription or crash at runtime
- constitute a memory management error

Question 2: Takedown

9 points

Consider the following C function:

```
int mysterysaja(int z) {
   int m = -1 & 0x7E;
   while((z -= m) > 031) m /= 2;
   return z;
}
```

a. What is the value of the constant `0x7E` converted into the following lower bases?

Base 10	Base 5	Base 2

b. What is the value returned by a call to `mysterysaja(217)`?
Note: recall that, in C, `031` is not the same integer value as `31`.

Question 3: Free

15 points

For the statements in each part of this question, indicate one or more potential statuses that may result, defined as follows:

ML Memory Leak
aka garbage creation, having unfreed memory that is no longer accessible.

BD Bad Dereference
dereferencing a NULL pointer or a pointer to memory that was never allocated or has already been freed.

IF Improper Free
freeing an address that was never the return value from a dynamic allocation or has already been freed. Note: `free(NULL);` is *not* an error.

OK Okay
exhibits no dynamic memory problem.

If different statuses could result depending on the result of a call to an allocation function or contents of memory, then mark all such possible statuses. An example of this is given, for which the status is **OK** if `r` points at an empty string and **IF** otherwise.

Each part of this question is independent of the others, but for all of them assume initially:

- `r` is a `char` pointer pointing to a `k`-byte allocation in the heap returned by `malloc`
  - at least one byte within this allocation has the character value `'\0'`
- `m` is a `char` pointer with value `NULL`
- `b` is a `size_t`

		ML	BD	IF	OK
ex.	`m = r; free(r - strlen(r));`	☐	☐	■	■

a.	`free(strcpy(malloc(strlen(r) + 1), r));`
	Note: recall `strcpy`'s function signature is
	`char *strcpy(char *dest, const char *src);`
		☐	☐	☐	☐

b.	`for(b = 0; b < k; b++) free(r + b);`	☐	☐	☐	☐
c.	`free(r++);`	☐	☐	☐	☐
d.	`m = r; free(m); printf("%lu", r);` (Note: `%lu` is a specifier for 64-bit unsigned values)	☐	☐	☐	☐
e.	`m = r; free(r); printf("%s", m);`	☐	☐	☐	☐
f.	`free(r = realloc(r, 2*k));`	☐	☐	☐	☐

Question 4: How It's Done

12 points

Write the following function, which trims (i.e., eliminates) spaces from the start and end of the string parameter and reduces any instances of multiple spaces within the rest of the string each to a single space. Concretely, if given a writable string with the contents on the left in the example below, where subscript $_s$ indicates a space character (`' '`), modify the string in-place and return it with the contents on the right:

"$_{sss}$Heels,$_{sss}$nails,$_s$blade,$_{ss}$mascara$_s$" → "Heels,$_s$nails,$_s$blade,$_s$mascara"

Handling the starting and interior spaces can be modeled as a DFA with two states: within a sequence of 1 or more spaces and within a word. You might draw this DFA on page 6 and consider the transitions and actions needed before completing the code scaffolded below:

```
char *Str_spacesDoneDoneDone(char *pcStr) {
   size_t rIndex = 0; /* index in string to read current contents */
   size_t wIndex = 0; /* index in string to write updated contents */
   /* initially set as within spaces sequence to suppress all leading spaces */
   int inSpaces = 1;
   assert(pcStr != NULL);

   while (pcStr[rIndex] != '\0') {
      if (pcStr[rIndex] == ' ') {









      }
      else {









      }


   }

   /* back up before any trailing spaces */
   if (wIndex > 0 && pcStr[wIndex - 1] == ' ') wIndex--;



}
```

(Question 4 was the last question. This page is intentionally left blank. It may be used for scratch work from any problem, however any answers given on this page will not be graded.)

Gee, *How You Like That*? We hope this exam was *Dynamite*, and not too *Spicy* or *Drama*. May your new *Path* and *Strategy* be: "*All I Wanna Do* is go back to my *APT.* for a *Soda Pop*, a *Perfect Night* and some good *Interpretation of Dreams*". Enjoy your *Spring Day*(s) and *Cherry Blossom Ending* – and may your break be *Golden*!