Слово, число, рисунок, музыка,– величайшие изобретения человечества.

Глава 2 Числа

Алгоритмы и задачи, рассматриваемые в этой главе, могут использоваться, как на начальном этапе обучения программированию – именованные числа, перевод чисел из одной системы счисления в другую, так и при изучении более сложных и специальных тем – модульная арифметика, шифрование.

Многие задачи этой главы являются прекрасными примерами при изучении темы классов. У класса двойственная природа – с одной стороны это модуль, с другой – тип данных. Рациональные числа, комплексные числа, простые числа, являются естественными примерами классов, поскольку интуитивно понятно, какой тип данных они задают. Данная глава заканчивается большим примером построения класса Rational - рациональных чисел.

Решение задач этой главы может поддерживать изучение лекций 9-16 учебника. Эту же мысль можно выразить и по-другому. Если считать, что приоритетом при изучении программирования являются алгоритмы и их эффективная реализация, то сведения по языку C#, приводимые в лекциях 9-16, могут быть полезны для решения тех или иных задач из разделов главы 2.

Цифры. Системы счисления

Для записи чисел привычным способом, знакомым еще с первых классов школы, является их запись в позиционной системе счисления. Напомним некоторые факты. В позиционной системе счисления всегда есть цифра 1. Считается, что единицу придумал бог, а остальные цифры придуманы человеком. Если так, то наиболее замечательной из человеческих придумок в этой области является введение цифры 0. Цифры позиционной системы упорядочены и каждая получатся из предыдущей прибавлением единицы. Число различных цифр в позиционной системе счисления задает основание системы счисления – p. В привычной для нас десятичной системе счисления p = 10 и цифрами являются знакомые всем символы: 0, 1, 2, … 9. В двоичной системе счисления цифр всего две – 0 и 1 и p = 2. В шестнадцатеричной системе счисления p =16 и привычных символов для обозначения цифр не хватает, так что дополнительно используются большие буквы латинского алфавита: 0, 1, 2, … 9, A, B, C, D, E, F, где A задает 10, а F – цифру 15. Поскольку в любой позиционной системе счисления цифры задают числа от 0 до p-1, то для числа p уже нет специального символа. Как следствие, в любой позиционной системе счисления основание системы счисления представляется числом 10, так что справедливы следующие соотношения: $2_{(10)} = 10_{(2)}$; $16_{(10)} = 10_{(16)}$; $p_{(10)} = 10_{(p)}$. Здесь и в дальнейшем при записи числа при необходимости будем указывать в круглых скобках и систему счисления. В обыденной жизни непреложным фактом является утверждение «2*2=4». Мы понимаем, что столь же верным является утверждение «2*2 = 11» (в троичной системе счисления), или, если хотите «2*2 = 10», – все зависит от системы счисления, в которой ведутся вычисления.

Целые числа в позиционных системах счисления записываются в виде последовательности подряд идущих цифр: $N=c_nc_{n-1}\ldots c_0$. Эта запись стала настолько естественной, что иногда теряется ее исконный смысл: $N = c_n10^n + c_{n-1}10^{n-1} + \ldots + c_110^1 + c_010^0$, при котором цифры в записи числа представляют собой коэффициенты разложения числа N по степеням основания, так что вклад каждой цифры в число определяется как самой цифрой, так и ее позицией k и равен c_k10^k. По причине того, что вклад каждой цифры в число зависит от ее позиции, система счисления и называется позиционной.

Запись чисел в позиционной системе легко обобщается и на числа с дробной частью: $N=c_nc_{n-1}\ldots c_0,d_1\ldots d_m$, где дробная часть отделяется от целой символом запятой или точки. И в этом случае остается справедливым разложение числа N по степеням основания, в котором цифры дробной части задают коэффициенты для отрицательных степеней основания:

$$N = c_n10^n + c_{n-1}10^{n-1} + \ldots + c_110^1 + c_010^0 + d_110^{-1} + d_210^{-2} + \ldots + d_m10^{-m} \qquad (1)$$

Понимание соотношения (1) достаточно для решения большинства задач, рассматриваемых в этом разделе, являющихся частными случаями следующей задачи: дано представление числа в системе с основанием p, требуется найти его представление в системе с основанием q. (Пример: найти запись числа $N=2BA37,5F_{(16)}$ в троичной системе счисления).

Рассмотрим возможную схему решения подобных задач. Зная основание системы счисления p и цифры в записи числа в этой системе счисления, нетрудно вычислить значение числа N в десятичной системе счисления. Для этого достаточно воспользоваться соотношением (1), в котором значения цифр и основание системы счисления – p задаются в десятичной системе и в ней же ведутся вычисления. Например:

$N=2BA37,5F_{(16)} = 2*16^4 + 11*16^3 +10*16^2 + 3*16^1 +7 + 5*16^{-1} + 15*16^{-2}$

Сложнее, но достаточно просто решается и обратная задача. Зная значение числа N в десятичной системе, нетрудно получить цифры, задающие его запись в системе с основанием q. Представим число N в виде суммы целой и дробной частей N= C+D. Рассмотрим схему получения цифр отдельно для целой части C и дробной – D. Пусть в системе с основанием q число C представимо в виде $C = c_n c_{n-1} \ldots c_0$. Тогда нетрудно получить его последнюю цифру c_0 и число $M = c_n c_{n-1} \ldots c_1$, полученное отбрасыванием последней цифры в записи числа C. Для этого достаточно воспользоваться операциями деления нацело и получения остатка при делении нацело: $c_0 = C\%p$; $M = C/p$;

Применяя этот прием n раз, получим все цифры в записи числа C. Заметьте, имеет место соотношение: N= (N/p)*p + N%p, где в соответствии с языком C# операция / означает деление нацело, а % – остаток от деления нацело. Чтобы получить все цифры и сохранить их в массиве, достаточно эту схему вставить в соответствующий цикл, что схематично можно представить следующим почти программным текстом:

```
M=C; i=0;
while(M!=0)
{
  c=M%p; M=M/p; Ar[i] =c; i++;
}
```

Для получения цифр дробной части применяется та же схема, но с некоторой модификацией. Если цифры целой части вычисляются, начиная с последней, младшей цифры числа, то цифры дробной части вычисляются, начиная с первой цифры после запятой. Для ее получения достаточно имеющуюся дробь умножить на основание системы счисления и в полученном результате взять целую часть. Для получения последующих цифр этот процесс следует применять к числу M, представляющему дробь, из которой удалена первая цифра: $d_1 =[D*p]$; $m=\{d*p\}$. Здесь [x] и {x} означают соответственно взятие целой и дробной части числа x. Хотя напрямую этих операций нет в языке C#, но их достаточно просто выразить имеющимися средствами этого языка.

Чтобы перейти от системы счисления p к системе счисления q, всегда ли следует использовать десятичную систему в качестве промежуточной: p→ 10 → q? Нет, не всегда. В ряде случаев удобнее использовать прием, основанный на следующем утверждении:

Если основания систем счисления связаны соотношением $p = q^k$, то для перехода от записи числа в системе с основанием p к записи числа в системе с основанием q достаточно каждую цифру системы p представить группой из k цифр системы q.

Для обратного перехода из q в p достаточно сгруппировать цифры системы q и каждую группу из k цифр заменить одной цифрой системы p. Доказательство этих утверждений оставляю читателю.

Для программистов особую важность представляют три системы счисления – двоичная, восьмеричная и шестнадцатеричная, основания которых связаны упомянутым соотношением: $16 = 2^4$; $8 = 2^3$. Рассмотрим число N, записанное в шестнадцатеричной системе $N = 2A,3E_{(16)}$. Чтобы получить его запись в двоичной системе, каждую цифру запишем в двоичной системе, представив ее группой из четырех двоичных цифр. Нетрудно видеть, что $N = 00101010,00111110_{(2)}$. Незначащие нули слева и справа могут быть отброшены, так что окончательно имеем: $N = 101010,0011111_{(2)}$. Переведем это число в восьмеричную систему. Для этого достаточно провести группирование по три цифры влево и вправо от запятой соответственно для целой и дробной части, так что получим: $N = 52,174_{(8)}$.

Римская система счисления

Еще сегодня для записи целых чисел, в особенности дат, используется римская система счисления. Эта система записи чисел не является позиционной. В ее основе лежит понятие человеческих рук с их пятью и десятью пальцами. Поэтому в этой системе есть цифры 1, 5 и 10, записываемые с помощью символов I, V, X. Помимо этого есть еще четыре цифры – 50, 100, 500, 1000, задаваемые символами L, C, D, M. В этой системе нет цифры 0 и она не является позиционной. Согласно правилам системы с помощью цифр римской системы можно записать все целые числа, не превышающие 4000. Как обычно запись числа представляет собой последовательность подряд идущих цифр, а значением числа является сумма цифр в его записи. Например число III означает I + I + I = 3. В записи числа старшие цифры предшествуют младшим, например CVI = C+V+I = 100+5+1=106. Из этого правила есть одно исключение. Младшая цифра может предшествовать старшей цифре и тогда вместо сложения применяется вычитание, например CIX = C+X-I = 100+10-1=109.

Задачи

2.1 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры. Получить n – число цифр в записи числа и целочисленный массив DigitsN такой, что $DigitsN[i] = c_i$.

2.2 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры. Получить строку strN, задающую запись числа N.
Указание: конечно, можно воспользоваться стандартным методом ToString, но в задаче требуется дать собственную реализацию этого метода.

2.3 Дано дробное число $N = 0.d_{m-1}\ldots d_0$, где d_i – это цифры. Получить m – число цифр в записи числа и целочисленный массив FractN такой, что $FractN[i] = d_i$.

2.4 Дано дробное число $N = 0.\ d_{m-1}\ldots d_0$, где d_i – это цифры. Получить строку strN, задающую запись числа N.
Указание: конечно, можно воспользоваться стандартным методом ToString, но в задаче требуется дать собственную реализацию этого метода.

2.5 Дано вещественное число с целой и дробной частью $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры. Получить n и m – число цифр в записи целой и дробной части числа и целочисленный массив DigitsN из n+m элементов такой, что первые его n элементов содержат цифры целой части, а последние m элементов – дробной.

2.6 Дано вещественное число с целой и дробной частью $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры. Получить строку strN, задающую запись числа N.

2.7 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в двоичную систему счисления $N = b_{k-1}\ldots b_0$, получить k – число цифр и целочисленный массив DigitsN такой, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в двоичной системе счисления.
Пример: $N = 17_{(10)} = 10001_{(2)}$

2.8 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в троичную систему счисления $N = b_{k-1}\ldots b_0$, получить k – число цифр и целочисленный массив DigitsN такой, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в троичной системе счисления.
Пример: $N = 17_{(10)} = 122_{(3)}$

2.9 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в четверичную систему счисления $N = b_{k-1}\ldots b_0$, получить k – число цифр и целочисленный массив DigitsN такой, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в четверичной системе счисления.
Пример: $N = 17_{(10)} = 101_{(4)}$

2.10 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в восьмеричную систему счисления $N = b_{k-1}\ldots b_0$, получить k – число цифр и целочисленный массив DigitsN такой, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в восьмеричной системе счисления.
Пример: $N = 17_{(10)} = 21_{(8)}$

2.11 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в шестнадцатеричную систему счисления $N = b_{k-1}\ldots b_0$, получить k – число цифр и целочисленный массив DigitsN такой, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в шестнадцатеричной системе счисления.
Пример: $N = 17_{(10)} = 11_{(16)}$

2.12 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в двоичную систему счисления $N = b_{k-1}\ldots b_0$. Получить строку strN, задающую запись числа N.
Пример: $N = 17_{(10)} = 10001_{(2)}$

2.13 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в троичную систему счисления $N = b_{k-1}\ldots b_0$. Получить строку strN, задающую запись числа N.
Пример: $N = 17_{(10)} = 122_{(3)}$

2.14 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в четверичную систему счисления $N = b_{k-1}\ldots b_0$. Получить строку strN, задающую запись числа N.
Пример: $N = 17_{(10)} = 101_{(4)}$

2.15 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в восьмеричную систему счисления $N = b_{k-1}\ldots b_0$. Получить строку strN, задающую запись числа N.
Пример: $N = 17_{(10)} = 21_{(8)}$

2.16 Дано целое число $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Перевести число N в шестнадцатеричную систему счисления $N = b_{k-1}\ldots b_0$. Получить строку strN, задающую запись числа N.
Пример: $N = 17_{(10)} = 11_{(16)}$

2.17 Дано дробное число $N = 0.d_{m-1}\ldots d_0$, где d_i – это цифры десятичной системы счисления. Перевести число N в двоичную систему счисления $N = 0.b_{k-1}\ldots b_0$, вычислив k цифр в его записи, сохраняя их в целочисленном массиве DigitsN таком, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в двоичной системе счисления.
Пример: $N = 0.17_{(10)} = 0.001010111_{(2)}$ при k=9.

2.18 Дано дробное число $N = 0.d_{m-1}\ldots d_0$, где d_i – это цифры десятичной системы счисления. Перевести число N в троичную систему счисления $N = b_{k-1}\ldots b_0$, вычислив k цифр в его записи, сохраняя их в целочисленном массиве DigitsN таком, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в троичной системе счисления.
Пример: $N = 0.17_{(10)} = 0.01112_{(3)}$ при k=5.

2.19 Дано дробное число $N = 0.d_{m-1}\ldots d_0$, где d_i – это цифры десятичной системы счисления. Перевести число N в четверичную систему счисления $N = b_{k-1}\ldots b_0$, вычислив k цифр в его записи, сохраняя их в целочисленном массиве DigitsN таком, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в четверичной системе счисления.
Пример: $N = 0.17_{(10)} = 0.02232_{(4)}$ при k=5.

2.20 Дано дробное число $N = 0.d_{m-1}\ldots d_0$, где d_i – это цифры десятичной системы счисления. Перевести число N в восьмеричную систему счисления $N = b_{k-1}\ldots b_0$, вычислив k цифр в его записи, сохраняя их в целочисленном массиве DigitsN таком, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в восьмеричной системе счисления.
Пример: $N = 0.17_{(10)} = 0.127_{(8)}$ при k=3.

2.21 Дано дробное число $N = 0.d_{m-1}\ldots d_0$, где d_i – это цифры десятичной системы счисления. Перевести число N в шестнадцатеричную систему счисления $N = b_{k-1}\ldots b_0$, вычислив k цифр в его записи, сохраняя их в целочисленном массиве DigitsN таком, что $DigitsN[i] = b_i$, где b_i – это цифры в записи числа N в шестнадцатеричной системе счисления.
Пример: $N = 0.17_{(10)} = 0.1B_{(16)}$ при k=5.

2.22 Дано вещественное число с целой и дробной частью $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры десятичной системы счисления. Перевести число N в двоичную систему счисления с заданной точностью, вычислив k цифр дробной части числа. Получить строку strN,

задающую запись числа N в этой системе счисления.
Пример: $N = 17.17_{(10)} = 10001.001010111_{(2)}$

2.23 Дано вещественное число с целой и дробной частью $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры десятичной системы счисления. Перевести число N в троичную систему счисления с заданной точностью, вычислив k цифр дробной части числа. Получить строку strN, задающую запись числа N в этой системе счисления.
Пример: $N = 17.17_{(10)} = 122.01112_{(3)}$

2.24 Дано вещественное число с целой и дробной частью $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры десятичной системы счисления. Перевести число N в четверичную систему счисления с заданной точностью, вычислив k цифр дробной части числа. Получить строку strN, задающую запись числа N в этой системе счисления.
Пример: $N = 17.17_{(10)} = 101.02232_{(4)}$

2.25 Дано вещественное число с целой и дробной частью $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры десятичной системы счисления. Перевести число N в восьмеричную систему счисления с заданной точностью, вычислив k цифр дробной части числа. Получить строку strN, задающую запись числа N в этой системе счисления.
Пример: $N = 17.17_{(10)} = 21.127_{(8)}$

2.26 Дано вещественное число с целой и дробной частью $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры десятичной системы счисления. Перевести число N в шестнадцатеричную систему счисления с заданной точностью, вычислив k цифр дробной части числа. Получить строку strN, задающую запись числа N в этой системе счисления.
Пример: $N = 17.17_{(10)} = 11.1B_{(16)}$

2.27 Дана строка, задающая представление целого числа $N = c_{n-1}\ldots c_0$, где c_i – это цифры десятичной системы счисления. Получить число N.
Эту задачу можно сформулировать и так: задайте собственную реализацию метода ToInt32 класса Convert.

2.28 Дана строка, задающая представление вещественного числа с целой и дробной частью: $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры десятичной системы счисления. Получить число N.
Эту задачу можно сформулировать и так: задайте собственную реализацию метода ToDouble класса Convert.

2.29 Дана строка, задающая в двоичной системе счисления представление целого числа $N = c_{n-1}\ldots c_0$, где c_i – это цифры двоичной системы счисления. Получить число N.
Эту задачу можно рассматривать, как расширение класса Convert: добавление метода FromBinaryToInt32.

2.30 Дана строка, задающая в двоичной системе счисления представление вещественного числа с целой и дробной частью: $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры двоичной системы счисления. Получить число N.
Эту задачу можно рассматривать, как расширение класса Convert: добавление метода FromBinaryToDouble.

