Богомолова Ольга Борисовна,
Усенков Дмитрий Юрьевич

«МАТРЕШКИ» ПОД НОМЕРОМ 11: ЗАДАЧИ НА РЕКУРСИЮ

Есть в программировании такой интересный прием, как *рекурсия* – ситуация, когда некоторая подпрограмма (процедура) в процессе своей работы вызывает сама себя; такая ситуация называется *прямой рекурсией*. Наглядным представлением рекурсии является всем известная русская матрешка. Еще один пример показан на иллюстрации ниже: это фотография руки, которая держит фотографию руки, которая… и так до бесконечности. Возможна также *косвенная рекурсия*, когда некоторая процедура (условно назовем ее «А») вызывает другую процедуру «Б», а уже процедура «Б» вызывает процедуру «А».

Важной частью любой рекурсивной подпрограммы является *условие прекращения рекурсивных вызовов* – условный оператор, в зависимости от истинности условия которого производится либо рекурсивный вызов, либо возврат некоторого конкретного константного значения. Иначе рекурсивные вызовы станут бесконечными и приведут к зависанию компьютера.

Конечно же, рекурсию не смогли обойти вниманием составители заданий ЕГЭ. Подобные задачи (ныне фигурирующие под номером 11) давно уже «прописались» в Едином ГосЭкзамене и традиционно вызывают значительные трудности у учащихся. Впрочем, научиться их решать достаточно просто, если понять сам принцип решения.

Задача 1. Алгоритм вычисления значения функции $F(n)$, где n – натуральное число, задан следующими соотношениями:

$F(1) = 2$; $F(2) = 3$;
$F(n) = F(n–1) + F(n–2)$ при $n > 2$.

Чему равно значение функции $F(5)$?

Решение

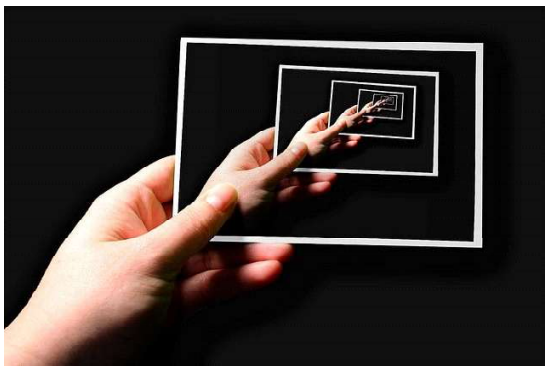
Это самая старая «ЕГЭшная» задача. Казалось бы, чего проще: мы знаем два первых значения некоторой последовательности чисел, а все остальные вычисляются по заданной рекуррентной формуле (в которой нетрудно узнать формулу последовательности чисел Фибоначчи). То есть достаточно поочередно расписывать значения F для каждого очередного n и при этом подставлять в формулу известные или ранее вычисленные значения:

$F(3) = F(3–1) + F(3–2) = F(2) + F(1) = 3 + 2 = 5$;

$F(4) = F(4–1) + F(4–2) = F(3) + F(2) = 5 + 3 = 8$;

$F(5) = F(5–1) + F(5–2) = F(4) + F(3) = 8 + 5 = 13$.

Ответ: 13.

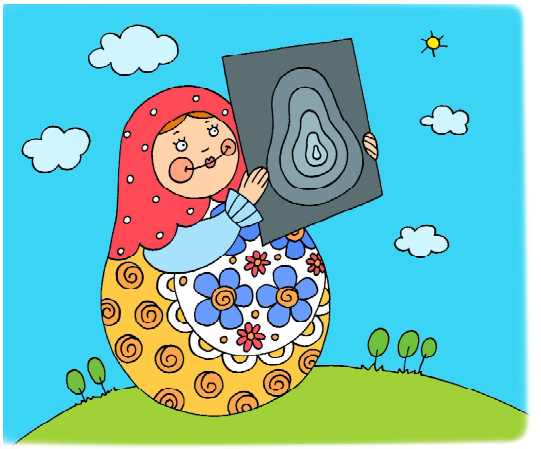


«Визуальный» пример рекурсии

«Размявшись» на этом простом примере, перейдем к более серьезным задачам, которые встречаются на экзаменах и тренажах.

