ЗАОЧНАЯ ШКОЛА СОВРЕМЕННОГО ПРОГРАММИРОВАНИЯ

Дорогие ребята, учителя информатики и математики!

Закончился первый (и уже начался второй) учебный год в Заочной школе современного программирования (ЗШСП), организованной математико-механическим факультетом Санкт-Петербургского государственного университета совместно с нашим журналом (научный руководитель школы профессор И.В. Романовский).

За это время мы получили несколько сотен писем, и более сотни ребят стали учениками школы. Хочется подвести промежуточные итоги работы школы, ответить на некоторые вопросы, которые часто задавали ребята и учителя, а порой и сами организаторы.

Какую цель ставит перед собой Заочная школа современного программировавния?

Как становятся учениками ЗШСП и как в ней учатся?

Какие встречаются ошибочные ожидания у ребят, обращающихся в школу?

Получают ли выпускники школы свидетельства об обучении и когда?

Какие изменения планируются в работе школы на очередной учебный год?

Итак, что же такое «Школа современного программирования». Многие ребята ошибочно полагали, что это еще один способ освоить какой-либо язык программирования (например, в письмах встречались VisualBasic, Delphi, Java). Разумеется, освоение нового языка программирования – дело полезное. Более того, в одном из занятий мы познакомили учеников школы с языком PostScript, на котором мало кто программировал, хотя работа с этим языком происходит практически за каждым компьютером, соединенным с принтером. (Правда работу эту осуществляют, как правило, не люди, а программы).

Языки, по мнению организаторов школы, не должны быть самоцелью, язык нужен программисту, чтобы эффективно решать стоящие перед ним задачи. Но чтобы решать содержательные задачи, нужно знание определенных ключевых идей программирования и математики. Поэтому, главной целью школы является не обучение современным языкам программирования (чем занимаются многочисленные курсы и чему посвящено множество учебных пособий), а обучение классическим и современным идеям и методам программирования, которые составляют «золотой фонд» информатики и лежат в основе профессионального образования современного программиста.

Занятия школы базируются на связях между информатикой и математикой, на тех разделах, которые называют иногда «компьютерной» или «дискретной» математикой и которые пока отсутствуют в школьных учебниках. Например, одно из занятий школы посвящено криптографии – науке очень древней и, благодаря появлению сети Интернет, очень современной.

Большое внимание уделяется разбору олимпиадных задач по информатике, поскольку только при решении сложных задач можно почувствовать изящество тех или иных идей программирования, красоту и эффективность прикладных алгоритмов. Именно большое количество олимпиадных задач и было основной причиной трудностей, возникавших у учеников школы.

Чтобы помочь учащимся в преодолении этих трудностей, с нового учебного года поступающие в ЗШСП ребята получат более сбалансированные по сложности комплекты задач, в заданиях будут выделяться основные и дополнительные задачи. Кроме того, ограничивается возраст поступления в школу: принимаются учащиеся не ниже 7 класса.

Заметим, что трудностей с решением задач было гораздо меньше у «коллективных» учеников - кружков, работающих под руководством школьного учителя, поэтому

администрация школы рекомендует использовать такую форму обучения шире.

Правила работы ЗШСП на следующий учебный год такие:

1) Для тех, кто поступает на первый курс обучения, задания будут высылаться, начиная с октября и с периодичностью 1 раз в месяц до апреля (6 заданий за учебный год); правила приема в школу помещены на информационной вкладке.

2) Для тех, кто продолжает обучение, задания будут публиковаться в журнале (и высылаться в виде брошюр); периодичность занятий – раз в два месяца до декабря (6 занятий за календарный год); работы проверяются по мере поступления.

3) Обучение в школе происходит по следующей схеме: ученику присылается текст лекции и задание в виде брошюры, ученик решает присланные задачи и (если умеет) пишет и отлаживает программы к задачам; письменные решения проверяются студентами университета, а программы пропускаются через систему тестирования, после чего работа с комментариями проверяющего возвращается ученику; кроме этого, ученикам высылаются брошюры с решениями.

4) После окончания обучения в школе, которое ученик может прервать в конце любого курса, выдается свидетельство ЗШСП с указанием всех тем, по которым выполнены задания, вместе с результатами их выполнения; свидельство об обучении «коллективного» ученика передается в учреждение, при котором работал кружок (учреждение на основе свидетельства, а также других сведений об участниках кружка может выдать свое персональное свидетельство каждому из членов кружка).

СОВЕТЫ УЧАЩИМСЯ И ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ РАБОТ

1. Перед решением задач прочтите лекцию и разберите примеры в ней.

2. Приступая к решению задач, помните, что предложенные задачи не обязывают вас использовать только материал соответствующего занятия. У задачи может быть множество разных хороших решений. Могут предлагаться также задачи не по теме – на сообразительность.

3. Теоретические решения (***Уровень 1***) требуют не просто «голого ответа», а объяснений, обоснований и доказательств.

4. Программы (***Уровень 2)*** проверяются автоматической системой тестирования, поэтому необходимо точно следовать указанным в условиях форматам ввода/вывода, а также указанным ниже правилам:
– все программы должны быть скомпилированы под DOS, вводить информацию из файла in.txt и выводить результат в файл out.txt; никаких других файлов программа использовать не должна, если это особо не оговорено в условии; при одинаковых входных данных программа должна выдавать одинаковый результат;
– названия исполнимых файлов с программой: <первые три буквы фамилии><двузначный номер занятия> <двузначный номер задачи>.exe
Например, pet0212.exe (двенадцатая задача второго занятия), sid0102.exe (вторая задача первого занятия);
– все тексты должны быть набраны в формате ASCII.

5. Тексты программ, возможно, будут читаться проверяющими, поэтому используйте поясняющие комментарии и уделяйте внимание структуре программы, а также названиям идентификаторов.

6. Если вы не можете прислать решения в электронном виде, присылайте подробно и аккуратно оформленные решения на бумаге. Если вы пишете программу на бумаге, то перед этим следует словами описать алгоритм решения.

7. Решения принимаются на дискетах и в письменном виде.

Наш адрес: 191025, Санкт-Петербург, ул. Марата 25
Заочная школа современного программирования
Телефон: (812) 164-13-55
E-mail: suvorova@ipo.spb.ru

Павлова Марианна Владимировна

ЗАНЯТИЕ 7.
РЕКУРСИВНЫЕ АЛГОРИТМЫ И ИХ ПОСТРОЕНИЕ

Начиная решать какую-нибудь задачу, нужно сначала подумать, какие действия и в какой последовательности необходимо выполнить, чтобы получить ответ. Эту последовательность действий принято называть *алгоритмом* решения задачи. Пока вы думаете, алгоритм находится в вашей голове и никому, кроме вас, неизвестен. Кроме того, задача может быть сложной, и алгоритм ее решения в этом случае будет длинным и запутанным и может «не поместиться» в голове. Для того, чтобы записать алгоритм, можно использовать разные способы:

а) по пунктам последовательно (и аккуратно) перечислить предполагаемые действия;

б) начертить схему, соответствующую решению данной задачи;

в) записать решение в виде программы на каком-нибудь языке программирования и др.

