ФМИ №2 2014: информатика № вёрстки 2	№ корректуры: Число ошибок: Верстал: Юля	Дата	Подпись:

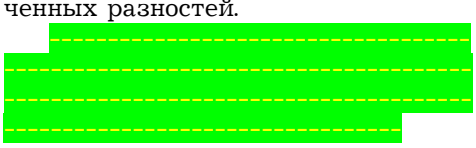
Плаксин Михаил Александрович
Кандидат физико-математических наук,
доцент Пермского филиала НИУ
Высшая школа экономики,
специалист по ТРИЗ 3-го уровня.

Кое-что о машинной арифметике,

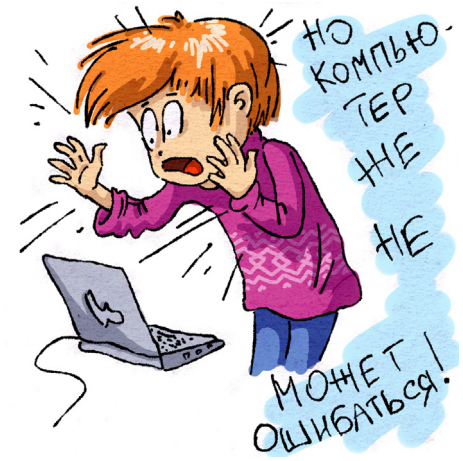
или Современный компьютер способен за одну секунду совершить столько же ошибок, сколько сто математиков за сто лет

Как вы думаете, что получится, если из числа вычесть его десятую часть, из оставшейся разности снова вычесть туже самую десятую, из оставшейся снова, и так – 10 раз? Иными словами, чему равна разность (X – 0.1*X – 0.1*X – 0.1*X – 0.1*X – 0.1*X – 0.1*X – – 0.1*X – 0.1*X – 0.1*X – 0.1*X)? Нуль? И я так думал когда-то. Но, по порядку...

Понадобилось мне как-то разложить по десятым долям числа 63 и 66. Ввёл числа в Excel, поделил на 10 и пошёл последовательно вычитать полученную десятую часть из исходных чисел, а потом из полученных разностей.



Вопрос: что останется из числа, если из него десять раз вычесть его десятую часть? Нуль? Только не с точки зрения Excel'а. Excel выдал вот что:



63	63,00000000000000000000	66	66,00000000000000000000
56,7	56,70000000000000000000	59,4	59,40000000000000000000
50,4	50,40000000000000000000	52,8	52,80000000000000000000
44,1	44,10000000000000000000	46,2	46,20000000000000000000
37,8	37,80000000000000000000	39,6	39,60000000000000000000
31,5	31,50000000000000000000	33	33,00000000000000000000
25,2	25,20000000000000000000	26,4	26,40000000000000000000
18,9	18,90000000000000000000	19,8	19,80000000000000000000
12,6	12,60000000000000000000	13,2	13,20000000000000000000
6,3	6,30000000000001000000	6,6	6,59999999999999000000
8,88178E – 15	0,00000000000000888178	– 8,8818E – 15	– 0,00000000000000888178

Правые колонки (со многими нулями после запятой) я добавил после того, как увидел левые, в которых в последней строке стояли совсем не нули.

Всего за 10 (!) шагов набежала погрешность, которую заметил Excel. Попробуем разобраться, откуда она взялась.

Машинную арифметику от обычной отличают две вещи.

Во-первых, машинная арифметика конечна. Поэтому машине недоступны в чистом виде иррациональные числа и бесконечные периодические дроби. Да и остальные рациональные числа доступны только «в переделах разрядной сетки». В «математической» арифметике $0,(9) = 1$. Почему? Числа не равны между собой, если модуль их разности больше нуля. В «математической» арифметике разность $(1 - 0,(9))$ меньше любого положительного числа. То есть, равна нулю. Не так в арифметике машинной. Там для числа хранится вполне определённое количество разрядов. Пусть (для определённости) их будет 10. Тогда число 0,(9) в машинной арифметике будет равно 0,9999999999. И разность между этим числом и числом 1 будет вполне конечным числом $-1 \cdot 10^{-10}$.

