|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ДИСЦИПЛИНА «Архитектура ЭВМ»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 1**

|  |  |
| --- | --- |
| **Студент**  Чалый А. А.  **Группа** ИУ7 – 52 Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Попов А.Ю. |  |

Москва.

2020 г.

# Оглавление

[Оглавление 2](#_Toc1)

[Task 1 3](#_Toc2)

[Задание 1 3](#_Toc3)

[Задание 2 12](#_Toc4)

[Задание 3 19](#_Toc5)

[Task 2 29](#_Toc6)

[Задание 1 29](#_Toc7)

[Задание 2 31](#_Toc8)

[Задание 3 36](#_Toc9)

# Task 1

## **Задание 1**

Создать хранилище в оперативной памяти для хранения информации о детях.

Необходимо хранить информацию о ребенке: фамилия и возраст.

Необходимо обеспечить уникальность фамилий детей.

Реализовать функции:

* CREATE READ UPDATE DELETE для детей в хранилище
* Получение среднего возраста детей
* Получение информации о самом старшем ребенке
* Получение информации о детях, возраст которых входит в заданный отрезок
* Получение информации о детях, фамилия которых начинается с заданной буквы
* Получение информации о детях, фамилия которых длиннее заданного количества символов
* Получение информации о детях, фамилия которых начинается с гласной буквы

Листинг 1. задание 1

**var** childlist;

**function** READ(index)

{

console.log(**"\n\tSurname: "** + childlist[index].surname + **"\n\tAge: "** + childlist[index].age + **"\n"**);

**return** 0;

}

**function** averageAge()

{

let agesum;

agesum = childlist.reduce(**function** (acc, obj) { **return** acc + obj.age;}, 0);

console.log(**"Average age is: "**, agesum / childlist.length);

**return** agesum / childlist.length;

}

**function** Olderchild\_info()

{

let age = -1;

let index\_oldest;

**if** (childlist.length === 0)

{

console.log(**"Childlist is empty!"**);

**return**;

}

**for** (let i = 0; i < childlist.length; i++)

{

**if** (childlist[i].age > age)

{

age = childlist[i].age;

index\_oldest = i;

}

}

console.log(**"Oldest child info:"**);

READ(index\_oldest);

**return** index\_oldest;

}

**function** childinfo\_between(lend, rend)

{

**if** (-1 < lend < childlist.length && -1 < rend < childlist.length && lend <= rend)

{

**for** (let i = lend; i <= rend; i++)

{

console.log(**"Child at index "** + i);

READ(i);

}

**return** 0;

}

**else**

{

console.log(**"Left or right end of interval is incorrect"**);

**return** -1;

}

}

**function** childinfo\_surnameAt(letter)

{

console.log(**"Child(s) wich surname starts at "** + letter);

**for** (let i = 0; i < childlist.length; i++)

{

**if** (childlist[i].surname.startsWith(letter) === **true**)

{

READ(i);

}

}

**return** 0;

}

**function** childinfo\_surnameLongerThan(cap)

{

console.log(**"Child(s) wich surname longer than "** + cap);

**for** (let i = 0; i < childlist.length; i++)

{

**if** (childlist[i].surname.length > cap)

{

READ(i);

}

}

**return** 0;

}

**function** childinfo\_surnameStartsAtVowel(vowel)

{

let vowels = [**'A'**, **'E'**, **'O'**, **'U'**, **'I'**, **'Y'**, **'А'**, **'О'**, **'Э'**, **'И'**, **'У'**, **'Ы'**, **'Е'**, **'Ё'**, **'Ю'**, **'Я'**];

**if** (vowels.includes(vowel) === **true**)

{

console.log(**"Child(s) wich surname starts at vowel "** + vowel);

**for** (let i = 0; i < childlist.length; i++)

{

**if** (childlist[i].surname.startsWith(vowel) === **true**)

{

READ(i);

}

}

}

**else**

{

console.log(**"Argument is not a vowel!"**);

**return** -1;

}

**return** 0;

}

module.exports = {

CREATE: **function**()

{

childlist = [];

**return this**;

},

READ: **function**(index)

{

console.log(**"\n\tSurname: "** + childlist[index].surname + **"\n\tAge: "** + childlist[index].age + **"\n"**);

**return this**;

},

UPDATE: **function**(child)

{

**if** (childlist.every(obj => obj.surname !== child.surname))

{

childlist.push(child);

console.log(**"Added child to childlist!"**);

