Комитет по народному образованию
Администрации Солнечногорского муниципального района
муниципальное общеобразовательное учреждение
Тимоновская средняя общеобразовательная школа
Солнечногорск-7 тел.:61-36-46
ул.Подмосковная email: timon.school@mail.ru

Реферат

"Дюжина задач с рекурсией"

Подготовила: учитель информатики
Тимоновской средней школы
Пушкова И. А.

Солнечногорск – 7
2012 учебный год

Понятие *"рекурсия"* происходит от слова *"recurrence"*, в переводе с английского означающего *возвращение, повторение.* Рекурсивный алгоритм – алгоритм, который в процессе работы обращается сам к себе. Реализация рекурсивного метода стала возможной с момента появления языков программирования Алгол, Паскаль и других, поддерживающих структуру программы с описанием локальных и глобальных переменных. Суть заключается в том, что при каждом вызове процедуры создается ее новая копия со своими переменными, но как только она заканчивает свою работу, то память, занятая этими локальными переменными, освобождается, а полученные результаты передаются в точку вызова.

Некоторые задачи являются рекурсивными по своему определению.

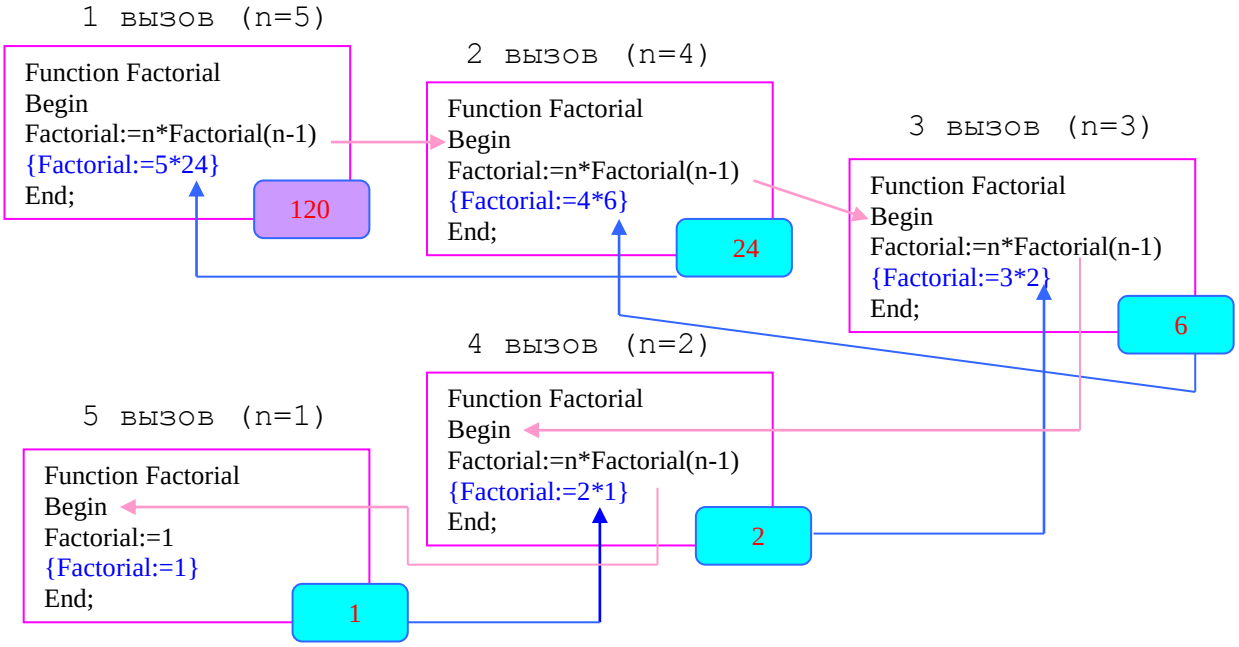
Задача 1

Вычисление факториала натурального числа.

$$N!=\begin{cases}1 & при \quad N=1\\ N*(N-1)!, & при \quad N>1\end{cases}$$

```
program rec1;
{-Вычисление факториала натурального числа }
var
  n:integer;
  f:longint;

function factorial(n: integer): longint;
begin
  if n=1 then factorial:=1
         else factorial:=n*factorial(n-1);
end;

begin
  writeln('Введите целое число: ');
  readln(n);
  f:=factorial(n);
  writeln(n:1,'! =',f);
  readln;
end.
```

Прорисуем вызовы функции factorial для n=5.



Первый вызов функции будет из основной программы. При каждом обращении к функции будет появляться свой набор локальных переменных со своими значениями. В данном случае для локальной переменной n выделяется память. После завершения работы эта часть памяти освобождается, и переменные удаляются.

