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 _____ Матросов Д.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Написание программы в стиле ООП на языке C++. Научиться проектировать сложные проекты.

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

Разработка.

Для решения поставленных задачь необходимо слеоать следующее:

- Написать класс Object от котороо буду унаследованы все игровые объекты (юниты, нейтральные объекты карты, база и т.д.)
- Написать классы юнитов, в часности класс BaseUnit, от котоого буду унаследованны все остальные классы юнитов, и который будет включать в себя общий интерфейс юнитов: функции атаки, передвижения, полуения урона.
- Написать класс игрового поля GameMap. Через этот класс можно взаимодействовать с объектами на поле, жобавлять и удалять новые объекты (для данной лабораторной работы этими объектами являются юниты)
- Для хранения в классе карты объектов написать свою реализацию списка с полным функцианалом: взятие первого или последнего элемента, добавление и удаление элементовб поиск элементов. Для каждого юнита уникальным индентификатором будет является его позиция на поле, так как несколько юнитов на одних координатах находится не могут.
- Реализовать итератор поля для его быстрого обхода.

Ход разработки:

1. Был разработан класс Object, общий родитель всех игровых объектов. (Реализацию см. в приложении А)
2. Были написанны классы юнитов, в том числе общий родитель всех юнитов класс BaseUnit. (Реализацию см. в приложении Б)
3. Был написанн класс GameMap. Реализованны функции добавления и удаления элементов, проверки на наличие объекта в ячейке поля. (Реализацию см. в приложении В)
4. Был реализова класс списка для хранения юнитов. Были реализованны методы поиска, добавления, удаления, взятия первого и последнего элемента. (Реализацию см. в приложении Г)

5. Для всех вышеперечисленных пунктов были реализованны операторы копирования-присваивания и конструкторы копирования.
6. Был реализованн класс итератора игрового поля. (Реализацию см. в приложении Д)

Выводы.

В ходе выполнения лабораторной работы было разработанно несколько классов для реализации части функционала игры, который будет полностью доработан в следующих лабораторных раотах.

ПРИЛОЖЕНИЕ А. КЛАСС OBJECT

```
class Object
{
public:
     Object();

     Object(const Object& copy); //конструктор копирования
(конструктор перемещения не требуется, так как нет ссылочных типов
данных внутри класса)

     ~Object();

     Object& operator=(const Object& copy); //оператор
копирования-присваивания (копирование-перемещение не требуется)

     int getX();
     int getY();
     Factions getFaction();
     ObjectTypes getType();

     void setX(int x);
     void setY(int y);
     void setFactions(Factions f);
     void setType(ObjectTypes t);

protected:

     Factions ObjectFactions;
     ObjectTypes Type;
```

```
     int X_CORD;
     int Y_CORD;

};

Object::Object(){}

Object::Object(const Object& copy) {
     X_CORD = copy.X_CORD;
     Y_CORD = copy.Y_CORD;
     ObjectFactions = copy.ObjectFactions;
     Type = copy.Type;
}

Object::~Object() {}

Object& Object::operator=(const Object& copy) {
     if (&copy == this) {
          return *this;
     }

     X_CORD = copy.X_CORD;
     Y_CORD = copy.Y_CORD;
     ObjectFactions = copy.ObjectFactions;
     Type = copy.Type;

     return *this;
}
```

```
int Object::getX() {
     return X_CORD;
}
int Object::getY() {
     return Y_CORD;
}

Factions Object::getFaction() {
     return ObjectFactions;
}
ObjectTypes Object::getType() {
     return Type;
}

void Object::setX(int x) {

     if (x >= 0) {

          X_CORD = x;

     }
     else {
          /*Бросить ошибку*/
     }
}
void Object::setY(int y) {

     if (y >= 0) {

          Y_CORD = y;
```

```
	}
	else {
		/*Бросить ошибку*/
	}
}

void Object::setFactions(Factions f) {
	ObjectFactions = f;
}

void Object::setType(ObjectTypes t) {
	Type = t;
}
```