Первые два способа записи алгоритмов годятся даже для учеников младших классов, которые совсем еще не знакомы с компьютером. В этом случае носителем информации является лист бумаги, но кто же будет исполнителем? Придется выполнять действия одно за другим самому автору (не делая при этом ошибок!). Третий способ хорош тем, что после того, как алгоритм будет записан, выполнение получившейся программы можно поручить компьютеру.

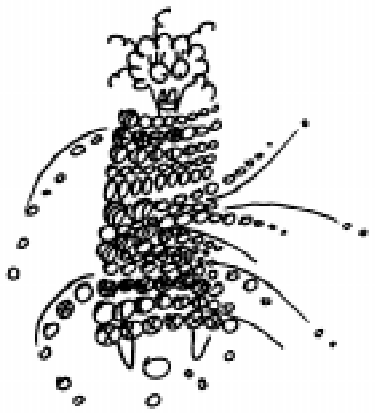
Если при решении задачи вы обнаружили, что задача сводится «сама к себе» в несколько упрощенном виде, значит вы получили рекурсивный вариант решения этой задачи, то есть построили *рекурсивный* алгоритм. Слово **рекурсия** происходит от латинского слова «recursio», что означает **возвращение**.

Рассмотрим простой пример.

У ниточки любимых маминых бус оторвался замочек, и вы решили бусы починить. Для этого вам нужно нанизать бусинки на новую ниточку, снимая их со старой. Ваши действия могут быть описаны так:

1) снять одну бусину и нанизать ее на новую нитку;

2) если на старой нитке бусинок не осталось, то работа завершена, в противном случае вам предстоит выполнить те же действия, только ниточка стала уже более короткой.



В этом примере рекурсивным получился алгоритм, описывающий действия исполнителя.

В следующем примере рассмотрим рекурсивное определение понятия «табун лошадей».

Будем считать, что

1) 5 лошадей – это табун;

2) табун + 1 лошадь – это табун.

Пусть теперь скачет 7 лошадей, и мы хотим узнать, можно ли их назвать табуном.

7 лошадей – это не 5 лошадей, следовательно, по пункту 1) вывод сделать нельзя. Попробуем воспользоваться пунктом 2) . Отделив 1 лошадь от общей группы, получим, 7 лошадей = 6 лошадей + 1 лошадь. Ясно, что если окажется, что 6

лошадей можно назвать табуном, то по пункту 2) определения 7 лошадей будут являться табуном. В свою очередь, 6 лошадей – это не 5 лошадей, так что пункт 1) опять не подходит, но можно отделить 1 лошадь и получить, что 6 лошадей = 5 лошадей + 1 лошадь. Если нам удастся показать, что 5 лошадей – это табун, то и 6 лошадей тоже смогут считаться табуном по пункту 2) нашего определения. Теперь осталось подвести итоги: 5 лошадей – это, конечно, табун по пункту 1), откуда следует, что (7 лошадей) = (6 лошадей + 1 лошадь) = ((5 лошадей + 1 лошадь) + 1 лошадь) = ((табун + 1 лошадь) + 1 лошадь) = (табун + 1 лошадь) = (табун).

Наглядно это рекурсивное определение можно изобразить так:

(_____ ? _____)
(_____ ? _____) + лошадь
(_____ ? _____) + лошадь
(_____табун _____) + лошадь
(_____табун _____) + лошадь
(_____табун_____)

Как мы видели на примерах, в рекурсивных алгоритмах всегда есть одна ветка, которая завершает процесс, и вторая ветка, которая вызывает этот же процесс.

При написании программы, соответствующей рекурсивному решению задачи, нужно убедиться в том, что выбранный язык программирования допускает использование рекурсивных процедур. Такие процедуры можно писать почти на всех современных языках программирования, которые появились после языка АЛГОЛ-60. Мы будем писать программы, используя язык ПАСКАЛЬ, в котором можно описывать как рекурсивные процедуры, так и рекурсивные функции.

Пример 1. Наибольший общий делитель.

В занятии 1 («Компьютерные инструменты в образовании» № 1, 1999 г.) алгоритмы нахождения наибольшего общего делителя двух целых чисел уже рассматривались. Сейчас опишем функцию, которая по двум заданным целым числам найдет их наибольший общий делитель при помощи алгоритма Евклида с делением.

Напомним, что процесс состоит в том, что мы будем находить остаток от деления первого числа на второе ($a \bmod b$). Если остаток окажется равным нулю, то наибольший общий делитель найден (он равен последнему делителю), если же остаток ненулевой, то нужно искать наибольший общий делитель для предыдущего делителя и остатка. Например, НОД (12,8) = НОД (8,4) = = НОД (4,0) = 4.

Если этот алгоритм записать в виде функции в программе, то не имеет значения, будет первое число больше второго или нет. Дело в том, что при нахождении первого остатка, если делимое меньше делителя, они просто поменяются местами. Например, НОД (3,21) = = НОД (21,3) = НОД (3,0) = 3.

Программа на Паскале может быть написана так:

```
Program P1;
Var x,y:integer;
Function NOD(a,b:integer):integer;
begin if b=0
     then NOD:=a
     else NOD:=NOD(b, a mod b)
end;
BEGIN
  writeln('Введите два целых числа
          для поиска их наиб. общего
          делителя');
  readln(x,y);
  writeln('Он равен ',NOD(x,y))
END.
```

При выполнении этой программы компьютер выведет на экран подсказку и будет ждать, когда пользователь введет два числа. Затем будет напечатан поясняющий текст и вызовется функция NOD, результат которой также будет напечатан. Для работы этой функции в качестве первого параметра будет передано первое из введенных чисел (оно будет играть роль делимого), а в качестве второго параметра – второе число (делитель).

Проверка $b = 0$ застраховывает нас от деления на ноль при первом вызове, а в дальнейшем дает ту ветку, по которой рекурсивные вызовы завершаются.

Теперь можно подвести первые итоги и дать некоторые определения.

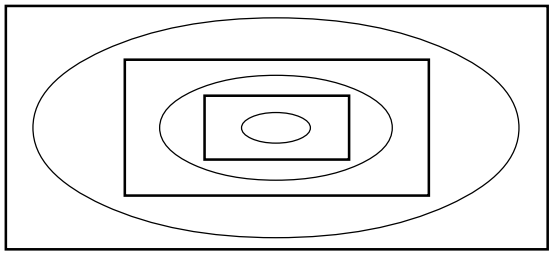
Мы узнали, что существуют алгоритмы, которые обращаются сами к себе конечное число раз. Такие алгоритмы называются рекурсивными алгоритмами и реализуются в программах в виде рекурсивных процедур или рекурсивных функций. При этом рекурсивным может быть *процесс* обработки данных (вспомним про мамины бусы) или сами *данные* (вспомним про табун лошадей).

