МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Вологодский государственный университет»
(ВоГУ)

Кафедра автоматики и вычислительной техники

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания к выполнению практической работы «Модели надежности программ»

Факультет электроэнергетический

Направления подготовки:
231000.62 – Программная инженерия
230100.62 – Информатика и вычислительная техника

Вологда 2015

УДК 681.3.06

Тестирование программного обеспечения: методические указания к выполнению практических работ «Модели надежности программ». - Вологда, ВоГУ, 2015. - 14 с.

Методические указания предназначены для проведения двух практических работ. В них имеется необходимый теоретический материал включены задачи для самостоятельной работы и вопросы для самоконтроля.

Утверждено редакционно – издательским советом ВоГУ.

Составитель: Сергушичева А.П., канд.техн. наук, доц. каф. АВТ,

Рецензент: Швецов А.Н., декан ФЗДО, доктор техн. наук, профессор

Практическая работа. **Модели надежности программ**

Основные теоретические положения

Надежность **(Reliability)** – набор атрибутов, относящихся к способности программного обеспечения сохранять свой уровень качества функционирования при установленных программными документами условиях за установленный период времени, в том числе и в условиях возникновения отклонений в среде функционирования, вызванных сбоями технических средств, ошибками во входных данных, ошибками обслуживания и другими дестабилизирующими воздействиями.

2.2. Основные показатели надежности ПО

1. *Вероятность безотказной работы* $P(t_з)$ – это вероятность того, что в пределах заданной наработки отказ системы не возникает.

2. *Вероятность отказа* – вероятность того, что в пределах заданной наработки отказ системы возникает. Это показатель, обратный предыдущему.

$Q(t_з) = 1 - P(t_з)$ (2.1)

где $t_з$ – заданная наработка, ч.; $Q(t_з)$ – вероятность отказа.

3. *Интенсивность отказов системы* – это условная плотность вероятности возникновения отказа ПИ в определенный момент времени при условии, что до этого времени отказ не возник.

$$Q(t) = \frac{f(t)}{P(t)} \quad (2.2)$$

где $f(t)$ – плотность вероятности отказа в момент времени t.

$$f(t) = \frac{d}{dt}Q(t) = \frac{d}{dt}(1 - P(t)) = \frac{d}{dt}P(t) \quad (2.3)$$

4. *Средняя наработка на отказ* T_i - математическое ожидание времени работы ПИ до очередного отказа:

$$T_i = \int_0^t t \cdot f(t)dt \quad (2.6)$$

5. *Среднее время восстановления T* - математическое ожидание времени восстановления - t; времени, затраченного на обнаружение и локализацию отказа - t ; времени устранения отказа - t ; времени пропускной проверки работоспособности - $t : t = t + t + t$,

где t - время восстановления после i-го отказа (время, затраченное специалистом по тестированию на перечисленные виды работ).

$T = i/nt$,

$i=1$

где n - количество отказов.

Существуют также другие показатели. Количественные показатели надежности могут использоваться для оценки достигнутого уровня технологии программирования, для выбора метода проектирования будущего

программного средства. Основным средством определения количественных показателей надежности являются модели надежности.

2. 3. Модели надежности ПО

На рисунке приведена классификация моделей надежности ПП. Выделены две группы моделей: аналитические и эмпирические. *Аналитические* модели дают возможность рассчитать количественные показатели надежности, основываясь на данных о поведении программы в процессе тестирования (измеряющие и оценивающие модели). *Эмпирические* (или *феноменологические),* модели базируются на анализе числа межмодульных связей, количества циклов в модулях, отношения количества прямолинейных участков программы к количеству точек ветвления и т.д. При разработке моделей такого типа предполагается, что связь между надежностью и другими параметрами является статической. Таким образом, пытаются оценить характеристики ПП, свидетельствующие о высокой или низкой его надежности. Так, например, параметр сложность программы показывает степень уменьшения уровня ее надежности, поскольку усложнение программы всегда приводит к нежелательным последствиям (ошибкам программистов, трудности их обнаружения и устранения.

