ОПЕРАЦИИ В ЯЗЫКЕ С

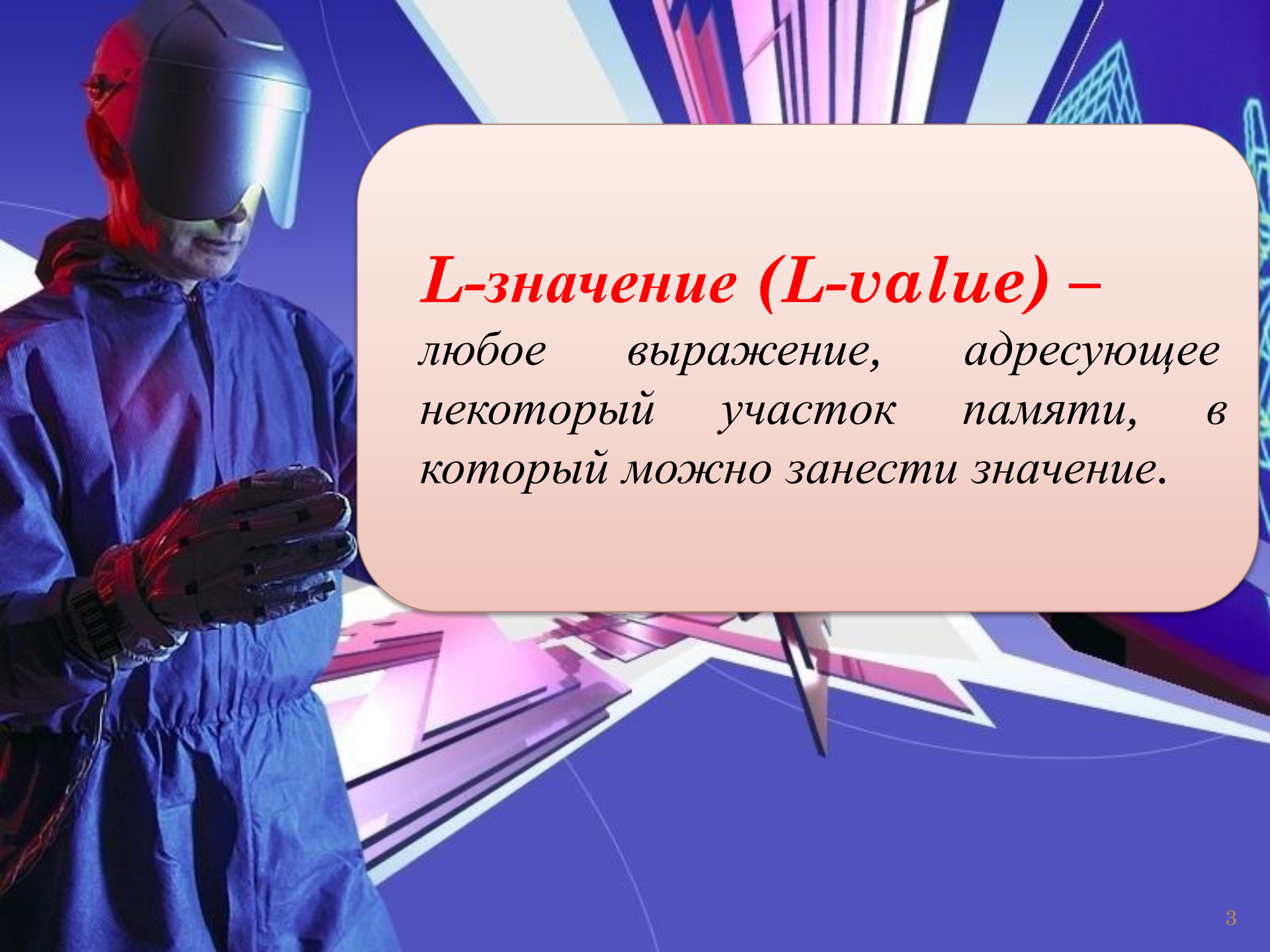
ОПЕРАЦИИ В ЯЗЫКЕ С .

В соответствии с количеством операндов, которые используются в операциях, они делятся на:

- **Унарные** - один операнд.
- **Бинарные** - два операнда.
- **Тернарную** - три операнда.

