УДК [004.421+004.438](076.5)

МЕТОДИКА БЫСТРОГО ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ НА ОСНОВЕ ИЗУЧЕНИЯ КЛАССОВ ЗАДАЧ (6–7)

***Юрий Александрович Аляев**, доц., доц. кафедры математики и естественно-научных дисциплин,*
E-mail: alyr1@yandex.ru,
Российская академия народного хозяйства и государственной службы при Президенте РФ (Пермский филиал),
http://perm.ranepa.ru

Предлагается методика быстрого обучения программированию на основе изучения классов задач, разработанная и применяющаяся на практике в процессе обучения программированию студентов вузов.

Ключевые слова: алгоритм, программа, язык программирования Паскаль, массив, ветвящиеся Паскаль-программы, циклические Паскаль-программы.

Введение

Разделы курса «Информатика» – алгоритмизация и программирование – остаются наиболее важными для формирования алгоритмического мышления. Поскольку в школах данные разделы преподаются в недостаточном объеме, в вузе возникает необходимость начинать обучение с нуля и достичь хорошего уровня программирования при ограниченном количестве часов преподавания.


Ю.А. Аляев

Этого удается добиться за счет применения рациональных методов обучения, прежде всего, последовательно проводя идеи обучения на основе выделения элементарных операций деятельности по построению алгоритмов и программ; выявления структуры алгоритма и форм ее записи на алгоритмическом языке; одинаковой формы алгоритма для решения задач с одинаковой структурой исходных данных [1–2]. Благодаря этим идеям, задачи по программированию удается разбить на ряд классов и типизировать методы решения задач каждого класса.

Предлагаемая методика быстрого обучения программированию на основе изучения классов задач, появилась и применяется на протяжении многих лет в процессе обучения программированию студентов пермских вузов благодаря В.П. Гладкову [2]. В статье рассматриваются методики решения по двум (6–7) из девятнадцати выделенных классов задач (1–5 классы задач рассмотрены в [3–4]).

6 Ветвящиеся Паскаль-программы

Задача 1. Написать программу по следующему алгоритму:

1. Начало;
2. Список данных: x, y, p, a, b – вещ.;
3. Ввод(x,y,p);
4. Вывод(x,y,p);
5. Если $x \le y \cdot p$ То
5.1. a:=y/p;
5.2. b:=x·y;
Иначе
5.3. $a:=e^{(y-x)/2}$;
5.4. b:=ln(y+p);
6. Конец-Если 5;

7. Вывод(a,b);
8. Конец.

Решение. Для написания программы используйте таблицу перевода алгоритмических конструкций в конструкции языка Паскаль [1]. Тогда оператор «Начало» алгоритма следует заменить в программе оператором заголовка программы:

program z1; Здесь z1 – имя программы.

Второй оператор алгоритма заменяется оператором описания переменных: var x,y,p,a,b:real;

За разделом описания переменных размещается раздел операторов, начинающийся со слова: begin.

Первым оператором этого раздела является оператор ввода, который на языке Паскаль записывается следующим образом:

read(x,y,p);

Оператор вывода в Паскаль-программе будет выглядеть так:

writeln(x,y,p);

Оператор ветвления, расположенный в предписании 5 алгоритма, в каждой ветви содержит по два оператора. Эти операторы в Паскаль-программе должны быть заключены в операторные скобки begin…end. Оператор ветвления, соответствующий предписанию 5 алгоритма, будет иметь вид:

```
if x<=y+p then
  begin
 a:=y/p;
 b:=x*y;
end
else
begin
 a:=exp((y–x)/2;
 b:=ln(abs(y+p));
end;
```

Конструкция «Конец-Если 5» алгоритма на язык Паскаль не переводится.

Оператор вывода, расположенный в предписании 7 алгоритма, на языке Паскаль записывается следующим образом:

writeln(a,b);

Этот оператор является последним в разделе операторов, поэтому после него нужно поставить оператор end. Вся программа будет выглядеть следующим образом:

```
program z1;
var x,y,p,a,b:real;{раздел описания переменных}
begin { раздел операторов}
read(x,y,p);
writeln(x,y,p);
if x<=y+p then
 begin
  a:=y/p;
  b:=x*y;
 end
else
 begin
 a:=exp((y–x)/2;
 b:=ln(abs(y+p));
 end;
writeln(a,b);
end.
```

Задача 2. Написать программу, позволяющую вычислить площадь треугольника, если заданы длины его сторон.

Решение. Математическая модель решаемой задачи имеет следующий вид:

Ввод: a, b, c – длины сторон треугольника.

Треугольник может быть построен по трем сторонам, если выполняются следующие условия, записанные в виде логического выражения: a>0 и b>0 и c>0 и a+b>c и b+c>a и a+c>b.

Проверка этого логического выражения позволяет «отфильтровать» все варианты неверных исходных данных.

После проверки логического выражения рассчитываются следующие переменные:

p=(a+b+c)/2;

$s = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$.

Вывод: s.

Алгоритм решения этой задачи в виде псевдокодов можно представить следующим образом:

1. Начало;
2. Список данных: a,b,c,p,s – вещ.;
3. Ввод(a,b,c);
4. Вывод(a,b,c);
5. Если (a>0) и (b>0) и (c>0) и (a+b>c)
   и (b+c>a) и (a+c>b) То
5.1. p:=(a+b+c)/2;
5.2. $s = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$;
5.3. Вывод(s);
   Иначе
5.4. Вывод(«Неверные исходные данные»);
6. Конец-Если 5;
7. Конец.

Pascal-программа решения этой задачи имеет вид:

```
program z2;
var a,b,c,p,s:real;
begin
read(a,b,c);
writeln('a=',a,' b=',b,' c=',c);
if (a>0) and (b>0) and (c>0) and (a+b>c)
 and (b+c>a) and (a+c>b) then
 begin
  p:=(a+b+c)/2;
  s:=sqrt(p*(p–a)*(p–b)*(p–c));
  writeln('s=',s);
 end
else
writeln('неверные исходные данные');
end.
```

Задача 3 [5]. Написать фрагмент программы, проверяющей, попадет ли точка M(x,y) в заштрихованную область (рисунок 1). Считать, что границы принадлежат области.