Задача 2. Имеется следующий рекурсивный алгоритм:

```
procedure F(n: integer);
begin
 writeln(n);
 if n < 6 then begin
   F(n+1);
   F(n*2)
 end
end;
```

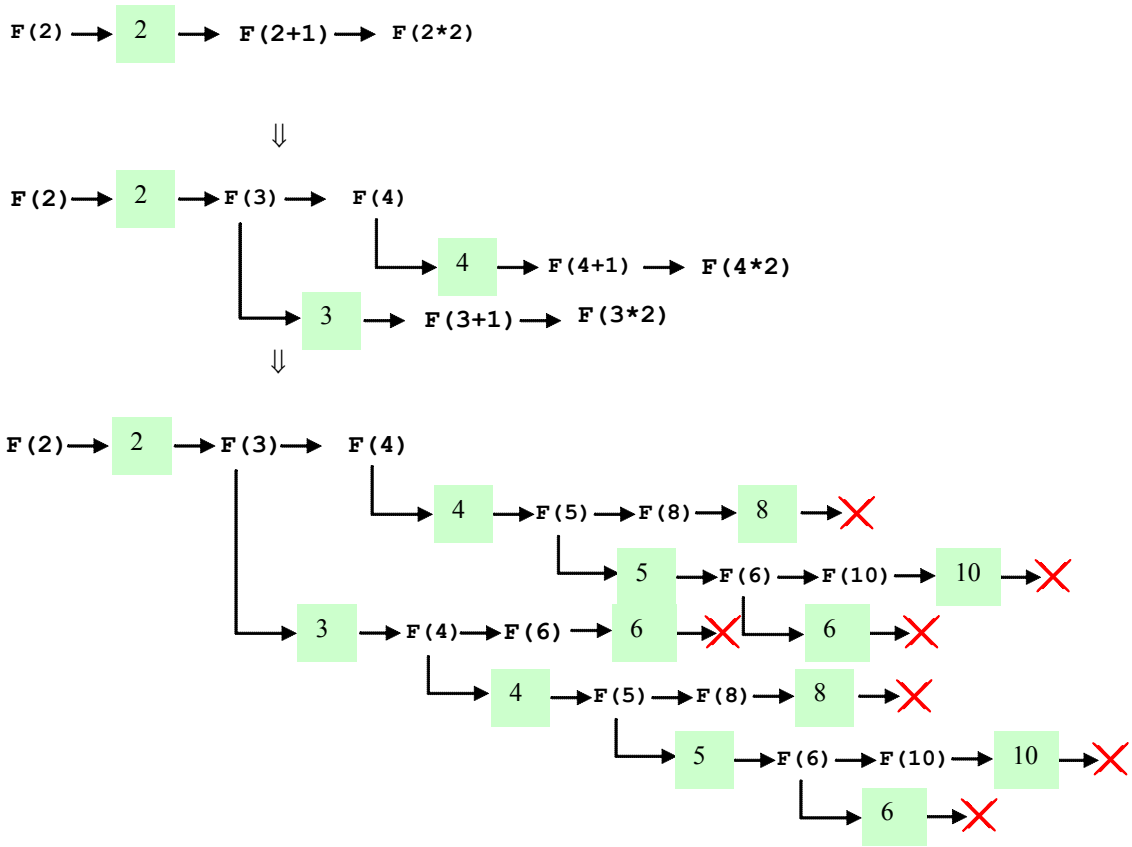


Чему равна сумма чисел, выведенных на экран при вызове `F(2)`?

Решение

Рассматриваемый алгоритм представляет собой прямую рекурсию, в которой имеются два вложенных рекурсивных вызова процедуры `F()`, а условием прекращения рекурсии является $n \geq 6$, когда рекурсивные вызовы больше не производятся.

Наиболее универсальным способом решения таких задач является построение «дерева вызовов» функции, в данном случае с последующим подсчетом суммы чисел на концах его «ветвей». Вот как для этой задачи можно построить «дерево вызовов» последовательно «по шагам»:



(Знак ✕ обозначает, что данная «ветвь» завершена, так как сработало условие прекращения рекурсии.)

Далее остается просуммировать числа, которые выводятся на экран при работе данного алгоритма (и которые в нашем «дереве» выделены зеленым фоном). Эта сумма равна 2 + 4 + + 8 + 5 + 10 + 6 + 3 + 6 + 4 + 8 + 5 + 10 + 6 = 77.

Однако даже для такой несложной задачи «дерево вызовов» получается довольно громоздкое, и к тому же при его рисовании требуется внимательность, чтобы ничего не упустить (а особенно – не забыть, что, даже если очередное значение *n* уже не удовлетворяет условию в операторе **if**, значение *n* все равно выводится на экран, так как соответствующий оператор располагается ***до*** условного оператора) и просуммировать все полученные числа. Как показала практика, с рисованием таких «деревьев» справляются далеко не все учащиеся.

