

*Каждый программный объект имеет область действия, которая определяется видом и местом его объявления.*

# Области действия идентификаторов

✓ Существуют следующие области действия:

- *блок,*
- *файл,*
- *функция,*
- *прототип функции,*
- *класс,*
- *поименованная область.*

## Блок

*• идентификаторы, описанные внутри блока, являются локальными.*

- **область действия** идентификатора начинается в точке определения и заканчивается в конце блока;
- **видимость** — в пределах блока и внутренних блоков;
- **время жизни** — до выхода из блока.

**После выхода из блока память освобождается.**

**Области действия идентификаторов.**

Файл

любого блока, функции, класса или пространства имен, имеют **глобальную видимость** и **постоянное**

- **время жизни** и являются **единственными** идентификаторами, используемыми, имеющими

Функция

такую область действия, являются метки операторов.

- *в одной функции все элементы структуры объединены и используются*
- **классов** (за исключением

Класс

статических элементов) являются видимыми лишь в пределах класса.

- *они образуются при создании переменной указанного типа и разрушаются при ее уничтожении.*

**Области действия идентификаторов.**

## Прототип функции


- идентификаторы, указанные в списке параметров прототипа (объявления) функции, имеют область действия только прототип функции.

## Поименованная область

- C++ позволяет явным образом задать область определения имен как часть глобальной области с помощью оператора **namespace**.

**Области действия идентификаторов.**

# **ОБЛАСТЬ ВИДИМОСТИ ИДЕНТИФИКАТОРОВ.**



*Область видимости совпадает с областью действия за исключением ситуации, когда во вложенном блоке описана переменная с таким же именем.*

*В этом случае внешняя переменная во вложенном блоке невидима, хотя он и входит в ее область действия.*

*Тем не менее, к этой переменной, если она глобальная, можно обратиться, используя операцию доступа к области видимости*

*⋮*

***Способ обратиться к скрытой локальной переменной отсутствует.***

# ПРОСТРАНСТВО ИМЕН.

✓ В каждой области действия различают так называемые пространства имен.

**Пространство имен** — область, в пределах которой идентификатор должен быть уникальным.

✓ В разных пространствах имена могут совпадать, поскольку разрешение идентификатора осуществляется по контексту идентификатора.

```
struct Node{  
    int Node;  
    int i ;  
} Node;
```

*В данном случае противоречия нет, поскольку имена типа, переменной и элемента структуры относятся к разным пространствам.*

# КЛАССЫ ИДЕНТИФИКАТОРОВ В C++.



*В C++ определено четыре отдельных класса идентификаторов, в пределах каждого из которых имена*

- Все они, кроме имен функций, могут быть переопределены во вложенных блоках.

- Каждое имя должно отличаться от имен других типов в той же области видимости.

- Имя элемента должно быть уникально внутри структуры, но может совпадать с именами элементов других структур.

# Внешние объявления

✓ Любая функция автоматически видна во всех модулях программы.

✓ Если требуется ограничить область действия функции файлом, в котором она описана, используется модификатор **static**.



# ***ВНЕШНИЕ ОБЪЯВЛЕНИЯ.***

*✓ Для того чтобы сделать доступной в нескольких модулях переменную или константу, необходимо:*

- 1. определить ее только в одном модуле как глобальную;*
- 2. в других модулях объявить ее как внешнюю с помощью модификатора **extern**.*

*✓ Другой способ — поместить это объявление в заголовочный файл и включить его в нужные модули.*

*Объявление, в отличие от определения, не создает переменную.*

*Объявление с **extern** не должно содержать инициализацию: если она присутствует, модификатор **extern** игнорируется.*

A person wearing a blue protective suit and a helmet with a visor is holding a device. The background is a stylized, futuristic cityscape with blue and purple tones.

*Все описания одной и той же  
переменной должны быть  
согласованы*

## Пример:

Описание двух глобальных переменных в файлах **one.cpp** и **two.cpp** с помощью заголовочного файла **my\_header.h**:

```
// my_header.h - внешние объявления
```

```
extern int a;
```

```
extern double b;
```

```
.....
```

```
// -----
```

```
// one.cpp
```

```
#include "my_header.h"
```

```
int a;
```

```
.....
```

```
// -----
```

```
// two.cpp
```

```
#include "my_header.h"
```

```
double b;
```

Обе переменные доступны в файлах **one.cpp** и **two.cpp**.

