CS261: HOMEWORK 3
Due 04/22/2016

Submit three files via the TEACH website:

`https://secure.engr.oregonstate.edu:8000/teach.php?type=want_auth`

General Instructions

This homework assignment consists of 2 parts, and will be graded for a total of 100 points. The points for each part are indicated at the beginning of the corresponding section in this writeup. For both part 1 and part 2, you are provided with header and implementation files. The implementation files should be completed without changing the existing content. If you have any questions regarding HW3, please email `cs261-001-sp16@engr.orst.edu`.

What to turn in

You should submit the following THREE completed files ALL AT THE SAME TIME:

- `list.c`
- `listbag.c`
- `cirListDeque.c`

Please use this file-naming convention. Make sure your code compiles with our makefile on the network server. We have zero tolerance for compiling errors. Try to compile on `flop.engr.oregonstate.edu`.

Design a number of test examples to thoroughly check for any errors in your code. For this purpose, we provide `main_deque.c` and `main_bag.c` files, which you should extend with more tests.

Part 1 – 50pts : Deque as Circularly-Doubly-Linked List

In this assignment, you will complete the implementation of a deque with Circularly-Doubly-Linked List with a sentinel. The list is circular, because the end points back to the beginning. Therefore, just one sentinel suffices. The sentinel is a special link, does not contain a value, and should not be removed. Using a sentinel makes some linked list operations easier and cleaner in implementation.

The header file and the implementation file, provided to you for this approach, are `cirListDeque.h` and `cirListDeque.c`, respectively. Some functions in the implementation (`cirListDeque.c`) have been completed. The comments for each function will help you understand what each function should be doing. Complete the remaining functions in `cirListDeque.c`. DO NOT change the provided functions and header files.

Provided Files for Part 1:

- `cirListDeque.h`
- `cirListDeque.c`

Scoring for Part 1:

1) `void initCirListDeque (struct cirListDeque *q)` = 2pts
2) `struct DLink * _createLink (TYPE val)` = 2pts
3) `void _addLinkAfter(struct cirListDeque *q, struct DLink *lnk, struct DLink *newLnk)` = 4pts
4) `void addBackCirListDeque (struct cirListDeque *q, TYPE val)` = 4pts
5) `void addFrontCirListDeque(struct cirListDeque *q, TYPE val)` = 4pts
6) `TYPE frontCirListDeque(struct cirListDeque *q)` = 2pts
7) `TYPE backCirListDeque(struct cirListDeque *q)` = 2pts
8) `void _removeLink(struct cirListDeque *q, struct DLink *lnk)` = 5pts
9) `void removeFrontCirListDeque (struct cirListDeque *q)` = 5pts
10) `void removeBackCirListDeque(struct cirListDeque *q)` = 5pts
11) `void freeCirListDeque(struct cirListDeque *q)` = 2pts
12) `int isEmptyCirListDeque(struct cirListDeque *q)` = 2pts
13) `void printCirListDeque(struct cirListDeque *q)` = 5pts
14) `void reverseCirListDeque(struct cirListDeque *q)` = 6pts

Part 2 – 50pts : List Bag with Recursions

The linked list implementation of a bag that we have discussed in class is an iterative version, in that the operations such as `contains()` and `remove()` are realized with iterative loops. An alternative is to implement these functions recursively.

Recall that recursion is used to implement the divide-and-conquer strategy, where the goal is to call the function itself, with smaller versions of the problem. For `contains()`, the recursive implementation is quite straightforward. The idea is to recursively call `contains()` each time with a smaller part of the list, where the base case is when the list contains only the sentinel. In the case of `contains()`, there is no ‘rebuilding’ upon return, it simply returns true or false.

The recursive `remove()` can also be implemented using the divide-and-conquer approach. In this case, the recursive process would operate on smaller lists, and re-build the list upon returning from the recursion. Specifically, the initial recursive call gets the entire list. If the element is not found in the front of the list, then the the list is broken into the current link being observed, and the rest of the list. The current link’s `next` is then set to the result of the recursive call on the rest of the list.

For this assignment, you are provided with the two header files and the two implementation files `type.h`, `listbag.h`, `list.c` and `listbag.c`, respectively. The goal is to implement all functionalities of a bag using the existing functions in `list.c` and `listbag.c`. Do not “invent the wheel”, but re-use existing functions, or functions that you have already written, as much as possible. For example, if you already have a function that initializes a general linked list, then you can immediately use a call to that function to initialize a bag. Some functions in the implementation files (`list.c` and `listbag.c`) have been completed. The comments for each function will help you understand what each function should be doing. Complete the remaining functions in both `list.c` and `listbag.c`. DO NOT change the provided functions and header files. Make sure that you implement the operations `contains()`, and `remove()` as recursive procedures using the divide-and-conquer approach. No partial credit will be given for any alternative implementation. Design a number of test examples to thoroughly check for any errors in your code.

Provided Files for Part 2:

- `type.h`
- `listbag.h`
- `list.c`
- `listbag.c`

Scoring for Part 2:

1) `void initList (struct list *lst)` = 2pts
2) `void _addLink(struct list *lst, struct DLink *lnk, TYPE v)` = 2pts
3) `void addFrontList(struct list *lst, TYPE e)` = 2pts
4) `void addBackList(struct list *lst, TYPE e)` = 2pts
5) `TYPE frontList (struct list *lst)` = 2pts
6) `TYPE backList(struct list *lst)` = 2pts
7) `void _removeLink(struct list *lst, struct DLink *lnk)` = 4pts
8) `void removeFrontList(struct list *lst)` = 2pts
9) `void removeBackList(struct list *lst)` = 2pts
10) `int isEmptyList(struct list *lst)` = 2pts
11) `int _contains_recursive(struct list *lst, struct DLink* current, TYPE e)` = 6pts
12) `void _remove_recursive(struct list *l, struct DLink* current, TYPE e, int* sz)` = 8pts
13) `void listRemove (struct list *lst, TYPE e)` = 2pts
14) `void freeList(struct list *lst)` = 2pts
15) `void addToBag(struct bag* b, TYPE val)` = 2pts
16) `void removeFromBag(struct bag* b, TYPE val)` = 2pts
17) `int bagContains(struct bag* b, TYPE val)` = 2pts
18) `int isEmptyBag(struct bag* b)` = 2pts
19) `void freeBag(struct bag *b)` = 2pts