# Лабораторная работа №2

**initializer\_list,** **move, default, delete, move итераторы**

## Задание 1. Создайте класс, который должен быть "оберткой" для вектора с УНИКАЛЬНЫМИ значениями любого типа в заданном диапазоне.

Внимание: при инициализации **НЕ** нужно менять порядок значений, заданный пользователем! При наличии повторяющихся значений нужно оставить первое!

* Для хранения элементов используйте **std::vector.**Замечание:
  + можно использовать std::vector в качестве внедренного объекта
  + а можно наследовать свой класс от std::vector – при этом важно каким должен быть спецификатор наследования
* Реализуйте конструктор, который может принимать любое количество значений (значения могут повторяться)
* Реализуйте метод добавления любого количества значений (значения могут повторяться)
* Реализуйте метод удаления любого количества значений (значения могут повторяться)
* Реализуйте метод сортировки, который будет принимать в качестве параметра признак по возрастанию / по убыванию
* и другие (полезные на Ваш взгляд) методы
* явно обозначьте – какие методы вы делегируете сгенерировать компилятору автоматически

## Задание 2. Разработайте класс, который реализует функциональность очереди с элементами любого типа.

Данные требуется хранить в динамическом массиве, при этом использовать массив как циклический буфер. Пояснение: так как очередь – это специфическая структура данных, для которой новые данные помещаются в конец, а «старые» данные изымаются из начала очереди => если последний элемент массива задействован, то начало скорее всего уже освободилось => «закольцовываем» буфер, продолжая заполнять с нулевого элемента.

Несмотря на указанную специфичность такой структуры данных, могут возникать ситуации, когда пользователь вызвал push(), а свободных элементов в очереди не осталось => при необходимости массив следует «расширять».

При реализации нужно обеспечить эффективную работу с динамической памятью=>

* предусмотреть наличие резервных элементов
* память без очевидной необходимости не перераспределять

Внимание!

1. Очередь реализуем без использования «сырой памяти»! А эффективность достигаем за счет использования move-семантики
2. Очередь выводим на печать с помощью range-base-for

Тестируем разработанный класс на приведенном ниже фрагменте. Замечание: если Вы не учились на предыдущих курсах и не разрабатывали используемый в примере класс MyString, просто замените его на std::string

Следующий фрагмент должен работать не только корректно, но и эффективно:

MyQueue<MyString> q1{ MyString(“AAA”), MyString (“qwerty”),<другие\_инициализаторы>};

//использование MyQueue в диапазонном for:

for (const auto& el : q1) { std::cout << el << ' '; }

MyString s(“abc”);

q1.push(s);

q1.push(MyString(“123”));

MyString s1 = q1.pop();

q1.push(“qqq”);

MyQueue < MyString > q2 = q1;

MyQueue < MyString > q22 = std::move(q1);

MyQueue < MyString > q3{10, MyString (“!”)}; //очередь должна содержать 10 элементов со строкой «!»

q1 = q3;

q2 = MyQueue < MyString > (5, MyString (“?”));

q1 = { MyString(“bbb”), MyString (“ssss”)};

## Задание 3. Реализуйте шаблон класса MyUniquePTR, который является оберткой для указателя на объект любого типа.

Задача – класс должен обеспечивать единоличное владение динамически создаваемым объектом. Проверить функционирование шаблона на примере MyString:

{

MyUniquePTR<MyString> p1(new MyString (“abc”));

std:: cout<<p1->GetString();

p1->SetNewString(“qwerty”);

MyString s2 = \*p1;

//MyUniquePTR< MyString > p2=p1; //здесь компилятор должен выдавать ошибку =>

Исправьте!

If(p1) {std::cout<<”No object!”} //а это должно работать

MyUniquePTR< MyString > p3(new MyString (“vvv”));

//p3 = p2; //и здесь компилятор должен выдавать ошибку

vector< MyUniquePTR< MyString >> v; //как проинициализировать???

list< MyUniquePTR< MyString >> l;

//как скопировать из v в l ???

}