ПОНЯТИЕ «РЕКУРСИЯ»

Задача нахождения НОД(N, M) при различных значениях N и M .

Пример программы PASCAL, реализующий данную задачу:

1 способ	2 способ

program nod_1;
Function nod(x,y:integer):integer;
begin
while y<>0 do
if x>=y then x:=x mod y
else begin x:=x+y;
y:=x-y;
x:=x-y;
end;
nod:=x;
end;
var M,N:integer;
begin
readln(N,M); writeln(nod(N,M));
readln;
end
	program nod_2;
	function nod(x,y:integer):integer;
	begin
	if y=0 then nod:=x
	else nod:=nod(y,x mod y);
	end;
	begin writeln(nod(12,18));
	writeln(nod(18,12));
	readln;
	end.

Подзадачи:

$$nod(x,y)=\begin{cases} x & , если \quad y=0; \\ nod\big(y,\mathrm{mod}(x,y)\big) & , если \quad y\neq 0. \end{cases}$$

1. НОД(X,Y) =X, если Y=0 .
2. НОД(X,Y) = НОД(Y, X mod Y) при Y≠0.

При использовании второго способа реализации алгоритма на Паскале реализована рекурсивная функция.

В общем случае на рекурсию следует смотреть как на введение в определение объекта ссылку на сам объект или как на прием сведения решения некоторой задачи к решению “более простой” задачи такого же класса. В программировании это выражается в построении программ (процедур и функций), которые при выполнении обращаются сами к себе непосредственно или через цепочку других программ.

Функция называется рекурсивной, если в её определении содержится вызов этой же функции.

Решение задачи рекурсивным методом	
Этапы	НОД(N, M) при различных значениях N и M
Параметризация - выявление совокупности исходных величин, определяющих постановку и решение задачи.	$nod(x,y):=\begin{cases} x \ \ if \quad y=0 \\ nod\big(y,\mathrm{mod}(x,y)\big) \end{cases}$
выделение базы - поиск подзадач, которые решаются без рекурсивного вызова.	$nod(x,y):=\begin{cases} x \ \ if \quad y=0 \end{cases}$

Декомпозиция - процесс последовательного разложения исходной задачи на серию более простых подзадач, аналогичных исходной задаче, каждая близка к тривиальному случаю, чем предыдущая. **проведение отложенных вычислений.**	$nod(12,18) := \left\| \begin{array}{l} x \ \ if \ \ y = 0 \\ nod(18, 12 \bmod 18) := \left\| \begin{array}{l} x \ if \ \ y = 0 \\ nod(12, 18 \bmod 12) := \left\| \begin{array}{l} x \ if \ \ y = 0 \\ nod(6, 12 \bmod 6) := \left\| 6 \ if \ \ 0 = 0 \right. \end{array} \right. \end{array} \right. \end{array} \right.$

$$nod(18,12) := \left\| \begin{array}{l} x \ \ if \ \ y = 0 \\ nod(12, 18 \bmod 12) = \left\| \begin{array}{l} x \ if \ \ y = 0 \\ nod(6, 12 \bmod 6) = \left\| 6 \ if \ \ 0 = 0 \right. \end{array} \right. \end{array} \right.$$

Понятия и термины, связанные с рекурсией

№	Понятие, Термин	Неформальное определение, Пояснение
1.	Рекурсия	1. Введение в определение объекта ссылку на сам объект. 2. Прием сведения решения некоторой задачи к решению серии задач, подобных исходной.
2.	Рекурсивный алгоритм (процедура, функция)	1. Алгоритм (функция, процедура) называется рекурсивным, если в его определении содержится прямой или косвенный вызов этого же алгоритма. 2. Рекурсивная функция – одно из математических уточнений интуитивного понятия вычислимой функции.
3.	Рекуррентное соотношение (рекуррентная формула)	Формула вида $a_{n+p}=F(a_n, a_{n+1},\ldots, a_{n+p-1})$ $(p\geq 1)$, позволяющая вычислять любой член бесконечной последовательности $a_1, a_2,\ldots$, если заданы её первые p членов. Определяемая рекуррентной формулой последовательность называется возвратной.
4.	Глубина рекурсивных вызовов	Количество элементов полной рекурсивной траектории в пространстве параметров.
5.	Рекурсивные вычисления Прямой и обратный ход вычислений	Прямой ход соответствует совокупности всех предвычислений, реализуемых до входа рекурсивной траектории в базу. Обратный ход – совокупности отложенных вычислений, производимым после встречи с индикатором завершения рекурсивных вызовов.
6.	Рекурсивный Стек	Область памяти, в которую заносятся значения всех локальных переменных алгоритма (программы) в момент рекурсивного обращения. Каждое такое обращение формирует один слой данных стека. При завершении вычислений по конкретному обращению α из стека считывается соответствующий ему слой данных, и локальные переменные восстанавливаются, снова принимая значения, которые они имели в момент обращения α.
7.	Адаптивный рекурсивный	Алгоритм, который благодаря рекурсивности учитывает те или иные индивидуальные характеристики решаемой задачи из области своего

