Рекурсия

Рекурсия – подпрограмма, которая в своем теле вызывает сама себя.

В языке Паскаль существует два вида подпрограмм: процедура (PROCEDURE) и функция (FUNCTION). Процедуры и функции в Паскале объявляются в разделе описания за разделом переменных.

```
Program ИмяПрограммы;
VAR ... // раздел описания переменных главной программы;

procedure ИмяПроцедуры;
var ...
begin
... //Тело процедуры
end;

begin
... //тело главной программы
end.
```

Описание подпрограмм

Подпрограмма — часть программы, оформленная в виде отдельной синтаксической конструкции и снабжённая именем (самостоятельный программный блок), для решения отдельных задач.

Процедуры

Описание процедуры:

```
Procedure <имя> (<список формальных параметров>);
 var … {раздел выполнения локальных имён}
Begin
 … {раздел выполнения операторов}
End;
```

Вызов процедуры:

<имя >(<список фактических переменных>);

Функции

Описание функции:

```
Function <имя> (<список формальных параметров>): тип;
 var … {раздел описания локальных имён}
Begin
 … {раздел выполняемых операторов}
<Имя функции>:=<значение>;
{обязательный параметр}
End;
```

Вызов функции:

< оператор>:= <имя функции>
(<список фактических переменных>);

1. В правой части оператора присваивания.
2. В выражении, стоящем в условии оператора разветвления.
3. В процедуре вывода, как результат работы функции.

У функций и процедур существуют **параметры** (переменные, которые передают какое - либо значение). Они бывают двух видов:

1) **Формальные** - те, которые находятся в описании подпрограммы

2) **Фактические** - те, которые передаются из основной программы в функцию или процедуру.

Фактические параметры должны соответствовать формальным по количеству, порядку следования и типу.

Также у подпрограммы существую **переменные**, с которыми она в дальнейшем работает. Они делятся опять же на два типа:

1) **Глобальные переменные**, то есть действующие во всей программе

2) **Локальные** - те, которые действуют только в процедуре или функции

В языке программирования Pascal рекурсивностью могут обладать как функции, так и процедуры.

Наличие условия в теле рекурсивной функции (или процедуры), при котором она больше себя не будет вызывать, очень важно. В противном случае, как и в ситуации с циклами, может произойти так называемое зацикливание.

При каждом рекурсивном вызове создается новое множество локальных переменных. То есть переменные, расположенные вне вызываемой функции, не изменяются.

Задача. Вывести цифры числа в обратном порядке.

Рассмотрим алгоритм решения:

Возьмем число 3096. Последняя цифра - остаток от деления на 10. Это число 6. Оно будет первой цифрой нового числа. Проверяем условие того, что 3096 при деление нацело на 10 больше нуля, то есть число еще не закончилось и есть еще цифры. Делим на 10 и у нас осталось число 309. Снова определяем остаток от деления на 10 и выводим на экран цифру 9, проверим условие 309/10 <>0 и разделим на 10 и получим число 30. Таким образом будем производить операции пока число не обратится в 0.

Воспользуемся рекурсивной процедурой rever:

```
procedure rever (n: integer); {n – искомое число, переданное в процедуру rever, формальная переменная}
begin
write (n mod 10); {выводим текущий остаток от деления на 10 - цифру}
if (n div 10)<>0 then rever (n div 10); {проверяем после деления останется число и передаем новое число в процедуру rever – рекурсивный вызов}
end;
begin
rever (3096);  {вызов процедуры из тела основной программы, 3096 – фактическая переменная}
readln;
end.
```

Работа процедуры:

1. Мы передаем число 3096.
2. Процедура rever выводит на экран остаток от деления на 10. Это число 6.
3 Переход на новую строку не происходит, т.к. используется write.
4 Проверяется условие того, что 3096 при деление нацело на 10 больше нуля.
5 Вызывается rever с фактическим параметром, равным 309.
6 Вторая запущенная процедура выводит на экран цифру 9 и запускает третью процедуру с параметром 30.
7. Третья процедура выводит 0 и вызывает четвертый rever с 3 в качестве параметра.
8. Четвертая процедура выводит 3 на экран и ничего больше не вызывает, т.к. условие (3 div 10) <> 0 ложно.
9. Четвертая процедура завершается и передает управление третьей. Третья процедура завершается и передает управление второй. Вторая процедура завершается и передает управление первой. Первая процедура завершается и передает управление в основную ветку программы.