Во-вторых, машинная арифметика – арифметика двоичная. Все современные ЭВМ исповедуют двоичную систему. И все числа, которые в них представимы, представляются в виде суммы степеней двойки. Для целой части числа это не страшно. Любое целое число можно собрать из степеней двойки (1, 2, 4, 8 и т. д.). Другое дело – часть дробная. Чтобы понять, что там происходит, используем следующую Excel'овскую таблицу (пояснения даны ниже):

Разряд	Степень двойки	Значение рациональное	Значение десятичное	Брать ли?	Накапливаемая сумма
1	2	1/2	0,5000000000000000	0	0,0000000000000000
2	4	1/4	0,2500000000000000	1	0,2500000000000000
3	8	1/8	0,1250000000000000	0	0,2500000000000000
4	16	1/16	0,0625000000000000	1	0,3125000000000000
5	32	1/32	0,0312500000000000	0	0,3125000000000000
6	64	1/64	0,0156250000000000	1	0,3281250000000000
7	128	1/128	0,0078125000000000	0	0,3281250000000000
8	256	1/256	0,0039062500000000	1	0,3320312500000000
9	512	1/512	0,0019531250000000	0	0,3320312500000000
10	1024	1/1024	0,0009765625000000	1	0,3330078125000000
11	2048	1/2048	0,0004882812500000	0	0,3330078125000000
12	4096	1/4096	0,0002441406250000	1	0,3332519531250000
13	8192	1/8192	0,0001220703125000	0	0,3332519531250000
14	16384	1/16384	0,0000610351562500	1	0,3333129882812500
15	32768	1/32768	0,0000305175781250	0	0,3333129882812500
16	65536	1/65536	0,0000152587890625	1	0,3333282470703120
		1/3	0,3333333333333330		
			Сумма*3	=	0,9999847412109370
			1 – Сумма*3	=	1,52587891E – 05

Поясним, о чём идёт речь.

Будем считать, что для хранения дробной части числа наша ЭВМ отводит 16 двоичных разрядов. Номера разрядов приведены в левой графе. Каждый разряд соответствует степени двойки – вторая графа. Поскольку речь идёт о дробной части, то и каждому разряду будет соответствовать обратное число: графа 3 – в рациональной записи, графа 4 – в десятичной. (Маленькая хитрость. В приведенную здесь Excel'овскую таблицу внесена ручная правка. Excel умеет отображать в рациональном виде только числа до 2^{-9}. Начиная с 10-го разряда в графе «Значение рациональное» отображаются нули.)

Теперь попробуем из этих чисел собрать что-нибудь простенькое, но степенью двойки не являющееся. Например, 1/3. Какие из разрядов войдут в двоичное представление числа 1/3 и какое значение они дадут в совокупности.

1-й разряд = 0,5; 0,5 > 1/3. Велик, пропускаем. Накопленное значение = 0.

2-й разряд = 0,25; 0 + 0,25 < 1/3. Берём. Накопленное значение = 0,25.

3-й разряд = 0,0125; 0,25 + 0,125 > >1/3. Пропускаем. Накопленное значение = 0,25.

4-й разряд = 0,0625; 0,25 + 0,0625= = 0,3125 < 1/3. Берём.

И т. д. Получается

$$\frac{1}{3} \approx \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + \frac{1}{1024} + \\ + \frac{1}{4096} + \frac{1}{16384} + \frac{1}{65536}.$$

Для представления числа 1/3 мы отобрали все имеющиеся у нас чётные разряды. Но разрядов этих – немного. Поэтому сумма их даст нам число 0,3333282470703120, которое от «истинной 1/3» отличается вполне ощутимо. Чтобы сделать эту

разницу более наглядной, возьмём для сравнения не 1/3, а 1/3*3. В «математической» арифметике это 1. У нас получилось 0,9999847412109370. Разница составляет 1,52587891E–05.

Много это или мало? Просуммируйте эту разницу хотя бы тысячу раз, и она вырастет до десятых долей. Просуммируйте 100 тысяч раз, и она перейдёт в целые числа. Но 100 тысяч – далеко не самое большое число повторов в современных вычислениях. А есть ведь ещё и другие слагаемые, которые тоже внесут свой вклад в накопление ошибки.