№	Понятие, Термин	Неформальное определение, Пояснение
	алгоритм	определения.

Примеры задач

Задача 1(4 способа). Составить программу-функцию вычисления факториала целого неотрицательного числа n.

Способ 1

Решение. Для целых неотрицательных чисел n факториал n обозначается через n! и определяется так:

$$n!=\begin{cases}1 & , \textit{если} \quad n=0 \quad \text{или} \quad n=1;\\ 1\cdot 2\cdot \ldots \cdot n & , \textit{если} \quad n\geq 2.\end{cases}$$

Случаи, для которых задача решается без рекурсивных вызовов, очевидны: f(0)=1, f(1)=1. Они и составляют базу рекурсии. Декомпозиция по параметру n реализуется по формуле: f(n) = n·f(n–1) (n = 1, 2, …). Поэтому вычисления f(n) можно организовать так:

```
program project1;
 var x:byte;
function f(n:byte):longint;
begin if n<=1then f:=1
else f:=n*f(n-1);
end;
begin  readln(x);writeln(f(x));
readln;
end.
```

Контрольные примеры.

$f(0)=1 \qquad f(5)=120 \qquad f(10)=3628800.$

Понять процесс реализации рекурсивных вызовов, то есть декомпозицию, и возвратов управления при организации отложенных вычислений f(n) можно из схемы (n = 3). Там около стрелок в круглых скобках жирными цифрами указаны номера последовательных шагов вычислений: (**1**), (**2**), (**3**) – декомпозиция; (**4**), (**5**), (**6**) – отложенные вычисления.

$$f(3) = 6$$
$$(\mathbf{6})\uparrow \quad \downarrow(\mathbf{1})$$
$$f(3):=\left|\begin{array}{l}1 \ \textit{if} \ \ 3\leq 1\\ 3\cdot f(2)\end{array}\right.$$
$$(\mathbf{5})\uparrow \quad \downarrow(\mathbf{2})$$
$$f(2):=\left|\begin{array}{l}1 \ \textit{if} \ \ 2\leq 1\\ 2\cdot f(1)\end{array}\right.$$
$$(\mathbf{4})\uparrow \quad \downarrow(\mathbf{3})$$
$$f(1):=\left|\begin{array}{l}1 \ \textit{if} \ \ 1\leq 1\\ 1\cdot f(0)\end{array}\right.$$

Способ2.Для решения задачи иногда удается построить несколько различных рекурсивных алгоритмов. Например, функцию f(n) можно было бы определить и так.

$$f(n):=\left|\begin{array}{l}1 \ \textit{if} \ \ n\leq 1\\ n\cdot(n-1)\cdot f(n-2)\end{array}\right.$$

```
program project2;
 var x:byte;
function f(n:byte):longint;
begin if n<=1then f:=1
else f:=n*(n-1)*f(n-2);
end;
begin  readln(x);writeln(f(x));
readln;
end.
```

Рекурсивное решение очень простое, но оно неоптимально по быстродействию: компьютер выполняет лишнюю работу, повторно вычисляя уже найденные значения.

Какой же выход? Напрашивается такое решение — хранить все предыдущие числа в массиве.

Задача 2. Пусть a и n – целые неотрицательные числа. Составить программу-функцию возвращающую величину a^n.

Способ1. Функция power(a,n) дает решение задачи за n рекурсивных вызовов:

$$power(a,n) := \begin{cases} 1 \ if \ \ n = 0 \\ power(a, n-1) \cdot a \end{cases}$$

```
program stepen;
 var x,y:byte;
function power(a,n:byte):longint;
begin  if n=0 then power:=1
else power:=a*power(a,n-1);
end;
begin readln(x,y); writeln(power(x,y)); readln; end.
```

Способ2 Можно составить программу, при исполнении которой значения переменных a и n не меняются, а значение некоторой другой переменной (например, b) становится равным a в степени n. Число действий (выполняемых операторов присваивания) будет порядка log2 n (то есть не превосходило бы C * log2 n для некоторой константы C; log2 n — это степень, в которую нужно возвести 2, чтобы получить n).

