

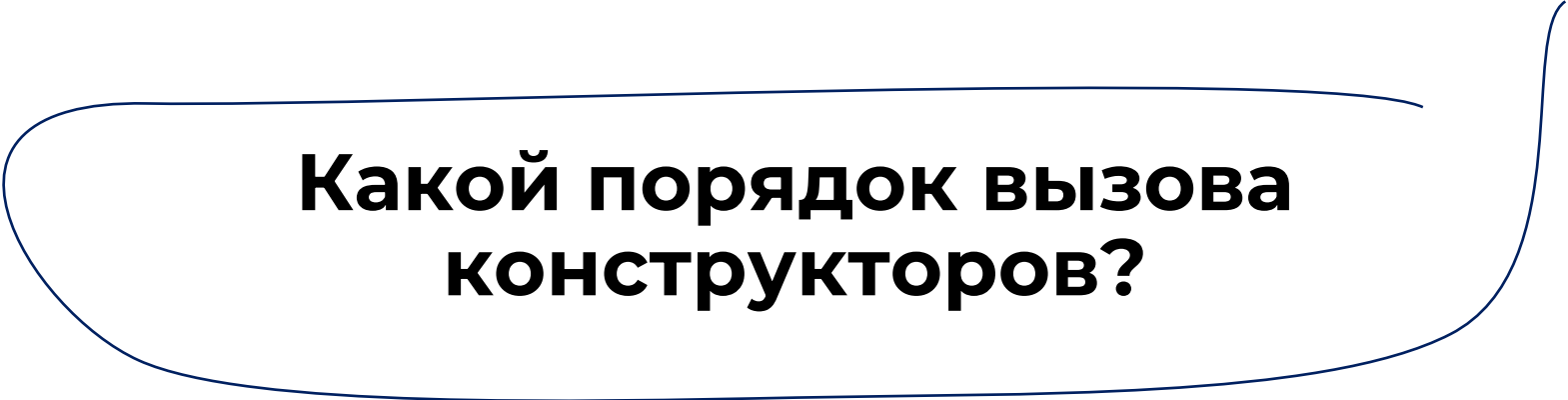
C++

Урок 18

A hand-drawn blue oval frame with a double-line border, centered on the page. The word "Повторение" is written inside this frame in a bold, black, sans-serif font.

Повторение

**Для чего нужны виртуальные
деструкторы?**



**Какой порядок вызова
конструкторов?**

**Какой порядок вызова
деструкторов?**

3 модуль



Что мы уже знаем?



Много чего..



**Осталось понять как со всем
этим работать...**

Контейнеры STL

STL

STL (Standard Template Library – Стандартная Библиотека Шаблонов). Библиотека STL содержит пять основных видов компонентов:

STL

STL (Standard Template Library – Стандартная Библиотека Шаблонов). Библиотека STL содержит пять основных видов компонентов:

- **Контейнер (container)**: управляет набором объектов в памяти.

STL

STL (Standard Template Library – Стандартная Библиотека Шаблонов). Библиотека STL содержит пять основных видов компонентов:

- **Контейнер (container)**: управляет набором объектов в памяти.
- **Итератор (iterator)**: обеспечивает для алгоритма средство доступа к содержимому контейнера.

STL

STL (Standard Template Library – Стандартная Библиотека Шаблонов). Библиотека STL содержит пять основных видов компонентов:

- **Контейнер (container)**: управляет набором объектов в памяти.
- **Итератор (iterator)**: обеспечивает для алгоритма средство доступа к содержимому контейнера.
- **Алгоритм (algorithm)**: определяет вычислительную процедуру.

STL

STL (Standard Template Library – Стандартная Библиотека Шаблонов). Библиотека STL содержит пять основных видов компонентов:

- **Контейнер (container)**: управляет набором объектов в памяти.
- **Итератор (iterator)**: обеспечивает для алгоритма средство доступа к содержимому контейнера.
- **Алгоритм (algorithm)**: определяет вычислительную процедуру.
- **Функтор (function object)**: инкапсулирует функцию в объекте для использования другими компонентами.

