

# Язык программирования Си

Приведение типов, операции

Потоковый ввод-вывод

*Югов Иван Олегович  
МОУ Гимназия №10, г. Тверь*

# Приведение типов

Переменной (параметру функции, результату функции и т. п.) присваивается значение, имеющее не её тип:

```
char K = 100; short P = K;
```

```
long long T = 100L;
```

```
float R = 3;
```

```
double Z = R;
```

```
float Bound() {return 10;}
```

Это возможно, т. к. программа произведёт *приведение типов* — преобразует (приведёт) передаваемое значение к требуемому типу.

Приведение типов выполняется автоматически.

# Приведение типов

Возможны проблемы:

```
short P = 500; char K = P;
```

Значение может выходить за рамки назначаемого типа — *переполнение*.

При этом:

- либо генерируется ошибка при выполнении программы (Runtime Error);
- либо в результате получается неверное значение.

*Контролируйте приведение типов.*

# Приведение типов

Приведения, не вызывающие проблем:

- «коротких» целочисленных типов — к более «длинным» с такой же знаковостью (*расширение типа*):

```
short ← char,
```

```
unsigned long long ← unsigned short;
```

- «коротких» вещественных типов — к более «длинным» (*расширение типа*):

```
double ← float, long double ← double;
```

- ЦЕЛЫХ ТИПОВ — К ВЕЩЕСТВЕННЫМ ДОСТАТОЧНОЙ ТОЧНОСТИ:

```
float ← char, float ← short, double ← long.
```

# Приведение типов

Возможны переполнения при приведении:

- «длинных» числовых типов — к более «коротким» (*сужение типа*):

```
char ← long (500), float ← double (7.5E+50);
```

- беззнаковых целочисленных типов — к знаковым целочисленным:

```
signed short ← unsigned short (40000);
```

- знаковых целочисленных типов — к беззнаковым целочисленным:

- `unsigned char ← signed char (-1).`





# Приведение типов

Можно явно указать тип, к которому нужно преобразовать значение  
(*операция приведения типа*):

```
char K = 120; P = (short)K;  
Z = (unsigned long long)40000;  
return (_Bool)F;
```

При приведении числового типа к типу `_Bool` значение 0 остаётся 0, любое другое — преобразуется к значению 1.

*Явное указание типа не предотвращает возможных проблем.*

# Операции

Группы операций (*операторов*):

- *арифметические* (сложение, умножение...);
- *сравнения* («равно», «меньше»...);
- *логические* («не», исключающее «или»...);
- *битовые* (сдвиги, битовые логические...);
- *присваивания*;
- *прочие* (индекс, разыменовывание, приведение типа, инкремент...).



# Свойства операций

**Операнд** — величина, над которой выполняется операция.

По числу операндов операции бывают:

- **унарные** (1 операнд):

`-F, (short) Z, i++;`

- **бинарные** (2 операнда):

`a = b, 7 - t, K >> 4, L & 0xFC;`

- **тернарные** (3 операнда):

`F ? a : b.`

# Свойства операций

По способу записи *унарные* операции бывают:

- *префиксные* — записываются перед операндом:

`!K, (float) Y, ~6, -F;`

- *постфиксные* — записываются после операнда:

`i++, j--.`

# Приоритет

*Приоритет* — свойство операции, влияющее на порядок вычисления её результата по отношению к другим операциям.

Приоритет выражается числом.

Больше число — выше приоритет.

*Операция вычисляется только после того, как над её операндами вычислены все операции с большим приоритетом:*

$z = C + A * (\text{long}) x + B * (\text{long}) y;$   
16 2 16 12 16 13      14      16 12 16 13      14      16

# Приоритет

Порядок вычисления, как правило, такой:

1. Унарные.
2. Арифметика, сдвиги.
3. Логика: сравнения, битовая, обычная.
4. Условие.
5. Присваивания.

Приоритет обычно соответствует «естественному» порядку их понимания.

