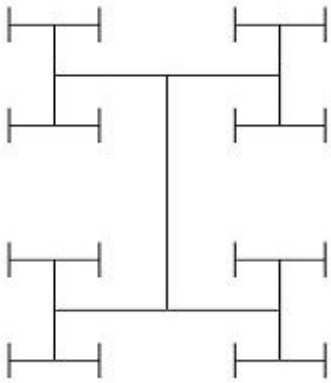
Рекурсивная снежинка

Условие задачи.

Нарисовать картинку, такую как показано ниже.



Начнем наши рассуждения с уточнения задачи. Из формы рисунка ясно, что длины линий уменьшаются, но не ясно на сколько. Величину уменьшения можно определить самостоятельно, так как никаких прямых указаний на этот счет в условии нет. Положим, что линии уменьшаются в два раза.

Решение, общий подход.

Задача кажется сложной. И метод, которым мы воспользуемся, выглядит весьма необычно. Не генерируя никаких идей, не создавая общего плана поиска решения, мы построим цепочкурассуждений, каждое из которых несложно и, как кажется, лишь немного будет продвигать нас к цели, но в конечном итоге получится требуемый результат. Воздержимся от общих схем, а просто продемонстрируем метод в действии и лишь затем скажем несколько общих слов.

Цепочка рассуждений.

Для начала заметим, что каждый отрезок порождает еще два, и это своего рода рекуррентное определение очередного отрезка. Любое рекуррентное определение наводит на мысль о построении рекурсивной процедуры. Отсюда следует, что головная программа может состоять из двух команд: это рисование самого первого отрезка и вызов процедуры рисующей новый отрезок. Ранее, однако, было сказано, что каждый отрезок порождает еще два, следовательно, в головной программе должно быть два вызова. Запишем то, что уже ясно:

```
line(300,50,300,250);
rec_uzor();
```

```
rec_uzor();
```

Далее займемся построением процедуры rec_uzor. Обратите внимание, что мы уже построили кусок программы и ни слова не сказали про алгоритм решения или какую-нибудь содержательную идею.

Сейчас попробуем определить, что должна знать процедура для выполнения своей работы. Процедура рисует новый отрезок от вершин уже нарисованного, следовательно, она должна знать координаты вершин предыдущего отрезка. Далее, процедура должна знать вертикальный или горизонтальный отрезок ей рисовать, для чего необходимо сообщить ей ориентацию предыдущего отрезка. Договоримся сами с собой, что 1 означает вертикальный отрезок, а 2 — горизонтальный. Далее, новый отрезок должен быть в два раза меньше предыдущего, следовательно, длина предыдущего также должна быть известна. Вроде бы вся необходимая информация для построения отрезка уже определена, но есть еще одно небольшое соображение по поводу рекурсивной природы процедуры. Мы знаем, что необходимо позаботиться о завершении цепочки рекурсивных вызовов. Пусть вызовы прекращаются тогда, когда номер очередного вызова достигнет определенного значения. Для обеспечения такой возможности необходимо передавать номер очередного вызова. Сейчас уже возможно записать немного больше текста. Во-первых, можем записать полностью текст головной программы:

```
driver:=detect;
initgraph(driver,mode,'');
line(300,50,300,250);
rec_uzor(300,50,1,200,1);
rec_uzor(300,250,1,200,1);
```

В первом вызове сообщается, что центр очередного отрезка должен находиться в точке (300, 50), предыдущий отрезок был вертикальным, его длина была 200 пикселов и данный вызов первый.

Можем записать заголовок нашей рекурсивной процедуры

```
procedure rec_uzor(x,y,k,l,n:integer);
```

Далее построим тело процедуры, для чего проанализируем, какую информацию несет каждая из переданных величин. Понятно, что каждая величина передается для чего-то, и это что-то выполняется каким-то набором команд, следовательно, если удачно проведем анализ, то нам удастся нарастить текст программы. Начнем с последней переменной, так как она несет самую общую информацию о структуре процедуры. А именно она говорит следующее: "если ее значение не достигло некоторой критической величины, то можно выполнять необходимые действия, иначе нельзя". Пусть это критическое значение равно 10. Тогда сразу появляется следующий текст процедуры

```
if n<10 then
  begin
  end;
```

Между begin и end находится весь остальной текст процедуры, о котором нам пока ничего не известно. Возьмем следующую переменную. Она несет в себе информацию о длине, про которую известно, что она должна быть уменьшена в два раза. Это можно сделать уже сейчас и фрагмент увеличивается еще на один оператор

```
if n<10 then
  begin
   l:=round(l/2);
  end;
```

Следующая величина говорит о том, что возможны две ситуации (ранее была нарисована вертикальная или горизонтальная линия), для каждой из которых данная величина принимает свое определенное значение. Это означает, что мы можем записать условный оператор, учитывающий два значения величины k. Текст процедуры еще немного увеличивается.

```
if n<10 then
  begin
   l:=round(l/2);
   if k=1 then begin
   end
  else begin
  end;
  end;
```

Для тех, кто предпочитает оператор-переключатель этот фрагмент текста можно переписать следующим образом:

```
if n<10 then
  begin
   l:=round(l/2);
   case k of
    1: begin
       end;
    2: begin
       end;
   end;
  end;
```

Наше обещание выполняется, до сих пор не было никаких рассуждений об алгоритме решения задачи, мы просто анализируем данные и рассуждаем о том, что надо сделать, чтобы получение процедурой переменных имело смысл, а текст процедуры растет, и каждый промежуточный результат кажется вполне логичным.