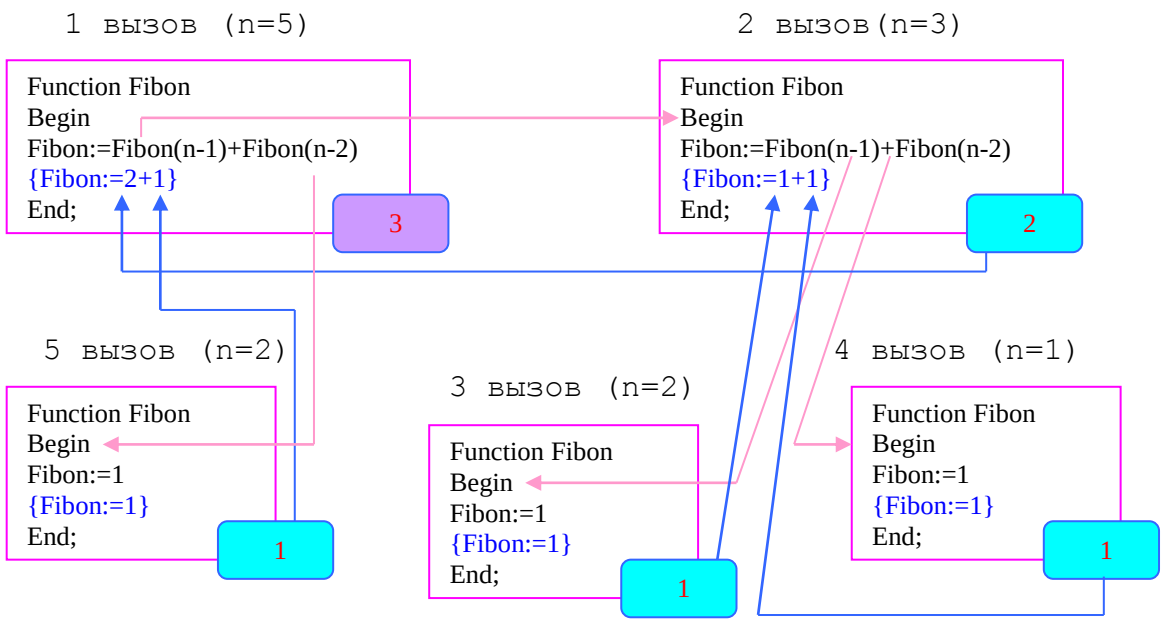
Задача 2

Найти первые N чисел Фибоначчи. Каждое число равно сумме двух предыдущих чисел при условии, что первые два равны 1 (1, 1, 2, 3, 5, 8, 13, 21,…), поэтому в общем виде N-е число можно определить так:

$$\Phi(n)=\begin{cases}1 & если \quad n=1 \ или \quad n=2\\ \Phi(n-1)+\Phi(n-2), & если \quad n>2\end{cases}$$

```
program rec2;
{-Числа Фибоначчи }
var
  n:integer;
  f:longint;

function fibon(n: integer): longint;
begin
  if (n=1) or (n=2) then fibon:=1
                    else fibon:=fibon(n-1)+fibon(n-2);
end;

begin
  writeln('Введите целое число: ');
  readln(n);
  f:=fibon(n);
  writeln('Число Фибоначчи = ',f);
  readln;
end.
```

Прорисуем вызовы функции fibon для n=5.



Задача 3

Составить программу вычисления значений функции Аккермана для неотрицательных чисел n и m, вводимых с клавиатуры.

$$A(n,m)=\begin{cases} m+1, & \textit{если} \quad n=0 \\ A(n-1,1), & \textit{если} \quad n\neq 0 \quad m=0 \\ A(n-1,A(n,m-1)), & \textit{если} \quad n>0 \quad m\geq 0 \end{cases}$$

```
program rec3;
{-Функция Аккермана }
var
  n,m:integer;
  A:longint;

function Akkerm(n,m: integer): longint;
begin
  if (n=0) then Akkerm:=m+1
           else if m=0 then Akkerm:=Akkerm(n-1,1)
                       else Akkerm:=Akkerm(n-1,Akkerm(n,m-1));
end;

begin
  writeln('Введите два целых неотрицательных числа (через пробел): ');
  readln(n,m);
  A:=Akkerm(n,m);
  writeln('Значение функции Аккермана = ',A);
  readln;
end.
```

Многие задачи не являются рекурсивными по определению, но некоторые из них можно свести к рекурсивным.

