**Логические значения. Тип boolean**

Примитивный тип boolean предназначен для хранения логических значений. Данный тип может принимать одно из двух возможных значений: true (истина) или false (ложь).

Значения boolean возвращаются во всех логических операциях (например, операции сравнения). Применяется при построении циклов.

**Символьный тип данных**

Символьный тип данных представляет собой один 16-битный Unicode символ. Unicode — это объединение десятков кодировок символов, он включает в себя латинский, греческий, арабский алфавиты, кириллицу и многие другие наборы символов.

Диапазон допустимых значений - от 0 (‘\ u0000’)  до 65536 (‘\ uffff’). Отрицательных значений для типа char не существует. Это единственный беззнаковый целочисленный тип данных в Java.

Символы char  можно задавать при помощи символа, заключенного в одинарные кавычки, а можно с помощью соответствующего кода-числа из таблицы Unicode-символов. Например символ ‘Ы’ соответствует числу  1067.

**Преобразование типов**

Иногда возникают ситуации, когда у вас есть величина какого-то определенного типа, а вам нужно ее присвоить переменной другого типа. Для некоторых типов это можно проделать и без приведения типа, в таких случаях говорят об автоматическом (неявном) преобразовании типов. В других случаях программист должен сам провести преобразование типа к нужному — это называется явным преобразованием типов.

**Автоматические (неявные) преобразования**

При изучении типов данных уже говорилось о том, что целочисленные типы данных byte и short при вычислении Java всегда приводит к типу int, а тип float к типу double. Фактически это и есть неявное преобразование типов.

Стрелками на рисунке показано, какие преобразования типов могут выполняться автоматически без потери точности. Пунктирными стрелками показаны автоматические преобразования с потерей точности.

Преобразования от меньшего типа к большему называются расширяющими и именно эти преобразования проходят без потери данных и без потери точности.

Можно сказать, что существуют следующие правила повышения типов

Если в выражении используются переменные типов byte, short и int, то во избежание переполнения тип всего выражения автоматически повышается до int. Если же в выражении тип хотя бы одной переменной - long, то и тип всего выражения тоже повышается до long.

Если выражение содержит операнды типа float, то и тип всего выражения автоматически повышается до float. Если же хотя бы один из операндов имеет тип double, то тип всего выражения повышается до double. По умолчанию Java рассматривает все литералы с плавающей точкой, как имеющие тип double.

Преобразования от большего типа к меньшему называются сужающими и в этих случаях вся ответственность по переносу данных ложится на программиста.

Явные преобразования

Все остальные преобразования типов, не рассмотренные ранее, осуществляются явным образом, а именно указанием нужного типа, заключенного в круглые скобки. Обычно это сужающие преобразования от типа с большей разрядностью к типу с меньшей разрядностью. При явном преобразовании программист должен внимательно оценить все риски потери данных и выбрать тип, который сведет эти риски к нулю.

```java

//Пример кода

byte b = 50;

b = b\* 2;

b = (byte) (b\* 2);

Ошибка: Error:(5, 14) java: incompatible types: possible lossy conversion from int to byte

При преобразовании значений с плавающей точкой к целочисленным значениям, происходит усечение дробной части.

```java

//Пример кода

double a = 56.9898;

int b = (int)a;

System.out.println(b); // 56

```

## Арифметические операции

Большинство операций в Java являются стандартными математическими операциями. Есть унарные операции (выполняются над одним операндом), бинарные - над двумя операндами, а также тернарные - выполняются над тремя операндами. Операндом является переменная или значение (например, число), участвующее в операции.

### Унарные арифметические операции

Унарные арифметические операции — это операции, которые производятся над одним числом:

++ (инкремент) и -- (декремент). Инкремент подразумевает под собой увеличение значения переменной на единицу, декремент — наоборот, уменьшение значения переменной на единицу.

Каждая из операций имеет две разновидности: префиксная и постфиксная. Префиксный тип операции означает, что сначала значение переменной увеличивается на 1, а затем это значение переменной передается дальше. Постфиксный тип операции означает, что сначала значение переменной передается дальше, а затем увеличивается на 1.

`

### Бинарные арифметические операции

К бинарным арифметическим операциям относятся:

\* вычитание («-»)

\* сложение («+»)

\* умножение («\\*»)

\* деление («/»)

\* остаток от деления («%»)

### Приоритет арифметических операций

Одни операции имеют больший приоритет чем другие и поэтому выполняются вначале. Операции в порядке уменьшения приоритета:

\* ++ (инкремент), -- (декремент)

\* \\* (умножение), / (деление), % (остаток от деления)

\* \+ (сложение), - (вычитание)

Приоритет операций следует учитывать при выполнении набора арифметических выражений. Чтобы задать свой порядок вычислений, нужно использовать ().