2.31 Дана строка, задающая в шестнадцатеричной системе счисления представление целого числа $N = c_{n-1}\ldots c_0$, где c_i – это цифры шестнадцатеричной системы счисления. Получить число N.
Эту задачу можно рассматривать, как расширение класса Convert: добавление метода FromHexToInt32.

2.32 Дана строка, задающая в двоичной системе счисления представление вещественного числа с целой и дробной частью: $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры двоичной системы счисления. Получить число N.
Эту задачу можно рассматривать, как расширение класса Convert: добавление метода FromHexToDouble.

2.33 Дана строка, задающая в двоичной системе счисления представление целого числа $N = c_{n-1}\ldots c_0$, где c_i – это цифры двоичной системы счисления. Получить строки str4N, str8N, str16N, задающие представление числа N в системах счисления: четверичной, восьмеричной, шестнадцатеричной.

Указание. *Используйте группирование цифр при переводе из одной системы счисления в другую.*

2.34 Дана строка, задающая в двоичной системе счисления представление вещественного числа с целой и дробной частью: $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры двоичной системы счисления. Получить строки str4N, str8N, str16N, задающие представление числа N в системах счисления: четверичной, восьмеричной, шестнадцатеричной.
Указание. *Используйте группирование цифр при переводе из одной системы счисления в другую.*

2.35 Дана строка, задающая в восьмеричной системе счисления представление целого числа $N = c_{n-1}\ldots c_0$, где c_i – это цифры восьмеричной системы счисления. Получить строки str4N, str2N, str16N, задающие представление числа N в системах счисления: четверичной, двоичной, шестнадцатеричной.
Указание. *Используйте группирование цифр при переводе из одной системы счисления в другую.*

2.36 Дана строка, задающая в восьмеричной системе счисления представление вещественного числа с целой и дробной частью: $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры восьмеричной системы счисления. Получить строки str4N, str2N, str16N, задающие представление числа N в системах счисления: четверичной, двоичной, шестнадцатеричной.
Указание. *Используйте группирование цифр при переводе из одной системы счисления в другую.*

2.37 Дана строка, задающая в шестнадцатеричной системе счисления представление целого числа $N = c_{n-1}\ldots c_0$, где c_i – это цифры шестнадцатеричной системы счисления. Получить строки str4N, str8N, str2N, задающие представление числа N в системах счисления: четверичной, восьмеричной, двоичной.
Указание. *Используйте группирование цифр при переводе из одной системы счисления в другую.*

2.38 Дана строка, задающая в шестнадцатеричной системе счисления представление вещественного числа с целой и дробной частью: $N = c_{n-1}\ldots c_0.d_{m-1}\ldots d_0$, где c_i, d_j – это цифры шестнадцатеричной системы счисления. Получить строки str4N, str8N, str2N, задающие представление числа N в системах счисления: четверичной, восьмеричной, двоичной.
Указание. *Используйте группирование цифр при переводе из одной системы счисления в другую.*

2.39 (*) Заданы p и q – основания двух систем счисления, strN – строка, задающая представление вещественного числа N в системе с основанием p. Получить строку, задающую представление числа N в системе с основанием q, возможно с некоторой точностью, заданной параметром k – числом цифр дробной части числа N при его записи в системе с основанием q.

2.40 (*) Дано число N и основание системы счисления p. Получить c_i – коэффициенты разложения числа N по степеням основания с заданной точностью Eps.
Указание: В данной задаче предполагается, что N, p, c_i являются вещественными числами и для c_i выполняется условие ($c_i < p$). Пример: N=30; p=2,5; $c_3 = 1$; $c_2 = 1,5$; $c_1 = 2$; $c_0 = 0$;

$$N = 1*2,5^3 + 1,5*2,5^2 + 2*2,5^1 + 0*2,5^0 = 30$$

2.41 Дано основание системы счисления p и c_i – коэффициенты разложения числа N по степеням основания. Получить число N.
Указание: В данной задаче предполагается, что N, p, c_i являются вещественными числами и для c_i выполняется условие ($c_i < p$). Пример: p=2,5; $c_3 = 1$; $c_2 = 1,5$; $c_1 = 2$; $c_0 = 0$;

$$N = 1*2,5^3 + 1,5*2,5^2 + 2*2,5^1 + 0*2,5^0 = 30$$

2.42 (*) Дана строка strRome, задающая представление целого числа N, меньшего 4000, в непозиционной римской системе счисления. Получить число N.
Пример: N= MMIV = 2004

2.43 (*) Дано целое число N, меньшее 4000. Получить строку strRome, задающую представление числа в непозиционной римской системе счисления.
Пример: N=2005 =MMV

Именованные числа

С давних пор числа применяются для измерения физических величин – длин, площадей, объемов. Как правило, при этом использовалась системы, вовсе не основанные на десятичной системе, а связанные с реально применяемыми мерами – бочонками, мешками и прочей применяемой тарой. Метрическая система мер, основанная на десятичной системе счисления, завоевала свои позиции лишь в последние два столетия, и мы стали применять километры и килограммы, килоджоули и килогерцы. Но рецидивы все еще дают себя знать, и примером тому является программисты, которые сравнительно недавно ввели систему мер для измерения объема информации. У нас байт равен 8 битам, а килобайт равен не 1000 байтов, как мог бы ожидать человек, далекий от программирования, а - 1024 байта. И связано это с любовью компьютеров к двоичной системе счисления, в которой 1 байт = $1000_{(2)}$ битов, а 1 Кб = $10000000000_{(2)}$ байтов.

Задачи

2.44 Задано число T (температура) и единица измерения (C – градусы по Цельсию, F – по Фаренгейту, R – по Реомюру, K – по Кельвину). Определить значения температуры в других шкалах, используя следующие соотношения:

$T_F = 1,8T_C + 32; \quad T_R = 0,8T_C; \quad T_K = T_C + 273,2;$

Пример: -40 (C) = -40 (F) = -32 (R) = 233,2 (K)

2.45 Дано число N, задающее расстояние, измеренное с точностью до долей миллиметра. Получите строку, задающее расстояние с использованием старинной китайской системы, в которой справедливы следующие соотношения:

$0,1мм. = ху; \quad 1мм. = мяо; \quad 1см. = хао; \quad 0,1м. = ли;$

$1м. = фэн; \quad 10м. = цунь; \quad 100м. = чжи; \quad 1км. = чан;$

2.46 Дано число N, задающее расстояние, измеренное с точностью до долей миллиметра. Получите строку, задающее расстояние с использованием старинной древнерусской системы, в которой справедливы следующие соотношения:

$1\,вершок = 44,45мм.; \quad 1четверть = 4\,вершка; \quad 1аршин = 4\,четверти;$

$1\,сажень = 3\,аршина; \quad 1верста = 500\,саженей; \quad 1миля = 7\,верст.$

2.47 Дано число N, задающее расстояние, измеренное с точностью до долей миллиметра. Получите строку, задающее расстояние с использованием английской системы мер длины, в которой справедливы следующие соотношения:

$1\,точка\,(po\,\text{int}) = 0,3528мм.; \quad 1\,линия\,(line) = 6\,точек; \quad 1дюйм = 12\,линий;$

$1\,фут = 12\,дюймов; \quad 1\,ярд = 3\,фута; \quad 1фарлонг = 220\,ярдов; \quad 1миля = 8\,фарлонгов.$

2.48 Дано вещественное число N, задающее объем хранимых данных в терабайтах. Выразите значение N в гигабайтах, мегабайтах, килобайтах, байтах, битах.

Классификация чисел

Числа разделяются на классы. Целые положительные числа – N = {1, 2, 3, … } – составляют множество натуральных чисел. Зачастую и 0 считают натуральным числом.

Множество целых чисел Z включает в себя все натуральные числа, число 0 и все натуральные числа, взятые со знаком минус: Z = {0, 1, -1, 2, -2, …}.

Каждое рациональное число x можно задать парой целых чисел (m, n), где m является числителем, n – знаменателем числа: x = m/n. Эквивалентным представлением рационального числа является его задание в виде числа, записанного в позиционной десятичной системе счисления, где дробная часть числа может быть конечной или бесконечной периодической дробью. Например, число x = 1/3 = 0,(3) представляется бесконечной периодической дробью.

Числа, задаваемые бесконечными непериодическими дробями, называются иррациональными числами. Таковыми являются, например все числа вида √p, где p – простое число. Иррациональными являются известные всем числа π и e.

Объединение множеств целых, рациональных и иррациональных чисел составляет множество вещественных чисел. Геометрическим образом множества вещественных чисел является прямая линия – вещественная ось, где каждой точке оси соответствует некоторое вещественное число, так что вещественные числа плотно и непрерывно заполняют всю вещественную ось.

Плоскость представляет геометрический образ множества комплексных чисел, где вводятся уже две оси – вещественная и мнимая. Каждое комплексное число, задаваемое парой вещественных чисел, представимо в виде: x = a+b*i, где a и b – вещественные числа, которые можно рассматривать, как декартовы координаты числа на плоскости.

Делители и множители

Рассмотрим сейчас классификацию, которая делит множество натуральных чисел на два подмножества – простых и составных чисел. В основе этой классификации лежит понятие делимости натуральных чисел. Если n делится нацело на d, то говорят, что d «делит» n,что записывают в виде: d|n. Заметьте, это определение, возможно, не соответствует интуитивному пониманию: d «делит» n, если n делится на d, а не наоборот. Число d называется делителем числа n. У каждого числа n есть два тривиальных делителя – 1 и n. Делители, отличные от тривиальных, называются множителями числа n. Число n называется простым, если у него нет делителей, отличных от тривиальных. Простые числа делятся только на 1 и сами на себя. Числа, у которых есть множители, называются составными. Число 1 является особым числом, поскольку не относится ни к простым, ни к составным числам. Отрицательные числа также не относятся ни к простым, ни к составным, но всегда можно рассматривать модуль числа и относить его к простым или составным числам.

Любое составное число N можно представить в виде произведения его множителей: N = q1 * q2 * … * qk. Это представление не единственно, например 96 = 8*12 = 2*3*16. Однако для каждого составного числа N существует единственное представление в виде произведения степеней простых чисел: $N = p_1^{r1} * p_2^{r2} * \ldots * p_k^{rk}$, где pi – простые числа и $p_k < p_{k+1}$, называемое разложением числа N на простые множители. Например $96 = 2^5 * 3^1$.

Если d|m и d|n, то d является общим делителем чисел m и n. Среди всех общих делителей можно выделить наибольший общий делитель, обозначаемый как НОД(m,n). Если НОД(m,n) = 1, то числа m и n называются взаимно простыми. Простые числа взаимно просты, так что НОД(q,p) =1, если q и p – простые числа.

Если m|A и n|A, то A является общим кратным чисел m и n. Среди всех общих кратных можно выделить наименьшее общее кратное, обозначаемое как НОК(m,n). Если НОК(m,n) = m*n, то числа m и n являются взаимно простыми. НОК(q, p) =q*p, если q и p – простые числа.

Если через P_m и Q_n обозначить множества всех простых множителей чисел m и n, то

$$НОД(m,n) = \prod d, \quad где \quad d \in P_m \cap Q_n$$
$$НОК(m,n) = \prod d, \quad где \quad d \in P_m \cup Q_n$$

Если получено разложение чисел m и n на простые множители, то, используя приведенные соотношения, нетрудно вычислить НОД(m,n) и НОК(m,n). Существуют и более эффективные алгоритмы, не требующие разложения числа на множители.

Алгоритм Эвклида

Эффективный алгоритм вычисления НОД(m,n) предложен еще Эвклидом. Он основывается на следующих свойствах НОД(m,n), доказательство которых предоставляется читателю:

$$НОД(m,m) = m;$$
$$НОД(m,n) = НОД(n,m);$$
$$НОД(m,n) = НОД(m-n,n);$$

Если m > n, то по третьему свойству его можно уменьшить на величину n. Если же m < n, то по второму свойству аргументы можно поменять местами и вновь придти к ранее рассмотренному

случаю. Когда же в результате этих преобразований значения аргументов сравняются, то решение будет найдено. Поэтому можно предложить следующую схему:

```
while(m != n)
{
  if(m < n) swap(m,n);
  m = m - n;
}
return(m);
```

Здесь процедура swap выполняет обмен значениями аргументов.

Если немного подумать, то становится ясно, что вовсе не обязательно обмениваться значениями, – достаточно на каждом шаге цикла изменять аргумент с максимальным значением. В результате приходим к схеме:

```
while(m != n)
{
  if(m > n) m = m - n;
  else n = n – m;
}
return(m);
```

Если еще немного подумать, то можно улучшить и эту схему, перейдя к циклу с тождественно истинным условием:

```
while(true)
{
  if(m > n) m = m - n;
  else if (n > m) n = n – m;
  else return(m);
}
```

Последняя схема хороша тем, что в ней отчетливо видна необходимость доказательства завершаемости этого цикла. Доказать завершаемость цикла нетрудно, используя понятие варианта цикла. Для данного цикла вариантом может служить целочисленная функция – max(m,n), которая уменьшается на каждом шаге, оставаясь всегда положительной.

Достоинством данной версии алгоритма Эвклида является и то, что на каждом шаге используется элементарная и быстрая операция над целыми числами – вычитание. Если допустить операцию вычисления остатка при делении нацело, то число шагов цикла можно существенно уменьшить. Справедливо следующее свойство:

$$НОД(m,n) = НОД(n, m\%n); \quad если\ m > n$$

Это приводит к следующей схеме:

```
int temp;
if(n>m) temp = m; m = n; n = temp; //swap(m,n)
while(m != n)
{
  temp = m; m = n; n = temp%n;
}
```

Для вычисления НОК(m, n) можно воспользоваться следующим соотношением:

$$НОК(m,n) = m * n / НОД(m,n)$$

А можно ли вычислить НОК(m, n), не используя операций умножения и деления? Оказывается можно одновременно с вычислением НОД(m,n) вычислять и НОК(m,n). Вот соответствующая схема:

```
int x = v = m, y = u = n,;
while(x != y)
{
  if(x > y){  x = x - y; v = v + u;}
  else {y = y – x; u = u + v;}
}
НОД = (x + y)/2; НОК = (u+v)/2;
```

Доказательство того, что эта схема корректно вычисляет НОД, следует из ранее приведенных свойств НОД. Менее очевидна корректность вычисления НОК. Для доказательства заметьте, что инвариантом цикла является следующее выражение:

$$v * y / НОД + u * x / НОД = 2 * НОК$$

Это соотношение выполняется после инициализации переменных до начала выполнения цикла. По завершении цикла, когда x и y становятся равными НОД, из истинности инварианта следует корректность схемы. Нетрудно проверить, что операторы тела цикла оставляют утверждение истинным. Детали доказательства оставляются читателям.

Понятие НОД и НОК можно расширить, определив их для всех целых чисел. Справедливы следующие соотношения:

$$НОД(m,0) = m; \quad НОК(m,0) = 0;$$
$$НОД(m,n) = НОД(abs(m), abs(n))$$
$$НОД(0,0) - \quad \textit{не определено}$$

Расширенный алгоритм Эвклида

Иногда полезно представлять НОД(m,n) в виде линейной комбинации m и n:

$$НОД(m,n) = a * m + b * n;$$

В частности вычисление коэффициентов a и b понадобится позже при рассмотрении алгоритма RSA – шифрования с открытым ключом. Приведу схему алгоритма, позволяющую вычислить тройку – d, a, b – наибольший общий делитель и коэффициенты разложения. Алгоритм удобно реализовать в виде рекурсивной процедуры ExtendedEuclid(m,n,d,a,b), которая по заданным входным аргументам m и n вычисляет значения аргументов d, a, b. Нерекурсивная ветвь этой процедуры соответствует случаю n=0, возвращая в качестве результата значения: d=m, a=1, b=0. Рекурсивная ветвь, построенная в предположении, что m>n, рекурсивно вызывает ExtendedEuclid(n, m % n, d, a, b) и затем изменяет полученные в результате вызова значения a и b следующим образом:

int t=b; b= a-b*(m/n); a=t;

Доказательство корректности этого алгоритма построить нетрудно. Для нерекурсивной ветви корректность очевидна, а для рекурсивной ветви нетрудно показать, что из истинности результата, возвращаемого при рекурсивном вызове, следует его истинность для входных аргументов после пересчета значений a и b.

Задачи

2.49 Даны m и n – натуральные числа. Вычислите НОД(m, n). При вычислениях не используйте операций умножения и деления.

2.50 Даны m и n – натуральные числа. Вычислите НОК(m, n).

2.51 Даны m и n – натуральные числа. Вычислите НОК(m, n). При вычислениях не используйте операций умножения и деления.

2.52 Даны m и n – целые числа. Вычислите НОД(m, n). При вычислениях не используйте операций умножения и деления.

2.53 Даны m и n – целые числа. Вычислите НОК(m, n). При вычислениях не используйте операций умножения и деления.

2.54 Даны m и n – целые числа. Вычислите НОД(m, n). При вычислениях используйте операцию взятия остатка от деления нацело.

2.55 Даны m и n – целые числа. Вычислите НОК(m, n). При вычислениях используйте операцию взятия остатка от деления нацело.

2.56 Даны m и n – целые числа. Вычислите тройку чисел – (d, a, b), используя расширенный алгоритм Эвклида.

2.57 Даны m и n – натуральные числа. Представьте НОД(m, n) в виде линейной комбинации m и n.

2.58 Даны m и n – целые числа. Представьте НОД(m, n) в виде линейной комбинации m и n.

