Программирование на языке C++
Лекция 7

Пространства имён

Александр Смаль

Пространства имён

Пространства имён (namespaces) — это способ разграничения областей видимости имён в C++.

Пространства имён

Пространства имён (namespaces) — это способ разграничения областей видимости имён в C++.

Имена в C++:

1. имена переменных и констант,
2. имена функций,
3. имена структур и классов,
4. имена шаблонов,
5. синонимы типов (typedef-ы),
6. enum-ы и union-ы,
7. имена пространств имён.

Примеры

В C для избежания конфликта имён используются префиксы.
К примеру, имена в библиотеке Expat начинаются с XML_.

```
struct XML_Parser;
int XML_GetCurrentLineNumber(XML_Parser * parser);
```

Примеры

В C для избежания конфликта имён используются префиксы.
К примеру, имена в библиотеке Expat начинаются с XML_.

```
struct XML_Parser;
int XML_GetCurrentLineNumber(XML_Parser * parser);
```

В C++ это можно было бы записать так:

```
namespace XML {
    struct Parser;
    int GetCurrentLineNumber(Parser * parser);
}
```

Тогда полные имена структуры и функции будут соответственно:
`XML::Parser` и `XML::GetCurrentLineNumber`.

Описание пространства имён

1. Пространства имён могут быть вложенными:

```
namespace items { namespace food {
    struct Fruit {...};
}}
items::food::Fruit apple("Apple");
```

Описание пространства имён

1. Пространства имён могут быть вложенными:

```
namespace items { namespace food {
    struct Fruit {...};
}}
items::food::Fruit apple("Apple");
```

2. Определение пространств имён можно разделять:

```
namespace weapons { struct Bow { ... }; }
namespace items   {
    struct Scroll { ... };
    struct Artefact  { ... };
}
namespace weapons { struct Sword { ... }; }
```

Описание пространства имён

1. Пространства имён могут быть вложенными:

```
namespace items { namespace food {
    struct Fruit {...};
}}
items::food::Fruit apple("Apple");
```

2. Определение пространств имён можно разделять:

```
namespace weapons { struct Bow { ... }; }
namespace items   {
    struct Scroll { ... };
    struct Artefact  { ... };
}
namespace weapons { struct Sword { ... }; }
```

3. Классы и структуры определяют одноимённый namespace.

Доступ к именам

1. Внутри того же `namespace` все имена доступны напрямую.

Доступ к именам

1. Внутри того же `namespace` все имена доступны напрямую.
2. `NS::` позволяет обратиться внутрь пространства имён NS.

```
namespace NS { int foo() { return 0; } }
int i = NS::foo();
```

Доступ к именам

1. Внутри того же `namespace` все имена доступны напрямую.
2. `NS::` позволяет обратиться внутрь пространства имён NS.

```
namespace NS { int foo() { return 0; } }
int i = NS::foo();
```

3. Оператор `::` позволяет обратиться к *глобальному пространству имён*.

```
struct Dictionary {...};

namespace items
{
    struct Dictionary {...};

    ::Dictionary globalDictionary;
}
```

Поиск имён

Поиск имён — это процесс разрешения имени.

1. Если такое имя есть в текущем `namespace`
   - выдать *все* одноимённые сущности в текущем `namespace`.
   - завершить поиск.
2. Если текущий `namespace` — глобальный
   - завершить поиск и выдать ошибку.
3. Текущий `namespace` $\leftarrow$ родительский `namespace`.
4. Перейти на шаг 1.

Поиск имён

```
int foo(int i) { return 1; }

namespace ru
{
    int foo(float f) { return 2; }

    int foo(double a, double b) { return 3; }

    namespace spb {
        int global = foo(5);
    }
}
```

Важно: поиск продолжается до первого совпадения.
В перегрузке участвуют только найденные к этому моменту функции.

Ключевое слово using

Существуют два различных использования слова using.

```
namespace ru
{
    namespace msk {
        int foo(int i) { return 1; }
        int bar(int i) { return -1; }
    }

    using namespace msk; // все имена из msk
    using msk::foo;      // только msk::foo

    int foo(float f) { return 2; }
    int foo(double a, double b) { return 3; }

    namespace spb {
        int global = foo(5);
    }
}
```

Поиск Кёнига

```
namespace cg {
    struct Vector2 {...};
    Vector2 operator+(Vector2 a, Vector2 const& b);
}
```

Поиск Кёнига

```
namespace cg {
    struct Vector2 {...};
    Vector2 operator+(Vector2 a, Vector2 const& b);
}
```

```
    cg::Vector2 a(1,2);
    cg::Vector2 b(3,4);
    b = a + b; // эквивалентно: b = operator+(a, b)
    b = cg::operator+(a, b); // OK
```

Поиск Кёнига

```
namespace cg {
    struct Vector2 {...};
    Vector2 operator+(Vector2 a, Vector2 const& b);
}
```

```
    cg::Vector2 a(1,2);
    cg::Vector2 b(3,4);
    b = a + b; // эквивалентно: b = operator+(a, b)
    b = cg::operator+(a, b); // OK
```

Argument-dependent name lookup (ADL, Поиск Кёнига)

При поиске имени функции на первой фазе рассматриваются имена из текущего пространства имён *и пространств имён, к которым принадлежат аргументы функции*.

Безымянный namespace

Пространство имён с гарантированно уникальным именем.

```
namespace { // безымянный namespace
    struct Test { std::string name; };
}
```

Это эквивалентно:

```
namespace $GeneratedName$ {
    struct Test { std::string name; };
}
using namespace $GeneratedName$;
```

Безымянные пространства имён — замена для `static`.

Заключение

Заключение

1. Используйте пространства имён для исключения конфликта имён.

Заключение

1. Используйте пространства имён для исключения конфликта имён.
2. Помните, что поиск имён прекращается после первого совпадения. Используйте `using` и полные имена.

Заключение

1. Используйте пространства имён для исключения конфликта имён.
2. Помните, что поиск имён прекращается после первого совпадения. Используйте `using` и полные имена.
3. Не используйте `using namespace` в заголовочных файлах.

Заключение

1. Используйте пространства имён для исключения конфликта имён.
2. Помните, что поиск имён прекращается после первого совпадения. Используйте `using` и полные имена.
3. Не используйте `using namespace` в заголовочных файлах.
4. Всегда определяйте операторы в том же пространстве имён, что и типы, для которых они определены.

Заключение

1. Используйте пространства имён для исключения конфликта имён.
2. Помните, что поиск имён прекращается после первого совпадения. Используйте `using` и полные имена.
3. Не используйте `using namespace` в заголовочных файлах.
4. Всегда определяйте операторы в том же пространстве имён, что и типы, для которых они определены.
5. Используйте безымянные пространства имён для маленьких локальных классов и как замену слова `static`.

Заключение

1. Используйте пространства имён для исключения конфликта имён.
2. Помните, что поиск имён прекращается после первого совпадения. Используйте `using` и полные имена.
3. Не используйте `using namespace` в заголовочных файлах.
4. Всегда определяйте операторы в том же пространстве имён, что и типы, для которых они определены.
5. Используйте безымянные пространства имён для маленьких локальных классов и как замену слова `static`.
6. Для длинных имён `namespace`-ов используйте синонимы:

```
namespace csccpp17 = ru::spb::csc::cpp17;
```