# ОСНОВЫ ТЕОРИИ И ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

**Пышкин Евгений Валерьевич**
к.т.н., доцент

# ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ

Блок 2. Типы данных и алгоритмы

# Понятие типа данных

- Предпосылки из математики
  - вещественные отличаются от комплексных
  - действительные отличаются от рациональных и трансцендентных
  - целочисленные вычисления отличаются от алгебры логики
- Предпосылки из теории множеств
  - логические парадоксы (антиномии)

# Логические парадоксы и типы

- Пусть имеется множество A всех множеств X, не содержащих себя в качестве одного из элементов. Содержит ли множество A себя в качестве элемента? (Рассел, 1902)
- Разрешение введением понятия типа
  - элементы множества приписываются некоторому типу Type1
  - сами множества – типу Type2
  - множества множеств – типу Type3

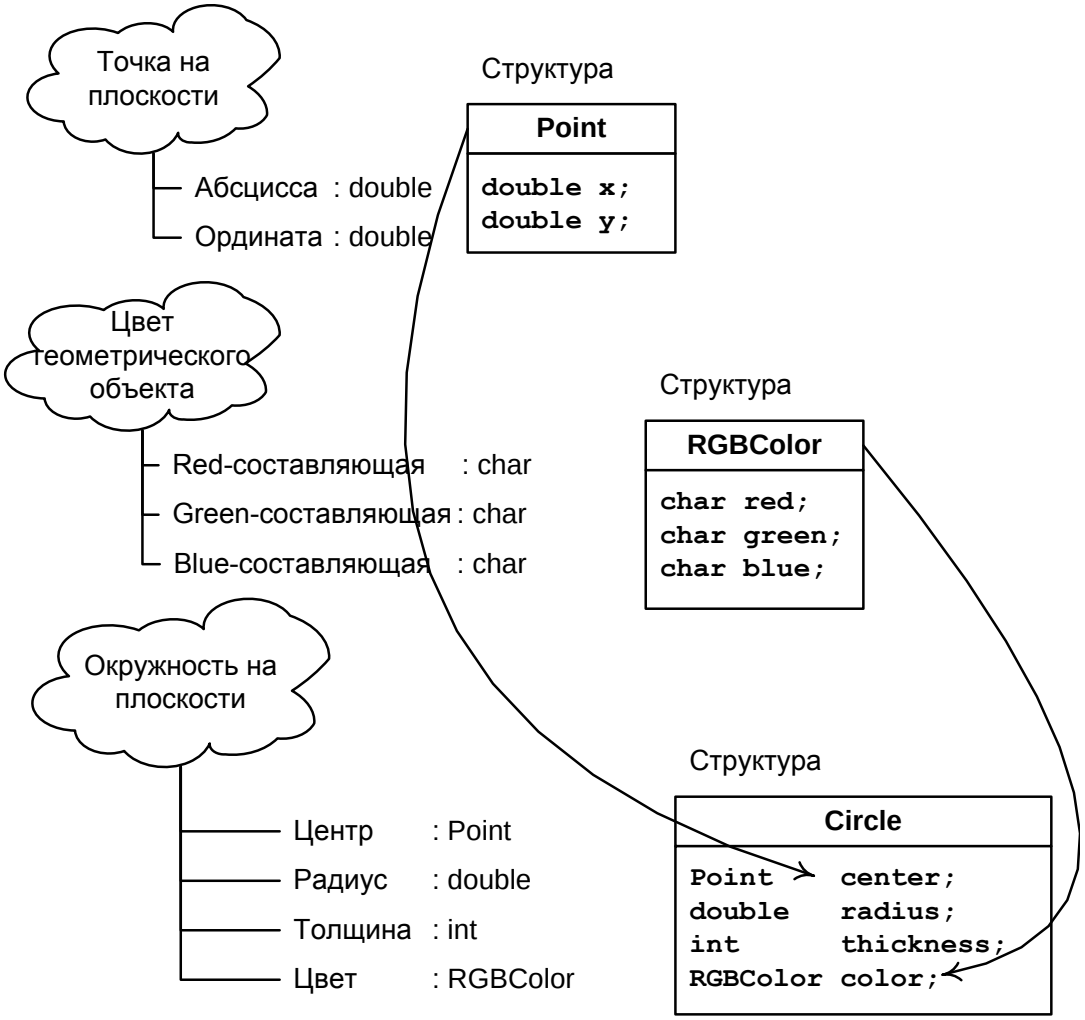
# Понятие типа по Ч. Хоару

1. Тип определяет класс значений, которые могут принимать переменная или выражение.
2. Каждое значение принадлежит одному и только одному типу.
3. Тип значения константы, переменной или выражения можно вывести либо из контекста, либо из самого операнда, не обращаясь к значениям, вычисляемым во время работы программы.
4. Каждой операции соответствует некоторый фиксированный тип ее операндов и некоторый фиксированный (обычно такой же) тип результата. Разрешение систематической неопределенности в случае, когда один и тот же символ применяется к операндам разного типа, производится на стадии компиляции.
5. Для каждого типа свойства значений и элементарных операций над значениями задаются с помощью аксиом.
6. При работе с языком программирования знание типа позволяет обнаруживать бессмысленные конструкции и решать вопрос о методе представления данных и преобразования их в ЭВМ.

# Типы и языки программирования

- Не все языки являются строго типизированными
- Встроенные типы
  - числовые
  - логические (булевские)
  - символьные
  - другие
- Интегральные типы
  - В C++: int, short, long, char, bool
- Составные (производные) типы
  - структуры, массивы, объединения…

# Конструирование составных типов



# Стандартные типы данных

- Целые типы
- Символьные типы
- Плавающие типы
- Булевский тип

# Спецификаторы целых типов со знаком

- **`char`** для работы с 8-разрядными числами со знаком в диапазоне -128..127
- **`short`** для работы с 16-разрядными числами со знаком в диапазоне -32 768..32 767
- **`long`** для работы с 32-разрядными числами со знаком в диапазоне -2 147 483 648..2 147 483 647
- **`int`** (стандартный целый тип), размер которого зависит от архитектуры компьютера
- (C99) **`long long`** для работы с 64-разрядными числами со знаком в диапазоне

  - 9 223 372 036 854 775 808.. 9 223 372 036 854 775 807

*См. заголовочный файл limits.h*

# Цепочки битов (целые без знака)

- **`unsigned char`** для работы с 8-разрядными векторами битов в диапазоне 0..255
- **`unsigned short`** для работы с 16-разрядными векторами битов в диапазоне 0..65 535
- **`unsigned long`** для работы с 32-разрядными векторами битов в диапазоне 0..4 294 967 295
- **`unsigned int`** для работы с 16- или 32-разрядными векторами битов в зависимости от архитектуры компьютера
- (C99) **`unsigned long long`** для работы с 64-разрядными векторами битов в диапазоне