Для того, чтобы рекурсивная процедура завершала свою работу после какого-то конечного числа вызовов, нужно, чтобы в самой процедуре было условие, при выполнении которого вызов описываемого процесса прекращается (рекурсивно завершающая ветка). Естественно, что это условие должно быть таким, что оно обязательно выполнится после конечного числа обращений к данной процедуре. Таким образом, получается, что однократное обращение к рекурсивной процедуре порождает многократный вызов ее из нее самой. Количество таких вызовов называется *глубиной* рекурсии.

В примере, где находится НОД (12,8), глубина рекурсии равна 4.

До сих пор мы имели дело с одним из видов рекурсии, когда процедура вызывала саму себя до тех пор, пока процесс не завершался. Такую рекурсию часто называют *прямой рекурсией*.

Однако бывают случаи, когда процедура вызывает себя не прямо, а через другую процедуру или цепочку других процедур. Такую рекурсию обычно называют *косвенной*. В этом случае все участвующие в вызовах процедуры будут рекурсивными. Если процедуру «А» представлять прямоугольником, а процедуру «В» – эллипсом, то случай их косвенной рекурсивности можно изобразить такой схемой:



Мы пока будем рассматривать только прямую рекурсию.

Пример 2.

В занятии 2 («Компьютерные инструменты в образовании» № 2, 1999 г.) предлагалась задача по переводу заданного в десятичной системе целого неотрицательного числа в двоичное число. Для решения этой задачи рекурсия оказывается как нельзя кстати. Действительно, мы должны последовательно выписывать остатки от деления на 2 сначала самого числа, затем частного от его деления на 2 и т.д. Сложность в том, что эти остатки должны распечатываться в обратном порядке по отношению к тому, как они получаются. Рекурсивный алгоритм позволяет «задержать» печать до тех пор, пока в частном не останется 1, а потом напечатать эту единицу и все остатки в обратном порядке.

Алгоритм печати двоичного представления заданного N можно записать так:
1) Если $N = 0$ или $N = 1$, то напечатать N, иначе
2) Найти двоичное представление числа (N div 2). Напечатать (N mod 2).

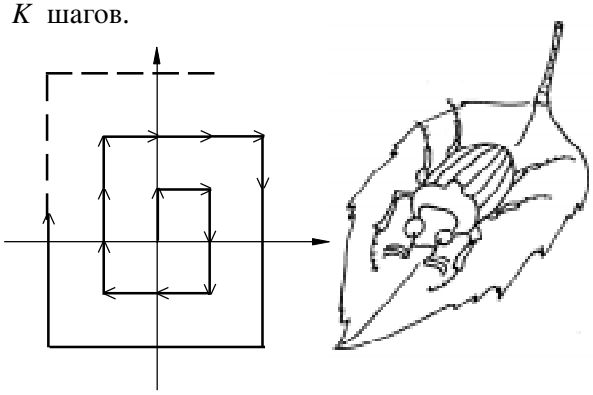
Если этот алгоритм записывать процедурой на Паскале, то она будет выглядеть так:

```
Procedure BIN(N:integer);
begin if (N=0) or (N=1)
      then write(N)
      else begin BIN(N div 2);
           write(N mod 2)
           end
end;
```

Пример 3.

Задача. Жук сидит на плоскости в некоторой точке с целочисленными координатами. За единицу времени жук переползает в ближайшую к нему точку с целочисленными координатами, не сползая с заданной ломаной, по направлению, указанному на схеме стрелками.

Уровень 1. Построить рекурсивный алгоритм, который опишет путь жука за K шагов.



Решение:

1) Если $K = 0$, то процесс закончен;
2) Если $K > 0$, то

а) сделать 1 шаг:

при $Y \le X$ и $Y > -X$ шагнуть вниз;
при $Y \le -X$ и $Y < X$ шагнуть влево;
при $X \le Y$ и $Y < -X+1$ шагнуть вверх;
при $Y \ge -X+1$ и $Y > X$ шагнуть вправо;

б) положить $K = K - 1$ и повторить процесс (Перейти на 1)).

Уровень 2. Описать рекурсивную процедуру, которая определит, где окажется жук через K шагов.

Ясно, что параметрами этой процедуры будут являться начальные координаты жука и количество шагов K. При этом параметры-координаты будут меняться на каждом шаге и после окончания работы будут иметь уже не те значения, которые они имели в начале программы, следовательно, эти параметры должны быть параметрами-переменными. Параметр, показывающий количество шагов, которые осталось сделать жуку, после каждого вызова уменьшается на единицу, а затем процедура вызывается снова. Этот параметр является параметром-значением, а процесс закончится, когда окажется, что жуку нужно сделать ноль шагов. Процедура может быть описана так:

```
Procedure BEETLE (var x,y:integer;
k:integer);
begin if k>0 then
 begin
  if (y<=x) and (y>-x)
  then y:=y-1
  else if (y<=-x) and (y<x)
       then x:=x-1
       else if (y>=x) and (y<-x+1)
            then y:=y+1
            else x:=x+1;
  BEETLE(x,y,k-1)
 end
end.
```

В программе нужно будет ввести исходные координаты жука и интересующее нас число шагов, вызвать процедуру BEETLE, а затем напечатать новые координаты жука. Обратите внимание на то, что здесь явно не выписана рекурсивно завершающая ветка, просто при выполнении условия $K = 0$ вызовы прекращаются, и работа процедуры заканчивается.

Теперь попробуем представить себе, какие действия происходят в компьютере, когда вызывается рекурсивная процедура. Для наглядности рассмотрим простой пример.

Пример 4.

Определение. Факториалом натурального числа N называется произведение всех целых чисел от 1 до N.

Факториал принято обозначать как N!. В математике функцию, вычисляющую факториал числа N, записывают так:

$$F(N) = \begin{cases} 1, \text{ при } N = 1, \\ (N-1)\cdot\ldots\cdot 1, \text{ при } N > 1. \end{cases}$$

Поскольку в последней строке произведение $(N-1)\cdot\ldots\cdot 1$ есть $(N-1)!$, то предыдущую запись можно переписать так:

$$F(N) = \begin{cases} 1, \text{ при } N = 1, \\ N \cdot F(N-1), \text{ при } N > 1. \end{cases}$$

Теперь хорошо видно, что определение факториала рекурсивно, и для его вычисления в программе может быть описана соответствующая рекурсивная функция.

