УДК 004.438

РЕКУРСИЯ КАК МЕТОД РЕШЕНИЯ МАТЕМАТИЧЕСКИХ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ПО ЭВМ

В.А. Касторнова

ФГБНУ «Институт стратегии развития образования Российской академии образования», Москва

kastornova_vasya@mail.ru

Аннотация

Рассматриваются вопросы реализации рекурсивных соотношений при решении математических задач с использованием инструментальных средств программного обеспечения ЭВМ. Подчеркивается, что составление программ решения таких задач способствует более глубокому пониманию сущности процесса рекурсии.

Ключевые слова: *рекурсивное соотношение, рекурсивный подход, рекурсивная формула, инструментальное средство, язык программирования, рекурсивная функция (процедура), рекурсивный выход, стек рекурсии*

Метод рекурсии является одним из мощных средств решения всевозможных математических задач, в основу которого заложен циклический процесс получения новых величин (данных) на основе ранее полученных. Создание этих новых величин происходит по некоторому алгоритму, который задается соответствующей рекурсивной формулой $x_i = f(x_{i-1})$. Вполне естественно, что при таком подходе первая исходная величина должна быть определена заранее. В более сложных случаях рекурсивная формула задается как функция многих переменных, представляющих собой несколько предыдущих значений, с помощью которых вычисляется очередное значение $x_i = f(x_{i-1}, x_{i-2}, \ldots, x_{i-n})$. Примерами рекурсивных формул являются формулы для вычисления членов арифметической и геометрической прогрессий, нахождения факториала и целой степени чисел, чисел Фибоначчи и другие. Рекурсивный подход используется и при решении других задач, например, при работе с древовидными структурами данных [4].

Реализация рекурсивных соотношений с помощью средств информационных (компьютерных) технологий осуществляется с помощью использования языков программирования высокого уровня. Практически все языки программирования обладают возможностями создания рекурсивных процедур и функций. Функция (процедура) называется *рекурсивной*, если в ее описании происходит вызов самой себя. Процесс обращения к ним называется *рекурсией* [3].

Продемонстрируем использование рекурсии на примере вычисления на языке Паскаль значения факториала произвольного натурального числа N. В математике известно рекурсивное определение факториала: n! = 1, при n = 0; n! = (n-1)!*n, при n > 0. Это рекурсивное определение можно реализовать с помощью соответствующей рекурсивной функции:

```
function FACTORIAL (VALUE: integer): integer;
begin
   if VALUE= 0 then FACTORIAL:= 1
                  else FACTORIAL:= VALUE*FACTORIAL (VALUE-1)
end;
```

Теперь можно обращаться к этой функции из тела основной программы, как показано в следующем примере:

```
program FINDFACTORIAL;
var N: integer;
begin
  writeln ('Введите число'); readln (N);
  if N<0 then writeln ('Нет факториала')
        else writeln ('Факториал', N, 'равен', FACTORIAL (N))
end.
```

Характерной особенностью построенной функции является наличие в ее теле оператора присваивания FACTORIAL:=VALUE*FACTORIAL(VALUE-1), где происходит вызов определяемой функции. Здесь идентификатор FACTORIAL в левой части оператора обозначает имя переменной для хранения значения функции, а в правой - имя вызываемой функции.

Важным моментом при составлении любой рекурсивной функции является организация выхода из рекурсии. При организации вычислений при помощи рекурсии всегда существует нерекурсивное решение, например, 0! = 1. Рекурсив-

ный процесс должен шаг за шагом так упрощать задачу, чтобы, в конце концов, для нее появилось нерекурсивное решение. В нашем случае условием завершения рекурсии является VALUE=0.

При описании рекурсивных функций необходимо хорошо представлять процесс осуществления вычислений. Всякая рекурсия состоит из *двух этапов*: углубление (погружение) внутрь рекурсии и выход из нее. На первом этапе никаких вычислений не производится, а идет только настройка рабочей формулы на конкретные операнды. На втором этапе происходит процесс вычислений по настроенным формулам.

Рассмотрим рекурсивный процесс на примере вычисления факториала для N = 3. Получим следующие шаги:

1) N = 3, где N<>0, следовательно, FACTORIAL:=3*FACTORIAL(2);

2) N = 2, где N<>0, следовательно, FACTORIAL:=2*FACTORIAL(1);

3) N = 1, где N<>0, следовательно, FACTORIAL:= 1*FACTORIAL (0);

4) N =0, следовательно, FACTORIAL:=1,

т.е. получили нерекурсивное значение. Углубление в рекурсию закончено, далее пойдет процесс выхода из нее с выполнением необходимых вычислений по полученным формулам снизу вверх.