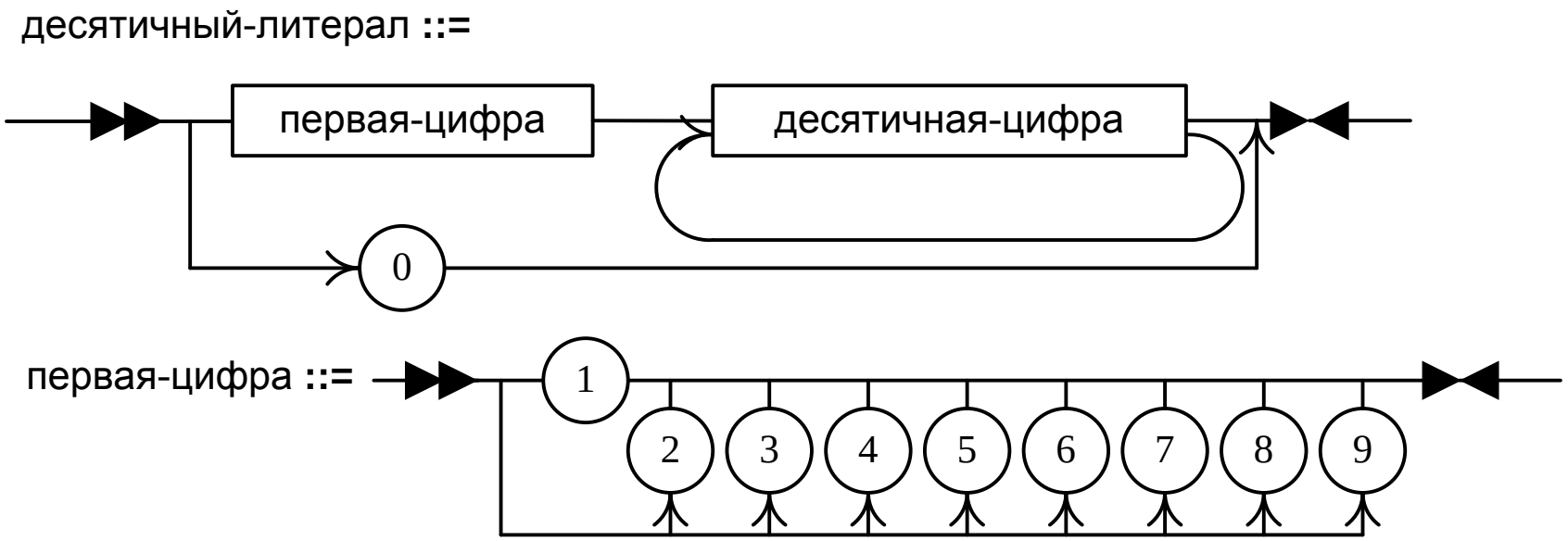
  0.. 18 446 744 073 709 551 615

```
unsigned int u;
if( u > -1 ) ...
```

# Microsoft Visual C++

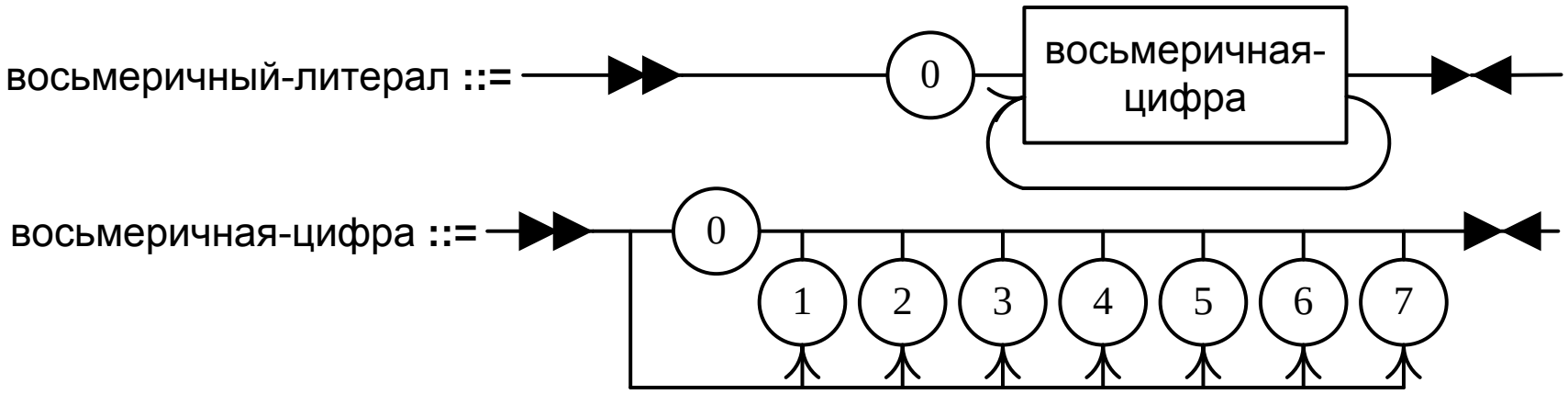
- **__int8** для работы с 8-разрядными числами со знаком в диапазоне -128..127
- **__int16** для работы с 16-разрядными числами со знаком в диапазоне -32 768..32 767
- **__int32** для работы с 32-разрядными числами со знаком в диапазоне -2 147 483 648..2 147 483 647
- **__int64** для работы с 64-разрядными числами со знаком в диапазоне от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807

# Целочисленные литералы




```
0    110    2570                00    0110    257A
```

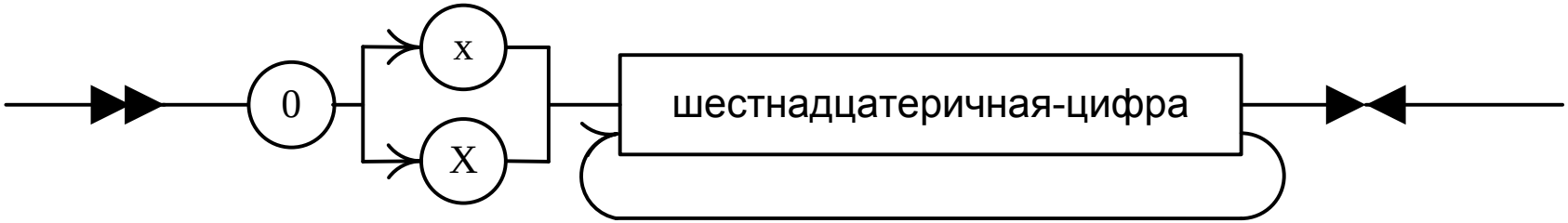
# Целочисленные литералы



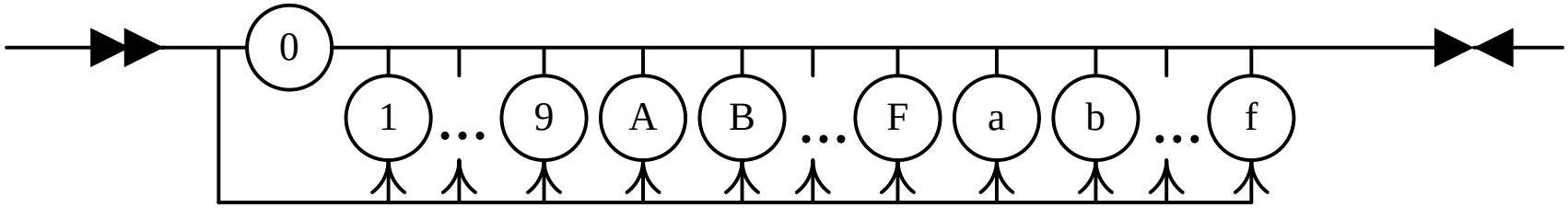

```
070    00110    02560          110    0258
```

