

[illegible]

Контрольные вопросы

- 6 классических операций сравнения
- Логические операции
- Что такое инструкция? Что такое блок?
- Как работает оператор if?
- Чем отличается набор ифов от лесенки?
- Тернарный оператор и switch

Что такое система счисления

Система счисления – это способ представления чисел и соответствующие ему правила действий над числами. **Система счисления** – это знаковая система, в которой числа записываются по определенным правилам с помощью символов некоторого алфавита, называемых **цифрами**.

Пример систем счисления

Например, вы видите перед собой несколько деревьев. Ваша задача — их посчитать. Для этого можно — загибать пальцы, делать зарубки на камне (одно дерево — один палец\зарубка) или сопоставить 10 деревьям какой-нибудь предмет, например, камень, а единичному экземпляру — палочку и выкладывать их на землю по мере подсчёта.

Современные системы счисления

Система счисления	Основание	Алфавит цифр
Десятичная	10	0,1,2,3,4,5,6,7,8,9
Двоичная	2	0,1
Восьмеричная	8	0,1,2,3,4,5,6,7
Шестнадцатеричная	16	0,1,2,3,4,5,6,7,8,9, A(10),B(11),C(12), D(13),E(14),F(15)

Двоичная система счисления

Эта система, в основном, используется в вычислительной технике.

Она была создана задолго до изобретения вычислительных машин и уходит “корнями” в цивилизацию Инков, где использовались кипу — сложные верёвочные сплетения и узелки.



Двоичная система

1011101

Двоичная позиционная система счисления имеет основание 2 и использует для записи числа 2 символа (цифры): 0 и 1.

В каждом разряде допустима только одна цифра — либо 0, либо 1.

Двоичная система

Примером может служить число **101**. Оно аналогично числу **5** в десятичной системе счисления. Для того, чтобы перевести из 2-й в 10-ю необходимо умножить каждую цифру двоичного числа на основание **2**, возведенное в степень, равную разряду. Таким образом, число $101_2 = 1*2^2 + 0*2^1 + 1*2^0 = 4+0+1 = 5_{10}$.

Восьмеричная система счисления

Восьмеричная система также чаще всего используется в областях, связанных с цифровыми устройствами. Характеризуется лёгким переводом восьмеричных чисел в двоичные и обратно, путём замены восьмеричных чисел на триплеты двоичных. Широко использовалась в программировании и компьютерной документации, однако позднее была почти полностью вытеснена шестнадцатеричной системой. На данный момент восьмеричная система применяется при выставлении прав доступа к файлам и прав исполнения для участников в ОС Linux.

Пример

Пример восьмеричного числа: 254.

Для перевода в 10-ю систему необходимо каждый разряд исходного числа умножить на 8^n , где n — это номер разряда. Получается, что $254_8 = 2 \cdot 8^2 + 5 \cdot 8^1 + 4 \cdot 8^0 = 128 + 40 + 4 = 172_{10}$.

Шестнадцатеричная система счисления

Шестнадцатеричная система широко используется в современных компьютерах, например при помощи неё указывается цвет: `#FFFFFF` — белый. Рассматриваемая система имеет основание `16` и использует для записи числа: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F`, где буквы равны 10, 11, 12, 13, 14, 15 соответственно.

Другие системы

Шестидесятеричная система счисления — позиционная система счисления по целочисленному основанию 60. Изобретена шумерами в III тысячелетии до н.э., использовалась в древние времена на Ближнем Востоке.