В выражение 1*FACTORIAL(0) вместо FACTORIAL(0) подставляется его значение 1, вычисляется произведение 1*1 и оно становится значением FACTORIAL(1). В выражение 2*FACTORIAL(1) вместо FACTORIAL(1) подставляется значение 1, вычисляется 2*1 и оно становится значением FACTORIAL(2). В выражение 3*FACTORIAL(2) вместо FACTORIAL(2) подставляется значение 2, вычисляется 3*2 и оно становится значением переменной FACTORIAL, которая возвращает в основную программу значение 3!.

Весь этот двухэтапный рекурсивный процесс реализуется в памяти ЭВМ с помощью организации в ней стека рекурсии. Дело в том, что для хранения значений переменной N (а значит и переменной VALUE) отводится не одна ячейка, а стек с именем N. В этот стек последовательно заносятся значения 3, 2, 1, 0, причем значение 0 есть признак конца заполнения стека. Затем начинает работать цикл с телом FACTORIAL:= FACTORIAL*N, где значения N выбираются последовательно из стека в порядке 1, 2, 3. Исходным же значением переменной FACTORIAL является 1, как значение 0! (см. таб. 1).

Таблица 1. Работа стека

Заполнение стека (углубление)	**Стек**	**Вычисление (разуглубление)**
FACTORIAL:= 1	0	FACTORIAL:= 1
FACTORIAL:= 1*FACTORIAL (0)	1	FACTORIAL:= 1*FACTORIAL
FACTORIAL:= 2*FACTORIAL (1)	2	FACTORIAL:= 2*FACTORIAL
FACTORIAL:= 3*FACTORIAL (2)	3	FACTORIAL:= 3*FACTORIAL

В заключение покажем, что часто рекурсивные функции строятся гораздо проще, чем нерекурсивные, хотя вполне понятно, что не всякая функция может быть переделана на рекурсивную. Сделаем это на примере уже построенной ранее функции POWER.

Действительно, данная функция, как и нахождение факториала, явно носит рекурсивный характер, исходя из ее определения: $X^n = 1$, если n = 0; $X^n = (X^{n-1})*X$, если n > 1. Поэтому ее вычисление похоже на вычисление факториала:

```
function POWER_REC (FACTOR: real; EXPONENT: integer): real;
begin
  if EXPONENT < 0 then POWER_REC:= 1/ POWER_REC (FACTOR, abs(EXPONENT))
                  else if EXPONENT > 0
      then POWER_REC:= FACTOR* POWER_REC (FACTOR, EXPONENT-1)
      else POWER_REC:= 1
end;
```

Помимо рекурсивных функций в языке Паскаль можно определять, по тому же принципу, и рекурсивные процедуры. Покажем, как рекурсивная функция может быть переделана в рекурсивную процедуру на примере вычисления факториала.

```
procedure FACTORIAL_REC (VALUE: integer; var F: integer);
begin
  if VALUE=0 then F:= 1
```

```
                    else begin
 FACTORIAL_REC (VALUE-1, F);
F:= F*VALUE
                        end;
  end;
```

Отметим, что для вычисления N! с помощью рекурсивной функции следует применить оператор присваивания F = *FACTORIAL(N)?,* а здесь необходимо вызвать эту процедуру с помощью оператора процедуры FACTORIAL_REC (N, F), где F - переменная для возвращения из процедуры значения N! [2].

То касается использования рекурсии при изучении древовидных структур данных, то она работает как при формировании самого дерева, так и при работе с ним. В двоичном дереве каждая вершина (кроме листа) имеет не более двух ветвей, которые называют левым и правым поддеревьями. Дерево называется *идеально сбалансированным*, если разница между числом вершин в его левом и правом поддеревьях (на всех уровнях) не более 1.

Для построения такого дерева из N элементов (вершин) используется следующее рекурсивное правило:

1. Выбирается один из элементов в качестве корня.
2. Строится левое поддерево с количеством вершин NL = N div 2.
3. Строится правое поддерево с числом вершин NR = N-NL-1.

Рекурсивное правило построения идеально сбалансированного дерева, сформулированное выше, лежит в основе рекурсивной функции формирования дерева. У этой функции в качестве параметра-аргумента выступает число вершин дерева, а значением функции является ссылка - указатель на следующую вершину. Функция формирования идеально сбалансированного дерева принимает вид:

```
 function FORMIR_TREE_BALANCE (N: integer): SS;
var Z: SS; NL, NR: integer;
begin
    if N = 0 then Z:= nil {Пустое дерево}
     else
      begin
         NL:= N div 2; NR:= N-NL-1; new (Z);
```

```
            write ('Ввести вершину'); read (Z^.k);
            Z^.left:=FORMIR_TREE_BALANCE (NL); {Формирование левого под-
дерева}
            Z^.right:=FORMIR_TREE_BALANCE (NR); {Формирование правого
поддерева}
          end;
        FORMIR_TREE:= Z; {Запоминание ссылки на корень дерева}
      end;
```

Формирование дерева производится с помощью рекурсивной функции, дающей ссылку на его корень. Поэтому при создании деревьев необходимо осуществлять вызов этой функции в основной программе, выделяя глобальную переменную типа SS для хранения ссылки на корень этого дерева. Например, с помощью оператора DER:= FORMIR_TREE_BALANCE (N) формируется дерево из N вершин, а ссылка на его корень присваивается переменной DER.

