|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 1 |

**Дисциплина:** Архитектура ЭВМ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-52Б |  |  | В.А. Иванов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  |  |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

Задание 1.1 Создать хранилище в оперативной памяти для хранения информации о детях.

**Условие**

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

**Программная реализация**

"use strict";

// Хранилище информации о детях

let data = [];

function print\_kid(kid)

{

    console.log("> " + kid.surname + " : " + kid.age);

}

function is\_surname\_in(surname\_)

{

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

    {

        if (data[i].surname == surname\_)

                return true;

    }

    return false;

}

function find\_index(surname\_)

{

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

    {

        if (data[i].surname == surname\_)

                return i;

    }

    return -1;

}

function average\_age()  // Вычисление среднего возраста

{

    let sum = 0;

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

        sum += data[i].age;

    return parseInt(sum/data.length);

}

function oldest\_kid()   // Поиск самого старшего ребёнка

{

    if (data.length == 0) return null;

    let max\_i = 0;

    for (let i=1; i < data.length; i++)

    {

        if (data[i].age > data[max\_i].age)

            max\_i = i;

    }

    return data[max\_i];

}

function in\_adge\_range(begin, end)  // Дети с возрастом в заданном отрезке

{

    let count = 0;

    console.log("Дети с возрастом от " + begin + " до " + end + " лет:");

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

    {

        if (begin <= data[i].age && data[i].age <= end)

        {

            print\_kid(data[i]);

            count++;

        }

    }

    if (!count)

        console.log("Дети с заданным возрастом не найдены");

}

function surname\_at(char)   // Дети с фамилией на заданную букву

{

    let count = 0;

    console.log("Дети с фамилией на букву " + char + ":");

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

    {

        if (data[i].surname[0] === char)

        {

            print\_kid(data[i]);

            count++;

        }

    }

    if (!count)

        console.log("Дети с заданным параметром не найдены");

}

function surname\_longer\_len(len)    // Дети с фамилией длинне, чем заданного кол-ва символов

{

    let count = 0;

    console.log("Дети с фамилией длиннее " + len + " символов:");

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

    {

        if (data[i].surname.length > len)

        {

            print\_kid(data[i]);

            count++;

        }

    }

    if (!count)

        console.log("Дети с заданным параметром не найдены");

}

function surname\_vowel()    // Дети с фамилией на гласную букву

{

    let count = 0;

    let vowel = "ауоыиэяюёе";

    console.log("Дети с фамилией на гласную букву" + ":");

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

    {

        if (vowel.search(data[i].surname[0].toLowerCase()) != -1)

        {

            print\_kid(data[i]);

            count++;

        }

    }

    if (!count)

        console.log("Дети с заданным параметром не найдены");

}

/// CDIO data

function CREATE(surname\_, age\_)

{

    if (!is\_surname\_in(surname\_))

    {

        let new\_child =

        {   surname : surname\_,

            age : age\_  };

        data.push(new\_child);

    }

    else

    {

        console.log("Запись не добавлена: Фамилия \"" + surname\_ + "\" уже в хранилище.");

    }

}

function DELETE(surname\_)

{

    let i = find\_index(surname\_);

    if (i != -1)

    {

        data.splice(i, 1);

    }

    else

    {

        console.log("Запись не удалена: Фамилия \"" + surname\_ + "\" не в хранилище.");

    }

}

function READ()

{

    if (!data.length) {

        console.log("Хранилище не содержит записей.");

        return;

    }

    console.log("\nСписок всех детей (> Фамилия : Возраст):");

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

    {

        print\_kid(data[i]);

    }

}

function UPDATE(pre\_surname, surname\_, age\_)

{

    let i = find\_index(pre\_surname);

    if (i != -1)

    {

        data[i].surname = surname\_;

        data[i].age = age\_;

    }

    else

    {

        console.log("Запись не обновлена: Фамилия \"" + pre\_surname + "\" не в хранилище.");

    }

}

function CLEAR()

{

    data = [];