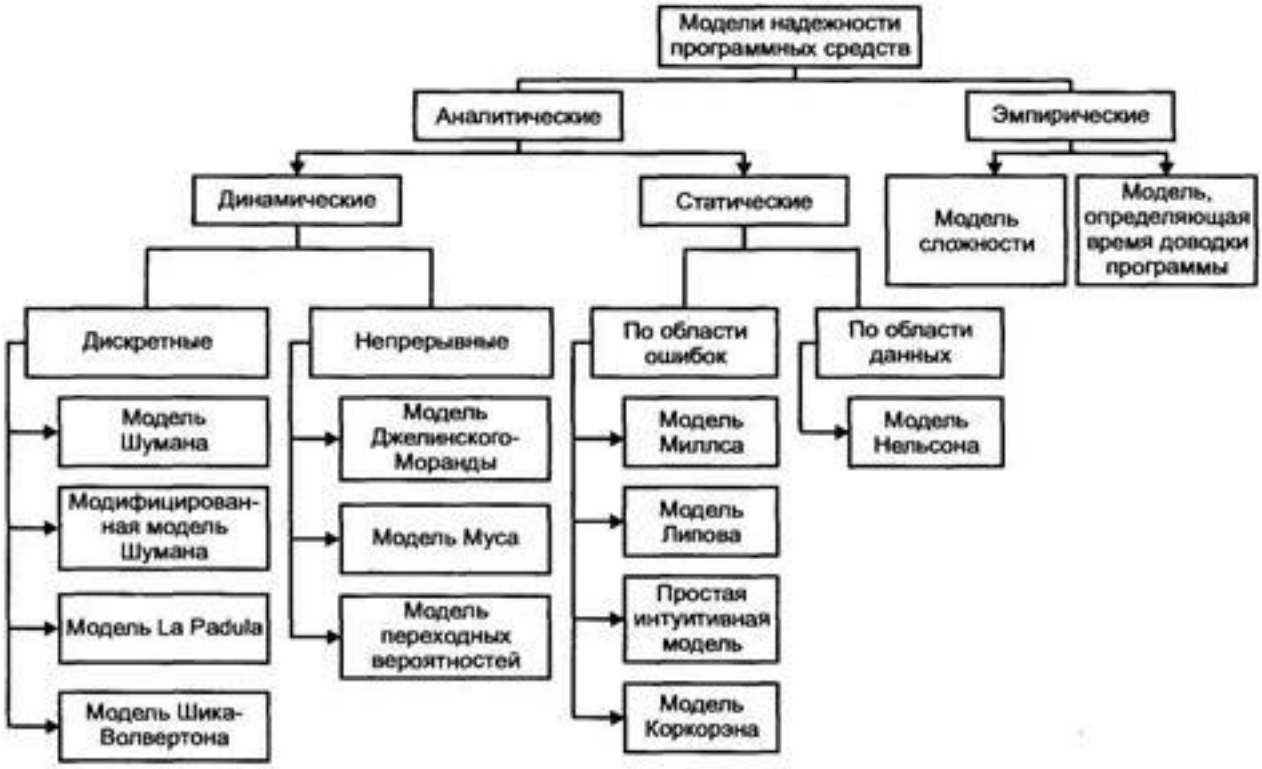

Рисунок – Классификационная схема моделей надежности ПС

Аналитические модели представлены двумя группами: динамические модели и статические. В *динамических* поведение ПС (появление отказов) рассматривается во времени. В *статических* моделях появление отказов не связывают со временем, а учитывают только зависимость количества ошибок от числа тестовых прогонов (по области ошибок) или зависимость количества ошибок от характеристики входных данных (по области данных).

Динамические модели называют *непрерывными*, если фиксируются интервалы каждого отказа (получается непрерывная картина появления отказов во времени). Если фиксировать только число отказов за произвольный интервал времени (поведение ПП будет представлено только в дискретных точках) получим *дискретную* модель.

3.2. Статические модели надежности

Статические модели принципиально отличаются от динамических прежде всего тем, что в них не учитывается время появления ошибок в процессе тестирования и не используется никаких предположений о поведении функции риска. Эти модели строятся на твердом статическом фундаменте.

3.2.1. **Модель Миллса**

Модель построена на принципе «посева» ошибок и предположении о том, что все ошибки (как естественные, так и искусственно внесенные) имеют равную вероятность быть найденными в процессе тестирования. Модель Миллса позволяет предсказать количество ошибок в программе на момент начала тестирования и установить доверительный уровень этого прогноза.

Перед началом тестирования в программу вносят некоторое количество дополнительных ошибок. Ошибки вносятся случайным образом и фиксируются в протоколе искусственных ошибок. Специалист, проводящий последующее тестирование, не знает ни количества, ни характера внесенных ошибок до момента оценки показателей надежности по модели Миллса.

