Программы с ветвлениями и циклами

На этом занятии будет приведено несколько примеров программирования циклических и ветвящихся алгоритмов.

Задача 1. Разложение числа на простые множители

Простое число — это натуральное число, большее единицы, которое имеет ровно два делителя: единицу и самого себя.

Основная теорема арифметики. Каждое натуральное число $n > 1$ представляется в виде произведения простых чисел, причём такое представление единственно с точностью до порядка следования сомножителей.

Найдём разложение натурального числа n на простые множители.

Решение

Вспомним рассуждения задачи 2 прошлого занятия. Из них можно сделать следующий вывод. Если у числа n есть делитель, больший целой части его квадратного корня $\lfloor\sqrt{n}\rfloor$, то обязательно существует и делитель, меньший $\lfloor\sqrt{n}\rfloor$. Для решения данной задачи можно

1. найти все простые числа d_i, не превосходящие $\lfloor\sqrt{n}\rfloor$;
2. делить на каждое из d_i число n, пока оно делится, заменяя значение n на $\frac{n}{d_i}$. Когда n на d_i делиться перестанет, оно не будет делиться и на меньшие d_i простые числа.

Но на самом деле вовсе не обязательно осуществлять поиск простых делителей. Достаточно это проделывать со всеми числами d_i от 2 до $\lfloor\sqrt{n}\rfloor$ в порядке возрастания. Если какое-то из d_i окажется составным, n на него делиться всё равно не будет, потому что мы уже разделили n на простые, входящие в разложение самого d_i. При этом каждый раз при изменении значения n следует бежать до нового значения $\lfloor\sqrt{n}\rfloor$ для того, чтобы в итоге в n оказалось единственное простое число. Однако, если n окажется квадратом какого-то натурального числа, после выполнения описанной последовательности действий n окажется равным единице. При d_i>2 чётные числа в качестве делителя проверять не будем — они не являются простыми и могут быть пропущены для ускорения работы алгоритма.

Ниже приведена программа с комментариями. Комментарии компилятором не обрабатываются. В Delphi и FreePascal комментарии заключаются в `{ комментарий }` или `(* комментарий *)`. Они могут распространяться на несколько строк. Если надо написать комментарий с данного места до конца строки, это делают после `// комментарий`. Комментарии в фигурных скобках, начинающиеся со знака $, являются на самом деле директивами управления компилятором. Одну из них мы уже неоднократно применяли для переключения FreePascal в режим совместимости с Delphi.

```
{$mode delphi}
Program factoriz;
Var n,i,d : Integer;
Begin
  Write('N=');
  Read(n);
  d:=2;
  While d <= trunc(sqrt(n)) Do
  Begin
    While n mod d = 0 Do (* Пока делится… *)
    Begin
      n:=n div d; (* … делим *)
      Write(d,' ')
    End;
```

```
    If d=2 Then d:=3
    Else d:=d+2;     (* Чётные пропускаем *)
  End;
  If n<>1 Then WriteLn(n);
  WriteLn (*Просто для перевода курсора на новую строку *)
End.
```

Задача 2. Нахождение НОД

Найти наибольший общий делитель (НОД) двух натуральных чисел a и b.

Решение

Многим с уроков математики известен следующий алгоритм решения этой задачи.

1. Разложить на простые множители число a. Получим множество A. Элементы этого множества могут повторяться.
2. Разложить на простые множители число b. Получим множество B.
3. Найти пересечение множеств, полученных в п. 1 и п. 2: $C = A \cap B$.
4. Перемножить все элементы множества C.

Например, пусть нам надо найти НОД чисел 150 и 1125. 150=2*3*5*5, 1125=3*3*5*5*5. Тогда НОД=3*5*5=75.

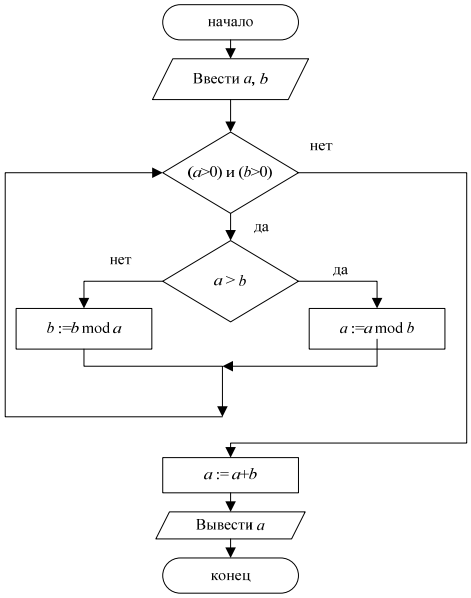
Но это не оптимальный алгоритм. Он требует выполнения большого объёма вычислений. Нахождение простых чисел само по себе уже непростая задача. А применяемые при составлении программ алгоритмы должны быть, по возможности, быстродействующими. Время работы многих программ является их очень важной характеристикой. А в некоторых случаях, например, в системах автоматического управления, может оказаться критичным.

