ECE 2230: Computer Systems Engineering **Fall 2021**
Machine Problem 2 **Due: 11:59 pm, Thursday, September 9**

The goal of this machine problem is to build an implementation of a list ADT which includes a set of fundamental procedures to manipulate the list. We will implement the list using two-way linked lists, and define a general-purpose interface that will allow us to use this list ADT for the remainder of the semester. Utilizing the basic list ADT, we will extend the MP1 model of an exposure notification system (ENS) for saving and managing exposure records and testing for matches.

A key element in this assignment is to develop a well-designed abstraction for the list so that we can return to this interface and replace it with either (a) a different application that needs access to a list or (b) change the underlying mechanism for organizing information in a sorted list (e.g., other than a two-way linked list).

You are to write a C program that will maintain the **two** lists of ephemeral ID records. One list is sorted based on the ephemeral ID (EP ID) and has a maximum number of records that can be inserted. The other list is unsorted and has no limit on the size of the list. The list ADT provides a general list package that is used for both of the lists. The code **must** consist of the following files:

`lab2.c`	– contains the main() function, menu code for handling simple input and output, and any other functions that are not part of the ADT.
`ens_support.c`	– contains subroutines that support handling of EP ID records.
`linkedlist.c`	– The two-way linked list ADT. The interface functions must be defined exactly as described in `linkedlist.h`. You are allowed include additional functions but the interface cannot be changed.
`ens_support.h`	– The data structures for the specific format of the EP ID records and the prototype definitions.
`linkedlist.h`	– The data structures and prototype definitions for the list ADT.
`datatypes.h`	– Key definitions of the EP ID record structure
`makefile`	– Compiler commands for all code files

Part 1: The two-way linked list ADT

Your program must use a two-way linked list to implement the list ADT. A list is an object, and `linked_construct()` is used to generate a new list. There can be any number of lists active at one time. For a specific list, there is no limit on the maximum size, and elements can be added to the list until the system runs out of memory. Your program must not have a memory leak, that is, at no time should a block of memory that you allocated be inaccessible.

The list ADT is designed so that it does not depend on the type of data that is stored except through a very limited interface. Namely, to allow the list ADT to be used with a specific instance of a data structure, we define the data structure in a header file called `datatypes.h`. This allows the compiler to map a specific definition of a structure to a type we call `data_t`. All of the procedures for the list ADT operate on `data_t`. In future programming assignments we will reuse the list ADT by simply modifying the `datatypes.h` file and then recompiling.

```
/*
/* datatypes.h
 *
```

```
 * The data type that is stored in the list ADT is defined here.  We define a
 * single mapping that allows the list ADT to be defined in terms of a generic
 * data_t.
 *
 * data_t: The type of data that we want to store in the list
 */

typedef struct epid_record_tag {
    unsigned int ep_id;         // ephemeral ID
    unsigned int rec_num;       // EPID sequential record number
    unsigned int time_received; // time in seconds when record recevied
    int source_type;    // wireless type: 0 bluetooth 1 wifi 2 LTE 3 other
    unsigned int mac_addr;      // MAC address of source device
    int authenticated;  // true or false
    int privacy;        // mode 0 for none, 1 for encrypted
    int channel;        // 1-10
    float rssi;         // received signal strength in dB
    float band;         // received frequency band in GHz
} epid_rec_t;

/* the list ADT works on data of this type */
typedef epid_rec_t data_t;
```

The list ADT must have the following interface, defined in the file `linkedlist.h`. We refer to this as the *public* information.