Понятно, что погрешность будет зависеть от разрядности ЭВМ. Это легко продемонстрировать с помощью нашей Excel'овской таблицы. Достаточно просмотреть, как возрастает значение в графе «Накопленная сумма».

Для 8-битной машины получим, что 1/3 = 0,33203125, 1/3 * 3 = = 0,99609375. Разница

(1 – 1/3) = 3,90625000E-03.

То есть отклонения достигли 3-го знака после запятой.

Для 4-битного представления получим 1/3 = 0,3125. 1/3 * 3 = 0,9375. Разница (1 – 1/3) = 6,25E-02.

Из проведённых наблюдений становятся понятны некоторые ограничения, установленные для компьютерных вычислений. Например, в подавляющем большинстве языков программирования границы и шаг цикла не могут быть заданы вещественными числами, только целыми. Там, где это правило не соблюдается, легко наткнуться на неприятные неожиданности. Автор этих строк как-то столкнулся с ситуацией, когда сумма трёх (!) слагаемых, каждое из которых было равно 0,1, оказалась больше 0,3! В программе на языке Reduce тело цикла `for k:= 0 step 0.1 until 0.3 do…` было повторено только 3 раза!

К счастью, в большинстве языков счётчик цикла не может быть вещественным. Но поскольку в таких конструкциях есть объективная необходимость, его приходится моделировать. Поэтому о накоплении погрешностей надо помнить при любых вещественных вычислениях.

Далее в качестве иллюстраций будем использовать крохотные программки, написанные на языке Паскаль в среде Турбо-Паскаль.

Первый пример – на моделирование вещественного шага цикла – приведён на рис. 1.

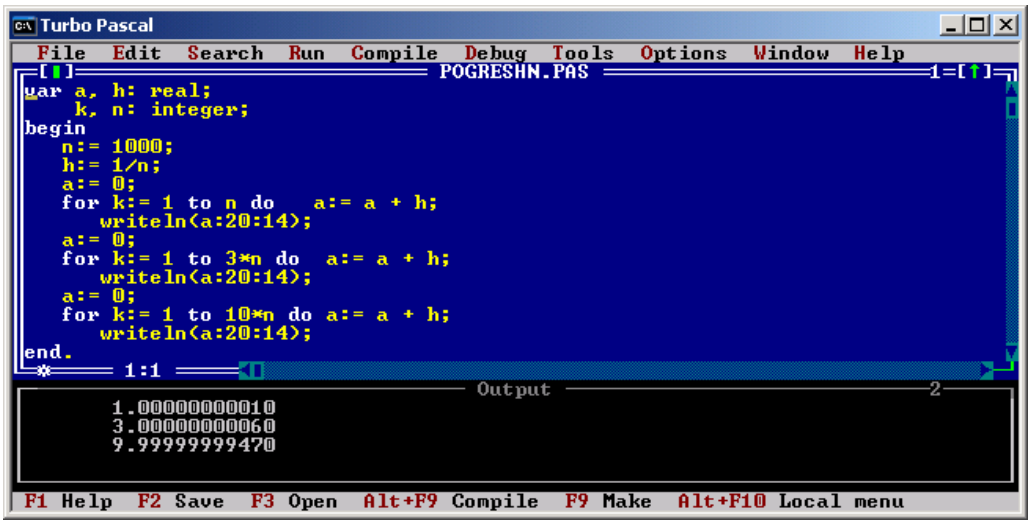


Рис.1. Накопление погрешности при вещественных вычислениях

Из приблизительного представления машинных чисел следует ещё одно правило, применяемое для сравнения вещественных чисел. Погрешности округления приводят к тому, что «лобовое» сравнение вещественных чисел может дать неверный результат.

Пусть вещественные переменные q и r обе равны единице. Но в одном случае вещественная единица будет представлена как 0,9999999999, а в другом – как 1,0000000001. В этом случае сравнение ($q = r$) вполне может дать «ложь». Лучше сравнивать модуль разности с некоторым ε: `abs(q-r) < eps`.