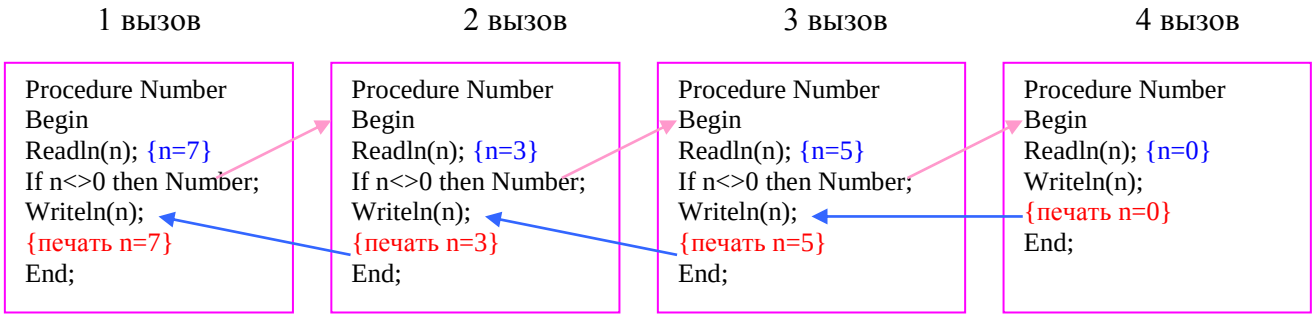
Задача 4

Составить программу ввода с клавиатуры последовательности чисел (окончание ввода - 0) и вывода ее на экран в обратном порядке.

```
program rec4;
{-Ввод и вывод последовательности чисел }

procedure number;
{-Рекурсивная процедура ввода и отображения числа}
var
  n: integer;
begin
  readln(n);
  if n<>0 then number;
  writeln(n);
end;

begin
  number;
  readln;
end.
```

После запуска программы на выполнение зададим последовательность чисел: 7 3 5 0

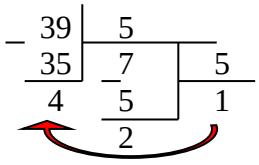
Прорисуем вызовы процедуры number для данного случая:



Задача 5

Составить программу для перевода числа из десятичной системы счисления в систему с основанием N, при условии 2≤N≤16. Значение N вводить с клавиатуры.

Для перевода числа из десятичной системы в систему с основанием N, необходимо делить данное число на N и фиксировать остатки от деления до тех пор, пока оно больше N-1. Результатом будет число, составленное из частного и остатков от деления. Например, перевести число 39 из десятичной системы счисления в систему с основанием 5:



$39_{10} = 124_5$

```
program rec5;
{-Перевод десятичного числа в другую систему счисления }
var
  n,k: integer;

procedure DecToN(k,n:integer);
begin
  if k>n-1 then DecToN(k div n,n);
  write(k mod n);
end;

begin
  writeln('Какое десятичное число перевести: ');
  readln(k);
  repeat
    writeln('В систему с каким основанием: ');
    readln(n);
  until (n>1) and (n<17);
  DecToN(k,n);
  readln;
end.
```

После запуска программы на выполнение зададим число в десятичной системе счисления (например, 39) и основание системы (5). Получим число 124, первая цифра (1) которого будет

выведена на экран из последнего вызова проседуры DecToN, следующая (2) из предпоследнего, последняя (4) – из первого.

Если нельзя извлечь рекурсию из условия задачи, то можно расширить задачу, обобщить ее. Удачное обобщение дает возможность увидеть рекурсию. После этого следует вернуться к частному случаю и решить исходную задачу.

Задача 6

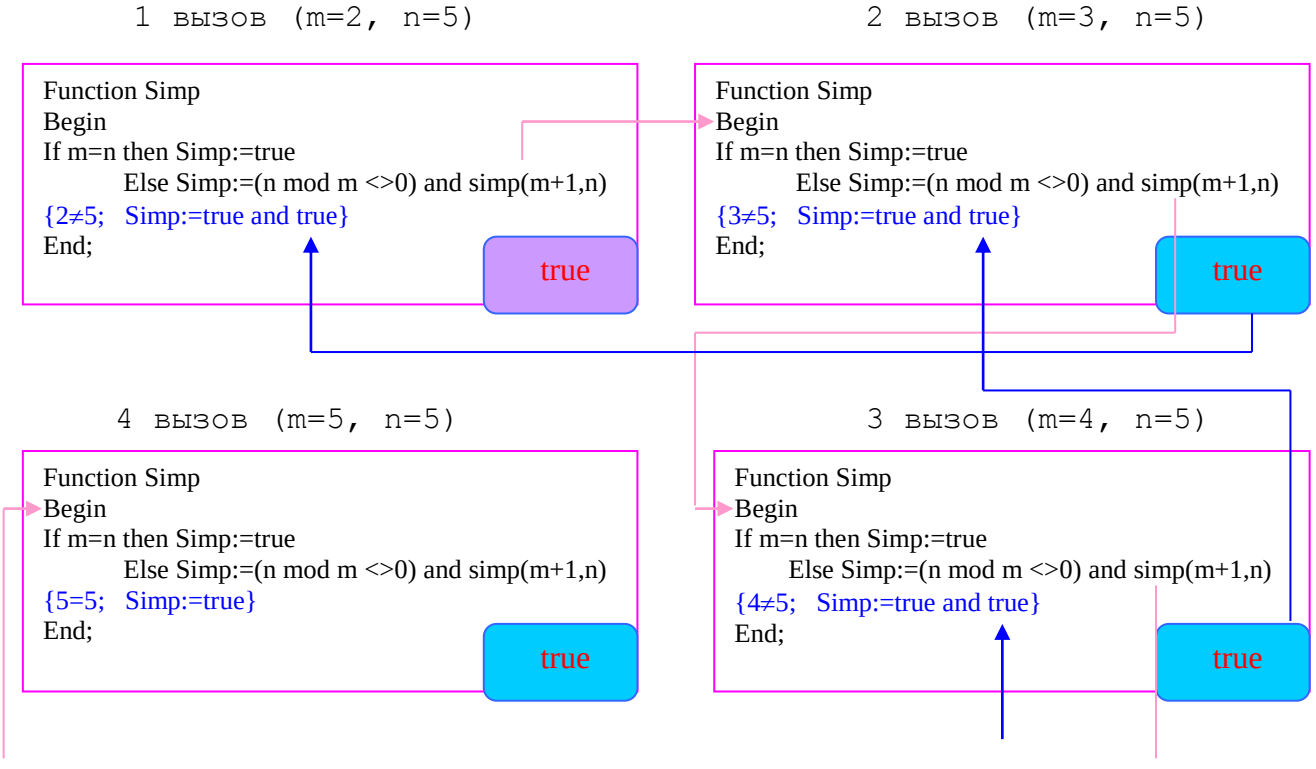
Определить, является ли данное натуральное число простым.

