

16.09.2024

Данные в С. Системы счисления. Простые числа. Асимптотика.

Филиппов Михаил Витальевич

m.filippov@g.nsu.ru

89232283872

Императивное программирование, 2024-2025

N * Новосибирский
государственный
университет
***НАСТОЯЩАЯ НАУКА**

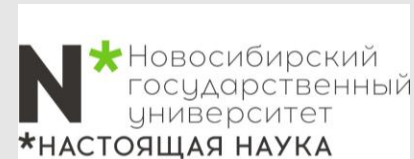


Давайте познакомимся



Филиппов Михаил Витальевич

- Окончил магистратуру ФФ НГУ
- Окончил аспирантуру ИТ СО РАН
- Являюсь м.н.с. ИТ СО РАН
- 7+ лет опыт в программировании C/C++



Адженда

**Системы
счисления**

30 минут

**Данные в С
(продолжение)**

30 минут

Простые числа

5 минут

Асимптотика

25 минут

Адженда

**Системы
счисления**

30 минут

**Данные в С
(продолжение)**

30 минут

Простые числа

5 минут

Асимптотика

25 минут

Что за эти ваши системы счисления?

Система счисления — это способ записи (представления) чисел.

Системы счисления подразделяются на **позиционные** и **непозиционные**, а позиционные, в свою очередь, — на **однородные** и **смешанные**.



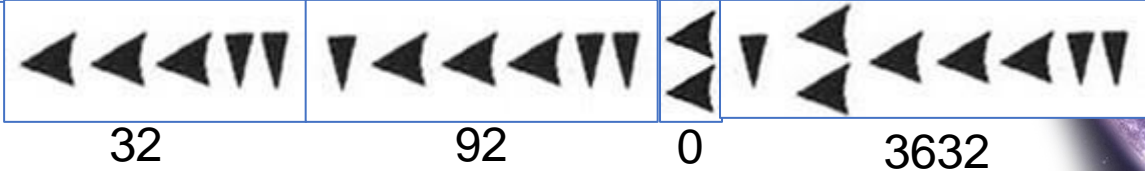
Примеры всех видов систем

Непозиционные:

Древнеегипетская десятичная система



Вавилонская шестидесятеричная система



Римская система – I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 и M = 1000
XXXII=(X+X+X)+(I+I)=30+2=32; CDXLIV = (D-C)+(L-X)+(V-I) = 400+40+4=444.

Позиционные однородные системы:

Десятичная, повседневная для нас (555); двоичная – в компьютерах (1100)

Позиционные смешанные системы:

Время: 23:50:11.0

Формула позиционных однородных систем

Чтобы перевести число в некоторую систему счисления с основанием M (цифрами $0, \dots, M-1$), иначе говоря, в M -ичную СС, нужно представить его в виде:

$$C = a_n * M^n + a_{n-1} * M^{n-1} + \dots + a_1 * M + a_0.$$

$a_{1..n}$ - цифры числа, из соответствующего диапазона. a_n - первая цифра, a_0 - последняя. Сравните эту запись с представлением числа, например, в десятичной системе



Как переводить между однородными системами? Примеры.

- $010\ 011\ 110_2 = (0*2^2+1*2^1+0*2^0) (0*2^2+1*2^1+1*2^0) (1*2^2+1*2^1+0*2^0) = 236_8$
- $101_2 = 1*2^2 + 0*2^1 + 1*2^0 = 4+0+1 = 5_{10}$
- $101,011_2 = (1*2^2 + 0*2^1 + 1*2^0), (0*2^{-1} + 1*2^{-2} + 1*2^{-3}) = (5), (0 + 0,25 + 0,125) = 5,375_{10}$
- $1001,01_2 = 001\ 001, 010 = (0*2^2 + 0*2^1 + 1*2^0) (0*2^2 + 0*2^1 + 1*2^0), (0*2^2 + 1*2^1 + 0*2^0) = (0+0+1) (0+0+1), (0+2+0) = 11,2_8$

Как переводить между однородными системами? Примеры.

$10,625_{10}$ в двоичную систему:

$$0,625 \cdot 2 = 1,25$$

$$0,250 \cdot 2 = 0,5$$

$$0,5 \cdot 2 = 1,0$$

Записав все остатки сверху вниз, получаем $10,625_{10} = (1010), (101) = 1010,101_2$

Переведем десятичное число 934 в шестнадцатеричную систему счисления.

$$\begin{array}{r|l} 934 & 16 \\ \hline 928 & 58 \\ \hline 6 & 48 \\ & 10 \end{array} \quad \begin{array}{r|l} 16 & \\ \hline 3 & \end{array}$$

$$934 = 3A6_{16}$$

Системы счисления с необычными основаниями: -2

Использование системы счисления по основанию -2 дает возможность выражать как положительные, так и отрицательные числа без явного указания их знака

$$(a_n...a_3a_2a_1a_0)=a_n(-2)^n +...+a_3(-2)^3 +a_2(-2)^2 +a_1(-2)^1 +a_0$$

-3	-2	
2	1	$-3 = -2*2 + 1$
-1	0	$2 = -2*(-1) + 0$
1	1	$-1 = -2*1 + 1$
0	1	$1 = -2*0 + 1$