```java

//Пример кода

int a = 8;

int b = 7;

int c = a + 5 \* ++b;

System.out.println(c);// 48

int d = (a + 5) \* ++b;

System.out.println(d);// 104

```

### Операции с присваиванием

Операцию присваивания можно скомбинировать с любой арифметической операцией и записать сокращенно, в виде рядом стоящих знаков присваивания и используемой арифметической операции.

```java

//Пример кода

int c = 3;

c += 5; //8

c -= 2; //6

c /= 3; //2;

```

Для выполнения различных математических операций в Java в пакете java.lang определен класс Math.

В классе определены две константы типа double: E и PI.

И большое многообразие математических функций:

| Тип | Метод | Описание |

|---------|-------------------------|-------------------------------------------------------------------------|

| double | abs(double a) | Возвращает абсолютное значение (модуль) числа типа double |

| double | exp(double a) | Возвращает экспоненту аргумента |

| double | max(double a, double b) | Возвращает больший из аргументов |

| double | random() | Возвращает случайное число от 0.0 (включительно) до 1 (не включительно) |

| double | sin(double a) | Возвращает синус аргумента |

| double | toRadians() | Преобразует градусы в радианы |

**Поразрядные операции**

Всем известно, что информация представляется в компьютере в двоичном виде нулями и единицами, битами. Однако, доступ к конкретным нулям и единицам невозможен. Прочитать или записать можно только сразу байт, а команд для чтения и записи бита не существует.

Байт — это минимальная единица адресации компьютерной памяти. Для большинства современных компьютеров это восемь бит.

Но есть команды, которые работают с каждым битом числа. Например, возможно заменить все биты в числе на противоположные. Если у нас есть два байта, то мы можем сделать логическую операцию И, ИЛИ или исключающее ИЛИ с каждой парой битов этих двух байтов. Это так называемые поразрядные операции.

Рассмотрим их подробно и сразу поговорим о применениях.

**Побитовое отрицание (NOT)**

Обозначается символом ~

Это унарная операция, которая меняет каждый бит на противоположный.

```java

int i = 123;

System.out.println(~i); // -124

**Побитовое И (AND)**

Обозначается символом &

Эта операция выставляет значение бита в 1 только в том случае, если соответствующие биты в первом и втором числе равны 1 одновременно.

Побитовое И зачастую применяют для того, чтобы узнать какое значение 0 или 1 стоит в определенном бите числа.

Тот бит, значение которого необходимо узнать, устанавливается в 1, остальные в 0. Полученное таким образом число называют маской.

Выполняют операцию побитового И между числом и маской.

Результат даст информацию о конкретном бите числа.

& 01100101 // 101

00000100 // 4

\_\_\_\_\_\_\_\_

00000100 // 4

& 01100001 //97

00000100 // 4

\_\_\_\_\_\_\_\_

00000000 // 0

Пример 1.7

Пусть нужно узнать 0 или 1 содержится во втором бите числа 101.

```java

int i = 101;

int mask = 4;

int result = i & mask;

System.out.println(result > 0 ? "YES" : "NO"); \\ YES

```

**Побитовое ИЛИ (OR)**

Операция обозначается символом |

Выставляет значение 1, если хотя бы один соответствующий бит в первом или во втором числе установлен в 1.