Обобщим задачу: Верно ли, что заданное натуральное число N не делится ни на одно число, большее или равное M, но меньшее N. Истина может быть в двух случаях:

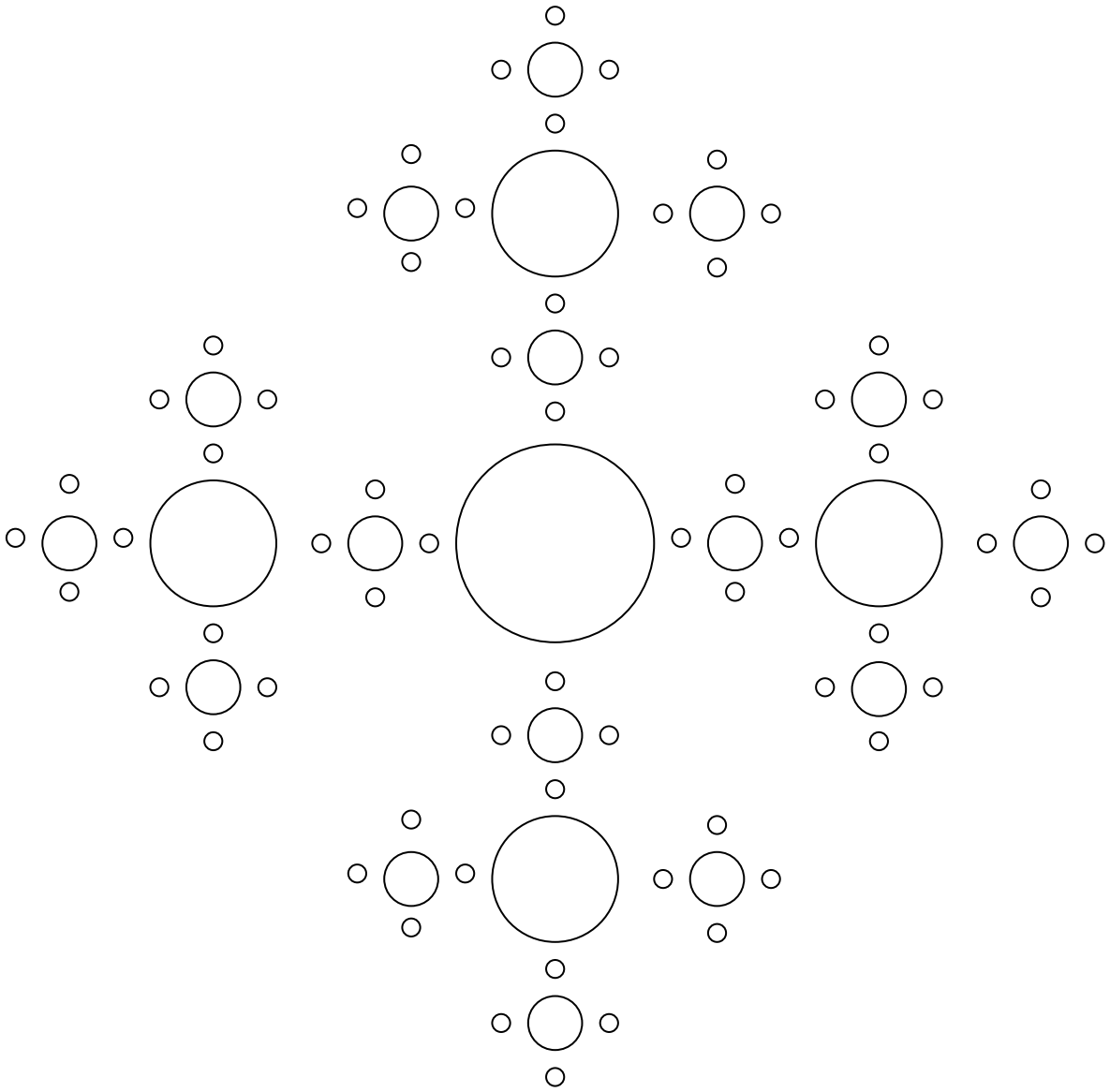
- если M=N;
- если N не делится на M и истина для чисел M+1 и N.