N (-2)	N(10)	N(10)	N(-2)	-N(-2)
0	0	0	0	0
1	1	1	1	11
10	-2	2	110	10
11	-1	3	111	1101
100	4	4	100	1100
101	5	5	101	1111
110	2	6	11010	1110
111	3	7	11011	1001
1000	-8	8	11000	1000
1001	-7	9	11001	1011
1010	-10	10	11110	1010
1011	-9	11	11111	110101
1100	-4	12	11100	110100
1101	-3	13	11101	110111
1110	-6	14	10010	110110
1111	-5	15	10011	110001

Системы счисления с необычными основаниями: -2

Диапазон представляет собой

от $2^n - (2^{n+1} - 2)/3$ до $2^n + (2^n - 1)/3$

или

от $(2^n - 1)/3 + 1$ до $(2^{n+2} - 1)/3$.

Сложение и вычитание:

$$0 + 1 = 1$$

$$1 - 1 = 0$$

$$\text{т.к. } 2 = 110, -1 = 11$$

$$1 + 1 = 110$$

$$11 + 1 = 0$$

$$1 + 1 + 1 = 111$$

$$0 - 1 = 11$$

$$11 - 1 = 10$$

$$\begin{array}{r}
 11 \quad 1 \quad 11 \quad \quad 11 \\
 \quad \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad \quad 19 \\
 + \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad +(-11) \\
 \hline
 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad \quad 8
 \end{array}$$

$$\begin{array}{r}
 1 \quad 11 \quad 1 \quad \quad 1 \\
 \quad \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad \quad 21 \\
 - \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad -(-38) \\
 \hline
 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad \quad 59
 \end{array}$$

Системы счисления с необычными основаниями: $-1 + i$

$N(-1 + i)$	$N(10)$	$N(10)$	$N(-1 + i)$	$-N(-1 + i)$
0	0	0	0	0
1	1	1	1	11101
10	$-1 + i$	2	1100	11100
11	i	3	1101	10001
100	$-2i$	4	111010000	10000
101	$1 - 2i$	5	111010001	11001101
110	$-1 - i$	6	111011100	11001100
111	$-i$	7	111000000	11000001
1000	$2 + 2i$	8	111000001	11000000
1001	$3 + 2i$	9	111001100	11011101
1010	$1 + 3i$	10	111001101	11011100
1011	2	11	100010000	11010000

Какое основание наиболее эффективно?

Предположения

- Стоимость схемы с b состояниями пропорциональна b
- Храним целые числа от 0 до M

Получаем

Количество памяти для хранения M : $\log_b M + 1$

Стоимость регистра $c = kb \log_b M + 1$,

Найти минимум – производная!

$$\frac{d}{db} kb \log_b M + 1 = \frac{d}{db} \frac{kb \ln M + 1}{\ln b} = k \ln M + 1 \frac{\ln b - 1}{\ln b^2} = 0$$

$b = e!!!$

Эффективнее двоичная или троичная?

$$\frac{c(2)}{c(3)} = \frac{2k \log_2 M + 1}{3k \log_3 M + 1} = \frac{2 \ln 3}{3 \ln 2} \approx 1.056$$

$$\frac{c(2)}{c(e)} \approx 1.062$$



Адженда

**Системы
счисления**

25 минут

**Данные в С
(продолжение)**

25 минут

Простые числа

15 минут

Асимптотика

25 минут

Ключевые слова для ТИПОВ ДАННЫХ

Ключевые слова в исходном стандарте K&R C	Ключевые слова, добавленные стандартом C90	Ключевые слова, добавленные стандартом C99
int	signed	_Bool
long	void	_Complex
short		_Imaginary
unsigned		
char		
float		
double		



Представление данных в разных системах счисления

```
#include <stdio.h>
int main(void)
{
    system("chcp 65001");
    int x = 100;
    int x_8 = 0100;
    int x_16 = 0x100;
    printf("десятичное = %d; восьмеричное = %o; шестнадцатеричное = %x\n", x, x, x);
    printf("десятичное = %d; восьмеричное = %#o; шестнадцатеричное = %#x\n", x, x, x);
    printf("десятичное = %d; восьмеричное = %o; шестнадцатеричное = %x\n", x_8, x_8, x_8);
    printf("десятичное = %d; восьмеричное = %#o; шестнадцатеричное = %#x\n", x_8, x_8, x_8);
    printf("десятичное = %d; восьмеричное = %o; шестнадцатеричное = %x\n", x_16, x_16, x_16);
    printf("десятичное = %d; восьмеричное = %#o; шестнадцатеричное = %#x\n", x_16, x_16, x_16);
    return 0;
}

//десятичное = 100; восьмеричное = 144; шестнадцатеричное = 64
//десятичное = 100; восьмеричное = 0144; шестнадцатеричное = 0x64
//десятичное = 64; восьмеричное = 100; шестнадцатеричное = 40
//десятичное = 64; восьмеричное = 0100; шестнадцатеричное = 0x40
//десятичное = 256; восьмеричное = 400; шестнадцатеричное = 100
//десятичное = 256; восьмеричное = 0400; шестнадцатеричное = 0x100
```

