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

ОТЧЕТ

по лабораторной работе №1

по дисциплине « Объектно-ориентированное программирование»

Тема: Создание игрового поля

Студент(ка) гр. 9382 Иерусалимов Н.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Создать класс игрового поля в стиле ООП.

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.** Итератор должен быть совместим со стандартной библиотекой.

Выполнение работы.

Class Cell

Класс клетки. Содержит два приватных поля Object state - что находится в клетке и bool isPassable проходимость клетки.

В публичном поле находится конструктор по умолчанию - Cell(); Метод void State(Object typeOfObject) задает тип объекта который находится на этой клетке и метод Object get_state() - возвращает тип объекта. Также в файле Cell.h содержится enum object где идет перечисление всех возможных типов объекта, каждому типу присвоена своя константа в соответствии их порядку

OBJECT_FREE - 0

OBJECT_WALL - 1

OBJECT_EXIT - 2

OBJECT_ENTRY - 3

OBJECT_ITEM - 4

class Field

Класс игрового поля. Элементами поля являются клетки — экземпляры класса Cell. При создании класса использован паттерн Синглтон. Приватные поля класса: int x — ширина поля , int y — длина поля, Cell** field - хранит двумерный массив с клетками, static Field* sing — хранит указатель на единственный экземпляр класса. Field(int x, int y) — конструктор, путем определения Cell** field и метода State(Object typeOfObject) создается классическое поле. Field() - конструктор по умолчанию, Field (const Field& other) — конструктор копирования, Field(Field&& other) - конструктор перемещения, Field &
operator = (const Field& other) - оператор присваивания, Field & operator = (Field && other) — оператор перемещения, ~Field() - деструктор.. Публичные static Field* GetInstanceOfField(int x, int y) — возвращает указатель на единственный экземпляр поля int getX() - выводит зачение поля x, int getY() - выводит значение поля y, Cell** getField() - возвращает указатель на field.

class Print

Класс отвечает за вывод на экран игрового поля класса Field. Содержит публичный метод void print(Field& field) для вывода поля на экран.

Тестирование.

Результаты тестирования представлены в табл. 1.

№	**Входные данные**	**Выходные данные**	**Комментарии**

1	int x = 3;
	int y = 3;
	Field* Level1 =
	Field::GetInstanceOfField(x,y);
		131
		101
		121
			Print p;
			p.print(*Level1);

2	int x = 3;
	int y = 3;
	Field* Level1 =
	Field::GetInstanceOfField(x,y);
	Field* Level2 =
	Field::GetInstanceOfField(125,123);
		//Level1
		111311
		100001
		100001
		100001
		100001
		111211
		//Level2
		111311
		100001
		100001
		100001
		100001
		111211
		0x55d3ed2f6e70
		0x55d3ed2f6e70
			Print p;
			p.print(*Level1);
			p.print(*Level2);
			
			std::cout<<Level1<<"\n";
			std::cout<<Level2<<"\n";
			
			//Синглтон работает
			//Адреса разные

Выводы.

Был реализован класс клетки, поля и вывод этого поля. Было реализовано пространство для игры то, где будет происходить те или иные действия.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: src/Cell.cpp

```
#include "../h/Cell.h"

Cell::Cell() {}

void Cell::State(Object typeOfObject){
	this->state = typeOfObject;
	if(typeOfObject == OBJECT_WALL) this->isPassable = 0;
}
Object Cell::get_state() {return this→state;}
```

Название файла: h/Cell.h

```
#ifndef CellH
#define CellH

#include <iostream>
enum Object{
	OBJECT_FREE,
	OBJECT_WALL,
	OBJECT_EXIT,
	OBJECT_ENTRY,
	OBJECT_ITEM,
};

class Cell {
	public:
	Cell();
	void State(Object typeOfObject);
	Object get_state();

	private:
	bool isPassable;
	Object state;

};
```

```
#endif
```