}

**Тесты**

Тест №1:

function test\_1()

{

    console.log("\nСоздание записей");

    CREATE("Иванов", 20);

    CREATE("Помоев", 23);

    CREATE("Иванов", 22);

    CREATE("Сучёчков", 4);

    CREATE("Песков", 2);

    READ();

    console.log("\nУдаление записей");

    DELETE("Помоев");

    DELETE("Брянская");

    READ();

    console.log("\nОбновление записей");

    UPDATE("Брянская", "Иванов", 20);

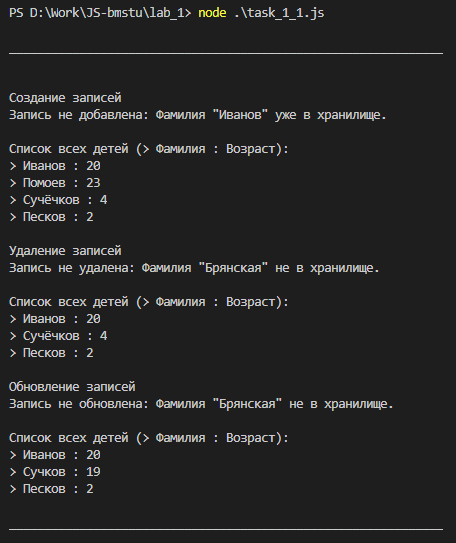
    UPDATE("Сучёчков", "Сучков", 19);

    READ();

    CLEAR();

}

Результат:



Тест №2:

function test\_2()

{

    CLEAR();

    CREATE("Иванов", 21);

    CREATE("Помоев", 21);

    CREATE("Буратинов", 20);

    CREATE("Сталин", 7);

    CREATE("Сучёчков", 3);

    CREATE("Усков", 2);

    READ();

    console.log("Средний возраст детей: " + average\_age());

    console.log("\nСтарший ребёнок:");

    print\_kid(oldest\_kid());

    in\_adge\_range(3, 20);

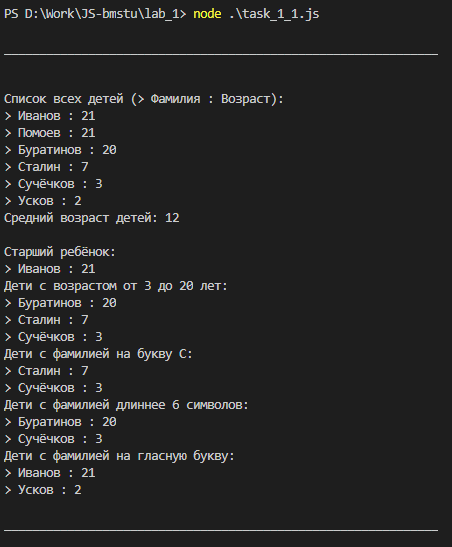
    surname\_at("С");

    surname\_longer\_len(6);

    surname\_vowel();

}

Результат:



Задание 1.2 Создать хранилище в оперативной памяти для хранения информации о студентах.

**Условие**

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

**Программная реализация**

"use strict";

// Хранилище информации о студентах

let data = [];

let student\_t =

{   group : null,

    id :    null,

    marks : null };

function print\_student(st)

{

    if (st.marks.length)

        console.log("> №"+ st.id + ", " + st.group + " : " + st.marks);

    else

        console.log("> №"+ st.id + ", " + st.group + " : без оценок");

}

function is\_id\_in(id\_)

{

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

    {

        if (data[i].id == id\_)

            return true;

    }

    return false;

}

function find\_index(id\_)

{

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

    {

        if (data[i].id == id\_)

                return i;

    }

    return -1;

}

function average\_mark(id\_)  // Средняя оценка указанного студента

{

    let pos = find\_index(id\_);

    if (pos == -1)

    {

        console.log("Запись не найдена: № билета " + id\_ + " нет в хранилище.");

        return;

    }

    let st = data[pos];

    if (st.marks.length != 0)

