* [Контейнер vector в STL](https://foxford.ru/lessons/32264/conspects/1)
* [Контейнер stack в STL](https://foxford.ru/lessons/32264/conspects/2)
* [Контейнер queue в STL](https://foxford.ru/lessons/32264/conspects/3)
* [Контейнер deque в STL](https://foxford.ru/lessons/32264/conspects/4)
* [Чтение и вывод в строки в языке C при помощи функций sprintf и sscanf](https://foxford.ru/lessons/32264/conspects/5)
* [Строковые потоки stringstream в С++](https://foxford.ru/lessons/32264/conspects/6)

**Контейнер vector в STL**

**Объявление вектора**

Вектор в STL - это аналог массива, контейнер, который позволяет осуществлять доступ к элементам по индексам. Вектор является шаблоном, и объявляется, как все остальные шаблоны.

Например, вектор целых чисел можно объявить так:

**vector <int> A;**

При таком создании вектор является "пустым", не содержащим элементов. Можно сразу же объявить вектор некоторого фиксированного размера, если задать конструктор, указав в конструкторе количество элементов вектора:

**vector <int> A(n);**

В этом случае значения элементов вектора будут нулевыми (а если элементами вектора являются не числа, а, например, строки или другие структуры данных, то элементы вектора будут принимать значения по умолчанию, то есть строки и вложенные вектора будут пустыми). При необходимости можно сразу же заполнить вектор некоторыми (одинаковыми), значениями, указав их в качестве второго параметра конструктора, например, для заполнения вектора значениями -1 можно его объявить так:

**vector<int> A(n, -1);**

Для использования контейнера vector необходимо подключить заголовочный файл vector:

**#include<vector>**

**Обращение к элементам вектора**

К элементам вектора можно обращаться по индексу, например, так:

**A[i]**

.

Есть и другой способ обращения к элементу вектора с индексом i: использование метода at:

**A.at(i)**

. Отличие метода at от обращения при помощи квадратных скобок в том, что при использовании метода at происходит проверка правильности индекса, и в случае выхода за границы вектора происходит ошибка исполнения. Это полезно при отладке программ.

При обращении к элементам вектора при помощи квадратных скобок корректность индекса не проверяется, и в случае выхода за границы вектора дальнейшее поведение программы может быть непредсказуемым.

Следует отметить, что работа с элементами вектора осуществляется медленнее, чем с элементами массива (даже при использовании квадратных скобок, то есть без проверки выхода за границы массива).

Помимо этого у вектора есть метод front(), возвращающий ссылку на первый элемент и метод back(), возвращающий ссылку на последний элемент вектора.

**Изменение размера вектора**

Размер вектора можно узнать при помощи универсального метода size(), возвращающего для всех контейнеров в STL их размер. Также есть метод empty(), возвращающий логическое значение (true, если вектор пустой).

Размер вектора можно изменить в любой момент, при помощи метода resize. У этого метода может быть один или два параметра. Вызов метода resize(n) изменяет размер вектора до n элементов (длина вектора может как уменьшится, так и увеличиться). Вызов метода resize(n, val) изменяет размер вектора до n элементов, и если при этом размер вектора увеличивается, то новые элементы получают значение, равное val.

Очень часто бывает полезно добавлять элементы в конец вектора по одному и удалять элементы из конца вектора по одному. Для добавления нового элемента, равного val, в конец вектора, используется метод push\_back(val). Для удаления последнего элемента вектора используется метод pop\_back() - он не возвращает значения.

Добавление элемента в конец вектора осуществляется в среднем за O(1). Это реализовано за счет того, что память для хранения элементов вектора выделяется "с запасом", то есть можно будет добавлять элементы по одному, пока не кончится запас памяти. Если запас памяти исчерпан, выделяется новая память, при этом "запас" размера вектора удваивается.

Очистить вектор можно при помощи метода clear().

**Вставка и удаление элементов в середину вектора**

Метод erase позволяет удалять из середины вектора один или несколько элементов. Этот метод работает с итераторами. Подробней про его использование можно прочитать [в документации](http://ru.cppreference.com/w/cpp/container/vector/erase).