𐎶 1	𐎵𐎶 11	𐎶𐎵 21	𐎶𐎶𐎶 31	𐎶𐎶𐎶𐎶 41	𐎶𐎶𐎶𐎶𐎶 51
𐎶𐎶 2	𐎶𐎶𐎶 12	𐎶𐎶𐎶𐎶 22	𐎶𐎶𐎶𐎶𐎶 32	𐎶𐎶𐎶𐎶𐎶𐎶 42	𐎶𐎶𐎶𐎶𐎶𐎶𐎶 52
𐎶𐎶𐎶 3	𐎶𐎶𐎶𐎶 13	𐎶𐎶𐎶𐎶𐎶 23	𐎶𐎶𐎶𐎶𐎶𐎶 33	𐎶𐎶𐎶𐎶𐎶𐎶𐎶 43	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 53
𐎶𐎶𐎶𐎶 4	𐎶𐎶𐎶𐎶𐎶 14	𐎶𐎶𐎶𐎶𐎶𐎶 24	𐎶𐎶𐎶𐎶𐎶𐎶𐎶 34	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 44	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 54
𐎶𐎶𐎶𐎶𐎶 5	𐎶𐎶𐎶𐎶𐎶𐎶 15	𐎶𐎶𐎶𐎶𐎶𐎶𐎶 25	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 35	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 45	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 55
𐎶𐎶𐎶𐎶𐎶𐎶 6	𐎶𐎶𐎶𐎶𐎶𐎶𐎶 16	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 26	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 36	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 46	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 56
𐎶𐎶𐎶𐎶𐎶𐎶𐎶 7	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 17	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 27	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 37	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 47	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 57
𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 8	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 18	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 28	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 38	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 48	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 58
𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 9	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 19	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 29	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 39	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 49	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 59
𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 10	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 20	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 30	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 40	𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶𐎶 50	



Побитовые операции

Побитовые операторы воспринимают операнды как последовательность из 32 битов (нулей и единиц). Они производят операции, используя двоичное представление числа, и возвращают новую последовательность из 32 бит (число) в качестве результата. Побитовые операции применяются для быстрого выполнения вычислений и меньшего потребления ресурсов, связанных с этими вычислениями.

Зачем они вообще нужны???

- Реализация криптографических алгоритмов, вычисление хэша
- Алгоритм генерации случайных чисел
- Экономия оперативной памяти
- Работа с битовыми коллекциями
- Реализация любых алгоритмов, требующих работу с битами

Техническое применение

Битовые операции применяются совсем не часто, но нужно быть готовым к тому, что однажды они встретятся. Обычно, с их помощью выполняют:

- проверки определённого бита на 0 или 1
- установки 0 или 1 в указанный бит, инвертирования
- умножения/целочисленного деления на 2 и выделения отдельных битов

Так, например, в сетевых интернет-технологиях операция И между значением IP-адреса и значением маски подсети используется для определения принадлежности данного адреса к подсети.

Побитовые операции

	ИЛИ (OR, дизъюнкция)
&	И (AND, конъюнкция)
^	ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)
~	унарный оператор NOT

<<	сдвиг влево
>>	сдвиг вправо

Побитовое ИЛИ (OR)

Выставляет значение в 1, если установлен соответствующий бит в первой или во второй

последовательности, или вместе

00000000 01111011 (123) |

00000001 11001000 (456)

=

00000001 11111011 (507)

Побитовое И (AND)

Обозначается символом **&**

Выставляет значение в 1, если установлены соответствующие биты в первой и второй последовательности **одновременно**

00000000 01111011 (123)

&

00000001 11001000 (456)

=

00000000 01001000 (57)

Исключающее ИЛИ (XOR)

Обозначается символом \wedge

Выставляет значение в 1, если установлен соответствующий бит или в первой или во второй во второй последовательности, но НЕ одновременно.

Если используется более двух последовательностей, то в результате будет единица тогда, когда общее количество единиц соответствующей позиции нечетное.

Пример XOR

00000000 01111011 (123)

^

00000001 11001000 (456)

=

00000001 10110011 (435)

Побитовое отрицание (NOT)

Обозначается символом \sim

Унарный оператор, т.е. применяется к одной последовательности. Превращает каждый бит в противоположный.

\sim

00000000 01111011 (123)

=

11111111 10000100 (-124)

Знаковый оператор сдвига влево <<

Все биты смещаются влево. Число справа дополняется нулём. Операция используется для быстрого умножения на 2. Если оператор применяется к числу, умножение на 2 которого будет больше максимального значения int (2147483647), то в результате будет отрицательное число. Это происходит потому, что крайний левый бит, который отвечает за знак числа, выставляется в единицу, что соответствует отрицательным числам.