2.59 Даны m и n – целые числа. Проверьте, являются ли числа m и n взаимно простыми.

Простые числа

Среди четных чисел есть только одно простое число – это 2. Простых нечетных чисел сколь угодно много. Нетрудно доказать, что число N = (p1 * p2 * … * pk) +1, где pi – подряд идущие простые числа, является простым. Так что, если построено k простых чисел, то можно построить еще одно простое число p, большее p_k. Отсюда следует, что множество простых чисел неограниченно. Пример: число N = 2*3*5*7 + 1 = 211 является простым числом.

Решето Эратосфена

Как определить, что число N является простым? Если допустима операция N % m, дающая остаток от деления числа N на число m, то простейший алгоритм состоит в проверке того, что остаток не равен нулю при делении числа N на все числа m, меньшие N. Очевидным улучшением этого алгоритма является сокращение диапазона проверки – достаточно рассматривать числа m в диапазоне [2, √N].

Еще в 3-м веке до н.э. древнегреческий математик Эратосфен предложил алгоритм нахождения простых чисел в диапазоне [3, N], не требующий операций деления. Этот алгоритм получил название «Решето Эратосфена». В компьютерном варианте идею этого алгоритма можно описать следующим образом. Построим массив Numbers, элементы которого содержат подряд идущие нечетные числа, начиная с 3. Вначале все числа этого массива считаются невычеркнутыми. Занесем первое невычеркнутое число из этого массива в массив SimpleNumbers – и это будет первое нечетное простое число (3). Затем выполним просеивание, проходя по массиву Numbers с шагом, равным найденному простому числу, вычеркивая все попадающиеся при этом проходе числа. При первом проходе будет вычеркнуто число 3 и все числа, кратные 3. На следующем проходе в таблицу простых чисел будет занесено следующее простое число 5, а из массива Numbers будут вычеркнуты числа, кратные 5. Процесс повторяется, пока не будут вычеркнуты все числа в массиве Numbers. В результате массив SimpleNumbers будет содержать таблицу первых простых чисел, меньших N.

Этот алгоритм хорош для нахождения сравнительно небольших простых чисел. Но если потребуется найти простое число с двадцатью значащими цифрами, то памяти компьютера уже не хватит для хранения соответствующих массивов. Замечу, что в современных алгоритмах шифрования используются простые числа, содержащие несколько сотен цифр.

Плотность простых чисел

Мы показали, что число простых чисел неограниченно. Понятно, что их меньше, чем нечетных чисел, но насколько меньше? Какова плотность простых чисел? Пусть π(n) – это функция, возвращающая число простых чисел, меньших n. Точно задать эту функцию не удается, но для нее есть хорошая оценка. Справедлива следующая теорема:

$$\pi(n) \underset{n\to\infty}{\to} n/\ln(n)$$

Функция π(n) асимптотически сверху приближается к своему пределу, так что оценка дает слегка заниженные значения. Эту оценку можно использовать в алгоритме решета Эратосфена для выбора размерности массива SimpleNumbers, когда задана размерность массива Numbers, и, наоборот, при заданной размерности таблицы простых чисел можно выбрать подходящую размерность для массива Numbers.

Табличный алгоритм определения простоты чисел

Если хранить таблицу простых чисел SimpleNumbers, в которой наибольшее простое число равно M, то достаточно просто определить является ли число N, меньшее M^2, простым. Если N меньше M, то достаточно проверить, находится ли число N в таблице SimpleNumbers. Если N больше M, то достаточно проверить, делится ли число N на числа из таблицы SimpleNumbers, не превосходящие значения √N. Понятно, что если у числа N нет простых множителей, меньших √N, то число N является простым.

Использование таблицы простых чисел требует соответствующей памяти компьютера, а следовательно ограничивает возможности этого алгоритма, не позволяя использовать его для нахождения больших простых чисел.

Тривиальный алгоритм

Если N – нечетное число, то проверить, что оно является простым, можно на основе определения простоты числа. При этом не требуется никакой памяти для хранения таблиц чисел, но, как всегда, выигрывая в памяти, проигрываем во времени. Действительно, достаточно проверить, делится ли нацело число N на подряд идущие нечетные числа в диапазоне [3, √N]. Если у числа N есть хоть один множитель, то оно составное, иначе – простое.

Все рассмотренные алгоритмы перестают эффективно работать, когда числа выходят за пределы разрядной сетки компьютера, отведенной для представления чисел, так что если возникает необходимость работы с целыми числами, выходящими за пределы диапазона System.Int64, то задача определения простоты такого числа становится совсем не простой. Существуют некоторые рецепты, позволяющие определить, что число является составным. Вспомним хотя бы известные со школьных времен алгоритмы. Если последняя цифра числа делится на 2, то и число делится на 2. Если две последние цифры числа делятся на 4, то и число делится на 4. Если сумма цифр делится на 3 (на 9), то и число делится на 3 (на 9). Если последняя цифра рана 0 или 5, то число делится на 5. Математики затратили много усилий, доказывая, что то или иное число является (или не является) простым числом. Сейчас есть особые приемы, позволяющие доказать, что числа некоторого вида являются простыми. Наиболее подходящими кандидатами на простоту являются числа вида 2^p -1, где p – это простое число. Например, доказано, что число 2^{19937} - 1, имеющее более 6000 цифр, является простым, но нельзя сказать, какие простые числа являются ближайшими соседями этого числа.

Задачи

2.60 Дано целое N. Используя алгоритм решета Эратосфена, найдите все простые числа, меньшие N.

2.61 Дано целое N. Используя алгоритм решета Эратосфена, найдите N первых простых чисел.

2.62 Дана таблица, содержащая N первых простых чисел. По заданному n < N вычислите разность между функцией π(n) и ее оценкой – n/ln(n).

2.63 Дана таблица, содержащая N первых простых чисел. Используя табличный алгоритм, вычислите все простые числа в диапазоне [M+1, M*M], где M – наибольшее простое число, хранимое в таблице.

2.64 Дано целое N. Постройте таблицу, содержащую N первых нечетных простых чисел. Используйте табличный алгоритм с постепенным заполнением таблицы, начиная со случая, когда в ней хранится только одно простое число 3.

2.65 Дано число N. Определите, является ли оно простым, используя тривиальный алгоритм.

2.66 Дано число N. Определите его первый простой множитель, если он существует.

Вычисления по модулю или модульная арифметика

В теории чисел и в практических алгоритмах большую роль играют вычисления по модулю. Начнем рассмотрение с ряда определений. Если остаток от деления нацело a на n совпадает с остатком от деления нацело b на n, то говорят, что a сравнимо с b по модулю n, и пишут:

$$a = b \pmod{n}$$

Если 0 <= b < n, то b совпадает со своим остатком и является представителем класса, в который входят все числа вида: k*n + b. Число n, заданное в качестве модуля, разбивает все множество целых чисел на n классов эквивалентности, где в каждый класс попадают числа, сравнимые с представителями классов – числами 0, 1, 2, …n -1.

Если для операций деления нацело и получения остатка использовать операции языка C# – / и %, то имеют место следующие соотношения:

$$b = a\%n; \quad k = a/n; \quad a = k*n + b;$$

Здесь следует иметь в виду одно обстоятельство. В качестве представителей классов эквивалентности принято рассматривать неотрицательные числа. Операция получения остатка % возвращает в качестве результата представителя класса, если a – неотрицательное число. Для отрицательных чисел эта операция возвращает отрицательный остаток, который легко превратить в представителя класса, добавив к нему значение модуля. Например, -5%3 = -2. Хотя здесь и не возвращается представитель класса, но это корректный результат в модульной арифметике, поскольку -2 = 1 (mod 3) и -5 = 1 (mod 3).

Если мы хотим найти x – представителя класса для отрицательных чисел в сравнении:

-a = x (mod n),

то можно воспользоваться либо соотношением:

x = (a/n +1)*n – a,

либо соотношением:

x = n – (-a % n)

Пусть

$$a = a' \pmod{n}; \quad b = b' \pmod{n};$$

Здесь a´ и b´ – это представители классов. Тогда справедливо:

$$(a + b) = (a' + b') \pmod{n}$$

$$a * b = a' * b' \pmod{n}$$

В модульной арифметике операции сложения и умножения над целыми числами заменяются соответствующими операциями над представителями чисел, а полученный результат операции заменяется представителем соответствующего класса. Вот пример:

(17 + 13) (mod 7) = (3 + 6) (mod 7) = 9 (mod 7) = 2;

(17 * 13) (mod 7) = (3 * 6) (mod 7) = 18 (mod 7) = 4;

Определим теперь обратные операции к сложению и умножению – вычитание и деление. С вычитанием все обстоит просто. Если верно, что (a + b) = c (mod n), то верно и соотношение a = (c-b) (mod n).

Если же мы хотим корректно определить деление, согласованное с умножением, то необходимо наложить на числа, участвующие в умножении, дополнительные требования. Для согласованного выполнения умножения и деления приходится ограничиваться не всеми представителями классов, а только теми, что являются взаимно простыми с n. Так что операция деления

$$(a/b) = (a'/b') \pmod{n}$$

Эта операция определена только для a´ , b´, взаимно простых с n. В этом случае сравнение b*x =a (mod n) имеет решение, и соответствующий элемент x рассматривается как результат операции деления a/b. Например, уравнение 5*x = 7 (mod 12) имеет решение и x = 7/5 (mod 12) = 11. В то же время уравнение 3*x = 7 (mod 12) решения не имеет.

Сама по себе операция умножения определена на всех числах, но операция умножения, согласованная с делением, определена только на некоторых числах.

Таблицы сложения и умножения

В модульной арифметике результаты сложения и вычитания, умножения и деления полностью описываются таблицами, задающими результаты операций над представителями классов. Для сложения и вычитания эта таблица имеет размерность n*n, где n – значение модуля. Для умножения и деления размерность таблицы может быть меньше n. Приведу пример:

Таблица 2.1 Таблица сложения (вычитания) по модулю n=6

аргументы	**0**	**1**	**2**	**3**	**4**	**5**
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Таблица 2.2 Таблица умножения (деления) по модулю n =12

аргументы	**1**	**5**	**7**	**11**
1	1	5	7	11
5	5	1	11	7
7	7	11	1	5
11	11	7	5	1

Эффективный алгоритм возведения в степень

Поскольку в модульной арифметике определена операция умножения, то естественным образом определяется операция возведения в целую степень $b = a^m \pmod{n} = a * a * \{k \text{ раз}\} * a \pmod{n}$. Эффективный алгоритм возведения в степень позволяет вместо n-1 умножений выполнять таких операций порядка $\log_2 n$.

Пусть даны a, m, n и нужно вычислить $b = a^m \pmod{n}$. Приведу схему алгоритма:

```
int x = a, y = m, z = 1;
while(y != 0)
{
  while(y%2 ==0)
  {
```

```
      x = x * x; y = y/2;
    }
    // y – нечетное
    z = z*x; y = y – 1;
  } b = z;
```

Для доказательства корректности этой схемы введем в качестве инварианта цикла следующее утверждение:

$a^m = z * x^y$;

Очевидно, что в результате инициализации утверждение становится истинным. По завершении цикла, когда y становится равным 0, инвариант обеспечивает требуемый результат. Нетрудно проверить, что операторы тела внешнего цикла сохраняют истинность инварианта. Детали доказательства остаются за читателем.

Модифицированный алгоритм возведения в степень с логарифмической сложностью

Рассмотри еще один алгоритм возведения в степень, имеющий такую же сложность, как и предыдущий алгоритм. Отличие состоит в другом представлении входных данных. В тех ситуациях, когда предстоит возведение в степень, задаваемую очень большим числом, то зачастую его разумно представлять двоичной записью числа – последовательностью битов.

Пусть даны a, m, n и нужно вычислить $b = a^m \pmod n$. И пусть m представлено двоичной записью: $m = m_k\, m_{k-1} \ldots\, m_0$. Приведу схему модифицированного алгоритма:

```
int y = 0, z = 1, i = k+1;
while(i != 0)
{
  i = i – 1;
  y = 2*y + m_i;
  z = z * z;
  if(m_i == 1) z = z*a;
} b = z;
```

Для доказательства корректности этой схемы зададим следующее утверждение в качестве инварианта цикла:

$(z = a^y)$ & (y – префикс числа m)

В результате инициализации первый член конъюнкции становится истинным по определению. Второй член конъюнкции также становится истинным, поскольку под префиксом m будем понимать число, представленное записью: $m_k\, m_{k-1} \ldots\, m_i$. При инициализации конечный индекс i больше начального индекса k, так что последовательность пуста и префикс равен 0, что согласуется с результатами инициализации. На каждом шаге цикла префикс расширяется на очередную двоичную цифру. В полном соответствии с этим изменяется и значение z, так что оба члена конъюнкции, а, следовательно, и инвариант цикла остается истинным. По завершении цикла y совпадает с числом m, и из истинности первого члена конъюнкции следует корректность вычислений.

Заметьте, переменная y не нужна для вычислений значения z, – она введена в интересах доказательства корректности схемы. В реальном алгоритме ее можно использовать на этапе отладки и отключить в работающем алгоритме.

Обе схемы применимы для возведения в степень, как в обычной арифметике, так и в модульной, в которой естественно все операции выполняются по модулю n.

Решение линейных модульных уравнений

Пусть a и n – произвольные положительные целые и b – произвольное целое. Рассмотрим решение уравнения:

$a*x = b \pmod n$

Это уравнение может иметь один или несколько корней, а может и не иметь ни одного корня. Имеют место следующие утверждения, которые приводятся без доказательства:

- Уравнение разрешимо, если и только если НОД(a, n)| b.
- Если уравнение разрешимо, то оно имеет ровно d корней, где d = НОД(a, n).
- Корни уравнения задаются соотношением: $x_0 = c_1(b/d) \pmod n$; $x_k = (x_0 + k\,(n/d)) \pmod n$ для всех k = 1, 2, …d-1. Коэффициент c_1 в этих формулах – это первый коэффициент в линейном разложении $d = c_1a + c_2n$. Его можно определить, используя расширенный алгоритм Эвклида для вычисления d, c_1 и c_2.

На основе этих утверждений легко строится алгоритм решения уравнения. Полагаю, что схему алгоритма можно не приводить. Ограничусь двумя примерами. Рассмотрим уравнение: $72x = 12 \pmod{42}$. Расширенный алгоритм Эвклида, вычисляя НОД(72, 42) и коэффициенты разложения, вернет в качестве результату тройку чисел: 6, 3, -5. Поскольку 6|12, то у нашего уравнения 6 корней и их нетрудно вычислить: 6, 13, 20, 27, 34, 41. В то же время уравнение $72x = 15 \pmod{42}$ не имеет ни одного решения, поскольку 15 не делится нацело на 6.

Задачи

2.67 Проверьте, какие из сравнений имеют место:

$126 = 8 \pmod{13}$; $152 = 9 \pmod{13}$; $152 = 9 \pmod{26}$; $n-1 = -1 \pmod n$

2.68 Докажите и проверьте, что данные сравнения имеют место:

$(a+1)^3 = 1 \pmod a$; $(a+1)^3 = 2a+1 \pmod a$; $(a-1)^3 = -1 \pmod a$; $(a-1)^3 = a-1 \pmod a$;

2.69 Вычислите x – представителя класса в сравнениях:

$175 = x \pmod{96}$; $-175 = x \pmod{97}$; $-1 = x \pmod n$; $k*n + 5 = x \pmod n$;

2.70 Даны целые a, b и натуральное число n. Напишите функцию, возвращающую true, если сравнение $a = b \pmod n$ имеет место.

2.71 Дано целое a и натуральное число n. Напишите функцию, возвращающую значение b, такую, что 0 <= b < n и сравнение $a = b \pmod n$ имеет место.

2.72 Напишите функцию, вычисляющую сумму целых a и b по модулю n.

2.73 Напишите функцию, вычисляющую разность целых a и b по модулю n.

2.74 Напишите функцию, вычисляющую произведение целых a и b по модулю n.

2.75 Напишите функцию, возвращающую частное целых a и b по модулю n. Если деление не определено, то функция возвращает значение -1.

2.76 Даны простые числа a и b. Определите наименьшее n, такое, что операция деления a на b по модулю n определена. Вычислите результат этой операции.

2.77 Для данного натурального n постройте таблицу сложения по модулю n.

2.78 Для данного натурального n постройте таблицу умножения по модулю n.

2.79 Напишите функцию, вычисляющую сумму и разность целых a и b по модулю n, используя построенную таблицу сложения.

2.80 Напишите функцию, вычисляющую произведение и частное целых a и b по модулю n, используя построенную таблицу умножения.

2.81 (*) Постройте Windows-приложение «Модульный калькулятор», выполняющий над аргументами четыре арифметических действия по модулю n: сложение вычитание, умножение, деление.

2.82 Напишите функцию, вычисляющую a^m, где m – целое, за время, пропорциональное $\log_2 m$. Все действия выполняются в обычной арифметике.

2.83 Напишите функцию, вычисляющую a^m, где m – целое, за время, пропорциональное $\log_2 m$. Число m задано в двоичной системе счисления. Все действия выполняются в обычной арифметике.

2.84 Напишите функцию, вычисляющую a^m, где m – целое, за время, пропорциональное $\log_2 m$. Все действия выполняются в модульной арифметике с заданным значением модуля n.

2.85 Напишите функцию, вычисляющую a^m, где m – целое, за время, пропорциональное $\log_2 m$. Число m задано в двоичной системе счисления. Все действия выполняются в модульной арифметике с заданным значением модуля n.