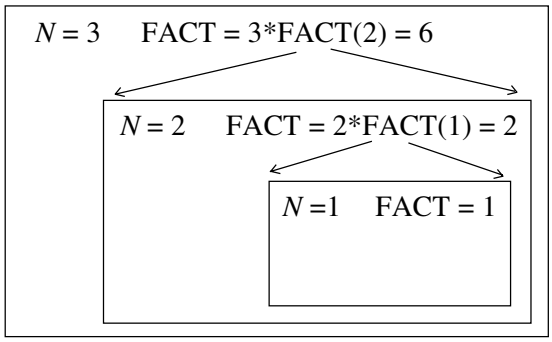
```
Program FACTORIAL;
Var N: integer;
Function FACT(N:integer):integer;
begin if N=1
     then FACT:=1
     else FACT:=N*FACT(N-1)
end;
BEGIN
writeln('Введите натуральное число,
       факториал которого вы хотите
        узнать');
readln(N);
writeln('Факториал ' ,N, ' равен ',
         FACT(N))
END.
```

Во-первых, необходимо отметить, что значение факториала можно вычислить на компьютере только для небольших значений N. Поскольку диапазон для целых чисел ограничен сверху числом 32767, а 8!=40320, то 8! уже не помещается в отведенный диапазон. Можно, конечно, описывать результат этой функции как WORD, тогда границей будет число 65535, но и это даст возможность вычислить лишь 8!, а 9!=362880 уже не поместится в отведенную разрядную сетку. По этой причине данный пример следует рассматривать как учебный, на котором нам будет удобно обсудить вопрос о вложенных рекурсивных вызовах.

Пусть мы хотим получить значение факториала при $N = 3$.

Вызвав FACT(N) при $N = 3$ компьютер попытается вычислить значение формулы 3*FACT(2), что не может быть сделано, пока не известно значение FACT(2), поэтому компьютер «возьмется» за нахождение этого значения. Для этого ему придется второй раз войти в функцию FACT с параметром $N = 2$. Теперь он временно «забудет» старое значение $N = 3$ и будет работать с $N = 2$. Для того, чтобы иметь возможность впоследствии «вспомнить» старое значение, будет отведена новая память под параметр N, куда и занесется значение 2. Попытка вычислить FACT(2) приведет к формуле 2*FACT(1), следовательно, придется опять прерывать вычисления, чтобы вычислить значение FACT(1). Компьютер в третий раз входит в функцию, опять отводит новую память под параметр N, и в эту память заносит число 1. На этот раз выполнится условие завершающей ветки и получится, что FACT(1) = 1. В этот момент последний вызов будет закрыт, последнее $N = 1$ будет «забыто», но «вспомнится», что перед этим было $N = 2$ и можно досчитать значение формулы FACT(2) = 2*FACT(1), что даст значение 2, которое сразу же будет использовано в первой прерванной формуле FACT(3) = 3*FACT(2) и даст нам окончательное значение 3*2 = 6.

Схематически эту последовательность вложенных вызовов можно изобразить, например, так:



Подробно разбирая этот пример, мы обязательно должны были отметить один очень важный факт: рекурсивные процедуры и функции коротко записываются, отражая математическую сущность алгоритма, делают программу красивой и наглядной, но при исполнении требуют большого объема памяти, особенно, если в процедуре много параметров и локальных переменных, а также, если глубина рекурсии большая. Именно поэтому в каждом конкретном случае нужно решать, стоит ли использовать рекурсивный алгоритм или постараться обойтись другими средствами, например, операторами цикла. В тех случаях, когда сами данные име-

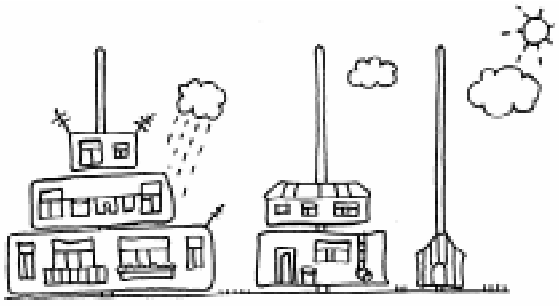
ют рекурсивную структуру, рекурсивные алгоритмы предпочтительнее.

Тем не менее, сделав это важное замечание, мы продолжим знакомство с рекурсивными алгоритмами и рассмотрим следующий пример.

Пример 5.

Задача. Каждый из вас видел детскую игрушечную пирамидку. Она состоит из одного колышка и нескольких дисков разного диаметра. Обычно, ребенка просят собрать все диски на колышек так, чтобы снизу вверх размеры дисков уменьшались. Этому правилу будем следовать и мы, только условия будут несколько иными. У нас будет три колышка (пронумеруем их числами 1, 2 и 3). На одном из колышков (пусть это будет колышек 1) находится правильно собранная пирамидка из N дисков. Задача состоит в том, чтобы перенести всю пирамидку на другой колышек, используя третий колышек в качестве вспомогательного. За один ход можно перекладывать с одного колышка на другой только один диск, и никогда больший диск нельзя класть на меньший.

Эта задача впервые была сформулирована в виде головоломки французским математиком Эдуардом Люка в 1883 году. Он предлагал переложить пирамидку из 8 дисков на один из пустых колышков, соблюдая перечисленные выше правила. Эту головоломку Люка назвал Ханойской башней, связывая свою игрушку с мифической легендой о башне Брамы, которая состоит из 64 золотых дисков и трех алмазных шпилей. По легенде Всевышний при сотворении мира поместил башню на первый шпиль и велел жрецам переместить башню на третий шпиль в соответствии с указанными правилами. В легенде утверждается, что жрецы «денно и нощно» трудятся до сих пор, но, когда они закончат работу, башня рассыплется, и наступит конец света. Мы не будем, конечно, выполнять работу жрецов, но постараемся построить алгоритм перемещения башни из N дисков с одного колышка на другой. При этом на каждом шаге будем печатать номера тех двух колышков, откуда берем диск и куда его кладем.



Уровень 1. Описать по шагам алгоритм переноса пирамиды.

Основная идея состоит в том, что для переноса самого нижнего диска на другой колышек этот колышек должен быть пустым, так как нижний диск имеет максимальный диаметр и ни на какой другой диск положен быть не может. Следовательно, к моменту переноса последнего диска должны выполняться следующие условия: вся пирамида высоты $N-1$ должна находиться в собранном виде на вспомогательном колышке; колышек, где будет окончательно собираться пирамида, должен быть пустым; на первом колышке должен лежать один нижний диск.

Тогда алгоритм может быть записан так:

1) Если $N = 0$, то ничего делать не нужно;
2) Если $N \geq 1$, то

a) перенести все диски, кроме последнего (самого нижнего), с колышка 1 на колышек 3;

b) переложить последний диск с колышка 1 на колышек 2;

c) перенести все диски с колышка 3 на колышек 2.

Ясно, что в пунктах a) и c) действия осуществляются над $N-1$ диском, то есть эти пункты содержат рекурсивное обращение к процессу переноса пирамиды высоты $N-1$.

Уровень 2. Написать программу, осуществляющую перенос пирамиды.

Прежде чем написать программу, соответствующую построенному на уровне 1 алгоритму, заметим, что в процедуре, осуществляющей перенос, нужно уметь задавать высоту переносимой пирамиды (N)

(при $N = 0$ процесс заканчивается), а также номера колышков, показывающих, откуда и куда переносится пирамида. Обозначим через a, b и c параметры, соответствующие номерам колышков. Эти параметры могут принимать значения 1, 2 или 3. Кроме того, на каждом уровне вызова все эти параметры должны иметь разные значения, а также должно выполняться соотношение $a+b+c = 1+2+3 = 6$, то есть $c = 6–a–b$.