    {

        let sum = 0;

        for (let i=0; i < st.marks.length; i++)

            sum += st.marks[i];

        sum /= st.marks.length;

        console.log("Средняя оценка студента №" + st.id + " равна " + sum);

    }

    else

    {

        console.log("Студент №" + st.id + " не имеет оценок");

    }

}

function print\_group(group\_)    // Вывод студентов данной группы

{

    let count = 0;

    console.log("Информация о студентах группы " + group\_ + ":");

    for (let i=0; i<data.length; i++)

    {

        if (data[i].group == group\_)

        {

            print\_student(data[i]);

            count++;

        }

    }

    if (!count)

    {

        console.log("Студенты заданной группы не найдены");

    }

}

function most\_marks\_group(group\_)   // Получение студента с наибольшим количеством оценок в данной группе

{

    let st = null;

    console.log("Информация о студентах группы " + group\_ + ":");

    for (let i=0; i<data.length; i++)

    {

        if (data[i].group == group\_)

        {

            if (!st || st.marks.length < data[i].marks.length)

                st = data[i]

        }

    }

    if (st)

    {

        console.log("Студент группы " + group\_ + " с наибольшим количеством оценок:");

        print\_student(st);

    }

    else

    {

        console.log("Студенты заданной группы не найдены");

    }

}

function no\_marks()     // Получение студентов без оценок

{

    let count = 0;

    console.log("Информация о студентах без оценок:");

    for (let i=0; i<data.length; i++)

    {

        if (!data[i].marks.length)

        {

            print\_student(data[i]);

            count++;

        }

    }

    if (!count)

    {

        console.log("Студенты без оценок не найдены");

    }

}

/// CDIO data

function CREATE(group\_, id\_, marks\_=[])

{

    if (!is\_id\_in(id\_))

    {

        let new\_st =

        {   group : group\_,

            id :    id\_,

            marks : marks\_ };

        data.push(new\_st);

    }

    else

    {

        console.log("Запись не добавлена: № билета " + id\_ + " уже в хранилище.");

    }

}

function DELETE(id\_)

{

    let i = find\_index(id\_);

    if (i != -1)

    {

        data.splice(i, 1);

    }

    else

    {

        console.log("Запись не удалена: № билета " + id\_ + " не в хранилище.");

    }

}

function READ()

{

    if (!data.length) {

        console.log("Хранилище не содержит записей.");

        return;

    }

    console.log("\nСписок всех студентов (> №билета, группа : оценки):");

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

    {

        print\_student(data[i]);

    }

}

function UPDATE(pre\_id, group\_, id\_, marks\_)

{

    let i = find\_index(pre\_id);

    if (i != -1)

    {

        data[i].group = group\_;

        data[i].id = id\_;

        data[i].marks = marks\_;

    }

    else

    {

        console.log("Запись не обновлена: № билета " + pre\_id + " не в хранилище.");

    }

}

function CLEAR()

{

    data = [];

}

**Тесты:**

Тест №1:

function test\_1()

{

    CLEAR();

    console.log("\nСоздание записей");

    CREATE("IU7-21", 1, [4]);

    CREATE("IU7-11", 0, []);

    CREATE("SM21-41", 151, [5, 2, 3]);

    CREATE("BMT3-61", 151, [5, 2, 3]);

    CREATE("IU7-45", 110, [2, 2, 2]);

    CREATE("IU7-45", 210, [5, 5, 5]);

    READ();

    console.log("\Удаление записей");

    DELETE(51);

    DELETE(151);

    READ();

    console.log("\Обновление записей");