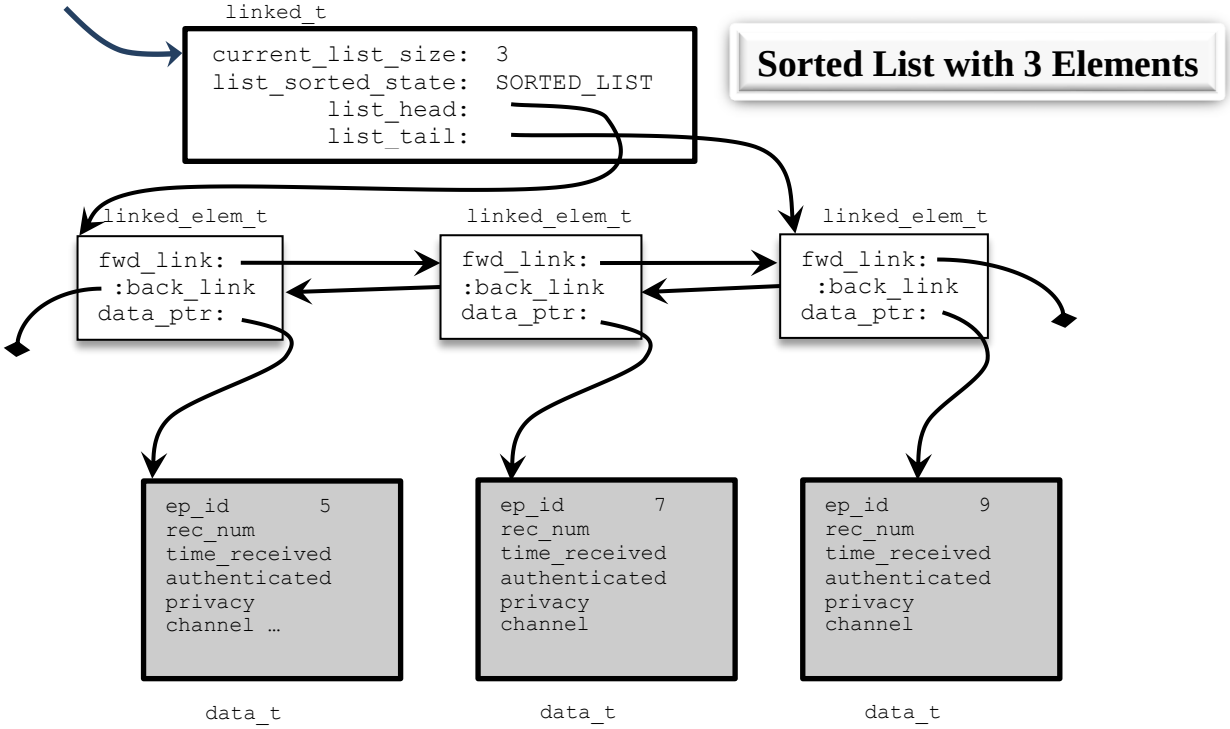
```
/* linkedlist.h
 *
 * Public functions for two-way linked list
 *
 * You should not need to change anything in this file.  If you do you
 * must get permission from the instructor.
 */

#include "datatypes.h"   // defines data_t

#define LLPOSITION_FRONT -987654
#define LLPOSITION_BACK  -234567

typedef struct linked_element_tag {
    // private members for linkedlist.c only
    data_t *data_ptr;
    struct linked_element_tag *back_link;
    struct linked_element_tag *fwd_link;
} linked_elem_t;

typedef struct list_header_tag {
    // private members for linkedlist.c only
    linked_elem_t *list_head;
    linked_elem_t *list_tail;
    int current_list_size;
    int list_sorted_state;
    // Private procedure for linkedlist.c only
    int (*comp_proc)(const data_t *, const data_t *);
} linked_t;

/* prototype definitions for functions in linkedlist.c */
linked_t * linked_construct(int (*fcomp)(const data_t *, const data_t *));
data_t *  linked_access(linked_t *list_ptr, int pos_index);
void      linked_destruct(linked_t *list_ptr);
void      linked_insert(linked_t *list_ptr, data_t *elem_ptr, int pos_index);
void      linked_insert_sorted(linked_t *list_ptr, data_t *elem_ptr);
```

```
data_t *  linked_remove(linked_t *list_ptr, int pos_index);
data_t *  linked_elem_find(linked_t *list_ptr, data_t *elem_ptr, int *pos_index,
        int (*fcomp)(const data_t *, const data_t *));
data_t *  linked_elem_remove_extremum(linked_t *list_ptr,
       int (*fcomp)(const data_t *, const data_t *));
int       linked_entries(linked_t *list_ptr);
```

The details of the procedures for the list ADT are defined in the comment blocks of the `linkedlist.c` template file. You can define additional *private* procedures to support the list ADT, but you must clearly document them as extensions.

When the list ADT is constructed, it must have the initial form as show in the figure below.

The list ADT supports two variations on a list. A list can be maintained in either sorted or unsorted order, depending on the procedures that are utilized. Multiple lists can be concurrently maintained. Here is an example of the list ADT data structure when three `epid_rec_t`'s have been added to the sorted list.