Учитывая все сказанное, процедуру переноса можно написать так:

```
Procedure Motion(N:integer;
                 a,b:integer);
begin if N>0
  then begin Motion(N-1, a, 6-a-b);
             Motion(1, a, b);
             Motion(N-1, 6-a-b, b)
       end
end.
```

Здесь использована одна процедура и для переноса пирамиды и для переноса одного диска. В ней нет оператора, который сообщал бы нам о том, как производился перенос. Напишем отдельную процедуру для переноса одного диска, включив в нее печать сообщения о переносе.

```
Procedure Print( a, b: integer );
begin writeln('Переносим с ',a,
              ' на ', b)
end;
```

Теперь можно написать всю программу.

```
Program HANOI;
Var N:integer;
Procedure Print( a, b: integer );
begin writeln('Переносим с ',a,
              ' на ', b)
end;
Procedure Motion(N:integer;
                 a,b:integer );
begin if N>0
  then begin Motion(N-1, a, 6-a-b);
             Print(a, b);
             Motion(N-1, 6-a-b, b)
       end
end;
BEGIN Writeln('Введите количество
              дисков в пирамиде');
    Read(N);
    Motion(N,1,2)
END.
```

Чтобы завершить разговор о Ханойской башне, полезно было бы решить вопрос о том, сколько всего переносов отдельных дисков необходимо сделать, чтобы переместить всю пирамиду из N дисков с одного колышка на другой. (Почему так долго возятся со своей работой жрецы?). Из наших рассуждений и описанной рекурсивной процедуры видно, что при $N = 0$ искомое число $K = 0$, так как ничего переносить не нужно, а при $N > 0$ придется два раза перенести пирамиду из N - 1 диска и еще переложить один диск. Оказывается, что более эффективного способа переноса не существует, а владеющие аппаратом доказательства методом математической индукции без труда получат формулу $K = 2^N – 1$, при любом неотрицательном N.

На этом заканчивается не только разбор задачи о Ханойской башне, но и наш сегодняшний разговор о рекурсии. Далее мы продолжим изучение этой интересной и полезной темы в процессе решения задач.

Литература.

1. Д. Баррон «Рекурсивные методы в программировании», М.: Мир, 1974.
2. Б. Мейер, К. Бодуэн «Методы программирования» т.2, М.: Мир, 1982.
3. Н. Вирт «Алгоритмы + структуры данных = программы», М.: Мир, 1985.
4. М.В. Дмитриева, А.А. Кубенский «Элементы современного программирования», Изд. СП Университета, 1991.
5. Р. Грэхем, Д. Кнут, О. Паташник «Конкретная математика», М.: Мир, 1998.

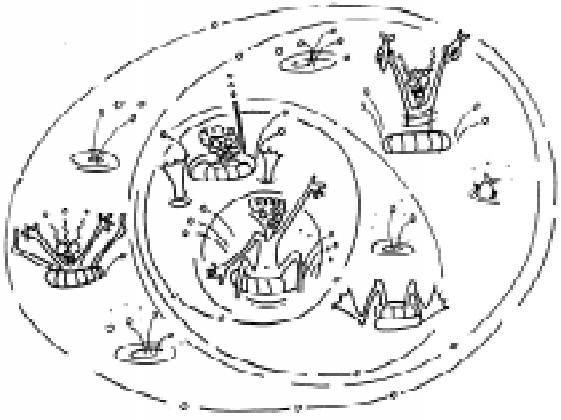
Павлова Марианна Владимировна
Паньгина Нина Николаевна

ПРИМЕРЫ И ЗАДАЧИ НА ТЕМУ «РЕКУРСИВНЫЕ АЛГОРИТМЫ И ИХ ПОСТРОЕНИЕ»

Помните детское стихотворение-считалку про 10 негритят? Оно может служить прологом к любой работе на тему «Рекурсия».

10 негритят пошли купаться в море,
10 негритят резвились на просторе,
Один из них пропал – и вот вам результат:

9 негритят пошли купаться в море,
9 негритят резвились на просторе,
Один из них пропал – и вот вам результат:
... ...
1 (из) негритят пошли(ел) купаться в море,
1 (из) негритят резвились(ся)на просторе,
Один из них пропал – и вот вам результат:
Нет больше негритят!



Первые три строчки этого стихотворения повторяются 10 раз с небольшим изменением - число негритят уменьшается с каждым разом на единицу. И только, когда число негритят уменьшилось до нуля, стихотворение заканчивается единственной строчкой «Нет больше негритят!». Попробуем написать рекурсивную процедуру, печатающую это стихотворение.

```
Procedure Negr(k: integer);
{k-число негритят, параметр процедуры}
Begin
If k=0   {проверка, что число
негритят равно нулю}
then writeln('Нет больше негритят!')
{выход из рекурсии}
else begin
          writeln(k,' негритят пошли
                купаться в море,');
          writeln(k,'  негритят
          резвились на просторе,');
          writeln('Один и них пропал
          - и вот вам результат:');
          Negr(k-1);
{Вызов процедуры с уменьшенным
на 1 параметром}
     end
End;
```

Вызов этой процедуры в основной программе будет выглядеть так: Negr(10).

Интересным является применение рекурсии при создании рисунков.

Пример 1.

Рассмотрим простейший рисунок из окружностей разных радиусов (рисунок 1).

Если вглядеться в него внимательно, то можно заметить, что рисунок начинается с центральной окружности самого

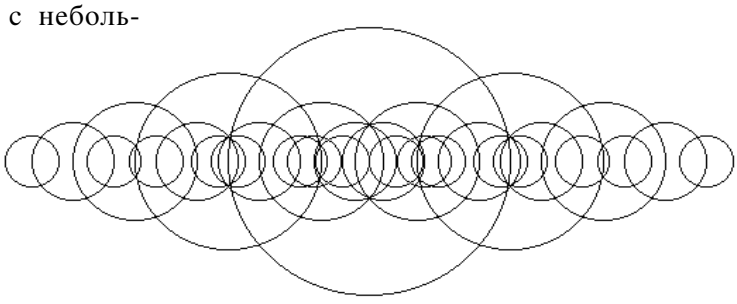


Рисунок 1.

большого радиуса. Затем осуществляется переход на концы горизонтального диаметра окружности, которые должны играть роль центров двух окружностей меньшего радиуса (примерно в полтора раза). Этот же процесс повторяется и с этими двумя окружностями, и с полученными четырьмя новыми, и так далее до тех пор, пока уменьшающийся радиус окружности не станет меньше первоначального в 1,5·1,5·1,5·1,5 раз (если посчитать, то должно выполниться четыре вложенных вызова рекурсивной процедуры).

