Лекция 11. Представление чисел и последствия этого

Представление целых чисел

Неотрицательное целое число — обычный двоичный код в рамках заданного количества байтов. $25 = 16 + 8 + 1 \ \to > 00011001 \qquad 0x19$

диапазон unsigned от 0 до $2^k - 1$, например, unsigned char — $0 - 255$

Отрицательные целые — дополнительный код = обратный код + 1 ???????

$-25 : 00011001 \ \to \ 11100110 + 1 \ \to \ 11100111$

в частности, $-1 \ \to \ 11111111$

Кстати, обратный код + 1 так же дает перевод и от отрицательных к положительным, действительно,

$-25 = 11100111 : 00011000 + 1 = 00011001 = 25$

Позволяет не задумываться о знаке при сложении

$25 + (-25) : 00011001 + 11100111 = 100000000$, старшая 1 выходит за разрядность

При сдвиге вправо знаковых целых заполнение происходит старшим битом (сохраняется знак)

```
25>>2     00011001 -> 00000110   (это 6)
(-25)>>2  11100111 -> 11111001   (это -7 = целая часть -25/4)
```

Переполнение — сумма положительных может стать отрицательным

сумма беззнаковых чисел может стать меньше исходных слагаемых

little endian — младший байт представления соответствует (начальному) адресу

big endian — старший байт представления соответствует (начальному) адресу

Представление вещественных чисел

Стандарт IEEE 754 (редакция 2008 г)

Число с плавающей точкой (floating point number)

$x = M \cdot 2^P$, M — мантисса, P — порядок, для единственности $1 \leq M < 2$.

Стандарт описывает много всего:

собственно представления чисел с разной точностью

понятия специальных чисел

операции и округления

вычисление элементарных функций

вычисления со специальными числами

и пр.

Мы рассмотрим только представления для float и double

```
float    знак 1 бит | порядок 8 бит  | мантисса  23 бита
double   знак 1 бит | порядок 11 бит | мантисса  52 бита
```

мантисса — без старшей единицы

порядок со сдвигом (на половину разрядности, т.е. $P + 2^{k-1} - 1$ т.е. +127 или +1023)

соответственно формально минимальный порядок

```
00000000   ->  2^{-127}              00000000000   ->  2^{-1023}
11111111   ->  2^{+128}              11111111111   ->  2^{+1024}
```

разбиение числовой оси:

```
  порядок мантисса
x|00...00|00...00        +- нуль
x|00...00|00...01        денормализованные числа (min)
x|00...00|11...11        денормализованные числа (max)
x|00...01|00...00        нормализованные числа (min)
x|11...10|11...11        нормализованные числа (max)
x|11...11|00...00        +- бесконечность  INF
x|11...11|xx...xx        NaN  not a number
```

Нетрудно вычислить диапазоны этих числе и расстояния между соседними представимыми числами

минимальное нормализованное число float $2^{-126} \sim 1.175 \cdot 10^{-38}$

максимальное нормализованное число float $2^{127}(2 - 2^{-23}) \sim 3.4 \cdot 10^{+38}$

шаг денормализованных чисел $2^{-126-23} = 2^{-149} \sim 1.4 \cdot 10^{-45}$

минимальное нормализованное число double $2^{-1022} \sim 2.2 \cdot 10^{-308}$

максимальное нормализованное число double $2^{1023}(2 - 2^{-52}) \sim 1.8 \cdot 10^{+308}$

шаг денормализованных чисел $2^{-1022-52} = 2^{-1074} \sim 4.9 \cdot 10^{-324}$

Вычисления проводятся с дополнительным количеством знаков (80 для мантиссы). Округление результата до ближайшего представимого числа при арифметических операциях.

Примеры: Несколько примеров влияния специфики представления чисел на вычисления

файл num.cpp

Погрешность элементарных операций

Как видно из вышесказанного, погрешность представления вещественных чисел имеет относительный характер, т.е. вместо точного значения x мы работаем с числом $x(1+\varepsilon)$, где ε есть величина порядка (полу)шага между двумя соседними представимыми числами из отрезка $[1, 2)$.

Таким образом, при выполнении арифметических операций точный ответ искажается, и мы можем оценить относительную погрешность такого искажения, исходя из погрешностей исходных операндов. Рассмотрим с этой точки зрения элементарные операции.

Пусть мы имеем два операнда x, y и соответственноо работаем с их искаженными погрешностью значениями $x(1+\varepsilon_x), y(1+\varepsilon_y)$. Так как на интересует оценка погрешности, т.е. ее максимально возможное значение, можно считать, что в операциях эта погрешность всегда распространяется наихудшим для нас образом (складывается и не компенсируется). И поэтому будем также для простоты считать, что $\varepsilon_x \sim \varepsilon_y \sim \varepsilon$.

Сложение, вычитание.

Представим результат операции в виде числа с относительной погрешностью, т.е. $(x \pm y)(1+\delta)$. Для наихудшего варианта имеем

$x(1+\varepsilon) \pm y(1+\varepsilon) \sim (x \pm y) + \varepsilon(|x| + |y|) = (x \pm y)\Big(1 + \varepsilon\frac{|x|+|y|}{x \pm y}\Big)$.

Это означает, что относительная погрешность результата $\delta = \varepsilon\frac{|x|+|y|}{x \pm y}$ может оказаться намного больше относительных погрешностей операндов, если, например, в знаменателе вычитаются два близких числа. Иногда подобный рост относительной погрешности может существенно повлиять на интерпретацию результата.

Пример. Квадратное уравнение.

Умножение.

$x(1+\varepsilon)\cdot y(1+\varepsilon) \sim (xy)(1+2\varepsilon+\varepsilon^2)$.

Отсюда видно, что при умножении относительная погрешность может возрасти в два раза в худшем случае.

Деление.

Деление можно свести к умножению на обратное значение. Т.е. достаточно рассмотреть как выглядит относительная погрешность для $1/x$.

$\frac{1}{x(1+\varepsilon)} = \frac{(1-\varepsilon)}{x(1-\varepsilon^2)} \sim \frac{1}{x}(1-\varepsilon)$.

Здесь отбросили квадрат погрешности как величину более высокого порядка. Таким образом, обратная величина имеет ту же относительную погрешность и соответственно деление ведет себя так же как и умножение, т.е. может увеличить относительную погрешность не более, чем в два раза.

На практике, конечно, погрешности могут компенсироваться за счет своих знаков, но особо рассчитывать на это не стоит.