CS 240: Programming in C
Midterm Exam 2
Spring 2025

Name: _____

Username/email: _____ @purdue.edu

Read all instructions before beginning the exam.

- This is a closed book examination. No materials other than those provided for you are allowed.
- You need only a pencil and eraser for this examination. If you use ink, use either black or blue ink. If you use pencil, your writing must be dark and clearly visible.
- This examination contains an amount of material that a well-prepared student should be able to complete in approximately two hours.
- This examination is worth a total of 100 points. Not all questions are worth the same amount. Plan your time accordingly.
- Write legibly. You should try to adhere to the course code standard when writing your solution(s). Egregious violations may result in point deductions. Function header comments are not required.
- You may leave after you have turned in all pages of the examination booklet. You will not be able to change any answers after turning in your examination booklet.
- If you finish your exam with less than 15 minutes remaining, please remain seated until the exam is over. We will call you down by row to turn in your exam.
- Read each question *carefully* and *only do what is specifically asked for* in that problem.
- Some problems require several steps. Show all your work. Partial credit can only be rewarded to work shown.
- Do not attempt to look at other students' work. Keep your answers to yourself. Any violation will be considered academic dishonesty.
- Write your username on *EVERY* page where indicated. Any page without a username will receive a zero for the material on that page.
- Read and sign the statement below. Wait for instructions to start the examination before continuing to the next page.

"I signify that the answers provided for this examination are my own and that I have not received any assistance from other students nor given any assistance to other students."

Signature: _____

- Do not open the examination booklet until instructed.

1. (25 points) Write short answers to the following questions.

   (a) (4 points) Write a segment of code to allocate an array of 32 `shorts` on the heap, and store the address of the allocated array into a pointer named `data`. Ensure that every item in the array is initialized to the value 0. Then, deallocate the array. Make sure to follow best practices when allocating and deallocating memory.

   (b) (3 points) For each statement below, circle whether it applies to the stack, the heap, both, or neither.

   Starts at a high memory address and grows downwards. Stack Heap Both Neither

   Variables allocated here exist until they are `free()`'d. Stack Heap Both Neither

   Contains variables that are local to functions. Stack Heap Both Neither

   Global and `static` local variables reside here. Stack Heap Both Neither

   Misuse can lead to memory leaks. Stack Heap Both Neither

   After a variable is destroyed, its address may be reused later. Stack Heap Both Neither

(c) (2 points) You are given the following code, along with the functions `void a()`, `void b()`, and `void c()`, whose contents are not shown:

```
void printnum(int *num_p) {
  assert(num_p != NULL);
  printf("%d\n", *num_p);
}

int main() {
  a();
  b();
  c();
  return 0;
}
```

Each of `a()`, `b()`, and `c()` call the `printnum()` function at least once, and possibly multiple times. When you run this program inside GDB, it prints the following message:

```
Program received signal SIGSEGV, Segmentation fault.
0x00005555555551b8 in printnum (num_p=0x6ee7ec15b604) at q1c.c:11
11          printf("%d\n", *num_p);
```

What command could you run in GDB to determine which function caused the error?

(d) (2 points) Define a pointer named `fptr` that points to a function that returns a pointer to an `int` and accepts two arguments, a pointer to a `char` and a `float`. Initialize the pointer to point to a function named `foobar`.

(e) (2 points) True or False: in a recursive function, all local variables are duplicated for each invocation of the function, except for `static` local variables.

(f) (2 points) Which of the following is true about the below definition?

```
const int *ptr = &some_int;
```

A. `ptr` points to an integer whose value cannot be modified, but `ptr` itself can be modified.

B. `ptr` points to an integer whose value can be modified, but `ptr` itself cannot be modified.

C. Neither `ptr` nor the integer it points to can be modified.

D. Both `ptr` and the integer it points to can be modified.

(g) (4 points) Consider the following code:

```
void swap(int *p1, int *p2) {
  int *temp = p1;
  p1 = p2;
  p2 = temp;
}

int main() {
  int odds[] = { 1, 3, 5, 7 };
  int evens[] = { 2, 4, 6, 8 };

  int *arr1 = odds;
  int *arr2 = evens;
  swap(arr1, arr2);

  assert(arr1 == evens && arr2 == odds);

  return 0;
}
```

Will the `assert` in `main()` fail? Why or why not? If it fails, rewrite the `swap()` function and the call to `swap()` within `main()` such that the `assert` does not fail.

(h) (3 points) Given the following macros:

```
#define TWICE (2)
#define DOUBLE(x) (TWICE * (x))
#define SQUARE(x) ((x) * (x))
#define DOUBLE_SQUARE(x) DOUBLE(SQUARE(x))
```

Write the expansion of the following line after the preprocessor step.

```
int a = DOUBLE_SQUARE(3 + 2);
```

(i) (3 points) Briefly describe why a callback would take a `void` pointer as a parameter.

2. (35 points) Write short answers to the following questions.

   (a) (5 points) Declare a structure, `kayak`, that holds one pointer to `char` (named `manufacturer`), one integer number named `length`, and a second integer named `max_weight`. This structure should be a valid doubly-linked list node. You may also declare your own type for this structure, if you wish, to slightly simplify the remaining questions.

(b) (5 points) Write a function, `create()`, that accepts three parameters—a string (the `manufacturer`), an integer value (the `length`), and a third integer value (the `max_weight`). Return a pointer to a newly allocated `struct kayak`, containing copies of all provided data. Include appropriate `assert()`s. Be sure to properly initialize all structure fields!

(c) (10 points) Write a function `reverse_list()` that returns nothing. It should accept one parameter: the *address* of a pointer to some `kayak` structure in a doubly-linked list.

Reverse the order of the list and set the pointer to be the new list head.

Include appropriate `assert()`ion checks. It is not an error if the list is empty.

(d) (15 points) Write a function `bankrupt()`, that accepts two parameters—the *address* of a pointer to a `struct kayak` somewhere in a doubly-linked list, and a string. Scan the entire list and remove all kayaks matching the specified manufacturer. Deallocate the associated memory. Return the number of kayaks removed (an integer). It is not an error for the list to be empty. Include appropriate `assert()`s. Upon return the pointer should point to the head of the remaining list, or `NULL` if empty.

3. (35 points) The following questions deal with structures that are dynamically allocated and form a binary tree.

   (a) (5 points) Declare a structure, `squirrel`, which would be a valid node in a binary tree containing in this order: an array of characters of `MAX_LEN` (the `sub_species`), a single-precision floating point number (the `avg_weight`), an integer (the `nut_rate`), and a second integer (the `avg_wake_time`). You may also declare your own type for this structure if you wish to slightly simplify the remaining questions.

(b) (10 points) Write a function, `sink_kayak()`, which accepts two parameters—a pointer to the root of a preexisting tree created using the `sub_species` as the comparison key and a pointer to a kayak from question 2. Find and return a pointer to the squirrel that is capable of sinking the kayak in the fewest days by filling it with acorns until it exceeds `max_weight`. Use `avg_weight` of the acorns each squirrel harvests, the `nut_rate` in acorns per hour, and the `avg_wake_time` in hours.

(c) (10 points) Define a *recursive* function, `mobilize_army()`, which accepts three parameters—a pointer to the root of a preexisting tree of squirrels, a pointer to a function that returns nothing and accepts one parameter, a pointer to a node, and a string. Traverse the tree in prefix order, invoking the function pointed to by the second parameter on any nodes with a `sub_species` matching the third argument.