Приоритет можно менять с помощью скобок  $()$ :  $R = (2 + 2) * 2;$

# Ассоциативность

**Ассоциативность** — свойство операции, означающее порядок вычисления в цепочке операций с таким же приоритетом. Бывает:

- **слева направо**:  $a + 7 - 2 + 6 - f - 2$ ;
- **справа налево**:  $a = b = c = d = 0$ .

**На порядок вычисления операций влияют их приоритет и ассоциативность.**

$z = C + A * (\text{long}) x + B * (\text{long}) y;$

16   2   16   12   16   13   14   16   12   16   13   14   16  
<   >   >   <   >   >   <   >   >   >   <



# Ассоциативность

Ассоциативность операций обычно соответствует «естественному» порядку их понимания:

- Унарные префиксные — справа налево.
- Унарные постфиксные — слева направо.
- Присваивания — справа налево.
- Остальные бинарные — слева направо.

# Арифметические операции

Определены 5 арифметических операций:

- **сложение** (+):  $5 + 6$ ,  $a + b + c$ ,  $1.5 + K$ ;
- **вычитание** (-):  $9.0 - 7.4$ ,  $-6E-3 - A$ ;
- **умножение** (\*):  $10 * a$ ,  $k * x$ ;
- **деление** (/):  $100 / 3$ ,  $Prime / 5$ ,  $4.2 / 0.6$ ;
- **получение остатка от деления (деление по модулю)** (%):  $100 \% 3$ ,  $U \% p$ .

Операнды целые или вещественные (для % — только целые).

Тип результата — «больший» из типов операндов.

Если появляется «минус», то знаковый.

# Смена знака

Определены 2 операции смены знака:

- **плюс (+)**:  $+b$ ,  $+(e - 2)$ ;
- **минус (смена знака) (-)**:  $-Y$ ,  $-(B + 2)$ .

Операнды целые или вещественные.

Тип результата — как у операнда.

Если появляется «минус», то знаковый.

Возможно расширение типа.

# БИТОВЫЙ СДВИГ

Определены 2 операции битового сдвига:

- *сдвиг вправо* ( $\gg$ ):  $A \gg 7$ ,  $0xFF \gg p$ ;
- *сдвиг влево* ( $\ll$ ):  $1 \ll p$ .

Операнды целые.

Тип результата — `int` или `long long`.

Второй операнд может быть отрицательным (сдвиг в обратную сторону).

Сдвиг вправо — арифметический, сохраняет старший бит (знак) операнда.

# БИТОВЫЙ СДВИГ

Сдвиги используются в операциях над битами, а также для быстрого умножения и деления целых чисел на степени двойки:

$$K \ll 4 \quad \sim \quad K * 16$$

$$P \gg 10 \quad \sim \quad P / 1024$$

(для положительных  $P$ )

$$P \gg 10 \quad \sim \quad (P - 1) / 1024$$

(для отрицательных  $P$ )



# Операции сравнения

Определены 6 операций сравнения:

- *равно* (`==`);
- *не равно* (`!=`);
- *больше* (`>`);
- *меньше или равно* (`<=`);
- *меньше* (`<`);
- *больше или равно* (`>=`).

Операнды логического типа (целые — 0 или 1),  
результат логического типа.

У операций «равно» и «не равно» приоритет  
ниже, чем у остальных операций сравнения.

*Не путайте сравнение на равенство (`==`)  
с присваиванием (`=`).*

# Логические операции

Определены 7 логических операций:

Логические:

- «НЕ» (!);
- «И» (& &);
- «ИЛИ» (| |).

Битовые:

- «НЕ» (~).
- «И» (&);
- «ИЛИ» (|);
- *исключающее «ИЛИ»* (^).

Операции «НЕ» — префиксные:  $\sim P$ , ! (K > 5).

Типы операндов и результата  
битовых операций — int или long long,  
логических — логический.

# Приращения

Определены 4 операции приращения:

- *префиксный инкремент* ( $++$ ):  $++P$ ;
- *префиксный декремент* ( $--$ ):  $--P$ ;
- *постфиксный инкремент* ( $++$ ):  $P++$ ;
- *постфиксный декремент* ( $--$ ):  $P--$ .

Увеличивают (уменьшают) значение операнда на единицу. Операнд — только переменная.