Решение. Точка принадлежит области, если она принадлежит части области, расположенной в I квадранте, или принадлежит части области, расположенной во II квадранте, или – в III квадранте, или в IV квадранте.

Обозначим условие, что точка лежит в части области в I квадранте через K1, во II квадранте – через K2, в III квадранте – через K3, в IV квадранте – через K4.

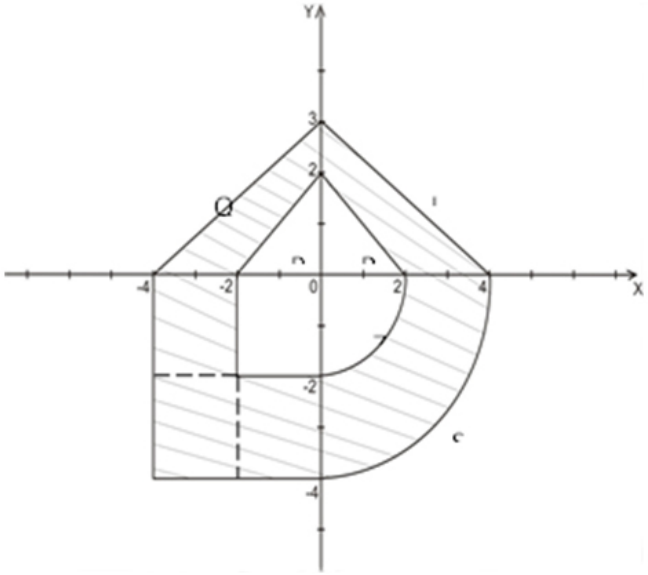

Рисунок 1 − Заданная область

Тогда, если логическое выражение K1 or K2 or K3 or K4 будет истинным, то точка M лежит в приведенной области, иначе – нет.

Раскроем условие K1. Координаты точки удовлетворяют условию K1, если x>=0 и y>=0, и лежат они ниже или на прямой, обозначенной L, и выше или на прямой, обозначенной R.

Будем искать уравнение прямой L в виде **y=a*x+b**. Так как эта прямая проходит через точки (0,3) и (4,0), то коэффициенты можно найти из системы уравнений:

$$\begin{cases} 3=a*0+b; \\ 0=a*4+b. \end{cases}$$

Отсюда b=3, a=–3/4. Уравнение прямой L имеет вид: **y=–3/4*x+3**.

Уравнение прямой R находится аналогично, с учетом того, что она проходит через точки (0,2) и (2,0). Уравнение прямой R имеет вид: **y=–x+2.**

Исходя из вышесказанного, условие K1 записывается в виде:

K1=(y<=–3/4*x+3) and (y>=–x+2) and (y>=0) and (x>=0).

Раскроем условие K2. Условие K2 находится аналогично, так как область представляет собой промежуток между парой прямых Q и P.

Прямая Q проходит через точки (0,3) и (–4,0) и имеет уравнение: **y= 3/4*x+3.**

Прямая P проходит через точки (0,2) и (–2,0) и имеет уравнение: **y= x+2.**

Условие K2 записывается в виде:

K2=(y<= 3/4*x+3) and (y>= x+2) and (y>=0) and (x<=0).

Раскроем условие K3. Часть области в III квадранте представляет собой объединение двух прямоугольников. Первый прямоугольник имеет вершины в точках с координатами (–4,0), (–2,0), (–2,–4), (–4,–4). Чтобы точка попала в этот прямоугольник, необходимо выполнение условия:

(–4<y<0) and (–4<x<–2).

Второй прямоугольник имеет вершины в точках с координатами (–4,–2), (0,–2), (0,–4), (–4,–4). Чтобы точка попала в этот прямоугольник, необходимо выполнение условия:

(–4<y<–2) and (–4<x<0).

Чтобы точка попала в часть области, расположенную в III квадранте, необходимо и достаточно, чтобы она попала в 1-й или 2-й прямоугольники. Условие этого попадания записывается так:

K3=((–4<y<0) and (–4<x<–2)) or ((–4<y<–2) and (–4<x<0)).

Раскроем условие K4.

Часть области, расположенная в IV квадранте, представляет собой часть кольца, ограниченного окружностями S и Z. Уравнение окружности S: $\mathbf{x^2+y^2=16}$. Уравнение окружности Z: $\mathbf{x^2+y^2=4}$.

Условие K4 запишется следующим образом:

K4=(x>=0) and (y<=0) and (x^2+y^2<=16) and (x^2+y^2>=4).

Таким образом, общее условие попадания точки в выделенную область, выглядит так:

((y<= –3/4*x+3) and (y>= –x+2) and (y>=0) and (x>=0)) or
((y<= 3/4*x+3) and (y>= x+2) and (y>=0) and (x<=0)) or
(((–4<y<0) and (–4<x<–2)) or ((–4<y<–2) and (–4<x<0))) or
((x>=0) and (y<=0) and (x^2+y^2<=16) and (x^2+y^2>=4)).

7 Циклические Паскаль-программы

Циклическим (циклом) называется такой алгоритм, в котором некоторая группа действий повторяется неоднократно. Группа действий, повторяемая в цикле, называется *телом цикла*. Однократное выполнение тела цикла называется *шагом*. Для того чтобы алгоритм не зациклился (не стал бесконечным), циклом надо управлять. Для этого используется специальная величина – параметр цикла. *Параметр (переменная) цикла* – это такая величина, которая изменяется от шага к шагу и по значению которой определяется, продолжать исполнение цикла или его закончить.

Для правильной организации цикла необходимо выполнить следующие шаги.