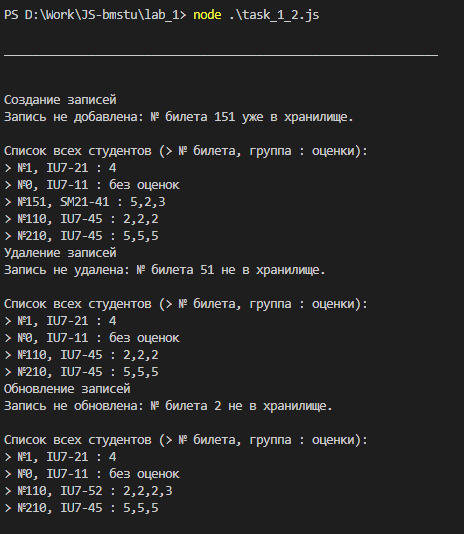
    UPDATE(2, "Nope-21", 2, []);

    UPDATE(110, "IU7-52", 110, [2, 2, 2, 3]);

    READ();

}

Результат:



Тест №2:

function test\_2()

{

    CLEAR();

    CREATE("IU7-21", 1, [4]);

    CREATE("IU7-11", 0, []);

    CREATE("SM21-41", 151, [5, 2, 3]);

    CREATE("IU7-45", 110, [2, 2, 2, 3]);

    CREATE("IU7-45", 210, [5, 5, 5]);

    CREATE("IU7-45", 220);

    READ();

    console.log("\nСредние оценки:\n");

    average\_mark(1);

    average\_mark(0);

    average\_mark(100);

    average\_mark(151);

    console.log("\nПоиск по группе:");

    print\_group("IU7-45");

    print\_group("IU7-42");

    print\_group("SM21-41");

    console.log("\nСуденты по группы с наибольшим числом оценок:");

    most\_marks\_group("IU7-45");

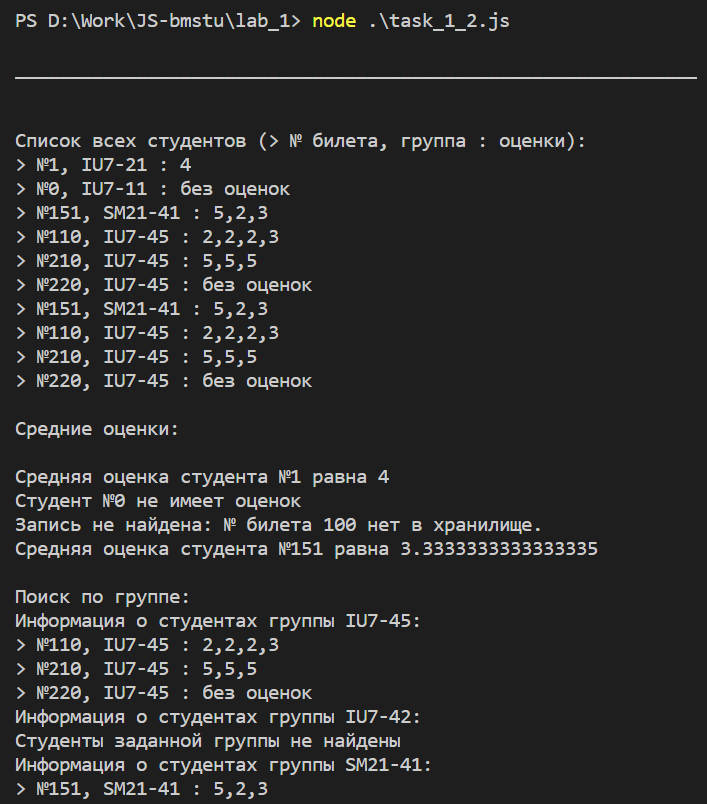
    most\_marks\_group("IU7-42");

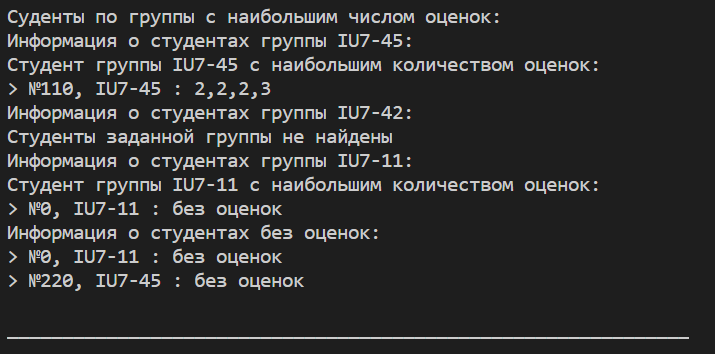
    most\_marks\_group("IU7-11");

    no\_marks();

}

Результат:





Задание 1.3 Создать хранилище в оперативной памяти для хранения точек.