# Целочисленные литералы

шестнадцатеричный-литерал **::=**




шестнадцатеричная-цифра **::=**

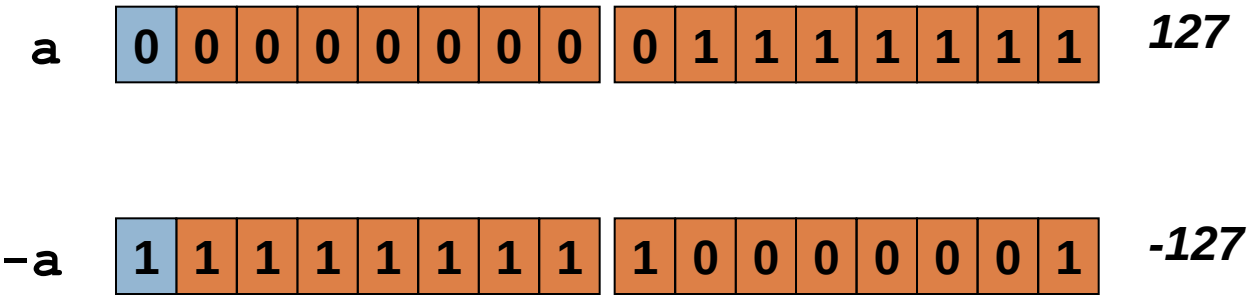



```
0x70    0Xffff    0X2EF7          0    ffff    0xG
```

# Операции над данными целых типов

- Одноместные операции
  - взятие с обратным знаком

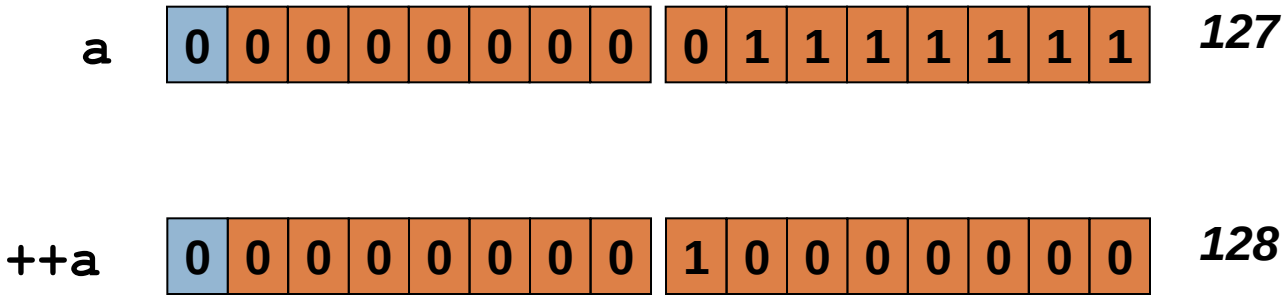
```
short a = 127;
```



# Операции над данными целых типов

- Одноместные операции
  - увеличение на 1 (префиксная форма)

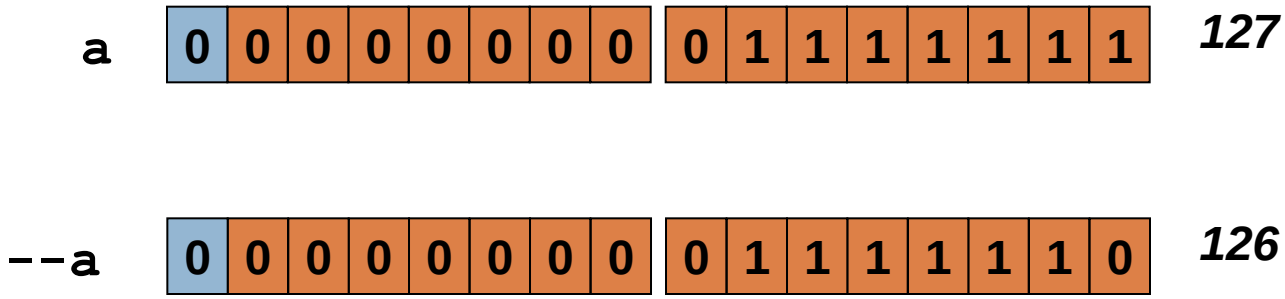
```
short a = 127;
```



# Операции над данными целых типов

- Одноместные операции
  - уменьшение на 1 (префиксная форма)

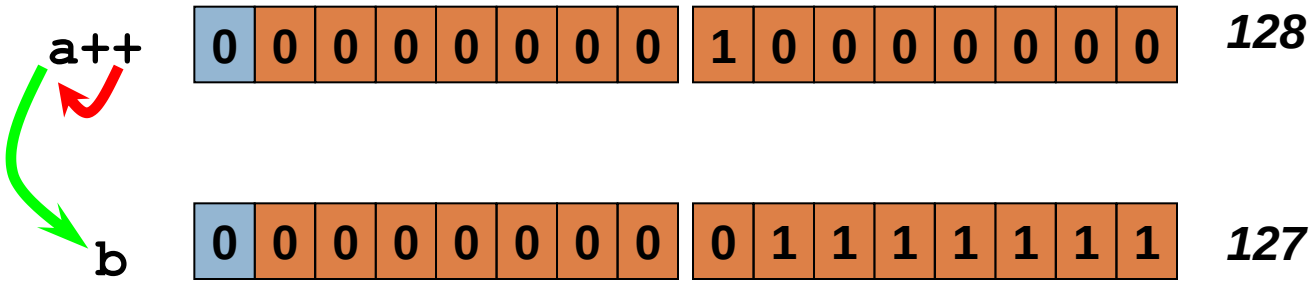
```
short a = 127;
```



# Операции над данными целых типов

- Одноместные операции
  - увеличение на 1 (постфиксная форма)