Все операции, кроме **условной** *и* **sizeof**, *могут быть* ***перегружены***.

Пробелы между символами внутри операции не



L-значение (L-value) –

любое выражение, адресующее некоторый участок памяти, в который можно занести значение.

L-ЗНАЧЕНИЕ (L-VALUE).

- ***Название L-value произошло от операции присваивания, поскольку именно ее левая (Left) часть определяет, в какую область памяти будет занесен результат операции.***

- ***Переменная является частным случаем L-значения.***

УНАРНЫЕ ОПЕРАЦИИ.

Операция	Краткое описание
++	увеличение на 1
--	уменьшение на 1
sizeof	размер
~	поразрядное отрицание
!	логическое отрицание
-	арифметическое отрицание (унарный минус)
+	унарный плюс
&	взятие адреса
*****	разадресация (косвенная адресация);
new	выделение памяти
Delete	освобождение памяти
(type)	преобразование типа

Унарные операции имеют наивысший приоритет

БИНАРНЫЕ ОПЕРАЦИИ.

ПРИОРИТЕТ

Операция	Описание	Операция	Описание	Операция	Описание
*	умножение	/	деление	%	остаток от деления
+	сложение	-	вычитание		
<<	сдвиг влево	>>	сдвиг вправо		

БИНАРНЫЕ ОПЕРАЦИИ.

ПРИОРИТЕТ

Операция	Описание	Операция	Описание	Операция	Описание	Операция	Описание
<	меньше	**<=**	меньше или равно	**>**	больше	**>=**	больше или равно
==	равно	**!=**	не равно				

БИНАРНЫЕ ОПЕРАЦИИ.

ПРИОРИТЕТ

Операция	Описание
&	поразрядная конъюнкция (И)
^	поразрядное исключающее ИЛИ
|	поразрядная дизъюнкция (ИЛИ)
&&	логическое И
||	логическое ИЛИ

ТЕРНАРНАЯ ОПЕРАЦИЯ.

ПРИОРИТЕТ

Операция	Описание
?:	условная операция

БИНАРНЫЕ ОПЕРАЦИИ.

Операция	Описание
=	присваивание
*=	умножение с присваиванием
/=	деление с присваиванием
%=	остаток отделения с присваиванием
+=	сложение с присваиванием
-=	вычитание с присваиванием
<<=	сдвиг влево с присваиванием
>>=	сдвиг вправо с присваиванием
&=	поразрядное И с присваиванием
|=	поразрядное ИЛИ с присваиванием
^=	поразрядное исключающее ИЛИ с присваиванием
,	последовательное вычисление

Эти операции имеют самый низкий приоритет

Операции увеличения и уменьшения на 1 (++ и --).

- Эти операции, называются **инкрементом** и **декрементом**

ОПЕРАЦИИ ++ И --.

Операции ++ и -- имеют две формы записи:

- ***Префиксная*** *- операция записывается перед операндом*
- ***Постфиксная*** *- операция записывается после операнда*

В ***префиксной*** *форме сначала изменяется операнд, а затем его значение становится результирующим значением выражения.*

В п *одное знач*

Операндом операции инкремента в общем случае является ***L-значение***.

Пример:

```
#include <stdio.h>
int main(){
    int x = 3, y = 3;
    printf("Значение префиксного выражения: %d\n", ++x);
    printf("Значение постфиксного выражения: %d\n", y++);
    printf("Значение x после приращения: %d\n", x);
    printf("Значение y после приращения: %d\n ", y);
return 0;
}
```

Результат:

```
Значение префиксного выражения: 4
Значение постфиксного выражения: 3
Значение x после приращения: 4
Значение y после приращения: 4
```

Операции увеличения и уменьшения на 1 (++ и --)

Операция определения размера sizeof.

- Операция **sizeof** предназначена для вычисления размера объекта или типа в байтах

ОПЕРАЦИЯ ОПРЕДЕЛЕНИЯ РАЗМЕРА SIZEOF.

