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 классов, конструкторов классов, методов классов, наследование

Студент гр. 8304 _____ Порывай П.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Научиться создавать классы и их конструкторы, реализовать методы классов, ознакомится с наследованием классов

Задание

Разработать и реализовать набор классов:

- Класс игрового поля
- Набор классов юнитов

Игровое поле является контейнером для объектов представляющим прямоугольную сетку. Основные требования к классу игрового поля:

- Создание поля произвольного размера
- Контроль максимального количества объектов на поле
- Возможность добавления и удаления объектов на поле
- Возможность копирования поля (включая объекты на нем)
- Для хранения запрещается использовать контейнеры из stl

Юнит является объектов, размещаемым на поля боя. Один юнит представляет собой отряд. Основные требования к классам юнитов:

- Все юниты должны иметь как минимум один общий интерфейс
- Реализованы 3 типа юнитов (например, пехота, лучники, конница)
- Реализованы 2 вида юнитов для каждого типа(например, для пехоты могут быть созданы мечники и копейщики)
- Юниты имеют характеристики, отражающие их основные атрибуты, такие как здоровье, броня, атака.
- Юнит имеет возможность перемещаться по карте

Баллы за лаб. работу (* отмечает необязательные пункты)

Ход работы

1) Был реализован класс playing_field, хранящий информацию о поле и само поле, которое хранит элементы класса cell, playing_field имеет также методы добавления/удаления юнита на поле(так как класс cell наследует класс unit то удаление объекта юнита не происходит, но удаляется динамически выделенная память под юнит).Для класса cell и playing_field созданы конструкторы копии, имеющие различную структуру(Так как при возвращении из функции и перегрузке оператора =, а также при одиночной перегрузке те присваивании конструкторы имеют разную роль, для cell показано возвращение из функции, для playinf_field присваивание) Для более детального объяснения работы конструктора копии при вызове каждой функции, связанной с ним на экран выводится соответствующее предложение

2)Был реализован класс unit, наследованый от archer, cavalry, foot хранящий основные данные об объекте(тип, вид, здоровье, ..). Для большей гибкости члены класса, хранящие данные, были объявлены, как protected.

3)Были реализованы «предки» для unit — это archer, cavalry, foot. Каждый из них содержит информацию об определенном виде оружия, принадлежащим только одному из них

Выводы.

Был разработан и реализован набор классов: класс игрового поля, удовлетворяющий требованиям, таким как, создание поля произвольного размера, контроль максимального количества объектов на поле.Есть возможность добавления, удаления юнитов. Юниты имеют один общий интерфейс, имеют возможность перемещаться по карте.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

cell_ancestors.h

```
#pragma once
#include <iostream>
#include <cstring>
#define nullptr NULL
using namespace std;

class foot {
protected:
	char e_weapon_foot[50];
public:
	foot() {

		std::cout << "\n\n Создаем объект пехота\n";

	}
	~foot() {

		std::cout << "Удаление объекта пехота\n\n";
	}

};

class archers {//Поменять название
protected:
	char e_weapon_luchniki[50];
public:

	archers() {

		std::cout << "Создаем объект лучники\n";
	}
	~archers() {
		std::cout << "Удаление объекта лучники\n";4
```

```
	}
	//
};

class cavalry {
protected:
	char e_weapon_konnica[50];
public://
	cavalry() {

		std::cout << "Создаем объект конница\n";

	}
	~cavalry() {

		std::cout << "Удаление объекта конница\n";
	}
};


class unit:public foot,public archers, public cavalry {
protected:

	char *type;
	char* kind;
	int health;
	int attack_force;
	int paracetomol;

public:

	char* get_type() {
		return type;
	};

	int get_health() {
		return health;
	};
```

```
	int get_attack_force() {
		return attack_force;
	};

	int get_paracetomol() {
		return paracetomol;
	};

	unit() {

		std::cout << "Создается объект юнит\n";
		//strcpy_s(type, 50, "Пустая клетка");

	};

	~unit() {

		std::cout << " Удаляется объект юнит\n";
	}


};



class cell :public unit {

	int write_in;
public:

	cell() {

		std::cout << "Создается объект клетка \n\n";
		type = nullptr;
		kind = nullptr;
		write_in = 0;//Клетка пустая
		//strcpy_s(type, 50, "Пустая клетка");
	}
```

```
	cell(const cell& ob);//конструктор копии lvalue-reference
	//cell(cell && s);//rvalue-reference перемещающий конструктор
	//cell& operator=(cell && s);

	~cell() {//Для корректного удаления при выделении памяти под наследованый класс

		if (write_in == 1) {

			delete type;
			delete kind;
		}

		std::cout << "\n\nУдаление объекта клетки\n";
	}



	int is_writein() {
		return write_in;
	}

	bool operator==(cell& op2) {

		if (strcmp(this->type, op2.type) == 0 && strcmp(this->kind, op2.kind) == 0)
			return true;
		else
			return false;
	}

	void set(const char* type_s, const char *kind_s);

	cell operator=(const cell& ob);

	void remove() {

		if (write_in == 1) {
			delete kind;
```

```
                delete type;

                write_in = 0;
            }
            else if (write_in == 0) {

                std::cout << "\nКлетка уже пуста\n";

            }

      }
      //void inic(const char* type, const char* kind, int healt, int
attack, int paracetomol);
};
```