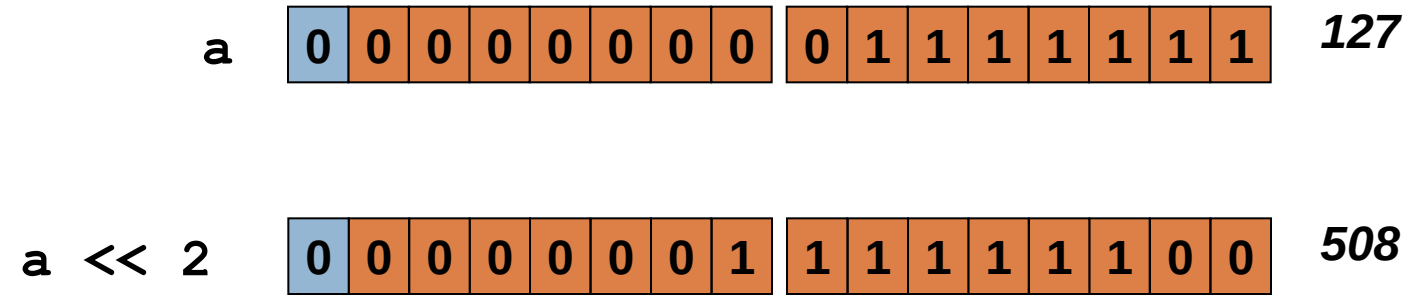
```
short a = 127;
short b = a++;
```



# Операции над данными целых типов

- Двуместные операции
  - сдвиг влево

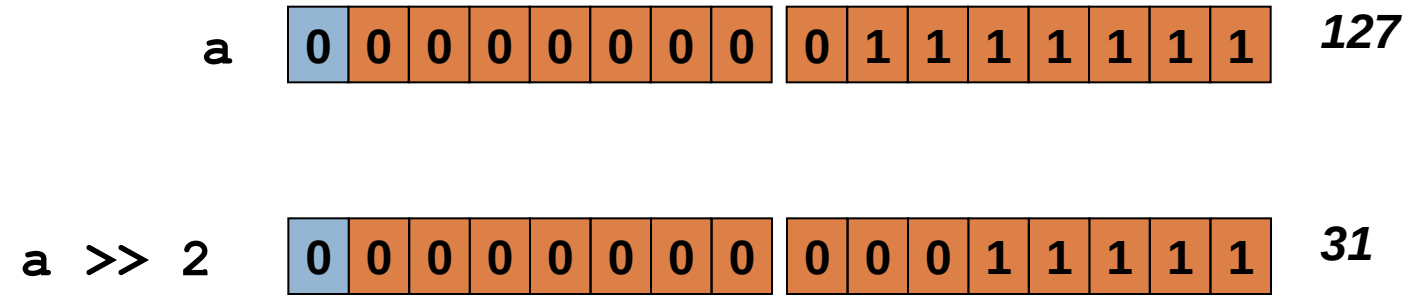
```
short a = 127;
short b = a << 2;
```



# Операции над данными целых типов

- Двуместные операции
  - сдвиг вправо

```
short a = 127;
short b = a >> 2;
```

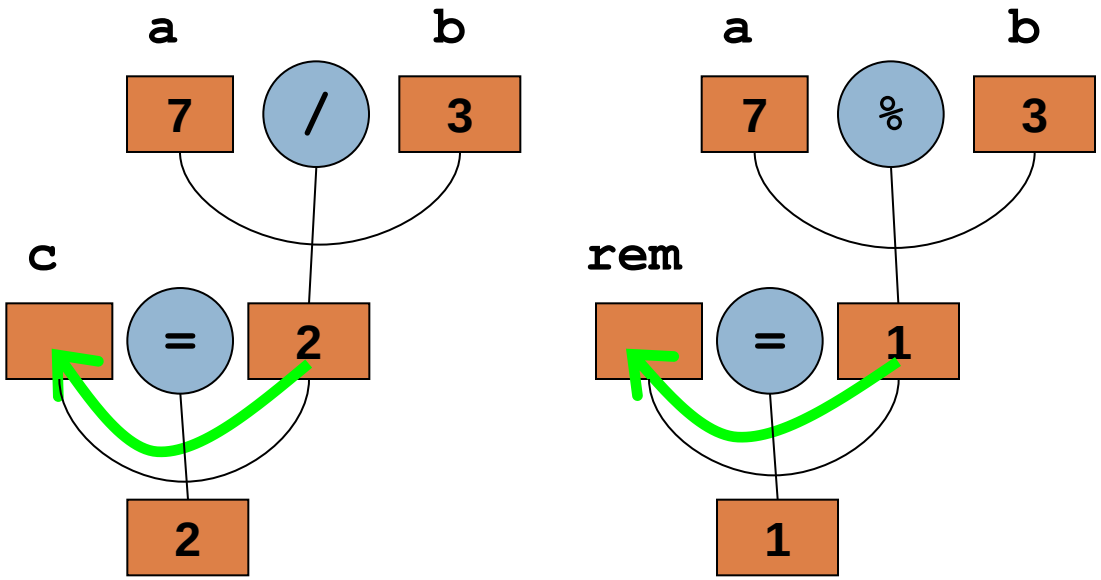


# Операции над данными целых типов

- Двуместные операции
  - мультипликативные (умножение, деление, остаток от деления)

```
int a = 7;
int b = 3;
int c;
int rem;

c = a / b;
rem = a % b;
```

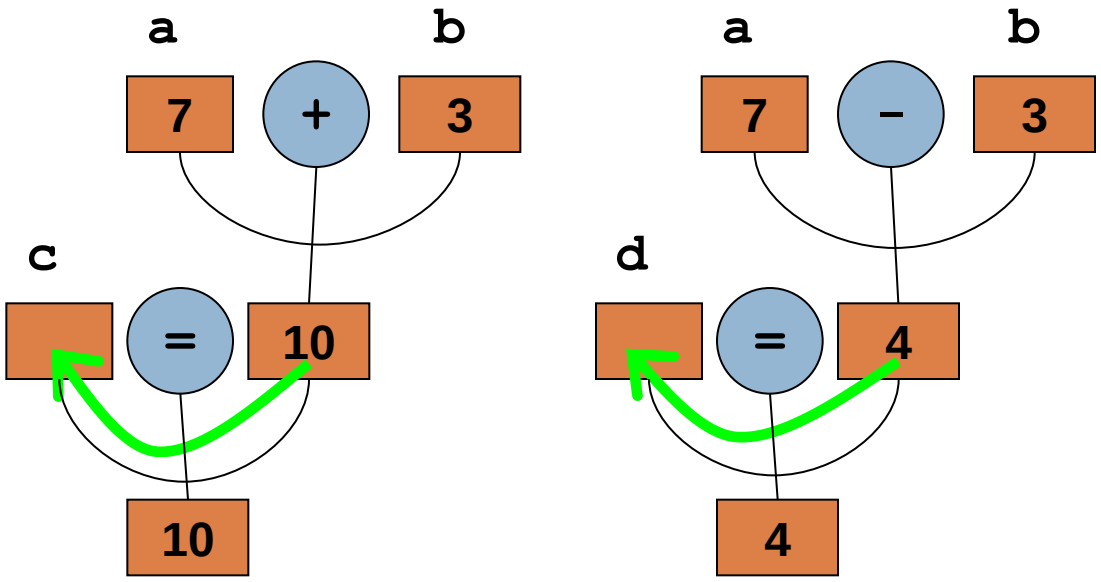


# Операции над данными целых типов

- Двуместные операции
  - аддитивные (сложение, вычитание)

```
int a = 7;
int b = 3;
int c;
int d;

c = a + b;
d = a - b;
```