Метод insert позволяет вставлять в середину вектора новый элемент, или несколько равных элементов, или другой вектор, или фрагмент другого вектора. Этот метод также работает с итераторами и про его использование можно прочитать [в документации](http://ru.cppreference.com/w/cpp/container/vector/insert).

Поскольку вставка и удаление элементов требуют сдвига других элементов вектора, эти операции имеют линейную сложность, то есть выполняются за время, пропорциональное длине вектора.

**Присваивание и сравнение векторов**

Содержимое одного вектора можно целиком скопировать в другой вектор при помощи операции присваивания:

**A = B**

.

Также вектора можно сравнивать на равенство и неравенство (

**A == B**

,

**A != B**

), и сравнивать их содержимое в лексикографическом порядке (

**A < B**

,

**A <= B**

,

**A > B**

,

**A >= B**

).

**Создание многомерных векторов**

Элементами вектора могут быть и другие вектора. Например, можно сделать вектор, каждый элемент которого представляет собой вектор целых чисел:

**vector < vector <int> > a;**

Тем самым, a[i] будет вектором целых чисел, а обращаться к j-му элементу вектора a[i] можно через a[i][j].

Чтобы создать двумерный вектор размером n×m можно внешний вектор объявить размером n, а затем в цикле изменить размер каждого вложенного вектора:

**vector < vector <int> > a(n);**

**for (int i = 0; i < n; ++i)**

**a[i].resize(m);**

Но можно сделать это и в одну строку, если передать вторым параметром для конструктора вектора конструктор, который создает вектор целых чисел длины m:

**vector < vector <int> > a(n, vector<int>(m));**

Заметим, что размеры вложенных векторов могут изменяться и быть различными.

Также можно создавать вектор из стеков, очередей, деков, можно создавать трехмерные векторы и т.д.

**Контейнер stack в STL**

**Контейнер stack в STL**

Стек (англ. stack) - это структура данных, в которой доступ осуществляется только к последнему добавленному элементу: новые элементы добавляются в конец стека, элементы удаляются также из конца стека, то есть при удалении элемента из стека удаляется последний добавленный элемент. В STL есть специальный контейнер stack, который реализует подобную функциональность. Для его использования необходимо подключить заголовочный файл stack:

**#include<stack>**

Объявляется стек, например, целых чисел так:

**stack <int> S;**

Стек поддерживает следущие методы:

|  |  |
| --- | --- |
| Название метода | Описание |
| size() | Возвращает размер стека |
| empty() | Возвращает true, если стек пуст, или false, если непуст |
| top() | Возвращает значение верхнего элемента в стеке |
| pop() | Удаляет верхний элемент из стека, не возвращает значение |
| push(elem) | Добавляет новый элемент elem на вершину стека |

Операции pop() или top(), вызванные для пустого стека, приведут к ошибке исполнения.

Также с объектами класса stack допустимы операции =, ==, !=, <, >, <=, >=.

Подробней о контейнере stack можно прочитать [в документации](http://ru.cppreference.com/w/cpp/container/stack).

**Контейнер queue в STL**

**Контейнер queue в STL**

Очередь (англ. queue) - это структура данных, в которой доступ осуществляется только к самому раннему добавленому элементу. Новые элементы добавляются в конец очереди, а удаляются из начала очереди.

В STL есть специальный контейнер queue, который реализует подобную функциональность. Для его использования необходимо подключить заголовочный файл queue:

**#include<queue>**

Объявляется очередь, например, целых чисел так:

**queue <int> S;**

Очередь поддерживает следущие методы:

|  |  |
| --- | --- |
| Название метода | Описание |
| size() | Возвращает размер очереди |
| empty() | Возвращает true, если очередь пуста, или false, если непуста |
| front() | Возвращает значение первого элемента в очереди |
| back() | Возвращает значение последнего элемента в очереди |
| pop() | Удаляет первый элемент из очереди, не возвращает значение |
| push(elem) | Добавляет новый элемент elem в конец очереди |

Операции pop() или top(), вызванные для пустой очереди, приведут к ошибке исполнения.