ПРИЛОЖЕНИЕ Б. КЛАССЫ ЮНИТОВ

```
/*
##########################################################
Класс базового юнита (может все и ничего одновременно)
##########################################################
*/

class BaseUnit : public Object
{
public:
     BaseUnit(int x, int y, Factions f);
     BaseUnit(const BaseUnit& copy);
     BaseUnit();
     ~BaseUnit();

     BaseUnit& operator=(const BaseUnit& copy);

     void Move(int x, int y);

     void Attack(std::shared_ptr<Object> target);

     void printStatistic();

protected:

     int MaxHp;
     int Hp;
     int MoveDistance;
     int Armor;
```

```
    int AttackDistance;
    int Damage;

    Jobs Job;

    void GetDamage(int damage);


};

/*
##########################################################
Классы юнитов конницы
##########################################################
*/

class Cavalry: public BaseUnit
{
public:
    Cavalry(int x, int y, Factions f);
    Cavalry(const Cavalry& copy);
    ~Cavalry();

    Cavalry& operator=(const Cavalry& copy);

private:

};

class Knight: public BaseUnit
```

```
{
public:
     Knight(int x, int y, Factions f);
     Knight(const Knight& copy);
     ~Knight();

     Knight& operator=(const Knight& copy);

private:

};



/*
##########################################################
Классы юнитов пехоты
##########################################################
*/

class Warrior : public BaseUnit
{
public:
     Warrior(int x, int y, Factions f);
     Warrior(const Warrior& copy);
     ~Warrior();

     Warrior& operator=(const Warrior& copy);

private:
```

```
};

class Piker : public BaseUnit
{
public:
     Piker(int x, int y, Factions f);
     Piker(const Piker& copy);
     ~Piker();

     Piker& operator=(const Piker& copy);

private:

};

/*
##########################################################
Классы юнитов дальнего боя
##########################################################
*/

class Archer : public BaseUnit
{
public:
     Archer(int x, int y, Factions f);
     Archer(const Archer& copy);
     ~Archer();

     Archer& operator=(const Archer& copy);
```

```
private:

};

class Arbalester : public BaseUnit
{
public:
     Arbalester(int x, int y, Factions f);
     Arbalester(const Arbalester& copy);
     ~Arbalester();

     Arbalester& operator=(const Arbalester& copy);

private:

};

/*
#######################################
Здесь и далее функции класса BaseUnite
#######################################
*/


BaseUnit::BaseUnit(int x, int y, Factions f){

     X_CORD = x;
     Y_CORD = y;
     ObjectFactions = f;
```

```
	Type = ObjectTypes::Unit;
	MaxHp = 1;
	Hp = 1;
	Damage = 0;
	AttackDistance = 0;
	MoveDistance = 0;
	Armor = 0;
	Job = Jobs::Standing;
}

BaseUnit::BaseUnit(const BaseUnit& copy) {
	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;
}

BaseUnit::BaseUnit(){}

BaseUnit::~BaseUnit(){}

BaseUnit& BaseUnit::operator=(const BaseUnit& copy) {
	if (&copy == this) {
```

```
            return *this;
        }

        X_CORD = copy.X_CORD;
        Y_CORD = copy.Y_CORD;
        ObjectFactions = copy.ObjectFactions;
        MaxHp = copy.MaxHp;
        Hp = copy.Hp;
        Damage = copy.Damage;
        AttackDistance = copy.AttackDistance;
        MoveDistance = copy.MoveDistance;
        Armor = copy.Armor;
        Job = copy.Job;
        Type = copy.Type;

        return *this;
    }

    void BaseUnit::Move(int x, int y) {
        if (abs(x - this->getX()) <= this->MoveDistance &&
            abs(y - this->getY()) <= this->MoveDistance) {

            /*Не забыть поймать ошибку*/

            this->setX(x);

            /*Не забыть поймать ошибку*/

            this->setY(y);
        }
```

```
}

void BaseUnit::GetDamage(int damage) {

     int IncomingDamage = damage - Armor;

     if (IncomingDamage <= 0) {

          IncomingDamage = 0;

     }

     Hp -= IncomingDamage;
}

void BaseUnit::Attack(std::shared_ptr<Object> target) {
     if (target->getFaction() != getFaction() &&
          target->getType() != ObjectTypes::NonPlayableStructure) {

          if (abs(target->getX() - getX()) <= AttackDistance &&
               abs(target->getY() - getY()) <= AttackDistance) {

               target->GetDamage(Damage);

          }
          else {
               /*Бросить ошибку "Слишком далеко!"*/
          }

     }
```

```
	else {
		/*Бросить ошибку "Ты вообще кого атакуешь?"*/
	}
}

void BaseUnit::printStatistic() {
	std::cout << X_CORD << " " << Y_CORD << " " << int(Type) << "
" << int(ObjectFactions) <<" "<< Hp << "/" << MaxHp  << std::endl;
}

/*
################################################