Вернемся к исходной задаче. Для этого в качестве значения M возьмем 2.

```
program rec6;
{ Является ли данное натуральное число простым? }

var
  m,n: integer;
  s:boolean;

function simp(m,n:integer): boolean;
begin
  if m=n then simp:=true
  else simp:=(n mod m<>0) and simp(m+1,n);
end;

begin
  write('Какое число: '); readln(n);
  s:=simp(2,n);
  if s then writeln('Число является простым')
  else writeln('Число не является простым');
  readln;
end.
```



Задача 7
Написать программу построения изображения:



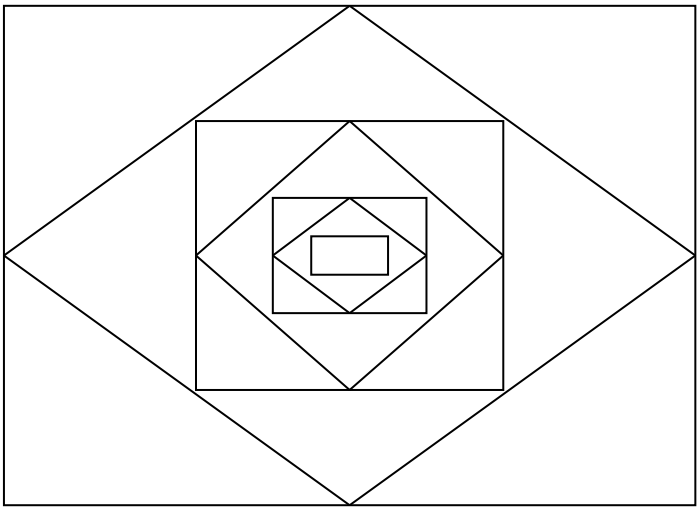
На рисунке большая окружность окружена четырьмя окружностями поменьше, каждая из которых, в свою очередь, окружена четырьмя окружностями с еще меньшими радиусами. Всего изображено четыре уровня окружностей. Для построения каждого уровня изображения, который включает одну окружность с четырьмя окружностями поменьше, опишем процедуру Picture.

```
program rec7;
{-Построение окружностей }
```

```
uses Graph, Crt;
var
  x,y,n,r,r1,cd,gm: integer;
  k1,k2:real;

procedure Picture(x,y,r,r1,n:integer);
var
  x1,y1,i:integer;
begin
  if n>0 then
         begin
           circle(x,y,r);
           r1:=trunc(r*k2);
           for i:=1 to 4 do
             begin
               x1:=trunc(x+r1*cos(pi/2*i)); { Координаты центра }
               y1:=trunc(y+r1*sin(pi/2*i)); {i-й окружности }
               picture(x1,y1,trunc(r*k1),r1,n-1);
             end;
         end;
end;

begin
  write('Введите количество уровней: ');
  readln(n);
  x:=300; y:=200; { Координаты центра большой окружности }
  writeln('Введите радиус первой окружности: ');
  readln(r);
  k1:=0.3; { Коэффициент для вычисления радиуса окружности след. уровня }
  k2:=2;   { Коэффициент для вычисления радиуса орбиты }
  cd:=detect; gm:=1;
  InitGraph(cd,gm,'\tp');
  picture(x,y,r,r1,n);
  readln;
  CloseGraph;
end.
```

Задача 8

Написать программу построения изображения:



```
program rec8;
{ Построение четырехугольников }

uses Graph, Crt;
var
  a1,a2,a3,a4,b1,b2,b3,b4,cd,gm,n: integer;

procedure Picture(a1,b1,a2,b2,a3,b3,a4,b4,n:integer);
var
  x1,y1,x2,y2,x3,y3,x4,y4,i: integer;
begin
  if n>0 then
     begin
       { Построение четырехугольника }
       line(a1,b1,a2,b2);
       line(a2,b2,a3,b3);
       line(a3,b3,a4,b4);
       line(a4,b4,a1,b1);
       { Вычисление координат вершин следующего четырехугольника }
       x1:=trunc((a1+a2)/2);     y1:=trunc((b1+b2)/2);
       x2:=trunc((a2+a3)/2);     y2:=trunc((b2+b3)/2);
       x3:=trunc((a3+a4)/2);     y3:=trunc((b3+b4)/2);
       x4:=trunc((a1+a4)/2);     y4:=trunc((b4+b1)/2);
       Picture(x1,y1,x2,y2,x3,y3,x4,y4,n-1);
     end;
end;

begin
  write('Введите количество уровней: '); readln(n);
  { Координаты вершин внешнего четырехугольника }
  a1:=0;b1:=0;a2:=400;b2:=0;a3:=400;b3:=400;a4:=0;b4:=400;
  cd:=detect; gm:=1;
  InitGraph(cd,gm,'\tp');
  Picture(a1,b1,a2,b2,a3,b3,a4,b4,n);
  readln;
  CloseGraph;
end.
```

