Floating-Point numbers

Форма представления действительных чисел:

- Числа с плавающей точкой (запятой)
- Числа с фиксированной точкой (Fixed-Point)

Тип	размер в байтах x86-32	x86-64	Приближенный диапазон значений и точности	
`float`	4	4	$10^{\mp 38}i$;	7 дес. цифр
`double`	8	8	$10^{\mp 307}$;	16 дес. цифр
`long double`	12	16	$10^{\mp 4931}$;	19 дес. цифр

Заголовочные файлы:

- `<float.h>` – параметры типов с плавающей точкой
- `<math.h>` – функции математической библиотеки
- `<fenv.h>` – доступ к сигналам и состояниям вычислителя с плавающей точкой (C99)

Представление чисел с плавающей точкой

Структура числа

- Мантисса: $\pm d_0.d_1 d_2 \dots d_{p-1}$
- Порядок: E
- p – точность представления
- β – основание системы счисления ($\beta = 2$)

$$\pm d.dd \dots d \times \beta^E = \pm(d_0 + d_1\beta^{-1} + \dots d_p\beta^{-(p-1)})\beta^E, \quad (0 \le d_i \le \beta)$$

Например для числа 0.1:

$\beta = 10, p = 3$ $\qquad 1.00 \times 10^{-1}$

$\beta = 2, p = 24$ $\qquad 1.10011001100110011001100 \times 2^{-4}$

Нормализованные числа

Для увеличения точности (p) мантиссу хранят в диапазоне $[1, \beta)$
Для $\beta = 2$ всегда выполняется $d_0 = 1$, поэтому оставляют только дробную часть: $d_1 d_2 \dots d_{p-1}$

Арифметические операции

Сложение и вычитание $\beta = 10, p = 7 \quad (\sim \textit{float})$

- **Поглощение значащих цифр малого числа:**
  $123456.7 + 101.7654 = 123558.4654$

```
123456.7       = 1.234567    x 10^5
101.7654       = 0.001017654 x 10^5
                 -----------------
                 1.235585    x 10^5
```

- **Потеря точности при вычитании:**
  $123457.1467 - 123456.659 = 0.4877$

```
123457.1467    = 1.234571 x 10^5
123456.659     = 1.234567 x 10^5
                 -----------------
                 0.000004 x 10^5
```

Сложение и вычитание $\beta = 10, p = 7$ $(\sim \mathit{float})$ (продолжение)

- **Нарушается ассоциативность:** $(a + b) + c \neq a + (b + c)$

$$123456.7 + 0.08 + 0.03 = 123456.81$$

```
123456.7 = 1.234567 x 10^5        0.08     = 8.000000 x 10^{-2}
0.08     = 0.000000 x 10^5        0.03     = 3.000000 x 10^{-2}
           ---------------                   ------------------
           1.234567 x 10^5                   1.100000 x 10^{-1}

           1.234567 x 10^5        123456.7 = 1.234567 x 10^5
0.03     = 0.000000 x 10^5        0.11     = 0.000001 x 10^5
           ---------------                   ---------------
           1.234567 x 10^5                   1.234568 x 10^5
```

Умножение и деление

В этих операциях потери точности нет:

$4734.612 * 541724.2 = 2564853898.0104$

```
4734.612        = 4.734612 x 10^3
541724.2        = 5.417242 x 10^5
                  ------------------
                  25.64854 x 10^8
```

Внимание

Ошибка округления может накапливаться . . .

Стандарт IEEE 754

Некоторые базовые форматы представлений:

Имя	Точность	E-min	E-max	$\sim$точность$_{10}$	$\sim$E-max$_{10}$
single	23+1	-126	$+127$	7.22	38.23
double	52+1	-1022	$+1023$	15.95	307.95
quadruple	112+1	-16382	$+16383$	34.02	4931.77
Intel 80x87 "co-processor"					
80-bit	63+1	-16382	$+16383$	19.27	4931.77

`sizeof()` для чисел с плавающей точкой

X86–32

- float $= 4 \Rightarrow$ single
- double $= 8 \Rightarrow$ double
- long double $= 12 \Rightarrow$ 80-bit

X86–64

- float $= 4 \Rightarrow$ single
- double $= 8 \Rightarrow$ double
- long double $= 16 \Rightarrow$ 80-bit

Бинарное представление

	Знак	Экспонента*	Дробная часть мантиссы
single	1-bit	8-bits	23-bits
double	1-bit	11-bits	52-bits
quadruple	1-bit	15-bits	112-bits
80-bits	1-bit	15-bits	63-bits

