Assignment #5: Heap Implementation of a ToDo List

In this assignment, you will implement a To-Do List application with a Heap-based Priority Queue.

The application allows its users to manage their prioritized to-do lists using console commands for operations such as adding new tasks, retrieving, or removing the highest priority task. Additionally, users can save the list to a file or load the list from a file.

To complete the assignment, you must implement the logic for the toDoList application, complete the Heap implementation of the Priority Queue, and complete the HeapSort algorithm.

Heap Implementation & ToDo List Application

We will approach this new interface as we have approached previous interfaces such as stack, bag, and queue. In particular, we will simply add the priority queue operations to the dynamic array implementation. In addition, you'll use the 'compare' function for comparisons, as we did in the previous assignment (BST), except that this time, the compare function will be pass as a parameter (using a pointer to the compare function).
The toDo list must provide 7 options to the user upon start:

'l' to load to-do list from file (function provided for you)
's' to save to-do list to file (function provided for you)
'a' to add a new task to the to-do list
'g' to get the first task from the to-do list
'r' to remove the first task from the to-do list
'p' to print the to-do list (function provided for you)
'e' to exit the program

Once the user picks an option, your program should carry out the task and then present the user with the 7 options again. This should continue until the user selects 'e' to exit the program.

We have provided the functions to save and load the todo list to and from a file. These functions are in the toDoList.[h,c] files.

Refer to Worksheets 33 and 34 for more details regarding the heap implementation of the priority queue interface and the heapsort algorithm.

The provided files are described as below (and can be found in the archive):

dynamicArray.c : Implementation of dynamic array and heap-based priority queue. You will finish the functions for a heap-based priority queue in this file.
dynamicArray.h : Header for dynamicArray.c . This file should not be changed.
toDoList.c : Implementation of functions specialized for a to-do list application. You will need to complte the createTask and compare() functions in this file. NOTE: You'll need to use the strcpy(...) function to copy the description into your task structure.
toDoList.h: Header for toDoList.c. This file should not be changed. It contains the type for a Task which will be stored in the heap.
type.h: Header that includes the definition of TYPE (for this assignment, it should be void *). Do not change this file.
main.c : Implementation of the main() function, which controls the interactions between the user and the program. You will build the application logic here!
makefile : The program's makefile.
todo.txt : An example of a file storing a to-do list saved by function saveList(). Your saveList() function should generate a to-do list file with the same format. You can also use this file to test your loadList() function.
program_demo.txt : Examples of command lines showing how a user can interact with the program. This file is provided for your reference.
main2.c : Test file for the heap sort algorithm.

Note that the provided makefile can be used to make either "prog" or "sprog". These are two different executables. "prog" is the executable for your toDoList application (it compiles main.c) . "sprog" is the executable for testing your HeapSort (it compiles main2.c). So, to compile and test your toDoList, type make prog . To compile and test your heapsort, type make sprog

Heap Sort

You must ALSO complete the _buildHeap() and sortHeap() functions to implement the heap sort algorithm in the dynamicArray.c file (as noted above!).

IMPORTANT NOTES

Some of you may wish to complete the toDoList application logic before completing the Heap code. To simulate the heap, you can use the following functions in the dynamicArray.c file:

addDynArrOrd [to replace addHeap, this function uses binary search to keep a sorted array]
getDynArr(da,0) [to replace getMinHeap, this function allows you to get the value at index 0 of the sorted array]
removeAtDynArr(da, 0) [to replace removeMinHeap, this functions allows you to remove the value at index 0 of the sorted array]

These can only be used temporarily however. You MUST replace them with the proper Heap functions!

Note that in toDoList.c, the printList() function, which prints the priority queue in priority order, makes use of some of the heap functions. For now, they have been replaced with these substitute functions above. You MUST fix these. To find them, search for FIXME in toDoList.c

Finally, the makefile should compile, but with several warnings. Those warnings should go away once you've properly implemented all functionality.

Challenge Yourself (and earn a few extra points!) - Duplicate Priorities

Imagine adding 10 tasks to the priority queue, all having the same priority value, say 1. Simulate this by hand. What order are they removed when you call removeMinHeap 10 times? Is this what you would expect? If not, modify your implementation to produce a more appropriate result when duplicate tasks are added and subsequently removed.

Rubric

main toDoList application driver 24
_adjustHeap 10
addHeap 10
getMinHeap 5
removeMinHeap 5
createTask 5
buildHeap 3
heapSort 3
smallerIndex 5
printList 10
Compile/Style 20

What to turn in via TEACH

Please remember that your code must compile and execute on flip.engr.oregonstate.edu!!

main.c
dynamicArray.c
toDoList.c
ExtraCredit.zip(optional)

Turn In Please submit via TEACH. and Canvas.