Использование символов: тип `char`

Тип данных `char` применяется для хранения символов, таких как буквы и знаки препинания, однако формально он также является целочисленным. **Почему?** Причина в том, что тип `char` в действительности хранит целые числа, а не символы. Для поддержки символов компьютер использует числовой код, в котором определенные целые числа представляют определенные символы.



ASCII

0 -	16 - ►	32 -	48 - 0	64 - @	80 - P	96 - '	112 - p
1 - ☺	17 - ◀	33 - !	49 - 1	65 - A	81 - Q	97 - a	113 - q
2 - ☹	18 - ⬆	34 - "	50 - 2	66 - B	82 - R	98 - b	114 - r
3 - ♥	19 - !!	35 - #	51 - 3	67 - C	83 - S	99 - c	115 - s
4 - ♦	20 - ¶	36 - \$	52 - 4	68 - D	84 - T	100 - d	116 - t
5 - ♣	21 - 6	37 - %	53 - 5	69 - E	85 - U	101 - e	117 - u
6 - ♠	22 - ▀	38 - &	54 - 6	70 - F	86 - V	102 - f	118 - v
7 -	23 - ⬇	39 - '	55 - 7	71 - G	87 - W	103 - g	119 - w
8 -	24 - ⬆	40 - (56 - 8	72 - H	88 - X	104 - h	120 - x
9 -	25 - ⬇	41 -)	57 - 9	73 - I	89 - Y	105 - i	121 - y
10 -	26 - ➡	42 - *	58 - :	74 - J	90 - Z	106 - j	122 - z
11 -	27 - ⬅	43 - +	59 - ;	75 - K	91 - [107 - k	123 - {
12 -	28 - ⬅	44 - ,	60 - <	76 - L	92 - \	108 - l	124 -
13 -	29 - ↔	45 - -	61 - =	77 - M	93 - j	109 - m	125 - }
14 - 🎵	30 - ▲	46 - .	62 - >	78 - N	94 - ^	110 - n	126 - ~
15 - ☼	31 - ▼	47 - /	63 - ?	79 - O	95 - ÷	111 - o	127 - ␣
16 - ►	32 -	48 - 0	64 - @	80 - P	96 -	112 - p	



ASCII, Unicode и ISO/IEC 10646

```
#include <stdio.h>
#include <stdlib.h> //для функции system
int main()
{
    system("chcp 65001"); //system("chcp 1251");
    printf("Привет мир!");
    return 0;
}
```

Active code page: 65001

Привет мир!

128 - A	144 - P	160 - a	176 - ☒	192 - L	208 - ⌚	224 - p	240 - Ě
129 - Б	145 - C	161 - б	177 - ☒	193 - ⌚	209 - ⌚	225 - c	241 - ě
130 - В	146 - Т	162 - в	178 - ■	194 - ⌚	210 - ⌚	226 - т	242 - є
131 - Г	147 - У	163 - г	179 -	195 - ⌚	211 - ⌚	227 - у	243 - е
132 - Д	148 - Ф	164 - д	180 -]	196 - —	212 - ⌚	228 - ф	244 - ě
133 - Е	149 - X	165 - е	181 - ⌚	197 - +	213 - ⌚	229 - x	245 - i
134 - Ж	150 - Ц	166 - ж	182 - ⌚	198 - ⌚	214 - ⌚	230 - ц	246 - ŷ
135 - З	151 - Ч	167 - з	183 - ⌚	199 - ⌚	215 - ⌚	231 - ч	247 - ŷ
136 - И	152 - Ш	168 - и	184 - ⌚	200 - ⌚	216 - ⌚	232 - ш	248 - °
137 - Й	153 - Щ	169 - й	185 - ⌚	201 - ⌚	217 - ⌚	233 - щ	249 - ●
138 - К	154 - Ъ	170 - к	186 - ⌚	202 - ⌚	218 - ⌚	234 - ъ	250 - .
139 - Л	155 - Ы	171 - л	187 - ⌚	203 - ⌚	219 - ⌚	235 - ы	251 - √
140 - М	156 - Ь	172 - м	188 - ⌚	204 - ⌚	220 - ⌚	236 - ь	252 - №
141 - Н	157 - Э	173 - н	189 - ⌚	205 - ⌚	221 - ⌚	237 - э	253 - Ø
142 - О	158 - Ю	174 - о	190 - ⌚	206 - ⌚	222 - ⌚	238 - ю	254 - ■
143 - П	159 - Я	175 - п	191 - ⌚	207 - ⌚	223 - ⌚	239 - я	255 - ■
144 - Р	160 - а	176 - ☒	192 - L	208 - ⌚	224 - p	240 - Ě	

- Unicode (коммерческая инициатива) - система для представления широкого разнообразия наборов символов, применяемых в различных частях мира (более 110 000 символов).
- Организация ISO и комиссия IEC (International Electrotechnical Commission — Международная электротехническая комиссия) — стандарт, получивший название ISO/IEC 10646.
- Unicode сохранил совместимость с более широким стандартом ISO/IEC 10646.