Здесь и далее функции классаов Cavalry и Knight
################################################
*/

Cavalry::Cavalry(int x, int y, Factions f)
{
	X_CORD = x;
	Y_CORD = y;
	ObjectFactions = f;
	Type = ObjectTypes::Unit;
	MaxHp = 100;
	Hp = MaxHp;
	Damage = 30;
	AttackDistance = 1;
	MoveDistance = 5;
	Armor = 10;
	Job = Jobs::Standing;
}
```

```
Cavalry::Cavalry(const Cavalry& copy) {
	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;
}

Cavalry::~Cavalry()
{

}

Cavalry& Cavalry::operator=(const Cavalry& copy) {
	if (&copy == this) {
		return *this;
	}

	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
```

```
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;

	return *this;
}

Knight::Knight(int x, int y, Factions f)
{
	X_CORD = x;
	Y_CORD = y;
	ObjectFactions = f;
	ObjectTypes::Unit;
	MaxHp = 200;
	Hp = MaxHp;
	Damage = 50;
	AttackDistance = 1;
	MoveDistance = 5;
	Armor = 20;
	Job = Jobs::Standing;
}

Knight::Knight(const Knight& copy) {
	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
```

```
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;
}

Knight::~Knight()
{

}

Knight& Knight::operator=(const Knight& copy) {
	if (&copy == this) {
		return *this;
	}

	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;
```

```
	return *this;
}

/*
################################################
Здесь и далее функции классаов Cavalry и Knight
################################################
*/

Warrior::Warrior(int x, int y, Factions f)
{
	X_CORD = x;
	Y_CORD = y;
	ObjectFactions = f;
	ObjectTypes::Unit;
	MaxHp = 150;
	Hp = MaxHp;
	Damage = 30;
	AttackDistance = 1;
	MoveDistance = 1;
	Armor = 30;
	Job = Jobs::Standing;
}

Warrior::Warrior(const Warrior& copy) {
	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
```

```
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;
}

Warrior::~Warrior()
{

}

Warrior& Warrior::operator=(const Warrior& copy) {
	if (&copy == this) {
		return *this;
	}

	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;
```

```
    return *this;
}

Piker::Piker(int x, int y, Factions f)
{
    X_CORD = x;
    Y_CORD = y;
    ObjectFactions = f;
    ObjectTypes::Unit;
    MaxHp = 200;
    Hp = MaxHp;
    Damage = 40;
    AttackDistance = 1;
    MoveDistance = 1;
    Armor = 45;
    Job = Jobs::Standing;

}

Piker::Piker(const Piker& copy) {
    X_CORD = copy.X_CORD;
    Y_CORD = copy.Y_CORD;
    ObjectFactions = copy.ObjectFactions;
    MaxHp = copy.MaxHp;
    Hp = copy.Hp;
    Damage = copy.Damage;
    AttackDistance = copy.AttackDistance;
    MoveDistance = copy.MoveDistance;
    Armor = copy.Armor;
    Job = copy.Job;
```

```
	Type = copy.Type;

}

Piker::~Piker()
{

}

Piker& Piker::operator=(const Piker& copy) {
	if (&copy == this) {
		return *this;
	}

	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;

	return *this;
}

/*
```

```
################################################
Здесь и далее функции классаов Cavalry и Knight
################################################
*/

Archer::Archer(int x, int y, Factions f)
{
	X_CORD = x;
	Y_CORD = y;
	ObjectFactions = f;
	ObjectTypes::Unit;
	MaxHp = 60;
	Hp = MaxHp;
	Damage = 40;
	AttackDistance = 6;
	MoveDistance = 2;
	Armor = 5;
	Job = Jobs::Standing;
}

Archer::Archer(const Archer& copy) {
	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
```

```
	Job = copy.Job;
	Type = copy.Type;

}

Archer::~Archer()
{

}

Archer& Archer::operator=(const Archer& copy) {
	if (&copy == this) {
		return *this;
	}

	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;

	return *this;
}
```

```
Arbalester::Arbalester(int x, int y, Factions f)
{
	X_CORD = x;
	Y_CORD = y;
	ObjectFactions = f;
	ObjectTypes::Unit;
	MaxHp = 70;
	Hp = MaxHp;
	Damage = 70;
	AttackDistance = 5;
	MoveDistance = 2;
	Armor = 10;
	Job = Jobs::Standing;
}

Arbalester::Arbalester(const Arbalester& copy) {
	X_CORD = copy.X_CORD;
	Y_CORD = copy.Y_CORD;
	ObjectFactions = copy.ObjectFactions;
	MaxHp = copy.MaxHp;
	Hp = copy.Hp;
	Damage = copy.Damage;
	AttackDistance = copy.AttackDistance;
	MoveDistance = copy.MoveDistance;
	Armor = copy.Armor;
	Job = copy.Job;
	Type = copy.Type;

}
```

```
Arbalester::~Arbalester()
{

}

Arbalester& Arbalester::operator=(const Arbalester& copy) {
     if (&copy == this) {
          return *this;
     }

     X_CORD = copy.X_CORD;
     Y_CORD = copy.Y_CORD;
     ObjectFactions = copy.ObjectFactions;
     MaxHp = copy.MaxHp;
     Hp = copy.Hp;
     Damage = copy.Damage;
     AttackDistance = copy.AttackDistance;
     MoveDistance = copy.MoveDistance;
     Armor = copy.Armor;
     Job = copy.Job;
     Type = copy.Type;

     return *this;
}
```