Каждый второй раз (не реже) будет выполняться первый вариант оператора выбора (если k нечетно, то после вычитания единицы становится четным), так что за два цикла величина k уменьшается по крайней мере вдвое.

Алгоритм анализа показателя степени	
Без рекурсии	С рекурсией

var n,a,k,b,c:longint;
begin
readln(a,n);
k := n; b := 1; c := a;
while k <> 0 do
begin
if k mod 2 = 0 then
begin k := k div 2;
c := c * c
end
else begin k := k - 1;
b := b * c
end ;
end;
writeln(b); readln;
end.
	var n,a:longint;
	function f(x,y:longint):longint;
	var z:longint;
	begin
	if y=0 then f:=1
	else if y mod 2 = 0 then f:=f(x*x,y div 2)
	else f:=f(x,y-1)*x;
	end;
	begin
	readln(a,n);
	writeln(f(a,n)); readln;
	end.

Задача 3. Составить программу-функцию вычисления n–го числа Фибоначчи, исходя из рекуррентного определения этих чисел:

f(0)=f(1)=1, f(n)=f(n–1)+f(n–2) (n=2,3,…).(1)

Способ1.

Решение. Наличие рекуррентного соотношения вида (1) сразу же определяет и базу рекурсии, и способ декомпозиции. Программа-функция fib(n) написана строго в соответствии с определением (1).

$$fib(n) := \begin{cases} 1 \ if \ \ n \le 1 \\ fib(n-1) + fib(n-2) \end{cases}$$

```
program fibonazzi;
var n:byte;
function fib(x:integer):integer;
begin if (x=0) or (x=1) then fib:=1
else fib:=fib(x-1)+fib(x-2);
end;
begin readln(n); writeln(fib(n)); readln; end.
```

Контрольные примеры.

$fib(1) = 0 \qquad fib(5) = 8 \qquad fib(10) = 89.$

Задача 4. Составить программу-функцию вычисления биномиальных коэффициентов C(n,m), где n m – целые и 0≤m≤n.

Решение. Известно, что

$$C(n,m) = \frac{n!}{m!\cdot(n-m)!} = \frac{n}{m}\cdot C(n-1,m-1).$$

Отсюда и вытекает справедливость следующего рекурсивного определения C(n,m):

$$C(n,m) := \begin{cases} 1 \ if \ \ m = 0 \\ \frac{n}{m}\cdot C(n-1,\ m-1) \end{cases}$$

```
program kombin;
 var x,y:integer;
function c(n,m:integer):real;
begin if m=0 then c:=1
else c:=n/m*c(n-1,m-1);
  end;
begin  readln(x,y);
writeln(c(x,y):5:2);
readln;
end.
```

Задача5. Способ вычисления биномиальных коэффициентов.

Рекурсивная программа-функция tripas(n) вычисляет треугольник Паскаля, то есть значения величин C(i,j) для (0≤i≤n , 0≤j≤i), исходя из формул непосредственно определяющих и декомпозицию и базу:

$C(i,\ j) = C(i-1,\ j-1) + C(i-1,\ j);$

$C(i,0) = 1,\ C(i,i) = 1; i < j\ C(i,\ j) = 0$

```
program tr_P;
const n=4;
var x,y:integer; a:array[1..n,1..n]of integer;
 function c(i,j:integer):integer;
 begin  if (j=1) or (i=j) then c:=1  else
 if i<j then c:=0
```

```
 else c:=c(i-1,j-1)+c(i-1,j);
 end;
begin
for x:=1 to n do for y:=1 to n do a[x,y]:=c(x,y);
readln;
end.
```

Контрольный пример для n=4.

$$a(4) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Задача 6. Количество делителей. Составить программу-функцию подсчета для натурального числа n количества всех его делителей.

Решение. Перейдем к более общей задаче. Подсчитаем для натурального числа n количества всех его делителей, меньших или равных заданному натуральному числу x. Пусть dn(n) и dnx(n,x) – соответственно функции для решения исходной и обобщенной задач. Очевидно, что dn(n)=dnx(n,n).