}

**else**

{

console.log(**"Child with this name already exists!"**);

}

**return this**;

},

DELETE: **function**(index)

{

console.log(**"This child was deleted:"**);

console.log(**"\n\tSurname: "** + childlist[index].surname + **"\n\tAge: "** + childlist[index].age);

childlist.splice(index, 1);

**return this**;

},

averageAge: averageAge,

Olderchild\_info: Olderchild\_info,

childinfo\_between: childinfo\_between,

childinfo\_surnameAt: childinfo\_surnameAt,

childinfo\_surnameLongerThan: childinfo\_surnameLongerThan,

childinfo\_surnameStartsAtVowel: childinfo\_surnameStartsAtVowel

}

Листинг 2. Тесты к заданию 1

**var** list = require(**"./task1\_1.js"**);

childs = [

{

surname: **'Сэм'**,

age: 2

},

{

surname: **'Эмили'**,

age: 5

},

{

surname: **'Мэт'**,

age: 7

},

{

surname: **'Брэд'**,

age: 12

},

{

surname: **'Шерри'**,

age: 3

},

{

surname: **'Керри'**,

age: 4

},

{

surname: **'Стелла'**,

age: 16

}

];

**var** result = list.CREATE().UPDATE(childs[0]).UPDATE(childs[1]).UPDATE(childs[2])

.UPDATE(childs[3]).UPDATE(childs[4]).UPDATE(childs[5]).UPDATE(childs[6])

result = list.DELETE(6).DELETE(2);

result.averageAge();

result.Olderchild\_info();

result.childinfo\_between(3, 5);

result.childinfo\_surnameAt(**'С'**);

result.childinfo\_surnameLongerThan(3);

result.childinfo\_surnameStartsAtVowel(**'Э'**);

## **Задание 2**

Создать хранилище в оперативной памяти для хранения информации о студентах.

Необходимо хранить информацию о студенте: название группы, номер студенческого билета, оценки по программированию.

Необходимо обеспечить уникальность номеров студенческих билетов.

Реализовать функции:

* CREATE READ UPDATE DELETE для студентов в хранилище
* Получение средней оценки заданного студента
* Получение информации о студентах в заданной группе
* Получение студента, у которого наибольшее количество оценок в заданной группе
* Получение студента, у которого нет оценок

Листинг 3. Задание 2

**var** studentlist;

**function** READ(index)

{

console.log(**"\n\tGroup: "** + studentlist[index].group + **"\n\tStudent card: "**

+ studentlist[index].studcard + **"\n\tMarks: "** + studentlist[index].marks.join(**', '**) + **"\n"**);

**return** 0;

}

**function** averageMarks()

{

let marksum;

**for** (let i = 0; i < studentlist.length; i++)

{

marksum = studentlist[i].marks.reduce(**function**(sum, current) {**return** sum + current}, 0);

console.log(**"Average mark of student at index "** + i + **" is: "**, marksum / studentlist.length);

}

**return** 0;

}

**function** groupInfo(group)

{

**if** (studentlist.length === 0)

{

console.log(**"StudentList is empty!"**);

**return** -1;

}

let count = 1;

**for** (let i = 0; i < studentlist.length; i++)

{

**if** (studentlist[i].group === group)

{

console.log(**"Student "** + count + **" at group "** + group + **" is"**);

count++;

READ(i);

}

}

**return** 0;

}

**function** mostMarksAtGroup(group)

{

**if** (studentlist.length === 0)

{

console.log(**"StudentList is empty!"**);

**return** -1;

}

let marknum = -1;

let studind;

**for** (let i = 0; i < studentlist.length; i++)

{

**if** (studentlist[i].group === group)

{

**if** (marknum < studentlist[i].marks.length)

{

marknum = studentlist[i].marks.length;

studind = i;

}

}

}

console.log(**"Student that has most marks in group "** + group + **" is"**);

READ(studind);

**return** studind;

}

**function** noMarks\_student()

{

console.log(**"Student(s) that does not have marks"**);

**for** (let i = 0; i < studentlist.length; i++)

{

**if** (studentlist[i].marks.length === 0)

{

READ(i);

}

}

**return** 0;

}

module.exports = {

CREATE: **function**()

{

studentlist = [];

**return this**;

},

READ: **function**(index)

{

console.log(**"\n\tGroup: "** + studentlist[index].group + **"\n\tStudent card: "**