Сначала определим действия, которые повторяются в цикле, т.е. его тело. Затем ответим на вопрос: «Сколько раз повторяется тело цикла?». Если на этот вопрос можно ответить, то цикл является *арифметическим* и его параметром служит счетчик. Если ответ отрицательный, то отвечаем на вопрос: «Сколько причин окончания цикла имеется?». Если имеется только одна причина окончания, то цикл *итерационный*. В качестве параметра здесь чаще всего выбирается величина, сравнимая с заданной точностью вычислений или преобразований. Если существует больше одной причины окончания цикла, то он *поисковый*. У такого цикла условие окончания задается сложным логическим выражением, содержащим конъюнкцию или дизъюнкцию.

На последнем шаге организации цикла необходимо правильно установить начальные значения для всех переменных, которые используются или изменяются в цикле. Эти переменные, как правило, находятся в условиях или в правых частях операторов присваивания. Для этого можно провести трассировку первоначального входа в цикл и определить, какие значения должны быть у используемых или изменяемых переменных.

Следует различать реальный циклический алгоритм (арифметический, итерационный, поисковый) и способ записи его в алгоритмическом языке. (Сравните ситуацию с числом и способами его записи в какой-нибудь системе счисления.)

В алгоритмическом языке Паскаль реальные циклы могут быть записаны с помощью следующих операторов:

while логическое_выражение do оператор;
repeat операторы_тела_цикла until логическое_выражение;
for параметр:=выражение_1 to выражение_2 do оператор;
for параметр:=выражение_1 downto выражение_2 do оператор;
M3: if логическое выражение then goto M1 else goto M2;
M1: тело цикла goto M3; M2:…

С помощью любого из перечисленных операторов можно записать цикл каждого из установленных нами типов.

Арифметический цикл. Для арифметического цикла заранее известно количество раз выполнения тела цикла, поэтому в качестве переменной используется счетчик.

Задача 1. Написать программу для вычисления суммы N первых натуральных чисел.

Решение. Требуется вычислить: 1+2+3+...+N. Понятно, что для нахождения суммы нужно каждый раз прибавлять очередное слагаемое; таким образом, для решения задачи необходимо организовать цикл. Исходным данным для решения является N.

Выясним, что повторяется в теле цикла. Для этого попробуем вычислять сумму вручную. Начинаем суммировать с нуля. Тогда

s_0=0,

$s_1=0+1$,
$s_2=0+1+2$,
$s_3=0+1+2+3$.

Найдем, чем каждый шаг отличается от соседнего. Для этого подсчитаем разности: $s_1–s_0=0+1–0=1$,

$s_2–s_1=0+1+2–(0+1)=2$,
$s_3–s_2=0+1+2+3–(0+1+2)=3$.

Отсюда $s_1=s_0+1$,
$s_2=s_1+2$,
$s_3=s_2+3$.

Рассуждая аналогично, можем заключить, что в общем случае для нахождения следующего значения суммы нужно к ее предыдущему значению прибавить очередное слагаемое. Обозначим очередное слагаемое через i, а номер шага – через j. Тогда $s_j=s_{j-1}+i$ повторяется в цикле. Здесь индексы указывают, что это одна и та же величина (сумма), но в разные моменты времени. При программировании эти индексы указывать не нужно, так как оператор присваивания выполняется в два этапа. На первом этапе выбираются значения всех указанных справа переменных, вычисляется значение выражения, стоящего справа. Затем вычисленное значение пересылается в переменную, указанную слева. Формулы, в которых новое значение вычисляется на основе одного или нескольких старых значений, называются рекуррентными. Полученная нами формула является рекуррентной формулой первого порядка, так как для вычисления нового значения достаточно одного предыдущего значения.

Известно, что для вычисления всей суммы нужно выполнить полученную формулу N раз. Следовательно, необходимо организовать арифметический цикл с параметром-счетчиком, обеспечивающим N отсчетов. Приводим далее программу на языке Паскаль:

```
program pr1;
{вычисление суммы натуральных чисел от 1 до N – вариант 1}
var     N:integer;      {последнее число суммы – исходное данное}
        s:integer;      {сумма – результат}
        i:integer;      {очередное слагаемое}
        j:integer;      {счетчик цикла}
begin write('Введите последнее число суммы ');
        readln(N);      {Ввод исходного данного}
        s:=0;           {Начинаем суммирование с нуля}
        i:=1;    {Первое слагаемое известно заранее}
        j:=0;           {Начальное значение счетчика}
        while j<N do  {Нужно выполнить цикл N раз}
        begin s:=s+i; {Добавили слагаемое к сумме }
                i:=i+1; {Перешли к следующему слагаемому}
                j:=j+1; {Увеличили счетчик на единицу}
        end;            {Закончилось тело цикла}
        write('Сумма чисел от 1 до ',N,' равна ',s);
                                {Вывод результата}
end.
```

Анализируя программу, можно заметить, что переменные i и j изменяются синхронно (т.е. по значению i можно установить значение j и наоборот), поэтому в данном случае переменная, изображающая слагаемое, может играть и роль счетчика. Тогда программа примет вид:

```
program pr2;
{вычисление суммы натуральных чисел от 1 до N – вариант 2}
```

```
var     N:integer;      {последнее число суммы – исходное данное}
        s:integer;              {сумма – результат}
        i:integer;              {счетчик цикла и слагаемое}
  begin write('Введите последнее число суммы ');
        readln(N);      {Ввод исходного данного}
        s:=0;           {Начинаем суммирование с нуля}
        i:=1;           {Начальное значение счетчика и слагаемого}
        while i<=N do {Нужно выполнить цикл N раз}
        begin s:=s+i; {Добавили слагаемое к сумме}
                i:=i+1; {Увеличили счетчик и слагаемое на единицу}
        end;                    {Закончилось тело цикла}
        write('Сумма чисел от 1 до ',N,' равна ',s);
                        {Вывод результата}
  end.
```

После написания программы ее нужно протестировать. Прежде убедимся, что цикл завершится. Это действительно так в случае, если N≥1, поскольку переменная i на каждом шаге цикла увеличивается от 1 до N. Если N – не положительно, то цикл не будет выполнен ни разу и будет выдан ответ 0, что соответствует нашим ожиданиям от правильно работающей программы.