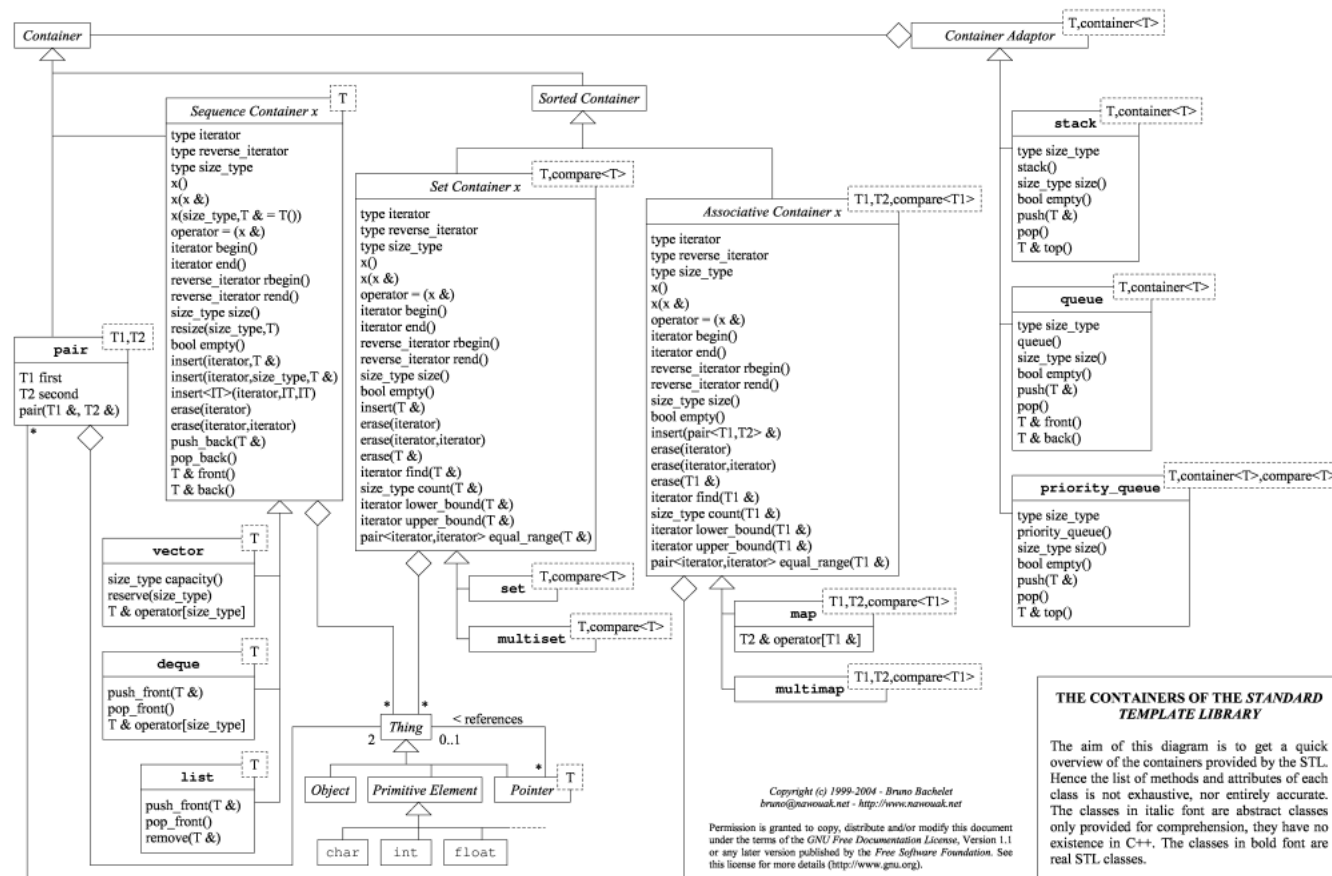
STL

STL (Standard Template Library – Стандартная Библиотека Шаблонов). Библиотека STL содержит пять основных видов компонентов:

- **Контейнер (container)**: управляет набором объектов в памяти.
- **Итератор (iterator)**: обеспечивает для алгоритма средство доступа к содержимому контейнера.
- **Алгоритм (algorithm)**: определяет вычислительную процедуру.
- **Функтор (function object)**: инкапсулирует функцию в объекте для использования другими компонентами.
- **Адаптер (adaptor)**: адаптирует компонент для обеспечения различного интерфейса.

STL

Контейнеры есть стандартные структуры данных, такие как список (**list**), вектор (**vector**), словарь (**map**) и многие другие.



STL

Доступ к элементам контейнера осуществляется через специальные объекты — **итераторы**.

Итератор, указывающий на первый элемент **iterator**
begin();

Итератор, указывающий за последний элемент **iterator**
end();

Итератор — объект, предоставляющий доступ к элементам контейнера и позволяющий их перебирать.

В **первых** реализациях стандартной библиотеки C++ итератор реализовывался как **указатель** на элемент контейнера. В современных реализациях это **класс, инкапсулирующий указатель** на объект контейнера.

Основные требования к итераторам — наличие операторов разыменования и инкремента.

Алгоритм — последовательность действий, которые приводят к результату.

Реализация алгоритмов происходит через **шаблонные ф-ии**. (для возможности обработки разных типов параметров).

Параметры у этих ф-ий являются **диапазоны**.

Функтор — функциональный объект, представляющий собой конструкцию, позволяющую использовать объект класса как функцию.

Для определения **функтора** достаточно описать класс, в котором переопределена операция **()**.

