Государственный комитет Российской Федерации
по высшему образованию

Самарский государственный аэрокосмический университет
имени академика С. П. Королева

ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ В ТУРБО ПАСКАЛЕ

Методические указания к лабораторным работам

Самара 1996

Составители: В. В. Семенов, Т. В. Макаренко

УДК 681.3.06

Использование подпрограмм в Турбо Паскале : Метод. указания к лабораторным работам / Самар. гос. аэрокосм. ун-т ; Сост. В. В. Семенов , Т. В. Макаренко. Самара, 1996. 16 с.

Содержатся рекомендации по выполнению лабораторных работ на популярном алгоритмическом языке программирования Турбо Паскаль для решения различных задач обработки данных с использованием подпрограмм. Подробно рассматриваются все вопросы, связанные с применением процедур и функций в программах. Приведены многочисленные примеры и даны рекомендации по способам передачи данных в подпрограммы и обратно. Предназначены для выполнения лабораторных работ по дисциплинам "ЭВМ и программирование", "Вычислительная техника и программирование", "Информатика" для студентов всех специальностей. Составлены на кафедре "Программное обеспечение вычислительных систем".

Печатаются по решению редакционно-издательского совета Самарского государственного аэрокосмического университета им. академика С. П. Королева

Рецензент А. О. Новиков

В языке Турбо Паскаль имеется два вида подпрограмм - процедуры и функции. Имея один и тот же смысл и аналогичную структуру, процедуры и функции различаются назначением и способом их использования.

Процедуры служат для задания совокупности действий, направленных на изменение внешней по отношению к ним программной обстановки, например, определения новых значений переменных или записи информации во внешние файлы (в частности, печать данных). Вызов процедуры на выполнение реализуется специальным оператором процедуры.

Смысл функций заключается в первую очередь в том, чтобы определить алгоритм вычисления нового значения некоторого простого или ссылочного типа. В этом отношении функции подобны выражениям, которые также вычисляют значения. Поэтому вызов функции является одним из допустимых операндов выражения, обозначая в нем то значение, которое вычисляет функция.

Структура подпрограмм

Напомним, что программа на языке Турбо Паскале всегда состоит из двух частей:

- *раздел описаний*, в котором описываются и определяются все объекты (именованные константы, переменные, типы, метки, процедуры и функции) используемые в программе;
- *раздел операторов*, которые непосредственно выполняют действия по реализации алгоритма.

Совокупность описаний и определений и следующая за ней последовательность операторов называется блоком. Кроме того, программа может быть снабжена заголовком, который задает имя программы. Текст всей программы должен завершаться символом "." (точка).

Структура подпрограммы почти буквально повторяет структуру всей программы, что имеет глубокий смысл (часть подобна целому). Так как любое имя в программе обязательно описывается перед тем, как оно появится среди исполняемых операторов, то и подпрограмма, как и любой другой объект,

должна быть предварительно определена в разделе описаний основной программы.

Определением (описанием) подпрограммы является ее собственный текст, который содержит заголовок и тело подпрограммы (блок).

В общем виде структуру программы, содержащей подпрограмму, можно представить в следующем виде:

```
program имя программы;
  ...
    заголовок подпрограммы
      раздел описаний подпрограммы
    begin                                  } Текст подпрограммы
      раздел операторов подпрограммы
    end;
  ...
begin
  раздел операторов основной программы
end.
```

В заголовке указываются тип подпрограммы (функция или процедура), имя подпрограммы и *формальные параметры*, если они есть. Для функции, кроме того, указывается тип возвращаемого ею результата. За заголовком следует тело подпрограммы, которое (подобно основной программе) состоит из раздела описаний и раздела исполняемых операторов. В разделе описаний подпрограммы могут, в свою очередь, находиться описания других подпрограмм низшего уровня, а в них - описания других подпрограмм и т. д.