Теперь проверим работу программы для такого N, ответ для которого заранее известен. Рассматриваемая программа предназначена только для иллюстрации методов построения алгоритмов, результат ее работы просто получить, воспользовавшись формулой s=N·(N+1)/2. Например, для N=10 получаем s=55. Выполнив программу для N=10, получаем тот же ответ.

Программа позволяет вводить только целое значение N, отвергнув данные других типов, например, вещественные, строковые и т.д.

Теперь проверим работу программы для N=300, получим s=–20386. Почему получено отрицательное число, ведь складывались положительные числа? Выполним программу для N=200, получим s=20100. Для N=400, получим s=14662. Почему сумма для меньшего значения N превосходит сумму для большего значения N? Оказывается, все дело в переполнении. Значения суммы для N=300 и N=400 больше максимально допустимого для типа integer (здесь maxint=32767), поэтому для этих значений программу использовать нельзя. Установим максимальное значение N, для которого ответ корректен. Решим неравенство N·(N+1)/2≤32767. Данное неравенство сводится к квадратному: N^2+N–65534≤0. Решением этого неравенства будет интервал –256.5≤N≤255.5. Следовательно, разработанная программа верна для всех N от 1 до 255. Если необходимо получать суммы для больших значений N, то следует перейти к другим типам данных, например, longint или real, и внести соответствующие исправления в программу.

Установив границы применимости программы, можно закончить тестирование.

Заметим, что тот же цикл можно было записать с помощью оператора for:

```
s:=0;
for i:=1 to N do s:=s+i;
```

или

```
s:=0;
for i:=N downto 1 do s:=s+i;.
```

Записи в этом случае получились более короткими, потому что оператор for специально предназначен для записи арифметических циклов. Запись с помощью цикла repeat until может быть такой:

```
s:=0;
i:=1;
```

```
repeat s:=s+i;
   i:=i+1
until i>N;.
```

Задача 2. Написать программу, рисующую клеточное поле 8x8.

Решение. Рисунок состоит из линий: девяти вертикальных и девяти горизонтальных. Линии отстоят друг от друга на величину h. За исходную точку рисунка можно примем левый верхний угол, задав его координаты: x, y. В теле цикла будут повторяться операторы, рисующие одну горизонтальную и одну вертикальную линии. Цикл выполнится девять раз. Установим, как вычисляются координаты начала каждой линии. Каждая линия имеет свой номер (в программе он называется i) и отстоит от первой линии на расстоянии, которое можно вычислить как (i–1)*h: вторая линия отстоит от первой на 1*h, третья – на 2*h и т.д. По другой координате длина линии должна быть равна 8*h.

```
readln(x,y,h); {вводим исходные данные}
i:=1;
while i<=9 do
begin line(x+(i–1)*h,y,x+(i–1)*h,y+8*h);
 {вертикальные прямые}
   line(x,y+(i–1)*h,x+8*h,y+(i–1)*h);
                              {горизонтальные прямые}
end.
```

Итерационный цикл

Задача 3. Написать программу для нахождения наибольшего общего делителя двух натуральных чисел.

Решение. Используя идею Евклида, будем вычитать из большего числа меньшее до тех пор, пока числа не сравняются. Получим цикл, счетчик для которого неизвестен заранее, но причина окончания у него одна – числа сравнялись. Цикл этот обязательно завершится, потому что уменьшаются натуральные числа, разность между двумя соседними натуральными числами и наименьшее натуральное число – единица. Такой цикл называется итерационным. Ниже приводится программа, реализующая решение задачи на языке Паскаль:

```
{вычисление НОД двух натуральных чисел}
var a,b:integer; {исходные натуральные числа}
  x,y:integer; {переменные, используемые для поиска НОД}
begin write('Введите два натуральных числа ');
          readln(a,b);
          x:=a; y:=b;
          while x<>y do
           if x>y
           then x:=x–y
           else y:=y–x;
          write(x);
end.
```

Поисковый цикл

Задача 4. Последовательность элементов задана формулой общего члена $a_i=\sin(i+i/n)$, где i изменяется от 1 до n (n – натуральное число). Написать программу нахождения первого элемента последовательности, большего заданного числа Z.

Решение. Исходными данными для решения задачи являются элементы последовательности: $a_1=\sin(1+1/n)$, $a_2=\sin(2+2/n)$,..., $a_n=\sin(n+n/n)$. В результате получаем либо номер элемента, большего заданного Z, либо ответ: «Такого элемента нет».

Решения задачи существуют два: первое – перебрали все элементы последовательности и не нашли нужного; второе – в процессе перебора обнаружился элемент, больший Z. В этом случае перебор прекращается и формируется ответ. Когда будет найден ответ, неизвестно, следовательно, это поисковый цикл.

Согласно первому решению проверим условие: i>n, где i – текущий элемент последовательности, а n – количество элементов в ней. Второе решение имеет два значения: «найдено» или «не найдено», логический эквивалент: true – найдено, а false – не найдено. Таким образом, условие окончания поиска может быть записано в виде (i>n) or f, что соответствует фразе: «Просмотрены все элементы или найдено». Просмотр элементов последовательности в цикле будет выполняться в случае ложности приведенного условия. Найдем его отрицание, воспользовавшись законом де Моргана: отрицание дизъюнкции равно конъюнкции отрицаний.

not((i>n) or f)=not(i>n) and not f=(i<=n) and not f.

Программа на языке Паскаль:

```
{поиск первого элемента последовательности, заданной формулой общего члена a(i)=sin(i+i/n), где i=1,2,...,n, большего заданного Z}
var n:integer; {количество элементов в последовательности}
        z:real; {заданное число}
        i:integer; {номер очередного элемента последовательности}
        f:boolean; {true, если найден искомый элемент}
begin
  write('Введите n и z ');
  readln(n,z);
  i:=1; f:=false;
  while (i<=n) and not f do {пока есть элементы и не найдено}
        if sin(i+i/n)>z
        then f:=true {искомый элемент найден}
        else i:=i+1; {переходим к следующему элементу}
  if f
  then writeln('Первый элемент больший ',z,' имеет номер ',i)
  else writeln('Среди элементов последовательности нет больших ',z);
end.
```

Способы определения действий, составляющих тело цикла. Наибольшую сложность представляет определение действий, повторяющихся в цикле. Рассмотрим некоторые приемы.