# Внешние объявления.



*Если переменная описана как **static**, область ее действия ограничивается файлом, в котором она описана.*

- Как правило, это делается в заголовочном файле, который затем подключается к модулям, использующим этот тип.

Нарушение этого правила приводит к ошибкам, которые трудно обнаружить.

- Поскольку компиляторы, как правило, не обладают возможностью сличать определения одного и того же типа в различных файлах.

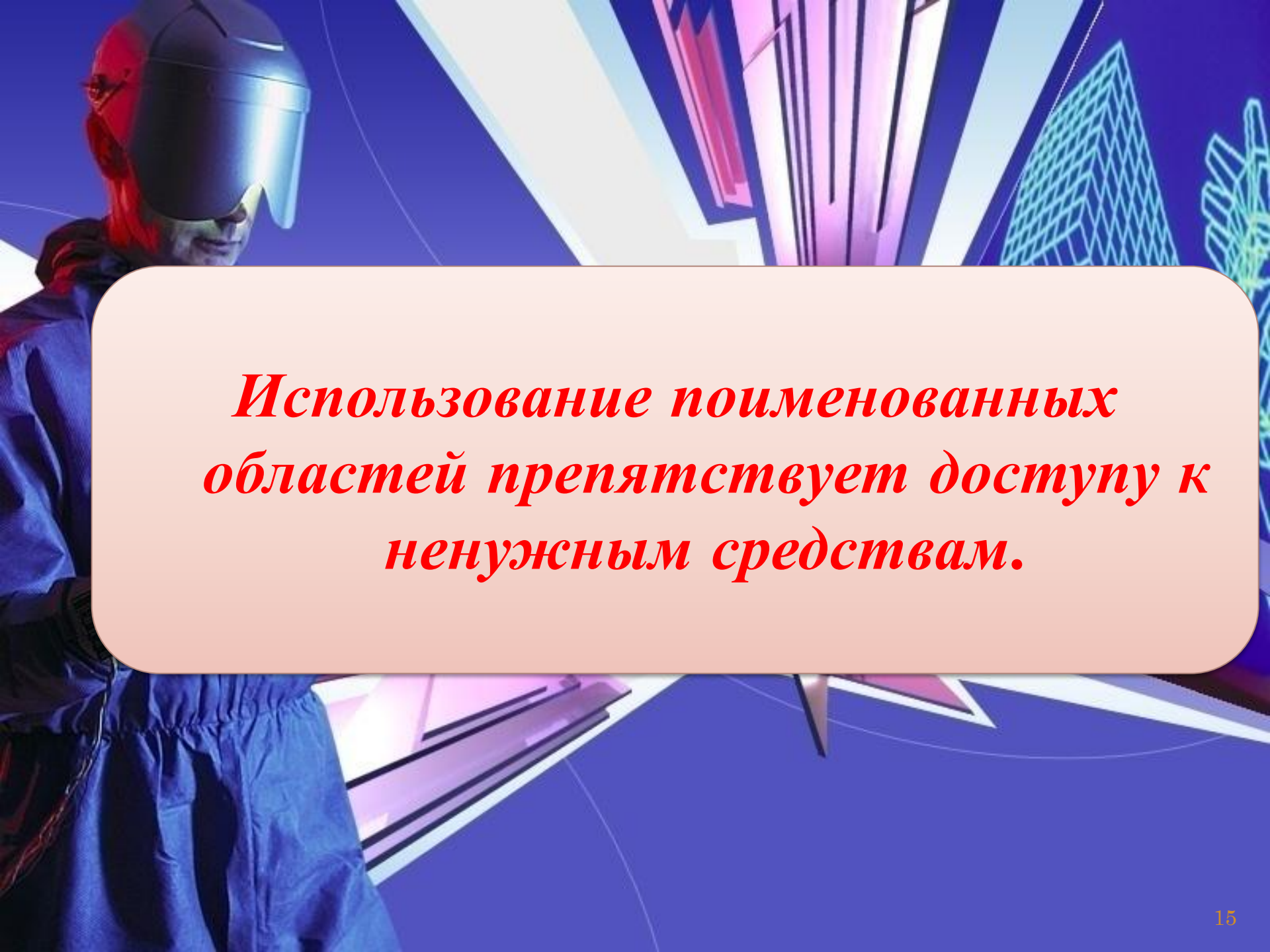
## Внешние объявления.

