МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «ООП»

Тема: Создание игрового поля

Студент гр. 9304 _____ Попов Д.С

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Научиться создавать классы на языке программирования C++

Задание.

Написать класс игрового поля, которое представляет из себя прямоугольник (двумерный массив). Для каждого элемента поля должен быть создан класс клетки. Клетка должна отображать, является ли она проходимой, а также информацию о том, что на ней находится. Также, на поле должны быть две особые клетки: вход и выход.

При реализации поля запрещено использовать контейнеры из stl

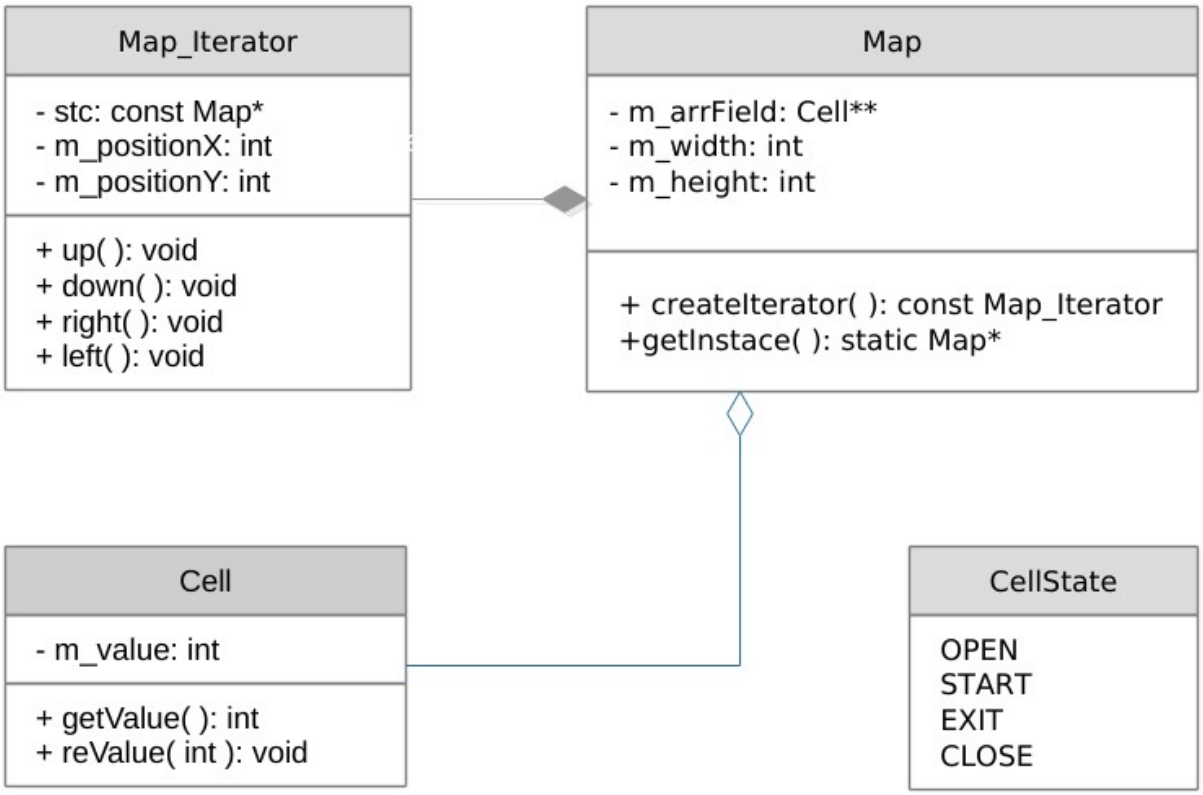
Обязательные требования:

- Реализован класс поля
- Реализован класс клетки
- Для класса поля написаны конструкторы копирования и перемещения, а также операторы присваивания и перемещения
- Поле сохраняет инвариант - из любой клетки можно провести путь до любой другой
- Гарантированно отсутствует утечки памяти

Дополнительные требования:

- Поле создается с использованием паттерна Синглтон
- Для обхода по полю используется паттерн Итератор

Выполнение работы.