+ studentlist[index].studcard + **"\n\tMarks: "** + studentlist[index].marks.join(**', '**) + **"\n"**);

**return this**;

},

UPDATE: **function**(student)

{

**if** (studentlist.every(obj => obj.studcard !== student.studcard))

{

studentlist.push(student);

console.log(**"Added student to studentlist!"**);

}

**else**

{

console.log(**"Student with this student card already exists!"**);

}

**return this**;

},

DELETE: **function**(index)

{

console.log(**"This student was deleted:"**);

READ(index);

studentlist.splice(index, 1);

**return this**;

},

averageMarks: averageMarks,

groupInfo: groupInfo,

mostMarksAtGroup: mostMarksAtGroup,

noMarks\_student: noMarks\_student

}

Листинг 4. Тесты к заданию 2

**var** list = require(**"./task1\_2.js"**);

students = [

{

group: **'ICS12'**,

studcard: **'18u672'**,

marks: [2, 3, 3, 3]

},

{

group: **'ICS14'**,

studcard: **'19b344'**,

marks: []

},

{

group: **'ICS52'**,

studcard: **'20s222'**,

marks: [4, 4, 4, 5, 5, 3]

},

{

group: **'ICS21'**,

studcard: **'18u333'**,

marks: [2, 3, 4, 5]

},

{

group: **'ICS52'**,

studcard: **'18u477'**,

marks: [3, 3, 3, 3, 4]

},

{

group: **'ICS36'**,

studcard: **'7u332'**,

marks: [5, 5, 5, 5, 4, 4, 4]

},

{

group: **'ICS22'**,

studcard: **'19u238'**,

marks: [2, 2, 2, 5, 5, 5]

}

];

**var** result = list.CREATE().UPDATE(students[0]).UPDATE(students[1]).UPDATE(students[2]).UPDATE(students[3])

.UPDATE(students[4]).UPDATE(students[5]).UPDATE(students[6]).DELETE(6);

result.averageMarks();

result.groupInfo(**'ICS52'**);

result.mostMarksAtGroup(**'ICS52'**);

result.noMarks\_student(**''**);

## **Задание 3**

Создать хранилище в оперативной памяти для хранения точек.

Необходимо хранить информацию о точке: имя точки, позиция X и позиция Y.

Необходимо обеспечить уникальность имен точек.

Реализовать функции:

* CREATE READ UPDATE DELETE для точек в хранилище
* Получение двух точек, между которыми наибольшее расстояние
* Получение точек, находящихся от заданной точки на расстоянии, не превышающем заданную константу
* Получение точек, находящихся выше / ниже / правее / левее заданной оси координат
* Получение точек, входящих внутрь заданной прямоугольной зоны

Листинг 5. Задание 3

**var** dotList;

**function** READ(index)

{

console.log(**"\n\tName: "** + dotList[index].name + **"\n\tX: "**

+ dotList[index].x + **"\n\tY: "** + dotList[index].y + **"\n"**);

**return** 0;

}

**function** getDistance(dot1, dot2)

{

**return** Math.pow(dot1.x - dot2.x, 2) + Math.pow(dot1.y - dot2.y, 2);

}

**function** LargestDistance()

{

**if** (dotList.length < 2)

{

console.log(**"There are less than 2 dots in dotList"**);

**return** -1;

}

let largestlen = 0;

let ind1 = 0, ind2 = 1;

**for** (let i = 0; i < dotList.length; i++)

{

**for** (let j = i + 1; j < dotList.length; j++)

{

let curlen = getDistance(dotList[i], dotList[j]);

**if** (curlen > largestlen)

{

largestlen = curlen;

ind1 = i;

ind2 = j;

}

}

}

largestlen = Math.sqrt(largestlen);

console.log(**'Largest len is between these dots and equals '**, largestlen);

READ(ind1);

READ(ind2);

**return** largestlen;

}

**function** dots\_inrange(dot, rad)

{

console.log(**"Dot(s) located from a given point at a distance not exceeding a given constant"**);

**for** (let i = 0; i < dotList.length; i++)

{

**if** (getDistance(dotList[i], dot) <= rad)

READ(i);

}

**return** 0;

}

**function** fieldDots(dot, part)

{

**switch** (part) {

**case 'right'**:

console.log(**"Dot(s) that are right then a given dot"**);

**for** (let i = 0; i < dotList.length; i ++)

{

**if** (dotList[i].x > dot.x)

READ(i);

}

**break**;

**case 'left'**:

console.log(**"Dot(s) that are left then a given dot"**);

