Имена и пространства имён — конспект темы

Зови меня по имени

Имя — это идентификатор, оно:

- состоит из латинских букв, цифр и знака подчёркивания;
- не может начинаться с цифры;
- может состоять даже из одного символа, например, знака подчёркивания: `_` — допустимое имя для сущности в C++.

Имён нет у макросов, определённых через `#define` или параметры компиляции. Макросы обрабатываются во время препроцессирования, и компилятор получает код, где никаких макросов уже нет.

Безымянными могут быть структура, класс и параметр функции или шаблона. Ещё можно не писать имена параметров при объявлении функции. Переменные и функции безымянными быть не могут.

Имя бывает составным — разбитым на лексемы. Так бывает с операциями. Операция — это обычная функция, имя которой состоит из ключевого слова `operator` и знака операции. Другой пример составного имени — имя деструктора класса, которое содержит знак `~`.

JSON (от англ. JavaScript Object Notation) — формат структурирования данных. Поддерживает словари, массивы, строки, булевы значения, целые числа и числа с плавающей запятой, специальное значение `null`.

XML (от англ. eXtensible Markup Language) — формат структурирования данных, который использует понятие тегов.

Пространства имён спешат на помощь

В C++ разные сущности кода могут иметь одинаковые имена.

Когда вы употребляете имя нескольких сущностей, имена начинают конкурировать друг с другом, и компилятор выбирает находящееся в наиболее близкой объемлющей области видимости.

Если несколько имён расположены одинаково по отношению к месту использования, возникает конфликт: компилятор не сможет выбрать между ними и выдаст ошибку.

Спасает от этого **пространство имён**. Чтобы его использовать, ннужно все определения и объявления в cpp и h-файлах заключить в конструкцию `namespace ...`:

```
//файл json.h

#pragma once

// #include ...

// начинаем пространство имён
namespace json {

// Стайлгайд Практикума запрещает
// увеличивать отступ внутри конструкции namespace {...}.
// Поэтому пишем с начала строки без отступа
class Node {
    // ...
};

class Document {
    // ...
};

Document Load(std::istream& input);

template <typename T>
inline T Node::AttributeValue(const std::string& name) const {
    // ...
}

// нужно закрыть namespace
}
```

Так будет выглядеть файл json.cpp:

```
#include "json.h"

using namespace std;

// начинаем пространство имён
namespace json {

Node::Node(vector<Node> array) : as_array(move(array)) {
}

// определения других конструкторов и методов класса Node

Node LoadNode(istream& input);

Node LoadArray(istream& input) {
    // ...
}

// определения других вспомогательных функций

Document Load(istream& input) {
    // ...
}
```

```
// нужно закрыть namespace
}
```

Теперь включение обоих заголовочных файлов в `main.cpp` не вызовет ошибок. Эта операция оказалась проще добавления префикса `JSON` или `XML` ко всем именам библиотек — не пришлось редактировать каждое имя. Но вне пространства имён, например в файле `main.cpp`, имя `Document` стало недоступно.

Нужно сообщить компилятору, документ из какого пространства имеется в виду. Для этого добавьте перед именем **квалификацию** — явное указание на то, в каком пространстве имён его искать.

```
vector<Spending> LoadFromXml(istream& input) {
    xml::Document doc = xml::Load(input);
    vector<Spending> result;
    for (const xml::Node& node : doc.GetRoot().Children()) {
        result.push_back({node.AttributeValue<string>("category"s), node.AttributeValue<int>("amount"s)});
    }
    return result;
}

vector<Spending> LoadFromJson(istream& input) {
    json::Document doc = json::Load(input);
    vector<Spending> result;
    for (const json::Node& node : doc.GetRoot().AsArray()) {
        result.push_back({node.AsMap().at("category"s).AsString(), node.AsMap().at("amount"s).AsInt()});
    }
    return result;
}
```

К именам библиотеки JSON добавлен префикс `json::`, а к именам XML — префикс `xml::`. Имя с квалификацией `json::Document` выглядит структурировано и его проще читать, чем `JSONDocument`. Внутри библиотеки нет необходимости каждый раз писать префикс.

Синтаксис пространств имён

Пространства имён создаются словом `namespace`. Одно и то же пространство имён может определяться сколько угодно раз в разных частях программы.

Пространства имён могут быть вложены друг в друга. Например, принято выделять вспомогательные функции и классы библиотеки во вложенное пространство имён `detail`.

Функции и методы, объявленные внутри пространства имён, можно определять:

1. внутри конструкции `namespace`,
2. вне конструкции `namespace`, явно указывая квалификацию каждого имени.

Когда применяется первый способ, определение целиком помещается в конструкцию `namespace` и квалифицировать имена того же пространства не нужно: при поиске сущности по имени компилятор обязательно будет принимать во внимание все имена этого пространства.

Второй способ позволяет определить объявленную внутри конструкции `namespace` функцию или метод, не прибегая к этой конструкции второй раз.

В отличие от определений, объявления функций и классов можно делать только первым способом — внутри конструкции `namespace`.

Сокращаем с using

Квалификация усложняет код. Чтобы сократить написание и упростить чтение, можно применить декларацию `using`:

```
using ini::Section;
```

Она позволяет убрать квалификацию у имён стандартной библиотеки и вместо `std::map<std::string, std::pair<std::string, std::optional<std::string_view>>>` написать `map<string, pair<string, optional<string_view>>>`.

`using`-декларация распространяется только на то имя, которое указано в ней явно, и только на ту область видимости, где она объявлена. `using` можно написать и вне областей видимости, например в начале файла. Тогда имя станет глобальным — будет распространяться на весь файл.

Использовать директиву `using namespace` нужно только в том месте, где она правда нужна, максимально сужая область её действия.

Особенно неприятно, когда к

Когда конструкция `using namespace` встречается в h-файле и её действие не ограничено областью видимости, она распространяется на все файлы, использующие этот заголовочный файл.

Если написать `using namespace my_beautiful_namespace` внутри другого пространства имён, оно примет все имена пространства `my_beautiful_namespace`:

О различиях декларации `using` и директивы `using namespace` читайте в документации.

Практические рекомендации

Используйте пространства имён:

- Если разрабатываете большую программу. В ней точно много имён и весьма вероятны конфликты.
- Если разрабатываете библиотеку. Неизвестно, в какой программе она будет использоваться, и какие имена в ней будут.
- Если в вашей программе возникают классы с префиксом или постфиксом. Например `AnimalFunnyPet`, `AnimalNastyRat`, `AnimalLucidLynx`, `AnimalNattyNarwhal`. В этом случае уместен

`namespace`, ведь префикс `Animal::` лучше, чем просто `Animal`.

В одно пространство имён объединяйте общие по смыслу классы и функции.

Не определяйте несколько пространств имён в одном файле.

Будьте осторожны при использовании `using namespace`. В h-файле неограниченный `using namespace` недопустим.