Демонстрационная программа char

```
#include <stdio.h>
int main(void)
{
    char broiled; /* объявление переменной типа char */
    broiled = 'T'; /* правильно */
    /* broiled = T; Неправильно! Компилятор считает, что T является переменной */
    /* broiled = "T"; Неправильно! Компилятор считает, что "T" является строкой */
    char grade = 'B';
    char beep = 65;
    printf("broiled = %c, grade = %c, beep = %c\n", broiled, grade, beep);
    char A = '9', B = '0';
    printf("A - B = %hd", A - B);
    return 0;
}
// broiled = T, grade = B, beep = A
// A - B = 9
```


Непечатаемые символы

Последовательность	Описание
<code>\a</code>	Предупреждение (стандарт ANSI C)
<code>\b</code>	Возврат на одну позицию влево
<code>\f</code>	Перевод страницы
<code>\n</code>	Новая строка
<code>\r</code>	Возврат каретки
<code>\t</code>	Горизонтальная табуляция
<code>\v</code>	Вертикальная табуляция
<code>\\</code>	Обратная косая черта (\)
<code>\'</code>	Одиночная кавычка (')
<code>\"</code>	Двойная кавычка (")
<code>\?</code>	Знак вопроса (?)
<code>\ooo</code>	Восьмеричное значение (о представляет восьмеричную цифру)
<code>\xhh</code>	Шестнадцатеричное значение (h представляет шестнадцатеричную цифру)

Примеры целочисленных констант

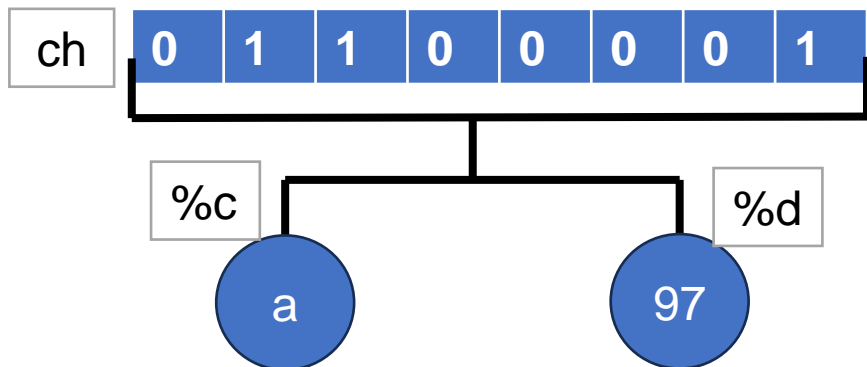
Начиная со стандарта C90

Тип	Шестнадцатеричная	Восьмеричная	Десятичная
char	\0x41	\0101	-
int	0x41	0101	65
unsigned int	0x41u	0101u	65u
long	0x41L	0101L	65L
unsigned long	0x41UL	0101UL	65UL
long long	0x41LL	0101LL	65LL
unsigned long long	0x41ULL	0101ULL	65ULL



Печатаемые символы

```
#include <stdio.h>
int main(void)
{
    system("chcp 65001");
    char ch;
    printf("Введите какой-нибудь символ.\n");
    scanf("%c", &ch); /* пользователь вводит символ */
    printf("Код символа %c равен %d.\n", ch, ch);
    return 0;
}
// Введите какой-нибудь символ.
// а
// Код символа а равен 97.
```



Переносимые типы: `stdint.h` и `inttypes.h`

`stdint.h`

- Язык C располагает типами, смысл которых *не зависит от системы* со стандарта C99 с помощью дополнительных заголовочных файлов (например, тип `int32_t`)
- Если компилятор не поддерживает типы с точной шириной, можно указать тип с минимально нужной шириной (например, тип `int_least8_t`)
- Если нужна скорость – высокоскоростные типы с минимальной шириной (например, тип `int_fast8_t`)
- Если нам нужен максимально возможный по памяти тип: `intmax_t`

`inttypes.h`

- Содействуют с вводом и выводом значений типов данных (например, не знаем `%ld` или `%d` -> `PRId32`)