Задача 9

Обобщить предыдущую задачу. Составить программу для построения Р-угольников.

```
program rec9;
{ Построение Р-угольников }

uses Graph, Crt;
const
  k=30;
type
  mas=array[1..k] of integer;
```

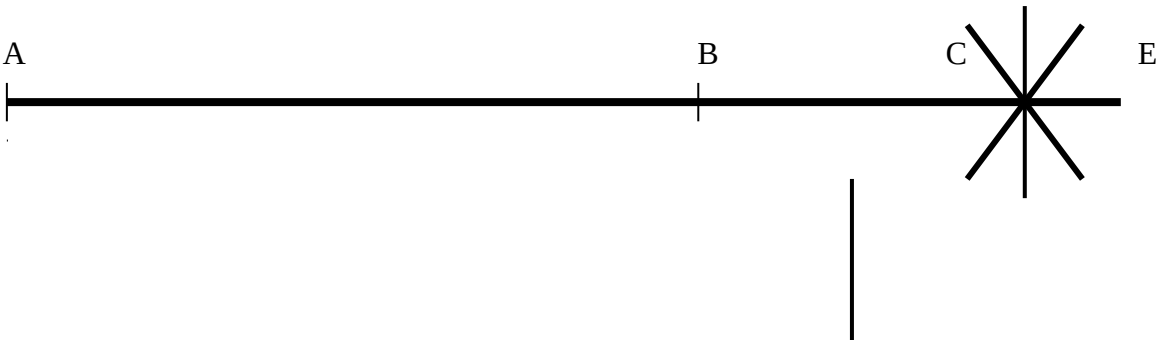
```
var
  cd,gm,n,p,i: integer;
  a,b: mas;

procedure Picture(x,y,n:integer);
var
  x,y:mas;
  i:integer;
begin
  if n>0 then
     begin
       for i:=1 to p do
         begin
           line(320,200,a[i],b[i]);
           a[i]:=trunc(x+l*cos(2*pi/p*(i-1)));
           b[i]:=trunc(y+l*sin(2*pi/p*(i-1)));
           for i:=1 to p-1 do
             begin
               x[i]:=trunc((a[i]+a[i+1])/2);
               y[i]:=trunc((b[i]+b[i+1])/2);
             end;
           x[p]:=trunc((a[p]+a[1])/2);
           y[p]:=trunc((b[p]+b[1])/2);
           Picture(x,y,n-1);
         end;
     end;
end;

begin
  write('Введите количество уровней: '); readln(n);
  write('Введите количество вершин: '); readln(p);
  for i:=1 to p do
    begin
      a[i]:=trunc(320+100*cos(2*pi/p*(i-1)));
      b[i]:=trunc(200+100*sin(2*pi/p*(i-1)));
    end;
  cd:=detect; gm:=1;
  InitGraph(cd,gm,'\tp');
  Picture(a,b,n);
  readln;
  CloseGraph;
end.
```

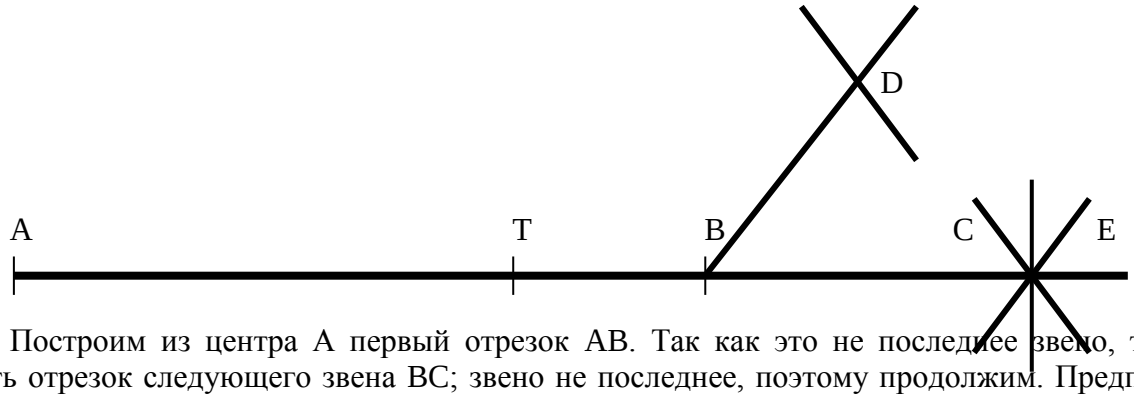
Задача 10

Написать программу построения изображения снежинки, число звеньев и количество ветвей которой может меняться пользователем.

