Многие задаются вопросом, так что же такое рекурсия? Рекурсия - это подпрограмма (процедура), вызывающая сама себя. А вот для чего она нужна? Я думаю, что после прочтения статьи вы и сами поймёте 🙄 Итак, начнём!

Представим, что мы пишем компьютерную игру на космическую тему. У нас есть "двухмерный" корабль, состоящий из нескольких рядов самостоятельных блоков, причём каждый блок может существовать самостоятельно (это, как бы отдельный мини корабль), или несколько блоков могут образовывать более большой корабль. На наш корабль напали пришельцы. Поскольку у нас стоят очень мощные пушки, наш корабль быстренько расстрелял всех пришельцев. Однако без ущерба не получилось - пришельцы разбили несколько блоков нашего корабля. Известны разбитые блоки. Нам нужно найти на сколько более маленьких кораблей развалился наш главный корабль.

Так можно визуализировать состояние корабля до атаки пришельцев (единички - целые блоки) :

```
111111111111
111111111111
111111111111
```

А вот так может выглядеть корабль после атаки пришельцев (нулями обозначены взорванные блоки) :

```
110110110111
100010010111
111011010011
```

В данном примере понятно, что корабль развалится на 4 более маленьких корабля.

Найти количество получившихся кораблей довольно просто. Идея такова:

Каждый корабль ограничен либо нулями, либо концом нашего блока. Значит, достаточно найти хотя бы по одной 1 из каждого корабля. Делается это так: находим одну единичку корабля, все остальные единички этого корабля обнуляем. Тогда мы найдём всего 4 единицы, т.е. это и будет количество наших кораблей. А вот обнулять все единички одного корабля мы будем рекурсией, т.к. не знаем размер каждого корабля. Как мы знаем, рекурсия - это процедура вызывающая сама себя. Рассмотрим это на нашем примере.

Мы нашли первую единичку корабля, это наша текущая клетка, она передаётся в рекурсию (точнее, передаётся не сама клетка, а её координаты). Рекурсия обнуляет эту клетку и пытается идти "вниз", "вверх", "вправо" и "влево" от текущей клетки. Из каждой клетки, в которую мы дошли, пытается сделать тоже самое. Почему пытается, да потому, что если текущая клетка - 0, то идти в неё нет смысла, наш корабль кончился. Если мы побывали в клетке, то там стоит 0, туда мы уже не пойдём. А вот и текст программы на паскале:

Pascal код:

```
Program Platform; {Как известно, название программы может быть любым}
var
   Ship : array [1..100,1..100] of byte; {Двухмерный массив для хранения корабля}
   i, j, n, m, Num : integer; { n x m - размеры корабля; i, j - счётчики циклов; }
     {Num - количество маленьких кораблей}

   procedure Rec(x, y: integer); {Это наша рекурсия}
   begin
     if (x<1) or (x>n) or (y<1) or (y>n) or (Ship[x, y]=0) then Exit;
       {Если мы вышли за границы, либо текущая клетка - 0, выходим}
```

```
    Ship[x, y]:=0; {Если всё нормально, обнуляем текущую клетку.}
      {Теперь мы знаем, что в ней уже были}
    Rec(x+1, y); {Пытаемся иди вниз и повторить действия}
    Rec(x-1, y); {Пытаемся идти вверх и повторить действия}
    Rec(x, y+1); {Пытаемся идти вправо и повторить действия}
    Rec(x, y-1); {Пытаемся идти влево и повторить действия}
  end;

begin
  readln(n, m); {Считываем размер нашего корабля}
  for i:=1 to n do {Эта строка отвечает за строки нашей матрицы...}
    for j:=1 to m do {...а эта за столбцы}
      read(Ship[i,j]); {Считываем значение каждой клетки}
  Num:=0; {Пока мы не нашли ниодного корабля, ведь мы и не искали ;) }
  for i:=1 to n do {Теперь нужно перебирать не только строки...}
    for j:=1 to m do {...но и столбцы матрицы}
      if Ship[i,j]=1 then {Если в массиве мы нашли единичку, то...}
      begin
        inc(Num); {Увеличиваем счётчик найденных кораблей...}
        Rec(i,j); {... и рекурсивно обнуляем остальные единички корабля}
      end;
  writeln(Num); {Выводим получившееся количество кораблей}
end.
```

Итак, мы уже решили свою первую задачку рекурсией. Теперь даже следующая задачка не составит для вас труда:

Давным-давно все живые существа жили на одной прямоугольной планете. Но, после попадания в их прямоугольную планету нескольких метеоритов, большая планета раскололась на несколько маленьких... Планеты продолжали жить своей жизнью, и на каждой из них через большое количество лет образовалось по новой цивилизации. Так сколько же образовалось цивилизаций?

Как вы уже поняли, задача полностью аналогична предыдущей, однако условие другое...

Из этого становится понятно, что под любым условием нужно видеть математическую суть задачи, а не легенду задачи. Очень часто тяжелее всего в задаче правильно понять условие, уловить математическую суть задачи. Но часто бывает, что если правильно понять условие, то, считайте, задача уже решена.

Теперь усложним нашу предыдущую задачу:

Пускай, наша планета развалилась на несколько цивилизаций. Нужно узнать "мощь" каждой цивилизации, т.е. по сколько клеточек содержит в себе каждая. Причём порядок вывода не важен.

Это можно сделать при помощи введения дополнительной переменной. Ну что ж, ближе к делу:

Введём дополнительную глобальную переменную, которая и будет содержать количество клеточек. Перед входом в рекурсию будем обнулять переменную "мощи". А в рекурсии, если мы идём в текущую летку, а не выходим из рекурсии, увеличиваем эту переменную.

Следовательно главный цикл программы изменится на:

Pascal код:

```
for i:=1 to n do
  for j:=1 to m do
    if Ship[i,j]=1 then
    begin
      Power:=0;
      Rec(i,j);
      writeln(Power);
    end;
```

Где Power - глобальная переменная "мощности", хранящая в себе количество клеточек текущей цивилизации. В данной реализации мощность будет выводится сразу же, после её обнаружения.

P.S. Переменную Num можно было оставить и выводить в конце количество.

А процедура рекурсии изменится на:

Pascal код:

```
procedure Rec(x, y: integer);
begin
  if (x<1) or (x>n) or (y<1) or (y>n) or (Ship[x, y]=0) then Exit;
  Ship[x]:=0;
  inc(Power);
  Rec(x+1, y);
  Rec(x-1, y);
  Rec(x, y+1);
  Rec(x, y-1);
end;
```

Ну вот мы и разобрались с рекурсией. На самом деле, рекурсия - очень полезный алгоритм. А главное, она очень легко пишется. Рекурсия использует глобальный стек, поэтому вам не придётся реализовывать свой, на этом экономится довольно много времени. Очень часто рекурсия может помочь в тяжёлых ситуациях.

Однако, рекурсия используется не только в "игрушечных" и олимпиадных задачах, но и в повседневных программах. Например, поиск файла в папке с подпапками проще всего реализовать рекурсией. И многое-многое другое. Каждый нормальный программист просто обязать уметь писать рекурсию.