`std::initializer_list` (C++11)

Мотивация

Известно, что для стандартных типов, а также для классов без приватных полей работает *агрегатная инициализация*:

```
int x{1};
float y{};  // инициализация нулем
long array[5]{1, 2, 3};  // [1, 2, 3, 0, 0]
```

```
struct S {
  int x;
  double y;
  int arr[2];
};

S s{1, 2.5, 3};  // x = 1, y = 2.5, arr[0] = 3, arr[1] = 0
```

Мотивация

Некоторые классы стандартной библиотеки удовлетворяют этим условиям и поэтому автоматически поддерживают инициализацию списком:

```
std::array<int, 3> a{1, 2};  // [1, 2, 0]

std::pair<int, double> p{1, 0.5};  // [1, 0.5]
```

Мотивация

А как быть с другими контейнерами? Вряд ли они содержат только открытые поля.

```
std::vector<int> v{1, 2, 3};

std::list<double> l{1, 3, 5};

std::unordered_map<int, std::string> m{{1, "one"}, {2, "two"}, {3, "three"}};
```

Но со своими классами, это точно не сработает

```
template <class T>
class Stack { /* представляем реализацию на динамическом массиве */ };

Stack<int> s{1, 2, 3};
```

```
error: no matching function for call to 'Stack<int>::Stack(<brace-enclosed initializer list>)'
```

`std::initializer_list`

`std::initializer_list` - шаблонный класс, представляющий легковесную обертку над константным массивом объектов.

`std::initializer_list`

Список объектов в фигурных скобках автоматически преобразуются в `std::initializer_list` в следующих ситуациях:

- При создании объекта с помощью объявления `auto`:

```
auto l = {1, 2, 3};  // type(l) == std::initializer_list<int>
```

- При создании объекта класса, поддерживающего конструктор от `std::initializer_list`:

```
Stack<T>::Stack(std::initializer_list<T>);
Stack<int> s{1, 2, 3};  // Ok
```

- При передаче в функцию, принимающую в качестве аргумента `std::initializer_list`:

```
void f(std::initializer_list<int>);
f({1, 2, 3});
```

`std::initializer_list`

Содержит всего 3 метода:

- `size()`, `begin()`, `end()`

Категория итератора - random access (contiguous, начиная с C++20)

Типы-члены:

- `value_type`, `reference`, `const_reference`, `size_type`, `iterator`, `const_iterator`

Пример: реализации конструктора

```
template <class T>
Stack<T>::Stack(std::initializer_list<T> lst)
  : buffer_(new T[lst.size()])
  , size_(0)
  , capacity_(lst.size()) {

  for (auto&& obj : lst) {
    Push(obj);
  }
}
```