Переносимые типы: stdint.h и inttypes.h - демо

```
#include <stdio.h>
#include <inttypes.h> // поддерживает переносимые типы
int main(void)
{
    system("chcp 65001");
    int32_t me32; // me32 -- это 32-битная переменная со знаком
    me32 = 45933945;
    printf("Сначала предположим, что int32_t является int: ");
    printf("me32 = %d\n", me32);
    printf("Далее не будем делать никаких предположений. \n");
    printf("Вместо этого воспользуемся \"макросом\" из файла inttypes.h: ");
    printf("me32 = %" PRIu32 "\n", me32);
    return 0;
}
//Сначала предположим, что int32_t является int: me32 = 45933945
//Далее не будем делать никаких предположений.
//Вместо этого воспользуемся "макросом" из файла inttypes.h: me32 = 45933945
```

Типы float, double и long double

Числа с плавающей запятой (float, double и long double) позволяют представлять намного больший диапазон чисел, включая десятичные дроби

float – 32 bit

- способен представлять минимум шесть значащих цифр и охватывал диапазон значений, по меньшей мере, от 10^{-37} до 10^{+37}

double – 64 bit

- способен представлять минимум десять значащих цифр и охватывал диапазон значений, по меньшей мере, от 10^{-37} до 10^{+37} (в реальности гораздо лучше)

long double - гарантировано только то, что точность типа long double, по меньшей мере, не уступает точности типа double

Число	Научная форма записи	Экспоненциальная форма записи
1 000 000 000	1.0×10^9	1.0e9
123 000	1.23×10^5	1.23e5
322,56	3.2256×10^2	3.2256e2
0,000056	5.6×10^{-5}	5.6e-5



Типы float, double и long double

Объявления:

- float noah, jonah;
- double trouble;
- float planck = 6.63e-34;
- long double gnp;

Константы:

-1.56E+12
2.87e-3
3.14159
.2
4e16
.8E-5
100.

Не применяйте пробелы в константах с плавающей запятой. Например, эта константа является недопустимой: 1.56 E+12

some = 4.0 * 2.0; По умолчанию 2.0 и 4.0 – double. Чтобы float - суффикс f или F (пример, 2.3f и 9.11E9F), long double – суффикс l или L (пример, 54.3l и 4.32e4L)

Шестнадцатеричная система – аналогично 0x или 0X в начале. Вместо экспоненты используется 2 для степени, обозначаемая как p

Пример: 0xa.1fp10 = $(10 + 1/16 + 15/256) * 2^{10} = 10364,0$



Типы float, double и long double- демо

```
#include <stdio.h>
int main(void)
{
    system("chcp 65001");
    float aboat = 32000.0;
    double abet = 2.14e9;
    long double dip = 5.32e-5;
    printf("%f может быть записано как %e\n", aboat, aboat);
    // для вывода следующей строки требуется компилятор,
    // поддерживающий C99 или более поздний стандарт
    printf("И его %a в шестнадцатеричной, представляющей степени 2, форме записи\n", aboat);
    printf("%f может быть записано как %e\n", abet, abet);
    printf("%Lf может быть записано как %Le\n", dip, dip);
    return 0;
}
// 32000.000000 может быть записано как 3.200000e+04
// И его 0xf.ар+11 в шестнадцатеричной, представляющей степени 2, форме записи
// 2140000000.000000 может быть записано как 2.140000e+09
// 0.000053 может быть записано как 5.320000e-05
```

Преобразование типов

1. Находясь в выражении, типы `char` и `short` (как `signed`, так и `unsigned`) автоматически преобразуются в `int` или при необходимости в `unsigned int`. (Если тип `short` имеет такой же размер, как у `int`, то размер типа `unsigned short` больше, чем `int`; в этом случае `unsigned short` преобразуется в `unsigned int`.) В K&R C, но не в текущей версии языка тип `float` автоматически преобразуется в `double`. Поскольку они являются преобразованиями в большие по размеру типы, они называются повышением.
2. Если в любую операцию вовлечены два типа, оба значения приводятся к более высокому из этих двух типов.
3. Порядок типов от высшего к низшему выглядит так: `long double`, `double`, `float`, `unsigned long long`, `long long`, `unsigned long`, `long`, `unsigned int` и `int`. Возможно одно исключение, когда `long` и `int` имеют одинаковые размеры; в этом случае `unsigned int` превосходит `long`. Типы `short` и `char` в этом списке отсутствуют, т.к. они уже должны были повыситься до `int` или, возможно, до `unsigned int`.
4. В операторе присваивания финальный результат вычислений преобразуется к типу переменной, которой присваивается значение. Процесс может привести к повышению типа, как описано в правиле 1, или к понижению типа, при котором значение преобразуется в более низкий тип.
5. При передаче в качестве аргументов функции типы `char` и `short` преобразуются в `int`, а `float` — в `double`. Это автоматическое повышение переопределяется прототипированием функций.



Преобразование типов

Ниже приведены правила для случаев, когда присвоенное значение не помещается в конечном типе.