**for** (let i = 0; i < dotList.length; i ++)

{

**if** (dotList[i].x < dot.x)

READ(i);

}

**break**;

**case 'higher'**:

console.log(**"Dot(s) that are higher then a given dot"**);

**for** (let i = 0; i < dotList.length; i ++)

{

**if** (dotList[i].y > dot.y)

READ(i);

}

**break**;

**case 'below'**:

console.log(**"Dot(s) that are below then a given dot"**);

**for** (let i = 0; i < dotList.length; i ++)

{

**if** (dotList[i].y < dot.y)

READ(i);

}

**break**;

}

}

**function** Areadots(dot1, dot2)

{

console.log(**"Dot(s) that are situated in a given rectangle"**);

**for** (let i = 0; i < dotList.length; i++)

{

**if** (Math.min(dot1.x, dot2.x) < dotList[i].x < Math.max(dot1.x, dot2.x)

&& Math.min(dot1.y, dot2.y) < dotList[i].y < Math.max(dot1.y, dot2.y))

{

READ(i);

}

}

**return** 0;

}

module.exports = {

CREATE: **function**()

{

dotList = [];

**return this**;

},

READ: **function**(index)

{

console.log(**"\n\tName: "** + dotList[index].name + **"\n\tX: "**

+ dotList[index].x + **"\n\tY: "** + dotList[index].y + **"\n"**);

**return this**;

},

UPDATE: **function**(dot)

{

**if** (dotList.every(obj => obj.name !== dot.name))

{

dotList.push(dot);

console.log(**"Added dot to dotList!"**);

}

**else**

{

console.log(**"Dot with this name already exists!"**);

}

**return this**;

},

DELETE: **function**(index)

{

console.log(**"This dot was deleted:"**);

READ(index);

dotList.splice(index, 1);

**return this**;

},

LargestDistance: LargestDistance,

dots\_inrange: dots\_inrange,

fieldDots: fieldDots,

Areadots: Areadots

}

Листинг 6. Тесты к заданию 3

**var** list = require(**"./task1\_3.js"**);

dots = [

{

name: **'1'**,

x: 5,

y: 7

},

{

name: **'2'**,

x: 6,

y: 9

},

{

name: **'3'**,

x: 2,

y: 2

},

{

name: **'4'**,

x: -3,

y: -4

},

{

name: **'5'**,

x: -6,

y: -7

},

{

name: **'6'**,

x: 7,

y: 7

},

{

name: **'7'**,

x: 1,

y: 0

}

];

**var** result = list.CREATE().UPDATE(dots[0]).UPDATE(dots[1]).UPDATE(dots[2]).UPDATE(dots[3])

.UPDATE(dots[4]).UPDATE(dots[5]).UPDATE(dots[6]).DELETE(6);

result.LargestDistance();

result.dots\_inrange(dots[6], 4);

result.fieldDots(dots[6], **'left'**);

result.Areadots(dots[6], dots[0]);

# Task 2

## **Задание 1**

Создать класс *Точка*.

Добавить классу точка *Точка* метод инициализации полей и метод вывода полей на экран

Создать класс *Отрезок*.

У класса *Отрезок* должны быть поля, являющиеся экземплярами класса *Точка*.

Добавить классу *Отрезок* метод инициализации полей, метод вывода информации о полях на экран, а так же метод получения длины отрезка.

Листинг 7. Задание 1

**class** Dot

{

constructor(x, y)

{

**this**.**x** = x;

**this**.**y** = y;

}

showDotFields()

{

***console***.log(**"Field X: "** + **this**.**x**);

***console***.log(**"Field Y: "** + **this**.**y**);

}

}

**class** Section

{

constructor(dot1, dot2)

{

**this**.**dot1** = dot1;

**this**.**dot2** = dot2;

}

showSectFields()

{

***console***.log(**"First dot fields are:"**)

**this**.**dot1**.showDotFields()

***console***.log(**"Second dot fields are:"**)

**this**.**dot2**.showDotFields()

}

**get** Length()

{

**return *Math***.sqrt((**this**.**dot1**.**x** - **this**.**dot2**.**x**) \*\* 2 + (**this**.**dot1**.**y** - **this**.**dot2**.**y**) \*\* 2)

}

}

***module***.**exports** = {

Dot:Dot,

Section:Section

}

Листинг 8. Тесты к заданию 1

**let linemodule** = ***require***(**"./task2\_1.js"**);

**let** dot = **new linemodule**.Dot(3, 3);

dot.showDotFields();

**let** dot2 = **new linemodule**.Dot(3, 4);

dot2.showDotFields();

**let** sect = **new linemodule**.Section(dot, dot2);

sect.showSectFields();

***console***.log(sect.Length);

## **Задание 2**

Создать класс *Треугольник*.

