Пространства имен

6 июля 2017 г.

Пространство имен

- именованная или неименованная область, в которой определяются типы, переменные, константы

```
namespace имя {
    данные;
}
```

```
using namespace std;

namespace MyNamespace {
    const int x = 100;
    void f() { cout << "Hello, world!" << endl; }
}
```

Назначение

1) объединить в один логический блок классы, связанные с ними функции и т. п.

```
namespace DateTime {
    class Date { … };
    class Time { … };

    Time operator+(const Time& t1, const Time& t2){ … }
    void print(const Time& t){ … }
    …
}
```

Назначение

2) избежать конфликта имен

```
// project1.h
    const int x = 100;
    void f() { std::cout << "Hello, world! " << std::endl; }
```

```
// project2.h
    const int x = 1000;
    void f() { std::cout << "Goodbye, world!" << std::endl; }
```

```
#include "project1.h"
#include "project2.h"

void g() {
   f();    // Error
   std::cout << x << std::endl;    // Error
}
```

Разрешение конфликта имен (пример)

```
namespace PetyaNamespace {
    const int x = 100;
    void f() { std::cout << "Hello, world!" << std::endl; }
}

namespace VasyaNamespace {
    const int x = 1000;
    void f() { std::cout << "Goodbye, world!" << std::endl; }
}

void g() {
    PetyaNamespace::f();     // OK
     std::cout << VasyaNamespace::x << std::endl;    // OK
}
```

Доступ к членам

- используется оператор разрешения области видимости::

имя_пространства::имя _члена

```
namespace DateTime {
    class Date { … };
    class Time { … };

    Time operator+(const Time& t1, const Time& t2){ … }
    void print(const Time& t){ … }
}

void f() { DateTime::Time time1; }
```

Глобальное пространство имен

- название области видимости, в которой существуют глобальные переменные
- нужно, когда в глобальной и локальной области видимости есть совпадающие имена

```
::имя_члена
```

```
namespace DateTime {
    void f() { std::cout << "Test 1" << std::endl; }
}

void f() { std::cout << "Test 2" << std::endl; }
```

Глобальное пространство имен (пример)

```
void f() { std::cout << "Test 2" << std::endl; }

namespace DateTime {
    void f() { std::cout << "Test 1" << std::endl; }
    void g() {
        f();
        ::f();
    }
}

void g() {
    DateTime::g();
}

void main() {
    g();
}
```

Повторные объявления

- не создают новое пространство имен
- второе пространство с тем же именем просто продолжает первое

```
namespace DateTime {
    void test1() { … }
}

namespace DateTime {
    void test2() { … }
}

namespace DateTime {
    void test3() { … }
}
```

=

```
namespace DateTime {
    void test1() { … }
    void test2() { … }
    void test3() { … }
}
```

Повторные объявления

- в разных фрагментах одного пространства имен не должно быть членов с одинаковыми именами

```
namespace DateTime {
    void test1() { … }
}

namespace DateTime {
    void test2() { … }
}

namespace DateTime {
    void test2() { … } // Error
}
```

=

```
namespace DateTime {
    void test1() { … }
    void test2() { … }
    void test2() { … }  // Error
}
```

Доступ к членам пространства имен

с квалификатором (через ::)

```
namespace DateTime {
    void test1() { … }
}

void f() {
    DateTime::test1();
}
```

без квалификатора

- с объявлением **using**
- с директивой **using**

Объявление using

- «подключает» определенный член пространства имен

```
using имя_пространства::имя_члена;
```

```
namespace DateTime {
    void test1() { … }
    void test2() { … }
}

void g() {
    using DateTime::test1;
    test1();
}
```

Директива using

- «подключает» все имена членов пространства имен

```
using  namespace имя_пространства;
```

```
namespace DateTime {
    void test1() { … }
    void test2() { … }
}

void g() {
    using namespace DateTime;
    test1();
    test2();
}
```

Использование using

- использование using должно быть
  не в глобальной области видимости,
  а в локальной (внутри функции, класса и т. п.,
  в верхней части файла *.cpp)
- не следует размещать директивы using
  в файлах заголовков (*.h)
- в именах функций всегда должны быть
  полные имена

Безымянное пространство имен

- ограничивает область видимости своих членов
- имеет уникальное имя, к которому нельзя обратиться
- может быть использовано только в том файле, где объявлено

```
namespace { … }
```

```
namespace {
    void f() { std::cout << "Test 1" << std::endl; }
}

void g() { ::f(); }
```

Псевдонимы пространства имен

- позволяют обращаться к пространству имен по более короткому квалификатору

```
namespace псевдоним_пространства = имя_пространства;
```

```
namespace DateTime {
    void f() { std::cout << "Test 1" << std::endl; }
}

void g() {
    namespace DT = DateTime;
    DT::f();    // DateTime::f();
}
```

Вложенные пространства имен

- вложенное пространство имеет неограниченный доступ к членам родительского пространства
- родительское пространство не имеет неограниченного доступа к членам вложенного

```
namespace Outer {
    void f1() { std::cout << 48 << std::endl; }

    namespace Inner {
            int x = 100;
            void f2() { f1(); }
    }
     void f3() { std::cout << Inner::x << std::endl; }
}
```

Вопросы?