# Поименованные области

Поименованные области служат для логического группирования объявлений и ограничения доступа к ним.

Простейшим примером применения является отделение кода, написанного одним человеком, от кода, написанного другим.

При использовании единственной глобальной области видимости формировать программу из отдельных частей очень сложно из-за возможного совпадения и конфликта



*Использование поименованных областей препятствует доступу к ненужным средствам.*

# ПОИМЕНОВАННЫЕ ОБЛАСТИ.

✓ *Объявление поименованной области (ее также называют пространством имен) имеет формат:*

```
namespace [ имя_области ]  
{ /* Объявления */ }
```

✓ *Поименованная область может объявляться неоднократно, причем последующие объявления рассматриваются как расширения предыдущих.*

*Таким образом, поименованная область может объявляться и изменяться за рамками одного файла.*



- Объявление объекта в неименованной области равнозначно его описанию как глобального с модификатором **static**.

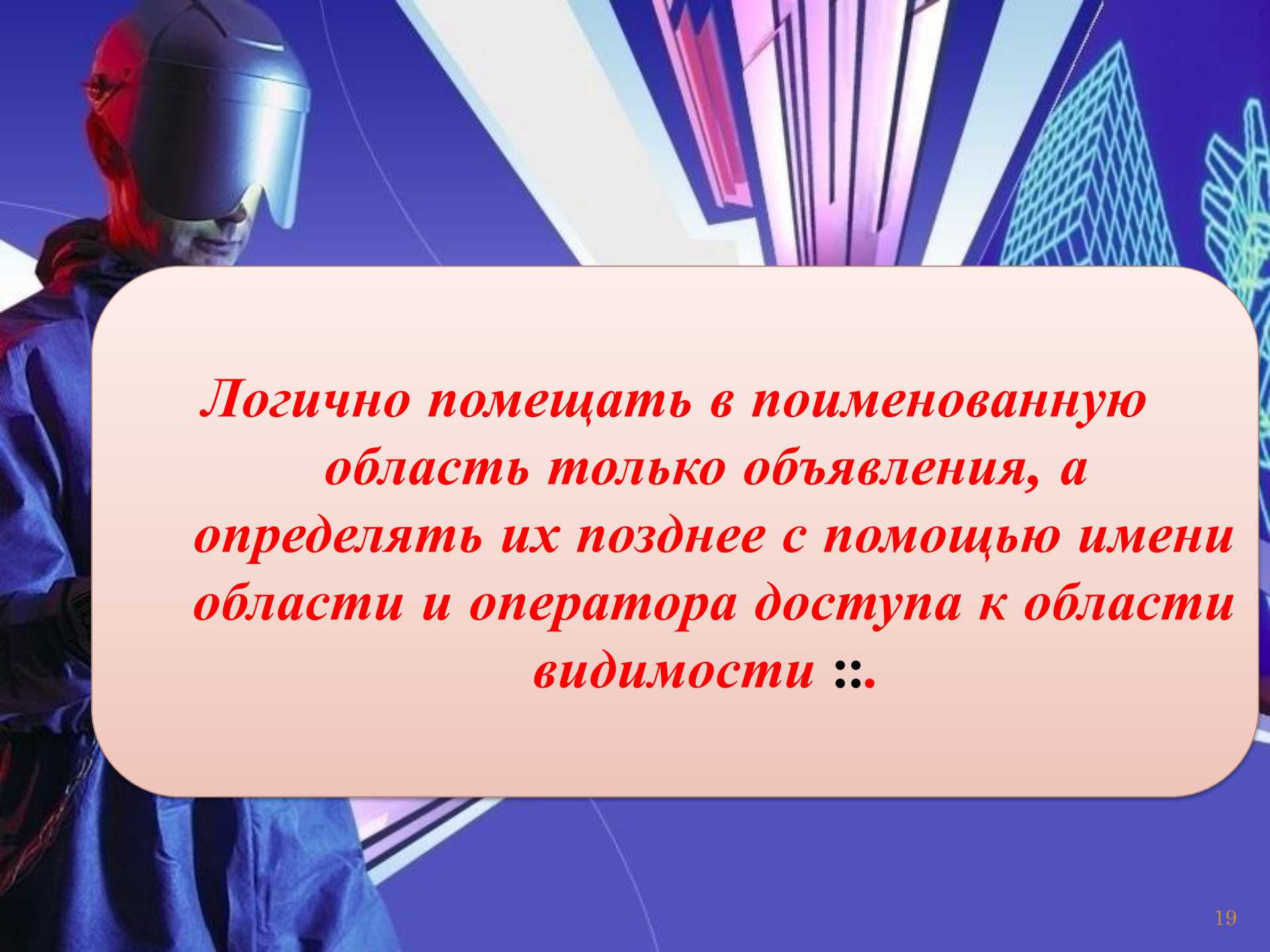
**Поименованные области.**

## Пример:

```
namespace demo{
    int i = 1;
    int k = 0;
    void func1(int);
    void func2(int) { /* ... */ }
}
namespace demo{ // Расширение
// int i = 2;    Неверно - двойное определение
    void func1(double); // Перегрузка
    void func2(int);    // Верно (повторное объявление)
}
```

В объявлении поименованной области могут присутствовать как объявления, так и определения.

# Поименованные области.



*Логично помещать в поименованную область только объявления, а определять их позднее с помощью имени области и оператора доступа к области видимости ::.*

**Пример:**

```
void demo::func1(int) { /* ... */ }
```

*Такой прием применяется для разделения интерфейса и реализации.*

Таким способом нельзя объявить новый элемент пространства имен.

**Поименованные области.**

Объекты, объявленные внутри области, являются видимыми с момента объявления.

- К ним можно явно обращаться с помощью имени области и оператора доступа к области видимости ::
- **demo::i = 100; demo::func2(10);**

**Поименованные области.**

Если имя часто используется вне своего пространства, можно объявить его доступным с помощью оператора **using**:

```
using demo::i;
```

- После этого можно использовать имя без явного указания области.