Задача 5. Написать программу для вычисления выражения: y=k(k+1)(k+2)...2k.

Решение. Выражение представляет собой произведение. Перепишем его так, чтобы все сомножители имели одинаковую форму: (k+0)(k+1)(k+2)...(k+k). Исходя из этого, установим, что отыскивается произведение сомножителей. Каждый сомножитель содержит переменную k, к которой прибавляется некоторое натуральное число. Это число меняется от сомножителя к сомножителю, поэтому обозначим его переменной i. Тогда общий вид сомножителя: (k+i), где i – изменяется от 0 до k. В теле цикла повторяется оператор вида y:=y*(k+i). Известно, что тело цикла должно выполниться k раз, следовательно, нужно построить арифметический цикл.

```
y:=1; {начальное значение произведения}
for i:=0 to k do y:=y*(k+i);.
```

В данной задаче использовался прием нахождения формулы общего члена на основе анализа структуры сомножителей.

Задача 6. Одним из способов построения тела цикла является сведение новой задачи к ранее решенной. Например, на основе методики решения предыдущей задачи, получим решение следующей.

Написать программы для вычисления значения по заданным формулам:

1) $k(k+m)(k+2m)...(k+m^2)$;
2) (k+3)(2k+6)(3k+9)..., всего в этой формуле должно быть n сомножителей;
3) $k+2k+3k+\ldots+k^2$.

Решение.

```
1) y:=1; {начальное значение произведения}
 for i:=0 to m do y:=y*(k+i*m);
2) y:=1; {начальное значение произведения}
 for i:=1 to n do y:=y*(i*k+3*i);
3) y:=1; {начальное значение произведения}
 for i:=1 to k do y:=y+k*i;.
```

Задача 7. Написать программу для вычисления выражения: cos(1+cos(2+...+cos(n–1+cos(n)...)).

Решение. Для начала запишем заданное выражение при различных значениях *n* и поместим ответ в переменную y.

Для n=3 получим y=cos(1+cos(2+cos(3))).

Для n=5 получим y=cos(1+cos(2+cos(3+cos(4+cos(5))))).

Заметим, что в данном выражении используются натуральные числа. Обозначим их переменной i, которая изменяется от n до 1. Для того чтобы вычислить это выражение вручную, начнем вычисления с самых вложенных скобок:

1) $y_1=\cos(5)$;
2) $y_2=\cos(4+\cos(5))$ или с учетом первого шага $\cos(4+y_1)$,

аналогично

3) $y_3=\cos(3+y_2)$;
4) $y_4=\cos(2+y_3)$;
5) $y_5=\cos(1+y_4)$.

Перепишем первый шаг как $y_1=\cos(5+y_0)$, где $y_0=0$; в теле цикла будет повторяться оператор y=cos(i+y). Количество повторений известно заранее и равно n, следовательно, цикл – арифметический. Здесь в качестве счетчика можно использовать переменную i.

```
y:=0;
for i:=n downto 1 do y:=cos(i+y);.
```

При решении этой задачи использовалась методика вычисления заданного значения вручную, наблюдение за процессом вычисления и обобщение выполняемых действий.

Задача 8. Написать программу для вычисления выражения

$y = \sqrt{n+\sqrt{n-1+\ldots+\sqrt{2+\sqrt{1}}}}$, при произвольном натуральном n.

Решение. Структура задач 7 и 8 одинакова, поэтому они обе должны иметь одинаковые по структуре алгоритмы решения. Отсюда получаем фрагмент программы.

```
y:=0;
for i:=1 to n do y:=sqrt(i+y);.
```

Используя решения задач 7.1.7 и 7.1.8, легко решить им подобные:

1) S1=sin(1+sin(2+sin(3+…+sin(n)…);
2) S2=|1+|2+|3+…+|n|…|;

3) $S3 = \sqrt{1+\sqrt{2+\sqrt{3+...+\sqrt{n}}}}$;

4) $$S4 = \cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{3+...}}}$$

$$\cfrac{}{n-1+\cfrac{1}{n}}$$

Задача 9. Написать программу для вычисления выражения n умножений. Например, для n=5*:* y=(((((x+a)x+a)x+a)x+a)x+a)x+a.

Решение. Задача аналогична предыдущим, но здесь используется функция умножения, а вместо отрезка натурального ряда – константа a. Однако обнаружить это сходство довольно трудно, поэтому установим оператор, повторяющийся в теле цикла, рассмотрев любые два соседних значения y, вычисленные вручную.

y_1=x+a.

y_2=(x+a)x+a или $y_2=y_1$x+a.

По аналогии делаем вывод, что в теле цикла для получения нового значения нужно предыдущее умножить на x и прибавить a. Если y_0=1, $y_1=y_0$x+a, то тело цикла будет выполняться n+1 раз.

```
y:=1;
for i:=1 to n+1 do y:=y*x+a;.
```

Другое решение получаем, если y_0=x, $y_1=(y_0+a)$x. В этом случае тело цикла выполняется n раз и после завершения цикла нужно прибавить a.

```
y:=x;
for i:=1 to n do y:=(y+a)*x;
y:=y+a;.
```

Задача 10. Написать программу для вычисления суммы n слагаемых $\frac{1}{3}+\frac{1}{8}+\frac{1}{15}+\frac{1}{24}+\frac{1}{35}+...$.

Решение. Поставим в соответствие каждому слагаемому его порядковый номер – i. Тогда легко заметить, что знаменатель вычисляется по формуле $(i+1)^2-1$. Здесь в цикле к сумме прибавляется очередное слагаемое. Цикл выполняется n раз, т.е. является арифметическим.

```
s:=0;
for i:=1 to n do s:=s+1/(sqr(i+1)–1);.
```

Задача 11. Часто для определения тела цикла используют пошаговую детализацию.