Операция* sizeof *две формы:

- **sizeof выражение**
- **sizeof (тип)**

Пример:

```
#include <iostream.h>
int main(){
    float x = 1;
    cout « "sizeof (float)  :" « sizeof (float);
    cout « "\nsizeof x :" « sizeof x;
    cout « "\nsizeof (x + 1.0)  :" « sizeof (x + 1.0);
return 0;
}
```

Результат:

```
sizeof (float)  : 4
sizeof x : 4
sizeof (x + 1.0)  : 8
```

Операция определения размера sizeof

*Последний результат связан с тем, что вещественные константы по умолчанию имеют тип **double**, к которому, как к более длинному, приводится тип переменной **x** и всего выражения.*

Скобки необходимы для того чтобы выражение, стоящее в них, вычислялось раньше операции приведения типа, имеющей больший приоритет, чем сложение.

Операция определения размера sizeof

Операции отрицания (-, ! и ~)

- Арифметическое отрицание (**-**)
- Логическое отрицание (**!**)
- Поразрядное отрицание (**~**)

Операции отрицания

Арифметическое отрицание (-)	• изменяет знак операнда целого или вещественного типа на противоположный
Логическое отрицание (!)	• дает в результате значение 0, если операнд истина (не нуль), и значение 1, если операнд равен нулю. • операнд должен быть целого или вещественного типа, может также иметь тип указатель
Поразрядное отрицание (~)	• инвертирует каждый разряд в двоичном представлении целочисленного операнда • поразрядное отрицание часто называют побитовым

Деление (/) и остаток от деления (%)

- Операция деления (**/**) применима к операндам арифметического типа
  - ***Если оба операнда целочисленные, результат операции округляется до целого числа, в противном случае тип результата определяется правилами преобразования.***
- Операция остатка от деления (**%**) применяется только к целочисленным операндам.
  - ***Знак результата зависит от реализации***

Пример:

```
#include <stdio.h>
int main(){
    int x = 11, y = 4;
    float z = 4;
    printf("Результаты деления: %d   %f\n",  x/y, x/z);
    printf("Остаток: %d\n", x%y);
return 0;
}
```

Результат:

```
Результаты деления: 2   2.750000
Остаток: 3
```

Деление (/) и остаток от деления (%)

Операции сдвига (<< и >>)

- Операции сдвига (**<<** и **>>**) применяются к целочисленным операндам, они сдвигают двоичное представление первого операнда влево или вправо на количество двоичных разрядов, заданное вторым операндом.

ОПЕРАЦИИ СДВИГА (<< И >>).

- ***При сдвиге влево (<<) освободившиеся разряды обнуляются.***

- ***При сдвиге вправо (>>) освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом в противном случае***

Операции сдвига не учитывают переполнение и потерю значимости.

Операции отношения (<, <=, >, >=, ==, !=)

- Операции отношения сравнивают первый операнд со вторым.

ОПЕРАЦИИ ОТНОШЕНИЯ.

- ***Операнды могут быть арифметического типа или указателями.***
- ***Результатом операции является значение*** **true** ***или*** **false.**
  - ***любое значение, не равное нулю, интерпретируется как***

Операции сравнения на равенство и неравенство имеют меньший приоритет, чем остальные операции сравнения.

Поразрядные операции (&, |, ^)

- поразрядная конъюнкция **&**
- поразрядная дизъюнкция **|**
- поразрядное исключающее ИЛИ **^**

ПОРАЗРЯДНЫЕ ОПЕРАЦИИ.

Поразрядные операции применяются только к целочисленным операндам и работают с их двоичными представлениями.

При выполнении операций операнды сопоставляются <u>побитно</u>.

поразрядная конъюнкция &	•бит результата равен 1 только тогда, когда соответствующие биты обоих операндов равны 1
поразрядная дизъюнкция |	•бит результата равен 1 тогда, когда соответствующий бит хотя бы одного из операндов равен 1
поразрядное исключающее ИЛИ ^	•бит результата равен 1 только тогда, когда соответствующий бит только одного из операндов равен 1.

Поразрядные операции

Пример:

```
#include <iostream.h>
int main(){
    cout << "\n 6  & 5 = " << (6 & 5);
    cout << "\n 6  |  5 = " << (6 | 5);
    cout << "\n 6  ^ 5 = " << (6 ^ 5);
return 0;
}
```

Результат:

```
6  & 5 = 4
6  |  5 = 7
6  ^ 5 = 3
```

Поразрядные операции

Логические операции (&& и ||)

- логическое И **&&**
- логическое ИЛИ **||**

ЛОГИЧЕСКИЕ ОПЕРАЦИИ.