Также с объектами класса queue допустимы операции =, ==, !=, <, >, <=, >=.

Подробней о контейнере queue можно прочитать [в документации](http://ru.cppreference.com/w/cpp/container/queue).

**Контейнер deque в STL**

**Контейнер deque в STL**

Деком (англ. deque, от double-ended queue) называется структура данных, которая позволяет добавлять элементы и в конец, и в начало, а также удалять элементы из конца и из начала. Все эти операции реализованы за O(1), то есть выполняются быстро.

В STL есть специальный контейнер deque, который реализует подобную функциональность. Для его использования необходимо подключить заголовочный файл deque:

**#include<deque>**

Объявляется дек, например, целых чисел так:

**deque <int> D;**

Дек поддерживает следущие методы:

|  |  |
| --- | --- |
| Название метода | Описание |
| size() | Возвращает размер дека |
| empty() | Возвращает true, если дек пуста, или false, если непуст |
| clear() | Очищает дек |
| front() | Возвращает значение первого элемента в деке |
| back() | Возвращает значение последнего элемента в деке |
| pop\_front() | Удаляет первый элемент из дека, не возвращает значение |
| pop\_back() | Удаляет последний элемент из дека, не возвращает значение |
| push\_front(elem) | Добавляет новый элемент elem в начало дека |
| push\_back(elem) | Добавляет новый элемент elem в конец дека |

У дека есть много общего с векторами. Например, к элементам дека можно обращаться по индексу, при помощи [i] или метода at(i). Эти обращения выполняются быстро - реализация дека насколько хитра, что можно быстро добавлять элементы в начало и в конец, а также быстро обращаться к элементу по его номеру. У дека, как и у вектора, есть методы [erase](http://ru.cppreference.com/w/cpp/container/deque/erase), [insert](http://ru.cppreference.com/w/cpp/container/deque/insert) и [resize](http://ru.cppreference.com/w/cpp/container/deque/resize), но они работают за линейное время.

Также с объектами класса deque допустимы операции =, ==, !=, <, >, <=, >=.

Подробней о контейнере deque можно прочитать [в документации](http://ru.cppreference.com/w/cpp/container/deque).

#### Чтение и вывод в строки в языке C при помощи функций sprintf и sscanf

В языке C есть функции sscanf и sprintf, позволяющие осуществлять форматный ввод-вывод с использованием строк. Например, пусть строка s содержит два целых числа, записанных через пробел и необходимо извлечь значения этих чисел и записать их в переменные a и b. Это делается при помощи функции sscanf, у которой первый параметр - строка, из которой читаются данные (строка языка C, то есть массив символов или указатель типа char \*), а остальные параметры - как у функции scanf, то есть форматная строка и адреса переменных для записи данных. Пример:

**sscanf(s, "%d%d", &a, &b);**

Функция sscanf реализует все возможности чтения данных, которые есть в функция scanf и fscanf.  
Аналогично для вывода данных не в файл, а в строку, используется функция sprintf. Её первый параметр - строка, остальные параметры - как для функции printf. Пример:

**sprintf(s, "%d %d", a, b);**

Эти функции объявлены в заголовочном файле stdio.h.

#### Строковые потоки stringstream в С++

Аналогично функциям sscanf и sprintf в языке C, в языке C++ есть возможность считывать данные из строк (как будто бы из файла) и выводить данные в строки.

Для этого используются описанные в заголовочном файле sstream объекты istringstream (поток для чтения из строки) и ostringstream(поток для вывода из строки).

Приведем пример использования этих потоков. Пусть в строке s записано два числа через пробел, мы считаем их при используя объект istringstream, связанный со строкой, затем выведем в объект ostringstream эти же числа, увеличенные на 1 и скопируем результат в строку s.

**string s = "123 456";**

**istringstream ss(s);**

**int a, b;**

**ss >> a >> b;**

**ostringstream os;**

**os << a + 1 << " " << b + 1;**

**s = os.str();**