1. Когда целевым является одна из форм целочисленного типа без знака, а присвоенное значение представляет собой целое число, лишние биты, делающие значение слишком большим, игнорируются. Например, если целевой тип — 8-битный `unsigned char`, то присвоенным значением будет результат деления исходного значения по модулю 256.

2. Если целевым типом является целый тип со знаком, а присвоенное значение — целое число, то результат зависит от реализации.

3. Если целевой тип является целочисленным, а присвоенное значение представляет собой значение с плавающей запятой, то поведение не определено. А что, если значение с плавающей запятой уместается в целочисленный тип? Когда типы с плавающей запятой понижаются до целочисленного типа, они усекаются или округляются в направлении нуля. Это означает, что и 23.12, и 23.99 усекаются до 23, а -23.5 усекается до -23.



Преобразование типов

```
#include <stdio.h>
int main(void)
{
    system("chcp 65001");
    char ch;
    int i;
    float f1;
    f1 = i = ch = 'C';
    printf("ch = %c, i = %d, f1 = %2.2f\n", ch, i, f1);
    ch = ch + 1;
    i = f1 + 2 * ch;
    f1 = 2.0 * ch + i;
    printf("ch = %c, i = %d, f1 = %2.2f\n", ch, i, f1);
    ch = 1107;
    printf("Теперь ch = %c\n", ch);
    ch = 80.89;
    printf("Теперь ch = %c\n", ch);
    return 0;
}
// ch = C, i = 67, f1 = 67.00
// ch = D, i = 203, f1 = 339.00
// Теперь ch = S
// Теперь ch = P
```

Операция приведения

```
int a;  
a = 1.6 + 1.7;  
a = (int) 1.6 + (int) 1.7;
```



Операция приведения () Ответ какой?

```
int a;
```

```
a = 1.6 + 1.7;
```

```
a = (int) 1.6 + (int) 1.7;
```

Что будет если переполнить
float?

Что будет если постоянно
уменьшать число в float?

Ошибки округления - демо

```
#include <stdio.h>
int main(void)
{
    system("chcp 65001");
    float a, b;
    b = 2.0e20 + 1.0;
    a = b - 2.0e20;
    printf("%f \n", a);
    return 0;
}
// 4008175468544.000000
```

Размеры типов

```
#include <stdio.h>
int main(void)
{
    system("chcp 65001");
    /* В стандарте c99 для размеров предусмотрен спецификатор %zd */
    printf("Тип int имеет размер %zd байт(ов).\n", sizeof(int));
    printf("Тип char имеет размер %zd байт(ов).\n", sizeof(char));
    printf("Тип long имеет размер %zd байт(ов). \n", sizeof(long));
    printf("Тип long long имеет размер %zd байт(ов). \n", sizeof(long long));
    printf("Тип doUle имеет размер %zd байт(ов) .\n", sizeof(double));
    printf("Тип long double имеет размер %zd байт(ов).\n", sizeof(long double));
    return 0;
}
// Тип int имеет размер 4 байт(ов).
// Тип char имеет размер 1 байт(ов).
// Тип long имеет размер 4 байт(ов).
// Тип long long имеет размер 8 байт(ов).
// Тип doUle имеет размер 8 байт(ов) .
// Тип long double имеет размер 16 байт(ов).
```

Аргументы и связанные с ними ловушки

```
#include <stdio.h>
int main(void)
{
    system("chcp 65001");
    int n = 4;
    int m = 5;
    float f = 7.0f;
    float g = 8.0f;
    printf("%d\n", n, m);      /*слишком много аргументов */
    printf("%d %d %d\n", n); /* аргументов недостаточно */
    printf("%d %d\n", f, g); /* неправильные типы значений */
    return 0;
}
// 4
// 4 112220480 220192896
// 0 0
```

Управляющие последовательности

```
#include <stdio.h>
int main(void)
{
    system("chcp 65001");
    float salary;
    printf("\aВведите желаемую сумму месячной зарплаты:"); /* 1 */
    printf(" $_____\\b\\b\\b\\b\\b\\b\\b\\b"); /* 2 */
    scanf("%f", &salary);
    printf("\n\t$%.2f в месяц соответствует $%.2f в год.", salary, salary * 12.0); /* 3 */
    printf("\rОго!\n"); /* 4 */
    return 0;
}
// Введите желаемую сумму месячной зарплаты: $_____
// Введите желаемую сумму месячной зарплаты: $1234567
//
// Ого!      $1234567.00 в месяц соответствует $14814804.00 в год.
```

Адженда

**Системы
счисления**

30 минут

**Данные в С
(продолжение)**

30 минут

Простые числа

5 минут

Асимптотика

25 минут

Работа с простыми числами

Простое число — это натуральное (целое положительное) число больше единицы, которое делится на единицу и на само себя. Все остальные натуральные числа больше единицы являются составными.