2.86 Напишите процедуру решения линейного модульного уравнения: $ax = b \pmod n$.

2.87 (*) Постройте Windows-приложение «Модульный калькулятор», выполняющий над аргументами операции модульной арифметики: сложения, вычитания, умножения, деления, возведения в целую степень и позволяющий находить решение линейного уравнения.

Проверка простоты чисел

Современные методы шифрования построены на использовании больших простых чисел, состоящих из нескольких сотен цифр. Обычно, двоичная запись таких чисел содержит 512 или 1024 бита. Как определить, что случайно построенное число с таким числом цифр является простым? Обычные алгоритмы, рассмотренные ранее, в этом случае работать не будут. Оказывается, что задача проверки простоты числа существенно проще, чем задача разложения составного числа на множители.

Рассмотрим несколько тестов, которые, хотя и не дают безошибочного ответа, но определяют простоту числа с высокой вероятностью. Заметьте, когда некоторый тест не дает точного ответа, то возможны ошибки первого и второго рода. Ошибка первого рода возникает в ситуации, когда число является простым, а тест называет его составным. Ошибка второго рода возникает в ситуации, когда число является составным, а тест называет его простым. Рассматриваемые в этом разделе тесты никогда не совершают ошибок первого рода, а вероятность ошибок второго рода крайне мала для больших чисел.

Тест на основе теоремы Ферма

Теорема Ферма формулируется следующим образом:

Для любого a, такого что $2 <= a < n$, справедливо: если n – простое число, то $a^{n-1} = 1 \pmod n$.

Следствие: если $a^{n-1} \neq 1 \pmod n$, то n – составное число.

Определение: Число n называется псевдопростым с основанием a, если n является составным, но для него выполняется условие $a^{n-1} = 1 \pmod n$.

Тест 1

Возьмем в качестве базы число 2. Для заданного n проверим выполнение условия $2^{n-1} = 1 \pmod n$. Если это условие не выполняется, то тест сообщает, что число n – составное, в противном случае тест говорит, что число простое (возможно допуская при этом ошибку).

В соответствии со следствием к теореме Ферма тест не делает ошибок, принимая решение о том, что число составное. Он может ошибиться, назвав простым псевдопростое число с основанием 2. Много ли таких чисел? Первым таким числом является число 341. Среди чисел, меньших 10000, таких чисел 22. С ростом n вероятность их появления уменьшается.

Тест 2

Будем выбирать в качестве базы последовательность случайно сгенерированных чисел a_1, a_2, … a_m в диапазоне [2, n-1]. Для заданного n и каждого a_i проверим выполнение условия $a_i^{n-1} = 1 \pmod n$. Если это условие не выполняется хотя бы для одного a_i, то тест сообщает, что число n – составное, в противном случае тест говорит, что число простое (возможно допуская при этом ошибку).

Понятно, что и этот тест никогда не совершает ошибки первого рода, а вероятность ошибки второго рода у него меньше, чем у теста 1, поскольку составное число n может быть псевдослучайным числом с основанием a и не быть таковым для основания b. Так число 341 является псевдослучайным с основанием 2 и не является таковым для основания 3.

Тест Миллера-Рабина

Этот тест является модифицированным вариантом теста 2. Он использует помимо теоремы Ферма еще одно важное утверждение, связанное с простыми числами. Имеет место следующая теорема:

Если n – простое число, то уравнение $x^2 = 1 \pmod n$ имеет ровно 2 тривиальных корня: +1 и -1, или, что то же, 1 и (n-1).

Следствие из теоремы: Если уравнение $x^2 = 1 \pmod n$ имеет нетривиальный корень, то n – составное число.

Это следствие и следствие теоремы Ферма объединяются в тесте Миллера-Рабина следующим образом. Число n-1, представляющее степень, в которую возводится основание a в теореме Ферма, является четным числом. Поэтому двоичная запись этого числа заканчивается k нулями (k >= 1). Представим число n-1 в виде $u*2^k$, где u – это нечетный префикс числа, заканчивающийся 1. После этого префикса в записи числа n-1 идут одни нули. При возведении основания a в степень n-1, как это требуется в тесте Ферма, можно вначале вычислить a^u, а затем k раз выполнять возведение в квадрат. В последнем цикле можно использовать следствие теоремы о корнях квадратного уравнения. Если $a^p \neq (1 \text{ или } n-1) \pmod n$, но на следующем шаге при возведении в квадрат получится значение равное 1, то уравнение $x^2 = 1 \pmod n$ имеет нетривиальный корень и согласно следствию число n является составным.

Таким образом тест Миллера-Рабина позволяет с одной стороны ускорить вычисления, поскольку не всегда приходится полностью вычислять степень основания, с другой стороны используются дополнительные факты, позволяющие отсечь псевдослучайные числа. Однако и в этом случае все еще остается некоторая минимальная вероятность ошибки второго рода.

Задачи

2.88 Построить процедуру определения простоты числа на основе теста Ферма (теста 1).

2.89 Построить процедуру определения простоты числа на основе модифицированного теста Ферма (теста 2).

2.90 Построить процедуру определения простоты числа на основе теста Миллера-Рабина.

Разложение числа на множители

Рассмотренные выше тесты проверки числа на простоту позволяют точно установить, что число является составным. Однако они не позволяют определить ни один из множителей этого числа. Задача нахождения множителей составного числа является вычислительно более сложной задачей, и современные компьютеры не могут в приемлемое время справиться с ее решением для 1024-битных чисел.

Для малых чисел существуют простые алгоритмы.

Метод «пробных делителей»

Этот наиболее очевидный алгоритм находит простые множители числа N, используя заданную последовательность делителей: d0=2, d1=3, d2 =5, … dk. Возрастающая последовательность dj заканчивается значением, не меньшим [√N], и должна содержать все простые числа в этом интервале, но может быть не только их. Можно использовать последовательность нечетных чисел в качестве значений dj для j >0, что позволяет не хранить элементы этой последовательности. Идея алгоритма проста, – последовательно делим число N на dj. Если N нацело делится на dj, то dj является множителем числа N, и оно заносится в массив множителей Factors. Затем процесс повторяется с новым значением N, равным N/dj, продолжая испытывать делители, начиная со значения dj. Процесс заканчивается, когда dj достигает значения [√N]. Массив Factors содержит все множители числа N.

Метод «простых делителей»

Это вариация предыдущего метода, когда в качестве делителей используется последовательность простых чисел. Чтобы применять этот алгоритм, необходимо предварительно получить таблицу простых чисел.

Метод Ферма

Этот алгоритм позволяет определить два множителя числа N. Его достоинство в том, что он не использует в основном цикле операции умножения и деления, ограничиваясь сложением и вычитанием. Естественной платой служит увеличение числа выполняемых операций. Идея алгоритма Ферма в следующем. Пусть N – неотрицательное число, а U и V – его множители, так что N=U*V, где U > = V, откуда автоматически следует, что U >= √N, V <= √N. Если N – простое число, то будем полагать, что U =N, а V =1. Рассмотрим два целых числа X и Y таких, что X >= Y. Подберем значения X и Y так, чтобы выполнялось соотношение $X^2 - Y^2 = N$. Заметьте, это всегда можно сделать. Тогда множителями N являются числа X+Y и X-Y. Они и будут выбраны в качестве множителей U и V. Если N – простое число, то X+Y будет равно N, а X-Y равно 1.

Пусть для X и Y заданы начальные значения: X=√N и Y=0, что эквивалентно предположению, что U= V=√N. Если окажется, что $X^2 - Y^2 = N$, то задача решена, если же $X^2 - Y^2 > N$, то увеличим Y на 1, в противном случае увеличим X на 1 и продолжим процесс. Процесс обязательно завершится, в худшем случае, когда X+Y станет равным значению N. Поскольку на каждом шаге значение X и Y изменяется на 1, то для ускорения вычислений можно избежать вычисления квадратов, используя известное рекуррентное соотношение: $(n+1)^2 = n^2 + 2*n + 1$. Алгоритм Ферма можно использовать для проверки того, что N – простое число. В этом случае число операций будет пропорционально (N-√N).

Приведу более подробную схему этого алгоритма:

```
int X, Y, x, y, r;
INIT: X = √N; Y = 0; x = 2*X+1; y =2*Y +1; r = X² – Y² – N;
while(r != 0)
{
  if(r < 0){r = r+x; x = x+2; X = X+1;}
  else {r = r-y; y= y+2; Y = Y+1;}
}
U = (x+y-2)/2; V = (x-y)/2;
```

Переменные X и Y в этой схеме введены лишь для облегчения доказательства ее корректности. В реальном алгоритме без них можно обойтись. Для доказательства корректности схемы заметим, что инвариантами цикла являются следующие три утверждения:

Inv1: $r = X^2 - Y^2 - N$; Inv2: $X+Y <= U <= N$; Inv3: $X-Y >= V >= 1$;

В результате инициализации все три утверждения становятся истинными. Нетрудно показать, что они остаются истинными на каждом шаге выполнения цикла.

Действительно, если выполняется условие (r < 0), то X увеличивается на 1, а r на величину x = 2*X +1, в результате чего $r = X^2 - Y^2 - N + 2*X + 1 = (X+1)^2 - Y^2 - N$ и инвариант Inv1 выполняется с новым значением X.

Для доказательства истинности инварианта Inv2 заметим, что из условия (r < 0) следует:

$(X+Y)(X-Y) < N = U*V$

С учетом истинности инварианта Inv3 отсюда следует, что (X+Y) < N. Поэтому при увеличении X на 1 сохраняется истинность инварианта Inv2. Аналогично доказывается истинность инварианта Inv3. Аналогично проводится разбор и для второй ветви оператора if.

Когда же цикл завершается при r = 0, то по определению X+Y и X-Y являются делителями числа N, так что алгоритм корректно решает задачу.

Для доказательства завершаемости цикла в качестве варианта цикла можно выбрать выражение N – (X+Y), которое остается неотрицательным в силу истинности инварианта Inv2 и уменьшается на 1 на каждом шаге цикла.

Эвристический алгоритм Полларда

Этот алгоритм с успехом применяется на практике, в том числе и для больших чисел. Он основан на эвристиках, которые хорошо работают в большинстве случаев, но не гарантируют ни успеха, ни малого времени работы. Приведу схему алгоритма Полларда:

```
int k,x,y,i,d, limit;
      i=1; x=rnd.Next(0, n-1); y=x; k=2; limit = n -(int) Math.Sqrt(n);
      while(true)
      {
        i++;
        x= x*x -1; x= x%n;
        d= NOD(n, Math.Abs(y-x));
        if((d!=1) &&(d!=n))break;
        if(i==k)
        {
          y=x; k*=2;
        }
        if(i > limit) break;
    } U = d; V = n/d;
```

В классическом варианте алгоритма формируется бесконечная последовательность значений x_k и y_k, начиная с некоторого случайного значения. Предложенная Поллардом стратегия рекуррентного вычисления значений x и y приводит к тому, что вероятность того, что разность x-y на некотором шаге станет множителем n, является весьма высокой. Я не буду приводить доводов в обоснование этого факта. Замечу лишь, что в предлагаемом варианте этого алгоритма строится конечная последовательность пробных значений, поскольку на практике крайне нежелательно использовать алгоритмы, допускающие зацикливание. Завершаемость алгоритма крайне важное свойство, которое следует гарантировать. По этим причинам в алгоритм добавлен барьер, ограничивающий цикл максимальным числом итераций, возможным в алгоритме Ферма. При желании величину барьера можно изменить.

Алгоритм Полларда на простых числах будет возвращать в качестве множителей значения N и 1, но в отличие от алгоритма Ферма он может, хотя и крайне редко, возвращать такие значения и для составного числа N.

Задачи

2.91 Напишите процедуру нахождения множителей числа, используя алгоритм простых делителей, хранящий таблицу простых чисел.

2.92 Напишите процедуру нахождения множителей числа, используя алгоритм пробных делителей, не требующий хранения таблицы.

2.93 Напишите процедуру нахождения минимального и максимального множителей числа, используя алгоритм Ферма.

2.94 Напишите процедуру, нахождения всех множителей числа N, используя алгоритм Ферма.

2.95 Напишите процедуру нахождения множителя числа, реализующую алгоритм Полларда.

2.96 Напишите процедуру, нахождения всех множителей числа N, используя алгоритм Полларда.

2.97 Постройте Windows-приложение, в котором оценивается эффективность метода Полларда. Оценку получите следующим образом. Генерируйте M раз случайное число N. Для каждой реализации вычислите по методу Полларда множитель d числа N и k – число шагов цикла, потребовавшихся для нахождения d. Вычислите значение p =k/N для каждой реализации и найдите среднее значение p по всем M реализациям.

2.98 Постройте Windows-приложение, в котором множители числа вычисляются по методу Ферма и по методу Полларда. Сравните эти два метода по числу требуемых шагов, по времени нахождения множителей.

Шифрование

С давних пор и до сегодняшних дней актуальной остается защита информации. Компьютерные технологии дали человечеству уникальные возможности по хранению информации и передачи ее из одной точки пространства в другую. Обратная сторона медали состоит в том, что трудно обеспечить секретность хранимых и передаваемых данных. Шифрование является одним из способов защиты данных и предназначено для решения трех основных задач:

- Конфиденциальность: защита данных пользователя или его идентификации от несанкционированного чтения;
- Целостность: защита данных от изменений;
- Аутентификация: гарантия того, что данные поступили от указанного в сообщении отправителя.

Применяемые схемы шифрования принято классифицировать следующим образом:

- Симметричное шифрование с закрытым ключом: применяется для обеспечения конфиденциальности данных. Для шифрования и дешифрования применяется один и тот же закрытый (секретный) ключ, который известен только получателю и отправителю сообщения.
- Ассиметричное шифрование с открытым ключом: применяется для обеспечения конфиденциальности данных. Для шифрования и дешифрования применяются разные ключи – открытый и закрытый, связанные определенным соотношением. Открытый ключ получателя, применяемый для шифрования, может быть известен не только получателю и отправителю сообщения. Для дешифрования применяется закрытый ключ получателя, известный только ему.
- Цифровая подпись: применяется в интересах аутентификации отправителя. В этом процессе используются и хэш-функции.
- Хеширование. Сжимает исходное сообщение, получая в результате статистически уникальный краткий образ сообщения, подобный отпечаткам пальцев.

Дадим теперь более точные определения основных понятий, используемых в криптографии. Пусть X и Y – это два множества, элементы которых будем называть данными. Под шифром будем понимать алгоритм или отображение:

$$y = F(k,x); \quad y \in Y; \quad x \in X; \quad k \in K;$$

Процесс получения элемента y по заданному элементу x называют шифрованием. Элементы x – это исходные данные, y – зашифрованные данные. При шифровании используется ключ k – элемент некоторого множества K, называемого множеством ключей. Всегда подразумевается возможность дешифрования – существование обратного отображения, позволяющего восстановить исходный элемент:

$$x = G(k,y) = G(k,F(k,x)); \quad y \in Y; \quad x \in X; \quad k \in K;$$

Когда говорят о шифровании, обычно рассматривают три стороны: отправитель сообщения, его получатель и противник. Отправитель шифрует исходные данные и посылает их получателю, которому это сообщение адресовано. Получатель дешифрует сообщение, имея ключ и применяя

известное ему отображение G(k,y). Противник, заинтересованный в расшифровке сообщения, может перехватить сообщение y. Его цель – расшифровать сообщение, не имея ключа и не зная алгоритма шифрования. Сложность шифра определяется тем, какие усилия могут понадобиться противнику для достижения его цели. В частном случае противнику может быть известен алгоритм дешифрования, но не известен ключ. В другом частном случае может быть известен ключ, но не известен алгоритм. Вспомните сказку о золотом ключике, когда мало было обладать ключом, но нужно было также знать, где находится заветная дверь, открываемая этим ключом.

Рассмотрим несколько примеров простых шифров.

Пример 1 (алгоритм сложения). Пусть X, Y, K – множества целых чисел, а алгоритм шифрования задается операцией сложения: $y = x+k$. Понятно, что существует обратное отображение: $x = y – k$.

Пример 2 (алгоритм сложения по модулю). Пусть X, Y, K – множества целых чисел в диапазоне [0, p-1], а алгоритм шифрования задается операцией сложения по модулю p: $y = (x+k) \pmod p$. Понятно, что существует обратное отображение: $x =(y – k) \pmod p$.

Шифрование применяется прежде всего к текстовым данным. В памяти компьютера тексты, как и любая другая информация, хранится в виде последовательности битов. При шифровании эта последовательность битов нарезается на блоки, как правило фиксированной длины, и каждый блок шифруется. Шифрование блока данных может проводиться независимо от остальных блоков, так что одинаковые блоки имеют одинаковые значения в зашифрованном сообщении. Чаще всего, при шифровании учитывается контекст и шифруется смесь блока с его соседями, например с предшествующим блоком. В этом случае задача раскрытия шифра значительно усложняется. Поскольку каждый блок битов можно рассматривать как число, то без потери общности данные можно рассматривать как числа, не зависимо от их содержательного смысла. В дальнейшем обсуждении предполагается, что сообщение задано числом.