Последовательность a_n задается так: a_1 – некоторое натуральное число, a_{n+1} – сумма квадратов цифр числа a_n, n>=1. Написать программу для нахождения m-го члена последовательности.

Решение. Для решения задачи нужно m раз выполнить вычисление очередного члена последовательности. Решение можно представить так:

```
{фрагмент 1}
readln(a,m);
for i:=2 to m do
begin {вычислить i-й член последовательности}
end;.
```

Сосредоточимся на определении члена последовательности. Необходим цикл, в котором цифры просматриваются по одной, начиная с единиц, и находится сумма их квадратов. Поскольку количество цифр заранее не известно, организуем итерационный цикл.

```
{фрагмент 2}
{находим сумму квадратов цифр}
s:=0;
while a>0 do
begin s:=s+sqr(a mod 10); {цифру единиц возводим в квадрат и прибавляем к сумме}
   a:=a div 10 {отбрасываем цифру единиц}
end;
a:=s; {запоминаем полученную сумму как очередной член последовательности}
```

Подставив фрагмент 2 в фрагмент 1, получим искомую программу, содержащую вложенный цикл.

```
var a, {член последовательности}
 s:integer; {сумма квадратов цифр предыдущего члена последовательности}
  i, {счетчик}
  m:integer; {номер последнего члена последовательности}
begin
  write('Введите 1-ый член последовательности и m ');
  readln(a,m);
  for i:=2 to m do
  begin s:=0;
   while a>0 do
   begin s:=s+sqr(a mod 10);
   a:=a div 10
   end;
   a:=s;
  end;
end.
```

Задача 12. Написать программу для вычисления s=1·2 + 2·3·4 + 3·4·5·6+...+n(n–1)...2·n.

Решение. Действуя по аналогии с задачей 7.1.11, получим решение, содержащее *вложенные* циклы:

```
s:=0;
i:=1;
while i<=n do
begin
  p:=1;
  j:=i; { переменная цикла вычисления произведения }
  while j<=2*i do
  begin
          p:=p*j;
          j:=j+1;
  end;
  s:=s+p;
  i:=i+1;
end.
```

Использование вложенных циклов упрощает программирование, но увеличивает время решения задачи. Наличие вложенного цикла – недостаток предложенного реше-

ния. От него можно избавиться, если при вычислении следующего слагаемого выполняется часть работы, которая уже была выполнена для предыдущего слагаемого. Выпишем i-ое слагаемое: $u_i=i(i+1)...(2\cdot i)$. (i–1)-ое слагаемое равно: $u_{i-1}=(i-1)i(i+1)...2(i-1)$. Найдем их отношение:

$$\frac{u_i}{u_{i-1}}=\frac{i(i+1)(i+2)...(2i-2)(2i-1)2i}{(i-1)i(i+1)...(2i-2)}=\frac{(2i-1)2i}{i-1}.$$

Отсюда $u_i=u_{i-1}\cdot(2i-1)2i/(i-1)$. Это соотношение называется *рекуррентным*. Используя его, получаем другое решение:

```
s:=0;
u:=2; {первое слагаемое}
i:=2; {номер очередного слагаемого}
while i<=n do
begin
  s:=s+u;
  u:=u*(2*i–1)*2*i/(i–1);
  i:=i+1;
end.
```

Задача 13. Определим операторы тела цикла, анализируя структуру входных или выходных данных.

Написать программу, печатающую квадрат со стороной из n звездочек (n>2), причем звездочки должны располагаться по периметру квадрата. Например, для n=4:

Решение. Вначале разберемся со структурой (строением) выходной информации с учетом последовательной печати. С этой точки зрения квадрат состоит из трех частей. Первая и последняя строки одинаковы и состоят из n звездочек. n–2 средних строки также имеют одинаковую структуру и начинаются, и кончаются звездочками, разделенными n–2 пробелами. Установленная структура определяет структуру алгоритма вывода квадрата:

1 Вывод строки из n звездочек.
2 Вывод n–2 строк.
3 Вывод строки из n звездочек.

Второй пункт детализируется также с учетом структуры выводимых строк:

2 Цикл повторить n–2 раза:
2.1 Вывод звездочки.
2.2 Вывод n–2 пробелов.
2.3 Вывод звездочки.

Подставив детализацию второго пункта в исходную структуру алгоритма и заменив словесную интерпретацию операторами языка Паскаль, получим программу.

```
{1} writeln; for i:=1 to n do write('*');
{2} for i:=1 to n–2 do
begin
  {2.1} writeln; write('*');
  {2.2} for j:=1 to n–2 do write(' ');
  {2.3} write('*')
end;
{3} writeln; for i:=1 to n do write('*');.
```

Задача 14. Задано натуральное n и ряд 1+2–3+4+5–6+.... Напечатать вычисляемый ряд, знак равно и ответ, если в ряде используется n слагаемых.

Решение. Если каждому слагаемому поставить в соответствие его номер, то легко заметить, что элементы с номерами, кратными трем, вычитаются из суммы. При выводе элементов ряда следует учесть, что перед первым элементом знак «+» печатать не надо. Решение будет представлять собой арифметический цикл, тело которого содержит условный оператор. Этот оператор прибавляет элемент к сумме, если его номер кратен трем, в противном случае – элемент вычитается из суммы.

```
var i, {очередной член ряда}
  n, {количество членов ряда}
  s:integer; {сумма}
begin
  write('Введите n ');
  readln(n);
  s:=0; {начальное значение суммы}
  for i:=1 to n do
  if i mod 3=0 {если очередной член кратен 3,}
  then begin s:=s–i; write('–',i); end {то вычтем его и напечатаем со знаком минус}
  else     begin s:=s+i;   if i=1 {иначе прибавим}
                    then write(i) {если член первый, то знак не
                                   печатаем}
                    else write('+',i); {иначе печатаем плюс}
           end;
  write('=',s);
end.
```

Задача 15. Для определения тела цикла используем метод модификаций, когда тело построенной ранее программы изменяется (модифицируется).