Нужны для облегчения, так и усложнения отдельные операций (пр. в криптографии)

Как найти алгоритмически?

- Деление на все возможные множители?

Улучшения?

- Нечетные, кроме 2
- До корня из числа
- Обновить максимальное количество потенциальных множителей

Будем возвращаться к ним на протяжении учебного года

Адженда

**Системы
счисления**

25 минут

**Данные в С
(продолжение)**


25 минут

Простые числа

15 минут

Асимптотика

25 минут



как преобразовать размытое
понятие «эффективный
алгоритм» в нечто более
конкретное?

Вычислительная разрешимость

как преобразовать размытое понятие «эффективный алгоритм» в нечто более конкретное?

- алгоритм называется эффективным, если его реализация быстро выполняется на реальных входных данных.

Типичная ситуация: два совершенно разных алгоритма примерно одинаково работают на входных данных размера 100; при 10-кратном увеличении размера данных один алгоритм продолжает работать быстро, а выполнение другого резко замедляется.

Итак, в **общем случае** мы будем проводить **анализ худшего случая времени** выполнения алгоритма.

- алгоритм считается эффективным, если он обеспечивает (на аналитическом уровне) качественно лучшую производительность в худшем случае, чем поиск методом «грубой силы»
- алгоритм считается эффективным, если он имеет полиномиальное время выполнения



Время выполнения

Время выполнения (с округлением вверх) разных алгоритмов для входных данных разного размера (для процессора, выполняющего миллион высокоуровневых команд в секунду). Если время выполнения алгоритма превышает 10²⁵ лет, мы просто используем пометку «очень долго»

n=	n	$n \log_2 n$	n^2	n^3	$1,5^n$	2^n	$n!$
10	< 1 с	< 1 с	< 1 с	< 1 с	< 1 с	< 1 с	4с
30	< 1 с	< 1 с	< 1 с	< 1 с	< 1 с	18 минут	10 ²⁵ лет
50	< 1 с	< 1 с	< 1 с	< 1 с	11 минут	36 лет	очень долго
100	< 1 с	< 1 с	< 1 с	1 с	12 892 лет	10 ¹⁷ лет	очень долго
1000	< 1 с	< 1 с	1 с	12 минут	очень долго	очень долго	очень долго
10000	< 1 с	< 1 с	2 минуты	12 дней	очень долго	очень долго	очень долго
100000	< 1 с	2 с	3 часа	32 года	очень долго	очень долго	очень долго
1000000	1 с	20 с	12 дней	31710 лет	очень долго	очень долго	очень долго

Алгоритмы

Алгоритм — это последовательность команд для исполнителя, обладающая рядом свойств:

- **полезность**, то есть умение решать поставленную задачу.
- **детерминированность**, то есть каждый шаг алгоритма должен быть строго определён во всех возможных ситуациях.
- **конечность**, то есть способность алгоритма завершиться для любого множества входных данных
- **массовость**, то есть применимость алгоритма к разнообразным входным данным.

Программа есть запись алгоритма на формальном языке.

Одна задача может иметь несколько алгоритмов и разные используемые ресурсы.

Разные исполнители - разные элементарные действия и элементарные объекты.

Эффективность - способность алгоритма использовать ограниченное количество ресурсов.

- **комбинационная сложность** - минимальное число элементов для реализации алгоритма в виде вычислительного устройства.
- **описательная сложность** - длина описания алгоритма на формальном языке
- **вычислительная сложность** - количество элементарных операций, выполняемых алгоритмом для неких входных данных.

Асимптотические верхние границы

Пусть $T(n)$ — функция (допустим, время выполнения в худшем случае для некоторого алгоритма) с входными данными размера n . (Будем считать, что все рассматриваемые функции получают неотрицательные значения.) Для другой функции $f(n)$ говорится, что $T(n)$ имеет порядок $f(n)$ (в условной записи $O(f(n))$), если для достаточно больших n функция $T(n)$ ограничивается сверху произведением $f(n)$ на константу. Также иногда используется запись $T(n) = O(f(n))$. А если выражаться точнее, функция $T(n)$ называется имеющей порядок $O(f(n))$, если существуют такие константы $c > 0$ и $n_0 \geq 0$, что для всех $n \geq n_0$ выполняется условие $T(n) \leq c \cdot f(n)$. В таком случае говорят, что T имеет асимптотическую верхнюю границу f . Важно подчеркнуть, что это определение требует существования константы c , работающей для всех n ; в частности, c не может зависеть от n .

Пример:

$T(n) = pn^2 + qn + r$, для $p > 0$, $q > 0$ и $r > 0$.

Мы утверждаем, что любая такая функция имеет $O(n^2)$.