11111111 11111111 11111111 10000101 (-123) << 1

11111111 11111111 11111111 00001010 (-246)

Знаковый оператор сдвига вправо >>

Все биты смещаются вправо. Число слева дополняется нулём, если число положительное и единицей, если отрицательное. Операция используется для быстрого деления на 2. Если делится нечётное число, то остаток отбрасывается для положительных чисел и сохраняется для отрицательных.

11111111 11111111 11111111 10000101 (-123) >> 1

11111111 11111111 11111111 11000010 (-62)

Когда количество сдвигов превышает количество разрядов

При использовании битовых сдвигов важно помнить, что когда количество сдвигов достигает количества разрядов, следующий сдвиг вернёт значение в исходное положение. Например, сдвиг влево:

0 - 000000000000000000000000000000001111011 (123)

1 - 0000000000000000000000000000000011110110 (246)

...

30 - 110000000000000000000000000000000000 (-1073741824)

31 - 100000000000000000000000000000000000 (-2147483648)

32 - 000000000000000000000000000000001111011 (123)

Приведение чисел к соответствующему типу данных

При использовании побитовых операций с типами данных `byte/short`, числа сначала приводятся к типу `int`, а если одно из чисел — `long`, то к `long`.

При сужении типа данных, левая часть битов просто отбрасывается.

Использование маски

Одним из приёмов работы с битовыми данными является использование масок. Маска позволяет получать значения только определённых битов в последовательности. Например, у нас есть маска **00100100**, она позволяет нам получать из последовательности только те биты, которые в ней установлены. В данном случае это 3-й и 7-й разряд. Для этого достаточно выполнить побитовое И с нашей маской и неким числом:

$$01010101 \ \& \ 00100100 = 00000100$$

Хранение в одной целочисленной переменной нескольких значений

При помощи битовых сдвигов можно хранить в одной целочисленной переменной несколько значений меньшей длины. Например, в первых нескольких битах можно хранить одно число, в следующих битах — другое. Требуется только знать, на сколько бит выполняется сдвиг и сколько бит занимает хранимое число. Для записи используется логическое ИЛИ, для получения — И.

Пример хранения

```
int age, height, weight, combined, mask;  
age = 29; // 00011101  
height = 185; // 10111001  
weight = 80; // 01010000  
combined = (age) | (height << 8) | (weight << 16);  
//00000000 01010000 10111001 00011101  
mask = 0b11111111; // двоичный литерал  
printf("Age: %d, height: %d, weight: %d", mask &  
combined, mask & combined >> 8, mask & combined >>  
16); //Age: 29, height: 185, weight: 80
```

Практика

- Сохранить в переменной `combined` не только информацию о возрасте, росте и весе, но ещё и о количестве зубов (1-32), поле (0-1), знании C++ (0-1).

Обмен переменных местами без использования временной переменной

Исключающее ИЛИ может быть использовано для обмена двух переменных без создания временной переменной:

```
int x = 10, y = 15;
```

```
x = x ^ y;
```

```
y = y ^ x;
```

```
x = x ^ y;
```

Работа с правами доступа

Принцип следующий: имеется последовательность из трех битов, где:

- 001 — первый бит отвечает за права на выполнение
- 010 — второй — запись
- 100 — третий — чтение

Работа с правами доступа

Имеем следующие константы.

```
const int EXECUTE = 1; //001
```

```
const int WRITE = 2; //010
```

```
const int READ = 4; //100
```

Допустим, нам требуется дать пользователю полный доступ к ресурсу. Для этого должен быть выставлен каждый бит:

```
int usersAccess = EXECUTE | WRITE | READ;
```

```
// получили значение 7 (111)
```


Работа с правами доступа

А теперь, допустим, что нам надо забрать у пользователя права на выполнение:

```
usersAccess = usersAccess ^ EXECUTE;  
// получили значение 6 (110)
```

Быстрое умножение и деление

Операции сдвига иногда используются для быстрого умножения или деления целых чисел на числа, равные степени двойки. Например, выражение $3 \ll 4$ соответствует умножению тройки на 2 в 4-й степени.

Спасибо за внимание!