Рекурсивную функцию dnx(n,x), по которой последовательно подвергаются испытанию на делители n все числа от 1 до x включительно, можно определить так:

$$dnx(n,x) := \begin{vmatrix} 1 \ if \ \ x = 1 \\ dnx(n,x-1) + if\left(\mathrm{mod}(n,x) = 0,1,0\right), \end{vmatrix}$$

$$dn(n) := dnx(n,n).$$

```
program kol_vo_del;
var z,a:integer;
function del(n,x:integer):integer;
begin if x=1 then del:=1
else if n mod x=0 then del:=1+del(n,x-1) else del:=del(n,x-1);
end;
begin readln(z,a);writeln(del(z,a));
readln;
end.
```

Контрольные примеры.

$$dnx(12,4) = 4, \qquad dnx(12,6) = 5, \qquad dn(12) = 6.$$

Задача 7. Составить программу-функцию проверяющую, является ли заданное натурально число n простым?

Решение. Пусть рекурсивная функция isprim(n) является решением задачи и

$$isprim(n) := \begin{cases} 1, & \textit{если } \text{n - простое}; \\ 0, & \textit{если } \text{n - не простое}. \end{cases}$$

Пусть a, b, n – натуральные числа и 2≤a≤b≤n. Верно ли, что заданное n не делится ни на одно целое из отрезка [a, b]? Пусть эту задачу решает функция

$$nodiv(n,a,b) := \begin{cases} 1, & \textit{если } \text{n } \textit{не} \text{ имеет целых делителей из [a, b]}; \\ 0, & \textit{если } \text{n имеет целые делители из [a, b]}. \end{cases}$$

Рекурсивный вариант реализации функции. Проверке на делитель n последовательно подвергаются числа: a, b, a+1, b–1,

$$nodiv(n,a,b) := \left| \begin{array}{l} 1 \;\; if \;\; a > b \\ \\ \left| \begin{array}{l} 0 \;\; if \;\; (\mathrm{mod}(n,a) = 0) + (\mathrm{mod}(n,b) = 0) \\ nodiv(n, a+1, b-1) \end{array} \right. \end{array} \right. .$$

```
program Prostoe;
 var x:integer;
function p(n,a,b:integer):integer;
begin if (a=b) or (n=2) then p:=1 else
if (n mod a=0) or (n mod b=0) or(n<2) then p:=0
else p:=p(n,a+1,b-1);
end;
begin  repeat readln(X);until x>1;
if p(x,2,x-1)=0
then writeln(x,'-sostawnoe')
else writeln(x,'-prostoe');
readln;
end.
```

Далее, натуральное число n≥2 является простым, если оно не имеет делителей на отрезке $\left[2, floor\left(\sqrt{n+1}\right)\right]$, поэтому характеристическая функция isprim(n) через функцию nodiv(n,a,b) может быть выражена так:

$$isprim(n) := nodiv\left(n, 2, floor\left(\sqrt{n+1}\right)\right).$$

Задача 8. Составить программу-функцию подсчета количества ξ(m) разбиений натурального числа m, то есть его представления в виде суммы натуральных чисел.

Решение. Пусть, например, m=6. Тогда разбиениями m являются его представления в виде:

6;
5+1;
4+2, 4+1+1;
3+3, 3+2+1, 3+1+1+1;
2+2+2, 2+2+1+1, 2+1+1+1+1;
1+1+1+1+1+1;

Таким образом, ξ(m)=11 и понятно, что простым перебором возможных случаев уже при m>10 справиться с задачей достаточно сложно.

Для решения исходной задачи перейдем к рассмотрению обобщенной задачи. Составить программу-функцию подсчета количества P(m,n) разбиений натурального числа m со слагаемыми, не превосходящими n. Ясно, что ξ(m)=P(m,m). Поэтому, достаточно научиться вычислять значения функции P(m,n). Но для неё нетрудно выделить рекурсивную базу и указать правило декомпозиции. Сделать это можно исходя из следующих вполне прозрачных свойств этой функции.