# Понятие типа по Ч. Хоару

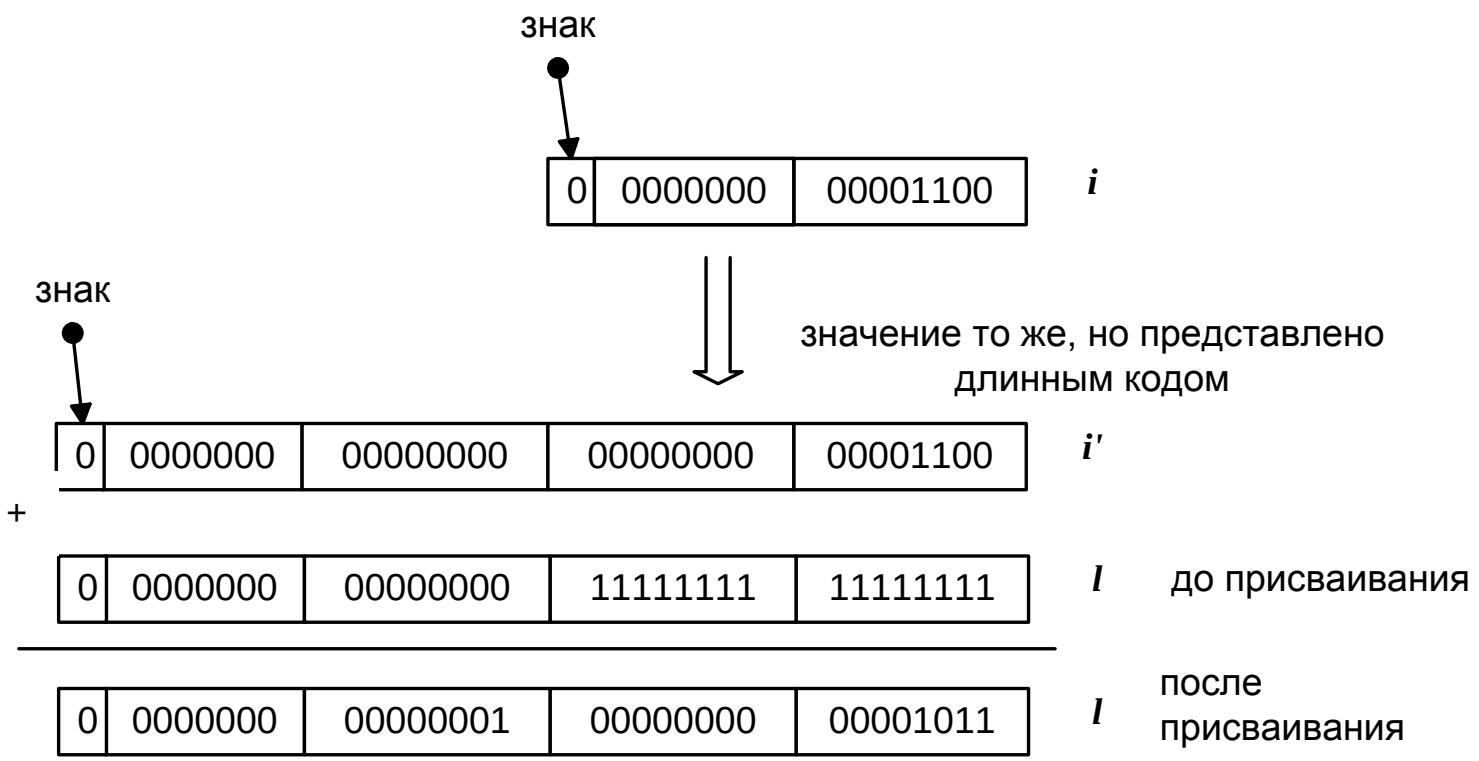
1. Тип определяет класс значений, которые могут принимать переменная или выражение.
2. Каждое значение принадлежит одному и только одному типу.
3. Тип значения константы, переменной или выражения можно вывести либо из контекста, либо из самого операнда, не обращаясь к значениям, вычисляемым во время работы программы.
4. Каждой операции соответствует некоторый фиксированный тип ее операндов и некоторый фиксированный (обычно такой же) тип результата. Разрешение систематической неопределенности в случае, когда один и тот же символ применяется к операндам разного типа, производится на стадии компиляции.
5. Для каждого типа свойства значений и элементарных операций над значениями задаются с помощью аксиом.
6. При работе с языком программирования знание типа позволяет обнаруживать бессмысленные конструкции и решать вопрос о методе представления данных и преобразования их в ЭВМ.

# Когда возникает необходимость преобразования типа

- Типы отличаются только разрядностью кода, и возможно строго определенное преобразование значения одного типа в другой за счет **изменения разрядности**
- Типы отличаются структурой кода, и возможно строго определенное преобразование за счет **изменения структуры кода**
- Типы отличаются семантически и преобразование возможно только в виде иной «интерпретации» кода, образуемого цепочкой двоичных разрядов, результат в общем случае непредсказуем