При симметричном шифровании предполагается, что ключ шифра известен как отправителю сообщения, так и его получателю. В этом состоит серьезный недостаток подобных шифров, поскольку у противника появляется возможность набрать большой объем сообщений, зашифрованных одним ключом, который одновременно является и ключом, используемым при дешифровании, что облегчает задачу раскрытия шифра. Считается, что ключ шифра нужно менять как можно чаще, порождая его случайным, не прогнозируемым способом. Но тогда возникает проблема доставки ключа получателю сообщения. Поэтому интерес представляет асимметричное шифрование, где рассматриваются шифры, удовлетворяющие следующим дополнительным условиям:

- отправитель и получатель сообщения пользуются разными ключами;
- ключи отправителя и получателя связаны некоторым однозначным соотношением;
- соотношение, связывающее ключи, таково, что, зная один ключ, крайне сложно определить другой ключ.

Следствием этих условий является тот странный на первый взгляд факт, что отправитель сообщения, которое он сам зашифровал, не имеет возможности его расшифровать – применить алгоритм дешифрования, поскольку не знает ключа получателя сообщения. Посылая сообщение, отправитель может посылать и ключ, с помощью которого это сообщение было зашифровано, но не позволяющий расшифровать сообщение. Такие схемы шифрования возможны. Они получили название шифрования с открытым ключом. Недостатком асимметричного шифрования является его трудоемкость, требующая больших временных затрат при передаче длинных сообщений. Поэтому применяется комбинирование двух подходов. Для пересылки закрытого ключа используется асимметричная схема шифрования с открытым ключом, а для шифрования самого сообщения применяется симметричная схема с закрытым ключом, посланным получателю сообщения.

Алгоритм Диффи-Хеллмана

Схема шифрования может применяться для выработки общего ключа, когда отправитель и получатель вначале обмениваются зашифрованными сообщениями с открытым ключом. После

расшифровки сообщений каждый из них получает одно и то же сообщение, представляющее общий ключ, используемый далее в шифровании и расшифровке.

Пусть p – простое число и b < p. Оба эти числа известны отправителю и получателю сообщения и только им – это закрытые данные. Отправитель и получатель независимо друг от друга вырабатывают случайные числа e и d и на их основе создают сообщения:

$r = b^e \pmod p$;

$t = b^d \pmod p$;

Эти сообщения они и посылают друг другу. Получив их, каждый из них способен выработать общий ключ k следующим образом:

$k = t^e \pmod p = (b^d \pmod p)^e \pmod p = r^d \pmod p = (b^e \pmod p)^d \pmod p = b^{ed} \pmod p$

Заметьте, противнику, перехватившему сообщения r и t, получить ключ k крайне трудно, когда приходится работать с достаточно большими числами, содержащими несколько сотен знаков. Точно также получателю и отправителю сообщения трудно определить секретную часть ключа своего партнера – по числу e трудно вычислить число d, а по d трудно вычислить число e.

Число e можно рассматривать как ключ отправителя, известный только ему. Алгоритм шифрования, применяемый отправителем, задается функцией $f(k,x) = x^e \pmod p$. Получатель пользуется своим ключом d и функцией $g(k,y) = y^d \pmod p$. Отправитель и получатель по-разному шифруют одно и тоже сообщение b, а в результате расшифровки полученных ими сообщений они получают одно и то же сообщение, содержащее общий ключ.

Алгоритм RSA

Особенность этого алгоритма шифрования с открытым ключом, получившего название по имени его авторов – Райвеста, Шамира, Адлемена, состоит в том, что сообщение, адресованное получателю, шифруется не ключом отправителя, а открытым ключом самого получателя, который в этом случае является, доступным не только отправителю, но возможно и противнику. При широком использовании этой системы шифрования открытые ключи пользователей могут храниться в общедоступной библиотеке в Интернете, так что каждый имеет возможность посылать зашифрованные сообщения всем пользователям, выложившим свои открытые ключи.

Расшифровать сообщение может только получатель, используя для этого закрытую часть ключа. Так что определение алгоритма – шифрование с открытым ключом – неточно. Фактически ключ состоит из двух половинок – открытой части ключа и закрытой. Для шифрования используется открытая часть ключа, для дешифрования – закрытая.

Рассмотрим подробнее основы системы шифрования с открытым ключом. Будем называть получателя и отправителя сообщения Алисой и Бобом, как это принято при рассмотрении алгоритмов шифрования. Обозначим открытую и закрытую части ключа Алисы через P_A и S_A, а для Боба – P_B и S_B. Сообщения M, также как и зашифрованные сообщения будем рассматривать как элементы множества D. Функцию, шифрующую сообщения с открытым ключом Алисы, будем обозначать как P_A (M). Обратную функцию дешифрования с закрытым ключом будем обозначать как S_A (M). К шифрованию предъявляются следующие требования:

$M = S_A (P_A (M)) = P_A (S_A (M))$ для любого M из D.

Чтобы применить для шифрования RSA алгоритм, Алиса и Боб выполняют следующие действия:

- Выбирают случайным образом два простых числа p и q, каждое из которых может иметь длину в 512 битов или более.
- Вычисляют n = pq и k =(p-1)*(q-1).
- Выбирают нечетное число e, меньшее k и взаимно простое с k.
- Вычисляют d из уравнения $ed = 1 \pmod k$. Поскольку e взаимно просто с k, то решение этого уравнения существует. Алгоритм решения такого уравнения был рассмотрен.
- Публикуют пару P = (e,n) как свой открытый ключ.

- Сохраняют в глубокой тайне свой закрытый ключ S – пару (d, n).

Когда Боб хочет послать Алисе сообщение M, то он применяет ее открытый ключ и шифрует сообщение следующим образом:

$C = P_A(M) = M^e \pmod n$

Алиса, получив зашифрованное сообщение C, дешифрует его, используя известный только ей закрытый ключ:

$S_A(C) = C^d \pmod n$

В результате дешифрования Алиса получает исходное сообщение $M = S_A(C)$.

Докажем корректность алгоритма RSA, то есть докажем, что действительно так построенные функции шифрования и дешифрования являются взаимно обратными и обеспечивают восстановление исходного сообщения.

Прежде всего, заметим, что по свойствам модульной арифметики:

$S(P(M)) = P(S(M)) = M^{ed} \pmod n$

Так как e и d связаны соотношением: $e*d = 1 \pmod k = 1 \pmod{(p-1)*(q-1)}$, то справедливо равенство: $e*d = 1 + c*(p-1)*(q-1)$. Тогда:

$M^{ed} = M*(M^{p-1})^{c(q-1)}) \pmod p$

По теореме Ферма отсюда следует:

$M^{ed} = M*(M^{p-1})^{c(q-1)}) \pmod p = M*(1)^{c(q-1)}) \pmod p = M \pmod p$

Аналогично можно показать, что имеет место соотношение: $M^{ed} = M \pmod q$. А тогда, согласно еще одной теореме $M^{ed} = M \pmod{p*q} = M \pmod n$. Что и доказывает корректность алгоритма RSA.

На чем основывается безопасность шифрования алгоритмом RSA? Ведь противник знает открытый ключ (n, e), поэтому он может разложить n на множители, определить p и q, а затем определить закрытый ключ d, также как это делает автор ключа. Все зиждется на сложности разложения больших чисел на простые множители. Если эту задачу удается решить, то расшифровать сообщение не представляет труда. Но до сих пор задача разложения числа на множители при длине числа в 1024 бита и более не под силу современным компьютерам.

RSA и общий ключ

Шифрование и дешифрование длинных сообщений по алгоритму RSA, где используются длинные ключи, требует значительного времени. Поэтому иногда применяется гибридная схема шифрования, где алгоритм RSA используется для выработки общего закрытого ключа. Если Алиса хочет послать Бобу большое зашифрованное сообщение, то она может поступить следующим образом. Она выбирает случайным образом ключ k, которым и шифрует свое сообщение M, а затем открытым ключом Боба шифрует сам ключ k, и посылает Бобу пару $(F(k, M), P_B(k))$. Боб вначале определяет ключ $k = S_B(P_B(k))$, а затем дешифрует и само сообщение $M = F^{-1}(k, F(k, M))$. В отличие от алгоритма Диффи-Хеллмана, в этом случае не требуется обмена сообщениями для выработки общего ключа.

Цифровая подпись и RSA

Алгоритм RSA используется не только для обеспечения секретности посылаемого текста, но и для решения других задач, связанных с шифрованием. Прежде всего требуется обеспечить подлинность сообщения, включающую его целостность и аутентичность. Проблема аутентичности сообщения состоит в следующем. Если Алиса получила сообщение, зашифрованное ее открытым ключом и подписанное именем Боба, как ей убедиться, что сообщение послано действительно Бобом, а не противником, которому, также как и Бобу, известен открытый ключ Алисы? Для решения этой задачи существует цифровая подпись, которая не поддается подделке. Используя алгоритм RSA, Бобу достаточно зашифровать подпись своим закрытым ключом. Никто другой, кроме Боба, не может зашифровать его подпись подобным образом, поскольку закрытый ключ известен только ему одному. Алиса, расшифровав эту подпись открытым ключом Боба, убедится в том, что письмо действительно подписано Бобом.

Целостность текста и RSA

Проблема целостности, связанная с защищенной передачей секретных данных, состоит в том, чтобы гарантировать отсутствие изменений в переданном тексте – отсутствие добавлений, удалений или замен в переданном тексте, случайных или преднамеренных. Как Алисе убедиться в том, что текст, посланный Бобом, дошел до нее без искажений? Для этого можно использовать так называемые «отпечатки пальцев» посылаемого сообщения. По сообщению M строится его сжатый образ H(M) такой, что крайне трудно подобрать другое сообщение M1, чтобы образы сообщений H(M) и H(M1) совпадали. Функцию H(M) часто называют хэш-функцией.

Чтобы гарантировать целостность сообщения M, Боб посылает Алисе пару (P_A(M), S_B(H(M))) – зашифрованное сообщение и отпечаток сообщения, зашифрованный секретным ключом Боба. Алиса, получив эту пару, дешифрует сообщение M своим закрытым ключом и дешифрует отпечаток, используя открытый ключ Боба. Затем она строит отпечаток полученного ею сообщения, и если он совпадает с отпечатком, присланным Бобом, то можно быть уверенным в том, что полученное сообщение M не подвергалось правке. Отпечаток одновременно может играть роль цифровой подписи.

Хэш-функции широко применяются в программировании в разных ситуациях. Чаще всего они применяются, когда необходимо отображать некоторое подмножество множества большой мощности на множество меньшей мощности. Вот типичный пример. Представьте себе, что нужно хранить не более N ключей, а ключами могут быть произвольные слова русского языка. Тогда можно иметь для хранения ключей массив из N элементов и при появлении нового ключа записать его в элемент массива, индекс которого вычисляется с помощью хэш-функции, отображающей бесконечное множество слов в конечное множество – диапазон целых чисел [1, N]. При этом может оказаться, что вычисленный индекс уже использован, тогда в таблице ищется первый свободный элемент.

Рассмотрим пример некоторой простейшей хэш-функции. Таблица из N чисел разбивается на участки неравной длины, число участков равно 33 – по числу букв русского алфавита. Длина участка определяется частотой слов в русском языке, начинающихся на данную букву. Хэш-функция, анализируя первую букву ключа, возвращает в качестве результата начало соответствующего участка в таблице ключей.

Если применить эту хэш-функцию для построения отпечатка сообщения, то две таблицы, построенные для двух разных сообщений, совпадут только тогда, когда сообщения имеют одинаковое число слов и все соответствующие слова обоих сообщений начинаются с одной и той же буквы.

Другим примером может служить хэш-функция, которая вычисляет код слова, а затем возвращает в качестве результата остаток по модулю N.

Хорошо работающие на практике хэш-функции, как правило, являются «ноухау» фирм, их придумавших и использующих. Алиса и Боб могут применять известную только им хэш-функцию.

Цифровые подписи и сертификаты

Алгоритм RSA может использоваться и при выдаче сертификатов – заверенных цифровых подписей. Идея сертификатов основывается на возможности создания специальных сертификационных центров, пользующихся доверием большинства пользователей. Каждый, кто хочет получить цифровую подпись, обращается в сертификационный центр и получает от него подпись, зашифрованную закрытым ключом сертификационного центра. Поскольку открытый ключ центра известен всем пользователям, то Алиса, получив от Боба сообщение с сертифицированной подписью, может расшифровать подпись и убедиться, что сообщение было отправлено именно Бобом.

Задачи

2.99 Разработайте Windows-приложение, в котором Боб и Алиса обмениваются сообщениями, создавая общий ключ по алгоритму Диффи-Хеллмана.

2.100 Разработайте Windows-приложение, в котором Боб и Алиса посылают друг другу сообщения, используя RSA алгоритм шифрования с открытым ключом.

2.101 Разработайте Windows-приложение, в котором Штирлиц перехватывает и расшифровывает сообщения Бормана и Мюллера, зашифрованные RSA-алгоритмом. Штирлицу удалось узнать открытые ключи Бормана и Мюллера, из чего ему стало ясно, что число n, применяемое при шифровании, не превосходит 10^9, так что у него появилась возможность узнать закрытые ключи недальновидных противников.

2.102 Разработайте Windows-приложение, в котором Боб и Алиса посылают друг другу сообщения с цифровой подписью.

2.103 Разработайте Windows-приложение, в котором предлагаются три различных варианта построения отпечатка сообщения.

2.104 Разработайте Windows-приложение, в котором Боб и Алиса посылают друг другу сообщения и их отпечатки, чтобы гарантировать целостность и аутентичность переданного текста.

2.105 Разработайте Windows-приложение, в котором предлагаются три различные хэш-функции для хранения и поиска конечного множества из N ключей в таблице. Каждый ключ может быть произвольной строкой текста.

Шифрование в Framework .Net

Для обеспечения безопасности данных пользователя Framework .Net содержит разнообразный набор средств, гарантирующий конфиденциальность, целостность и аутентичность хранимых и передаваемых данных. Пространство имен Security, вложенное в пространство System, структурировано и содержит несколько специальных пространств имен, отвечающих за разные аспекты безопасности. Упомяну только два пространства – Permissions и Cryptography. Первое из них содержит классы, ведающие разрешениями на допуск к данным и проверку ролей пользователей. О классах из пространства Cryptography скажу чуть подробнее. Два абстрактных класса из этого пространства – SymmetricAlgorithm и ASymmetricAlgorithm являются родительскими классами, от которых наследуются классы, реализующие симметричные и ассиметричные алгоритмы шифрования. Если создается собственный класс шифрования, то разумно сделать его наследником этих абстрактных классов. Еще одним абстрактным классом, являющимся наследником класса SymmetricAlgorithm, является класс DES (Data Encryption Standart), потомки которого должны реализовать алгоритмы, удовлетворяющие принятому в США стандарту шифрования с закрытым ключом. Потомок этого класса – DesCryptoServiceProvider из библиотеки FCL задает реализацию этого стандарта.

Для асимметричного шифрования с открытым ключом аналогичную роль играют классы RSA и RSACryptoServiceProvider. Методы Encrypt и Decrypt последнего класса позволяют шифровать и дешифровать сообщения на основе алгоритма RSA.

Абстрактный класс DSA и его потомки позволяют задавать цифровую подпись. Абстрактный класс HashAlgorithm и его потомки позволяют строить «отпечатки» сообщений, используя различные хэш-функции. Важную роль в шифровании сообщений играет класс CryptoStream, позволяющий представлять сообщение в виде потоков битов и реализовать поблочный способ шифрования.

Шифрование используется и для внутренних целей Framework .Net, например для идентификации сборок на основе цифровых подписей.

Задачи

2.106 Напишите Windows-приложение, в котором используется класс DesCryptoServiceProvider для шифрования и дешифрования файлов.

2.107 Разработайте Windows-приложение, в котором Боб и Алиса посылают друг другу сообщения, используя RSA алгоритм шифрования из библиотеки FCL.

2.108 Разработайте Windows-приложение, в котором Штирлиц перехватывает и пытается расшифровать сообщения Бормана и Мюллера, зашифрованные RSA-алгоритмом из библиотеки FCL.

2.109 Разработайте Windows-приложение, в котором Боб и Алиса посылают друг другу сообщения с цифровой подписью, используя классы из библиотеки FCl.

2.110 Разработайте Windows-приложение, в котором для построения отпечатка сообщения используются классы библиотеки FCL.

2.111 Разработайте Windows-приложение, в котором Боб и Алиса используют классы из библиотеки FCL для обмена сообщениями, гарантирующие целостность и аутентичность переданного текста.

Числа Фибоначчи

Разговор о числах можно продолжать до бесконечности. Закончим эту главу рассмотрением чисел, странным образом появляющихся в самых разных областях человеческой жизни.

Числа Фибоначчи задают бесконечную последовательность F_0, F_1, F_2, … F_n, …, которую можно определить следующими рекуррентными соотношениями:

$$F_0 = 0; F_1 = 1;$$
$$F_k = F_{k-1} + F_{k-2}; \quad для\ k > 1 \qquad (*)$$

Итальянский математик Фибоначчи, впервые предложивший эту последовательность еще в 1202 году, и в честь которого они впоследствии были названы, привел при их описании задачу «о кроликах», которую можно рассматривать, как простейшую модель роста популяции. У Кеплера эти числа появились при изучении распределения положения листьев на стебле. В программировании эти числа появляются при изучении сложности алгоритма Эвклида, при рассмотрении алгоритмов сортировки файлов на внешних носителях и во многих других ситуациях. Приведу некоторые соотношения, характеризующие свойства чисел Фибоначчи, и соотношения, позволяющие применять различные алгоритмы их вычисления.