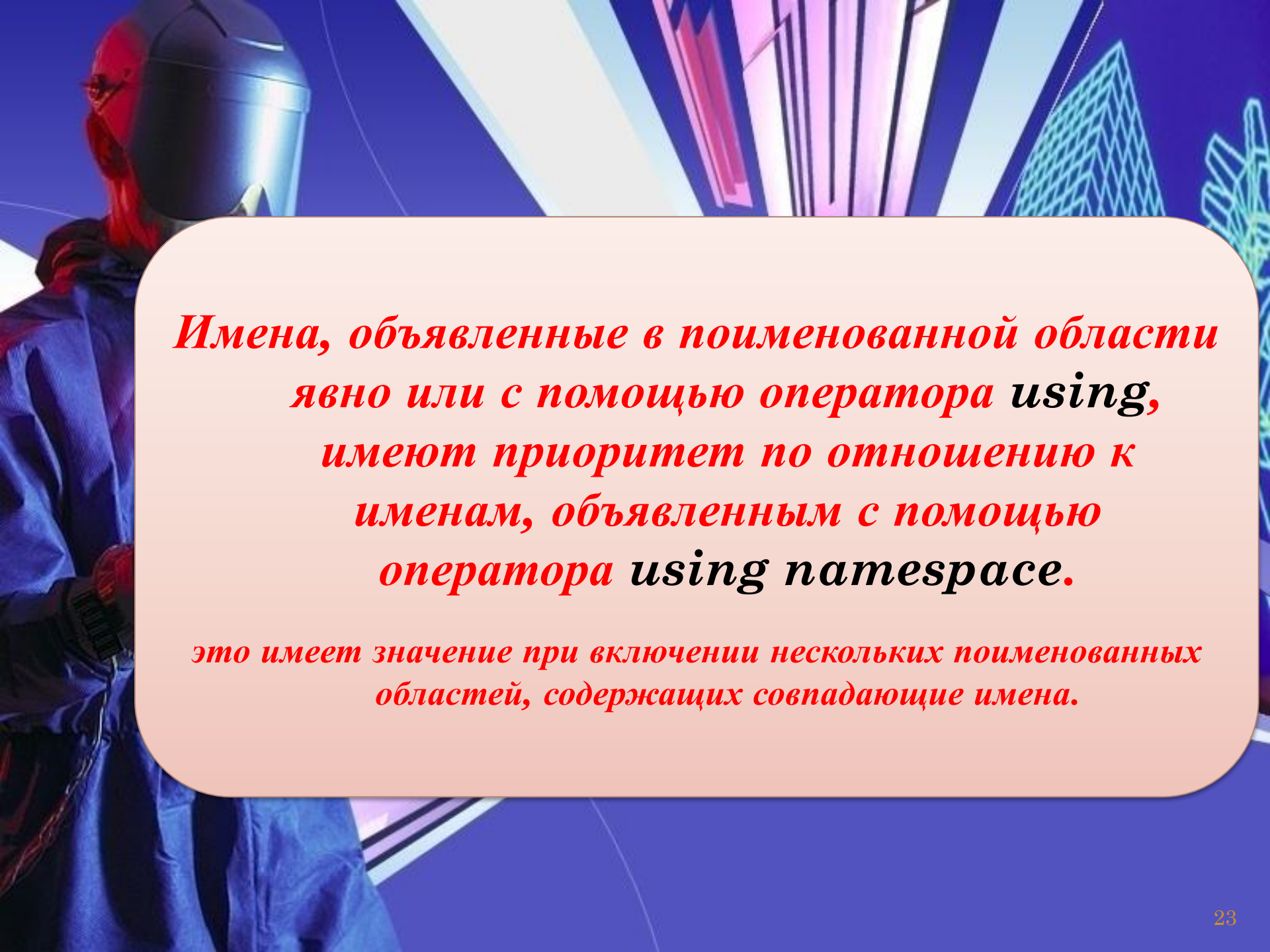
Если требуется сделать доступными все имена из какой-либо области, используется оператор **using namespace**:

```
using namespace demo;
```

*Операторы **using** и **using namespace** можно использовать и внутри объявления поименованной области, чтобы сделать в ней доступными объявления из другой области:*