- ***Операнды логических операций И (&&) и ИЛИ (||) могут иметь арифметический тип или быть указателями.***
  - ***при этом операнды в каждой операции могут быть различных типов.***
- ***Преобразования типов не производятся, каждый операнд оценивается с точки зрения его эквивалентности нулю.***
  - ***операнд, равный нулю, рассматривается как* false, *не***
- ***Л***

Если значения первого операнда достаточно, чтобы определить результат операции, второй операнд не вычисляется.

Логические операции

логическое И
&&

- результат операции имеет значение **true** только если оба операнда имеют значение **true**

логическое ИЛИ
||

- результат операции имеет значение **true,** если хотя бы один из операндов имеет значение **true**

Операции присваивания (=, +=, –=, /=, *= и т. д.)

- Операции присваивания могут использоваться в программе как законченные операторы.

Формат операции *простого присваивания* (=):

операнд_1 = операнд_2

- ***мнемоническое правило: «присваивание — это передача данных "налево"»***

Операции присваивания (=, +=, −=, /=, *= и т. д.)

Пример:

```
#include <iostream.h>
int main(){
      int a = 3, b=5, c=7;
     a = b; b = a; c = c + 1;
          cout << "a = " << a;
          cout << "\t b = " << b;
          cout << "\t c = " << c;
return 0;
}
```

Результат:

a = 5 b = 5 c = 8

Операции присваивания

В сложных операциях присваивания

(+=, *=, /= и т.п.)

при вычислении выражения, стоящего в правой части, используется и

L-значение из левой части

- Например, при *сложении с присваиванием* ко второму операнду прибавляется первый, и результат записывается в первый операнд
- То есть выражение **a += b** является более компактной записью выражения **a = a + b**

Операции присваивания

Условная операция (?:)

- Эта операция тернарная, то есть имеет три операнда.

Условная операция (?:)

Формат условной операции:

операнд_1 ? операнд_2 : операнд_3

Первый операнд может иметь арифметический тип или быть указателем

- операнд, равный нулю, рассматривается как **false**
- не равный нулю — как **true**

*Если результат вычисления операнда 1 равен **true**, то результатом условной операции будет значение второго операнда, иначе — третьего операнда.*

- Вычисляется всегда либо второй операнд, либо третий
- Их тип может различаться

*Условная операция является сокращенной формой условного оператора **if***

Пример:

```
#inc1ude <stdio.h>
int main(){
    int  a = 11, b = 4, max;
    max = (b > a)? b : a;
    printf("Наибольшее число: %d", max);
return 0;
}
```

Результат:

Наибольшее число: 11

Условная операция (?:)

Другой пример применения условной операции:

*Требуется, чтобы некоторая целая величина увеличивалась на **1**, если ее значение не превышает **n**, а иначе принимала значение **1**:*

```
i = (i < n) ? i + 1: 1;
```

Условная операция (?:)

Операции выполняются в соответствии с *приоритетами*

- для изменения порядка выполнения операций используются круглые скобки

Если в одном выражении записано несколько операций одинакового приоритета:

- унарные операции, условная операция и операции присваивания выполняются ***справа налево***
- остальные — ***слева направо***

Например:

