מת״מ - 234122
תרגיל בית #3 - חלק יבש

- בחירת מבני נתונים:

1. רשת חברתית פייסבוק, המבנה המתאים ביותר הינו קבוצה(Set).
לכל משתמש ברשת יש צורך בשמירת כל החברים שלו, אשר נבדלים ע״י מזהים ייחודיים. מבנה הקבוצה דורש שכל איבר בקבוצה יהיה ייחודי, כלומר לא יכול להיות שני איברים בקבוצה בעלי אותם מזהים.
לכן נבחר במבנה הקבוצה כך שלכל משתמש ברשת יהיה מבנה נתונים מסוג קבוצה, כך שיכיל את כלל החברים שלו בלי חזרות(כל חבר יהיה ייחודי).
2. חברת ביטוח, המבנה המתאים ביותר הינו רשימה(List).
הדרישה שיש לשמור את מספרי הלקוחות אשר רשומים לחברה ע״פ סדר ההרשמה למערכת, מסתדר מצוין עם מבנה הרשימה מאפשר להוסיף איברים חדשים לרשימה במקום מסוים(לדוגמא, בהתחלה או בסוף כמו שנדרש).
בנוסף נשמר סדר האיברים ברשימה, כנדרש.
3. שמירת חפצים במועדון, המבנה המתאים ביותר הינו מחסנית(Stack).
קיימת הנחה כי לקוחות המגיעים מאוחר יותר, עוזבים את המועדון מוקדם מלקוחות אחרים. לכן יש כאן חשיבות למי נכנס ראשון ומי נכנס אחרון וכך האחרון יצא ראשון והמבקר שנכנס ראשון יצא אחרון.
כלומר זה מבנה FILO(First In Last Out), המאפיין במדויק את מבנה הנתונים מסוג מחסנית.

- תכנות גנרי:

```
int binaryFind(Element[] elements_array, int array_size, Element to_find,
                                CompareElements CompareElement);
```

פונקציה זו מקבלת את הפרמטרים הבאים:

- elements_array: מערך עצמים מטיפוס לא ידוע(Element).
- array_size: גודל מערך העצמים.
- to_find: העצם אותו צריך למצוא.
- CompareElement: הפונקציה בעזרתה מתבצע ההשוואה של כל שני עצמים.
  הפונקציה צריך לקבל שני איברים מטיפוס Element ולהחזיר 0 במידה והם שווים,
  במידה והראשון גדול מהשני 1, במידה והשני גדול מהראשון -1.

```
typedef int(*CompareElements)(Element, Element);
```

קוד:

```
typedef void* Element;
typedef int(*CompareElements)(Element, Element);

int binaryFindRec(Element[] elements_array, int low, int high, Element to_find,
            CompareElements CompareElement){
        assert(elements_array != NULL && array_size != 0 && to_find != NULL &&
                CompareElement != NULL);

        if (low > high){
                return -1;
        }
        int mid = (low + high)/2;
        if (CompareElement(elements_array[mid],to_find) == 0){
                return mid;
        } else if(CompareElement(elements_array[mid],to_find) == 1){
                return binaryFindRec(elements_array, low, mid - 1, to_find,
                                CompareElement);
        } else if (CompareElement(elements_array[mid],to_find) == -1){
                return binaryFindRec(elements_array, mid + 1, high, to_find,
                                CompareElement);
        }
        assert(0); // cannot get here, debug check.
        return -1;
}

int binaryFind(Element[] elements_array, int array_size, Element to_find,
                CompareElements CompareElement){
        if (elements_array == NULL || array_size == 0 || to_find == NULL ||
                CompareElement == NULL){
                return -1;
        }
        return binaryFindRec(elements_array, 0, array_size-1, to_find,
                        CompareElement);
}
```

- רשימות מקושרות:

```
typedef bool(*CompareElements)(int);

Node concatList(Node head1, Node head2, CompareElements CompareElement){
	if (CompareElement == NULL){
		return NULL;
	}

	Node new_list = malloc(sizeof(new_list*));
	if (new_list == NULL){
		return NULL;
	}
	Node ptr_head = head1;
	Node ptr_new_list = new_list;
	while (ptr_head != NULL){
		if (CompareElement(ptr_head->n) == true){
			ptr_new_list->n = ptr_head->n;
			ptr_new_list = ptr_new_list->next;
			ptr_head = ptr_head->next;
		}
	}
	ptr_head = head2;
	while (ptr_head != NULL){
		if (CompareElement(ptr_head->n) == true){
			ptr_new_list->n = ptr_head->n;
			ptr_new_list = ptr_new_list->next;
			ptr_head = ptr_head->next;
		}
	}
	return new_list;
}
```

המשך רשימות מקושרות -

```
void someProg(){
	Node list1 = malloc(sizeof(new_list*));
	if (new_list == NULL){
		return NULL;
	}
	for (int i = 3; i>0; i--){
		if (addToStart(list1, i) == false){
			destroyList(list1);
			return;
		}
	}
	Node list2 = malloc(sizeof(new_list*));
	if (new_list == NULL){
		destroyList(list1);
		return NULL;
	}
	for (int i = 3; i>0; i--){
		if (addToStart(list1, i) == false){
			destroyList(list1);
			destroyList(list2);
			return;
		}
	}
	Node odd_nums = concatList(list1, list2, oddNumber);
	Node even_nums = concatList(list1, list2, evenNumber);
	destroyList(list1);
	destroyList(list2);
	destroyList(odd_nums);
	destroyList(even_nums);
	return;
}
```

```
bool addToStart(Node list, int num){
	if (list == NULL){
		return false;
	}
	Node new_node =
malloc(sizeof(new_node*));
	if (new_node == NULL){
		reutrn false;
	}
	new_node->next = list;
	list = new_node;
	return true;
}
```

```
bool oddNumber(int num){
	if (num%2 == 0){
		return true;
	}
	return false;
}

bool isPrime(int num){
   int prime = 1;
   for(i=2; i <= sqrt(num); ++i){
      if(num%i == 0) {
         prime = 0;
         break;
      }
   }
   if (prime) return true;
   return false;
}
```