**Условие**

Неоходимо 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

**Программная реализация**

"use strict";

let EPS = 1e-5;

// Хранилище информации о точек

let data = [];

let point\_t =

{   x: 0,

    y: 0 };

function new\_point(x\_, y\_)

{

    return { x: x\_,     y: y\_ };

}

function point\_str(p)

{

    return ("(" + p.x + ", " + p.y + ")");

}

function print\_point(p)

{

    console.log("> " + point\_str(p)+ ";");

}

function find\_index(x\_, y\_)

{

    for (let i=0; i<data.length; i++)

    {

        if (Math.abs(data[i].x - x\_) < EPS &&

            Math.abs(data[i].y - y\_) < EPS)

            return i;

    }

    return -1;

}

function find\_distance(p1, p2)

{

    return Math.sqrt((p1.x - p2.x)\*\*2 + (p1.y - p2.y)\*\*2);

}

function max\_distance()

{

    if (data.length < 2)

    {

        console.log("Недостаточно точек для операции");

        return;

    }

    let p1=0, p2=1;

    let max\_d = find\_distance(data[p1], data[p2]);

    for (let i=0; i<data.length; i++)

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

        {

            let d = find\_distance(data[i], data[j]);

            if (d > max\_d)

            {

                max\_d = d;

                p1 = i;

                p2 = j;

            }

        }

    console.log("Максимальное расстояние: от " + point\_str(data[p1]) +

    " до " + point\_str(data[p2]) + " равно " + max\_d);

}

function lower\_ox()     // Вывод точек ниже оси ОХ

{

    function cmp(p) {

        return p.y < 0;

    }

    print\_by\_key(cmp, "Ниже оси OX");

}

function higher\_ox()    // Вывод точек выше оси ОХ

{

    function cmp(p) {

        return p.y > 0;

    }

    print\_by\_key(cmp, "Выше оси OX");

}

function to\_left\_oy()   // Вывод точек левее оси ОY

{

    function cmp(p) {

        return p.x < 0;

    }

    print\_by\_key(cmp, "Левее оси OY");

}

function to\_right\_oy()  // Вывод точек правее оси ОY

{

    function cmp(p) {

        return p.x > 0;

    }

    print\_by\_key(cmp, "Правее оси OY");

}

function closer\_then(x\_, y\_, dist)      // Вывод точек удалённых от (x,y) меньше, чем на dist

{

    if (dist < 0) {

        console.log("Ошибка: отрицательное расстояние");

        return;

    }

    let p0 = new\_point(x\_, y\_);

    function cmp(p) {

        return (find\_distance(p, p0) < dist)

    }

    print\_by\_key(cmp, "Расстояние от ("+x\_+", "+y\_+") не превышает " + dist);

}

function inside\_zone(x\_min, x\_max, y\_min, y\_max)    // Вывод точек внутри зоны от (x\_min, y\_min) до (x\_max, y\_max)

{

    if (x\_min > x\_max || y\_min > y\_max) {

        console.log("Некорректные параметры зоны");

    }

    function cmp(p){

        return (x\_min<=p.x && x\_max>=p.x) && (y\_min<=p.y && y\_max>=p.y);

    }

    print\_by\_key(cmp, "Внутри зоны от ("+x\_min+", "+y\_min+") до ("+x\_max+", "+y\_max+")");

}

function print\_by\_key(func, f\_name="Признак не указан")     // Функция, выводящая точки по функции-компаратору

{

    let count = 0;

    console.log("\nТочки по признаку: " + f\_name +  ":");

    for (let i=0; i<data.length; i++)