Следующая особенность машинной арифметики, которая следует из конечности разрядной сетки, – слишком большие числа ей противопоказаны. Они просто не поместятся в имеющееся число разрядов. В лучшем случае произойдёт аппаратное прерывание. Но возможны более печальные (для нас) случаи, когда сумма двух больших положительных целых чисел окажется числом отрицательным или положительным, но маленьким.

В Турбо-Паскале вещественное переполнение всегда вызывает аппаратное прерывание. Что касается целочисленного переполнения, то контроль за ним регулируется. Делается это, как обычно в Турбо-Паскале двумя способами:

1) с помощью управляющих комментариев {$Q+} и {$Q–}. Первый из них включает контроль целочисленного переполнения, второй – отключает;

2) переключением опции компилятора **Options/Compiler.../Overflow checking**.

По умолчанию контроль целочисленного переполнения отключён.

Примеры переполнений приведены на рис. 2, 3, 4. На рис. 2 при вычислении цикла произошло прерывание из-за вещественного переполнения.

Рис. 2. Вещественное переполнение

На рис. 3 программа выполняется при отключенном контроле целочисленного переполнения. Переполнение происходит при вычислении переменной n. Вычислительная система на него никак не реагирует. На примере хорошо видно, как меняется значение целочисленной переменной n в результате многократного умножения. 100 * 100 = 10 000; = 10 000*100 = 16 960! 16 960 * 100 = – 7 936! – 7 936 * 100 = – 7 168! И т. д.

```
Turbo Pascal
File  Edit  Search  Run  Compile  Debug  Tools  Options  Window  Help
                          PEREPOLN.PAS                                1
var a: real;
    k,n: integer;
begin
   a:= 1;   n:= 1;
   for k:= 1 to 10 do begin
      a:= a*100;   n:= n*100;
      writeln('a= ',a:25:1,'  n= ',n);
   end;
end.
                            Output                                2=[↑]
a=                     100.0  n= 100
a=                   10000.0  n= 10000
a=                 1000000.0  n= 16960
a=               100000000.0  n= -7936
a=             10000000000.0  n= -7168
a=           1000000000000.0  n= 4096
a=         100000000000000.0  n= 16384
a=       10000000000000000.0  n= 0
a=     1000000000000000000.0  n= 0
a=   100000000000000000000.0  n= 0
F1 Help  ↑↓→←  Scroll  F10 Menu
```

Рис. 3. Целочисленное переполнение при отключенном контроле

На рис. 4 та же самая программа выполняется при включенном контроле. В этом случае целочисленное переполнение приводит к аппаратному прерыванию так же, как и вещественное.

```
Turbo Pascal
File  Edit  Search  Run  Compile  Debug  Tools  Options  Window  Help
                          PEREPOLN.PAS                            1=[↑]
 Error 215: Runtime error.
    k,n: integer;
begin
   a:= 1;
   n:= 1;
   for k:= 1 to 10 do begin
      a:= a*100;
      n:= n*100;
      writeln('a= ',a:25:1,'  n= ',n);
   end;
end.
          8:1
                            Output                                2
a=                     100.0  n= 100
a=                   10000.0  n= 10000
Runtime error 215 at 0BED:006F.
F1 Help  F2 Save  F3 Open  Alt+F9 Compile  F9 Make  Alt+F10 Local menu
```

Рис. 4. Целочисленное переполнение при включенном контроле

Надо подчеркнуть, что диапазон значений типа integer в Турбо-Паскале весьма невелик: от – 32768 до 32767. Но в Турбо-Паскале (в отличие от стандартного Паскаля) существует ещё несколько целых типов как большего, так и меньшего размера. В частности, тип longint охватывает значения от – 2147483648 до 2147483647.

Конечное число разрядов ограничивает представимые числа и «сверху» и «снизу». Выход за разрядную сетку «вверх» – это переполнение. Но возможен выход и «вниз». В результате расчётов может быть получено число с мантиссой настолько маленькой, что с точки зрения машинной разрядной сетки это число будет восприниматься как нуль. Такая ситуация называется потерей значимости. Пример её приведён на рис. 5.