* Для показателя используется представление целых чисел *excess-K* с $K = 2^{(n-1)} - 1$, где n – число бит в поле экспоненты

Специальные числа

- **Ноль (0)** – нулевые значения и в поле экспоненты и в поле дробной части. **N.B.** Существует как $+0$, так -0!
- **Денормализованные числа** – нулевые значения в поле экспоненты. В этом случае считают лидирующий бит $d_0 = 0$. Например, для double: $(-1)^s \times 0.d_1 d_2 \ldots d_{52} \times 2^{-1022}$
- **Бесконечность (∞)** – единицы в поле экспоненты и нули в поле дробной части. **N.B.** Существует как $+\infty$, так $-\infty$!
- **NaN (Not a number)** – единицы в поле экспоненты и ненулевая мантисса. Знак NaN не имеет значения.

Операции со специальными числами NaN и ∞

фрагмент программы

```
double one = +1, zero = 0;
double p_inf = one/zero;
double m_inf = one/-zero;
printf(" one/zero=%+f  one/-zero=%+f zero/zero=%+f\n",
        one/zero,one/-zero,zero/zero);
printf(" one/+inf=%+f one/-inf=%+f inf*zero=%+f\n",
        one/p_inf,one/m_inf,p_inf*zero);
printf(" -inf+inf=%+f -inf*+inf=%+f +inf/+inf=%+f\n",
        m_inf+p_inf,m_inf*p_inf,p_inf/p_inf);
printf(" log(+inf) = %+f log(-inf) = %+f\n",
        log(p_inf),log(m_inf));
```

```
one/zero=+inf         one/-zero=-inf         zero/zero=-nan
one/+inf=+0.000000    one/-inf=-0.000000     inf*zero=-nan
-inf+inf=-nan         -inf*+inf=-inf         +inf/+inf=-nan
log(+inf) = +inf      log(-inf) = +nan
```

Стандарт IEEE 754 определяет операции с NaN и ±INF так:

операция	результат
$Number / \pm\infty$	0
$\pm\infty \times \pm\infty$	$\pm\infty$
$\pm Nonzero/0$	$\pm\infty$
$\infty + \infty$	∞
$\pm 0/\pm 0$	NaN
$\infty - \infty$	NaN
$\pm\infty/\pm\infty$	NaN
$\pm\infty \times 0$	NaN

Целые числа: деление на ноль

фрагмент программы

```
int one = 1;
int zero = 0;
fprintf(stderr,"\n Test zero division one= %d, zero= %d\n"
        " one/zero= ",one,zero);
fprintf(stderr,"%d\n",one/zero);
```

Output: Исключение в операции с плавающей точкой

```
 Test zero division one= 1, zero= 0
 one/zero= Floating point exception
```

Что такое “Floating point exception”?

В данном контексте «Исключение в операции с плавающей точкой» – имя сигнала SIGFPE.
Имя SIGFPE – неточное, так как сюда же входят и операции с целыми числами, сохраняется для обратной совместимости кода.
В русскоязычной литературе SIGFPE иногда называют «ошибочная арифметическая операция»

Как избежать целого деления на ноль:

```
int div(int x, int y) {
  /* return x if y==0 and x/y otherwise */
  return x /(y + (y==0));
}
```

Поддержка стандарта IEEE 754 в C (C99)

`#include <math.h>`, fp – число с плавающей точкой

`NAN`, `INFINITY`, `HUGE_VAL`, `HUGE_VALF`, `HUGE_VALL` – определяют константы (макросы) для fp

`int fpclassify(fp)` – в зависимости от fp возвращает:
`FP_INFINITY`, `FP_NAN`, `FP_NORMAL`, `FP_SUBNORMAL` или `FP_ZERO`

`int isnan(fp)` – возвращает ненулевое значение, если fp не является числом (`NAN`)

`int isinf(fp)` – возвращает 1 если `fp=+INFINITY` и -1 если `fp=-INFINITY`

`int isfinite(fp)` – возвращает ненулевое значение, если fp не `NAN` и не `INFINITY`

`int isnormal(fp)` – возвращает ненулевое значение, если fp представляет собой нормализованное число

Машинная точность ϵ

Def: ϵ – наименьшее положительное число такое, что $1 + \epsilon \neq 1$

По смыслу, ϵ – максимальная относительная ошибка представления ненулевого вещественного числа $\left|\frac{Fp(x)-x}{x}\right| < \epsilon$