Взамен рисования «дерева вызовов» можно предложить *табличный метод решения*. Суть его в том, что мы, начиная с исходного значения *n* (в данном случае 2), составляем таблицу, в которой расписываем и то, что выводится на экран, и требуемые рекурсивные вызовы процедуры с соответствующими новыми значениями *n*, которые тоже помещаем в нашу таблицу… и так до тех пор, пока не дойдем до условия прекращения рекурсии. А после этого мы снова проходим по составленной таблице уже в обратном порядке, вычисляя возвращаемые при рекурсивных вызовах значения и подставляя эти значения в предыдущие строки таблицы.

Вспомним основную часть рассматриваемого рекурсивного алгоритма:

```
writeln(n);
 if n < 6 then begin
   F(n+1);
   F(n*2)
 end
```

Распишем построение таблицы для этой задачи подробно. Начинаем с заданного начального значения *n* = 2 и не забываем, что значение n выводится на экран в любом случае (так как команда **writeln(n)** располагается вне оператора **if**). Получающиеся новые вызовы процедуры **F** будем выделять голубым фоном.

n	F(n+1)	F(n*2)	Вывод на экран
2	F(3)	F(4)	2

Теперь добавляем в таблицу строки для *n* = 3 и *n* = 4:

n	F(n+1)	F(n*2)	Вывод на экран
2	F(3)	F(4)	2
3	F(4)	F(6)	3
4	F(5)	F(8)	4

В таблице появились новые вызовы процедуры **F** для *n* = 5, 6 и 8. Добавляем в таблицу и их. Но не забываем, что для значений 6 и 8 уже срабатывает условие завершения рекурсии, и никаких вызовов F для них уже не будет. Но само значение *n* на экран по-прежнему выводится!

n	F(n+1)	F(n*2)	Вывод на экран
2	F(3)	F(4)	2
3	F(4)	F(6)	3
4	F(5)	F(8)	4
5	F(6)	F(10)	5
6	–	–	6
8	–	–	8

Наконец, добавляем последнюю строку для `F(10)`:

n	`F(n+1)`	`F(n*2)`	**Вывод на экран**
2	`F(3)`	`F(4)`	2
3	`F(4)`	`F(6)`	3
4	`F(5)`	`F(8)`	4
5	`F(6)`	`F(10)`	5
6	–	–	**6**
8	–	–	**8**
10	–	–	**10**

Видим, что никаких новых рекурсивных вызовов процедуры `F` больше нет. Можно приступать ко второй части построения таблицы – к просмотру ее строк с вызовами F построчно снизу вверх. При этом в каждой очередной строке сразу будем вычислять сумму выводимых на экран чисел, прибавляя к уже записанному значению то, что должно выводиться за счет указанных в таблице рекурсивных вызовов. Полученное очередное значение суммы является результатом данного вызова F и затем, в свою очередь, подставляется в предыдущую строку таблицы. И так – пока мы не дойдем до самой первой строки, содержащей указанный в условии задачи изначальный вызов рекурсивной функции.

n	`F(n+1)`	`F(n*2)`	**Вывод на экран**
5	`F(6)`	`F(10)`	5 + **6** + **10** = 21
6	–	–	**6**
8	–	–	8
10	–	–	**10**

Аналогично поступаем с предыдущей строкой для $n = 4$:

n	`F(n+1)`	`F(n*2)`	**Вывод на экран**
4	`F(5)`	`F(8)`	4 + **21** + **8** = 33
5	`F(6)`	`F(10)`	**21**
6	–	–	6
8	–	–	**8**
10	–	–	10

Теперь добавляем строку для предыдущего значения $n = 3$ (стрелки уже рисовать не будем, только выделим цветом соответствующие значения):

n	`F(n+1)`	`F(n*2)`	**Вывод на экран**
3	`F(4)`	`F(6)`	3 + **33** + **6** = 42
4	`F(5)`	`F(8)`	**33**
5	`F(6)`	`F(10)`	21
6	–	–	**6**
8	–	–	8
10	–	–	10

И, наконец, добавляем самую первую строку для $n = 2$:

n	F(n+1)	F(n*2)	Вывод на экран
2	F(3)	F(4)	2 + 42 + 33 = 77
3	F(4)	F(6)	42
4	F(5)	F(8)	33
5	F(6)	F(10)	21
6	–	–	6
8	–	–	8
10	–	–	10

Полученное при последнем сложении число 77 и есть ответ.

Ответ: 77.