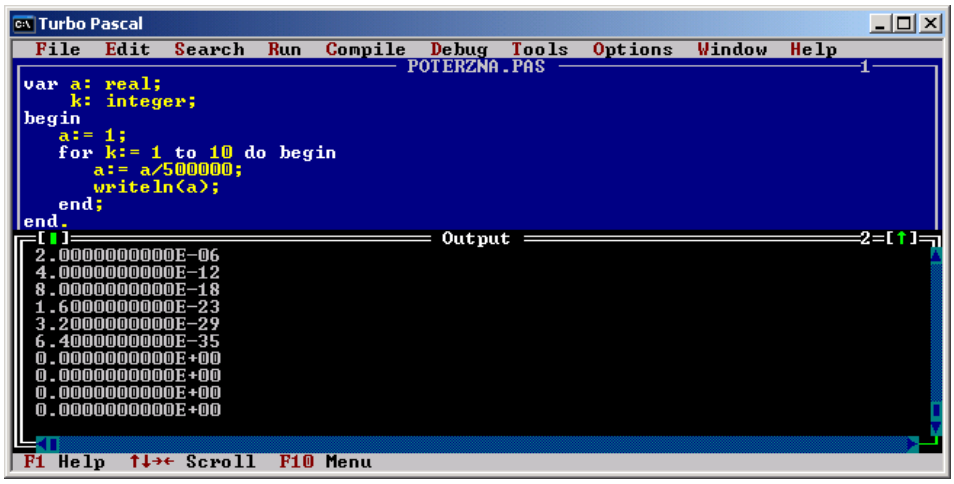


Рис. 5. Потеря значимости

Особенно неприятен тот момент, что как переполнение, так и потеря значимости могут произойти в промежуточных вычислениях. Конечный результат может иметь «нормальный» вид, но промежуточные могут оказаться слишком большими или слишком маленькими. Естественно, о правильности конечного результата в данном случае говорить не приходится. Для машинной арифметики порядок выполнения операций может оказаться весьма существенным. Это в математике

a * b / c = a / c * b.

В программировании – не всегда. Пример см. на рис. 6.

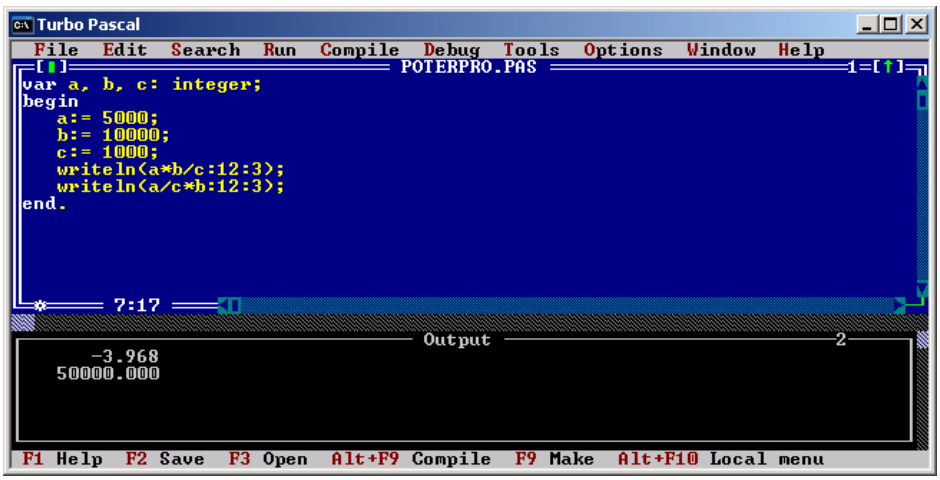


Рис. 6. Переполнение при промежуточных вычислениях

Представьте себе, что подобные формулы используются для вычисления вашей заработной платы.

Вычислительные машины были созданы для того, чтобы заменить человека при выполнении объёмных математических расчётов. И с возложенной на них задачей они справляются с честью. Но тем более необходимо помнить об отличиях арифметики компьютерной от арифметики математической.