Чтобы вывести дерево на экран, необходимо предусмотреть его обход с помощью рекурсивной процедуры, аналогичной процедуре его создания:

```
    procedure VIVOD_TREE (Z: SS; N: integer; var Y: integer);
  var i: integer;
  begin
     Y:= (N-5)/5 - 1;{Подсчет числа уровней дерева}
     if Z <> nil then
       begin
          VIVOD_TREE (Z^.right, N+5, Y);
          for i:=1 to N do write (' ');
          writeln (Z^.k);
          writeln;
          VIVOD_TREE (Z^.left, N+5, Y);
       end;
    end;
```

Эта рекурсивная процедура выводит на экран элементы слева направо, располагая дерево в горизонтальном положении, где крайним левым элементом оказывается корень дерева, а все правые элементы - листья. Между соседними уровнями процедура оставляет 5 пробелов, а между элементами одного

уровня - одну строку. В процедуре выходной параметр Y служит для указания числа уровней построенного дерева. Формула (N-5)/5-1 дает число уровней, т. к. по построению между элементами соседних уровней находится 5 пробелов. По завершению работы последним выводится на экран самый левый (самый нижний и, значит, самый удаленный от левого края экрана) лист дерева.

Рекурсивный подход проявляет себя и при решении задачи поиска элемента (вершины) в дереве. Так как образование дерева с помощью рекурсивной функции идет по двум ветвям, то и поиск элемента тоже должен реализовываться по тому же принципу. Результат поиска элемента в дереве может быть представлен двумя способами: 1) выводом значения логической переменной о наличии/отсутствии элемента в дереве; 2) определением ссылки на звено, содержащее искомый элемент. Процедуру поиска элемента в дереве организуют в виде рекурсивной процедуры, в которой имеются:

1) входные параметры (параметры-значения) - ссылка на дерево (т.е. на корень дерева, где ищется элемент) и значение элемента поиска;

2) выходной параметр (параметр-переменная) - ссылка на найденный элемент.

Процедура поиска элемента в дереве имеет вид:

```
procedure POISK (S: SS; ZNACH: integer; var ELEM: SS);
begin
      if S <> nil then if S^.k = ZNACH then ELEM:= S
                  else
          begin
              POISK (S^.left, ZNACH, ELEM);
              POISK (S^.right, ZNACH, ELEM);
          end;
end;
```

Рекурсивный поиск элемента ZNACH в дереве заканчивается получением ссылки на него, значение которой присваивается переменной ELEM. Если такого элемента в дереве нет, то переменная ELEM не будет определена, т.к. на оператор ELEM:= S программа выходит только при условии S^.K = ZNACH. Неопределенность значения переменной ELEM означает, что в ней может находиться "мусор", поэтому, чтобы этого не происходило, нужно перед использованием про-

цедуры поиска в основной программе присвоить результирующей переменной значение NIL. Это значение будет показывать, что элемент не найден. Если в качестве входного параметра для процедуры взять корень дерева, то ответ будет получен сразу, без проведения рекурсии.

Рекурсивная процедура поиска POISK обходит сначала левое поддерево, а потом правое. Если искомый элемент находится в правом поддереве, то после обхода левого поддерева при обнаружении искомого элемента рекурсия прекращается, поэтому лишних шагов не делается. При поиске элемента в левом поддереве после его нахождения обход левого поддерева прекращается, но происходит (согласно работе рекурсивной процедуры) переход на правое поддерево, и выполняются шаги, уже не влияющие на результат поиска. Это можно увидеть на экране, если добавить в процедуру POISK вывод проходимых вершин во время поиска элемента в дереве, что отражено в процедуре POISK_TRACE:

```
procedure POISK_TRACE (S: SS; ZNACH: integer; var ELEM: SS);
begin
  if S <> nil then if S^.k = ZNACH then  begin write (S^.k:3); i:= i+1; ELEM:= S
end
                else
               begin write (S^.k:3); i:= i+1;
                 POISK_TRACE (S^.left, ZNACH, ELEM);
                 POISK_TRACE (S^.right, ZNACH, ELEM);
               end;
 end;
```