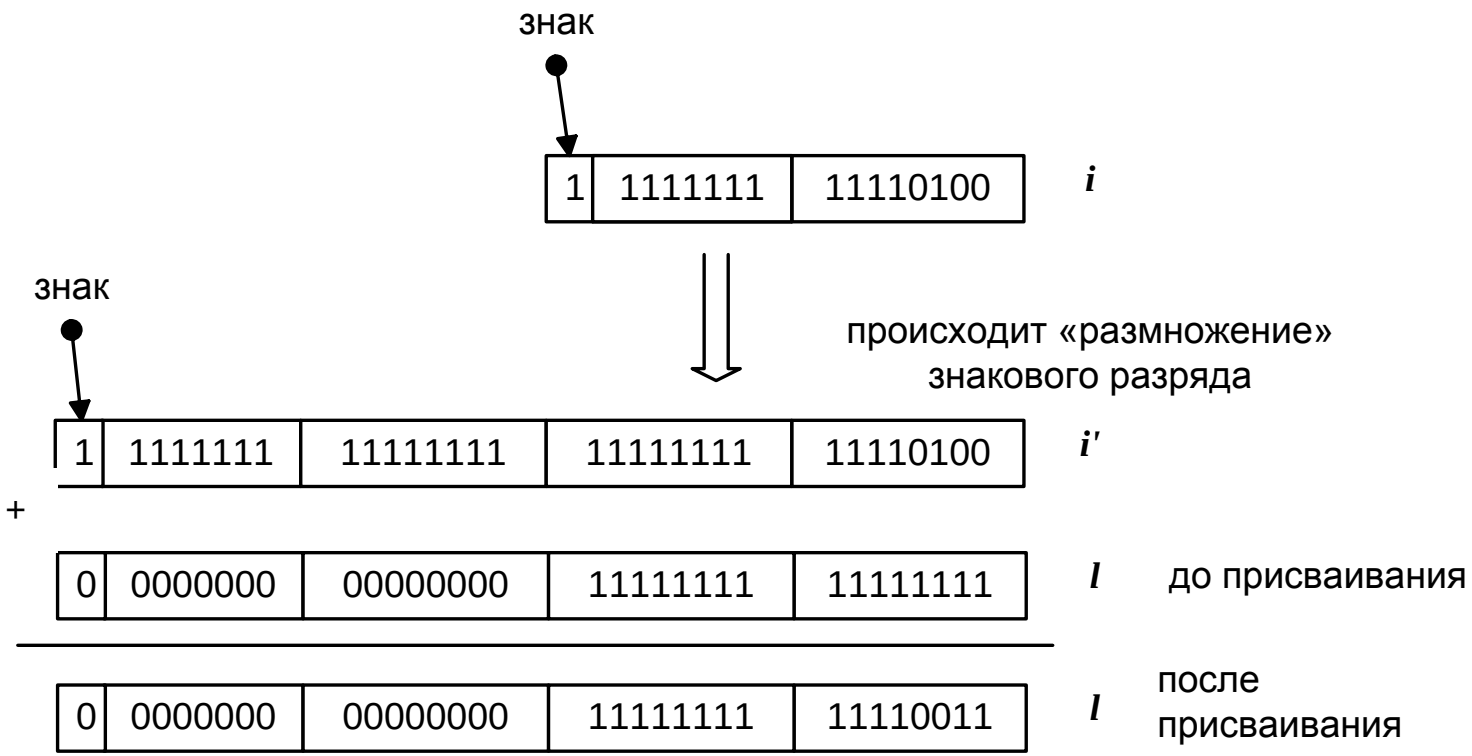
# Неявные преобразования типов

```
short i = 12;
long  l = 65535;
l = l + i;
```



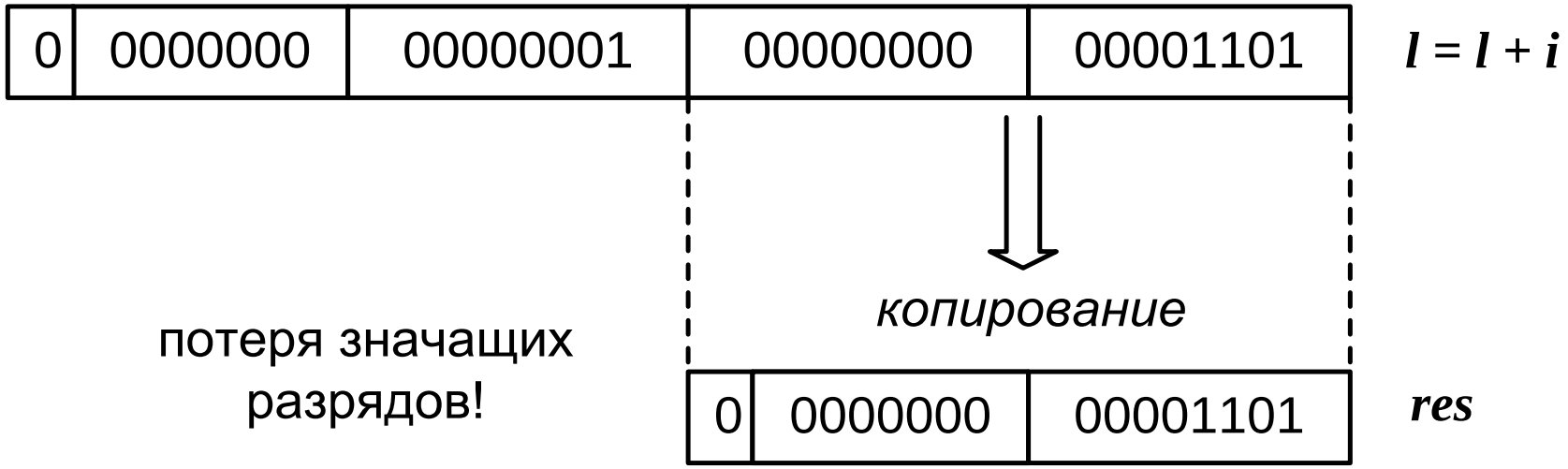
# Неявные преобразования типов

```
short i = -12;
long  l = 65535;
l = l + i;
```



# Неявные преобразования типов

```
short i = 12;
long  l = 65535;
short res;
l = l + i;
res = l; // ???
```



# Явные преобразования типов

```
res = (short) l;
```

```
res = static_cast <short> ( l );
```

- Когда результирующее приведение типа достигается неявным преобразованием, явное преобразование использовать

  ***НЕ НУЖНО!***

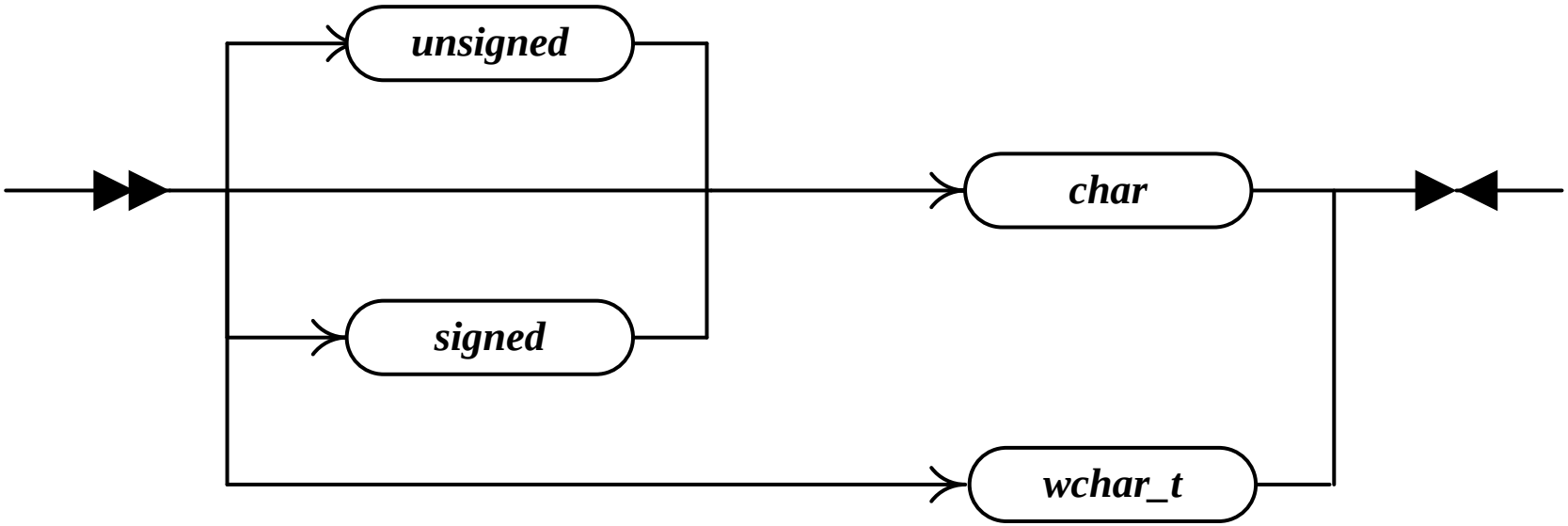
- ***Общефилософский принцип: «Не все то, что можно сделать, нужно делать»***