Процедуры. Параметры формальные и фактические

Когда какая-либо часть программы используется более одного раза, то вовсе необязательно повторять текст; эту часть можно оформить в виде процедуры, дав ей имя и вызывая каждый раз, когда необходимо выполнить эту часть программы. Описание процедуры, как следует из вышесказанного, состоит из заголовка и тела (блока) процедуры. Заголовок процедуры имеет вид:

PROCEDURE **имя** (список *формальных параметров*);

где **Procedure** - зарезервированное (служебное) слово,
имя - любой допустимый идентификатор.

Список *формальных параметров* не обязателен и может отсутствовать (в этом случае скобки также не ставятся). Если же он есть, то в нем должны быть перечислены имена переменных и их тип, например:

```
Procedure FANTA ( a: Real; B1: Integer; Status: Char );
```

Описания переменных в списке отделяются друг от друга точками с запятой. Несколько однотипных переменных можно объединять в подсписки, например:

```
Procedure PEPSI ( A, b, Omega: Real; Alfa, Sur_1, Log, povs: Boolean );
```

Операторы подпрограммы рассматривают список *формальных параметров* как своеобразное расширение раздела описаний, поэтому все переменные из этого списка могут использоваться в любых выражениях внутри подпрограммы. Вот простейший пример: процедура печати двух целых чисел в обратном порядке:

```
Procedure Reverse ( a, b: Integer );
  begin
    WriteLn ( b:3, a:3 );
  end;
```

Для выполнения операторов, определенных в подпрограмме, т. е. для вызова процедуры, используется оператор процедуры, который состоит из ее имени и списка *фактических параметров* (если в определении процедуры был указан список *формальных параметров*). После завершения выполнения процедуры управление передается на оператор, непосредственно следующий за вызовом процедуры. Например, для вызова рассмотренной выше процедуры Reverse могут быть использованы следующие операторы:

```
REVERSE ( x, y );
Reverse ( 4, 5*Sqrt ( y-x ) );
```

Вряд ли использование такой процедуры может представлять практический интерес, однако из этого примера видно, что *фактические параметры* могут быть константами (4), именами переменных (x, y) или выражениями (5*Sqrt (y-x)). Каждый раз при вызове процедуры Reverse (,) вычисляются *фактические параметры*, и их *значения* подставляются на место соответствующих им *формальных параметров* a и b.

При вызове любой подпрограммы необходимо неукоснительно соблюдать следующее правило. *Количество, тип и порядок следования фактических параметров должны строго соответствовать количеству, типу и порядку следования соответствующих формальных параметров !*

Параметры-значения и параметры-переменные

Предположим, что вместо печати значений в обратном порядке необходимо поменять местами значения двух переменных целого типа. Приведенная ниже процедура, несмотря на кажущуюся очевидность, для этого совсем не пригодна:

```
Procedure Swop ( a, b: Integer );
  Var just: Integer;
    begin
      Just := a ;  a := b ;  b := Just ;
    end;
```

Предположим, что значения переменных **x** и **y** предварительно определены, а процедура **Swop** вызывается следующим образом:

```
Swop ( x, y );
```

Далее произойдет следующее. Значение переменной **x** будет помещено в **a**, а значение **y** в **b**. В подпрограмме значения переменных **a** и **b** действительно поменяются местами, но после возвращения в основную программу переменные **x** и **y** не изменятся. Дело в том, что формальные параметры **a** и **b** в заголовке подпрограммы описаны как *параметры - значения*.

Для решения поставленной задачи необходимо задать формальные параметры как *параметры - переменные*. В этом случае подпрограмма получит доступ к переменным из основной программы. Для этого перед именем соответствующих переменных в списке формальных параметров заголовка подпрограммы указывается служебное слово **VAR**. Например, для рассматриваемого примера заголовок процедуры будет иметь следующий вид:

```
Procedure Swop ( Var a, b: Integer );
```

Поэтому, *в заголовке процедуры указывайте* **VAR** *перед теми переменными, значения которых должны быть изменены!*

Очевидно, что если какие-то формальные параметры подпрограммы были заданы как *параметры - переменные*, то при вызове этой подпрограммы соответствующие им фактические параметры могут быть только переменными. Для *параметров - значений*, как уже показывалось на примере, соответствующие фактические параметры при вызове подпрограммы могут быть любым выражением подходящего типа.