cell_ancestors.cpp

```
#include"cell_ancestors.h"

/*  ?конструктор перемещения
cell::cell(cell && s){


	std::cout << "\nВызван конструктор перемещения\n";

	type = s.type;
	s.type = nullptr;


}


cell& cell::operator=(cell&& ob) {


	std::cout << "\nВызвана операторная функция со значением
rvalue\n";
	if (type != nullptr)
		delete type;

	type = ob.type;


	health = ob.health;
	attack_force = ob.attack_force;
	paracetomol = ob.paracetomol;
	strcpy_s(e_weapon_foot, 50, ob.e_weapon_foot);
	strcpy_s(e_weapon_luchniki, 50, ob.e_weapon_luchniki);
	strcpy_s(e_weapon_konnica, 50, ob.e_weapon_konnica);
	return *this;
}

cell init_move() {
```

```
	std::cout << "\nВызвана функция init_move()\n";
	cell ob;
	char type[50] = "";

	std::cout << "Введите тип\n";
	std::cin.getline(type, 50);

	ob.set(type);


	return std::move(ob);//Здесь вызывается консруктор копии
}

*/


cell::cell(const cell& ob) {

	type = new char[50];
	kind = new char[50];
	strcpy(type, ob.type);
	strcpy(kind, ob.kind);


	std::cout << "\nКонструктор копии клетки\n";

	if (strcmp(type, "Пехота") == 0) {

		strcpy(e_weapon_foot,  "деревянная палка");//надо?
		health = 20;
		attack_force = 20;
		paracetomol = 10;
	}
	else if (strcmp(type, "Лучники") == 0) {

		strcpy(e_weapon_luchniki, "камешек");
		health = 5;
```

```
            attack_force = 10;
            paracetomol = 15;

      }
      else if (strcmp(type, "Конница") == 0) {

            strcpy(e_weapon_konnica, "голова грифона");

            health = 15;
            attack_force = 30;
            paracetomol = 5;

      }

}


void cell::set(const char* type_s , const char* kind_s) {

            type = new char[50];
            kind = new char[50];

            strcpy(type, type_s);//Перепроверить как эта функция
работает сделать через strlen
            strcpy(kind, kind_s);

            if (strcmp(type, "Пехота") == 0) {

                  strcpy(e_weapon_foot, "деревянная палка");//граница
копирования
                  health = 20;
                  attack_force = 20;
                  paracetomol = 10;

            }
            else if (strcmp(type, "Лучники") == 0) {
```

```
                strcpy(e_weapon_luchniki, "камешек");
                health = 5;
                attack_force = 10;
                paracetomol = 15;

        }
        else if (strcmp(type, "Конница") == 0) {

                strcpy(e_weapon_konnica, "голова грифона");

                health = 15;
                attack_force = 30;
                paracetomol = 5;

        }


        if (strcmp(type, "Пустая клетка") != 0)
                write_in = 1;


}


cell cell::operator=(const cell& ob) {

    std::cout << "\n Перегрузка оператора = \n";
    if (type != nullptr ) {



                delete type;
                type = new char[50];


                delete kind;
                kind = new char[50];
```

```
}
else {
	type = new char[50];
	kind = new char[50];
}

std::cout <<"\n"<< ob.type<<"\n";
std::cout << "\n" << type << "\n";

strcpy(type, ob.type);
strcpy(kind, ob.kind);


if (strcmp(type, "Пехота") == 0) {

	strcpy(e_weapon_foot, "деревянная палка");
	health = 20;
	attack_force = 20;
	paracetomol = 10;
}
else if (strcmp(type, "Лучники") == 0) {

	strcpy(e_weapon_luchniki, "камешек");
	health = 5;
	attack_force = 10;
	paracetomol = 15;

}
else if (strcmp(type, "Конница") == 0) {

	strcpy(e_weapon_konnica, "голова грифона");

	health = 15;
	attack_force = 30;
	paracetomol = 5;
```

```
	}


	if (strcmp(type, "Пустая клетка") != 0)
		write_in = 1;

	return *this;

}


cell init() {


	std::cout << "\nВызвана функция init()\n";
	cell ob;
	char type[50] = "";
	char kind[50] = "";

	std::cout << "Введите тип\n";
	std::cin>>type;
	std::cout << "Введите вид\n";
	std::cin>>kind;

	ob.set(type , kind);


	return ob;//Здесь вызывается консруктор копии

}
```