Между тем эффективный алгоритм решения этой задачи известен очень давно.

Алгоритм Евклида

Пока оба числа отличны от нуля, заменять большее из них остатком от деления большего на меньшее. Результат — отличное от нуля число, полученное после выхода из цикла. (Т.е. если оказалось a=0, то результат в переменной b, иначе результат в a.)

Для повышения эффективности алгоритма заметим, что вместо сравнения с нулём числа a можно просто найти сумму a+b. Она и будет результатом. Это работает быстрее, потому что операции сравнения и передачи управления одной из ветвей алгоритма теперь не замедляют работу программы. Вот блок-схема полученного алгоритма.



Проиллюстрируем эго работу на том же примере: найдём НОД чисел *a*=150 и *b*=1125.

1. *b* := 1125 mod 150. Теперь *b*=75.
2. *a* := 150 mod 75. Теперь *a*=0.
3. *a* := 0+75. *a*=75 и есть НОД.

Понятно, что в заголовке цикла вместо знака «больше» можно было поставить знак «не равно».

Напишем программу. Будем требовать, чтобы она могла работать для как можно больших целых чисел, представление которых возможно стандартными типами данных языка Паскаль. Поэтому тип данных выберем Int64.

```
{$mode delphi}
Program NOD;
Var a,b : Int64;
Begin
  Write('a,b=');
  Read(a,b);
  While (a>0) and (b>0) Do
     If a>b
     Then a:=a mod b
     Else b:=b mod a;
  a:=a+b;
  WriteLn('НОД=',a)
End.
```

Если вы будете испытывать программу в Delphi, то «{$mode delphi}» не пишите. Если же во FreePascal, то можно применить тип данных qword. Максимально возможное значение чисел в этом случае будет ещё больше.

Задача 3. Быстрое возведение в степень

Вычислить x^n, где *x* — вещественное число, *n* — целое неотрицательное.

Решение

Простейший алгоритм решения этой задачи следует из определения степени с натуральным показателем и с показателем, равным нулю: взять некоторую переменную *p*, равную единице, и *n* раз её умножить на *x*. Но этот алгоритм неэффективен. Он выполняем слишком много лишних умножений.

Количество умножений можно значительно уменьшить, если при чётном *n* воспользоваться равенством $x^n = (x^2)^{\frac{n}{2}}$. В этом случае количество действий сокращается примерно в 2 раза и показатель степени остаётся целым. При нечётном *n* пользуемся равенством $x^n = x \cdot x^{n-1}$, превращая нечётный показатель степени в чётный. И так, пока *n*, постоянно уменьшаясь, не превратится в нуль. Количество действий, выполняемых в этом алгоритме, порядка log *n*.

Ниже приведена программа, реализующая этот алгоритм. Для переменной *x* выбран тип Extended, чтобы хранить число с максимальной точностью. Имейте в виду, что эта точность всегда ограничена. Поэтому, если ваша программа должна работать только с целыми числами и давать точные результаты, используйте целочисленные типы данных, например, Int64.

```
{$mode delphi}
Program power;
Var x,p : Extended;
    n   : Integer;
Begin
  Write('x,n = ');
  Read(x,n);
  p:=1;
  While n>0 Do
  Begin
    If n mod 2 = 1
    Then p:=p*x;
```

```
    x:=x*x;
    n:=n div 2
  End;
  WriteLn(p:0:5)
End.
```

В теле цикла этой программы сначала проверяется *n* на нечётность и при необходимости выполняется одно умножение, приводя показатель степени к чётному значению. Потом основание степени заменяется его квадратом, и лишь в конце корректируется значение *n*. Заметим, что уменьшение *n* на единицу после p:=p*x никакого эффекта не дало бы.

Теперь о WriteLn(p:0:5). Нуль в ширине поля вывода означает лишь то, что на ширину поля вывода мы никаких ограничений не накладываем. Просто хотим видеть 5 знаков в записи дробной части числа.

Задачи для самостоятельного решения

1. Напишите программу вычисления наименьшего общего кратного двух чисел. Учитывайте, что НОД(a,b) * НОК(a,b) = a*b.
2. Напишите программу вычисления значения выражения $\frac{a}{b}+\frac{c}{d}$, где *a, c* — целые, а *b, d* — натуральные числа. Ответ должен быть несократимой дробью. Выведите его в формате *m*/*n*.
3. Напишите программу получения канонического разложения натурального числа на простые множители. Каноническим разложением числа *n* называется разложение вида $n = p_1^{k_1} \cdot p_2^{k_2} \cdot \ldots \cdot p_m^{k_m}$, где $p_1 < p_2 < \ldots < p_m$ — простые числа, $k_1, k_2, \ldots, k_m$ — натуральные числа. Показатель степени, равный единице, не выводите. Операцию умножения при выводе обозначайте *, возведение в степень ^. Например, для *n*=15000 программа должна вывести 15000=2^3*3*5^4.