# Резюме

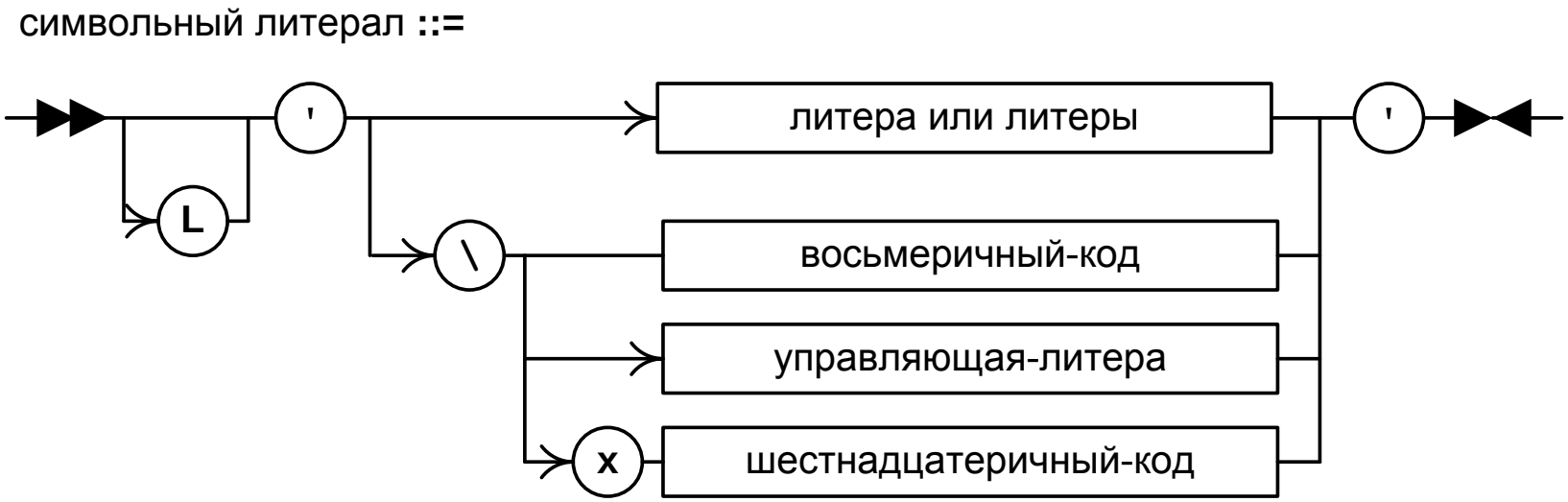
- Целые типы для точных вычислений
- В результате выполнения операции над операндами целого типа получается результат целого типа
- При выполнении вычислений возможны преобразования типов
- Многие операции над целыми данными могут быть применены и к данным других типов
- Решение многих задач связано с обработкой данных целых типов

# Символьные типы

спецификация-символьного-типа **::=**



# Символьные литералы




```
'A'   '\0101'   '\x41'   L'ab'              65   '65'
```

# Символьные литералы

- Литералы специального вида

```
// Литерал  DEC   HEX
   '\0'   // 000   0x00 Null         NUL   Нуль-символ
   '\a'   // 007   0x07 Alert        BEL   Звуковой сигнал
   '\b'   // 008   0x08 Backspace    BS    Удаление
          //                                предшествующего
          //                                символа
   '\t'   // 009   0x09 Tab          HT    Табуляция
   '\n'   // 010   0x0A New line     LF    Новая строка
   '\r'   // 013   0x0D Return       CR    Возврат каретки
   '\f'   // 014   0x0E Formfeed     FF    Прогон формата

   '\"'   // 034   0x22 Quotation mark     Кавычка
   '\''   // 039   0x27 Apostrophe         Апостроф
   '\\'   // 092   0x5C Backslash          Обратная косая черта
```

# Резюме

- В разных языках символьные данные могут поддерживаться различным образом
- Однобайтовый и двухбайтовый форматы представления кодов символов
- В C++ обработка данных символьных типов практически не отличается от действий над целыми кодами
- Дополнительную наглядность тексту придает использование символьных литералов
- Особый случай использования обработки последовательностей символов – строки символов (обсуждаются позднее)

# Понятие алгоритма

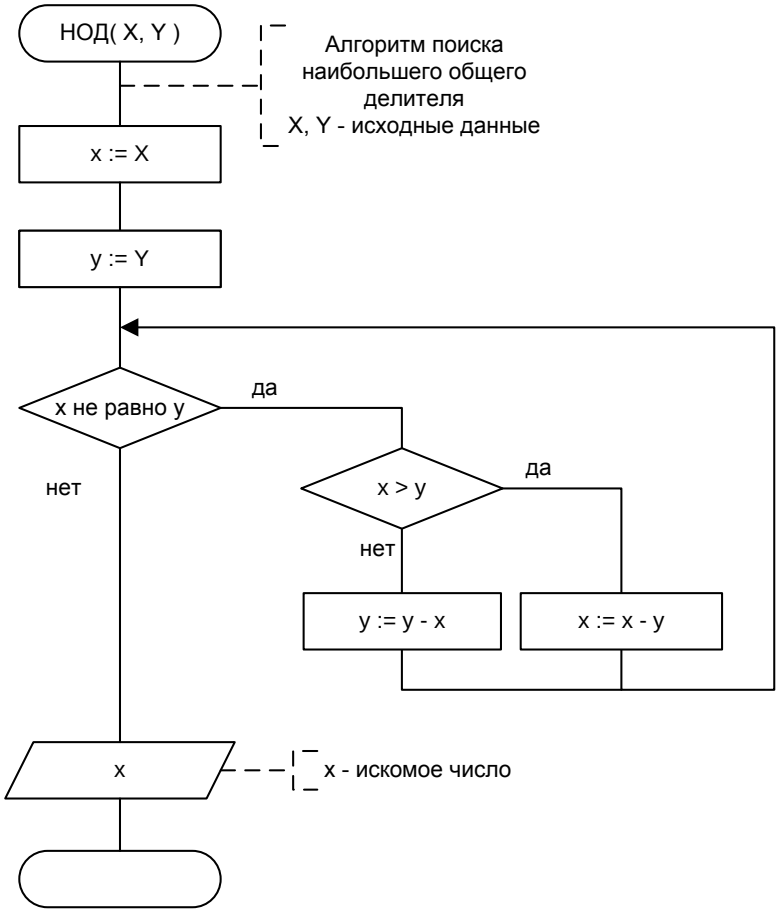
- **Алгоритм** – однозначно определенная на некотором языке конечная последовательность предписаний (инструкций, команд), задающая порядок исполнения элементарных операций для систематического решения задачи
- Любое разумное определение алгоритма, которое может быть предложено в будущем, окажется эквивалентным уже известным определениям

# Формы записи алгоритма

- Текстуальная форма записи
  1. Скопировать значение X во вспомогательную переменную x.
  2. Скопировать значение Y во вспомогательную переменную y.
  3. Если x≠y, перейти к п. 4, иначе – к п. 7.
  4. Если x>y, перейти к п. 5, иначе – к п. 6.
  5. Записать в x результат вычисления выражения x-y и перейти к п. 3.
  6. Записать в y результат вычисления выражения y-x и перейти к п. 3.
  7. Конец. x – результат работы.