Класс *Треугольник* должен иметь поля, хранящие длины сторон треугольника.

Реализовать следующие методы:

* Метод инициализации полей
* Метод проверки возможности существования треугольника с такими сторонами
* Метод получения периметра треугольника
* Метод получения площади треугольника
* Метод для проверки факта: является ли треугольник прямоугольным

Листинг 9. Задание 2

**let** linemodule = require(**"./task2\_1.js"**);

**class** Triangle

{

constructor(line1, line2, line3)

{

**this**.line1 = line1;

**this**.line2 = line2;

**this**.line3 = line3;

}

isValid()

{

**let** a = **this**.**line1**.Length;

**let** b = **this**.**line2**.Length;

**let** c = **this**.**line3**.Length;

**if** (a + b > c && a + c > b && b + c > a)

{

***console***.log(**"Triangle exists!"**);

**return true**;

}

**else**

***console***.log(**"Triangle not exists!"**);

**return false**;

}

**get** getPerimeter()

{

**return this**.**line1**.Length + **this**.**line2**.Length + **this**.**line3**.Length;

}

**get** getSquare()

{

**let** p = **this**.getPerimeter;

**let** a = **this**.line1.Length;

**let** b = **this**.line2.Length;

**let** c = **this**.line3.Length;

**return** Math.sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

\_\_checkRight(cat1, cat2, hyp)

{

**let** eps = 1e-10;

**return** (cat1 \*\* 2 + cat2 \*\* 2 - hyp \*\* 2) <= eps;

}

isRight()

{

**let** a = **this**.line1.Length;

**let** b = **this**.line2.Length;

**let** c = **this**.line3.Length;

**if** (**this**.\_\_checkRight(a, b, c) || **this**.\_\_checkRight(a, c, b) || **this**.\_\_checkRight(b, c, a))

{

console.log(**"Triangle is right!"**);

**return true**;

}

**else**

{

console.log(**"Triangle is not right!"**);

**return true**;

}

}

}

module.exports = {

Triangle:Triangle

}

Листинг 10. Тесты к заданию 2

**let linemodule** = require(**"./task2\_1.js"**);

**let** triangmodule = require(**"./task2\_2.js"**);

**let** dot = **new** linemodule.Dot(3, 3);

dot.showDotFields();

**let** dot2 = **new** linemodule.Dot(3, 4);

dot2.showDotFields();

**let** dot3 = **new** linemodule.Dot(5, 3);

dot3.showDotFields();

**let** line1 = **new** linemodule.Section(dot, dot2);

**let** line2 = **new** linemodule.Section(dot2, dot3);

**let** line3 = **new** linemodule.Section(dot, dot3);

**let** triang = **new** triangmodule.Triangle(line1, line2, line3);

triang.isValid();

triang.isRight();

console.log(triang.getPerimeter);

console.log(triang.getSquare);

## **Задание 3**

Реализовать программу, в которой происходят следующие действия:

Происходит вывод целых чисел от 1 до 10 с задержками в 2 секунды.

После этого происходит вывод от 11 до 20 с задержками в 1 секунду.

Потом опять происходит вывод чисел от 1 до 10 с задержками в 2 секунды.

После этого происходит вывод от 11 до 20 с задержками в 1 секунду.

Это должно происходить циклически.

Листинг11. Задание 3

**"use strict"**;

**let *seconds*** = 0;

**let *flag*** = 0;

**function** *intervaloid*(delay, flag)

{

**let** interval1 = *setInterval*(() => {

**if** (***seconds*** >= 0)

{

***seconds***++;

**let** message;

**if** (flag === 1)

message = **"Seconds: "** + (***seconds*** + 10);

**else**

message = **"Seconds: "** + ***seconds***;

***console***.log(message);

}

**if**(***seconds*** === 10)

{

*clearInterval*(interval1);

***seconds*** = 0;

**if** (flag === 0)

*intervaloid*(1000, 1);

**else**

*intervaloid*(2000, 0);

}

}, delay);

}

**let *delay*** = 2000;

*intervaloid*(***delay***, ***flag***);