    {

        let p = data[i];

        if (func(p))

        {

            print\_point(p);

            count++;

        }

    }

    if (!count)

    {

        console.log("Точки не найдены");

    }

}

// CDIO data

function CREATE(x\_, y\_)

{

    let new\_p =

    {   x: x\_, y: y\_};

    data.push(new\_p);

}

function DELETE(x\_, y\_)

{

    let i = find\_index(x\_, y\_);

    if (i != -1)

        data.splice(i, 1);

    else

        console.log("Точка (" + x\_ + ", " + y\_ + ") не найдена");

}

function READ()

{

    if (!data.length) {

        console.log("Хранилище не содержит записей.");

        return;

    }

    console.log("\nСписок всех точек:");

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

    {

        print\_point(data[i]);

    }

}

function UPDATE(old\_x, old\_y, x\_, y\_)

{

    let i = find\_index(old\_x, old\_y);

    if (i != -1)

    {

        data[i].x = x\_;

        data[i].y = y\_;

    }

    else

        console.log("Точка (" + old\_x + ", " + old\_y + ") не найдена");

}

function CLEAR()

{

    data = [];

}

**Тесты**

Тест №1:

function test\_1()

{

    console.log("\nСоздание записей");

    CREATE(1, 0);

    CREATE(1, 0);

    CREATE(2, 0);

    CREATE(2, -3.14);

    CREATE(4, 0);

    READ();

    console.log("\nУдаление записей");

    DELETE(1, 0);

    DELETE(2, 0);

    DELETE(3, 0);

    READ();

    console.log("\Обновление записей");

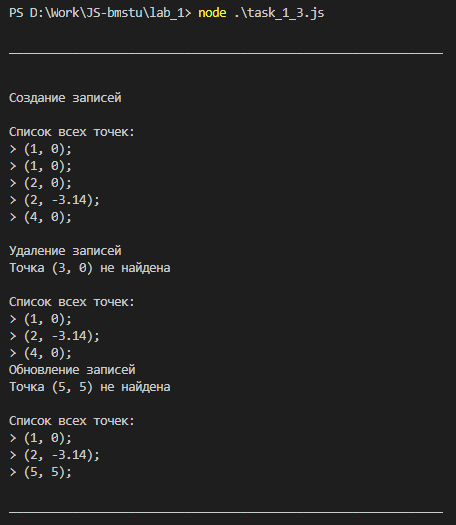
    UPDATE(5, 5, 10, 10);

    UPDATE(4, 0, 5, 5);

    READ();

}

Результат:



Тест №2:

function test\_2()

{

    CREATE(1, 0);

    CREATE(1, 0);

    CREATE(2, 0);

    CREATE(2, -3.14);

    CREATE(4, 0);

    CREATE(-3, 5);

    CREATE(0, 0);

    READ();

    max\_distance();

    closer\_then(0, 0, 3);

    inside\_zone(-1, 1, 0, 2);

    lower\_ox();

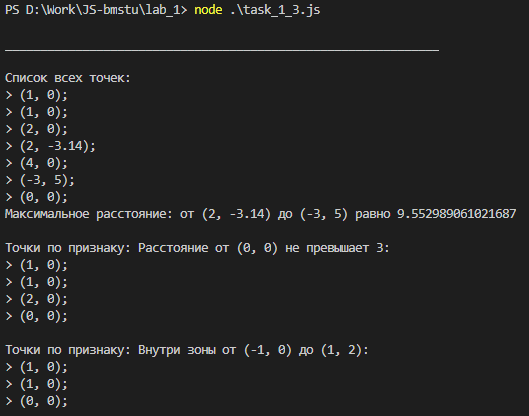
    higher\_ox();

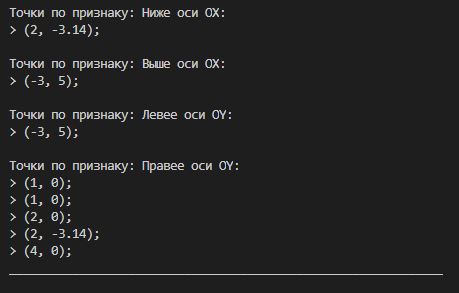
    to\_left\_oy();

    to\_right\_oy();

}

Результат:





Задание 2.1

**Условие**

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 также метод получения длины отрезка.

