Lab #12

Spring 2025

Requirements

In this lab, you'll expand your understanding of Heaps by implementing additional features and operations.

```
typedef struct Animal {
    int num_of_legs;
    char name[20];
} Animal;

typedef int (*AnimalCompareFunc)(Animal *a, Animal *b);

typedef struct Heap {
    Animal* data;
    struct Heap *left;
    struct Heap *right;
    int size;
} Heap;

typedef struct HeapInstance {
    Heap* root;
    AnimalCompareFunc cmp;
} HeapInstance;
```

1 create_animal

```
Animal* create_animal(int legs, const char* name);
```

Dynamically creates an Animal and returns a pointer to an Animal struct on success, or NULL on failure.

2 create_node

```
Heap* create_node(Animal* data);
```

Creates a Heap node with the given Animal data that was created and returns a pointer to a Heap struct, or NULL on failure.

3 insert

```
void insert(HeapInstance* heap, Animal* data);
```

Inserts the Animal data into the heap using the **`insert_key`** helper function.

4 insert_key

```
Heap* insert_key(Heap* root, Animal* data, AnimalCompareFunc cmp);
```

This function is a recursive function that inserts the data into the heap based on the passed function pointer (AnimalCompareFunc), which determines if it will be a min-heap or a max-heap. Returns the root node of the heap and NULL on failure. The function pointer AnimalCompareFunc, could be a comparator based on the ascending/descending animal name, or ascending/descending number of legs.

5 update_size

```
void update_size(Heap *node)
```

This function updates the size (the number of nodes of the subtree) of the Heap node, which is the subtree's root. The update is invoked during the **`insert_key`**.

6 swap_data

```
void swap_data(Heap* a, Heap* b)
```

This function swaps the Heap node pointers.

7 print_level_order

```
void print_level_order(Heap *root);
```

This function prints the Heap using a level-order traversal.

Hint: This function may be implemented as a recursive function. However, it may be simpler if you implement it by following a loop-based solution. Consider adding the children to an array/queue to print them in level-order. You may want to have a front and rear variable/pointer to the array/queue of children to control your loop.

8 compare_by_legs_min

```
// use for min-heap based on legs
```

```
int compare_by_legs_min(Animal* a, Animal* b);
```

This function returns 1 if the number of legs for ‘a’ less than ‘b’, and 0 otherwise. Must be passed to the function pointer **`AnimalCompareFunc cmp`**.

9 compare_by_legs_max

```
// use for max-heap based on legs
```

```
int compare_by_legs_max(Animal* a, Animal* b);
```

This function returns 1 if the number of legs for ‘a’ is greater than ‘b’, and 0 otherwise. Must be passed to the function pointer **AnimalCompareFunc cmp**.

10 compare_by_name_asc

```
int compare_by_name_asc(Animal* a, Animal* b);
```

This function returns 1 if the name for ‘a’ is less than ‘b’, and 0 otherwise. Must be passed to the function pointer **AnimalCompareFunc cmp**.

11 compare_by_name_desc

```
int compare_by_name_desc(Animal* a, Animal* b);
```

This function returns 1 if the name for ‘a’ is greater than ‘b’, and 0 otherwise. Must be passed to the function pointer **AnimalCompareFunc cmp**.

12 free_heap

```
void free_heap (HeapInstance* heap);
```

Frees all memory used by the HeapInstance and Heap nodes.

Rubric

- (1 pts) create_animal(): Correct dynamic allocation, name assignment, and error handling.
- (1 pts) create_node(): Proper creation of a heap node with valid linkage and data pointer.
- (2 pts) insert(): Correct invocation of insert_key and assignment back to root.
- (4 pts) insert_key(): Recursive insert logic, maintaining heap property using comparator.
- (1 pts) update_size(): Accurate size update of the heap node based on subtree sizes.
- (1 pts) swap_data(): Swaps two Heap nodes’ data pointers correctly.
- (4 pts) print_level_order(): Implements level-order traversal correctly using queue logic.
- (1 pts) compare_by_legs_min(): Returns correct comparison result (1 if a < b, 0 otherwise).
- (1 pts) compare_by_legs_max(): Returns correct comparison result (1 if a > b, 0 otherwise).
- (1 pts) compare_by_name_asc(): Lexicographic comparison: returns 1 if a < b.
- (1 pts) compare_by_name_desc(): Lexicographic comparison: returns 1 if a > b.
- (2 pts) free_heap(): Frees all allocated memory recursively and avoids leaks.