Перечисление *CellState* отвечает за состояние клетки. Существует четыре состояния: клетка пуста — *OPEN*, клетка не проходима — *CLOSE*, клетка входа — *START*, клетка выхода — *EXIT*.

Класс *Cell* является сруктурной единицей поля. Клетка имеет тип(состояние) — *m_value*. Данное поле является приватным, поэтому для взаимодействия с ним созданы публичные методы *getValue* и reValue.

Класс *Map* реализован при помощи шаблона Синглтон. Поля m_*width* и m_*height* хранят размеры поля. Поле состоит из клеток. Для этого был создан двумерный динамический массив *m_arrField*. Обращение происходит по двойному индексу: по высоте и длине. Метод g*etInstance* создаёт единственный экземпляр класса *Map* с заданными размерами и возвращает указатель на него. При повторном вызове метода возвращает указатель на уже

созданный экземпляр. Также были реализованны приватные конструкторы копирования и перемещения.

Для обхода по полю был создан класс *Map_Iterator* с использованием шаблона Итератор. В приватных полях *m_positionX* и *m_positionY* хранятся индексы по высоте и длине текущего элемента. Методы *up u down* изменяют значение *m_positionX* на единицу, тем самым имитируя передвижение по оси X. Методы *left* и *right* изменяют значение *m_positionY* на единицу, тем самым имитируя передвижение по оси Y. Попытка переместится за пределы поля не приведет к изменению индексов. Метод *result* выводит поле в консоль и отображает позицию итератора.

Для проверки работоспособности классов был создан *UnitTest*. Его запуск и завершение без ошибок показывает проверку классов.

Разработанный программный код см. в приложении А.

Выводы.

Научились создавать классы на языке программирования C++.

Были реализованы классы *Cell*, *Map*, *Map_Iterator*, а также перечисление *CellState*. Класс *Map* был создан с использованием шаблона Синглтон. Класс *Map_Iterator* был создан с использованием шаблона Итератор. Для проверки реализованных классов был создан *UnitTest*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Cell.cpp

```
#include "Cell.h"

Cell::Cell():m_value(CellTypes::OPEN){}
Cell::Cell(int a):m_value(a){}

int Cell::getValue(){
    return m_value;
}

void Cell::reValue(int newValue){
    m_value = newValue;
}
```

Название файла: Cell.h

```
#pragma once

namespace CellTypes{
    enum CellState{
        OPEN, START, EXIT, CLOSE
    };
}

class Cell{
    int m_value;        //Значение клетки
public:
    Cell();
    Cell(int);
    int getValue();
    void reValue(int newValue);
};
```

Название файла: Map.cpp

```
#include "Map.h"
#include "Cell.h"
#include "Map_Iterator.h"


Map* Map::m_ptrMap = nullptr;

Map* Map::getInstance(){
    if(m_ptrMap == nullptr){
        m_ptrMap = new Map();
    }
    return m_ptrMap;
}

Map* Map::getInstance(int sizeX, int sizeY){
    if(m_ptrMap == nullptr){
        m_ptrMap = new Map(sizeX, sizeY);
    }
    return m_ptrMap;
}

Map::Map():m_width(5), m_height(5){     //Карта будет статичной?
    m_arrField = new Cell*[m_width];
    for(int i = 0; i < m_width; i++){
        m_arrField[i] = new Cell[m_height];
    }
    m_arrField[0][0].reValue(CellTypes::START);
                    m_arrField[m_width     -     1][m_height     -
1].reValue(CellTypes::EXIT);
}

Map::Map(int sizeX, int sizeY):m_width(sizeX), m_height(sizeY){
    m_arrField = new Cell*[m_width];
    for(int i = 0; i < m_width; i++){
```

```
            m_arrField[i] = new Cell[m_height];
        }
        m_arrField[0][0].reValue(CellTypes::START);
                        m_arrField[m_width     -     1][m_height     -
1].reValue(CellTypes::EXIT);
    }

    Map::~Map(){
        for(int i = 0; i < m_width; i++){
            delete[] m_arrField[i];
        }
        delete[] m_arrField;
    }

    Map_Iterator* Map::createIterator() const{
        return new Map_Iterator(this);
    }
```