STL

```
#include <iostream>
#include <vector>
using namespace std;

class summator : private vector<int>
{
public:
    summator(const vector<int> &ini)
    {
        for (auto &x : ini)
            this->push_back(x);
    }

    int operator()(bool even)
    {
        int sum = 0;
        auto i = begin();
        if (even)
            i++;
        while (i < end())
        {
            sum += *i++;
            if (i == end())
                break;
            i++;
        }
        return sum;
    }
};

int main()
{
    summator sums(vector<int>({1, 2, 3, 4, 5, 6, 7, 8, 9, 10}));

    cout << "сумма чётных = " << sums(true) << endl
          << "сумма нечётных = " << sums(false) << endl;
}
```

Пример использование функтора

Плюсы Функтора:

- Его можно параметризовать при создании объекта (перед вызовом) используя конструктор объекта с параметрами.
- Может создаваться временный объект исключительно на время выполнения функционального вызова.

STL

```
#include <iostream>
using namespace std;

class calculate
{
    char op;

public:
    calculate(char op) : op(op) {}
    int operator()(int op1, int op2)
    {
        switch (op)
        {
            case '+':
                return op1 + op2;
            case '-':
                return op1 - op2;
            case '*':
                return op1 * op2;
            case '/':
                return op1 / op2;
            case '%':
                return op1 % op2;
            case '^':
            {
                int ret = op1;
                while (op2-- > 1)
                    ret *= op1;
                return ret;
            }
            default:
                cout << "неразрешённая операция" << endl;
                return 0;
        }
    }
};

int main()
{
    char oper;
    int op1, op2;
    cout << "выражение для вычисления (<op1><знак><op2>): " << endl;
    cin >> op1 >> oper >> op2;
    cout << op1 << ' ' << oper << ' ' << op2 << " = "
        << calculate(oper)(op1, op2) << endl;

    return 0;
}
```

Пример использование функтора

Адаптеры — адаптация уже существующих понятий библиотеки под конкретные, часто используемые цели.

Часто такая адаптация делается посредством **ограничения функциональности базового понятия** под запросы адаптера.

В библиотеке представлены адаптеры контейнеров, итераторов и функций.



Пары

Пары

Пара - переменная, позволяющая хранить в себе два значения.



Пары

Синтаксис:

```
pair <string, double> Izmerenia;
```

Переменная `Izmerenia` будет хранить в себе значение типа **string** и значение типа **double**.

Пары

Синтаксис:

```
pair <string, double> Izmerenia;
```

```
pair<string, double> Izmerenia;
```

```
Izmerenia.first = "Alex"; //доступ к первому значению
```

```
Izmerenia.second = 178.12; //доступ ко второму значению (полю)
```

Пары

Синтаксис:

```
pair<string, double> Izmerenia;  
Izmerenia = make_pair("Alex", 178.12); //присвоить сразу два значения
```

Задача

Задача: создайте пару translator, где будет занесено слово и его перевод на английском языке.

А что если мы хотим много пар?

**В какую структуру их можно
занести?**

Пары

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main()
{
    vector<pair<string, string> > vocabulary;
    string str1, str2;
    int cnt;
    cin >> cnt;
    for (int i = 0; i < cnt; i++)
    {
        cin >> str1 >> str2;
        vocabulary.push_back(make_pair(str1, str2));
    }

    for (auto &it : vocabulary)
    {
        cout << it.first << " " << it.second << endl;
    }

    // Задача: создайте словарь пар translator, где будут занесены слова и их переводы
    на английском языке.
}
```

Задача

Задача: создайте словарь пары translator, где будут занесены слова и их переводы на английском языке.



Словари



Что такое словарь?

Как вы им пользовались?

Словари

Словарь - сборник слов в алфавитном порядке, с пояснениями, толкованиями или с переводом на другой язык.



Словари

Словарь - сборник слов в алфавитном порядке, с пояснениями, толкованиями или с переводом на другой язык.

Словарь C++ (map) – это ассоциативный контейнер, который работает по принципу — [ключ — значение].

Словари

Словари часто называют также **ассоциативными массивами** или **отображениями**.

Словарь построен на основе пар значений, первое из которых представляет собой ключ для идентификации элемента, а второе — собственно элемент

Словари

Примеры ассоциативных контейнеров:

- `map` – словарь уникальных ключей,
- `multimap` – словарь ключей с дубликатами,
- `set` - множество,
- `multiset` – мультимножество,
- `bitset` – битовое множество (набор битов).

Easy Peasy



It comes from a 1970's [british TV](#) commercial for [Lemon Squeezy](#) detergent. They were with a little girl who points out dirty greasy dishes to an adult (mom or relative) and then this adult produces Lemon Squeezy and they clean the dishes quickly. At the end of the commercial the girl says "[Easy Peasy Lemon Squeezy](#)".

Today it is a silly way to state something was or will be very easy.

I will be [in an out](#), [easy peasy](#).

Can you open this [jar of pickles](#)? Sure thing easy peasy.

by [Creepy Gnome](#) August 2, 2009

**Приведите свой пример
ассоциативного массива**

Словари

