**Лабораторная работа № 10**

**Последовательные контейнеры**

**Введение**

*Используйте cppreference.com*

Часть стандартной библиотека шаблонов – standard template library (STL) позволяет представляет пользователю различные уровни абстракции для взаимодействия с данными. Это может быть как структура данных для работы с битами как битовая маска – *std::bitset,* также для работы с стандартными и пользовательскими типами – контейнеры. Хорошим тоном является при написании кода на C++ использование структур данных из STL и других библиотек по требованиям (etc., boost).

К примеру, использование массива. На языке С использование массива будет следующим

|  |
| --- |
| **int** arr[10] = { 0 };  arr[0] = 4; |

Однако, написание кода в С-style является вынужденной практикой за редким исключением, продиктованные обоснованными причинами проекта. В качестве примера исключение может служить реализация контейнеров, т.к там мы используем указатели, хотя в нормальной практике использование указателей прямо следует о плохом коде. Поэтому следует взаимодействовать с массивом в C++ через контейнеры STL – *std::array.*

|  |
| --- |
| std::array<**int**, 5> array = {1, 2, 3, 4, 5};  array[2] = 2; |

Почему же использование контейнеров из STL более предпочтительней? В STL содержится множества алгоритмов и структур данных. Как известно, C++ поддерживает шаблоны, что говорит нам о том, что мы можем писать шаблонный код. Благодаря STL мы может оперировать структурами и алгоритмами и в случае, если нам потребуется изменить структуру, то нам не нужно будет переписывать большое количество кода, а только поменять тип у идентификатора, где используется эта структура.

**STL**

Главной особенностью STL является не количество алгоритмов или структур данных, а стандартизация всех сущностей в библиотеке. Эта стандартизация, к примеру, основана на концепции итераторов, которая позволяет алгоритмам STL работать с различными контейнерами, не зная деталей их реализации. Благодаря этому, контейнеры и алгоритмы STL обладают свойством заменяемости: мы можем использовать разные контейнеры с одними и теми же алгоритмами. При этом, принцип хранения данных, сложность поиска элементов и использование памяти могут отличаться.

**Контейнеры**

В STL есть последовательные, ассоциативные и адаптеры над контейнерами. В этой лабораторной работе исследуются последовательные контейнеры. К последовательным контейнерам относятся: array, vector, deque, forward\_list, list, string и basic\_string. По сути, это реализация известных структур данных. Знание того, как работают такие фундаментальные структуры данных, дает понимание того, как они работают в STL. Останется только изучить доступные методы, а также понять их сложность и область применения.

**Полезные ссылки**

[Массив](https://en.cppreference.com/w/cpp/container/array)

[Динамический массив](https://en.cppreference.com/w/cpp/container/vector)

[Двухсторонняя очередь](https://en.cppreference.com/w/cpp/container/deque)

[Односвязный список](https://en.cppreference.com/w/cpp/container/forward_list)

[Двусвязный список](https://en.cppreference.com/w/cpp/container/list)

**Задания:**

1. Для начала, чтобы понимать, что происходит в контейнерах требуется изучить какие последовательные контейнеры бывают, основные методы и когда их следует использовать – для этого можно перейти на cppreference и почитать документацию на каждый контейнер и на основные методы

2. Изучить исходной код одного из компилятора, можно выбрать GCC или Clang. Clang (LLVM) может быть чуть более читабельным и модульным, но GCC также подойдет.