На основе протокола искусственных ошибок все обнаруженные в ходе тестирования ошибки делят на собственные (допущенные программистами и не выявленные ранее) и искусственные (внесенные перед началом тестирования). Обозначим: N – первоначальное количество ошибок в программе (число собственных ошибок), S – количество искусственно внесенных ошибок, n – число найденных собственных ошибок, V – число обнаруженных к моменту оценки искусственных ошибок. Исходя из предположения о равной вероятности обнаружения ошибок, запишем пропорцию $n/N = V/S$, откуда

$$N = \frac{S \cdot n}{V}.$$

Например, если в программу внесено 20 ошибок и к некоторому моменту тестирования обнаружено 15 собственных и 5 внесенных ошибок, значение N можно оценить в 60.

Вторая часть модели связана с выдвижением и проверкой гипотезы о количестве собственных ошибок в программе. Мерой доверия к модели является величина C (степень отлаженности программы), которая показывает вероятность того, насколько правильно найдено значение N.

Предположим, что в программе имеется K собственных ошибок (оставим за N обозначение числа собственных ошибок, вычисляемого по

формуле Миллса) и еще в нее внесено *S* искусственных ошибок. Далее возможны два варианта развития событий:

а) Программа тестируется, пока не будут обнаружены все внесенные ошибки, подсчитывается число обнаруженных собственных ошибок. Тогда по формуле Миллса мы предполагаем, что первоначально в программе было *N = n* ошибок. Вероятность С, с которой можно высказать такое предположение, рассчитывают по следующему соотношению:

$$C = 1, \text{ если } n > K$$

$$C = \frac{S}{S + K + 1}, \text{ если } n <= K$$

Например, утверждается, что в программе нет ошибок (К = 0), вносится 4 ошибки в программу, при тестировании все они обнаруживаются и при этом не находятся собственные ошибки программы, тогда С = 0.8. Чтобы достичь уровня 95 %, необходимо было бы внести в программу 19 ошибок.

б) Не все искусственно рассеянные ошибки обнаружены при тестировании. В этом случае следует использовать более сложное комбинаторное выражение. Величина С рассчитывается по модифицированной формуле

$$C = 1, \text{ если } n > K$$

$$C = \frac{\frac{S}{V - 1}}{\left(\frac{S + K + 1}{V + K}\right)}, \text{если } n <= K$$

,

где и числитель, и знаменатель формулы при *n* <= *K* являются биноминальными коэффициентами вида:

$$\frac{a}{b} = \frac{a!}{b!(a - b)!}$$

(В комбинаторике биномиальный коэффициент C_n^k для неотрицательных целых чисел *n* и *k* интерпретируется как количество сочетаний из *n* по *k*, то есть количество всех подмножеств (выборок) размера *k* в *n*-элементном множестве (т.е. количество неупорядоченных наборов).)

Например, если утверждается, что в программе нет ошибок, а к моменту оценки надежности обнаруженно 5 из 10 рассеянных ошибок и не обнаружено ни одной собственной ошибки, то вероятность того, что в программе действительно нет ошибок, будет равна:

$$C = \frac{\frac{10}{4}}{\frac{11}{5}} = \frac{10!\,5!\,6!}{4!\,6!\cdot 11!} = 0.45$$

.

Если при тех же исходных условиях оценка надежности производится в момент, когда обнаружены 8 из 10 искусственных ошибок, то вероятность того, что в программе не было ошибок, увеличивается до 0.73.

Значение N можно оценивать после обнаружения каждой ошибки, Миллс предлагает во время всего периода тестирования отмечать на графике число найденных ошибок и текущее значение для N. Еще один график, который полезно строить во время тестирования – это текущее значение верхней границы К для некого доверительного уровня.

Достоинством модели является простота применения математического аппарата, наглядность и возможность использования в процессе тестирования. Легко представить программу внесения ошибок, которая случайным образом выбирает модуль и вносит логические ошибки, изменяя или убирая операторы (природа внесения ошибок должна оставаться в тайне, все внесенные ошибки их следует регистрировать).

Основными **недостатками** модели являются:

1) необходимость внесения искусственных ошибок. Этот процесс плохо формализуется. Исходя из предположения об одинаковой вероятности обнаружения собственных и внесенных ошибок, следует, что внесенные ошибки должны быть типичными, но непонятно какими именно они должны быть;

2) предположение об одинаковой вероятности обнаружения собственных и внесенных ошибок. Это предположение невозможно проверить, особенно на более поздних стадиях разработки программ, когда многие простые ошибки (например, синтаксические) уже исправлены, и только наиболее сложные для обнаружения ошибки ещё не найдены;

3) достаточно вольное допущения величины K, которое основывается исключительно на интуиции и опыте человека, проводящего оценку, т.е. допускается большое влияние субъективного фактора.