Асимптотические нижние границы

Время выполнения для худшего случая $T(n)$ имеет порядок $O(n^2)$ бывает нужно показать, что эта верхняя граница является лучшей из возможных. Для этого следует выразить условие, что для входных данных произвольно большого размера n функция $T(n)$ не меньше произведения некой конкретной функции $f(n)$ на константу. (В данном примере в роли $f(n)$ оказывается n^2). Соответственно говорят, что **$T(n)$ имеет нижнюю границу $\Omega(f(n))$** (также используется запись $T(n) = \Omega(f(n))$), если существуют такие константы $\epsilon > 0$ и $n_0 \geq 0$, что для всех $n \geq n_0$ выполняется условие $T(n) \geq \epsilon \cdot f(n)$. По аналогии с записью $O(\cdot)$ мы будем говорить, что T в таком случае имеет асимптотическую нижнюю границу f . И снова следует подчеркнуть, что константа ϵ должна быть фиксированной и не зависящей от n .

Пример:

$T(n) = pn^2 + qn + r$, для $p > 0$, $q > 0$ и $r > 0$.

Мы утверждаем, что любая такая функция имеет $T(n) = \Omega(n^2)$.



Асимптотические точные границы

Если можно показать, что время выполнения $T(n)$ одновременно характеризуется $O(f(n))$ и $\Omega(f(n))$, то возникает естественное ощущение, что найдена «правильная» граница: $T(n)$ растет в точности как $f(n)$ в пределах постоянного коэффициента.

Пример:

$T(n) = pn^2 + qn + r$, для $p > 0$, $q > 0$ и $r > 0$.

Любая такая функция одновременно имеет $O(n^2)$ и $\Omega(n^2)$.

Для выражения этого понятия тоже существует специальная запись: **если функция $T(n)$ одновременно имеет $O(f(n))$ и $\Omega(f(n))$, говорят, что $T(n)$ имеет $\Theta(f(n))$, а $f(n)$ называется асимптотически точной границей для $T(n)$.**

$T(n) = pn^2 + qn + r$ имеет $\Theta(n^2)$.

Имеются две функции f и g , для которых $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ существует и равен некоторому числу $c > 0$. Тогда $f(n) = \Theta(g(n))$.



Свойства асимптотических скоростей роста

Транзитивность

Если функция f асимптотически ограничена сверху функцией g , а функция g , в свою очередь, асимптотически ограничена сверху функцией h , то функция f асимптотически ограничена сверху функцией h . Аналогичное свойство действует и для нижних границ. Более точно это можно записать в следующем виде.

(а) Если $f = O(g)$ и $g = O(h)$, то $f = O(h)$.

(б) Если $f = \Omega(g)$ и $g = \Omega(h)$, то $f = \Omega(h)$.

(в) Если $f = \Theta(g)$ и $g = \Theta(h)$, то $f = \Theta(h)$.

Суммы функций

Пусть k — фиксированная константа, а f_1, f_2, \dots, f_k и h — функции, такие что $f_i = O(h)$ для всех i . В этом случае $f_1 + f_2 + \dots + f_k = O(h)$. (Аналогично с $\Omega(h)$, $\Theta(h)$.)



Асимптотические границы для некоторых распространенных функций

- Полиномиальные функции
- Логарифмические функции
- Экспоненциальные функции

Примеры:

- $T(N) = N + N^2 \sim O(N^2)$
- $T(N) = N + \log(N) \sim O(N)$
- $T(N) = N + e^N \sim O(e^N)$

Итак, логарифмические, полиномиальные и экспоненциальные функции служат полезными ориентирами в диапазоне функций, встречающихся при анализе времени выполнения. Логарифмические функции растут медленнее полиномиальных, а полиномиальные растут медленнее экспоненциальных.



Почему чаще используют $O(N)$

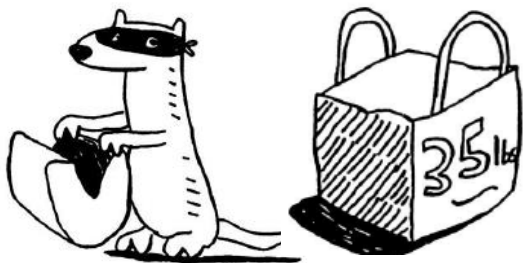
1 задача



N – элементов

1. Найти элемент со значением i – сложность $O(N)$
 2. Сложить все элементы – сложность $O(N)$
- А Ω , Θ – какая сложность?

2 задача о рюкзаке – для воров актуальна



20кг айфонов – $100000 \cdot 100$
25кг самсунгов – $90000 \cdot 120$
10кг сяоми – $15000 \cdot 100$
5кг гитар – $1000000 \cdot 5$
8кг косметики – 6500000
100 грамм золота - 1000000

$O(N)$ – перебор

$$F(N) = \sum_{k=0}^N C_k^N$$

Резюме

- Мы поговорили о разных системах счисления и научились переходить между ними
- Немного познакомились с необычными система счислений
- Закончили разбирать основные типы данных и научились преобразовывать типы
- Разобрали как искать простые числа просто, но не разобрали как искать их сложно
- Разобрали основные асимптотики