```
#include <map>
```

- Подключаем библиотеку

Создание словаря

Словари

```
#include <map>
```

- Подключаем библиотеку

```
map<тип ключа, тип значения> название;
```

Создание словаря

Словари

```
#include <map>
```

- Подключаем библиотеку

```
map<тип ключа, тип значения> название;
```

```
int main()
{
    map<string, string> urbanDict;    //создали массив с ключем и значением типа string
    map<string, double> medicalMeasure;
    //создали массив с ключем типа string, а значением типа double
}
```

Создание словаря

Словари

```
map<string, double> medicalMeasure = {"Marti", 14.5},  
                                       {"Gloria", 10.15},  
                                       {"Melmon", 30.15}};
```

Инициализация словаря в момент
объявления

Словари

```
int main()
{
    map<string, double> medicalMeasure;

    medicalMeasure["Marti"] = 14.5;
    medicalMeasure["Gloria"] = 10.15;
    medicalMeasure["Melmon"] = 30.15;

    cout << medicalMeasure["Melmon"] << endl; // 30.15
}
```

Добавление элемента

Словари

```
int main()
{
    map<string, double> medicalMeasure;

    medicalMeasure["Marti"] = 14.5;
    medicalMeasure["Gloria"] = 10.15;
    medicalMeasure["Melmon"] = 30.15;
    medicalMeasure["Gloria"] = 25.15;

    cout << medicalMeasure["Gloria"] << endl;
}
```

Что будет выведено?

Словари

```
int main()
{
    map<string, double> medicalMeasure;

    medicalMeasure["Marti"] = 14.5;
    medicalMeasure["Gloria"] = 10.15;
    medicalMeasure["Melmon"] = 30.15;

    // 1 способ
    for (auto &it : medicalMeasure)
    {
        cout << it.first << " " << it.second << endl;
    }

    // 2 способ
    for (auto it = medicalMeasure.begin(); it != medicalMeasure.end(); it++)
    {
        cout << it->first << " " << it->second << endl;
    }

    // 3 способ
    for (auto it = medicalMeasure.begin(); it != medicalMeasure.end(); it++)
    {
        cout << (*it).first << " " << (*it).second << endl;
    }
}
```

Вывод эл-тов

Задача

Задача: Реализуйте словарь buildings, в котором содержится информация о самых высоких зданиях в мире.

Входные данные: ввод информации в словарь с **клавиатуры**.

Выходные данные: высота здания по его названию.

Задача

Задача: Реализуйте словарь buildings, в котором содержится информация о самых высоких зданиях в мире.

Входные данные: название строения.

Выходные данные: высота строения.

Словари

Ассоциативные массивы относятся к STL контейнерам.
Также в STL есть vector.

Словари

Ассоциативные массивы относятся к STL контейнерам.
Также в STL есть vector.

Какие методы мы использовали в vector?

Словари

Ассоциативные массивы относятся к STL контейнерам.
Также в STL есть vector.

Какие методы мы использовали в vector?

- `find(ключ)`
- `count (ключ)`
- `erase (ключ)`
- `size()`
- `clear()`
- `empty()`
- `swap(контейнер)`

Словари

Ассоциативные массивы относятся к STL контейнерам. Также в STL есть vector.

Какие методы мы использовали в vector?

- `find(ключ)` – возвращает итератор на найденный элемент. Если не нашел эл-т, то итератор на конец.
- `count (ключ)` - возвращает кол-во эл-тов по заданному ключу.
- `erase (ключ)`
- `size()`
- `clear()`
- `empty()`
- `swap(контейнер)`

[map документация](#)

Словари

Ассоциативные массивы относятся к STL контейнерам. Также в STL есть vector.

Какие методы мы использовали в vector?

- `find(ключ)` – возвращает итератор на найденный элемент. Если не нашел эл-т, то итератор на конец.
- `count (ключ)` - возвращает кол-во эл-тов по заданному ключу. В `map` вернет либо 0, либо 1.
- `erase (ключ)` – удаляет эл-т по заданному ключу.
- `size()` – кол-во эл-тов в словаре.
- `clear()` – удаление всех эл-тов из словаря.
- `empty()` – определяет пустой ли словарь.
- `swap(контейнер)` – меняет элементы между двумя сопоставлениями

Задача

Задача: Реализуйте словарь buildings, в котором содержится информация о самых высоких зданиях в мире.

Входные данные: ввод информации в словарь с **клавиатуры**.

Выходные данные: высота здания по его названию.