Теперь рассмотрим более сложный вариант задания, в котором вывод на экран производится и безусловно (вне оператора `if`) и внутри оператора `if`, если условие в нем истинно.

Задача 3. Имеется следующий рекурсивный алгоритм:

```
procedure F(n: integer);
begin
 writeln(n);
 if n < 7 then begin
   writeln(n);
   F(n+1);
   F(n+2);
   F(n*2)
 end
end;
```

Чему равна сумма чисел, выведенных на экран при вызове `F(2)`?

Решение

Уже познакомившись с решением предыдущей задачи, будем записывать решение более коротко.

Таблицу начинаем составлять, начиная с исходного значения $n = 2$. Теперь в ней будет три графы для записи вложенных рекурсивных вызовов, а в графе «Вывод на экран» не забываем указывать два значения, если условие в `if` истинно.

n	F(n+1)	F(n+2)	F(n*2)	Вывод на экран
2	F(3)	F(4)	F(4)	2 + 2
3	F(4)	F(5)	F(6)	3 + 3
4	F(5)	F(6)	F(8)	4 + 4
5	F(6)	F(7)	F(10)	5 + 5
6	F(7)	F(8)	F(12)	6 + 6
7	–	–	–	7
8	–	–	–	8
9	–	–	–	9
10	–	–	–	10
12	–	–	–	12

А теперь идем по таблице в обратном порядке (снизу вверх):

n	F(n+1)	F(n+2)	F(n*2)	Вывод на экран
2	F(3)	F(4)	F(4)	2+2+232+121+121 = **478**
3	F(4)	F(5)	F(6)	3+3+121+66+39 = 232
4	F(5)	F(6)	F(8)	4+4+66+39+8 = 121
5	F(6)	F(7)	F(10)	5+5+39+7+10 = 66
6	F(7)	F(8)	F(12)	6+6+7+8+12 = 39
7	–	–	–	7
8	–	–	–	8
9	–	–	–	9
10	–	–	–	10
12	–	–	–	12

Ответ: 478.

Теперь попробуем решить новую задачу с еще одним небольшим усложнением.

Задача 4. Имеется следующий рекурсивный алгоритм:

```
procedure F(n: integer);
begin
 writeln(n);
 n := n+2;
 if n < 8 then begin
   F(n+2);
   F(n+3);
end
end;
```

Чему равна сумма чисел, выведенных на экран при вызове `F(1)`?

Решение

Здесь вне оператора `if` добавился оператор увеличения значения *n*, но нам составлять таблицу это не помешает – нужно лишь не забывать, что в условии оператора `if` и в последующих вложенных вызовах `F` участвует уже измененное значение *n*.

n	Новое *n*	F(n+2)	F(n+3)	Вывод на экран
1	3	F(5)	F(6)	1
5	7	F(9)	F(10)	5
6	8	–	–	6
9	11	–	–	9
10	12	–	–	10

В графу «Вывод на экран» попадают первоначальные значения *n*, так как вывод производится до изменения *n*. Что будет выводиться, если команда вывода будет стоять после операции `ifn:=n+2` или внутри оператора `if`, читатели, наверное, уже определили сами.

Теперь после пути «туда», подобно небезызвестному хоббиту, возвращаемся «обратно»:

n	Новое *n*	F(n+2)	F(n+3)	Вывод на экран
1	3	F(5)	F(6)	1+24+6 = **31**
5	7	F(9)	F(10)	5+9+10 = 24
6	8	–	–	6
9	11	–	–	9
10	12	–	–	10

Ответ: 31.

Теперь попробуем тем же способом решить версию такого задания с выводом «звездочек».

Задача 5. Дан рекурсивный алгоритм:

```
procedure F(n: integer);
begin
writeln('*');
 if n > 0 then begin
   writeln('*');
   F(n-2);
   F(n div 2);
   F(n div 2);
 end
end;
```

Сколько символов «звездочка» будет напечатано на экране при выполнении вызова `F(7)`?