It is critical that the `linkedlist.c` procedures be designed to not depend on the details of our ephemeral ID records except through the definitions contained in `datatypes.h`. In particular, the letters "`ens_`" , and the member names in epid_rec_t *must not* be found in `linkedlist.c`. It is also critical that the internal details of the data structures in the `linkedlist.c` file are kept private and are not accessed outside of the `linkedlist.c` file. The members of the structures `linked_t` and `linked_elem_t` are private and their details *must not* be found in code outside of the `linkedlist.c` file. In particular, the letters "`->data_ptr`", "`->fwd_link`", "`->list_head`", "`->current_list_size`", etc. are considered private to the list ADT and *must not* be found in any `*.c` file except `linkedlist.c`.

Hint: Begin this assignment by completing all of Part 1 only. All work for Part 1 is contained in `linkedlist.c`. Test your `linkedlist.c` module using `driver.c`. Only when Part 1 is completed and tested should you move on to Part 2.

Part 2: The extended functions for ENS system

Our exposure notification system (ENS) from MP1 is extended to now use two lists: a contact list which is maintained in sorted order using the same requirements at for MP1 and an exposure list which is a simple queue that is not sorted. Both lists must use the new two-way linked list ADT. The ENS accepts new ephemeral ID details and inserts the record into the contact list. The contact list is sorted based on the EP ID, and just as in MP1, the number of records that can be stored in this list is specified by the **`CREATE`** command. The commands that operation on the sorted contact list are the same as for MP1 with the exception that the **`MATCH`** command is updated. When the **`MATCH`** command gives a list of EP IDs that have reported an infection, the program should look for a matching EP ID in the contact list. If found, the record should be removed from the contact list and appended to the end of the exposed list. The following commands can be given to the ENS and operate in the same was as MP1.

```
CREATE list-size
ADDID epid
QUERY epid
CLEAROLD time
TRIM
PRINT
```

In addition add the following user functions for an *exposure* list that is *unsorted* and does *not* have a limit on the number of elements in the list. The exposed list if treated as a queue in first-in-first-out (FIFO) order and there is no limit on the size of the queue. For a FIFO queue, a new record is added at the tail and removed from the head. The **`MATCH`** command processes a list of EP IDs that have known infections. If the matching EP ID is found in the contact list, it is removed and appended to the exposed list at the tail position. The **`RECEXP`** command counts the number of recent exposures. The command scans the exposure list and prints each EP ID record that has a `time_received` value greater than or equal to the `time` value. The list is not modified. The **`CLEAREXP`** command simply removes all records from the exposure list. The **`PRINTEXP`** command prints each record in the exposure list.

```
MATCH threshold
RECEXP time
CLEAREXP
PRINTEXP
```

STATS prints the size of both the sorted contact and the unsorted exposure list
QUIT ends the program and cleans up both lists.

Implement the procedures to complete these commands in the `ens_support.c` file. The procedures must be implemented using the list ADT as the mechanism to store the EP ID records. You can modify the design of the `ens_support` code and header files to support your design. However, you cannot add any new `printf` commands. Only `printfs` provided in the `ens_support.c` file can be used. (Note: while you can change the `ens_support.h` header file, you CANNOT change the `linkedlist.h` or `datatypes.h` files).

Submit all the files listed on page 1 **and** a test script to the ECE assign server. You submit by email to ece_assign@clemson.edu. Use as subject header ECE223-1,#2. The 223-1 identifies you as a member of Section 1 (the only section this semester). The #2 identifies this as the second assignment. When you submit to the assign server, verify that you get an automatically generated confirmation email within a few minutes. If you don't get a confirmation email listing all the attachments, your submission was not successful. You must include your files as attachments, and your email must be in text only format. You can make more than one submission but we will only grade the final submission. A re-submission must include all files. You cannot add files to an old submission.

To receive credit for this assignment your code must compile and at a minimum evaluate the provided `testinput.txt` and identically match the `expectedoutput.txt`. Code that does not compile or crashes before completing the example input will not be accepted or graded. If your code is not a perfect match you must contact the instructor or TA and fix your code. Code that correctly handles the example input but otherwise fails many other cases will be scored at 50% or less.

See the ECE 223 Programming Guide for additional requirements that apply to all programming assignments.

Work must be completed by each individual student, and see the course syllabus for additional policies.