Харківський національний університет імені В.Н. Каразіна
Факультет комп’ютерних наук

Звіт
з лабораторної роботи №1
дисципліна: «Операційні системи»

Виконала: студентка групи КС34
Євстігнєєва Марія Олексіївна
Перевірив: Паршинцев Богдан Володимирович

Харків 2022

Задание №0

В пространстве заданы n материальных точек. С некоторого момента точка с наименьшей массой исчезает, передавая свою массу ближайшей к ней точке. Так продолжается до тех пор, пока не останется одна точка. Реализовать этот процесс и найти оставшуюся точку.

Результати:

```
Enter the amount of points: 10

1 - Random array
2 - Enter the array
Enter the number: 1

Your array:
[x = 37, y = 91, weight = 66
x = 99, y = 4, weight = 91
x = 30, y = 1, weight = 93
x = 29, y = 32, weight = 77
x = 3, y = 79, weight = 32
x = 52, y = 82, weight = 69
x = 21, y = 55, weight = 37
x = 8, y = 46, weight = 74
x = 67, y = 26, weight = 6
x = 35, y = 10, weight = 7]

The lightest point: x = 67, y = 26, weight = 6

The closest point: x = 35, y = 10, weight = 7

The lightest point disappeared and gave its weight to closest point:
x = 35, y = 10, weight = 13

The lightest point: x = 35, y = 10, weight = 13

The closest point: x = 30, y = 1, weight = 93

The lightest point disappeared and gave its weight to closest point:
x = 30, y = 1, weight = 106

The lightest point: x = 3, y = 79, weight = 32

The closest point: x = 21, y = 55, weight = 37

The lightest point disappeared and gave its weight to closest point:
x = 21, y = 55, weight = 69

The lightest point: x = 37, y = 91, weight = 66

The closest point: x = 52, y = 82, weight = 69

The lightest point disappeared and gave its weight to closest point:
x = 52, y = 82, weight = 135
```

```
The lightest point: x = 21, y = 55, weight = 69

The closest point: x = 8, y = 46, weight = 74

The lightest point disappeared and gave its weight to closest point:
x = 8, y = 46, weight = 143

The lightest point: x = 29, y = 32, weight = 77

The closest point: x = 8, y = 46, weight = 143

The lightest point disappeared and gave its weight to closest point:
x = 8, y = 46, weight = 220

The lightest point: x = 99, y = 4, weight = 91

The closest point: x = 30, y = 1, weight = 106

The lightest point disappeared and gave its weight to closest point:
x = 30, y = 1, weight = 197

The lightest point: x = 52, y = 82, weight = 135

The closest point: x = 8, y = 46, weight = 220

The lightest point disappeared and gave its weight to closest point:
x = 8, y = 46, weight = 355

The lightest point: x = 30, y = 1, weight = 197

The closest point: x = 8, y = 46, weight = 355

The lightest point disappeared and gave its weight to closest point:
x = 8, y = 46, weight = 552

------------------------------------------------------

The final point is: x = 8, y = 46, weight = 552
```

Код:

```
#include "task1.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

struct point {
   int x, y, weight;
};
int size;

int main(int argc, char const *argv[])
{
        srand(time(NULL));
        int n;
        do {
                printf("Enter the amount of points: ");
                scanf("%d", &n);
                if(n < 5) printf("\nThe number can't be less than 5! Try again!\n");
        } while (n < 5);
        size = n;
        struct point points [size];
        printf("\n1 - Random array\n2 - Enter the array\n");
        int choice;
        do {
                printf("Enter the number: ");
                scanf("%d", &choice);
                if(choice != 1 && choice != 2) break;
        } while (choice != 1 && choice != 2);
```

```
	if(choice == 1) {
		for(int i = 0; i < n; i++) {
			points[i].x = rand()%100;
			points[i].y = rand()%100;
			points[i].weight = rand()%100;
		}
	}
	else {
		for(int i = 0; i < n; i++) {
			int x, y, weight;
			printf("\nPoint number %d\n", i+1);
			printf("Enter the x: ");
			scanf("%d", &x);
			printf("Enter the y: ");
			scanf("%d", &y);
			printf("Enter the weight: ");
			scanf("%d", &weight);
			points[i].x = x;
			points[i].y = y;
			points[i].weight = weight;
		}
	}
	printf("\nYour array: \n[");
	for(int i = 0; i < n; i++) {
		if(i != 0) printf("\n");
		printf("x = %d, y = %d, weight = %d", points[i].x, points[i].y,
points[i].weight);
	}
	printf("]\n");
```

```
	while(size != 1) {
		int indLightest = 0;
		for(int i = 0; i < size; i++) {
			for(int j = 0; j < size; j++) {
				if(points[i].weight < points[indLightest].weight)
					indLightest = i;
			}
		}
		printf("\nThe lightest point: x = %d, y = %d, weight = %d\n", points[indLightest].x, points[indLightest].y, points[indLightest].weight);
		point_disappear(&points[0], indLightest);
	}
	printf("\n_____\n");
	printf("\nThe final point is: x = %d, y = %d, weight = %d\n", points[0].x, points[0].y, points[0].weight);
	return 0;
}

void point_disappear(struct point* points, int index) {
	struct point ourPoint = points[index];
	int indClos = closest_point(points, index);
	if(indClos == index) return;
	struct point closP = points[indClos];
	printf("\nThe closest point: x = %d, y = %d, weight = %d\n", points[indClos].x, points[indClos].y, points[indClos].weight);
	points[indClos].weight += ourPoint.weight;
```

```
	printf("\nThe lightest point disappeared and gave its weight to closest point:\nx = %d, y = %d, weight = %d\n", points[indClos].x, points[indClos].y, points[indClos].weight);
	for(int i = index; i < size - 1; i++) {
		points[i] = points[i+1];
	}
	size--;
}

float distance(struct point pOne, struct point pTwo) {
	float dist = powf((powf(pTwo.x-pOne.x, 2)+powf(pTwo.y-pOne.y, 2)), 0.5);
	return dist;
}

int closest_point(struct point* points, int index) {
	struct point ourPoint = points[index];
	float closest;
	struct point closP;
	int indClos = 0;
	if(index != 0) {
		closest = distance(ourPoint, points[0]);
		closP = points[0];
		indClos = 0;
	}
	else if (index == 0 && size > 1) {
		closest = distance(ourPoint, points[1]);
		closP = points[1];
		indClos = 1;
```

```
	}
	else if (index == 0 && size == 1) return index; // if only one point left
	float dist = 0;
	for(int i = 0; i < size; i++){
		if (i != index) {
			dist = distance(ourPoint, points[i]);
			if (dist < closest) {
				closest = dist;
				closP = points[i];
				indClos = i;
			}
		}
	}
	return indClos;
}
```