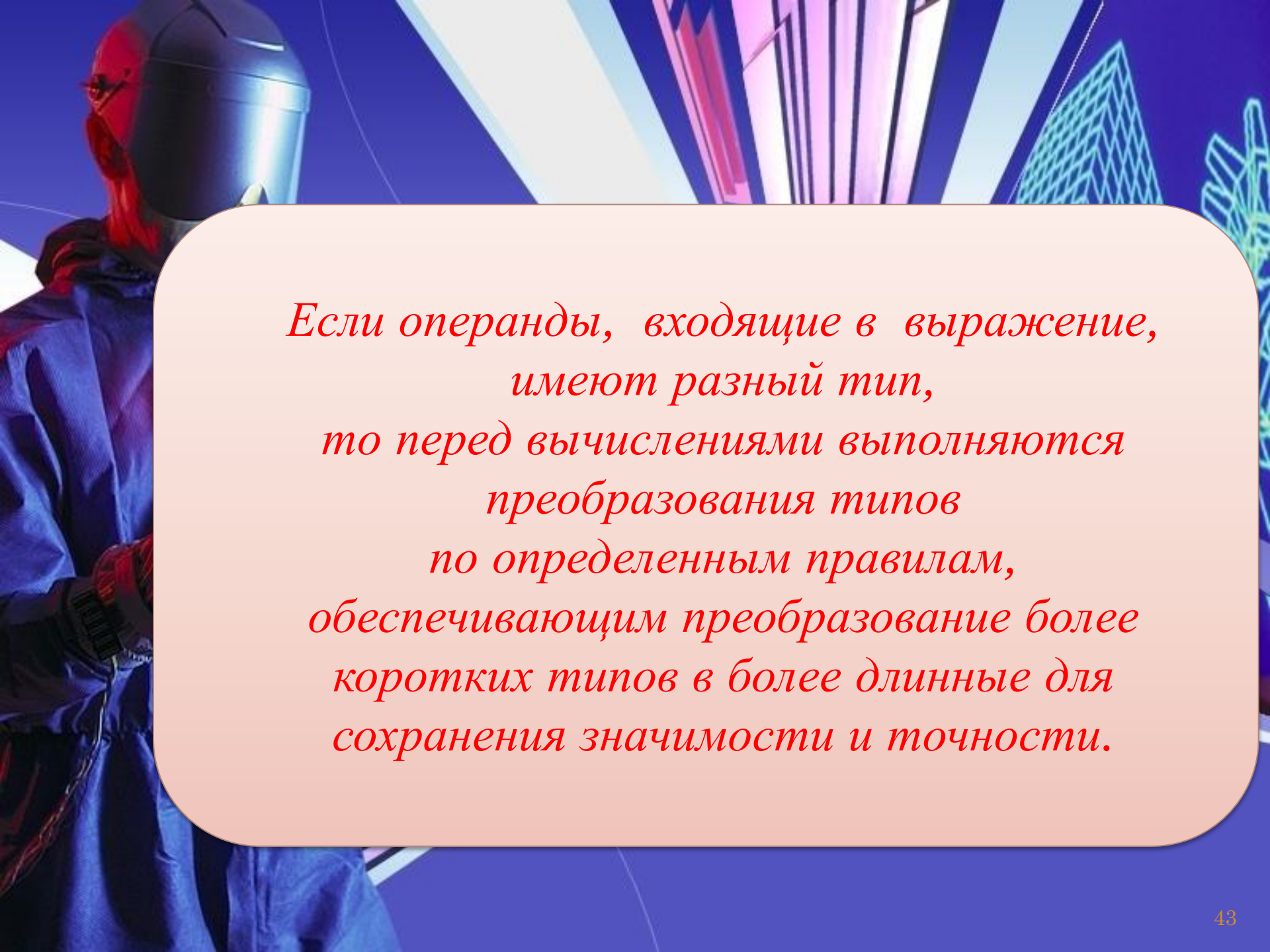
- **a = b = c** *означает* **a = (b = c)**
- **a + b + c** *означает* **(a + b) + c**

Замечания

в **C** допустимы выражения вида **a = b = c**

- сначала вычисляется выражение **b = c**
- затем его результат становится правым операндом для операции присваивания переменной **a**

Замечания



Если операнды, входящие в выражение, имеют разный тип, то перед вычислениями выполняются преобразования типов по определенным правилам, обеспечивающим преобразование более коротких типов в более длинные для сохранения значимости и точности.

Преобразования бывают двух типов:

изменяющие внутреннее представление величин

- с потерей точности
- без потери точности

изменяющие только интерпретацию внутреннего представления

К первому типу относится, например, преобразование целого числа в вещественное, ко второму — преобразование знакового целого в беззнаковое.

Замечания

В любом случае величины типов **char**, **signed char, unsigned char, short int** и **unsigned short int**

преобразуются в тип **int**, если он может представить все значения, или в **unsigned**

В противном случае

После этого операнды преобразуются к типу наиболее длинного из них, и он используется как тип результата.

Программист может задать преобразования типа явным образом

Замечания