Центр снежинки имеет координаты (x,y). Количество ветвей P вводится с клавиатуры, а число звеньев n задается в виде константы. Длина каждого звена уменьшается в 4 раза по отношению к предыдущему. Рассчитанные длины n звеньев сохраняются в массиве.

Предлагается следующая логика решения задачи. Пусть точка A – центр снежинки.





Построим из центра A первый отрезок AB. Так как это не последнее звено, то будем рисовать отрезок следующего звена BC; звено не последнее, поэтому продолжим. Предположим, что нарисовали CE – ветвь самой маленькой снежинки. Нарисуем все ветви этой снежинки и вернемся в точку B. Так как это не последняя ветвь данной снежинки, то нарисуем следующую ветвь – отрезок BD, а затем снова полностью самую маленькую снежинку. Пусть значение *к* равно текущему номеру звена снежинки (в начальный момент оно равно *n*). Тогда при переходе от точек C,D к точке B мы должны "вспомнить" координаты точки B и номер ветви, рисуемой из точки B. При переходе от точки B к точке A мы должны "вспомнить" координаты точки A и номер ветви снежинки, рисуемой из точки A. Эта логика реализуется с помощью рекурсии.

```
program rec10;
{ Построение снежинки }

uses Graph, Crt;
const
  n=5; { Число звеньев снежинки }
var
  p,x,y,i,cd,gm: integer;
  l: array[1..5] of integer; { Длины звеньев }

procedure Picture(x,y,k:integer);
var
  x1,y1,i:integer;
begin
  if k>0 then
     begin
       for i:=1 to p do
         begin
           x1:=trunc(x+l[k]*cos(2*pi/p*(i-1))); { Координаты конца }
           y1:=trunc(y+l[k]*sin(2*pi/p*(i-1))); { очередного звена }
           line(x,y,x1,y1); {построение звена}
           Picture(x1,y1,k-1);
         end;
     end;
end;

begin
  write('Введите количество ветвей (3..14): '); readln(p);
  x:=320; y:=200; { Центр снежинки }
  l[n]:=100; { Длина первого звена, n – количество звеньев }
  for i:=n-1 downto 1 do   { Длина каждого звена уменьшается в }
    l[i]:=trunc(l[i+1]/4); { 4 раза по отношению к предыдущему }
  cd:=detect; gm:=1;
  InitGraph(cd,gm,'\tp');
  Picture(x,y,n);
  readln;
  CloseGraph;
```

```
end.
```

Многие комбинаторные задачи можно представить как поиск элементов конечного множества, удовлетворяющих определенным условиям. Очевидно, всегда есть "лобовое" решение, состоящее в переборе всех элементов этого множества с проверкой соответствующих условий. Однако, даже для простых задач такой перебор потребует огромных затрат. Для решения следующих задач используем перебор с возвратом.

Задача 11

Задача о шахматном коне. Существуют способы обойти шахматным конем шахматную доску, побывав на каждом поле по одному разу. Составить программу нахождения всех возможных способов обхода доски.

Пусть шахматная доска имеет m горизонталей и n вертикалей. Находясь на конкретном поле [i,j] , конь может сделать один из восьми вариантов ходов:

j

i

	8		1	
7				2
		**		
6				3
	5		4	

В массивах di[1..8], dj[1..8] будем хранить приращения к значениям координат, в сумме с которыми i,j дают возможные координаты нового хода коня.

Выполнив очередной ход конь может попасть:

- за пределы доски;
- на поле, на котором уже был;
- на поле, на котором не был.

Учитывая это, создадим массив A: array[-1..m+2,-1..n+2] of integer, где

$$A[i,j]=\begin{cases}0 & \textit{если поле } [i,j] \textit{ конем не посещалось}\\ -1 & \textit{если поле } [i,j] \textit{ находится за пределами доски}\\ t & \textit{если поле } [i,j] \textit{ посещалось на ходе } t\end{cases}$$