ПРИЛОЖЕНИЕ В. КЛАСС GAMEMAP

```
class GameMap
{
public:
	GameMap(int x, int y);
	GameMap(const GameMap& copy);

	~GameMap();

	bool addObject(BaseUnit& obj);
	bool removeObject(BaseUnit& obj);

	std::shared_ptr<BaseUnit> Check(int x, int y);

	bool moveObject(std::shared_ptr<BaseUnit> obj, int x, int y);

	GameMap& operator=(const GameMap& copy);

	typedef OwnIterator<MyList> iterator;

	iterator begin();

private:

	int ObjectNumber;
	int MaxObjectNumber;

	int y_size;
```

```
        int x_size;

        MyList Obj;
};


GameMap::GameMap(int x, int y)
{

        if (x > 0 && y > 0) {
                MaxObjectNumber = x * y;
                ObjectNumber = 0;
                x_size = x;
                y_size = y;
        }
        else {
                MaxObjectNumber = 0;
                ObjectNumber = 0;
                x_size = 0;
                y_size = 0;
        }
}

GameMap::GameMap(const GameMap& copy) {

        MaxObjectNumber = copy.MaxObjectNumber;
        ObjectNumber = copy.ObjectNumber;
        x_size = copy.x_size;
```

```
        y_size = copy.y_size;
        Obj = copy.Obj;
}

GameMap::~GameMap()
{

}

GameMap& GameMap::operator=(const GameMap& copy) {

        MaxObjectNumber = copy.MaxObjectNumber;
        ObjectNumber = copy.ObjectNumber;
        x_size = copy.x_size;
        y_size = copy.y_size;
        Obj = copy.Obj;

        return *this;

}

bool GameMap::addObject(BaseUnit& obj) {

        if (obj.getX() >= 0 && obj.getY() >= 0) {

                if (Obj.Find(obj.getX(), obj.getY()) == nullptr && ObjectNumber < MaxObjectNumber) {
```

```
                Obj.Push(obj);

                ObjectNumber++;

                return true;
            }

            return false;

        }
        else {

            /*Бросить ошибку "Мимо игрового поля"*/
            return false;

        }

}
bool GameMap::removeObject(BaseUnit& obj) {

        if (Obj.Remove(obj.getX(), obj.getY())) {
            ObjectNumber--;
            return true;
        }
        return false;
}

std::shared_ptr<BaseUnit> GameMap::Check(int x, int y) {
```

```
	return Obj.Find(x, y);
}

bool GameMap::moveObject(std::shared_ptr<BaseUnit> obj, int x, int y) {

	if (x >= 0 && y >= 0 && x < x_size && y < y_size) {
		if (Check(x, y) == nullptr) {
			obj->Move(x, y);
			return true;
		}
		else {
			return false;
		}
	}
	else {
		/*Бросить ошибку*/
		return false;
	}
}


GameMap::iterator GameMap::begin()
{
	return iterator(Obj.begin());
}
```

```
struct Node
{
    std::shared_ptr<BaseUnit> data;
    std::shared_ptr<Node> next;
};

class MyList
{
private:
    std::shared_ptr<Node> head;
public:

    MyList()
    {
        head = nullptr;
    }

    MyList(const MyList& copy);

    ~MyList();

    MyList& operator=(const MyList& copy);


    MyList* begin() {
        return this;
    }
```

```
    MyList end();

    Node takeHead() {
        return *head;
    }

    void Push(BaseUnit& d);
    std::shared_ptr<BaseUnit> Find(int x, int y);
    bool Remove(int x, int y);
    MyList& operator++();
};

MyList::MyList(const MyList& copy) {
    std::shared_ptr<Node> tmp_head = copy.head;
