* [Представление целых чисел в C/C++](https://foxford.ru/lessons/32261/conspects/1)

**Представление целых чисел в C/C++**

**Представление беззнаковых целых чисел**

В языках C/C++ для хранения целых чисел выделяется фиксированный размер памяти (в отличии от, например, языка Python, где целые числа имеют произвольную длину, ограниченную только возможностями компьютера). Например, если для хранения целочисленной переменной выделено 4 байта (32 бита), то эта переменная может принимать 232 различных значений. В таких языках программирования, как C/C++ или Pascal бывают два вида целочисленных типов данных — знаковые и беззнаковые типы. В беззнаковом типе данных хранятся только неотрицательные значения. Минимальное беззнаковое значение, которое можно сохранить в такой переменной, записывается нулевыми битами и соответствует числу 0. Максимальное значение, которое можно записать в такой переменной, кодируется одними единицами и для 32-битной переменной равно 232−1. Далее в таблице приведены названия различных беззнаковых целых типов для различных языков программирования.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Разрядность, бит | Максимальное значение | C, C++ (32-битный) | С, C++ (64-битный) | Pascal |
| 16 | 216−1=65535 | unsigned short | unsigned short | word |
| 32 | 232−1≈4,2⋅109 | unsigned int, unsigned long | unsigned int | dword |
| 64 | 264−1≈1,8⋅1019 | unsigned long long | unsigned long, unsigned long long | qword |

**Представление знаковых целых чисел (дополнительный код)**

Для представления знаковых типов данных используется дополнительный код. Такое представление данных удобно для реализации арифметических операций с целыми числами. В дополнительном коде число -1 кодируется одними единичными битами. В этом случае при увеличении числа -1 на 1, как если бы это увеличение происходило в двоичной системе счисления «в столбик», все биты обнулятся и произойдет перенос в старший разряд, который не может быть сохранен в переменной данного типа и получится число 0. Аналогично, число -2 представляется в памяти, как 11…110, и при увеличении этого числа на 1 получится число -1. Число -3 кодируется, как 11…101 и т. д. Минимальное значение, которое может быть в данном случае записано в переменной знакового типа представляется в памяти, как 100…000, оно соответствует значению −231 для 32-битного типа, это — минимальное отрицательноые число, представимое в данном типе. А значение 011…11 соответствует положительному числу 231−1, и это максимальное положительное число, которое представимо в таком типе.

Также можно заметить, что для преставления отрицательного числа −n необходимо взять двоичное представление числа n, уменьшить его на 1, затем инвертировать все биты числа (заменить значение 0 на 1 и заменить 1 на 0). Такая операция также назыается дополнением, поэтому и код называется дополнительным.

Ниже приведена таблица различных знаковых типов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Разрядность, бит | Минимальное значение | Максимальное значение | C, C++ (32-битный) | С, C++ (64-битный) | Pascal (16-битный) | Pascal (32-битный) |
| 16 | −215=−32768 | 215−1=32767 | short | short | shortint, int | shortint |
| 32 | −231≈−2,1⋅109 | 231−1≈2,1⋅109 | int, long | int | longint | int, longint |
| 64 | −263≈−9,2⋅1018 | 263−1≈9,2⋅1018 | long long | long, long long | int64 | int64 |

**Битовые операции**

С целочисленными типами данных возможно осуществление битовых операций, которые применяют соответствующую логическую операцию ко всем соответствующим битам числа.

|  |  |  |
| --- | --- | --- |
| Название операции | C, С++, Python | Pascal |
| Битовое И | & | and |
| Битовое ИЛИ | | | or |
| Битовое исключающее ИЛИ | ^ | xor |
| Битовое отрицание | ~ | not |
| Битовый сдвиг влево | << | shl |
| Битовый сдвиг вправо | >> | shr |

Например, пусть даны два числа:

**int a = 5;**

**int b = 6;**

Их представления в двоичной системе счисления будут такими:

a = 00…00101

b = 00…00110

Если к каждой паре соответствующих бит применить операцию "И", то получится число, имеющее двоичное представление 00…00100, то есть число 4. Поэтому следующий код:

**cout << (a & b) << endl;**

выведет число 4.

Аналогично, в результате применения побитовой операции "ИЛИ" получится двоичное 00…00111, то есть 7,

**cout << (a | b) << endl;**

выведет число 7.

А применение операции XOR (исключающее или)

**cout << (a ^ b) << endl;**

даст число 3, то есть двоичное 00…00011.

Побитовое отрицание даст число, которое в дополнительном коде будет отрицательным. Например, битовое отрицание числа a даст двоичное число 11…11010, что соответствует в дополительном коде числу -6.

**cout << ~a << endl;**

выведет -6.

Битовый сдвиг влево на 1 равносилен операции умножения на 2, а битовый сдвиг влево на k равносилен умножению на 2k. Например, следующий код

**cout << (a << 2) << endl;**

выведет число 24. Обратите внимание, при выводе через cout также используются знаки "<<", поэтому мы поставим скобки, чтобы не путать операцию битового сдвига с выводом.

Битовый сдвиг вправо числа на k равносилен делению на 2k с отбрасыванием дробной части. Например,

**cout << (a >> 1) << endl;**

выведет число 3, а

**cout << (a >> 2) << endl;**

выведет число 1.