Header файл:

```
#ifndef functions_h_
#define functions_h_

struct point;
void point_disappear(struct point*, int);
int closest_point(struct point*, int);
float distance (struct point, struct point);

#endif
```

Makefile:

```
task1: task1.c task1.h
	gcc $^ -o $@ -lm
```

Задание №4

Создайте аналог массива — списка (*ArrayList*) языка *Java*. Реализуйте следующую функциональность:

1. добавление элемента в конец списка — метод *add(item)*;
2. вставка элемента в середину списка — метод *insert(index, item)*;
3. количество элементов в массиве — метод *size()*;
4. удаление элементов по индексу — метод *remove(index)*;
5. изменение значения существующего элемента — метод *set(index, item)*;
6. получение значения заданного элемента — метод *get(index)*;

Результати:

```
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>1
Enter number: 1
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>1
Enter number: 2
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>1
Enter number: 3
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>7
1 2 3
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>1
Enter number: 1
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
```

```
>>>4
Enter pos: 1
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>1
Enter number: 7
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>7
1 3 1 7
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>2
Enter pos: 1
Enter number: 7
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>7
1 7 3 1 7
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>3
Size: 5
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
```

```
>>>4
Enter pos: 2
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>7
1 7 1 7
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>5
Enter pos: 3
Enter new number: 12
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>7
1 7 1 12
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
>>>6
Enter pos: 5
Incorrect position
Enter pos: 2
Num: 1
1. Add
2. Insert
3. Size
4. Remove
5. Set
6. Get
7. Print
0. Exit
```

Код:

Main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stddef.h>

int main(){

  int size;
  while(1){
    printf("Enter the start size: ");
    scanf("%d", &size);
    if(size < 1)
      printf("Array can't have less than 1 element\n");
    else
      break;
  }
  struct array * arr = array_create(size);
  int choice;
  int act_size = 0;

  while(1){
    printf("1. Add to the end\n2. Insert\n3. Check the size\n4. Remove the element\n5. Set a new value\n6. Get value\n7. Print the array\n0. Exit\n Enter the number: ");
    scanf("%d", &choice);
    switch(choice){
      case 1:{
```

```
      int num;
      printf("Enter number: ");
      scanf("%d", &num);
      array_element_set(arr, act_size, &num, sizeof num);
      act_size++;
      break;
    }
    case 2:{
      if(act_size == 0) printf("\nEnter the element firstly\n");
      int num, pos;
      while(1){
        printf("Enter pos: ");
        scanf("%d", &pos);
        if(pos < 0 || pos > act_size)
          printf("Incorrect position\n");
        else
          break;
      }
      printf("Enter number: ");
      scanf("%d", &num);
      for(int i = act_size + 1; i > pos; i--) {
        array_element_set(arr, i, array_element_get(i-1), sizeof
array_element_get(i-1));
      }
      array_element_set(arr, pos, &num, sizeof num);
      act_size++;
      break;
    }
    case 3:{
```