Поиск варианта обхода начинается с левой верхней клетки, то есть с A[1,1]. Выполнить ход можно только в том случае, если очередная клетка свободна (A[i,j]=0). Сделать ход конем означает изменить соответствующий элемент массива A на значение t. Если очередная клетка занята или находится за пределами доски, то нужно попытаться из этой же клетки сделать другой ход – взять следующие значения из массивов di, dj. Если же ни один из 8 вариантов ходов из текущего положения не приведет к выполнению хода, то мы должны вернуться на 1 или серию ходов назад и попытаться найти новый вариант обхода. Для этого надо взять следующие значения из массивов di, dj. Поиск обхода завершается, если t=n*m. Реализовать такую логику поиска обхода удобно с помощью рекурсии.

```
program rec11;
{ Обход шахматным конем шахматной доски }
const
```

```
  n=5;
  m=5;
  di: array[1..8] of integer = (-2,-1,1,2,2,1,-1,-2);
  dj: array[1..8] of integer = (1,2,2,1,-1,-2,-2,-1);
var
  a: array[-1..n+2,-1..m+2] of integer;
  i,j,l,p: integer;
  q: boolean;



procedure Print;
{ Вывод содержимого двумерного массива }
begin
  for i:=1 to n do
  begin
    for j:=1 to m do write(a[i,j]:3);
    writeln;
  end;
  p:=p+1;
  writeln(p); { Номер обхода }
end;

procedure Rec (i,j,t: integer);
{ Рекурсивная процедура поиска обхода }
var
  k: integer;
begin
  if(a[i,j]<>0) then Exit { Выход за пределы доски или клетка занята }
  else
     begin
       a[i,j]:=t; {ход с номером t}
       if t=n*m then begin Print; q:=true end { Выполнен полный обход }
       else
         for k:=1 to 8 do Rec(i+di[k],j+dj[k],t+1); { Выбор нового хода }
         a[i,j]:=0;
     end;
end;

begin
  p:=0; { Счетчик полных обходов - решений }
  q:=false; { Признак наличия решений }
  { Заполнение двумерного массива }
  for i:=-1 to n+2 do
    for j:=-1 to m+2 do
      if (i<1) or (i>n) or (j<1) or (j>m) then
         a[i,j]:=-1;
  for i:=1 to n do
    for j:=1 to m do
      Rec(i,j,1);
  if q then
     write('Число решений: ',p)
  else
     writeln(^G'Обход выполнить невозможно!');
  readln;
end.
```

Задача 12

Задача о коне Аттилы. ("Трава не растет там, где ступил мой конь!"). На шахматной доске стоят белый конь и черный король. Некоторые поля доски считаются "горящими". Конь должен дойти до неприятельского короля, повергнуть его и вернуться на исходное место. При этом ему запрещено становиться как на "горящие" поля, так и на поля, которые уже пройдены.

```
program rec12;
{$M 65200,0,655360 }  { Увеличение размера стека }
const
  n=8; { Размер доски }
  di: array[1..8] of integer = (-2,-1,1,2,2,1,-1,-2);
  dj: array[1..8] of integer = (1,2,2,1,-1,-2,-2,-1);
var
  a: array[-1..n+2,-1..n+2] of integer;
  i,j,l,w,ik,jk,il,jl: integer;
  r: boolean;

procedure Print;
{ Вывод результата обхода }
begin
  writeln('Обход:');
  for i:=1 to n do
    begin
      for j:=1 to n do
        if (i=ik) and (j=jk) then
           write(' K',a[i,j]:2)
        else
           if (i=il) and (j=jl) then
              write(' A',a[i,j]:2)
           else
              write(a[i,j]:4);
      writeln;
    end;
  readln; { Задерживает результат на экране для просмотра }
end;

procedure Rec(i,j,t:integer);
{ Поиск обхода }
var
  k:integer;
begin
  if not r then
     begin
       if (i=il) and (j=jl) and (a[ik,jk]<>0) then
          begin
            r:=true;
            Print;
          end;
   if a[i,j]<>0 then Exit
   else
      begin
        a[i,j]:=t;
        for k:=1 to 8 do
            Rec(i+di[k],j+dj[k],t+1);
        a[i,j]:=0;
      end;
   end;
end;
```

```
begin
  Writeln ('Задача о коне Аттилы');
  Randomize;
  { a[i,j]=-1, если клетка за пределами доски или горящее поле }
  { a[i,j]= 0, если свободная клетка }
  for i:=-1 to n+2 do
    for j:=-1 to n+2 do
      if (i<1) or (i>n) or (j<1) or (j>n) then
         a[i,j]:=-1
      else
         a[i,j]:=0;


  { Создание "горящих" полей }
  for w:=1 to 5 do
    begin
      i:=trunc(random(8));
      j:=trunc(random(8));
      a[i,j]:=-1;
    end;

  { ik,jk – координаты клетки короля выбираются случайным образом }
  repeat
    i:=trunc(random(8));
    j:=trunc(random(8));
  until a[i,j]=0;
  ik:=i; jk:=j; writeln('Король ',ik:3,jk:3);

  { il,jl – координаты клетки короля }
  repeat
    i:=trunc(random(8));
    j:=trunc(random(8));
  until (a[i,j]=0) and not((i=ik)and(j=jk));

  il:=i; jl:=j; writeln('Конь Аттилы ',il:3,jl:3);

  r:=false; { Признак наличия решений }
  Rec(il,jl,1); { Конь Аттилы начинает обход }
end.
```

Итак, мы рассмотрели способы решения нескольких задач, являющихся рекурсивными по определению, и задач, которые таковыми не являются, но могут быть оптимально решены с использованием рекурсивного метода. Данный реферат был составлен по материалам занятий в гимназических классах и предлагается в качестве пособия для подготовки к решению олимпиадных задач.