Продолжим анализ. Осталось две переменные, имеющие смысл координат. Они могут потребоваться только для рисования текущего отрезка. Если необходимо нарисовать горизонтальный отрезок, то его координата Y конечно равна той, которая была получена процедурой, а координаты *X* левого и правого конца пусть отличаются от известной координаты центра на величину длины. Для вертикального отрезка наоборот, координата *X* остается без изменений, а координаты Y верхнего и нижнего конца отличаются от координат центра на величину длины. Теперь можно записать две процедуры рисования отрезка.

```
if n<10 then
  begin
    l:=round(l/2);
    if k=1 then begin
                  x1:=x-l;x2:=x+l;
```

```
                line(x1,y,x2,y);
              end
          else begin
                y1:=y-l;y2:=y+l;
                line(x,y1,x,y2);
              end;
  end;
```

И последний момент. В самом начале сказано, что каждый нарисованный отрезок порождает два вызова процедуры, для каждого из которых опять так же, как и в начале, простыми рассуждениями мы можем определить передаваемые параметры.

```
if n<10 then
  begin
    l:=round(l/2);
    if k=1 then begin
                  x1:=x-l;x2:=x+l;
                  line(x1,y,x2,y);
                  rec_uzor(x1,y,2,l,n+1);
                  rec_uzor(x2,y,2,l,n+1);
                end
           else begin
                  y1:=y-l;y2:=y+l;
                  line(x,y1,x,y2);
                  rec_uzor(x,y1,1,l,n+1);
                  rec_uzor(x,y2,1,l,n+1);
                end;
  end;
```

Текст процедуры готов полностью, как и было обещано, мы ни разу не обмолвились об алгоритме, не было никаких общих рассуждений, только небольшие пошаговые построения, но программа удалась. Ее полный текст приведен в листинге

Листинг

```
program example;
       uses crt,graph;
       var
        driver,mode:integer;
     procedure rec_uzor(x,y,k,l,n:integer);
      var
       x1,y1,x2,y2:integer;
     begin
      if n<10 then
        begin
          l:=round(l/2);
          if k=1 then begin
                        x1:=x-l;x2:=x+l;
                        line(x1,y,x2,y);
                        rec_uzor(x1,y,2,l,n+1);
                        rec_uzor(x2,y,2,l,n+1);
                      end
                 else begin
```

```
                        y1:=y-l;y2:=y+l;
                        line(x,y1,x,y2);
                        rec_uzor(x,y1,1,l,n+1);
                        rec_uzor(x,y2,1,l,n+1);
                      end;
        end;
     end;
     begin
      driver:=detect;
      initgraph(driver,mode,'');
      line(300,50,300,250);
      rec_uzor(300,50,1,200,1);
      rec_uzor(300,250,1,200,1);
end.
```

А теперь попробуем поговорить о методе. Сущность его заключается в том, что данные прямо и непосредственно определяют выполняемые над ними действия. В каком-то смысле данные всегда определяют действия, но не всегда так явно. Например, вспомним задачу получения выборок, в ней предыдущая выборка без всякого сомнения определяет последующую, но между предыдущей и последующей выборкой располагается достаточно сложная логика, причем в этой логике возможны варианты, очень сильно отличающиеся друг от друга (в задаче о выборках у нас было два решения). В задаче о снежинке не так. Например, величина длины определяет один простой оператор присваивания, самая первая величина (номер вызова) определяет конструкцию ветвления, которая впоследствии сильно разрастается, но в момент своего написания она достаточно проста, написать ее по другому можно, но данный вид очень естественен, и даже иное написание (например, с помощью оператора case) по сути будет тем же самым.

Конечно, приходится признать, что выбор задачи очень удачен и применить этот метод на произвольной задаче вряд ли получится так же эффективно, но нужно помнить, что *данные всегда в той или иной степени определяют выполняемые над ними операции*. И напоследок сформулируем еще одно очень полезное правило, дополняющее только что записанное.

Правило учета логики данных

Если была введена величина, значит она была введена для каких-то целей. Следовательно, если в вашей программе есть неиспользованная переменная, то скорее всего в вашей программе ошибка.

Это правило уже не имеет прямого отношения к решенной задаче, оно из нее вытекает. Ведь если данные определяют действия, и с каждым данным должна быть сопоставлена группа операторов, то отсутствие этой группы прямым и явным образом указывает на возможность ошибки. Кстати это настолько очевидно, что даже зачастую определение неиспользованных переменных берут на себя компиляторы.