При помощи масок можно не только узнать, чему равно значение конкретного бита числа, но и установить его значение:

| 01100101 // 101

00000100 // 4

\_\_\_\_\_\_\_\_

01100101 // 101

| 01100001 // 97

00000100 // 4

\_\_\_\_\_\_\_\_

01100101 // 101

```java

int i = 10;

int mask = 4;

i | mask;

```

**Исключающее ИЛИ (XOR)**

Обозначается символом ^

Эта операция выставляет значение в 1, если соответствующий бит равен 1 только в одном из операндов, но не одновременно.

^ 01100101 // 101

00000100 // 4

\_\_\_\_\_\_\_\_

01100001 // 94

^ 01100001 // 97

00000100 // 4

\_\_\_\_\_\_\_\_

01100101 // 101

Операция XOR имеет много практических применений. Самое известное и часто применяемое из которых — шифрование. Если примерить эту операцию к значению с каким-то ключом (другим значением), первое изменится, но при повторном применении восстанавливается.

Например:

5 ^ 7 → 2 (значение зашифровалось)

2 ^ 7 → 5 (используя ключ, можно расшифровать)

Cимволы текста можно рассматривать как их числовые коды.

Идея шифрования при помощи XOR проста: выполним операцию XOR каждого символа c ключом. Текст станет нечитаемым — зашифруется. Получатель, знающий ключ, сможет легко восстановить текст, опять выполняя операцию XOR c каждым символом с тем же ключом. Конечно, ключ можно подобрать. Но систему можно улучшить, если выбирать для каждого шифруемого символа новый ключ из известной передающему и принимающему последовательности. Например, можно шифровать при помощи текста этого пособия. В этом случае пособие выступает в роли некого сложного ключа. Расшифровка будет возможна, только если злоумышленник будет знать, как зашифровано сообщение и иметь это пособие. Такой способ кодирования называется «одноразовый блокнот».

**Знаковый оператор сдвига влево**

Обозначаются символами <<

Операция, в результате которой все биты первого операнда смещаются влево и число справа дополняется нулем. Это операция повторяется столько раз, сколько задано вторым операндом. Один сдвиг влево соответствует умножению на 2.

0001 << 2 = 0100 = 4

```java

int i = 5 << 3;

System.out.println(i); // 40

```

**Знаковый оператор сдвига вправо**

Обозначается символами >>

Операция, в результате которой все биты первого операнда смещаются вправо и число слева дополняется нулем, если число положительное, и единицей, если отрицательное. Это повторяется столько раз, сколько задано вторым операндом. Один сдвиг вправо соответствует делению на 2. Если делится нечетное число, то округляется до меньшего целого для положительных чисел, а для отрицательных округляется до большего.

4 = 0100 >> 2 = 0001 = 1

3 = 0011 >> 1 = 0001 = 1

-3 = 1111 1101 >> 1 = 1111 1110 = -2

```java

System.out.println(42 >> 2); // 10

System.out.println(-42 >> 2); // -11

```

**Беззнаковый оператор сдвига**

Операция, в результате которой все биты первого операнда смещаются вправо и число слева дополняется нулем, даже если операция выполняется с отрицательными числами. Операция повторяется столько раз, сколько задано вторым операндом.

Отсюда и название оператора — беззнаковый. В результате применения оператора всегда получается положительное число, так как в Java левый бит отвечает за знак числа. Операция так же, как и знаковый оператор сдвига вправо, соответствует делению числа на два за исключением первого сдвига в отрицательном числе.

4 = 0100 >>> 2 = 0001 = 1

3 = 0011 >>> 1 = 0001 = 1

-3 = 1111 1101 >> 1 = 0111 1110 = 2147483646

```java

System.out.println(-123 >>> 1); // 2147483586

int a = sc.nextInt();

int i = sc.nextInt();

//Обнуление последних i-ых битов в числе

//int b = (a >> i) << i;

//Устанавливаем i-ый бит в единицу

//int b = a | (1 << i);

//Инвертируем i-ый бит

//int b = a ^ (1 << i);

//Обнуляем все биты числа, кроме последних i-ых

//int b = a & ((1 << i) - 1);

if ((a & (a - 1)) == 0) {

System.out.println("YES");

} else {

System.out.println("NO");

}

}

//умножение битов

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int b = (1 << n);

System.out.println(b);