Простая программа вычисления ϵ

```
double eps() {
  double one = 1;
  double eps = one;
  double tmp = 0;
  do {
     eps *= 0.5;
     tmp = one + eps;
  } while( tmp > one );
  return eps*2;
}
```

```
eps(double) = 2.22045e-16  log_2(eps) = -52
eps(long double) = 1.0842e-19  log_2(eps) = -63
eps(float) = 1.19209e-07  log_2(eps) = -23
```

Сравнение чисел с плавающей точкой

Равенство (==)?

```
if( result == expectedResult ) ...
```

- **Маловероятно**, что результат истинен
- Поведение нестабильно

Пример

```
void foo(double x, double y) {
  if (cos(x) != cos(y)) printf( " Huh?!?\n");
}
int main() {
  foo(1.0, 1.0);
  return 0;
}
```

Возможный результат:

```
Huh?!?
```

Тестовая программа на сравнение fp-чиселы

```
double x = 0., y = 0.;
int i;
for(i = 0; i < 10; i++) {
   x += 0.1;
   if( i%2 == 0 ) {
     y += 0.2;
   } else {
     printf(" %19.17f %19.17f --> %3d %3d %3d\n",
       x, y, (x==y), IsEqualABS(x,y,-1), IsEqualREL(x,y,-1));
   }
}
```

Output: `x, y -> ==, IsEqualAbs(), IsEqualRel()`

```
 0.20000000000000001 0.20000000000000001 -->   1   1   1
 0.40000000000000002 0.40000000000000002 -->   1   1   1
 0.59999999999999998 0.60000000000000009 -->   0   1   1
 0.79999999999999993 0.80000000000000004 -->   0   1   1
 0.99999999999999989 1.00000000000000000 -->   0   1   1
```

Сравнение с ε: абсолютная ошибка

```
int IsEqualABS(double x, double y, double epsilon)
{
  static double eps_m = -1.;
  if( eps_m < 0 ) eps_m = eps(); // machine epsilon
  if( epsilon <= eps_m ) epsilon = eps_m;
  return fabs(x-y) < epsilon;
}
```

Достоинства и недостатки (pro et contra)

- **pros** Если диапазон значений x и y известен и ограничен, то эта проверка очень проста и эффективна
- **cons** Не работает, если ε меньше, чем возможная разница для $|x - y|$, так как x, y – числа с плавающей точкой.
  Например для `float x = 12345.678, y = 12345,679;`
  возможная разница должна быть не меньше `0.1`

Сравнение с ε: относительная ошибка

```
int IsEqualREL(double x, double y, double epsilon)
{
  static double eps_m = -1.;
  if( eps_m < 0 ) eps_m = eps(); // machine epsilon
  if( epsilon < eps_m ) epsilon = eps_m;
  double ax = fabs(x);
  double ay = fabs(y);
  return fabs(x-y) < epsilon * ( (ax<ay) ? ax : ay);
}
```

pro et contra

- **pros** Более общий способ сравнения чисел, работающий вне зависимости от абсолютных значений x, y
- **cons** Плохо подходит для чисел близких к нулю (и совсем работает для $x = y = 0$).
  Например для `x=-1.e-10; y= +1.e-10` $\left|\frac{x-y}{max(x,y)}\right| = 2$

Преобразование типов данных

Явное преобразование: операция приведения типов

```
char c = 'A';
int i = 100;
float f = 0.5;
double d = 1.9;

i = (int) c;        /* char -> int */
d = (double) i;     /* int -> double */
i = (int) d;        /* дробная часть отбрасывается */
c = (char) i;       /* int -> char, получим 'd' */
d = (double) f;     /* float -> double, нули в мантиссе */
f = (float) d;      /* последние цифры теряются */
```

Неявное преобразование

Принцип преобразования: «меньший тип» → «больший»

`char` → `int` → `long int` → `float` → `double` → `long double`

```
(  c  /  i  )  +  (  f  *  d  )  -  (  f  +  i  );
   |     |           |     |           |     |
  int    |         double  |           |    float
   |_____|           |_____|           |_____|
      |                 |                 |
     int              double             float
      |_________________|                 |
               |__________________________|
                            |
                          double
```

Полезные библиотеки и программы

IEEE 754 floating-point test

`http://www.netlib.org/paranoia/`

Программа проверяющая на соответствие стандарту IEEE 754

GMP — The GNU Multiple Precision Arithmetic Library

`http://gmplib.org`

- Поддерживает работу с целыми знаковыми числами, числами с плавающей точкой а так же рациональными числами
- Точность вычислений практически неограниченна: 2^{31}-bit для 32-битных машин и 2^{37}-bit для 64-битных машин

Fixed point maths library: `libfixmath`

`https://code.google.com/p/libfixmath/`

Библиотека для работы с числами с фиксированной точкой