Параметры-массивы и параметры-строки

Пусть перед нами поставлена задача нахождения максимального элемента в одномерном массиве произвольной величины и индекса (номера) этого элемента. Разобьем эту задачу на три самостоятельные части, оформленные в виде процедур:

а) ввод размера одномерного массива и его элементов;

б) нахождение максимального элемента в массиве и его индекса;

в) вывод на печать исходного массива и результатов.

Следовательно, нам необходимо знать, как передать одномерный массив из подпрограммы (процедура ввода) и в подпрограмму (процедуры поиска максимума и вывода на печать).

Прежде всего, необходимо объявить в разделе описаний основной программы новый производный тип **VECTOR**, который будет иметь структуру одномерного массива из **MMM** элементов вещественного типа:

```
Type VECTOR = ARRAY [1 .. MMM ] Of Real;
```

Теперь этот новый производный тип можно использовать так же, как и стандартные типы. Например, описание массива **C** будет выглядеть следующим образом:

```
Var C: Vector;
```

При вызове процедур мы будем использовать переменную (массив) **C** в качестве фактического параметра. При задании массива в качестве формального параметра в заголовке подпрограммы также используется производный тип **Vector**. Переменная с производным типом, как и любая другая переменная в списке формальных параметров, может задаваться в качестве параметра-значения или параметра-переменной.

Пример.

```
Program MAXIMUM_in_Vector;
  Const MMM = 100;
  Type VECTOR = ARRAY [1 .. MMM ] Of Real;
  Var C: Vector;
      MAX: Real;
      MAK, IMAX, I: Integer;

PROCEDURE INPUT( Var C: Vector; Var MAK: Integer );
{ Ввод размера МАССИВА и его элементов }
  Var j: Integer;
    Begin
      Write ( 'Введите количество элементов = ' );
      ReadLn (MAK);
      For j := 1 to MAK do
        begin
          Write ( 'Введите элемент C[', j, ']=' );
          ReadLn ( C[j] );
        end;
    End; { КОНЕЦ процедуры INPUT }

PROCEDURE MAXIM ( B: Vector; N: Integer; Var INDEX: Integer;
                                          Var MAXIC: Real );
{ В массиве B из N элементов найти максимальный элемент - MAXIC
  и его индекс - INDEX }
```

```
  Var i: Integer;
    Begin
      MAXIC := B[1]; INDEX := 1;
      For i := 2 to N do
        If B[i] > MAXIC Then begin MAXIC := B[i]; INDEX := i; end;
    End; { КОНЕЦ процедуры MAXIM }
PROCEDURE OUTPUT( C: Vector; MAK, IMAX: Integer; MAX: Real );
{ Вывод МАССИВА и результатов }
  Var i: Integer;
    Begin
      WriteLn ( ' Исходный массив C:' );
      For i := 1 to MAK do
        begin
          Write( ' C[', i:2, ']=', C[i]:4:2 );
          { Вывод по 5 элементов в строке }
          If (i div 5)*5 = i Then WriteLn ( '' );
        end;
      WriteLn ( '' );
      WriteLn ( ' МАКСИМУМ: C[', IMAX, ']=', MAX:5:1 );
    End; { КОНЕЦ процедуры OUTPUT }
BEGIN { НАЧАЛО основной программы }
  INPUT  ( C, MAK );
  MAXIM  ( C, MAK, IMAX, MAX );
  OUTPUT( C, MAK, IMAX, MAX );
END.  { КОНЕЦ основной программы }
```

Использование в качестве параметров подпрограмм строковых переменных (тип **STRING**) имеет одну особенность. При определении переменной в разделе описаний основной программы и в списке формальных параметров подпрограммы нельзя указывать максимальную длину строки, например:

```
Var ST: String;
  ...
Procedure COLA ( Var TXT: String );
  ...
```

