Лекция 9.

Контейнеры.

Векторы.

Контейнер, поддерживающий свойства массива, с возможностью вставки в конец и изменением размеров.

```
// #include <vector>

std::vector<int> vec1;
vec1.push_back(123);
vec.resize(10);
vec[2] = 2;
```

Есть методы `.size()`, `.enc()`, `.begin()`,

Можем написать свой

```
class VectorInt {
public:
  VectorInt() {

  }
  ~VectorInt() {
    delete[] begin_;
  }

  size_t size() const {
    return size_;
  }

  int operator[](size_t idx) const {
    return begin_[idx];
  }

private:
  int* begin_ = nullptr;
  size_t size_ = 0;
}
```

Итератор

Range based for

Чтобы сократить перебор символов в какой-то одной коллекции, придумали такую структуру

```
for(const char& c : arr) {

}
```

под капотом рабоатет она примерно так:

```
{
  const char (&__range1)[10] = arr;
  const char* __begin1 = __range1;
  const char* __end1 = __range1 + 10L;
  for(; __begin1 != __end1; ++__begin1) {
    const char & c = *__begin1;
  }
}
```

```
std::array<int, 10> arr;
arr.size();
```

Это просто обертка над массивом. Он удобнее в чтении (в смысле кодстайла).
Есть методы `.fill(n)` — заполнить значениями `n`; `.size()` — узнать размер (он всегда фиксированный).

```
class NodeIterator {
public:
  NodeIterator& operator++() {};

  // ...

}

class List {
public:
  List()_ {
    fake_ = new Node{0, nullptr, nullptr};
    fake_->next = fake_->prev;
    fake_->next = fake_;
  }

  void erase(Node* node) {
    node->prev->next = node->next;
    node->next->prev = node->prev;
    delete node;
  }

  void push_back(int value) {
    Node* next = new Node{value, nullptr, nullptr};
    next->prev = last_;
    next->next = nullptr;
    last_ = next;
  }
private:
  Node* fake_;
}
```