|  |  |
| --- | --- |
| ДИСЦИПЛИНА | Технологии индустриального программирования |
| ИНСТИТУТ | ИПТИП |
| КАФЕДРА | Индустриального программирования |
| ВИД УЧЕБНОГО МАТЕРИАЛА | Методические указания к практическим занятиям по дисциплине |
| ПРЕПОДАВАТЕЛЬ | Александров Алексей Сергеевич |
| СЕМЕСТР | 3 семестр, 2024/2025 уч. год |

# Практическое занятие №13

**Контейнеры в Qt. Базовые контейнеры.**

Стандартная библиотека C++ содержит набор шаблонных контейнеров. Контейнеры, по своей сути, напоминают массивы, однако имеют свои отличия. Как минимум, все контейнеры являются классами, которые предоставляют методы для получения размера контейнера, благодаря чему нет необходимости отдельно хранить его размер, в отличие от динамических массивов.

Стандартная библиотека C++ содержит контейнеры двух типов:

1. Последовательные контейнеры обеспечивают эффективный последовательный доступ к элементам. Примерами последовательных структур данных являются двусвязный список и массив, которые в C++ представлены типами list и vector, соответственно.
2. Ассоциативные контейнеры обеспечивают быстрый поиск элементов. Примером ассоциативных структур данных являются множество и словарь, которым соответствуют типы set и map. Тип set позволяет хранить уникальные объекты различных типов, эффективно добавлять, удалять объекты и выполнять поиск. Тип map позволяет хранить пары ключ-значение, причем ключи должны быть уникальными.

Разные типы контейнеров C++ разработаны таким образом, чтобы работа с ними происходила единообразно. Во многих случаях можно изменить тип контейнера, не изменяя часть кода, в которой происходит работа с контейнером.

Наиболее часто используемым контейнером является vector. По своей сути, вектор очень близок к массиву – они оба хранят упорядоченный набор элементов, однако вектор, в отличие от массива, может динамически увеличивать свой размер, по мере необходимости, без необходимости копировать данные и пересоздавать массив нужного размера. В последней версии Qt вектор представлен классом QList.

***Примечание:*** *В версии Qt5 и ниже вектор представлен классом QVector, а QList является реализацией связного списка. Начиная с версии Qt6 от реализации связного списка в виде QList было решено отказаться в пользу std::list. Актуальная реализация QList представляет из себя ту же реализацию, что и QVector в Qt5. Класс QVector в Qt6 сохранился как обёртка над QList с целью возможности плавного перехода на новую реализацию, поэтому при написании кода рекомендуется использовать класс QList.*

В Qt класс QList<T> является универсальной реализаций списка в Qt. Он хранит свои элементы в соседних ячейках памяти и обеспечивает быстрый доступ на основе индекса.

При создании списка необходимо в <> указать тип данных, который он должен хранить:

QList<int> integerList;

QList<QString> stringList;

При создании списка можно указать, под какое количество элементов необходимо выделить память сразу:

QList<QString> list(200);

Также можно указать значение по умолчанию, которым будут инициализированы все элементы списка:

QList<QString> list(200, "Pass");

Для доступа к элементу списка необходимо использовать оператор []:

if (list[0] == "IPTIP") // получение значения для сравнения

list[0] = "ИПТИП"; // запись значения

Однако оператор [] выполняется медленно, поэтому для чтения элементов лучше использовать метод at() – он возвращает константную ссылку, что позволяет сэкономить время, например, при проверке значений в цикле:

for (qsizetype i = 0; i < list.size(); ++i) {

if (list.at(i) == "ИПТИП")

cout << "Найдено ИПТИП на позиции:" << i << endl;

}

При необходимости список можно преобразовать в обычный массив с помощью метода data():

QList<int> list(10);

int \*data = list.data();

for (int i = 0; i < 10; i++)

data[i] = 2 \* i;

В списке можно выполнять поиск элементов. Метод contains() возвращает, есть ли в списке хотя бы одно совпадение с искомым значением, методы indexOf() и lastIndexOf() возвращают первую и последнюю позицию, соответственно, искомого элемента в списке, если элемент отсутствует – возвращают значение -1.

Для управления данными, QList предоставляет основные функции для добавления, перемещения и удаления элементов: insert(), replace(), remove(), prepend(), append().