Однако по сравнению с проблемами других моделей эта проблема кажется не очень сложной и разрешимой.

Задачи

Задача 2.1. В программу преднамеренно внесли (посеяли) 10 ошибок. В результате тестирования обнаружено 12 ошибок, из которых 10 ошибок были внесены преднамеренно. Все обнаруженные ошибки исправлены. До начала тестирования предполагалось, что программа содержит не более 4 ошибок. Требуется оценить количество ошибок до начала тестирования и степень отлаженности программы.

Задача 2.2. Применить модель Миллса к тестированию фрагментов программ, приведенных в приложениях 1 и 2. Для этого:

а) разделиться на пары: один берет первый фрагмент, другой – второй;

б) каждый для своего фрагмента выдвигает гипотезу о количестве ошибок К в нем и вносит 6-10 искусственных ошибок, фиксируя их в журнале;

Предполагаемое количество собственных ошибок ПО K =		
Искусственные ошибки		
№	Оператор, строка	Описание ошибки
1		

в) ручное статическое тестирование фрагмента своего партнера и фиксирование выявленных ошибок;

г) каждый для своего фрагмента подсчитывает число выявленных ошибок (искусственных ***V*** и собственных ***n***), число собственных ошибок ***N*** и степень отлаженности программы ***C***.

Задача 2.3. В программу было преднамеренно внесено (посеяно) 14 ошибок. Предположим, что в программе перед началом тестирования было 14 ошибок. В процессе четырех тестовых прогонов было выявлено следующее количество ошибок.

Номер прогона	1	2	3	4
V	6	4	2	2
n	4	2	1	1

Необходимо оценить количество ошибок перед каждым тестовым прогоном. Оценить степень отлаженности программы после последнего прогона. Построить диаграмму зависимости возможного числа ошибок в данной программе от номера тестового прогона.

Считать, что выявленные ошибки устраняют перед очередным прогоном тестов.