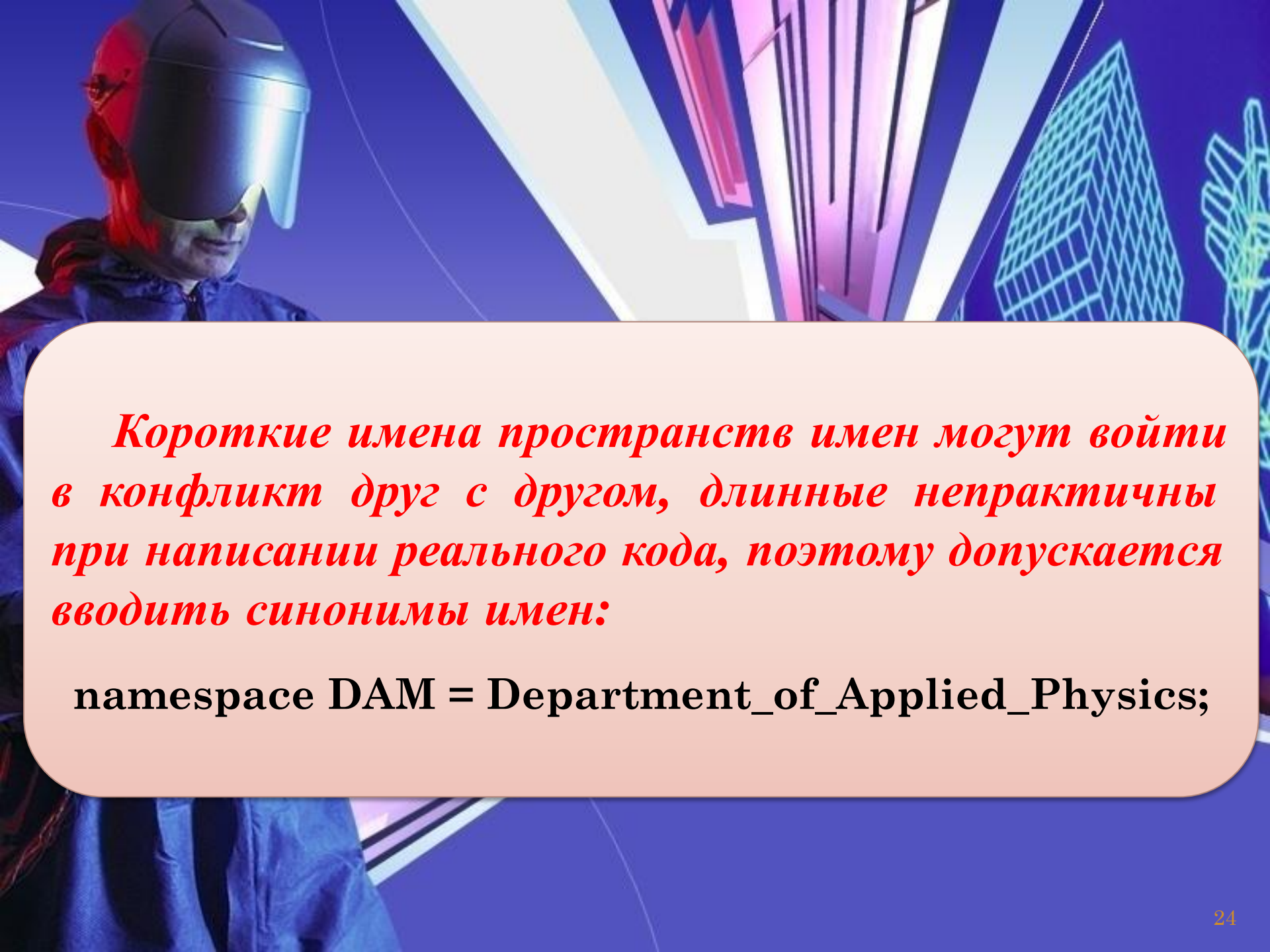
```
namespace Department_of_Applied_Physics  
{ using demo::i; // ... }
```

## Поименованные области.



*Имена, объявленные в поименованной области явно или с помощью оператора **using**, имеют приоритет по отношению к именам, объявленным с помощью оператора **using namespace**.*

*это имеет значение при включении нескольких поименованных областей, содержащих совпадающие имена.*



*Короткие имена пространств имен могут войти в конфликт друг с другом, длинные непрактичны при написании реального кода, поэтому допускается вводить синонимы имен:*


**namespace DAM = Department\_of\_Applied\_Physics;**



# Пространства имен стандартной библиотеки

Объекты стандартной библиотеки  
определены в пространстве имен **std**.


# *ПРОСТРАНСТВА ИМЕН СТАНДАРТНОЙ БИБЛИОТЕКИ.*

 *Объявления стандартных средств ввода/вывода C в заголовочном файле <stdio.h> помещены в пространство имен следующим образом:*


```
// stdio.h  
namespace std{  
    int feof(FILE *f);  
    .....  
}  
using namespace std;
```

