Лекция 10.

Контейнеры.

Строка.

```
// #include <string>

std::string s;
```

В ней есть удобные методы:

- `s.size();`
- `s.capacity();`
- `s.begin();`
- `s.end();`
- `s.push_back();`
- `s.pop_back();`

Каждая строка заканчивается на `'\0'`. Это было сделано для совместимости с простыми сишными строками, чтобы функции написанные для тех строк, работали и для плюсовых. Если при изменении она увеличивается, то она окпируется на динамическую память.

Короткие строки хранятся на стеке, потому что само создание в динамической памяти заканчивает больше времени.

String View.

Допустим мы хотим вывести часть строк. Можем сделать так

```
void foo(const std::string& s) {
  auto begin = s.begin() + 5;
  auto end = begin + 10;

  while (begin != end) {
    std::cout << *begin++;
  }
}
```

Что немного долго и неудобно. Есть еще метод `s.substr(beginn, end)`, но он создает копию строки.

Тогда на помощь приходит класс `std::string_view`. Он не владеет строкой, а просто ссылается на часть нее.

```
std::string str{"abcdefghijklmnopqrstuvwxyz"};
std::string_view sv{str};
```

Есть опасность: поскольку этот объект ссылается на адрес в пмяти, то если эта строка в какой-то момент будет удалена, то `std::string_view` будет ссылаеться туда, где до этого была строк. Чтение этого приведет к *undefined behaviour.*

Еще одна опасность: если строка переносится в динаическую память и `std::string_view` будет ссылаться все ещ ена старое место.

Поэтому не надо изменять строку, пока сущесвтует `std::string_view`, который ссылается на нее.

Дек (deque).

```
// #include <deque>

std::deque<int> d;
```

В памяти он выглядит как несколько блоков массивов по 256 элемнетов, связанных между собой.
`[100] <--> [256] <--> [256] <--> [256] <--> [256] <--> [23]`
Таким образом сохраняется обращение по индексу элемнента, но будет тратиться время на поиск блока.
Между блоками и внутри блоков элементы не перемещаются.
Он лучше по памяти, чем вектор, но это ценой траты по времени.

Очередь (queue).

```
// #include <queue>

std::queue<int> q;
```

Это просто обертка вокруг дека — адаптор.

Стек (stack).

```
// #include <queue>

std::queue<int> q;
```

Это тоже адаптор для дека.

Про дек, стек и очередь.
Тогда справедливый вопрос: почему не использовать всегда дек?
Ответ прост: надо брать ровно то, что нужно от структуры. Это сделает код логинчным, а остальным, читающим код, будет понятнее, что будет происходить в программе.