Решение

Все различие здесь – в том, что в графе «Вывод на экран» числа будут обозначать не сами значения, выводимые на экран, а количества выводимых туда «звездочек» (при первичном заполнении – отмечаем, что *по одной* в каждой команде вывода). И, конечно же, не забываем, что здесь имеются две команды вывода на экран. Кроме того, здесь имеется два одинаковых рекурсивных вызова `F(n div 2)`, и, чтобы про них не забыть, в таблице мы запишем две соответствующие графы – пусть даже информация в них будет дублироваться.

n	`F(n-2)`	`F(n div 2)`	`F(n div 2)`	Вывод на экран
7	`F(5)`	`F(3)`	`F(3)`	1+1
5	`F(3)`	`F(2)`	`F(2)`	1+1
3	`F(1)`	`F(1)`	`F(1)`	1+1
2	`F(0)`	`F(1)`	`F(1)`	1+1
1	`F(-1)`	`F(0)`	`F(0)`	1+1
0	–	–	–	1
–1	–	–	–	1

А теперь идем обратно:

n	`F(n-2)`	`F(n div 2)`	`F(n div 2)`	Вывод на экран
7	`F(5)`	`F(3)`	`F(3)`	1+1+45+17+17 = **81**
5	`F(3)`	`F(2)`	`F(2)`	1+1+17+13+13 = 45
3	`F(1)`	`F(1)`	`F(1)`	1+1+5+5+5 = 17
2	`F(0)`	`F(1)`	`F(1)`	1+1+1+5+5 = 13
1	`F(-1)`	`F(0)`	`F(0)`	1+1+1+1+1 = 5
0	–	–	–	1
–1	–	–	–	1

Ответ: 81.

А если в задаче используется еще и косвенная рекурсия, и не одна, а две функции перекидывают друг другу вложенные вызовы, словно играя в пинг-понг? Ну что же – нам это тоже не помешает, – разве что заставит параллельно заполнять две части таблицы для `F` и для `G`.

Задача 6. В программе содержатся две рекурсивные функции **F** и **G**:

```
function F(n: integer): integer;
begin
  if n > 3 then
    F := F(n - 2) + G(n - 3)
  else
    F := n - 1;
  end;
function G(n: integer): integer;
begin
  if n > 3 then
    G := G(n - 2) + F(n - 3)
  else
    G := n + 1;
  end;
```

Чему равно значение, вычисленное при выполнении вызова **F(8)**?

Решение

Построение таблицы начинаем с заданного значения $n = 8$. Саму таблицу удобно строить из двух частей (соответствующих вызовам **F(n)** и **G(n)**); при этом, как и раньше, сначала мы только записываем вызовы функций, а затем (на обратном пути) в обратном порядке снизу вверх записываем получаемые значения. А для большей компактности будем делать записи в таблице более кратко.

n	F(n)		*n*	G(n)
F(8)	F(8)=F(6)+G(5)		G(5)	G(5)=G(3)+F(2)
F(6)	F(6)=F(4)+G(3)		G(3)	G(3)=G(1)+F(0)
F(4)	F(4)=F(2)+G(1)		G(1)	G(1)= 1+1 = 2
F(2)	F(2) = 2 – 1 = 1			
F(0)	F(0) = 0 – 1 = –1			

F(n)		G(n)
F(8)= 4+2 = 6		G(5)= 1+1 = 2
F(6)= 3+1 = 4		G(3)= 2–1 = 1
F(4)= 1+2 = 3		G(1) = 2
F(2) = 1		
F(0) = –1		

Ответ: 6.

А «на закуску» предлагаем еще один новый тип задач на косвенную рекурсию. Такая задача выглядит более сложной, но на самом деле она даже проще, чем ранее рассмотренные.

Задача 7. Имеются алгоритмы рекурсивных процедур **F** и **G**. Определите сумму чисел, выведенных на экран при выполнении вызова **F(93)**.

```
procedure F(n:integer); forward;
procedure G(n:integer); forward;

procedure F(n: integer);
 begin
  if n mod 5 > 3 then G(n div 5)
```

```
  else G(n div 2)
 end;

procedure G(n: integer);
 begin
  write(n, ' ');
  if n > 1 then F(n-1);
 end;
```

Решение

Для решения составим таблицу, начиная отсчет с заданного значения *n* (промежуточные вычисления для определения выполняемых вызовов `F(n)` и `G(n)` выполним «в уме»):

`F(n)`	`F(93)`	`F(45)`	`F(21)`	`F(9)`	**Вызовы прекращены**
`G(n)`	`G(46)`	`G(22)`	`G(10)`	`G(1)`	**Итоговая сумма:**
вывод на экран	46	22	10	1	46+22+10+1 = **79**

Ответ: 79.

Богомолова Ольга Борисовна, доктор педагогических наук, почетный работник сферы образования Российской Федерации, Заслуженный учитель города Москвы, учитель информатики и математики ГБОУ СОШ № 1360, г. Москва,

Усенков Дмитрий Юрьевич, Московский государственный институт индустрии туризма имени Ю.А. Сенкевича, г. Москва.