Рекурсивная процедура, выполняющая такой рисунок, должна иметь в качестве передаваемых параметров координаты центра окружности и величину радиуса.

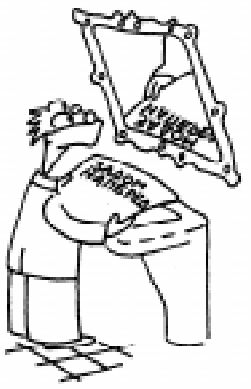
Программа, при помощи которой будет нарисован этот рисунок, может быть написана так:

```
Program RIS1;
Uses Graph;
Var x,y,r,gd,gm:integer;
Procedure Ris(x,y,r:integer);
begin
If r>=10
     then begin
          Circle (x,y,r);
          Ris(x-r,y,r*2 div 3);
          Ris(x+r,y,r*2 div 3)
          end
end;
BEGIN
gd:=detect;
gm:=1;
InitGraph(gd,gm,'');
     Ris(320,240,100);
Readln;
CloseGraph;
END.
```

Пример 2.

На входном потоке находится слово (последовательность литер), заканчивающаяся пробелом. Написать программу, которая позволит напечатать это слово «задом наперед», то есть выписывая буквы слова с конца.

В этом примере мы опишем процедуру, у которой не будет параметров, так как при каждом ее вызове будет вводиться одна литера, которая будет сравниваться с пробелом. Сама же программа будет состоять только из вызова этой процедуры.



```
Program REVERSE;
Procedure REV;
Var c:char;
  begin read(c);
  If c<>' '
  then begin REV; write(c)
       end
   end;
BEGIN REV
END.
```

Теперь переходим к более сложным задачам.

Пример 3.

Фракталами называются множества, части которых являются повторением образов самих множеств. Изображения фракталов вызывают обычно у всех большой интерес.

Рассмотрим процесс сгибания бумажной полоски: если взять полоску бумаги, согнуть ее пополам k раз и развернуть полоску так, чтобы углы на сгибах стали равны 90°, то, посмотрев на торец полоски, можно увидеть ломаную, которая называется «драконовой» ломаной k-го порядка, где k – количество сгибов. Схема этого процесса изображена на рисунке 2.

Опишем способ создания драконовой ломаной.

На ломаной нулевого порядка, как на гипотенузе, строим прямой угол, на по-

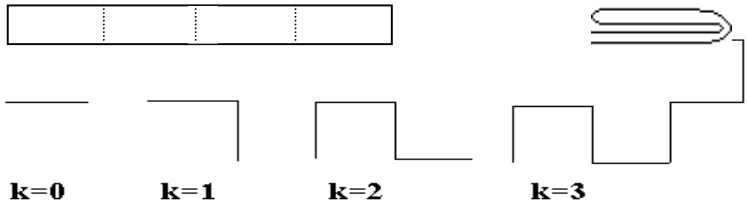
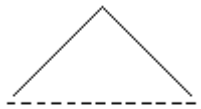
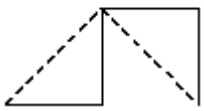


Рисунок 2.

лученных сторонах прямого угла строим тоже прямые углы, но один развернут в правую сторону, а другой - в левую. Данный процесс повторяется k раз.



Напишем соответствующую программу, которая по заданному числу k рисует «драконову» ломаную k-го порядка.



```
Program DRACON;
Uses Graph;
Var Gd,Gm,k:integer;
Procedure st (x1, y1, x2, y2,
k:integer);
var xn,yn:integer;
begin
If k > 0 then
begin
  xn:=(x1+x2)div 2 + (y2-y1)div 2;
  yn:=(y1+y2)div 2 - (x2-x1)div 2;
  st(x1,y1,xn,yn,k-1);
  st(x2,y2,xn,yn,k-1);
end
else Line(x1,y1,x2,y2);
end;
BEGIN
 Gd:=Detect; Gm:=1;
 InitGraph(Gd,Gm,'');
 writeln ('Введите номер уровня ');
 readln(k);
 st(200, 300, 500, 300, k);
 readln;
 CloseGraph;
END.
```

Если с помощью этой программы построить ломаную дракона 14-го порядка, то мы получим образ множества, называемого фракталом Хартера-Хейтуэя, рисунок которого приведен на рисунке 3.

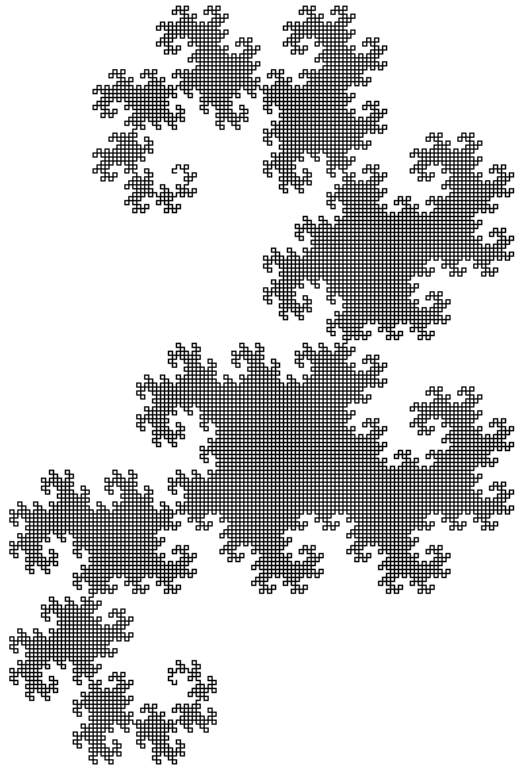


Рисунок 3.

ЗАДАЧИ.

Задача 1.

Возвести заданное вещественное число X в заданную натуральную степень N. В этой задаче совсем легко найти формулу: $X^N = X \cdot X^{N-1}$, но такой алгоритм потребует N-1 повторения процесса. Гораздо более эффективным будет следующий алгоритм:

$$X^N = \begin{cases} X, & \text{если } N = 1 \\ (X^{N/2})^2, & \text{если } N - \text{четно} \\ (X^{(N-1)/2})^2, & \text{если } N - \text{нечетно} \end{cases}$$

Уровень 1. Описать по шагам алгоритм «быстрого» возведения в степень. Продемонстрировать его работу при $X = 2$, $N = 11$.

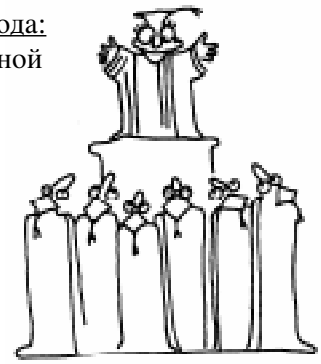
Уровень 2. Описать рекурсивную функцию «быстрого» возведения в степень. Написать программу, которая возводит заданное вещественное число в натуральную степень.