**Программная реализация**

class Point {

    constructor(x, y, z) {

        this.x = x;

        this.y = y;

        this.z = z;

    }

    printFields() {

        console.log(`(${this.x}, ${this.y}, ${this.z})`);

    }

}

class Segment {

    constructor(p1, p2) {

        this.p1 = p1;

        this.p2 = p2;

    }

    getLength() {

        return Math.sqrt(

            (this.p1.x - this.p2.x)\*\*2 +

            (this.p1.y - this.p2.y)\*\*2 +

            (this.p1.z - this.p2.z)\*\*2

        );

    }

    printFields() {

        console.log("Точки отрезка:");

        this.p1.printFields();

        this.p2.printFields();

    }

}

**Тесты**

Тест №1:

function main() {

    let p1 = new Point(0, 0, 0);

    let p2 = new Point(1, 0, 2);

    let p3 = new Point(-1.141, 2, 3.1);

    let p4 = new Point(0, 0, 0);

    console.log("Точки:");

    p1.printFields();

    p2.printFields();

    p3.printFields();

    p4.printFields();

    console.log("\nОтрезок №1:");

    let s = new Segment(p3, p2);

    s.printFields();

    console.log("Длина отрезка = " + s.getLength());

    console.log("\nОтрезок №2:");

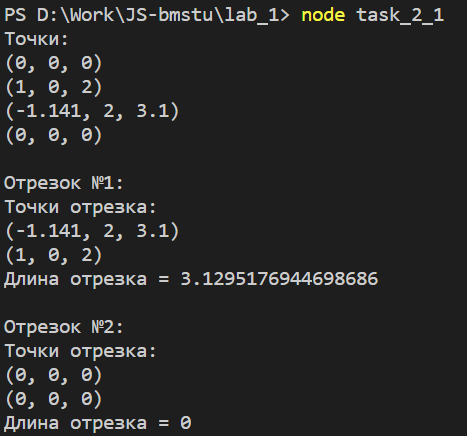
    s = new Segment(p1, p4);

    s.printFields();

    console.log("Длина отрезка = " + s.getLength());

}

Результат:



Задание 2.2

**Условие**

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

**Программная реализация**

function isEqual(a, b) {

    return Math.abs(a-b) < Number.EPSILON;

}

class Triangle {

    constructor(a, b, c) {

        this.a = a;

        this.b = b;

        this.c = c;

    }

    isExist() {    // Проверка возможности существования треугольника

        if (this.a>0 && this.b>0 && this.c>0) {

            let m = Math.max(this.a, this.b, this.c);

            return 2\*m < this.getPerimeter();

        }

        else {

            return false;

        }

    }

    isRight() {     // Проверка прямоугольности

        let hypotenuse\_2 = Math.max(this.a, this.b, this.c) \*\* 2;

        let cathetus\_sum2 = this.a\*\*2 + this.b\*\*2 + this.c\*\*2 - hypotenuse\_2;

        return isEqual(cathetus\_sum2, hypotenuse\_2);

    }

    getPerimeter() {    // Периметр

        return this.a + this.b + this.c;

    }

    getArea() {         // Площадь

        let p = this.getPerimeter() / 2;

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

    }

    toString() {

        return `triangle: ${this.a}, ${this.b}, ${this.c}`;

    }

}

**Тесты**

Тест №1:

function test1() {

    let tr = new Triangle(0, 1, 1);

    console.log("Is " + tr.toString() + " exist - " + tr.isExist());

    tr = new Triangle(-1, 1, 1);

    console.log("Is " + tr.toString() + " exist - " + tr.isExist());

    tr = new Triangle(3, 1, 1);

    console.log("Is " + tr.toString() + " exist - " + tr.isExist());

    tr = new Triangle(2, 3, 1);