Справедливо соотношение:

$$F_{n+1} * F_{n-1} - F_n^{\,2} = (-1)^n$$

Докажем его по индукции. Нетрудно проверить, что оно выполняется при n=1, 2. Предположим, что оно выполняется для всех k <=n. Тогда:

$$F_{n+2} * F_n - F_{n+1}^{\,2} = (F_{n+1} + F_n) * F_n - F_{n+1}^{\,2} =$$
$$F_{n+1}(F_n - F_{n+1}) + F_n^{\,2} = -F_{n+1} * F_{n-1} + F_n^{\,2} = (-1)^{n+1}$$

Из этого соотношения в частности следует, что два соседних числа Фибоначчи взаимно просты.

Также по индукции легко доказывается важное соотношение:

$$F_{n+m} = F_m * F_{n+1} + F_{m-1} * F_n \qquad (**)$$

Соотношение (**) позволяет задать более эффективные алгоритмы вычисления чисел Фибоначчи по сравнению с прямым использованием рекуррентных соотношений (*), данных при их определении. Если уже построена и хранится в памяти таблица первых n чисел Фибоначчи, то, используя соотношение (**), можно вычислить за константное время (два умножения и одно сложение) значение любого числа Фибоначчи в интервале от n+1 до 2n.

Если не хранить таблицу, то все равно можно ускорить вычисление числа F_{2n}, ограничившись вычислением F_n из соотношения (*), а затем применить соотношение (**). Для больших n это позволит почти вдвое сократить время вычислений.

Удивительным образом числа Фибоначчи появляются и в искусстве, – они тесно связаны с числом Фидия – Ф, характеризующим так называемое «золотое сечение» или «божественную пропорцию». Об этой связи много говорится в модном романе Дэна Брауна «Код Да Винчи». Рассмотрим и мы ее более подробно, поскольку она дает возможность построить еще один эффективный алгоритм вычисления чисел Фибоначчи. Начнем с определения числа Ф, названного в честь греческого скульптора, для работ которого характерно применение золотого сечения.

Рассмотрим задачу деления отрезка на две части. Большую часть отрезка обозначим через x, меньшую – y. Если большая часть так относится к меньшей, как целое к большей части, то говорят, что имеет место «золотое сечение» отрезка. В этом случае отношение x и y составляет божественную пропорцию, дающую эстетически идеальное соотношение размеров. Отношение x/y называют числом Фидия и обозначают буквой Ф. Имеют место следующие соотношения:

$$x + y = 1; \quad \frac{x}{y} = \frac{1}{x}; \quad x^2 = 1 - x; \quad x = \frac{-1 + \sqrt{5}}{2};$$

$$\Phi = \frac{x}{y}; \quad \Phi^2 = 1 + \Phi; \quad \Phi = \frac{1 + \sqrt{5}}{2};$$

Покажем теперь, как связаны числа Фибоначчи с числом Ф. Докажем, что справедливо следующее соотношение:

$$F_n = \frac{\Phi^n - (1 - \Phi)^n}{\sqrt{5}} \qquad (***)$$

Доказательство проведем методом математической индукции. Нетрудно проверить, что соотношение выполняется для F_1 и F_2. Предположим теперь, что оно выполняется для всех 1<=k <n. Покажем его справедливость и для F_n:

$$F_n = F_{n-1} + F_{n-2} = \frac{\Phi^{n-1} - (1 - \Phi)^{n-1}}{\sqrt{5}} + \frac{\Phi^{n-2} - (1 - \Phi)^{n-2}}{\sqrt{5}} =$$

$$\frac{\Phi^{n-2}(1 + \Phi) - (1 - \Phi)^{n-2}(2 - \Phi)}{\sqrt{5}} = \frac{\Phi^n - (1 - \Phi)^n}{\sqrt{5}}$$

Последнее соотношение получено с учетом того, что $(1+\Phi) = \Phi^2$ и $(2-\Phi) = (1-\Phi)^2$.

Соотношение (***) можно использовать для вычисления значения числа Фибоначчи за время T = O(log n), применяя эффективный алгоритм возведения в степень. Учитывая, что $(1-\Phi)$ меньше 1, то для больших значений n можно не вычислять $(1-\Phi)^n$, поскольку его вклад не будет влиять на конечное значение, округляемое при переходе от вещественного значения правой части к целочисленному значению F_n.

Конечно, соотношение (***) будет давать лучшие по времени результаты только для больших n, поскольку оно требует выполнения более дорогих операций умножения, в то время как соотношение (*) ограничивается лишь операциями сложения.

Заметьте, с ростом n отношение F_{n+1} к F_n стремится к «божественной пропорции» – числу Ф.

Наряду с простыми числами Фибоначчи определяют числа Фибоначчи более высоких порядков. Так числа Фибоначчи второго порядка $^{(2)}F_n$ определяются следующим образом:

$$^{(2)}F_0 = 0; \quad ^{(2)}F_1 = 1;$$

$$^{(2)}F_{n+2} = {}^{(2)}F_{n+1} + {}^{(2)}F_n + F_n; \qquad (****)$$

Задачи

2.112 Задача Фибоначчи о кроликах. Подсчитайте число пар кроликов, которое Вы можете получить за N лет. В начальный период Вам подарена новорожденная пара кроликов. Каждая пара кроликов начинает давать приплод через месяц после рождения, и каждый месяц приносит по одной паре кроликов. Кролики бессмертны.

2.113 Напишите алгоритм вычисления чисел Фибоначчи, используя соотношение (*). Подсчитайте требуемое время T(n) для вычисления F_n, постройте график и оцените значение константы k в зависимости T(n) = k*n.

2.114 Напишите алгоритм вычисления чисел Фибоначчи в диапазоне [n, 2n], используя соотношение (**) и построенную таблицу первых n чисел Фибоначчи.

2.115 Напишите алгоритм вычисления чисел Фибоначчи, используя соотношение (**). Подсчитайте требуемое время T(n) для вычисления F_n, постройте график и оцените значение константы k в зависимости T(n) = k*n.

2.116 Напишите алгоритм вычисления чисел Фибоначчи, используя соотношение (***). Подсчитайте требуемое время T(n) для вычисления F_n, постройте график.

2.117 Напишите алгоритм вычисления чисел Фибоначчи, используя соотношение (***), в котором опущено вычисление $(1-Ф)^n$. Оцените, при каком значении n этот алгоритм начинает давать правильные результаты.

2.118 Напишите алгоритм вычисления чисел Фибоначчи второго порядка, используя соотношение (****). Подсчитайте требуемое время T(n) для вычисления $^{(2)}F_n$, постройте график и оцените значение константы k в зависимости T(n) = k*n.

Проекты

2.119 Постройте класс Rational для работы с рациональными числами. Определите в этом классе методы (Plus, Minus, Mult, Div, Pow) и операции (+, -, *, /, ^) над рациональными числами, задающими сложение, вычитание, умножение, деление и возведение в целую степень. Для возведения в степень используйте эффективный алгоритм, требующий логарифмического времени. Определите в этом классе методы (Eq, Neq, Gr, Less) и операции отношения (==, !=, >, <). Задайте в классе рациональные константы – 0 и 1. Переопределите в классе метод ToString, задающий строку, представляющую рациональное число в удобном для восприятия виде.

2.120 Постройте Windows-приложение: Калькулятор для работы с рациональными числами.

2.121 Постройте класс Complex для работы с комплексными числами. Определите в этом классе методы (Plus, Minus, Mult, Div, Pow) и операции (+, -, *, /, ^) над комплексными числами, задающими сложение, вычитание, умножение, деление и возведение в целую степень. Для возведения в степень используйте эффективный алгоритм, требующий логарифмического времени. Определите в этом классе методы (Eq, Neq, Gr, Less) и операции отношения (==, !=, >, <).Задайте в классе комплексные константы – 0, 1 и i. Переопределите в классе метод ToString, задающий строку, представляющую комплексное число в удобном для восприятия виде.

2.122 Постройте Windows-приложение: Калькулятор для работы с комплексными числами.

2.123 Постройте класс Modular для работы с целыми числами в модульной арифметике. Определите в этом классе методы (Plus, Minus, Mult, Div, Pow) и операции (=, +, -, *, /, ^), задающие сложение, вычитание, умножение, деление и возведение в целую степень по заданному модулю m. Для возведения в степень используйте эффективный алгоритм, требующий логарифмического времени. Определите в классе метод, дающий решение сравнения a = x (mod m), где 0 =< x < m. Определите в классе метод, определяющий все решения уравнения ax = b (mod m), если таковые решения существуют. Переопределите в классе метод ToString, задающий строку, представляющую число как результат сравнения a = b (mod m), где 0 =< b < m.

2.124 Постройте Windows-приложение: Калькулятор для работы с модульной арифметикой.

2.125 Постройте класс Number, поле которого содержит целое число a. Методы класса позволяют определить, является ли число простым, а для составного числа найти разложение на множители. Методы класса для заданного целого числа b позволяют найти наибольший общий делитель, наименьшее общее кратное a и b, позволяют определить, являются ли эти числа взаимно простыми.

2.126 Постройте Windows-приложение для работы с классом Number.

2.127 Постройте класс Crypto, предоставляющий возможности шифрования с открытым и закрытым ключом, создание цифровых подписей и отпечатков сообщений.

Примеры

Шифрование файлов на основе использования алгоритмов DES библиотеки FCL

Построим Windows-приложение, позволяющее шифровать и дешифровать произвольные файлы. Для шифрования и дешифрования будем использовать алгоритмы шифрования с закрытым ключом. Используем для этих целей реализацию алгоритмов, соответствующую стандарту DES, предлагаемую соответствующими классами библиотеки FCL.

Первым делом я создал новый проект с именем CryptoTest и спроектировал форму, показанную на рис. 2_1, в которой есть три текстовых окна и три командные кнопки. В текстовых окнах задаются имя исходного файла, подвергающегося шифрованию, имя файла после его шифрования и имя файла, полученного при дешифровании. Обработчики событий Click командных кнопок позволяют создать объект, выполняющий шифрование, и вызвать его методы для шифрования и дешифрования файла.

Рис. 2_1 Шифрование и дешифрование файлов

В класс Form1, создаваемый по умолчанию в Windows-проектах, я добавил поле с описанием объекта класса CryptoTesting:

```
CryptoTesting ct ;
```

Класс CryptoTesting нам предстоит еще создать. Методы этого класса должны позволить нам шифровать и дешифровать файлы. Обработчики событий Click командных кнопок нашей формы просты и естественны. Вот как выглядит обработчик, создающий объект ct класса CryptoTesting:

```
private void button3_Click(object sender, System.EventArgs e)
    {
      string name_s, name_c, name_d;
      name_s= textBox1.Text;
      name_c= textBox2.Text;
      name_d= textBox3.Text;
      ct = new CryptoTesting(name_s,name_c,name_d);
    }
```

Как видите, обработчик вызывает конструктор класса, передавая ему в качестве аргументов имена соответствующих файлов, с которыми придется работать методам класса.

Еще более просто устроены обработчики для двух оставшихся командных кнопок – они просто вызывают соответствующие методы класса:

```
private void button1_Click(object sender, System.EventArgs e)
    {

      ct.CipherFile();
    }

    private void button2_Click(object sender, System.EventArgs e)
    {
      ct.DecipherFile();
    }
```

Закончив с интерфейсом, перейдем к основному делу – проектированию класса CryptoTesting. Этот класс будет использовать классы библиотеки FCL, необходимые для шифрования и работы с файлами. При объявлении класса следует задать ссылки на соответствующие пространства имен:

```
using System.Security.Cryptography;
using System.IO;
```

А теперь займемся самим классом и начнем с проектирования его полей. Вот они:

```
//поля
    DESCryptoServiceProvider dcp;
    string sn, cn, dn;
    const int blocksize = 100;
```

Основным является конечно же объект dcp библиотечного класса DESCryptoServiceProvider. Класс этот является наследником абстрактного класса DES и задает реализацию алгоритма шифрования с закрытым ключом, отвечающего стандартам, принятым в США. Строковые поля задают имена файлов, используемых в процессе шифрования. Константа blocksize задает размер блоков, создаваемых в процессе работы с шифруемым и дешифруемым файлами.

Определим теперь конструктор класса:

```
public CryptoTesting(string n1, string n2, string n3)
    {
      dcp  = new DESCryptoServiceProvider();
      dcp.GenerateKey();
      dcp.GenerateIV();
      sn= n1; cn= n2; dn=n3;
    }
```

В конструкторе класса создается объект dcp, а затем вызываются два его метода, позволяющие задать важнейшие свойства объекта – Key и IV – ключ и вектор инициализации. Оба эти свойства представляют массивы байтов, размер которых должен отвечать требованиям стандарта DES. По умолчанию размер ключа равен 64 битам. Конечно можно самому сгенерировать значения этих свойств, но вполне разумно воспользоваться, как это сделано в нашем конструкторе, методами, предлагаемыми в классе – GenerateKey и GenerateIV, генерирующими случайные значения ключа и инициализирующего вектора.

Рассмотрим теперь метод, выполняющий шифрование файла:

```
    public void CipherFile()
    {
      //Шифрование файла. Исходный и шифруемый файлы
      //рассматриваются как бинарные файлы - потоки байтов
      FileStream fin = new FileStream (sn, FileMode.Open,
        FileAccess.Read);
      FileStream fout = new FileStream(cn, FileMode.OpenOrCreate,
        FileAccess.Write);
      fout.SetLength(0);
      //bin - промежуточный массив для хранения очередного блока байтов
      //len, rdlen, totlen -
      // длина текущего блока, общее число прочитанных байтов, длина файла
      byte[] bin = new byte[blocksize];
      int len;
      long rdlen = 0;
      long totlen = fin.Length;
      //encStream - объект, описывающий шифруемый поток
      CryptoStream encStream = new CryptoStream
```

```
            (fout, dcp.CreateEncryptor(),CryptoStreamMode.Write);
        //Исходный файл читается блок за блоком,
        //каждый из них шифруется и записывается в выходной поток
        while(rdlen < totlen)
        {
          len = fin.Read(bin, 0, blocksize);
          encStream.Write(bin, 0, len);
          rdlen = rdlen + len;
        }
        encStream.Close();
        fout.Close();
        fin.Close();
    }
```

Этот метод нуждается в пояснениях. В шифровании и дешифровании файлов важную роль играет класс CryptoStream. При создании объекта этого класса – encStream конструктору переданы три аргумента. Первый из них определяет текущий поток – файл, в который поблочно будут записываться шифруемые данные. Метод, который следует применить при шифровании – dcp.CreateEncryptor, передается в качестве второго аргумента. Третий аргумент задает статус текущего потока.

Запись и шифрование в текущий поток выполняется в цикле по числу записываемых блоков. Метод Write объекта encStream берет блок из буфера, комбинирует его с ранее записанным в файл блоком, шифрует его закрытым ключом, используя метод шифрования, переданный в момент создания объекта, а затем шифрованный блок записывает в текущий поток. Для шифрования первого блока используется инициализирующий вектор IV объекта dcp.

Отсюда следует достаточно простая схема шифрования любого файла. Вначале нужно создать два объекта класса FileStream, они названы fin и fout. Первый из них задает исходный файл, рассматриваемый как поток байтов независимо от его реальной природы, а второй определяет создаваемый зашифрованный файл. После этого создается уже рассмотренный нами объект класса CryptoStream. Затем организуется цикл, в котором исходный файл нарезается на блоки, помещаемые в буфер, роль которого играет массив bin. Метод Write, как было сказано выше, этот блок шифрует и записывает в выходной файл.

Схема дешифрования абсолютно симметрична. Единственное отличие состоит в методе шифрования, передаваемому конструктору при создании объекта encStream. Если при шифровании ему передавался метод dcp.CreateEncryptor, то при дешифровании имя метода меняется на dcp.CreateDecryptor. Приведу уже без всяких комментариев текст метода DecipherFile:

```
    public void DecipherFile()
    {
        //Дешифрование
        FileStream fin = new FileStream (cn, FileMode.Open,
          FileAccess.Read);
        FileStream fout = new FileStream(dn, FileMode.OpenOrCreate,
          FileAccess.Write);
        byte[] bin = new byte[blocksize];
        int len;
        long rdlen = 0;
        long totlen = fin.Length;
```

```
        CryptoStream encStream = new CryptoStream
             (fout, dcp.CreateDecryptor(), CryptoStreamMode.Write);
        while(rdlen < totlen)
        {
          len = fin.Read(bin, 0, blocksize);
          encStream.Write(bin, 0, len);
          rdlen = rdlen + len;
        }
        encStream.Close();
        fout.Close();
        fin.Close();
      }
```

Рассмотренная здесь схема шифрования и дешифрования файла вполне может использоваться Алисой и Бобом при передаче сообщений. Сообщение Алисы представляется в виде файла, она его шифрует рассмотренным здесь способом и посылает Бобу зашифрованный файл. Естественно, чтобы Боб мог расшифровать файл, Алиса должна передать ему закрытый ключ и вектор IV. Для передачи этих данных Алиса использует схему шифрования с открытым ключом.

Пример проектирования: класс, DLL, и многое другое

Рассмотрим сейчас достаточно большой пример, в котором создадим решение, содержащее три проекта, один из которых будет DLL, а два других – консольным и Windows-приложениями, использующих эту DLL. По выбору пользователя на выполнение может запускаться либо консольное, либо Windows-приложение.