Натуральные числа в произвольном порядке вводятся с клавиатуры. Признаком окончания ввода является число нуль. Написать программу, определяющую наибольшее введенное число и его номер.

Решение. Для решения задачи сравним числа друг с другом, причем для следующего сравнения отбирается максимальное из уже просмотренных чисел. Поскольку счетчик цикла заранее не известен, то используем итерационный цикл.

```
read(a); {Ввели первое число}
na:=1; {Его же рассматриваем в качестве первого кандидата на максимум}
if a<>0
then begin read(b); {Вводим следующее число}
   i:=2; {номер введенного числа}
   while b<>0 do
   begin if a<b then begin a:=b; na:=i; end;
   {Если a<b, то b – новый кандидат на максимум}
   read(b); {Вводим следующее число}
   i:=i+1;
   end;
   write(a,' ',i);
  end
else write('нет чисел для ввода');.
```

Модификация 1. Если условный оператор в теле цикла заменить оператором if a<=b then begin a:=b; na:=i; end;, то модифицированная программа отыщет номер последнего по порядку максимального числа, если среди чисел есть несколько, совпадающих по величине с максимальным.

Модификация 2. Если условный оператор в теле цикла заменить оператором if a>b then begin a:=b; na:=i; end;, то модифицированная программа отыщет номер первого по порядку минимального числа, если среди чисел есть несколько, совпадающих по величине с минимальным.

Модификация 3. Если условный оператор в теле цикла заменить оператором if a>=b then begin a:=b; na:=i; end;, то модифицированная программа отыщет номер последнего по порядку минимального числа, если среди чисел есть несколько, совпадающих по величине с минимальным.

Модификация 4. Если условный оператор в теле цикла заменить оператором if a<b then begin a:=b; na:=i; s:=1; end else if a=b then s:=s+1; и в самом начале программы добавить оператор s:=1, то модифицированная программа будет отыскивать количество элементов, совпадающих по величине с максимальным.

Задача 16. Задана неубывающая последовательность натуральных чисел. Проверить, действительно ли она является неубывающей.

Будем считать, что элементы последовательности хранятся в одномерном массиве. В массиве – n элементов.

Последовательность является неубывающей, если для любой пары рядом стоящих элементов левый элемент не больше правого. Имеем две эквивалентные записи определения неубывающей последовательности:

1) ∀i(a[i]<=a[i+1]) для 1<=i<=n–1;

2) ∀i(a[i–1]<=a[i]) для 2<=i<=n.

Отрицание любого из этих определений означает, что последовательность НЕ является неубывающей: not (∀i(a[i]<=a[i+1])) для 1<=i<=n–1 или ∃i(a[i]>a[i+1]) для 1<=i<=n–1. Полученная формула указывает на необходимость поиска в последовательности элементов, для которых выполняется условие: a[i]>a[i+1]. Получим алгоритм:

```
f:=true; {первоначально считаем, что искомого условия в последовательности нет}
i:=1;
while (i<=n–1)and f do
   if a[i]>a[i+1]
   then f:=false
   else i:=i+1;
if f
then write(’последовательность упорядочена’)
else write(’последовательность НЕ упорядочена’);.
```

Задача 17. Дано натуральное число n. Выяснить, входит ли цифра 2 в запись этого числа.

Решение. В теле цикла повторяются те же операции (найти и проанализировать последнюю цифру, отбросить ее). Имеем два решения: цифра 2 входит в запись числа n и цифра 2 не входит в запись этого числа. Следовательно, данный цикл должен быть построен как поисковый с проверкой каждого решения.

```
f:=false;
while (n>0) and not f do
  if n mod 10 =2 then f:=true else n:=n div 10;
if f then write(’цифра 2 входит в число ’)
else write(’цифра 2 НЕ входит в запись числа’);.
```

Задача 18. Числа Фибоначчи определяются так: два первых числа равны 1, каждое следующее равно сумме двух предыдущих. В частности: $u_3=u_2+u_1=1+1=2$;

$u_4=u_3+u_2=1+2=3$;

$u_5=u_4+u_3=2+3=5$;

$u_6=u_5+u_4=3+5=8$ и т.д.

Найти n-е число Фибоначчи.

Решение. Для получения очередного числа достаточно хранить два предыдущих, т.е. в программе постоянно используются три соседних числа Фибоначчи. Введем три переменные: A хранит u_k, B хранит u_{k-1}, C хранит u_{k-2}. Для вычисления следующего числа Фибоначчи необходимо провести сдвиг, т.е. переписать содержимое переменной B в C, а содержимое переменной A в B.

```
c:=1; {значение первого числа известно}
b:=1; {значение второго числа тоже известно}
k:=3; {начинаем вычисление с третьего числа}
while k<=n do {цикл, пока не найдено n-е число}
begin a:=b+c;{вычисляем следующее число как сумму двух предыдущих}
  c:=b; {сдвигаем b в c для нахождения следующего числа}
  b:=a; {сдвигаем a в b для нахождения следующего числа}
  k:=k+1 {увеличиваем счетчик найденных чисел}
end;
write(a); {печатаем найденное число}
```

Задача 19. Разработать алгоритм быстрого вычисления a^n, где n –целое, a – вещественное. Разрешается использовать операции сложения и умножения.

Решение.

$$a^n=\begin{cases} a*a*...*a, & \text{повторенное } n \text{ раз}, \quad n>0, \\ 1, & n=0, \\ 1/(a*a*...a*), & \text{повторенное } |n| \text{ раз}, \quad n<0. \end{cases}$$

Произведение повторяется дважды. Вычислим его один раз, а затем проверим n и вычислим степень. Схематически эту работу можно записать так:

1) вычислить a*a*...a, повторенное |n| раз;

2) если n<0, то ответ: 1/(a*a*...*a).

Для вычисления a*a*...*a воспользуемся тривиальным алгоритмом. Соответствующая программа на языке Паскаль приведена ниже. При n=0 программа выдает корректный ответ y=1, потому что тело цикла не выполнится ни разу (или выполнится 0 раз).

```
y:=1;
for i:=1 to abs(n) do y:=y*a;
if n<0 then y:=1/y;.
```

В построенном алгоритме цикл выполняется |n| раз. Ускорим его работу.