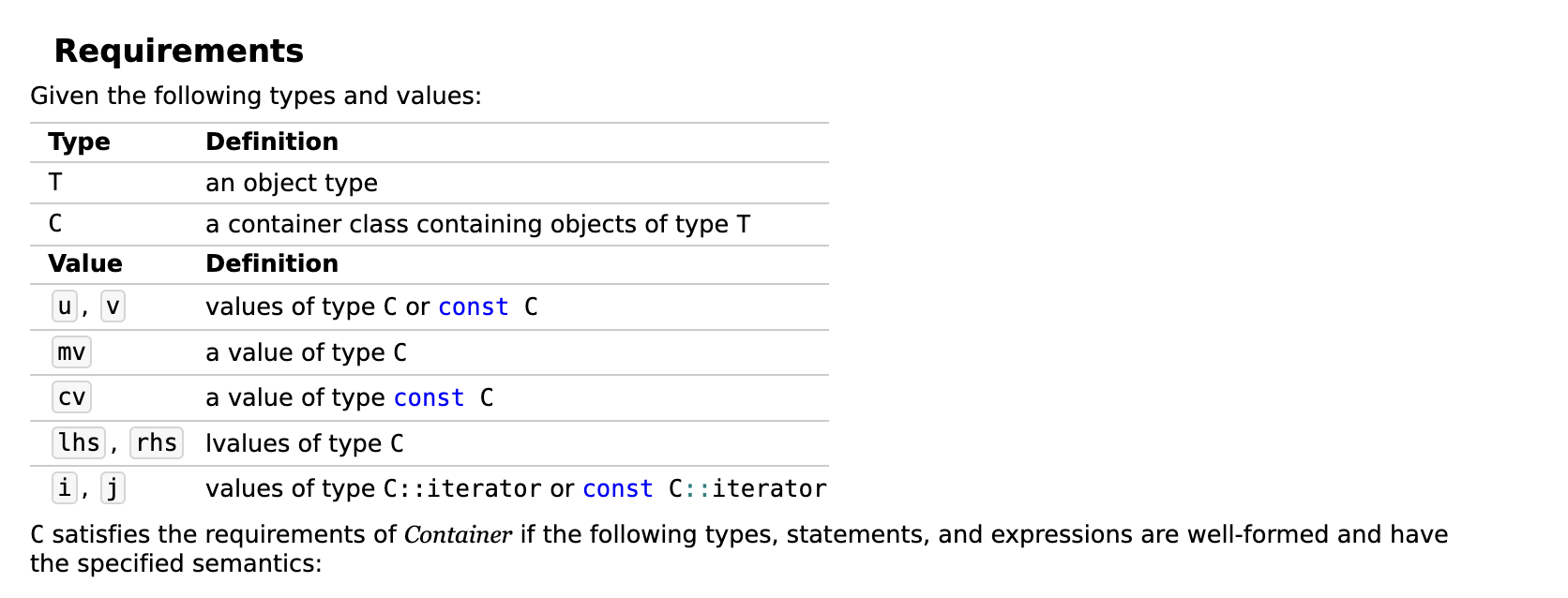
Объяснить как устроены эти структуру в реализации и их работу. *Важно понять принцип работы, а не какие-либо c++ хаки, которые встретятся в коде*.

Структуры для изучения – Массив, Динамический массив, Двухсторонняя очередь, Двусвязный и односвязный список. Ссылки - [array](https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/std/array) [vector](https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/bits/stl_vector.h) [deque](https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/bits/stl_deque.h) [forward\_list](https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/bits/forward_list.h) [list](https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/bits/stl_list.h).

Примерно, что нужно понимать – как размещаются данные в структуре, как происходит добавление, удаление, из чего состоит сама по себе структура и какие у нее есть *helper'ы* (возможно адапторы)для работы. Big O нотацию для каждой структуру и для каждого метода (удаление, добавление, поиск и т.д.)

Для следующего задания следует создавать класс, с таким же названием, как и в STL, однако делать обертку в *namespace* *miet*

3. Самостоятельно реализовать контейнер для динамического массива(шаблонный), при этом следуя требования – в полях контейнера должны обязательно содержаться поля, которые указаны в разделе ***Requirements,*** за исключение поля итератора т.к эта тема еще не изучена (см скриншот снизу).



Какие методы должны быть: at, front, back, data, empty, size, max\_size, reserve, capacity, clear, push\_back, pop\_back

Требование к коду: код должен писаться по определенному стандарту (C++17 с использованием соответствующих преимуществ стандарта), минимальное использованием C-style кода, использовать адекватный code-style.

4. Написать тесты для проверки корректности работы структуру и продемонстрировать работу, используя инструмент gTest, Обязательно проверка краевых случаев, также генерация значений средствами gTest.