Наша DLL будет содержать класс, на примере проектирования которого постараемся продемонстрировать многие детали – различные типы конструкторов класса, закрытие свойств, создание собственных констант в классе, перегрузку операций, переопределение методов родительского класса, выбрасывание исключений.

Чтобы не усложнять задачу, выберем достаточно понятную проблемную область. Давайте создадим полноценный класс для работы с рациональными числами. Консольное и Windows-приложение в нашем решении будут представлять калькуляторы для работы с рациональными числами.

Первоначальное построение решения (solution)

Новое решение создадим в среде Visual Studio 2005 и начнем как обычно с выбора типа проекта. Первым проектом, помещаемым в решение, пусть будет DLL, а посему в качестве типа проекта выберем библиотеку классов – Class Library. Дадим решению имя RationalsCalc, а проекту – RationalsLibrary.

В созданное по умолчанию пространство имен Classes добавим класс Rational. Прежде чем определять детали класса в тег summary, предшествующий классу, добавим неформальную его спецификацию:

```
/// <summary>
  /// Класс Rational.
  /// определяет новый тип данных - рациональные числа и основные
  /// операции над ними - сложение, умножение, вычитание и деление.
  /// Рациональное число задается парой целых чисел (m,n) и изображается,
  /// обычно в виде дроби m/n. Число m называется числителем,
  /// n - знаменателем.
```

```
/// Знаменатель не должен быть равен нулю!
/// Для каждого рационального числа существует
/// множество его представлений, например 1/2, 2/4, 3/6, 6/12 -
/// задают одно и тоже рациональное число. Среди всех представлений
/// можно выделить то, в котором числитель и знаменатель взаимно
/// несократимы. Такой представитель будет храниться в полях класса.
/// Операции над рациональными числами определяются естественным
/// для математики образом.
/// </summary>
public class Rational
{
     // Описание тела класса Rational
}//Rational
```

Поля класса Rational

Два целых числа m и n представляют рациональное число. Они и становятся полями класса. Совершенно естественно сделать эти поля закрытыми (private), поскольку рациональное число рассматривается как нечто неделимое и клиент класса не должен иметь доступа к составляющим рационального числа. Поэтому для этих полей не будут даже определяться процедуры-свойства – они будут полностью закрыты. Вот объявление полей класса:

```
//Поля класса. Числитель и знаменатель рационального числа.
    long m,n;
```

Конструкторы класса Rational

Инициализация полей конструктором по умолчанию никак не может нас устраивать, поскольку нулевой знаменатель это нонсенс. Поэтому определим конструктор с аргументами, которому будут передаваться два целых – числитель и знаменатель создаваемого числа. Кажется, что это единственный разумный конструктор, который может понадобиться нашему классу. Однако чуть позже мы добавим в класс закрытый конструктор и статический конструктор, позволяющий создать константы нашего класса. Вот определение конструктора:

```
/// <summary>
    /// Конструктор класса. Создает рациональное число
    /// m/n эквивалентное a/b, но со взаимно несократимыми
    /// числителем и знаменателем. Если b=0, то выбрасывается
    /// исключение специально спроектированного класса
        /// RationalException
    /// </summary>
    /// <param name="a">числитель</param>
    /// <param name="b">знаменатель</param>
    public Rational(long a, long b)
    {
        const string mes = "Некорректная попытка создать рациональное "
        + " число, знаменатель которого равен 0";
        if (b == 0) throw (new RationalException(mes));
        else
```

```
            if (a == 0)
            {
               m = 0; n = 1;
            }
           else
            {
                 //приведение знака
                if (b < 0) { b = -b; a = -a; }
                //приведение к несократимой дроби
                   long d = nod(a, b);
                   m = a / d; n = b / d;
            }
        }//Конструктор
```

Как видите, конструктор класса может быть довольно сложным. В нем, как в нашем случае, может проверяться корректность задаваемых аргументов. Задание нулевого знаменателя для рационального числа некорректно. Если такая ситуация возникает, то в конструкторе выбрасывается исключение.

При проектировании класса следует проектировать и специальные классы исключений, позволяющие классифицировать однозначным образом возникающие ситуации. Такой класс RationalException и был создан. Опишем его чуть позже.

За исключением случая, когда знаменатель равен нулю, все остальные пары целых чисел считаются корректными. Однако корректность пары a и b, заданной клиентом класса, еще не означает, что именно эта пара будет представлять рациональное число. В большинстве случаев заданная пара чисел приводится к эквивалентному рациональному числу, в котором знаменатель всегда положителен и не имеет общих делителей с числителем. По ходу дела вызывается закрытый метод класса, вычисляющий значение НОД(a,b) – наибольшего общего делителя чисел a и b.

Алгоритм вычисления НОД(a,b) подробно рассматривался в этой главе, так что приводимая здесь реализация не требует дополнительных пояснений:

```
    /// <summary>
        /// Закрытый метод класса.
        /// Возвращает наибольший общий делитель чисел a,b
        /// </summary>
        /// <param name="a">первое число</param>
        /// <param name="b">второе число, положительное</param>
        /// <returns>НОД(a,b)</returns>
        long nod(long m, long n)
        {
          long p=0;
          m = Math.Abs(m); n = Math.Abs(n);
          if(n>m){p=m; m=n; n=p;}
          do
          {
            p = m%n; m=n; n=p;
```

```
        }while (n!=0);
        return(m);
    }//nod
```

Класс RationalException

Все классы исключений, как стандартные, так и создаваемые пользователем, являются наследниками класса Exception. В большинстве случаев можно ограничиться наследованием свойств и методов родительского класса, не определяя в потомке ничего нового. Однако придется добавить в класс конструкторы, поскольку они не наследуются. Тела конструкторов можно сделать пустыми, полагаясь на вызов конструкторов базового класса. Именно так и будет устроен наш класс. Вот его текст:

```
/// <summary>
    /// Исключение этого класса выбрасывается при попытке
    ///  создания рационального числа с нулевым знаменателем
    /// </summary>
    class RationalException : Exception
    {

        public RationalException()
        {
        }
        public RationalException(string message)
            : base(message)
        {
        }
        public RationalException(string message, Exception e)
            : base(message, e)
        {
        }
    }
```

Более подробно с классом Exception и другими классами исключений можно познакомиться в лекции 23 основного для нас учебника [1].

Нужно ли обрабатывать исключение в классе Rational?

В конструкторе класса Rational в нужный момент будет вызываться исключение:

```
        if (b == 0) throw (new RationalException(mes));
```

Заметьте, в момент выбрасывания исключения создается объект класса RationalException, вызывается один из спроектированных конструкторов, ему передается сообщение, однозначно идентифицирующее ситуацию. Текст переданного сообщения становится значением свойства Message объекта, характеризующего исключение.

Хочу обратить внимание на технологию работы. При обнаружении ситуации не делается попытка уведомить клиента о возникшей ошибке и проложить работу. Нет, в подобной ситуации следует выбрасывать исключение, что может привести к завершению работы приложения, если это исключение не будет должным образом обработано. Но это правильная технология работы, – продолжать работать с некорректными данными не следует. Важно лишь, в момент выбрасывания исключения тщательно квалифицировать причину его появления.

Почему выбрасываемое исключение не обрабатывается в самом классе Rational? Ведь вполне можно было бы организовать вычисления в try-catch блоках, перехватить исключение и обработать его – выдать сообщение, скорректировать некоторым образом введенную пару чисел, организовать диалог с пользователем. Всего этого делать не следует. Класс Rational может понять, что ситуация возникла, но он не может понимать, как ее исправить. Это может сделать только клиент, по вине которого возникла такая ситуация. Именно он должен обрабатывать исключительную ситуацию, именно он понимает, как с ней справиться. Он нарушил спецификации класса, он и должен исправлять ситуацию. В обязанности класса входит уведомление клиента о нарушении им спецификаций класса с предоставлением клиенту всей необходимой информации. Это уведомление делается в жесткой форме – форме исключительной ситуации, что приведет к прерыванию работы, если клиент не примет надлежащих мер. Позже мы увидим, как клиент класса обрабатывает это исключение и исправляет возникшую ситуацию.

Конечно, возможно, что клиент не нарушил спецификаций и исключительная ситуация возникла по вине методов самого класса, некорректно обрабатывающих корректные данные. В таких случаях исправление ситуации, перехват возникающих исключений входит в обязанность самого класса. Ситуация с нулевым знаменателем к таким случаям не относится.

Методы класса Rational

После некоторого отступления вернемся к проектированию класса Rational. Заметьте, в основе каждого класса лежит абстрактный тип данных, при определении которого нас интересует не столько то, как представлены данные, а то, что с ними можно делать, – какие операции разрешены над данными.

Если поля класса почти всегда закрываются, чтобы скрыть от пользователя представление данных класса, то среди методов класса всегда есть открытые методы – это те сервисы (службы), которые класс предоставляет своим клиентам и наследникам. Но не все методы открываются. Часть методов класса может быть закрытой, скрывая от клиентов детали реализации, необходимые в интересах внутреннего использования. Заметьте, скрытие представления данных, скрытие методов и скрытие реализации делается не по соображениям утаивания того, как реализована система. Чаще всего, ничто не мешает клиентам ознакомиться с полным текстом класса. Скрытие делается в интересах самих клиентов. При сопровождении программной системы изменения в ней неизбежны. Клиенты не почувствуют на себе негативные последствия изменений, если они делаются в закрытой части класса. Чем больше закрытая часть класса, тем меньше влияние изменений на клиентов класса. Знание спецификаций класса и его интерфейса – это, что необходимо клиенту для успешной работы с классом.

Примером закрытого для клиентов метода является уже приведенный метод класса nod. Клиентам, которым класс Rational интересен тем, что позволяет работать с рациональными числами, нет дела до того, что методам класса Rational понадобилось вычислять наибольший общий делитель, и им должно быть неважно, какой алгоритм при этом использовался.

Арифметические операции класса Rational

Поскольку рациональные числа это прежде всего числа, то давайте определим для них основные арифметические операции: сложение и вычитание, умножение и деление, возведение в целую степень. Реализуются все эти операции как операции над дробями, поэтому алгоритмы в комментариях не нуждаются. Привычно рассматривать эти операции как бинарные и писать x+y, где x и y рациональные числа, выступающие как равноправные партнеры. Для объектного стиля характерна другая форма записи. Здесь пишут x.Plus(y). Объект x выступает в роли хозяина, он вызывает метод класса Plus, у которого один аргумент – слагаемое, добавляемое к текущему объекту. Понятно, что у объекта y те же права, что и у объекта x, так что аналогичный результат будет получен при вызове y.Plus(x).

Вот определение методов, реализующих четыре основные арифметические операции над рациональными числами:

```
public Rational Plus(Rational a)
    {
       long u,v;
```

```
        u = m*a.n +n*a.m; v= n*a.n;
        return( new Rational(u, v));
    }//Plus
public Rational Minus(Rational a)
    {
        long u,v;
        u = m*a.n - n*a.m; v= n*a.n;
        return( new Rational(u, v));
    }//Minus
public Rational Mult(Rational a)
    {
        long u,v;
        u = m*a.m; v= n*a.n;
        return( new Rational(u, v));
    }//Mult
public Rational Divide(Rational a)
    {
        long u,v;
        u = m*a.n; v= n*a.m;
        return( new Rational(u, v));
    }//Divide
```

Несколько слов следует сказать о возведении в целую степень. Проще всего реализовать эту операцию простым умножением. Действительно, учитывая, что числитель и знаменатель – это целые числа, то показатель степени n не может быть слишком большим числом, поскольку результат может просто выйти за пределы разрядной сетки, отводимой для данных типа long. Тем не менее в приводимой здесь реализации используется алгоритм, эффективный при больших n, требующий лишь log(n) операций умножения. Этот алгоритм подробно рассматривался в этой главе. Заметьте, реализация построена на использовании закрытого метода, так что ничто не мешает сменить реализацию при необходимости, это никак не отразится на программе клиента, вызывающего метод возведения в степень.

```
public Rational Pow(int k)
 {
     long u, v;
     u = P(m,k); v = P(n,k);

     return (new Rational(u, v));
 }//Pow
 //Возведение в степень. Алгоритм эффективен при больших k,
 //поскольку требует порядка log(k) умножений
 long P(long a, int k)
 {
     //вычисляет a^k
     long z = 1;
```

```
            while (k != 0)
            {
                while (k % 2 == 0)
                {
                    k /= 2; a *= a;
                }
                //k- нечетно
                {
                    z *= a; k--;
                }
            }
            return z;
        }
```

Перегрузка арифметических операций

Хотя методы и хороши, но привычнее использовать знаки арифметических операций. Особенно они удобны при построении сложных арифметических выражений. Поэтому наряду с методами определим и соответствующие операторы (знаки операций). Знаки операций являются перегруженными. Это означает, что операции + соответствует множество различных реализаций, так что запись x+y выполняется по-разному в зависимости от типов аргументов. К уже имеющемуся множеству реализаций добавим и реализации, позволяющие выполнять операции над рациональными числами.

Для определения знаков операций в языке C# используется специальное ключевое слово operator. Операция определяется как статический метод с двумя аргументами. Вот как это делается:

```
    public static Rational operator +(Rational r1, Rational r2)
        {
          return (r1.Plus(r2));
        }
    public static Rational operator -(Rational r1, Rational r2)
        {
          return (r1.Minus(r2));
        }
    public static Rational operator *(Rational r1, Rational r2)
        {
          return (r1.Mult(r2));
        }
    public static Rational operator /(Rational r1, Rational r2)
        {
          return (r1.Divide(r2));
        }
    public static Rational operator ^(Rational r1, int n)
        {
          return (r1.Pow(n));
```

```
    }
```

Все эти операции реализуются вызовом соответствующего метода класса.

Операции отношения

Определим теперь операции отношения над рациональными числами – равно, не равно, больше, меньше и другие. Можно было бы, как в случае арифметических операций, вначале определить соответствующие методы, а уже потом через них определить операции. Но в данном случае определим только операции, не вводя традиционных методов.

```
public static bool operator ==(Rational r1, Rational r2)
    {
      return((r1.m ==r2.m)&& (r1.n ==r2.n));
    }
    public static bool operator !=(Rational r1, Rational r2)
    {
      return((r1.m !=r2.m)|| (r1.n !=r2.n));
    }
    public static bool operator <(Rational r1, Rational r2)
    {
      return(r1.m * r2.n < r2.m* r1.n);
    }
    public static bool operator >(Rational r1, Rational r2)
    {
      return(r1.m * r2.n > r2.m* r1.n);
    }
    public static bool operator <(Rational r1, double r2)
    {
      return((double)r1.m / (double)r1.n < r2);
    }
    public static bool operator >(Rational r1, double r2)
    {
      return((double)r1.m / (double)r1.n > r2);
    }
```

Обратите внимание, операции отношения определены не только над рациональными числами. Допускается сравнение рациональных и вещественных чисел, что позволяет сравнивать на больше-меньше числа типа Rational и числа любых арифметических типов – int, long, float, совместимых с типом double.

Переопределение методов родительского класса

Созданный нами класс Rational, как и все классы в языке C#, является наследником класса object. При этом наследуются методы родителя и их реализация, что не всегда приемлемо для наследника. Поэтому наследник, как правило переопределяет реализацию методов родителя с учетом собственной специфики.

Переопределим метод ToString так, чтобы он выдавал информацию о значении рационального числа в традиционном виде m/n. Переопределим также метод GetHashCode. Если клиент захочет размещать рациональные числа в хэш-таблице, то для каждого объекта требуется возвращаемый методом хэш-код, определяющий местоположение числа в таблице. Этот код

естественно определять, как некоторую функцию от полей класса. В нашем случае используем для этих целей побитовую операцию «исключающее или», выполняемую над полями класса. Поскольку мы определили операцию эквивалентности над рациональными числами, то разумно переопределить и родительский метод Equal. Вот код этих методов:

```
public override string ToString()
 {
     return m.ToString() + "/" + n.ToString();
 }
 public override bool Equals(object obj)
 {
     return (this == (Rational)obj);
 }
 public override int GetHashCode()
 {
     return (int)(m ^ n);
 }
```

Другие методы класса

Определим еще пару методов класса. Метод PrintRational по сути является модификацией метода ToString. Отличие состоит лишь в том, что к возвращаемой строке добавляется имя объекта, предаваемое методу в качестве аргумента:

```
public string PrintRational(string name)
    {
      return(name + " = " + m.ToString() +"/" +n.ToString());
    }//PrintRational
```

Более интересным является метод Round, позволяющий производить округление рационального числа, получая его приближенное значение. При выполнении последовательности операций над рациональными числами значение числителя и знаменателя как правило возрастает, хотя само число как дробь может быть невелико. В особенности это характерно при выполнении операции возведения в степень. В таких ситуациях полезна операция округления, упрощающая представление числа. Реализация этой операции базируется на методе округления целых чисел. Приведу тексты открытого и закрытого методов, занимающихся округлением:

```
public Rational RationalRound(int k)
        {
            // округляет рациональное число x, оставляя не более
            // k значащих цифр в числителе и знаменателе
            long u = LongRound(m, k);
            long v = LongRound(n, k);
            return (new Rational(u,v));
        }//RationalRound
        long LongRound(long x, int k)
        {
            // округляет число x, оставляя не более
            // k значащих цифр. Пример: 126784 -> 127000 (k=3)
            string s1 = x.ToString();
```

```
            if (k >= s1.Length) return x;
            string s2 = s1.Substring(0, k);
            string s3 = s1.Substring(k,1);
            if (int.Parse(s3) >= 5)
                s2 = (int.Parse(s2) + 1).ToString();
            for (int i = 0; i < s1.Length - k; i++)
                s2 += "0";
            return (long.Parse(s2));
        }//LongRound
```

Константы класса и специфические конструкторы

Последняя важная тема, которую мы разберем на примере конструирования класса Rational – это способ создания собственных констант класса. Предположим, что мы в классе Rational хотим создать две именованные константы – Zero и One, задающие соответственно нуль и единицу типа Rational. Задать эти константы, используя обычную для констант конструкцию const невозможно хотя бы по той причине, что такие константы должны быть инициализированы в момент объявления константными выражениями, но никаких константных выражений типа Rational быть не может, поскольку класс еще только объявляется. Необходимы обходные пути.