Название файла: Map.h

```
    #pragma once

    class Map_Iterator;
    class Cell;

    class Map{
        static Map* m_ptrMap;
        Cell** m_arrField;
        unsigned int m_width;
        unsigned int m_height;
        Map();
        Map(int, int);
        Map(Map& other) = delete;
         Map(Map&& other) = delete;
        Map& operator = (const Map&) = delete;
         Map&& operator = (const Map&&) = delete;
    public:
```

```
        friend class Map_Iterator;
        Map_Iterator* createIterator() const;
        static Map* getInstance();
        static Map* getInstance(int, int);
        ~Map();
    };
```

Название файла: Map_Iterator.cpp

```
    #include "Map_Iterator.h"
    #include "Cell.h"
    #include "Map.h"
    #include <iostream>

    class Map_Iterator;

    Map_Iterator::Map_Iterator(const Map *s){
        stc = s;
        for(int i = 0; i < s->m_width; i++){
            for(int j = 0; j < s->m_height; j++){
                                if(s->m_arrField[i][j].getValue()  ==
CellTypes::START){
                    m_positionX = i;
                    m_positionY = j;
                }
            }
        }
    }

    //Map_Iterator::~Map_Iterator(){}

    void Map_Iterator::up(){
        if(m_positionX > 0){
                            if(stc->m_arrField[m_positionX   -   1]
[m_positionY].getValue() != CellTypes::CLOSE){
```

```
                m_positionX--;
            }
        }
    }

    void Map_Iterator::down(){
        if((m_positionX + 1) < stc->m_width){
                            if(stc->m_arrField[m_positionX   +   1]
[m_positionY].getValue() != CellTypes::CLOSE){
                m_positionX++;
            }
        }
    }

    void Map_Iterator::right(){
        if((m_positionY + 1) < stc->m_height){
                        if(stc->m_arrField[m_positionX][m_positionY  +
1].getValue() != CellTypes::CLOSE){
                m_positionY++;
            }
        }
    }

    void Map_Iterator::left(){
        if(m_positionY > 0){
                        if(stc->m_arrField[m_positionX][m_positionY  -
1].getValue() != CellTypes::CLOSE){
                m_positionY--;
            }
        }
    }

    int Map_Iterator::getValue(){
        return stc->m_arrField[m_positionX][m_positionY].getValue();
    }

    void Map_Iterator::result(){
```

```
        for(int i = 0; i < stc->m_width; i++){
            for(int j = 0; j < stc->m_height; j++){
                if((i == m_positionX) && (j == m_positionY)){
                    std::cout << "+ ";
                }else{
                     std::cout << stc->m_arrField[i][j].getValue() <<
" ";
                }
            }
            std::cout << "\n";
        }
    }
```

Название файла: Map_Iterator.h

```
#pragma once

class Map;

class Map_Iterator{
    const Map *stc;
    int m_positionX;
    int m_positionY;
public:
    Map_Iterator(const Map*);
    //~Map_Iterator();
    void up();
    void down();
    void left();
    void right();
    int getValue();
    void result();
};
```

Назваие файла: UnitTest.cpp

```
#include <cassert>
```

```
#include "UnitTest.h"

void UnitTest::Assert(){
    assert(0);
}

void UnitTest::AssertEqual(int a, int b){
    assert(a == b);
}

void UnitTest::AssertNotEqual(int a, int b){
    assert(a != b);
}

void UnitTest::AssertGreaterEqual(int a, int b){
    assert(a >= b);
}

void UnitTest::AssertLessEqual(int a, int b){
    assert(a <= b);
}

void UnitTest::AssertGreater(int a, int b){
    assert(a > b);
}
void UnitTest::AssertLess(int a, int b){
    assert(a < b);
}
```