playing_field.h

```
#include"cell_ancestors.h"
cell init();
class playing_field {

	int max_obj;
	int height;
	int width;
	cell** p_f;

public:


	playing_field(int height, int width);
	//playing_field(const playing_field& ob2);//Конструктор копии


	void remove_obj(int i, int j);
	playing_field() {

		max_obj = 0;
		height = width = 0;
		p_f = nullptr;

		std::cout << "Объект создан с помощью конструктора без
параметров\n\n";

	}

	void inic_cell(int i, int j);//клетка
	//Для след функций можно сделать проверку указателя p_f
	char* type_cell(int i, int j) {
```

```
            return p_f[i][j].get_type();

    }


    int health_cell(int i, int j) {

         return p_f[i][j].get_health();
    }

    int attack_cell(int i, int j) {

         return p_f[i][j].get_attack_force();

    }

    int drug_cell(int i, int j) {

         return p_f[i][j].get_paracetomol();
    }

    void output_field();
    int get_height() {
         return height;

    }

    int get_width() {
         return width;
    }

    void move(int i1, int j, int i2, int j2);

    //playing_field operator=(playing_field& ob);
    ~playing_field();
```

```
};
```

playing_field.cpp

```
#include"cell_ancestors.h"
#include"playing_field.h"

void playing_field::move(int i1, int j1, int i2, int j2) {

      std::cout << "\nВызвана функция move\n";
      p_f[i1][j1] = p_f[i2][j2];//вызывается конструктор копии;

      remove_obj(i2, j2);

}



void playing_field::remove_obj(int i, int j) {


      std::cout << "\nУдаляем юнит на клетке\n";
      p_f[i][j].remove();

}


 playing_field::~playing_field() {


       if (p_f) {

             for (int i = 0; i < height; i++) {
```

```
                delete[] p_f[i];//таким образом удаляется массив
объектов


            }

            delete p_f;
      }

            std::cout << "Объект поле удален\n";



}




void playing_field::output_field() {



     for (int i = 0; i < height; i++) {


           std::cout << "\n";

           for (int j = 0; j < width; j++) {


                if (p_f[i][j].is_writein() == 1) {

                     std::cout << p_f[i][j].get_type();
                     std::cout << " ";
```

```
                }
                else if (p_f[i][j].is_writein() == 0) {

                     std::cout << "Пустая клетка";
                     std::cout << " ";


                }

           }

     }

           std::cout << "\n";

}

void playing_field::inic_cell(int i, int j) {

     std::cout << "\nВызвана инициализация клетки\n";


     p_f[i][j] = init();


}




playing_field::playing_field(int height, int width) {

     this->height = height;
     this->width = width;


     max_obj = height * width;
```

```
	p_f = new cell * [height];

	for (int i = 0; i < height; i++) {

		p_f[i] = new cell[width];

	}



	std::cout << "Поле создано с помощью конструктора с
параметрами \n\n";
}
```

main.cpp

```
#include"playing_field.h"
#include"cell_ancestors.h"

int main()
{

	int x,y,a,b;
	std::cout<<"\n Введите размер поля(формат:x,y)\n";
	cin>>x>>y;


	playing_field ob1(x, y);
	ob1.output_field();

	std::cout<<"\n	Какую	клетку	инициализировать?
(формат:строка,столбец)\n";
	cin>>x>>y;


	ob1.inic_cell(x, y);
	ob1.output_field();

	std::cout<<"\nПередвинуть объект по полю(Куда строка\столбец
откуда строка\столбец)\n";
	std::cin>>x>>y>>a>>b;
	ob1.move(x, y, a, b);
```

```
        ob1.output_field();

        std::cout<<"\nУдалить юнит(строка\столбец)\n";
        cin>>x>>y;
        ob1.remove_obj(x, y);
        ob1.output_field();


        std::cout<<"\nДалее создается единичный объект типа клетка, его
необходимо инициализировать\n";

        cell ob;
           ob = init();
           std::cout << ob.get_type();
Return 0;
}
```

UML диаграмма