```
      printf("\nSize: %d\n", array_size(arr));
      break;
    }
    case 4:{
      if(act_size == 0) printf("\nEnter the element firstly\n");
      int pos;
      while(1){
        printf("Enter pos: ");
        scanf("%d", &pos);
        if(pos < 0 || pos >= act_size)
          printf("Incorrect position\n");
        else
          break;
      }
      for(int i = pos; i < act_size - 1; i++) {
        array_element_set(arr, i, array_element_get(i+1), sizeof
array_element_get(i+1));
      }
      act_size--;
      break;
    }
    case 5:{
      if(act_size == 0) printf("\nEnter the element firstly\n");
      int num, pos;
      while(1){
        printf("Enter pos: ");
        scanf("%d", &pos);
        if(pos < 0 || pos > act_size)
          printf("Incorrect position\n");
```

```
    else
      break;
  }
  printf("Enter new number: ");
  scanf("%d", &num);
  array_element_set(arr, pos, &num, sizeof num);
  break;
}
case 6:{
  if(act_size == 0) printf("\nEnter the element firstly\n");
  int pos;
  while(1){
    printf("Enter pos: ");
    scanf("%d", &pos);
    if(pos < 0 || pos >= act_size)
      printf("Incorrect position\n");
    else
      break;
  }
  printf("\nElement: %d\n", array_element_get(arr, pos));
  break;
}
case 7:{
  printf("\nArray:\n");
  for(int i = 0; i < array_size(arr); i++) {
    printf(" %d ", array_element_get(arr, i));
  }
  break;
}
```

```
      case 0:{
        return 0;
      }
      default:{
        printf("Error\n");
        break;
      }
    }
  }
}
```

Array.h:

```
#ifndef ARRAY_H_INCLUDED
#define ARRAY_H_INCLUDED

#include "element.h"
#include <stdbool.h>
#include <stddef.h>

struct array;

struct array * array_create(size_t size);
size_t array_size(struct array * ar);
bool array_element_set(struct array * ar, size_t ind, void * val, size_t size);
void * array_element_get(struct array * ar, size_t ind);
void array_destroy(struct array ** el);
void map(struct array * el, void (*f)(void * arg));

#endif // ARRAY_H_INCLUDED
```

Element.h:

```
#ifndef ELEMENT_H_INCLUDED
#define ELEMENT_H_INCLUDED

#include <stdbool.h>
#include <stdbool.h>
#include <stddef.h>

struct m_elem;

struct m_elem * element_create();
bool element_set(struct m_elem * el, void * val, size_t size);
void * element_get(struct m_elem * el);
void element_destroy(struct m_elem ** el);
void apply(struct m_elem * el, void (*f)(void * arg));

#endif // ELEMENT_H_INCLUDED
```

Array.c:

```
#include <stdlib.h>
#include <stdio.h>

#include "array.h"

struct array {
    struct m_elem ** array;
    size_t size;
};



struct array * array_create(size_t size) {
```

```
    struct array * res = malloc(sizeof (struct array));
    if (!res) {
        fprintf(stderr, "Array allocation error\n");
        return res;
    }
        res->size = size;
        res->array = (struct m_elem **)calloc(size, sizeof(struct m_elem*));
        if (!(res->array)) {
                fprintf(stderr, "Allocation Memory Error (inner buuffer)\n");
                return res;
        }
        return res;
}

size_t array_size(struct array * ar) {
    return ar->size;
}


bool array_element_set(struct array * ar, size_t ind, void * val, size_t size) {
    if (ind < ar->size) {
          if (!ar->array[ind]) ar->array[ind] = element_create();
        return element_set(ar->array[ind], val, size);
    }
        fprintf(stderr, "Index value error (index %d >= array size %d)!!!\n",
(int)ind, (int)ar->size);
    return false;
}
```

```
void * array_element_get(struct array * ar, size_t ind) {
    if (ind < ar->size) {
        return element_get(ar->array[ind]);
    }
    return NULL;
}


void array_destroy(struct array ** el) {
    size_t i;
    for (i = 0; i < (*el)->size; i++) {
        if ((*el)->array[i]) element_destroy(((*el)->array+i));
    }
    free(*el);
    *el = NULL;
}


void map(struct array * el, void (*f)(void * arg)) {
    size_t i;
    for (i = 0; i < el->size; i++) {
        //f(element_get(el->array[i]));
        apply(el->array[i], f);
    }
}
```

Element.c:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>

#include "element.h"

struct m_elem {
    void * elem;
};

struct m_elem * element_create(){
    struct m_elem * res = malloc(sizeof(struct m_elem));
    if (!res) {
        fprintf(stderr, "Element Allocation Error\n");
    } else {
        res->elem = NULL;
    }
    return res;
}


bool element_set(struct m_elem * el, void * val, size_t size) {
    if (el->elem) {
        free(el->elem);
        el->elem = NULL;
    }
    el->elem = malloc(size);
         if (!el->elem) {
                fprintf(stderr, "Allocation memory error!!!\n");
                return false;
         }
```

```
        memcpy(el->elem, val, size);
        return true;
}

void * element_get(struct m_elem * el) {
   return el->elem;
}

void element_destroy(struct m_elem ** el) {
   free((*el)->elem);
        (*el)->elem = NULL;
        free(*el);
        (*el) = NULL;
}

void apply(struct m_elem * el, void (*f)(void * arg)) {
   f(el->elem);
}
```