Название файла: UnitTest.h

```
#pragma once

class UnitTest{
public:
    static void Assert();
```

```
    static void AssertEqual(int a, int b);
    static void AssertNotEqual(int a, int b);
    static void AssertGreaterEqual(int a, int b);
    static void AssertLessEqual(int a, int b);
    static void AssertGreater(int a, int b);
    static void AssertLess(int a, int b);
};
```

Название файла: UnitTest1.cpp

```
#include <iostream>
#include "../ClasesForProject/Map.h"
#include "../ClasesForProject/Cell.h"
#include "../ClasesForProject/Map_Iterator.h"
#include "UnitTest.h"

using namespace std;
using namespace CellTypes;


int main(){

    cout << "Проверка работоспособности Cell..." << "\n";
    Cell Cell1;
    UnitTest::AssertEqual(Cell1.getValue(), OPEN);
    Cell1.reValue(CLOSE);
    UnitTest::AssertEqual(Cell1.getValue(), CLOSE);
    Cell Cell2 = Cell1;
    UnitTest::AssertEqual(Cell2.getValue(), CLOSE);
    cout << "Проверка пройдена!" << "\n";

    cout << "Проверка работоспособности Map..." << "\n";
    Map* map1 = Map::getInstance();
    Map* map2 = Map::getInstance();
    try{
        if(map1 != map2){
```

```
                throw -1;
            }
        }
        catch(int){
            UnitTest::Assert();
        }
        cout << "Проверка пройдена!" << "\n";
        /*
         cout << "Проверка работоспособности Map_Iterartor..." << "\
n";
        Map_Iterator* iter1 = map1->createIterator();
        */
       delete map1;
    }
```

Название файла: main.cpp

```
    #include "./ClasesForProject/Map.h"
    #include "./ClasesForProject/Map_Iterator.h"
    #include <iostream>
    #include <string>

    int main(){

                Map_Iterator*   iter   =   Map::getInstance(10,10)-
>createIterator(); //10-10 = размеры поля
        std::string route;

        //Проверка работоспособности

        while(iter->getValue() != 2){
            iter->result();
            std::cout << "Куда?" << "\n";
            std::cin >> route;
            if(route == "d"){
                iter->down();
            }
            if(route == "u"){
```

```
            iter->up();
        }
        if(route == "l"){
            iter->left();
        }
        if(route == "r"){
            iter->right();
        }

        if(route == "g"){
            std::cout << iter->getValue() << "\n";
        }
    }
    std::cout << "you win!" << "\n";



    delete Map::getInstance();
    delete iter;
    return 0;
}
```

Название файла: Makefile

```
DWC=./ClasesForProject/
DWT=./Tests/

all: MyGame

MyGame: main.o Map.o Cell.o Map_Iterator.o
     g++ main.o Map.o Cell.o Map_Iterator.o -o MyGame

main.o: main.cpp $(DWC)Map.h $(DWC)Map_Iterator.h
     g++ -c main.cpp
```

```
	Map_Iterator.o:  $(DWC)Map_Iterator.cpp  $(DWC)Map_Iterator.h  $
(DWC)Map.h $(DWC)Cell.h
		g++ -c $(DWC)Map_Iterator.cpp

	Map.o:	$(DWC)Map.cpp	$(DWC)Map.h	$(DWC)Cell.h	$
(DWC)Map_Iterator.h
		g++ -c $(DWC)Map.cpp

	Cell.o:	$(DWC)Cell.cpp $(DWC)Cell.h
		g++ -c $(DWC)Cell.cpp

	UnitTest1.o: $(DWT)UnitTest1.cpp $(DWC)Cell.h $(DWT)UnitTest.h
		g++ -c $(DWT)UnitTest1.cpp

	UnitTest.o: $(DWT)UnitTest.h
		g++ -c $(DWT)UnitTest.cpp



	clean:
		rm -rf  *.o MyGame test



	test: UnitTest1.o UnitTest.o Cell.o Map.o Map_Iterator.o
		g++  UnitTest1.o  UnitTest.o  Cell.o  Map.o  Map_Iterator.o  -o
test
```