# Формы записи алгоритма

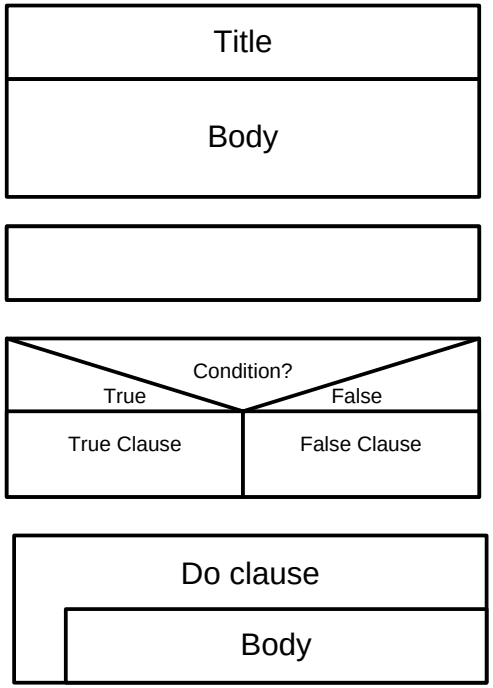
- Схема алгоритма (flowchart)



# Формы записи алгоритма

- Другие диаграммы
  - Диаграммы Нэсси-Шнейдермана (NSD)

*Основные элементы NSD*



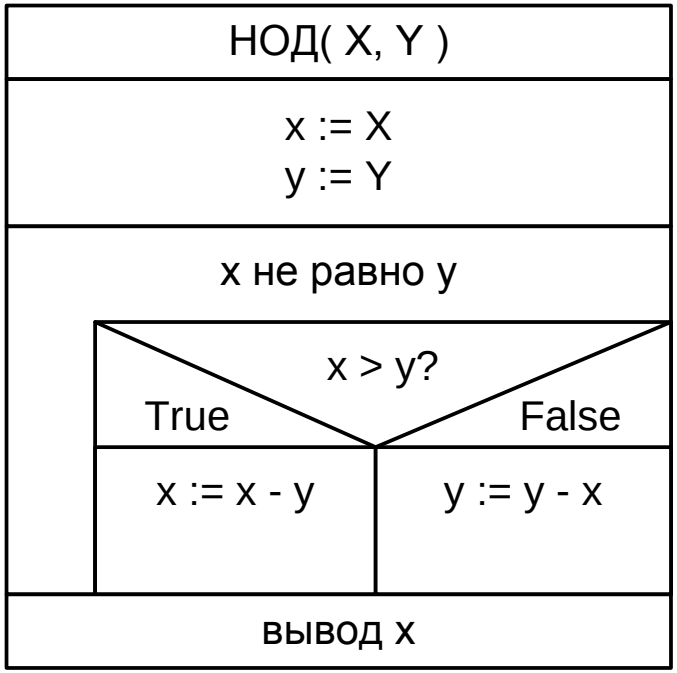

NSD-программа

Процесс

Ветвление

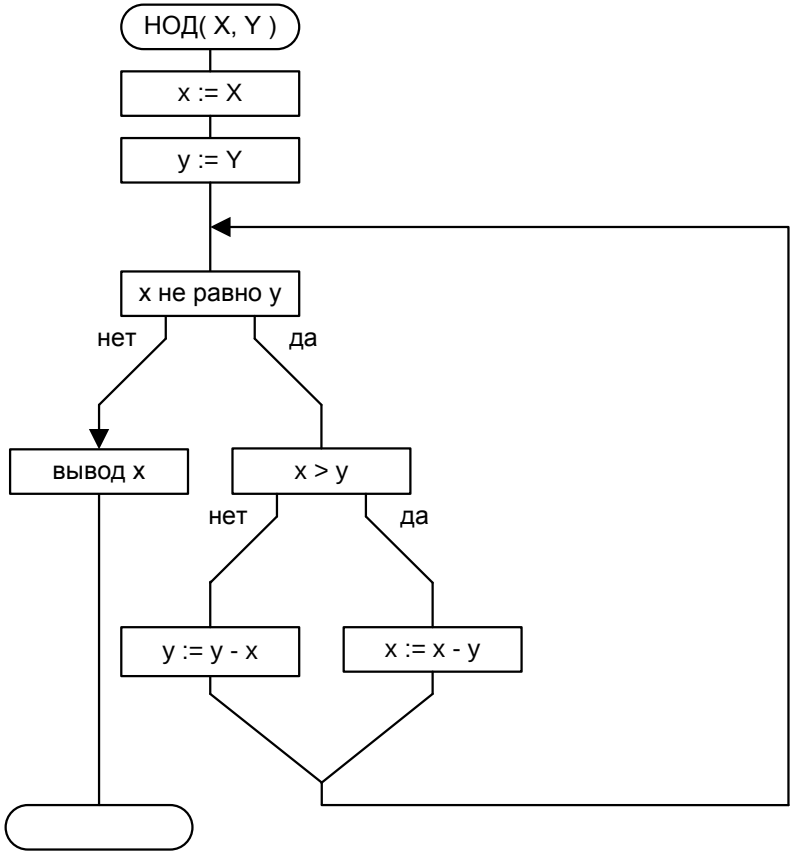
Цикл с предусловием

*NSD-диаграмма алгоритма НОД( X, Y )*



# Формы записи алгоритма

- Другие диаграммы
  - Диаграммы Дейкстры



# Формы записи алгоритма

- Псевдокод

```
НОД( X, Y )
    x:=X;
    y:=Y;
    пока( x ≠ y ) повторять
        если( x > y ) то x:=x-y;
        иначе           y:=y-x;
    конец цикла
    вывести x
конец
```

# Резюме

- Алгоритмы могут быть определены на базе различных текстовых и тексто-графических формализмов
- Формальные модели для проектирования различных классов программных систем
- Современная тенденция: уход от определения алгоритма как первичного этапа проектирования
- Визуализация самого процесса проектирования