Прежде всего заметим, что константы по сути являются статическими полями, используемыми только для чтения. Это позволяет определить константы следующим образом:

```
//Константы класса 0 и 1 - Zero и One
    public static readonly Rational Zero, One;
```

Константы объявлены, но значений они не получили. Проблема состоит в том, что константы должны получить свои значения еще до того, как создан первый из объектов класса. Можно ли это сделать? Оказывается можно. Напомню, объекты, являющиеся экземплярами класса, динамически создаются в ходе выполнения программы в тот момент, когда выполняется операция new и вызывается конструктор класса. Однако кроме этих объектов автоматически создается еще один объект для каждого класса. Этот объект описывает класс, а не отдельные экземпляры. Этот объект содержит статические поля класса и ссылки на статические методы класса. Этот объект создается статическим конструктором класса. Никаких параметров этот конструктор естественно не имеет, поскольку вызывается автоматически. Вызывается этот конструктор до начала создания динамических объектов класса. Статический конструктор можно переопределить в классе и это открывает путь к заданию значения констант – статических полей класса.

Как правило, в классе определяется еще один конструктор – закрытый конструктор класса. Этот конструктор и создает объекты класса Rational, представляющие значения констант. Этот конструктор вызывается в теле статического конструктора. В совокупности это и позволяет справиться с проблемой констант. Вот как выглядит соответствующий код:

```
/// <summary>
    /// Закрытый конструктор класса. Имеет дополнительный
    /// параметр t, чтобы его сигнатура отличалась от
    /// сигнатуры открытого конструктора
    /// </summary>
    private Rational(long a, long b, string t)
    {
      m= a; n =b;
    }
    static Rational()
```

```
{
  //Статический конструктор класса без параметров
  // Определяет константы класса
  Zero = new Rational(0, 1, "private");
  One  = new Rational (1, 1, "private");
}//Статический конструктор
```

Итоги

На примере класса Rational рассмотрены многие важные детали, возникающие при проектировании классов. Подведем некоторые итоги:

- Проектируя класс, следует предусмотреть возможность возникновения исключительных ситуаций, при которых нормальная работа с объектами класса представляется невозможной. В таких случаях следует одновременно с проектируемым классом создавать специальные классы исключений. Это позволит в момент обнаружения исключительной ситуации выбросить (throw) исключение специального типа. Исключения предоставляют корректный способ борьбы с исключительными ситуациями.
- Поля класса как правило закрываются и недоступны клиентам класса. Различные стратегии доступа реализуются с помощью специальных процедур-свойств. В нашем примере эти процедуры отсутствуют, и у клиента нет способа добраться до поля класса. Рациональное число воспринимается как неделимое целое.
- Часть методов класса всегда открыты для клиента. Эти методы составляют интерфейс класса. Это тот набор служб, который класс предоставляет своим клиентам. Чтобы клиент мог эффективно использовать класс, ему достаточно знать сигнатуру открытых методов класса, знать спецификацию класса и спецификацию открытых методов.
- Часть методов класса закрыты для клиентов класса и используются для внутренних потребностей класса. Скрытие полей, скрытие методов, скрытие реализации делается прежде всего в интересах клиента. Если клиенты не используют информацию, находящуюся в закрытой части класса, то неизбежные изменения, сделанные в закрытой части класса, никак не скажутся на программе клиента.
- Во многих случаях наряду с традиционными методами класса полезно в классе определять операции (операторы или знаки операций). Это облегчает запись выражений над объектами класса.
- Важную роль в классе играют конструкторы. Создать объекты класса можно только с помощью конструктора. Как правило в классе создаются несколько открытых конструкторов с разным набором аргументов. Кроме того у класса всегда есть статический конструктор, вызываемый автоматически в интересах создания объекта, связанного с классом, а не с отдельными его экземплярами.
- Собственные константы класса нельзя создать обычным способом. Они создаются, используя статические поля класса, инициализируемые в момент вызова статического конструктора.

Калькулятор рациональных чисел – Windows-приложение

Теперь, когда у нас есть библиотека, позволяющая работать с рациональными числами, построим простейший калькулятор для работы с ними. Добавим в наше решение новый проект. Это можно сделать по-разному, но проще всего, не выходя из существующего решения, в меню

File выбрать New|Project. В открывшемся окне нового проекта в качестве типа проекта выбрать Windows Application, в окне Solution выбрать из списка «Add to Solution», дать имя проекту, например WRationalCalc, как сделал я, и приступить к работе над новым проектом.

Первое, что следует сделать, установить ссылку из проекта на созданную и хранящуюся в решении DLL. Для этого в меню Project выбрать пункт AddReference, в открывшемся окне выбрать вкладку Projects и из списка проектов, уже включенных в решение, выбрать имя проекта RationalsLibrary, хранящего построенную DLL. Заметьте, если нужно сослаться на проект из другого решения, то для этой цели существует вкладка Browse, позволяющая установить путь к нужному проекту.

Подготовительные работы на этом закончены и можно приступить непосредственно к построению калькулятора. Я не стал усложнять задачу и ограничился построением достаточно простого калькулятора. Его внешний вид в процессе работы показан на рис. 2_2.

Рис. 2_2 Калькулятор рациональных чисел в процессе работы

Спецификации, которые должен знать пользователь для корректной работы с калькулятором, описаны в теге summary класса Form1. Приведу описание этого класса, но прежде напомню, что проект создается в среде Visual Studio 2005, а потому класс Form1 представлен несколькими файлами и это та часть класса, в которой содержатся данные и обработчики событий, определяемые пользователем.

```
/// <summary>
  /// Калькулятор для работы с рациональными числами.
  /// В окна r1 и r2 следует ввести рациональные числа
  /// в формате m/n. После нажатия командных кнопок r1 и r2
  /// калькулятор готов к работе. Для выполнения операции над числами
  /// следует нажать кнопку с соответствующей операцией:
  /// (+, -, *, /, ^,Round).
  /// Результат операции сохраняется в r1.
  /// Аргумент для возведения в степень и округления задается в окне n.
  /// При неверном вводе данных выдается сообщение.
  /// </summary>
   public partial class Form1 : Form
   {
       Rational r1 = new Rational(1, 1);
       Rational r2 = new Rational(1, 1);
       int k = 0;
       public Form1()
       {
           InitializeComponent();
       }
    //обработчики событий кнопок калькулятора
   }
```

Три поля класса задают аргументы операций, выполняемых над рациональными числами, обеспечивая интерфейс с пользователем.

Рассмотрим обработчики событий кнопок r1, r2, n, передающих в поля класса данные, введенные пользователем в окна формы:

```
    private void button1_Click(object sender, EventArgs e)
```

```
{
    //ввод r1
    string s, s1, s2;
    int ind,m, n;
    s = textBox1.Text;
    try
    {
        ind = s.IndexOf('/');
        s1 = s.Substring(0, ind);
        s2 = s.Substring(ind + 1);
        m = int.Parse(s1);
        n = int.Parse(s2);
        r1 = new Rational(m, n);
        textBox1.Text = r1.PrintRational("");
    }
    catch (Exception ex)
    {
        textBox1.Text = ex.Message +
               "\r\nЗадайте корректный формат: n/m";
    }

}

private void button2_Click(object sender, EventArgs e)
{
    //ввод r2
    string s, s1, s2;
    int ind, m, n;
    s = textBox2.Text;
    try
    {
        ind = s.IndexOf('/');
        s1 = s.Substring(0, ind);
        s2 = s.Substring(ind + 1);
        m = int.Parse(s1);
        n = int.Parse(s2);
        r2 = new Rational(m, n);
        textBox2.Text = r2.PrintRational("");
    }
    catch (Exception ex)
    {
```

```
                textBox2.Text = ex.Message +
                    "Задайте корректный формат: n/m";
            }
        }
        private void button5_Click(object sender, EventArgs e)
        {
            //ввод k
            k = int.Parse(textBox3.Text);
            textBox3.Text = "=" + k.ToString();
        }
        private void button9_Click(object sender, EventArgs e)
        {
            //чистка окон r1, r2, n;
            textBox1.Text = "";
            textBox2.Text = "";
            textBox3.Text = "";
        }
```

Ввод данных, осуществляемый пользователем, всегда следует контролировать. Поэтому преобразование данных из полей формы в поля класса организовано в виде try-catch блока, предусматривающее возможность появления в процессе ввода исключительных ситуаций и их обработку.

Пользователи склонны периодически нарушать заданные спецификации. У нарушений могут быть две разные причины – либо знаменатель рационального числа равен нулю, либо неверно задан формат вводимых данных. В первом случае ситуация обнаружится в конструкторе класса Rational, где будет выброшено исключение соответствующего типа. Во втором случае ошибка обнаружится в процессе преобразования введенной строки в число. В обоих случаях пользователю будет выдано сообщение, описывающее возникшую ситуацию, и он сможет повторить ввод.

Обработчики события Click для кнопок, издающих операции над рациональными числами устроены просто и похожи друг на друга. В классе Rational определены знаки операций, что позволяет переменной r1 присвоить бинарное выражение с заданной бинарной операцией. Лишь в обработчике, реализующем операцию округления рационального числа, вызывается метод RationalRound класса Rational.

```
private void button4_Click(object sender, EventArgs e)
        {
           //сложение
            r1 = r1 + r2;
            textBox1.Text = r1.PrintRational("");
        }

        private void button3_Click(object sender, EventArgs e)
        {
            //вычитание
            r1 = r1 - r2;
```

```
            textBox1.Text = r1.PrintRational("");
        }

        private void button7_Click(object sender, EventArgs e)
        {
            //умножение
            r1 = r1 * r2;
            textBox1.Text = r1.PrintRational("");
        }

        private void button6_Click(object sender, EventArgs e)
        {
            //деление
            r1 = r1 / r2;
            textBox1.Text = r1.PrintRational("");
        }

        private void button8_Click(object sender, EventArgs e)
        {
            //возведение в целую степень
            r1 = r1 ^ k;
            textBox1.Text = r1.PrintRational("");
        }

        private void button10_Click(object sender, EventArgs e)
        {
            //округление до k значащих цифр
            r1 = r1.RationalRound(k);
            textBox1.Text = r1.PrintRational("");
        }
```

Для демонстрации сложно устроенных выражений над рациональными числами на форму посажена кнопка Expression, обработчик события Click которой вычисляет сложно устроенное выражение:

```
    private void button11_Click(object sender, EventArgs e)
        {
            //пример выражения над рациональными числами
            if ((r1 * r1 - r2 ^ 3) > Rational.One)
                r1 = r1 / r2 + new Rational(24, 46);
            else
                r1 = (r1 + r2) ^ 2;
            textBox1.Text = r1.PrintRational("");
        }
```

Консольный калькулятор для работы с рациональными числами

Для полноты картины добавим в наше решение еще один проект, реализующий калькулятор для работы с рациональными числами, но уже в консольном варианте. Добавление нового проекта в решение и его связывание с библиотекой выполнялось по уже описанной схеме.

Построим традиционное консольное приложение, где в некотором цикле, управляемом пользователем, на каждом шаге ему предлагается меню из возможных команд. Пользователь выбирает команду, вводит нужные данные для этой команды, затем команда обрабатывается, результат ее выполнения выводится на экран, и все повторяется, пока пользователь не откажется от продолжения работы.

Анализ выбора пользователя строится по известной схеме разбора случаев, а тело каждого варианта case практически совпадает с тем, что делается в обработчике события Click при выборе соответствующей кнопки в калькуляторе Windows. Приведу текст консольного проекта:

```
namespace ConsoleCalc
{
 class Program
 {
     static Rational r1, r2;
     static int n;
     static int menu;
     static void Main(string[] args)
     {
         string answer = "yes";
         string invite;
         r1 = new Rational(1,1);
         r2 = new Rational(1,1);
         invite = "Выберите пункт меню:\r\n";
         invite += "0 - ввод r1\r\n";
         invite += "1 - ввод r2\r\n";
         invite += "2 - r1 = r1+r2\r\n";
         invite += "3 - r1 = r1-r2\r\n";
         invite += "4 - r1 = r1*r2\r\n";
         invite += "5 - r1 = r1/r2\r\n";
         invite += "6 - (r1 > r2)?\r\n";
         invite += "7 - (r1 < r2)?\r\n";
         invite += "8 - (r1 == r2)?\r\n";
         invite += "9 - r1 = r1^n\r\n";
         invite += "10 - r1 = r1.Round(n)\r\n";
         do
         {

             Console.WriteLine(invite);
             menu = int.Parse(Console.ReadLine());
             Parsing();
```

```
            Console.WriteLine("Продолжим? (yes/no)");
            answer = Console.ReadLine();
        } while (answer == "yes");
    }
    static void Parsing()
    {
        //Разбор случаев. Выполняются действия согласно
        //выбранному пункту меню.
        int a=1, b=1;
        bool r = true;
        switch (menu)
        {
            case 0:
                {
                    Console.WriteLine("Введите целое - числитель r1");
                    a = int.Parse(Console.ReadLine());
                    Console.WriteLine("Введите целое - знаменатель r1");
                    b = int.Parse(Console.ReadLine());
                    r1 = new Rational(a, b);
                    break;
                }
            case 1:
                {
                    Console.WriteLine("Введите целое - числитель r2");
                    a = int.Parse(Console.ReadLine());
                    Console.WriteLine("Введите целое - знаменатель r2");
                    b = int.Parse(Console.ReadLine());
                    r2 = new Rational(a, b);
                    break;
                }
            case 2:
                {
                    Console.WriteLine(r1.PrintRational("r1"));
                    Console.WriteLine(r2.PrintRational("r2"));
                    r1 = r1+r2;
                    Console.WriteLine(r1.PrintRational("r1+r2"));
                    break;
                }
            case 3:
                {
                    Console.WriteLine(r1.PrintRational("r1"));
```

```
            Console.WriteLine(r2.PrintRational("r2"));
            r1 = r1 - r2;
            Console.WriteLine(r1.PrintRational("r1-r2"));
            break;
        }
    case 4:
        {
            Console.WriteLine(r1.PrintRational("r1"));
            Console.WriteLine(r2.PrintRational("r2"));
            r1 = r1 * r2;
            Console.WriteLine(r1.PrintRational("r1*r2"));
            break;
        }
    case 5:
        {
            Console.WriteLine(r1.PrintRational("r1"));
            Console.WriteLine(r2.PrintRational("r2"));
            r1 = r1 / r2;
            Console.WriteLine(r1.PrintRational("r1/r2"));
            break;
        }
    case 6:
        {
            Console.WriteLine(r1.PrintRational("r1"));
            Console.WriteLine(r2.PrintRational("r2"));
            r = (r1 > r2);
            Console.WriteLine("(r1>r2) == "+r.ToString());
            break;
        }
    case 7:
        {
            Console.WriteLine(r1.PrintRational("r1"));
            Console.WriteLine(r2.PrintRational("r2"));
            r = (r1 < r2);
            Console.WriteLine("(r1<r2) == " + r.ToString());
            break;
        }
    case 8:
        {
            Console.WriteLine(r1.PrintRational("r1"));
            Console.WriteLine(r2.PrintRational("r2"));
```

```
                    r = (r1 == r2);
                    Console.WriteLine("(r1==r2) == " + r.ToString());
                    break;
                }
            case 9:
                {
                    Console.WriteLine("Введите целое - степень r1");
                    n = int.Parse(Console.ReadLine());
                    Console.WriteLine(r1.PrintRational("r1"));
                    r1 = r1 ^ n;
                    Console.WriteLine(r1.PrintRational("r1^n"));
                    break;
                }
            case 10:
                {
                    Console.WriteLine("Введите целое - " +
                        "число значащих цифр при округлении r1");
                    n = int.Parse(Console.ReadLine());
                    Console.WriteLine(r1.PrintRational("r1"));
                    r1 = r1.RationalRound(n);
                   Console.WriteLine(r1.PrintRational("r1.RationalRound("
                        + n.ToString()));
                    break;
                }
        }
    }
}
```

На рис. 2_3 показан процесс работы с консольным калькулятором:

Рис. 2_3 Консольный калькулятор в процессе работы

В заключение несколько слов о том, как запускать нужный проект на выполнение. В построенном нами решении два проекта – консольный и Windows, каждый из которых является выполняемым. Выбрав для любого из этих проектов в меню Project пункт «Set as StartUp Project», можно установить, какой из проектов будет запускаться на выполнение. Большего можно добиться на странице свойств решения, где можно не только установить стартовый проект, но запустить оба проекта одновременно, включив возможное свойство «Multiple startup projects».