    head = nullptr;

    while (tmp_head != nullptr) {
        BaseUnit tmp_obj = BaseUnit(*tmp_head->data.get());
        Push(tmp_obj);
        tmp_head = tmp_head->next;
    }
}

MyList::~MyList() {}

MyList& MyList::operator=(const MyList& copy){
```

```
    std::shared_ptr<Node> tmp_head = copy.head;
    head = nullptr;

    while (tmp_head != nullptr) {
        BaseUnit tmp_obj = BaseUnit(*tmp_head->data.get());
        Push(tmp_obj);
        tmp_head = tmp_head->next;
    }

    return *this;
}

void MyList::Push(BaseUnit& d)
{
    Node new_nd;
    new_nd.data = std::make_shared<BaseUnit>(d);
    new_nd.next = nullptr;
    std::shared_ptr<Node> nd = std::make_shared<Node>(new_nd);

    if (head == nullptr)
        head = nd;
    else
    {
        std::shared_ptr<Node> current = head;
        while (current->next != nullptr)
            current = current->next;
        current->next = nd;
    }
```

```
}

std::shared_ptr<BaseUnit> MyList::Find(int x, int y) {
    std::shared_ptr<Node> tmp_head = head;
    while(tmp_head != nullptr) {
        if (tmp_head->data->getX() == x && tmp_head->data->getY() == y) {
            return tmp_head->data;
        }
        tmp_head = tmp_head->next;
    }
    return nullptr;
}

bool MyList::Remove(int x, int y) {
    std::shared_ptr<Node> tmp_head = head;
    std::shared_ptr<Node> previous_elem;
    while (tmp_head != nullptr) {
        if (tmp_head->data->getX() == x && tmp_head->data->getY() == y) {
            previous_elem->next = tmp_head->next;
            return true;
        }
        previous_elem = tmp_head;
        tmp_head = tmp_head->next;
    }

    return false;
}
```

```
MyList MyList::end() {
    if (this->head != nullptr) {
        MyList tmp_list = *this;
        while (tmp_list.head->data != nullptr) {
            tmp_list.head = tmp_list.head->next;
        }
        return tmp_list;
    }
    return *this;
}

MyList& MyList::operator++() {
    if (this->head != nullptr) {
        MyList tmp_list = *this;
        tmp_list.head = tmp_list.head->next;
        return tmp_list;
    }
    return *this;
}
```

```
template<typename ValueType>
class OwnIterator : public std::iterator<std::input_iterator_tag, ValueType>
{
	friend class OwnContainer;
private:
public:
	OwnIterator(ValueType* p);
	bool operator!=(OwnIterator const& other) const;
	bool operator==(OwnIterator const& other) const;
	typename OwnIterator<ValueType>::reference operator*() const;
	OwnIterator<ValueType>& operator++();
private:
	ValueType* p;
};

template<typename ValueType>
OwnIterator<ValueType>::OwnIterator(ValueType* p) : p(p)
{

}

template<typename ValueType>
bool OwnIterator<ValueType>::operator!=(OwnIterator const& other) const
{
	return p != other.p;
}
```

```
template<typename ValueType>
bool OwnIterator<ValueType>::operator==(OwnIterator const& other) const
{
	return p == other.p;
}

template<typename ValueType>
typename OwnIterator<ValueType>::reference OwnIterator<ValueType>::operator*() const{
	return *p;
}

template<typename ValueType>
OwnIterator<ValueType>& OwnIterator<ValueType>::operator++()
{
	++p;
	return this;
}
```