1. P(m,1)=1 – существует только одно разбиение m, в котором слагаемые не превосходят единицы, а именно: m=1+1+…+1.
2. P(1,n)=1 – число единица имеет только одно представление при любом n.
3. P(m,n)=P(m,m) при n>m – слагаемые, большие m в разбиениях отсутствуют.
4. P(m,m)=P(m,m−1)+1 – существует лишь одно разбиение со слагаемым равным m. Все иные разбиения имеют слагаемые не превосходящие m−1.
5. P(m,n)=P(m,n−1)+P(m−n,n) (n<m). Обоснование этого соотношения проводится так. Все разбиения m на сумму слагаемых, не превосходящих n можно разбить на два непересекающихся класса: суммы, не содержащие n в качестве слагаемого и суммы,

содержащие такое n. Количество элементов первого класса равно P(m,n–1), а количество элементов второго класса подсчитаем так. Без учета слагаемого n суммы элементов второго класса равны m–n. Значит их общее количество равно P(m–n,n) и, следовательно, общее количество элементов второго класса также равно этой величине. Тем самым свойство 5 установлено.

Первые два свойства определяют базу рекурсии, а три следующие задают декомпозицию. Строго в соответствии с этими утверждениями и составлена рекурсивная программа-функция deco(n,m) для вычисления величины P(m,n).

$$deco(m,n) := \left| \begin{array}{l} 1 \quad if \quad (n=1)+(m=1) \\ \left| \begin{array}{l} deco(m,m) \quad if \quad n>m \\ \left| \begin{array}{l} deco(m,m-1)+1 \quad if \quad m=n \\ deco(m,n-1)+deco(m-n,n) \end{array} \right. \end{array} \right. \end{array} \right. ,$$

$\xi(m) := P(m,m)$.

```
program P6_5plus1;
  function deco(m,n:integer):integer;
  begin if (n=1) or(m=1)then deco:=1
  else if n>m then deco:=deco(m,m)
  else if m=n then deco:=deco(m,m-1)+1
  else deco:=deco(m,n-1)+deco(m-n,n)
  end;
begin writeln(deco(30,30)); readln; end.
```

Контрольные примеры.

$\xi(10) = 42$, $\xi(20) = 627$, $\xi(30) = 5604$,

$\xi(10) = 37338$, $\xi(10) = 204226$.

Задача 9. Последовательность из латинских букв строится следующим образом. На нулевом шаге она пуста. На каждом последующем шаге последовательность удваивается, то есть приписывается сама к себе, и к ней слева добавляется очередная буква алфавита (a,b,c,…). По заданному числу n определить символ, который стоит на n-ом месте последовательности, получившейся после шага 26.

Решение. Эта задача предлагалась участникам областных туров Всероссийской олимпиады школьников по информатике в 1998 году. Приведем первые шаги формирования последовательности: 0 – пустая последовательность, 1 – a, 2 – baa,3 – cbaabaa, 4 – dcbaabaacbaabaa, … . Мы имеем явно рекурсивный процесс.

Построим более общую программу-функцию, чем это требуется по условиям задачи. Пусть abra(k,n) – n-ая буква в последовательности, полученной на шаге k (k=1..26). Тогда ясно, что abra(k,1) равно k-ой букве латинского алфавита. Этот факт можно взять в качестве базы рекурсии, а функцию для определения этой буквы записать так:

$letter(k) := substr("abcdefghijklmnopqrstuvwxyz", k-1, 1)$.

Декомпозицию удобно организовать по k, проводя “раскрутку” последовательности по шагам в обратном направлении. Это приводит к следующей функции:

$$abra(k,n) := \left| \begin{array}{l} letter(k) \quad if \quad n=1 \\ abra\left(k-1, n-if\left(n \le 2^{k-1}, 1, 2^{k-1}\right)\right) \quad otherwise \end{array} \right.$$

k	N														
1	1														A
															1
2	3												**b**	a	a
													1	2	3

3	7								c	b	A	a	**b**	a	a
									1	2	3	4	5	6	7
4	15	D	c	b	a	a	b	a	a	c	b	A	a	**b**	a	A
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Abra(4,13)=abra(3,13-1-(2^3-1))=abra(2,5-1-(2^2-1)=abra(2,1)

```
program Project1;
const alfa:string[26]=('abcdefghijklmnopqrstuvwxyz');
 function power(a,t:byte):longint;
 begin if t=0 then power:=1 else power:=a*power(a,t-1);
 end;
function  abra(k,n:byte):integer;
begin if n=1 then abra:=k
else  if n<=power(2,k-1) then abra:=abra(k-1,n-1)
else abra:=abra(k-1,n-power(2,k-1))
end;
begin  writeln(alfa[abra(26,4)]);
readln;
end.
```

Контрольные примеры.

$abra(4,6) = "b"$ $abra(26,4) = "w"$ $abra(26,2^{26} - 127) = "g"$.