В итоге, процедура rever была вызвана четыре раза, хотя из основной программы к ней было единственное обращение.

Задача. Найти первых n чисел Фибоначчи.

Числа Фибоначчи — числовая последовательность, первые два элемента которой равны 1, а каждый последующий равен сумме двух предыдущих.

Вывод программы должен выглядеть следующим образом: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, …

Алгоритм решения: Допустим f_n – одно из чисел Фибоначчи, тогда $f_1=1$, $f_2=1$, $f_3= f_2+f_1$, $f_4= f_3+f_2$, тогда $f_n =f_{n-1}+f_{n-2}$.

Пусть рекурсивная функция поиска числа Фибоначчи fib(n). Тогда f_1 и f_2 будут условием выхода из рекурсии:

```
If (f=1) or (f=2) then fib:=1,
```

а для получения нового числа Фибоначчи будем самой рекурсивной функции присваивать сумму вызовов рекурсии со значениями n-1 и n-2:

```
fib:=fib(n-1)+fib(n-2);
```

Осталось создать основную программу и прописать ряд для чисел Фибоначчи.

Найти первых n чисел Фибоначчи.

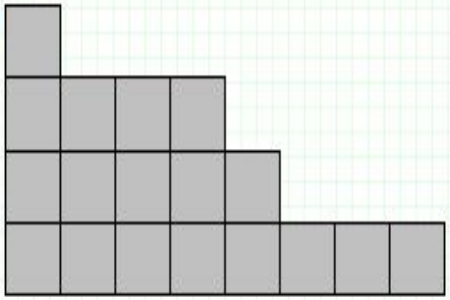
```
Program Fib;
Var i,n:Integer;
{Функция поиска f-ого числа Фибоначчи}
Function fib(f:integer):integer;
Begin
If (f=1) or (f=2) then fib:=1 else fib:=fib(f-1)+fib(n-2);
End;

Begin
Readln(n);
For i:=1 to n do Write(fib(i):4);
Readln;
End.
```

ЗАДАЧА №16

Лесенка

(Время: 1 сек. Память: 16 Мб Сложность: 55%)

Лесенкой называется набор кубиков, в котором каждый более верхний слой содержит кубиков меньше, чем предыдущий. Требуется написать программу, вычисляющую число лесенок, которое можно построить из N кубиков.



Входные данные

Во входном файле INPUT.TXT записано натуральное число N ($1 \le N \le 100$) – количество кубиков в лесенке.

Выходные данные

В выходной файл OUTPUT.TXT необходимо вывести число лесенок, которые можно построить из N кубиков.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3	2
2	6	4

Алгоритм решения:

Допустим у нас 3 кубика. Берем все кубики и выкладываем в 1 слой – это первая лесенка, далее берем 1 кубик и кладем на верх – это вторая лесенка. Суть рекурсии в том что когда мы будем "отбирать" от первого ряда больше кубиков, то мы не будем учитывать первый ряд, а просто представим что из взятого надо построить опять же лесенку. Алгоритм похож на Фибоначчи.

```
Var  d : LongInt; { количество возможных лесенок }
     N : ShortInt; { количество кубиков }
```

Процедура разложения числа N:

```
Procedure BreakDown (min, max : ShortInt);
{min – минимально возможное количество кубиков для существования лесенки, max – количество кубиков N}
     Var  i : ShortInt;
     Begin
     …
     End;
```

Допустим i - количество кубиков, отведенных на нижний слой. Тогда следующий слой начнется с i + 1 кубика и должен состоять по условию из max − i кубиков, тогда:

```
for i := min to max do BreakDown (i + 1, max − i); {вызываем процедуру для построения нового слоя лесенки}
if (max = 0) then Inc (d); {если не можем добавить новый слой, то лесенка существует}
```

Осталось собрать все в процедуру и вызвать из основной программы, передав процедуре начальное и конечное значения кубиков (например, min=1, max=N)