Abstract Linked-List – Home Exercise

Part I:

Write an abstract Linked-List module that supports manipulating a linked-list of elements with the following functionality:

1. PUSH – adds an element to the end of a linked-list. Returns a pointer (void*) to the beginning of the linked-list.
2. POP – Removes the last element from the linked-list. Returns a pointer (void*) to the modified linked-list and sets its 2nd parameter to point to the element removed or NULL if the linked-list was empty.
3. SHIFT – Add an element at the beginning of a linked-list. Returns a pointer (void*) to the new beginning of the linked-list.
4. UNSHIFT – Remove the first Element on the linked-list. Returns a pointer (void*) to the modified linked-list and sets its 2nd parameter to point to the element removed or NULL if the linked-list was empty.
5. SIZE – Returns the number of elements in the linked-list.

A module implementation means that you write a .c file with all the code and a .h file with all the definitions, types and function signatures of your module.

Use the filenames as follows:

1. Abstract Linked-list module code: Abll.c
2. Abstract Linked-list module header: Abll.h

Note that the Abll.* module is not aware of the element values. In other words, it does not recognize the pay-load fields. However, the Abll.* module can ASSUME that the elements are ***linkable***. Meaning the structure that defines the element **must** contain a ***LINK*** structure as its first field (as seen in the lecture):

The LINK structure is defined in Abll.h as:

```
typedef struct _link {
      struct _link *next;
} LINK;
```

The module's function public signatures (those that can be called from outside the module) shall be:

1. PUSH – void *Abll_push(void *this, void *element);
2. POP – void *Abll_pop(void *this, void **element);
3. SHIFT – void *Abll_shift(void *this, void *element);
4. UNSHIFT – void *Abll_unshift(void *this, void **element);
5. SIZE – int Abll_size(void *this);

Where,

this – is a pointer to the linked-list (of ***Elements***) to operate on or NULL if the linked-list is empty

Example program to test Part I:

```
#include <stdio.h>
#include "Abll.h"

typedef struct _linkable_person
{
    LINK link;
    int id;
    char first[20], last[40];
    struct _date
    {
        int d, m, y;
    } born;
} Person, *PPerson;

int main(void)
{
    Person people[] = {
        {{NULL}, 1111111, "Israel", "Israeli", {5, 5, 1988}},
        {{NULL}, 2222222, "Douglas", "Adams", {1, 11, 1948}},
        {{NULL}, 3333333, "Tuval", "Shem Tov", {15, 7, 1992}},
        {{NULL}, 4444444, "Shimrit", "Cohen", {3, 12, 1980}}};

    PPerson guild = NULL, p;          // A guild is an association of craftsmen or merchants (it has a list of members)
    guild = Abll_push(guild, &people[0]);
    guild = Abll_push(guild, &people[1]);
    guild = Abll_shift(guild, &people[2]);
    guild = Abll_shift(guild, &people[3]);

    p = guild;
    printf("The members of the guild are:\n");
    for (int i=0 ; i<4 ; ++i, p=(PPerson)p->link.next)
        printf("id:%d[%s %s] =>\n", p->id, p->first, p->last);
    printf("(END)\n");

    for (int i=0 ; i<2 ; ++i)
    {
        guild = Abll_pop(guild, (void**)&p);
        printf("id:%d[%s %s]\n", p->id, p->first, p->last);
        guild = Abll_unshift(guild, (void**)&p);
        printf("id:%d[%s %s]\n", p->id, p->first, p->last);
    }

    if (guild)
        printf("And now guild contains: %p\n", guild);
    else
        printf("And now guild is EMPTY\n");
}
```

The expected output of this is:

```
The members of the guild are:
id:4444444[Shimrit Cohen] =>
id:3333333[Tuval Shem Tov] =>
id:1111111[Israel Israeli] =>
id:2222222[Douglas Adams] =>
(END)
id:2222222[Douglas Adams]
id:4444444[Shimrit Cohen]
id:1111111[Israel Israeli]
id:3333333[Tuval Shem Tov]
And now guild is EMPTY
```

Part II:

Add a filtering function to your Abll.* module. The filtering function accepts a ***filter function*** (as an argument) and an ***int*** reference value. The filtering function uses the (provided) filter function to filter out only those elements in the linked-list that pass the filter. I.e., running the filter on an element produces a **TRUE** or **FALSE** return value (depending on if the element passes the filter or not).

The job of the filtering function is to **REMOVE** elements that **DO NOT** pass the filter from the linked-list, so that only elements that ***pass*** the filter ***remain*** in the linked-list. The function also creates a linked-list of the elements that were removed and returns a pointer to that list via its last parameter or **NULL** if no elements were removed. The function returns a pointer to the modified (filtered) linked-list.