Название файла: src/Field.cpp

```
#include "../h/Field.h"

Field::Field(){}

Field::Field(int x, int y){
	this->y = y;
	this->x = x;
	field = new Cell* [x];
	for(int i =0; i < x; ++i){
		field[i] = new Cell[y];
	}
	for(int i = 0;i <x;++i){
		for(int j = 0;j <y;++j){
			if(i==0||i==x-1||j==0||j==y-1){
				field[i][j].State(OBJECT_WALL);
			}else field[i][j].State(OBJECT_FREE);
		}
	}
	field[x-1][y/2].State(OBJECT_EXIT);
	field[0][y/2].State(OBJECT_ENTRY);
}

//Конструтор копирования
Field::Field (const Field& other){
	x = other.x;
	y = other.y;
	this->field = new Cell* [x];
	for(int i =0; i < x; ++i){
		this->field[i] = new Cell[y];
	}

	for(int i =0; i < x; ++i){
		for(int j = 0; j <y;++j){
			this->field[i][j].State(other.field[i][j].get_state());
		}
	}
```

```
}

//Конструктор перемещения
Field::Field(Field&& other){
	x = other.x;
	y = other.y;
	this->field = new Cell* [x];
	for(int i =0; i < x; ++i){
		this->field[i] = new Cell[y];
	}

	for(int i =0; i < x; ++i){
		for(int j = 0; j <y;++j){
			this->field[i][j]= other.field[i][j];
		}
	}
	other.~Field();
}

//оператор присваивания
Field & Field::operator = (const Field& other){

	if(this->field != nullptr){
		for (int i = 0; i < this->x;++i)
			delete[] field[i];
		delete[] field;
	}
	this->x = other.x;
	this->y = other.y;

	this->field = new Cell* [x];
	for(int i =0; i < x; ++i){
		this->field[i] = new Cell[y];
	}

	for(int i =0; i < x; ++i){
		for(int j = 0; j <y;++j){
			this->field[i][j].State(other.field[i][j].get_state());
		}
```

```
        }
        return *this;
}

//Оператор перемещения
Field & Field::operator = (Field && other){
        if(&other == this)
        return *this;
   for (int i = 0; i < x; i++){
      delete [] this->field[i];
   }
   delete [] this->field;
        this->x = other.x;
        this->y = other.y;

        this->field = new Cell* [x];
        for(int i =0; i < x; ++i){
                this->field[i] = new Cell[y];
        }

        for(int i =0; i < x; ++i){
                for(int j = 0; j <y;++j){
                        this->field[i][j] = other.field[i][j];
                }
        }
        return *this;
}



Field* Field::sing = nullptr;

Field* Field::GetInstanceOfField(const int x, const int y){
        if(sing==nullptr){
          sing = new Field(x,y);
        }
        return sing;
}
```

```
int Field::getX(){
	return x;
}

int Field::getY(){
	return y;
}

Cell** Field::getField(){
	return field;
}

Field::~Field(){
	for (int i=0;i<this->x;++i){
		delete [] this->field[i];
	}
	delete [] this->field;
}
```

Название файла: h/Field.h

```
#ifndef FieldH
#define FieldH

#include <iostream>
#include <fstream>
#include "Cell.h"

class Field{
	private:
	int x, y;
	Cell** field;			//Поле разбитое на клетки

	Field();
	Field(Field&& other);				//Конструктор перемещения
	Field (const Field& other);		//Конструктор копирования
	Field & operator = (const Field& other);	//Оператор присваивания
	Field & operator = (Field && other);		//Оператор перемещения
	Field(int x, int y);			//Создание поля
```

```
	public:

	static Field* GetInstanceOfField(const int x,const int y);		//Синголтон, через этот метод будем
объявлять поля
	Cell** getField();
	~Field();//Деструктор
	int getX();
	int getY();
	protected:
	static Field* sing;
};
#endif
```

Название файла: src/Print.cpp

```
#include "../h/Print.h"

void Print::print(Field& field){
	for(int i =0; i < field.getX();++i){
		for(int j = 0;j < field.getY();++j){
			std::cout<<field.getField()[i][j].get_state();
		}
		std::cout<<'\n';
	}
}
```

Название файла: h/Print.h

```
#ifndef PrintH
#define PrintH

#include <iostream>
#include "Field.h"
 using namespace std;
class Print{
	public:
	void print(Field& field);
};
```

```
#endif
```

Название файла: src/main.cpp

```
#include "../h/Field.h"
#include "../h/Print.h"
#include <iostream>
int main(){
	int x = 6;
	int y = 6;
	Field* Level1 = Field::GetInstanceOfField(x,y);
	Field* Level2 = Field::GetInstanceOfField(x,y);
	std::cout<<Level1<<"\n";
	std::cout<<Level2<<"\n";
	Print p;
	p.print(*Level1);
	p.print(*Level2);
	return 0;
}
```