Чтобы блокировать переход на правое поддерево необходимо сопроводить рекурсивный вызов правой ветви условием того, что ELEM = NIL. Действительно, это условие показывает, что элемент в левом поддереве не найден и надо продолжать поиск. Если же элемент найден в левом поддереве, то выполнится условие ELEM < > NIL и произойдет выход из рекурсии. Получаем процедуру POISK_CHORT короткого поиска в сбалансированном дереве:

```
procedure POISK_CHORT (S: SS; ZNACH: integer; var ELEM: SS);
     begin
  if S <> nil then begin
                  if S^.k = ZNACH  then  ELEM:= S;
```

```
        write (S^.k:3);  i:= i+1;
        if ELEM = nil then POISK_CHORT (S^.left, ZNACH, ELEM);
        if ELEM = nil then POISK_CHORT (S^.right, ZNACH, ELEM);
      end;
  end;
```

Поиск элемента начинается с левого поддерева. Если элемент найден (в любом поддереве), то поиск прекращается, так как переменная ELEM получает значение ссылки на найденный элемент, то есть значение ELEM отлично от NIL. Особенностью работы процедуры POISK_CHORT является то, что в ней есть глобальная переменная I для подсчета числа пройденных вершин, которая должна быть обнулена в основной программе перед вызовом этой процедуры [1].

Были рассмотрены примеры использования рекурсии при работе с идеально сбалансированным двоичным деревом, однако этот подход распространяется и на другие типы деревьев, таких как дерево поиска.

В заключение отметим, что использование средств информационных технологий при изучении математики, на наш взгляд, существенно повышается мотивация работы обучающихся, так как они позволяют более наглядно и понятно познавать протекающие в учебном материале процессы. И ярким примером тому служит проблематика рекурсии. Можно на словах дать ее понятие, рассказать о ее особенностях на некоторых наглядных примерах. Но дело обстоит совсем иначе, когда помимо знания теории решаются практические задачи на составление компьютерных программ, где подключатся рекурсивные процедуры и функции, создание которых не возможно без глубокого понимания процесса рекурсии.

СПИСОК ЛИТЕРАТУРЫ

1. *Касторнов А.Ф.*, *Касторнова В.А.* Языки программирования и их роль в становлении предметной области «Информатика» // Педагогическая информатика, 2016, № 1, С. 59–68.

2. *Касторнова В.А.* Структуры данных и алгоритмы их обработки на языке программирования Паскаль. СПб.: БХВ-Петербург, 2016, 304 с.

3. *Мальцев А.И.* Алгоритмы и рекурсивные функции. М.: Наука, 1986, 386 с.

4. *Пильщиков В.Н., Горячева И.В., Бордаченкова Е.А.* Решение задач с использованием рекурсии. Учебно-методическое пособие. М.: МГУ, 2012, 38 с.

RECURSION AS A MATHEMATICAL PROBLEMS SOLVING METHOD USING INSTRUMENTAL SOFTWARE RESOURCES

Vasilina Kastornova

The Federal State Budgetary Scientific Institution "Institute for Strategy and Theory of Education of the Russian Academy of Education", Moscow

kastornova_vasya@mail.ru

Abstract

The article discusses the recursive ratios implementation in solving mathematical problems using instrumental software resources. It is emphasized that programs development for solving such problems contributes to a deeper understanding of the recursion process essence.

Keywords: *recursive ratio, recursive method, recursive formula, instrumental resource, programming language, recursive function (procedure), recursive outlet, recursion stack*

REFERENCES

1. *Kastornov A.F., Kastornova V.A.* Yazy`ki programmirovaniya i ix rol` v stanovlenii predmetnoj oblasti «Informatika» // Pedagogicheskaya informatika, 2016, No 1, S. 59–68.

2. *Kastornova V.A.* Struktury` danny`x i algoritmy` ix obrabotki na yazy`ke programmirovaniya Paskal`. SPb.: BXV-Peterburg, 2016, 304 s.

3. *Mal`cev A.I.* Algoritmy` i rekursivny`e funkcii. M.: Nauka, 1986, 386 s.

4. *Pil`shhikov V.N., Goryacheva I.V., Bordachenkova E.A.* Reshenie zadach s ispol`zovaniem rekursii. Uchebno-metodicheskoe. M.: MGU, 2012, 38 s.

СВЕДЕНИЯ ОБ АВТОРЕ

КАСТОРНОВА Василина Анатольевна – кандидат педагогических наук, доцент, старший научный сотрудник лаборатории математического общего образования и информатизации ФГБНУ «Институт стратегии развития образования Российской академии образования», Москва, Россия.

Vasilina KASTORNOVA – Candidate of Pedagogical Sciences, Associate Professor, Senior Researcher of the Laboratory of Mathematical General Education and Informatization of the Federal State Budgetary Scientific Institution «Institute for Strategy and Theory of Education of the Russian Academy of Education», Moscow, Russia.

email: kastornova_vasya@mail.ru

Материал поступил в редакцию 12 сентября 2019 года