Результат того же типа, что и операнд.

# Приращения

Префиксный инкремент (декремент) изменяет значение аргумента и возвращает его новое значение:

$k = 7; p = ++k;$

$k = 8, p = 8$

$k = 7; p = --k;$

$k = 6, p = 6$

Постфиксный инкремент (декремент) возвращает текущее значение аргумента и только потом его изменяет:

$k = 7; p = k++;$

$k = 8, p = 7$

$k = 7; p = k--;$

$k = 6, p = 7$

# Присваивания

Определены 10 операций присваивания:

$=$ ,  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$ ,  $<<=$ ,  $>>=$ ,  $\&=$ ,  $\^=$ ,  $|=$ .

Левый операнд — только переменная.

Сами возвращают присвоенное значение с типом левого операнда:  $a = b + (c = 6) * 2$ .

Присваивания с операциями работают так:

$k += 7$ ;      аналогично       $k = k + 7$ ;

$T *= 7 + e$ ;      аналогично       $T = T * (7 + e)$ ;

Приоритет — низкий.

Ассоциативность — справа налево.



# Условие

Единственная тернарная операция — условие:

*операнд1 ? операнд2 : операнд3*

Если значение первого операнда «истинно» (не ноль), то возвращает второй операнд, иначе возвращает третий операнд:

*если операнд1 , то операнд2 , иначе операнд3*

`max = x > y ? x : y;`

`sign = a == 0 ? 0 : a > 0 ? 1 : -1;`

Ассоциативность — справа налево.

# ПОТОКОВЫЙ ВВОД-ВЫВОД

*Потоковый ввод-вывод доступен в C++.*

Используется заголовочный файл `iostream`.

Для его использования пишем в начале кода:

```
#include <iostream>
```

Также включаем пространство имён:

```
using namespace std;
```

Каждая программа в операционной системе по умолчанию уже может работать с несколькими стандартными потоками.

# ПОТОКОВЫЙ ВВОД-ВЫВОД

**Поток** — способ единообразной работы с файлами, устройствами ввода-вывода и т. п.

Обычно определены 4 стандартных потока:

Название	Имя	Примечание
Стандартный ввод	<code>cin</code>	Клавиатура
Стандартный вывод	<code>cout</code>	Экран
Стандартный вывод ошибок без буферизации	<code>cerr</code>	
Станд. вывод ошибок с буферизацией	<code>clog</code>	

Если не указать пространство имен в заголовке, то придётся это делать перед именами потоков: `std::cin`, `std::cout`.

# ПОТОКОВЫЙ ВВОД-ВЫВОД

Вывод значений констант, переменных и т. п. на экран — с помощью операции <<:

```
cout << a;
```

Ввод значений переменных с клавиатуры — с помощью >>:

```
cin >> K;
```

Можно вводить (выводить) несколько значений:

```
cin >> A >> B >> C;
```

# ПОТОКОВЫЙ ВВОД-ВЫВОД

**Манипуляторы** задают некоторые параметры ввода-вывода.

Манипулятор	Назначение
<code>endl</code>	перевод строки
<code>dec</code>	меняет формат вывода чисел на десятичный
<code>oct</code>	меняет формат вывода чисел на восьмеричный
<code>hex</code>	меняет формат вывода чисел на шестнадцатеричный
<code>showpos</code>	показывает «+» перед неотрицательными числами
<code>showpoint</code>	показывает десятичную точку
<code>noshowpoint</code>	скрывает десятичную точку

Примеры:

```
cin >> oct >> N; cout << dec << p << endl;
```



# ПОТОКОВЫЙ ВВОД-ВЫВОД

Ряд параметров определяется с помощью функций:

Функция	Назначение
<code>width(int x)</code>	минимальное число знаков до следующего вывода
<code>fill(char x)</code>	устанавливает символ-заполнитель и возвращает предыдущий символ-заполнитель; по умолчанию в качестве символа-заполнителя используется пробел
<code>precision(int x)</code>	устанавливает число значащих цифр для чисел с плавающей точкой

Примеры:

```
cout.width(4);
```

```
C = cout.fill();
```