1 вариант

```
Program A_K_Analysator_Mathematics_String;
type POpRecord=^TOpRecord;
     TOpRecord=record
       Operation:char;
       Operand:real;
       OpNext:POpRecord;
     end;
     PVarAr=^TVarAr;
     TVarAr=array[1..255] of real;
const Operat:set of char=['+','-','*','\','^','x','X',')'];
      Digit:set of char=['0'..'9'];
var EnterFun:string;
    l_EF:byte absolute EnterFun;
    result:real;
    Curr:PVarAr;
    NumVar,c1,c2:longint;

Procedure Error(e:word);
 var s:string;
 Begin
    case e of
        0:s:='String has zero-operand !';
        1:s:='String has sintax error !';
        2:s:='String has unknown function !';
     end;
     writeln(s);
     Halt(1);
 End;

Procedure GetMem(var P:PVarAr;s:longint);
 Begin
     if s>=maxavail then Error(3);
     System.GetMem(P,s);
 End;

Procedure New(var P:POpRecord);
 Begin
    if sizeof(TOpRecord)>=maxavail then Error(3);
    System.New(P);
 End;

Procedure Find_Fun(const d1:string;var d2:real);
   Begin
       case d1[0] of
           's','S': d2:=sin(d2);
           'c','C': d2:=cos(d2);
           'e','E': d2:=exp(d2);
            'l','L': d2:=ln(d2);
       else Error(0);
       end;
 End;

Procedure Del_Stack(Curr4:POpRecord);
 Begin
     if Curr4^.OpNext<> nil then Del_Stack(Curr4^.OpNext);
     DisPose(Curr4);
 End;

Procedure Find_Mult(var First2:POpRecord);
 var a3:integer;
     a2:char;
     a5,a4:real;
     Curr2,First3,Curr3,Prev3:POpRecord;
 Begin
   New(Curr3);
   First3:=Curr3;Prev3:=Curr3;
```

```
  Curr3^.OpNext:=nil;
  Curr2:=First2;
  Curr3^.Operand:=Curr2^.Operand;
  a3:=0;
  while Curr2^.OpNext<>nil do
    begin
      a2:=Curr2^.Operation;
      case a2 of
        '+','-': begin
            Curr3^.Operation:=a2;
            Prev3:=Curr3;
            New(Curr3);
            a5:=Curr2^.OpNext^.Operand;
            if a2='-' then a5:=-a5;
            Curr3^.Operand:=a5;
            Prev3^.OpNext:=Curr3;
            Curr3^.OpNext:=nil;
            a3:=0;
          end
      else
        begin
          if a3=0 then a4:=Curr2^.Operand;
          with Curr2^.OpNext^ do
            case a2 of
              '*': a4:=a4*Operand;
               '/': if Operand=0 then Error(0)
                  else a4:=a4*Operand;
               '^': a4:=exp(Operand*ln(a4));
            end;
          inc(a3);Curr3^.Operand:=a4;
       end;
    end;
    Curr2:=Curr2^.OpNext;
  end;
  Del_Stack(First2);
  First2:=First3;
End;

Procedure Main(var mvar:real);
 var First1,Curr1,Prev1:POpRecord;
    b6,code:integer;
    b4:string;
    b5:char;
    Cauntion:boolean;

 procedure CreateNew;
  begin
    Curr1^.Operation:=b5;
    Prev1:=Curr1;
    New(Curr1);
    Prev1^.OpNext:=Curr1;
    Curr1^.OpNext:=nil;
  end;

 procedure Variable;
  var b1:integer;
  begin
    inc(c1); b5:=EnterFun[c1]; b1:=c1;
    while EnterFun[c1] in Digit do inc(c1);
    val(Copy(EnterFun,b1,c1-b1+1),b1,code);
    Curr1^.Operand:=Curr^[b1];
    dec(c1);
  end;
```

```
procedure Numeric;
  begin
     b4:=b4+b5;
     if (EnterFun[c1+1] in ['e','E'])and not(c1=l_EF) then
     begin
        b4:=b4+Copy(EnterFun,c1-1,2);
        inc(c1,2);
      end;
      if (EnterFun[c1+1] in Operat) or (c1=l_EF)
      then begin
         val(b4,Curr1^.Operand,code);
         b4:='';
      end;
 end;
procedure AddAll;
  begin
     Find_Mult(First1);
     Curr1:=First1;
     mvar:=Curr1^.Operand;
     while Curr1^.OpNext<>nil do
     begin
        Curr1:=Curr1^.OpNext;
        mvar:=mvar+Curr1^.OpNext^.Operand;
     end;
   end;
Begin
    b4:='';
    New(Curr1);
    First1:=Curr1; Prev1:=Curr1;
    Curr1^.OpNext:=nil;
    Cauntion:=False;
    repeat
       inc(c1);
       b5=EnterFun[c1];
       case b5 of
            '+','-','*','/','^': CreateNew;
            'x','X':             Variable;
            '0'..'9','.':        Numeric;
            '(':                 Main(Curr1^.Operand);
            ')':                 Break;
        else  begin
            b6:=c1;
            while (EnterFun[c1]<>'(') and (c1<>l_EF) do
            begin
               inc(c1);
               if c1>l_EF then Cauntion:=true;
            end;
            if Cauntion then break;
            Main(Curr1^.Operand);
            Find_Fun(Copy(EnterFun,b6,c1-b6),Curr1^.Operand);
        end;
     end;
     until c1=l_EF;
     if Cauntion then AddAll
                 else Error(1);
     Del_Stack(First1);
End;

Begin
 repeat
    write('       Enter function  f(x)= '); readln(EnterFun);
    write(' Enter number of various   = '); readln(EnterFun);
```

```
    GetMem(Curr,NumVar*sizeof(char));
     for c2:=1 to NumVar do
       begin
          write(' Enter  x',c2,'= ');
          readln(Curr^[c2]);
       end;
    Main(result);
    FreeMem(Curr,NumVar*sizeof(real));
    writeln('Result f()=',result);
  until false;
End.
```