Формат ввода:

Число X

Число N

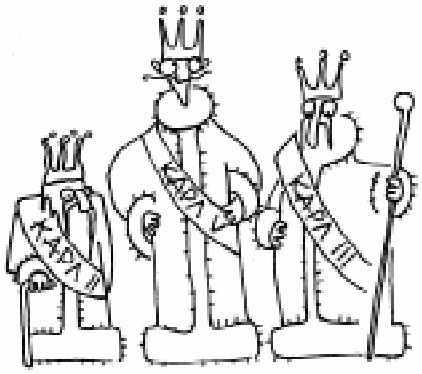
Формат вывода:
Значение полученной степени



Пример.
Ввод:
2
4
Вывод:
16

Задача 2.

Последовательность чисел 0, 1, 1, 2, 3, 5, ..., где каждое число, начиная с третьего, есть сумма двух предыдущих чисел последовательности, называется последовательностью чисел Фибоначчи.



При заданном *N* требуется найти первые *N* чисел Фибоначчи.

Уровень 1. Описать алгоритм, который для заданного *N* получит число Фибоначчи с номером *N*. Вычислить тринадцатое число Фибоначчи.

Уровень 2. Описать рекурсивную функцию, которая выдает значение *k*-ого числа Фибоначчи.

Примечание: Обратить особое внимание на неэффективность использования формулы F(*k*)=F(*k*-1)+F(*k*-2), поскольку в вызове F(*k*-1) содержится вызов F(*k*-2) и т.д., что порождает многочисленные повторные вычисления одного и того же числа.

Формат ввода:
Число *N*

Формат вывода:
Значение *N*-ого числа Фибоначчи

Пример.
Ввод:
9
Вывод:
21

Задача 3.

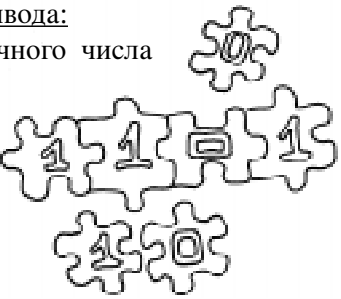
Во входном потоке находится последовательность литер, которые могут быть только символами «0» или «1». Пробел - признак конца последовательности. Любая другая литера - ошибка. Нужно найти десятичное число, равное двоичному числу, заданному этой последовательностью.

Уровень 1. Построить по шагам алгоритм перевода двоичного числа в десятичное число. Найти десятичное представление для последовательностей: 100, 110001, 10000011.

Уровень 2. По заданной последовательности найти десятичное число, описав соответствующую рекурсивную функцию.

Формат ввода:
Последовательность нулей и единиц

Формат вывода:
Значение десятичного числа



Пример.
Ввод:
10111
Вывод:
23

Задача 4.

Уровень 2. Во входном потоке находится последовательность литер, которые могут быть только цифрами. Пробел - признак конца последовательности. Любая другая литера - ошибка. «Собрать» десятичное число из цифр этой последовательности, описав рекурсивную функцию.

Формат ввода:
Последовательность литер, являющихся цифрами

Формат вывода:
Значение полученного числа

Пример.

Ввод:

4521

Вывод:

4521

В примере 1 мы подробно разобрали рекурсивный алгоритм создания рисунка из окружностей. Теперь можно рассмотреть более сложные, но в то же время и более красивые рисунки, например, «веточки» (рисунок 4 и рисунок 5) и «снежинки» (рисунок 6 и рисунок 7).

При создании таких рисунков «из реальной жизни» можно проявить настоящее творчество, например, нарисовать:

– снежинки, различные по форме или падающие по всему экрану;

– веточки разного цвета, разной пушистости и разного размера (в длину ветки, угол наклона и цвет вносится случайная составляющая, рисунок 5);

– а если на последнем уровне вложенности веточки рисовать желтые кружочки, то из голой зимней ветки можно получить пушистую весеннюю мимозу.

Задача 5.

Уровень 1. Описать по шагам, как нарисовать веточку, изображенную на рисунке 4.

Уровень 2. Написать программу, рисующую на экране дисплея веточку, подобную изображенной на рисунке 4.

Формат ввода:

Входных данных нет

Формат вывода:

Рисунок веточки

Задача 6.

Уровень 1. Описать по шагам, как нарисовать веточку, изображенную на рисунке 5.

Уровень 2. Написать программу с использованием случайных составляющих параметров, рисующую веточку, подобную изображенной на рисунке 5.

Формат ввода:

Входных данных нет

Формат вывода:

Рисунок веточки

Задача 7.

Уровень 1. Описать по шагам, как нарисовать снежинку, показанную на рисунке 6.

Уровень 2. Написать программу, рисующую снежинку, изображенную на рисунке 6.

Формат ввода:

Входных данных нет

Формат вывода:

Рисунок снежинки

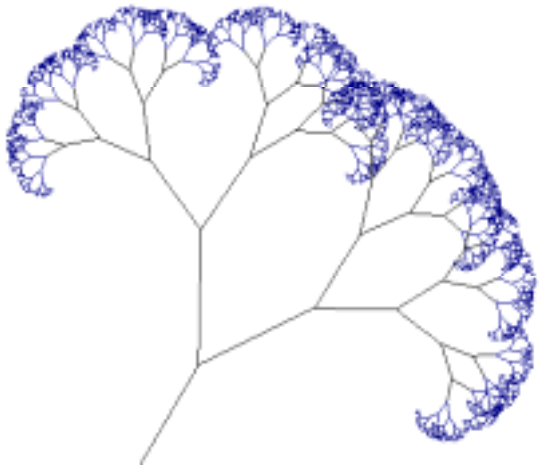


Рисунок 4.

Рисунок 5.

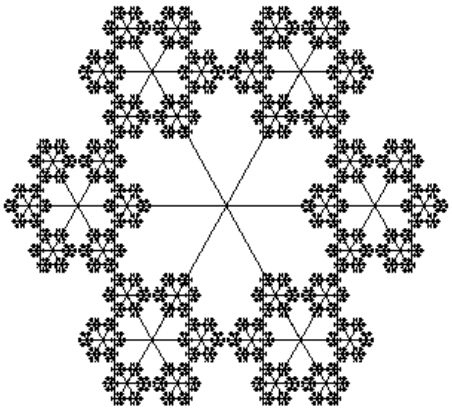


Рисунок 6.

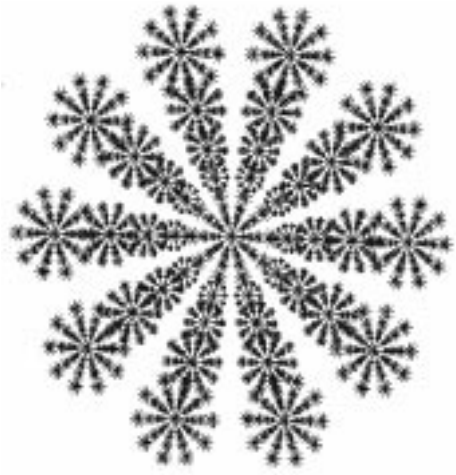


Рисунок 7.

Задача 8.

