תרגיל בית 3 חלק יבש – מבוא לתכנות מערכות

תאריך הגשה: 14.6.17

מגישים:

סהר כהן, ת.ז: 206824088
עודד ורמשטיין, ת.ז: 209320332

1) **קבוצה**: ברשת החברתית, לאוסף החברים לא קיים סדר בו הם מסודרים ואין שום חסם ידוע על גודל אוסף זה. בנוסף, ברשת החברתית כל ישות היא אלמנט ייחודי, שהרי ברשת חברתית מתוקנת לא נמצא שכפולי משתמשים. כמובן שגם אין משמעות לסדר האיברים, ולכן זהו מקרה אידאלי לשימוש בקבוצה.
2) **רשימה**: חברת הביטוח מעוניינת לשמור את מספרי הלקוחות בסדר מסוים שהוא הסדר בו נכנסו למערכת. כמו בדוגמה הקודמת, אין חסם ידוע על מספר הלקוחות. בנוסף, מחסנית לא תעמוד בדרישות מכיוון שאם ירצו למחוק את מספרו של לקוח מסוים אז יצטרכו למחוק את כל מספרי הלקוחות שנכנסו אחריו. לבסוף, נוכל לומר שרשימה תמלא משימה זו בצורה הטובה ביותר.
3) **מחסנית**: במועדון זה – מי שנכנס אחרון יוצא ראשון. זהו עקרון LIFO אשר נוכל לקבל הגדרתית דרך מחסנית. ניתן גם לממש מחסנית ללא חסמי גודל (למשל, באמצעות מערך משתנה או רשימה גנרית), ולכן מחסנית הינה אידאלית למקרה זה.

```c
/*
 * Comparison between two objects:
 * if a>b returns a positive number
 * if a<b returns a negative number
 * if a=b returns 0
 */
typedef int (*CmpFunction)(void* a,void* b);

int binaryfind(void** array, void* x,int len,CmpFunction compare) {
    int low = 0;
    int high = len-1;
    int mid = 0;
    int result = 0;
    while (low<high){
        mid = (low+high)/2;
        result = compare(array[mid], x);
        if(result>0) high = mid;
        else if(result<0){
            low = mid+1;
        }
        else{
            return mid;
        }
    }
    return -1;
}

/*  parameters:
    array: the array of the objects, type of each object is void*.
    x: the object that is needed to find in the array.
    len: the length of the array.
    compare: compares between two objects and changes according to the type of
    the object.
*/
```

```c
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include<stdio.h>
typedef struct node_t* Node;
struct  node_t{
    int n;
    Node next;
};
typedef int (*ConditionFunction)(int);

int IsPrime(int n);
int IsNotEven(int n);
Node nodeCreate(Node node);
Node concatLists(Node head1, Node head2, ConditionFunction condition);
static void printNode(Node head);
static Node createNodeOfArray(int *arr, int length);



int main(){
int arr1[6] = {1, 4, 5, 6, 13};
int arr2[7] = {13, 14, 15, 20, 21};

Node head1 = createNodeOfArray(arr1, 6);
Node head2 = createNodeOfArray(arr2, 7);

Node listOdd = concatLists(head1, head2, IsNotEven);
Node listPrime = concatLists(head1, head2, IsPrime);

printNode(listOdd);
printNode(listPrime);

return 0;
}
```

```
return 0;
}

int IsPrime(int n){
    int i;
    if (n<2) return 0;
    for(i = 2; i<=sqrt(n);i++){
        if(n%i == 0 ) return 0;
    }
    return 1;
}

int IsNotEven(int n){
    if((n%2) == 0) return 0;
    return 1;
}
Node concatLists(Node head1, Node head2, ConditionFunction condition){
    Node top_node = NULL;
    Node current_node = NULL;
    while(head1 != NULL){
        Node new_node = NULL;
        if(condition(head1->n) == 1){
            new_node = nodeCreate(new_node);
            new_node->n = head1->n;
            if(top_node == NULL){
                top_node = new_node;
                current_node = top_node;
            }
            current_node->next = new_node;
            current_node = new_node;

        }
        if(head1 != NULL) head1= head1->next;

    }
```

```
        }
    while(head2 != NULL){
            Node new_node = NULL;
            if(condition(head2->n) == 1){
                new_node = nodeCreate(new_node);
                new_node->n = head2->n;
                if(top_node == NULL){
                    top_node = new_node;
                    current_node = top_node;
                }
                current_node->next = new_node;
                current_node = new_node;

            }
            if(head2 != NULL) head2= head2->next;

        }
    return top_node;

}

Node nodeCreate(Node new_node){
     new_node = malloc(sizeof(*new_node));
    if(!new_node) return NULL;
new_node->n = 0;
new_node->next = NULL;
return new_node;
}

static Node createNodeOfArray(int *arr, int length){
    Node head = NULL;
    head = nodeCreate(head);
    int i = 0;
    Node node = head;
```

```
}

Node nodeCreate(Node new_node){
     new_node = malloc(sizeof(*new_node));
    if(!new_node) return NULL;
new_node->n = 0;
new_node->next = NULL;
return new_node;
}

static Node createNodeOfArray(int *arr, int length){
    Node head = NULL;
    head = nodeCreate(head);
    int i = 0;
    Node node = head;
    while(i< length){
        int *num = malloc(sizeof(*num));
        if(num == NULL) return NULL;
        *num = arr[i];
        node->n = *num;
        if(i != length-1){
            node->next = nodeCreate(node->next);
            node = node->next;
        }
        i++;
    }
    return head;
}
static void printNode(Node head){
    while(head != NULL){
        printf(" %d", head->n);
        head = head->next;
    }
    printf("\n");
}
```