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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Списки и векторы

Студент гр. 7304 _____ Нгуен К.Х.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Исследование структур и операций над вектором и списком на языке C ++.

Задание

1. Вектор

Необходимо реализовать

+ конструкторы и деструктор для контейнера вектор.

+ операторы присваивания и функцию assign для контейнера вектор.

+ функции resize и erase для контейнера вектор.

+ функции insert и push_back для контейнера вектор.

Поведение реализованных функций должно быть таким же, как у класса std::vector (http://ru.cppreference.com/w/cpp/container/vector). Семантику реализованных функций нужно оставить без изменений.

В данном уроке предполагается реализация упрощенной версии вектора, без резервирования памяти под будущие элементы.

2. Список

Необходимо реализовать список со следующими функциями:

+ вставка элементов в голову и в хвост,

+ получение элемента из головы и из хвоста,

+ удаление из головы, хвоста и очистка

+ проверка размера.

+ деструктор

+ конструктор копирования,

+ конструктор перемещения,

+ оператор присваивания.

+ необходимо реализовать итератор для списка. Для краткости реализации можно ограничиться однонаправленным изменяемым (неконстантным)

итератором. Необходимо реализовать операторы: =, ==, !=, ++ (постфиксный и префиксный), *, ->.

С использованием итераторов необходимо реализовать:

+ вставку элементов (Вставляет value перед элементом, на который указывает pos. Возвращает итератор, указывающий на вставленный value),

+ удаление элементов (Удаляет элемент в позиции pos. Возвращает итератор, следующий за последним удаленным элементом).

Поведение реализованных функций должно быть таким же, как у класса std::list (http://ru.cppreference.com/w/cpp/container/list).

Семантику реализованных функций нужно оставить без изменений.

Требования к реализации: при выполнении этого задания вы можете определять любые вспомогательные функции. Вводить или выводить что-либо не нужно. Реализовывать функцию main не нужно. Не используйте функции из cstdlib (malloc, calloc, realloc и free).

Экспериментальные результаты.

Class vector

Создает новый контейнер из разнообразных источников данных, возможно, используя предоставленный пользователем аллокатор `alloc`.

```
explicit vector(size_t count = 0)
```
Создает контейнер с `count` экземплярами `T`, инициализированными конструктором по-умолчанию. При этом копирования не происходит.

```
vector(InputIterator first, InputIterator last)
```
Создает контейнер с содержимым диапазона `[first, last)`.

```
vector(std::initializer_list<Type> init)
```
Создает контейнер с содержимым списка инициализации `init`.

```
vector(const vector& other)
```
Конструктор копирования. Создает контейнер с копией содержимого `other`.

`vector(vector&& other)`
Конструктор перемещения. Создает контейнер с содержимым `other` путём перемещения данных.

`~vector()`
Уничтожает контейнер. После вызова деструктора высвобождается используемая память.

`vector& operator=(const vector& other)`
Заменяет содержимое контейнера. Скопируйте оператор присваивания. Заменяет содержимое с копией содержимого `other`.

`vector& operator=(vector&& other)`
Переместите оператор присваивания. Заменяет содержимое с теми `other`использованием семантика переноса (т.е. данные в `other` перемещается из `other` в этот контейнер). `other`находится в силе, но неопределенное состояние после.

`void assign(InputIterator first, InputIterator last)`
Заменяет содержимое с копиями тех, кто в диапазоне `[first, last)`

`void resize(size_t count)`
Изменяет размер контейнера, чтобы содержать `count` элементы. Если текущий размер меньше, чем `count`, дополнительные элементы добавляются и инициализируется `value`. Если текущий размер больше `count`, контейнер сводится к ее первые элементы `count`.

`iterator erase(const_iterator pos)`
Удаляет элемент в позиции `pos`.

`iterator erase(const_iterator first, const_iterator last)`
Удаляет элементы в диапазоне `[first; last)`.

`iterator insert(const_iterator pos, const Type& value)`
Вставляет `value` перед элементом, на который указывает `pos`.

`iterator insert(const_iterator pos, InputIterator first, InputIterator last)`
Вставляет элементы из диапазона `[first, last)` перед элементом, на который указывает `pos`.

`void push_back(const value_type& value)`
Добавляет данный элемент `value` до конца контейнера.

```
reference at(size_t pos)
const_reference at(size_t pos) const
```
Операция доступа к элементу вектора по индексу. Возвращает ссылку на элемент по индексу `pos`.

```
reference operator[](size_t pos)
const_reference operator[](size_t pos) const
```
Возвращает ссылку на элемент по индексу `pos`.

```
iterator begin()
const_iterator begin() const
```
Возвращает итератор на первый элемент контейнера.

```
iterator end()
const_iterator end() const
```
Возвращает итератор на элемент, следующий за последним элементом контейнера. Этот элемент выступает в качестве заполнителя; попытке доступа к нему приводит к неопределенному поведению.

```
size_t size() const
```
количество элементов в контейнере

```
bool empty() const
```
Проверки, если контейнер не имеет элементов, т.е. является ли `begin() == end()`.

Class list

`list()`

Создает контейнер с `count` экземплярами `T`, инициализированными конструктором по-умолчанию. При этом копирования не происходит.

`~list()`

Уничтожает контейнер. После вызова деструктора высвобождается используемая память.

`list(const list& other)`

Конструктор копирования. Создает контейнер с копией содержимого `other`.

`list(list&& other)`

Конструктор перемещения. Создает контейнер с содержимым `other` путём перемещения данных.

`list& operator= (const list& other)`

Заменяет содержимое контейнера. Скопируйте оператор присваивания. Заменяет содержимое с копией содержимого `other`.

`void push_back(const value_type& value)`
Добавляет данный элемент `value` до конца контейнера.

`iterator insert(iterator pos, const Type& value)`
Вставляет `value` перед элементом, на который указывает `pos`.

`iterator erase(iterator pos)`
Удаляет элемент в позиции `pos`.

`void push_front(const value_type& value)`
Добавляет данного элемента `value` на начало контейнера.

`reference front()`
`const_reference front() const`
Предоставляет доступ к первому элементу

`reference back()`
`const_reference back() const`
Предоставляет доступ к последнему элементу

`void pop_front()`
Удаляет первый элемент контейнера.

`void pop_back()`
Удаляет последний элемент из контейнера.

`void clear()`
Удаляет все элементы из контейнера. Делает недействительными все ссылки, указатели или итераторы указывающие на удалённые элементы. Может также сделать недействительными итераторы после конца последовательности.

`bool empty() const`
Проверки, если контейнер не имеет элементов, т.е. является ли `begin() == end()`.

`size_t size() const`
Возвращает количество элементов в контейнере

`list::iterator begin()`
Возвращает итератор на первый элемент

`list::iterator end()`
Возвращает итератор на элемент, следующий за последним

Выводы.

В результате лабораторной работы рассмотрены структура и операции списка и вектора на языке C ++. Реализованы функции для работы со списком и вектором. Изучены различия в их структуре в памяти и скорости для каждой операции. Изучены ситуации, в которых лучше использовать список или вектор.