Уровень 1. Описать по шагам, как нарисовать снежинку, показанную на рисунке 7.

Уровень 2. Написать программу, рисующую снежинку, изображенную на рисунке 7.

Формат ввода:

Входных данных нет

Формат вывода:

Рисунок снежинки

Задача 9.

Для следующей задачи рассмотрим пример из области биологии. С помощью рекурсии можно построить модель Бенуа Мандельброта человеческого легкого, а именно, иерархию трахей и бронхов.

Указание. Из рисунка 8 видно, что третья ветвь составляет по длине примерно половину первой ветви. Уменьшается также толщина ветвей-сосудов.

Уровень 2. Написать программу, рисующую модель легкого, приведенную на рисунке 8.

Формат ввода:

Входных данных нет

Формат вывода:

Графическое изображение легкого

Задача 10.

На рисунке 9 представлена салфетка Серпинского. Как она образуется? Рисуется треугольник и в нем средние линии. В образованных при углах исходного треугольника новых треугольниках

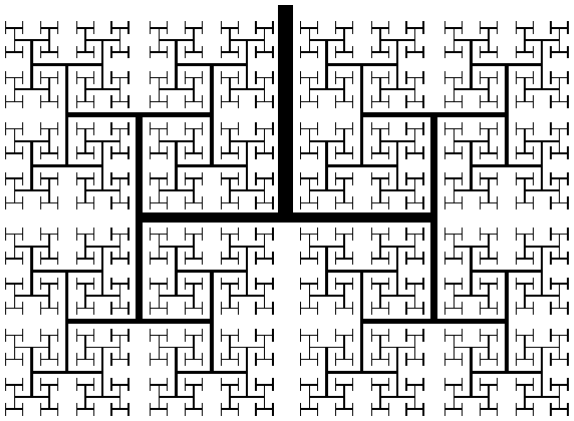


Рисунок 8.

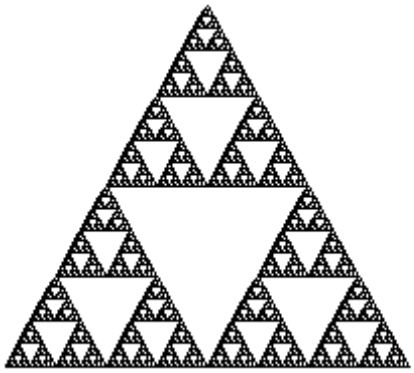


Рисунок 9.

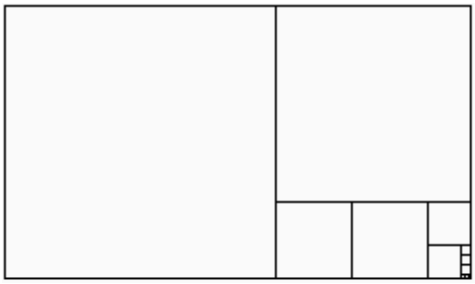


Рисунок 10.

опять рисуются средние линии и т.д. до заданного порядка вложенности. Интересно, что полученная фигура допускает другое (не рекурсивное) построение с помощью моделирования методом Монте-Карло (но об этом в другой раз).

Уровень 2. Написать программу, рисующую салфетку Серпинского, приведенную на рисунке 9.

Формат ввода:
Входных данных нет
Формат вывода:
Изображение салфетки Серпинского

Задача 11.

Уровень 2. Рассмотрим задачу о разрезании прямоугольника с натуральными длинами сторон a и b на квадраты максимальной площади (задача на использование алгоритма Евклида). Предлагается не просто решить эту задачу, но и сделать наглядное графическое сопровождение с помощью рекурсии (рисунок 10).

Указание. Параметрами рекурсивной процедуры являются координаты вершин диагонали прямоугольника (x_1,y_1) и (x_2,y_2). Как только стороны прямоугольника становятся равными, то есть $(x_2–x_1 = y_2–y_1)$, осуществляется выход из процедуры, если же нет, то вычисляется длина меньшей стороны k (это и будет сторона квадрата), и вдоль большей стороны отрезается (рисуется) квадрат. Затем происходит обращение к рекурсивной процедуре с измененными параметрами, либо, вместо x_1, берется x_1+k (если сторона по оси x больше), либо вместо y_1 берется y_1+k (если сторона по оси y больше).

Формат ввода:
Два целых числа через пробел
Формат вывода:
Изображение схемы деления

Далее приводится несколько задач повышенной сложности для программирования.

Задача 12.

Во входном потоке последовательность литер, заканчивающаяся пробелом. При помощи рекурсивной логической функции определить, является ли эта последовательность правильным идентификатором. (Правильным идентификатором считается любая конечная последовательность букв или цифр, начинающаяся с буквы).

Формат ввода:
Последовательность букв и/или цифр
Формат вывода:
«Последовательность является правильным идентификатором» или
«Последовательность не является правильным идентификатором»

Пример.
Ввод:
Max12
Вывод:
Последовательность является правильным идентификатором

Задача 13.

Во входном потоке последовательность литер, признаком конца которой является знак равенства. В самой последовательности могут встречаться только цифры и знаки «+». При этом вся последовательность представляет собой формулу сложения однозначных целых чисел. Описать рекурсивную функцию, которая найдет значение этой формулы или сообщит об ошибке.

Формат ввода:

Чередующаяся последовательность цифр и знаков «+», заканчивающаяся знаком «=»

Формат вывода:

Значение суммы

Пример.

Ввод:

1+3+6+2 =

Вывод:

12

Задача 14.

Во входном потоке последовательность литер, признаком конца которой является знак «=». В самой последовательности могут встречаться только цифры и знаки «+» или «–». При этом вся последовательность представляет собой формулу арифметической суммы однозначных целых чисел. Описать рекурсивную функцию, которая найдет значение этой формулы или сообщит об ошибке.

Формат ввода:

Чередующаяся последовательность цифр и знаков «+» и «–», заканчивающаяся знаком «=»

Формат вывода:

Значение суммы

Пример.

Ввод:

2+5–7+1+9 =

Вывод:

10

Задача 15.

Во входном потоке последовательность литер, признаком конца которой является «=». В самой последовательности могут встречаться только цифры и знаки «+» или «–». При этом вся последовательность представляет собой формулу арифметической суммы произвольных целых чисел. Описать рекурсивную процедуру или функцию, которая найдет значение этой формулы или сообщит об ошибке.

Формат ввода:

Последовательность цифр и знаков «+» или «–», заканчивающаяся знаком «=»

Формат вывода:

Значение суммы

Пример.

Ввод:

45+6–35+4–17 =

Вывод:

3

На дискетке, приложенной к журналу, вы найдете файл Hanoi.exe, который содержит демонстрационно-игровую программу для решения задачи «Ханойские башни».

Павлова Марианна Владимировна, старший преподаватель кафедры информатики СПбГУ.

Паньгина Нина Николаевна, преподаватель ОИ и ВТ лицея № 8 г. Сосновый Бор.