The filtering function signature is:

```
typedef enum _bool {FALSE, TRUE} BOOL;
void *AbFilter(void *this, BOOL (*FilterElement)(void*, int), int val, void **removed);
```

Note that the caller needs to specify the filterElement function because the Abll.* module does not know what fields (pay-load) an element contains. The filterElement function takes a void* that points to an element and an int value that is referenced in the filtering process. The ***AbFilter()*** function needs to execute the ***FilterElement()*** function on each element of the linked list and call it with the ***val*** parameter. Then it decides if the element passes the filter or not based on the return value from ***FilterElement().***

Example program to test part II:

```
#include <stdio.h>
#include "Abll.h"

BOOL filterCourses(void* ele, int val);
BOOL filterGrade(void* ele, int val);

typedef struct _linkable_student
{
    LINK link;
    int student_id;
    char first[10], last[12];
    int numberOfCourses;
    double average;
} Student, *PStudent;

void displayList(PStudent studentL)
{
    if (!studentL)
        printf("     <<Empty>>\n");
    for( ; studentL ; studentL=(PStudent)studentL->link.next)
        printf("%d %-10s %-12s has %d courses Avg:%g\n",
            studentL->student_id, studentL->first, studentL->last,
            studentL->numberOfCourses, studentL->average);

    printf("\n");
}
```

```
int main(void)
{
    Student shenkar[] = {
        {{NULL}, 1111111, "Amos", "Levi", 12, 82.8},
        {{NULL}, 2222222, "Tanchum", "Haroea", 4, 92.5},
        {{NULL}, 3333333, "Gali", "Shabat", 20, 62.9},
        {{NULL}, 4444444, "Hanoch", "Cohen", 18, 73.3},
        {{NULL}, 5555555, "Tali", "Bahat", 6, 72.2},
        {{NULL}, 6666666, "Sharon", "Shalev", 14, 81.4}
    };

    PStudent Tashaf = NULL;
    for (int i=0 ; i<sizeof(shenkar)/sizeof(Student); ++i)
        Tashaf = Abll_shift(Tashaf, &shenkar[i]);

    printf("Original List:\n");
    displayList(Tashaf);

    PStudent filtered, removed;

    filtered = AbFilter(Tashaf, filterCourses, 10, (void**)&removed);
    printf("\nAfter filtering for courses>10:\n");
    displayList(filtered);
    printf("Removed:\n");
    displayList(removed);

    filtered = AbFilter(filtered, filterGrade, 80, (void**)&removed);
    printf("\nAfter filtering the above for grade Average > 80:\n");
    displayList(filtered);
    printf("Removed:\n");
    displayList(removed);
}

BOOL filterCourses(void* ele, int val)
{
    PStudent student = (PStudent)ele;
    return student->numberOfCourses > val;
}

BOOL filterGrade(void* ele, int val)
{
    PStudent student = (PStudent)ele;
    return student->average > (double)val;
}
```

The expected output of this is:

```
Original List:
6666666 Sharon     Shalev       has 14 courses Avg:81.4
5555555 Tali       Bahat        has 6 courses Avg:72.2
4444444 Hanoch     Cohen        has 18 courses Avg:73.3
3333333 Gali       Shabat       has 20 courses Avg:62.9
2222222 Tanchum    Haroea       has 4 courses Avg:92.5
1111111 Amos       Levi         has 12 courses Avg:82.8
```

```
After filtering for courses>10:
6666666 Sharon     Shalev       has 14 courses Avg:81.4
4444444 Hanoch     Cohen        has 18 courses Avg:73.3
3333333 Gali       Shabat       has 20 courses Avg:62.9
1111111 Amos       Levi         has 12 courses Avg:82.8

Removed:
5555555 Tali       Bahat        has 6 courses Avg:72.2
2222222 Tanchum    Haroea       has 4 courses Avg:92.5


After filtering the above for grade Average > 80:
6666666 Sharon     Shalev       has 14 courses Avg:81.4
1111111 Amos       Levi         has 12 courses Avg:82.8

Removed:
4444444 Hanoch     Cohen        has 18 courses Avg:73.3
3333333 Gali       Shabat       has 20 courses Avg:62.9
```

Submit:

1. Submit in singles only
2. Each submit must be the sole work of the student that submitted the work
3. You only need to submit the module: **Abll.c** and **Abll.h** files
4. Make sure that your module operates correctly on the given test files, but also make up some tests of your own to verify the module is correct. Your work will be tested with the given test files and with other tests as well
5. Think about singular cases that need to work, like getting an empty list as an argument. Take care of what needs to happen in each case.
6. If you want to output an error message, do so **only** to the ***stderr*** channel.
7. Verify that your **Abll** module works with other pay-loads. Such as, **strings**, **complex numbers**, **points**, etc. Make up your own ***filter*** functions to test with.