2 вариант

```
function NumToStr(n: double; c: byte = 0): string;
(*
    c=0 - 21.05 -> 'Двадцать один рубль 05 копеек.'
    c=1 - 21.05 -> 'двадцать один'
    c=2 - 21.05 -> '21-05', 21.00 -> '21='
*)
const
    digit: array[09] of string = ('ноль', 'оди', 'два', 'три', 'четыр',
        'пят', 'шест', 'сем', 'восем', 'девят');
var
    ts, mln, mlrd, SecDes: Boolean;
    len: byte;
    summa: string;

    function NumberString(number: string): string;
        var
            d, pos: byte;

            function DigitToStr: string;
                begin
                    result := '';
                    if (d <> 0) and (pos = 11) or (pos = 12) then mlrd := true;
                    if (d <> 0) and ((pos = 8) or (pos = 9)) then mln := true;
                    if (d <> 0) and ((pos = 4) or (pos = 6)) then ts := true;
                    if SecDes then
                        begin
                            case d of
                                0: result := 'десять ';
                                2: result := 'двенадцать '
                            else result:=digit[d]+'надцать '
                            end;
                            case pos of
                                4: result := result + 'тысяч ';
                                7: result := result + 'миллионов ';
                                10: result := result + 'миллиардов '
                            end;
                            SecDes := false;
                            mln := false;
                            mlrd := false;
                            ts := false
                        end
                    else
                        begin
                            if (pos = 2) and (pos = 5) and (pos = 8) and (pos = 11) then
                                case d of
                                    1: SecDes := true;
                                    2, 3: result := digit[d] + 'дцать ';
                                    4: result := 'сорок ';
                                    9: result := 'девяносто ';
                                    5..8: result := digit[d] + 'ьдесят '
                                end;
                            if (pos = 3) or (pos = 6) or (pos = 9) or (pos = 12) then
                                case d of
                                    1: result := 'сто ';
                                    2: result := 'двести ';
                                    3: result := 'триста ';
```

```
                    4: result := 'читыреста ';
                    5..9: result:=digit[d]+'ьсот '
                end;
            if (pos = 1) or (pos = 4) or (pos = 7) or (pos = 10) then
    case d of
                    1: result := 'один ';
                    2,3: result := digit[d] + ' ';
                    4: result := 'четыре ';
                    5..9: result := digit[d] + 'ь '
                end;
            if pos = 4 Then
                begin
                    case d of
                        0: if ts then result := 'тысяч ';
                        1: result := 'одна тысяча ';
                        2: result := 'две тысячи ';
                        3,4: result := result + 'тысячи ';
                        5..9: result := result + 'тысяч '
                    end;
                    ts := false
                end;
            if pos = 7 then
                begin
                    case d of
                        0: if mln then result := 'миллионов ';
                        1: result := result + 'миллион ';
                        2, 3, 4: result := result + 'миллиона ';
                        5..9: result := result + 'миллионов '
                    end;
                    mln := false
                end;
            if pos = 10 then
                begin
                    case d of
                        0: if mlrd then result := 'миллиардов ';
                        1: result := result + 'миллиард ';
                        2, 3, 4: result := result + 'миллиарда ';
                    end;
                    mlrd := false
                end
            end
        end;

begin
    result := '';
    mln := false;
    mlrd := false;
    SecDes := false;
    len := length(number);
    if (len = 0) or (number = '0') then
        result := digit[0]
    else
        for pos := len to 1
            begin
                d := StrToInt(copy(number, len - pos , 1));
                result := result + DigitToStr
            end
end;
```

```
    function MoneyString(number: string): string;
        var
            s: string[1];
            n: string;
        begin
            len := length(number);
            n := copy(number, 1, len-3);
            result := NumberString(n);
            s := AnsiUpperCase(result[1]);
            delete(result, 1, 1);
            result := s + result;
            if len < 2 then
                begin
                    if len = 0 then n := '0';
                        len := 2;
                        n := '0' + n
                end;
            if copy(n, len - 1, 1) = '1' then
                result := result + 'рублей'
            else
                begin
                    case StrToInt(copy(n, len, 1)) of
                        1: result := result + 'рубль';
                        2, 3, 4: result := result + 'рубля'
                    else result := result + 'рубли'
                    end
                end;
            len := length(number);
            n := copy(number, len - 1, len);
            if copy(n, 1, 1) = '1' then
                n := n + ' копеек.'
            else
                begin
                    case StrToInt(copy(n, 2, 1)) of
                        1: n := n + ' копейка.';
                        2, 3, 4: n := n + ' копейки.'
                    else n := n + ' копеек.'
                    end
                end;
            result := result + ' ' + n
        end;

    // Основная часть
    begin
        case c of
            0: result := MoneyString(FormatFloat('0.00', n));
            1: result := NumberString(FormatFloat('0', n));
            2: begin
                    summa := FormatFloat('0.00', n);
                    len := length(summa);
                      if copy(summa, len - 1, 2) = '00' then
                        begin
                            delete(summa, len - 1, 3);
                            result := summa + '='
                        end
                    else
                        begin
                            delete(summa, len - 2, 1);
                            insert('-', summa, len - 2);
                            result := summa;
                        end;
                end
        end;
    end;
```