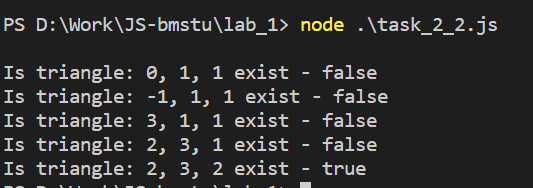
    console.log("Is " + tr.toString() + " exist - " + tr.isExist());

    tr = new Triangle(2, 3, 2);

    console.log("Is " + tr.toString() + " exist - " + tr.isExist());

}

Результат:



Тест №2:

function test2() {

    let tr = new Triangle(2, 3, 1.01);

    console.log(tr.toString());

    console.log("Is right - " + tr.isRight());

    console.log("P = " + tr.getPerimeter());

    console.log("S = " + tr.getArea());

    console.log();

    tr = new Triangle(3, 5, 4);

    console.log(tr.toString());

    console.log("Is right - " + tr.isRight());

    console.log("P = " + tr.getPerimeter());

    console.log("S = " + tr.getArea());

    console.log();

    tr = new Triangle(1, 1, 1);

    console.log(tr.toString());

    console.log("Is right - " + tr.isRight());

    console.log("P = " + tr.getPerimeter());

    console.log("S = " + tr.getArea());

    console.log();

    tr = new Triangle(8, 6, 10);

    console.log(tr.toString());

    console.log("Is right - " + tr.isRight());

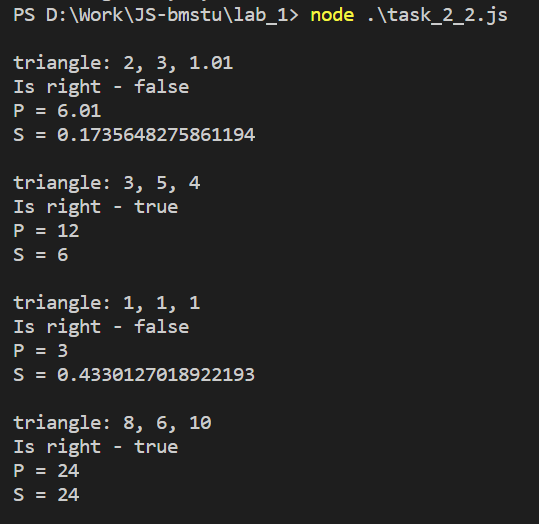
    console.log("P = " + tr.getPerimeter());

    console.log("S = " + tr.getArea());

    console.log();

}

Результат:



Задание 2.3

**Условие**

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.

**Программная реализация**

"use strict";

let interval;

let begin\_t;

let func\_n = 0;

let n = 1;

let func\_arr = [f1, f2, f1, f2, fstop];

let delay\_arr = [2000, 1000, 2000, 1000, 0];

function f1() {

    console.log(n, Date.now() - begin\_t);

    n++;

    if (n >= 11)

    {

        func\_n++;

        clearInterval(interval);

        interval = setInterval(func\_arr[func\_n], delay\_arr[func\_n]);

    }

}

function f2() {

    console.log(n, Date.now() - begin\_t);

    n++;

    if (n >= 21)

    {

        n = 1;

        func\_n++;

        clearInterval(interval);

        interval = setInterval(func\_arr[func\_n], delay\_arr[func\_n]);

    }

}

function fstop() {

    console.log("End");

    clearInterval(interval);

}

function main() {

    console.log("Вывод: число, время с начала работы (мс)");

    begin\_t = Date.now();

    interval = setInterval(func\_arr[func\_n], delay\_arr[func\_n]);

}

main();

**Тесты**

Результат (для отражения задержек во времени в отчёте, выводится время, прошедшее с запуска функции в мс):