Опыт 1. Найти a^7

```
a2:=a*a;
a4:=a2*a2;
a7:=a4*a2*a;.
```

Опыт 2. Найти a^9

```
a2:=a*a;
a4:=a2*a2;
a8:=a4*a4;
a9:=a8*a;.
```

Опыт 3. Найти a^{15}

```
a2:=a*a;
a4:=a2*a2;
a8:=a4*a4;
a15:=a8*a4*a2*a;.
```

Заметим, что в каждом из проведенных опытов количество операций умножения меньше n, однако используются дополнительные переменные, от которых можно сразу избавиться, учитывая вычисленные значения в результате. Продолжим опыты. Будем применять следующие обозначения: a – исходное данное и переменная для промежуточных степеней, y – ответ.

Опыт 4. Найти a^7

y:=a; {a}
a:=a*a; {a^2}
y:=y*a; {a^3}
a:=a*a; {a^4}
y:=y*a; {a^7}.

Опыт 5. Найти a^9

y:=a; {a}
a:=a*a; {a^2}
a:=a*a; {a^4}
a:=a*a; {a^8}
y:=y*a; {a^9}.

Опыт 6. Найти a^{15}

y:=a; {a}
a:=a*a; {a^2}
y:=y*a; {a^3}
a:=a*a; {a^4}
y:=y*a; {a^7}
a:=a*a; {a^8}
y:=y*a; {a^{15}}.

В опытах 4–6 повторяются либо два оператора: y:=y*a; a:=a*a;, либо один – a:=a*a;. Для того чтобы установить условия, при которых выполняются два оператора, заметим, что 7=4+2+1, 9=8+1, 15=8+4+2+1. Для вычисления седьмой степени перемножим четвертую, вторую и первую степени основания. Аналогично для девятой и пятнадцатой степеней. Проведя дополнительные опыты, определим, что оператор y:=y*a;, когда остаток от деления показателя степени на два равен 1, или оператор y:=y*a; выполняется для тех степеней, у которых соответствующие им разряды двоичного представления показателя степени равны 1. Сравнить: $7=111_2$, $9=1001_2$, $15=1111_2$.

Используя вышесказанное, получим:

```
y:=1;
p:=0; {знак показателя степени}
if n<0 then begin p:=1; n:=–n end;
while n>0 do
begin if n mod 2 =1 then y:=y*a;
  a:=a*a;
  n:=n div 2;
end;
if p=1 then y:=1/y;.
```

Задачи, которые имеют такую же структуру исходных данных, это прежде всего, нахождение произведения, которое определяется как: y=a*b=a+a+...+a, повторенное b раз. Заменив в предыдущем алгоритме операцию умножения сложением, получим:

```
y:=0; {начальное значение для сложения}
while b>0 do
begin if b mod 2 =1 then y:=y+a;
  a:=a+a;
  b:=b div 2;
```

```
end.
```

Аналогичный алгоритм применяется в следующей задаче: задана строка и целое число n. Получить строку, состоящую из n повторений исходной строки. Например, задано x=’pas’ и n=4. Должно получиться y=’paspaspaspas’. С этой работой легко справляется приведенный ниже фрагмент.

```
y:=’’; {начальное значение для сцепления}
while n>0 do
begin if n mod 2 =1
  then y:=y+a; {здесь ’+’ задает операцию сцепления}
  a:=a+a;
  n:=n div 2;
end.
```

Таким образом, представлены методики решения по двум из девятнадцати выделенных классов задач:

6 Ветвящиеся Паскаль-программы.

7 Циклические Паскаль-программы.

В следующей статье мы продолжим знакомство с методикой быстрого обучения программированию на основе изучения классов задач. Будут рассмотрены методики по следующим трем из девятнадцати выделенных классов задач:

8 Задачи, решаемые методом перебора.

9 Одномерные массивы.

10 Двухмерные массивы.

Литература

1. *Аляев Ю.А.* Алгоритмизация и языки программирования Pascal, C++, Visual Basic / Ю.А. Аляев, О.А. Козлов. М.: Финансы и статистика, 2002, 2004, 2007. 320 с.

2. *Аляев Ю.А.* Практикум по алгоритмизации и программированию на языке Паскаль / Ю.А. Аляев, В.П. Гладков, О.А. Козлов. М.: Финансы и статистика, 2004, 2007. 528 с.

3. *Аляев Ю.А.* Методика быстрого обучения программированию на основе изучения классов задач (1–3) // Образовательные ресурсы и технологии. 2015'1(9). С. 3–14. URL: http://www.muiv.ru/vestnik/pdf/pp/ot_2015_1_3-14.pdf

4. *Аляев Ю.А.* Методика быстрого обучения программированию на основе изучения классов задач (4–5) // Образовательные ресурсы и технологии. 2015'2(10). С. 3–16. URL: http://www.muiv.ru/vestnik/pdf/pp/ot_2015_2_3-16.pdf

5. *Гладков В.П.* Методика синтеза разветвляющихся алгоритмов на примере задачи определения попадания точки в заданную область / В.П. Гладков, Ю.А. Аляев // Рождественские чтения: материалы XVIII Региональной научно-методической конференции по вопросам применения ИКТ в образовании 16 января 2015 г. / отв. за вып. Ю.А. Аляев, С.В. Русаков; Перм. гос. нац. исслед. ун-т. Пермь, 2015. Вып. 18. С. 13–15.

Methods of the quick education to programming on base of the study of the classes of the problems (6–7)

***Yuri Alexandrovich Alyaev**, assistant professor, assistant professor of the pulpit mathematicians and naturally-scientific discipline, the Russian Presidential Academy of National Economy and Public Administration (Permskiy branch)*

Is offered methods of the quick education to programming on base of the study of the classes of the problems, designed and using in practice in process of the education to programming student high school.

The Keywords: algorithm, program, programming language Pascal, array, branching and looping Pascal-programs.