*Это обеспечивает совместимость сверху вниз.*

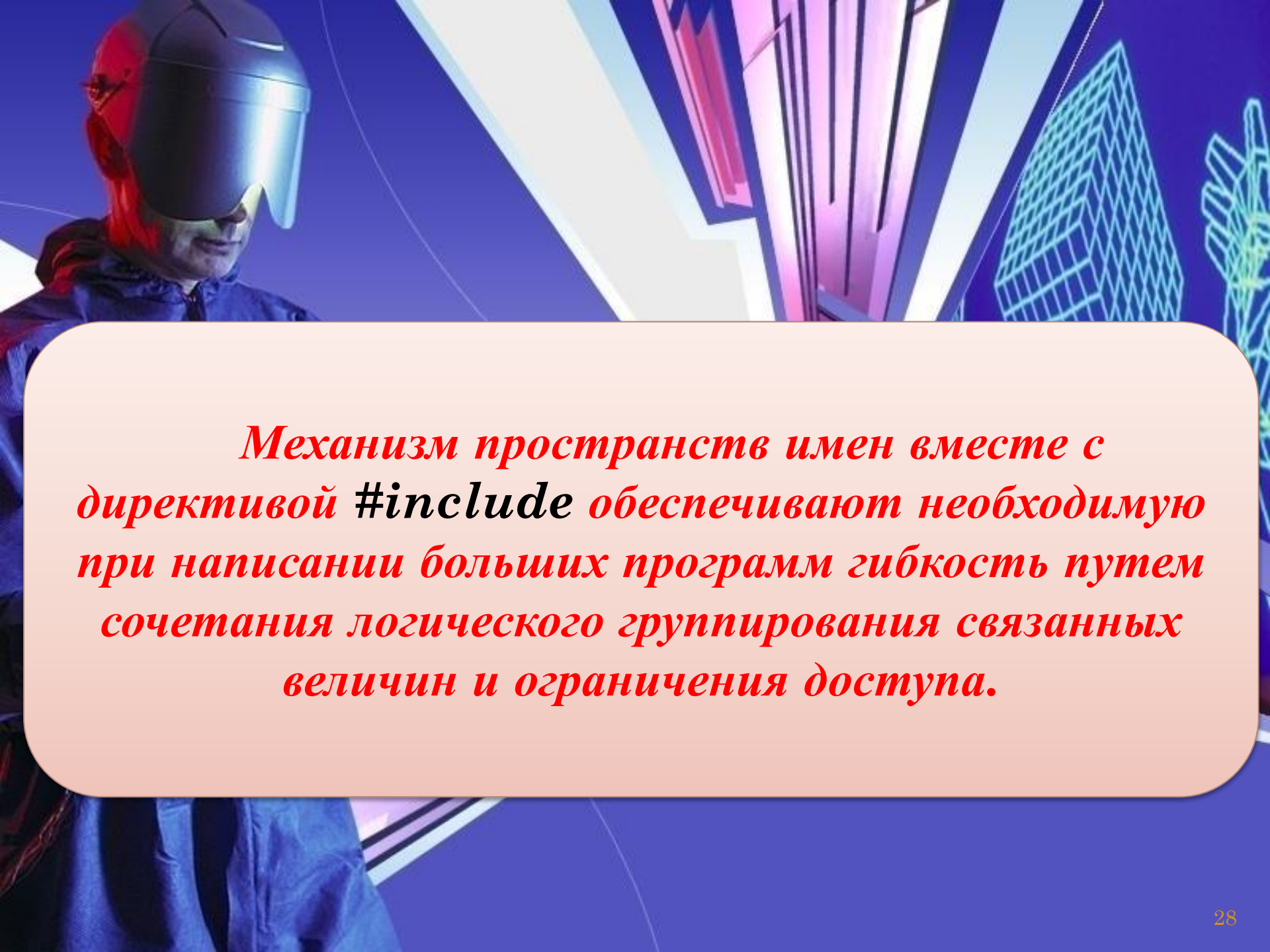
# *ПРОСТРАНСТВА ИМЕН СТАНДАРТНОЙ БИБЛИОТЕКИ.*

 Для тех, кто не желает присутствия неявно доступных имен, определен новый заголовочный файл `<cstdio>`:

```
// cstdio.h
namespace std{
    int feof(FILE *f);
    .....
}
```

 Если в программу включен файл `<cstdio>`, нужно указывать имя пространства имен явным образом:

```
std::feof(f)
```



*Механизм пространств имен вместе с директивой `#include` обеспечивают необходимую при написании больших программ гибкость путем сочетания логического группирования связанных величин и ограничения доступа.*



*Продуманное разбиение программы на модули, четкая спецификация интерфейсов и ограничение доступа позволяют организовать эффективную работу над проектом группы программистов.*

# СПАСИБО ЗА ВНИМАНИЕ !!!

□ До встречи на экзамене!!!



*Искренне Ваш, И.В. Климов.*