Можно поступить иначе. По аналогии с использованием массивов в качестве параметров подпрограмм описывается новый производный тип, имеющий структуру символьной строки с заданной максимальной длиной, который можно применять наравне с простыми стандартными типами. Например:

```
Type TEXT = String [ 57 ];
```

```
...
Var ST: TEXT;
...
Procedure COLA ( Var TXT: TEXT );
...
```

Локализация имен. Побочный эффект

Как уже было сказано, в разделе описаний подпрограммы могут в свою очередь содержаться определения других подпрограмм, которые будем считать вложенными, т. е. подпрограммами следующего уровня. Предположим, что в некоторой программе определены две подпрограммы **A** и **B**. В разделе описаний подпрограммы **A** содержатся определения подпрограмм **A1** и **A2**, которые являются вложенными в подпрограмму **A**. В подпрограмме **B** также описаны две подпрограммы **B1** и **B2**. И, наконец, в подпрограмме **B2** определены еще две подпрограммы **B21** и **B22** самого нижнего для данной программы уровня. На рисунке наглядно представлена структура такой программы:

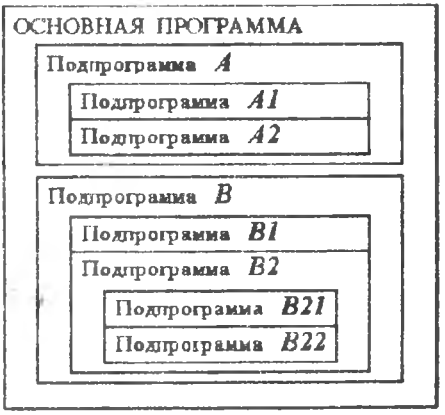

Рис.

Формальные параметры подпрограммы, заданные как параметры-значения, и все имена объектов, определенные в подпрограмме (именованные константы, переменные, типы, метки и другие подпрограммы), называются *локальными* именами и считаются доступными только в данной подпрограмме. Например, в рассматриваемом примере из основной программы можно вызвать подпрограммы **A** и **B**, но нельзя вызвать ни одну из вложенных в них подпрограмм **A1**, **A2**, **B1** и т. д.

Локальные переменные (например, переменные **i** из процедур **Maxim** и **Output** в программе предыдущего раздела) создаются при вызове подпрограммы. После завершения подпрограммы *локальные* переменные уничтожаются. Таким образом, переменная **i** из процедуры **Maxim** не имеет ничего общего с переменной **i**, объявленной в процедуре **Output**.

С другой стороны, все объекты, описанные в подпрограмме (в том числе и другие подпрограммы), т. е. *локальные* для нее, являются доступными для всех вложенных в нее подпрограмм. Поэтому мы будем называть их *глобальными* по отношению к вложенным подпрограммам нижних уровней. Например, из подпрограмм **A1** и **A2** можно вызвать подпрограмму **A**, использовать переменные из нее и из основной программы.

При взаимодействии подпрограмм одного уровня вступает в силу основное правило Турбо Паскаля: любой объект (в том числе и подпрограмма) должен быть описан перед тем, как он будет использован. Другими словами, подпрограмме доступны только те имена верхнего уровня (*глобальные* по отношению к ней), которые описаны до нее. Например, из подпрограммы **B** можно вызвать подпрограмму **A**, но из **A** вызвать **B** невозможно (точнее, такая возможность появляется только при использовании *опережающего описания*).

Когда подпрограмма обращается к какой-либо переменной, то, прежде всего, отыскивается *локальная*, т. е. внутренняя переменная с данным именем. Изменение значения этой переменной внутри подпрограммы никак не влияет на *глобальные* переменные с таким же именем. Если же *локальной* переменной с данным именем не существует, то используется *глобальная* переменная из вызывающей подпрограммы более высокого уровня или из другой подпрограммы, определенной ранее.

Если подпрограмма *изменяет* значение переменной, объявленной вне этой подпрограммы, то говорят, что подпрограмма имеет *побочный эффект*.

Побочные эффекты часто появляются случайно. Многократно используя переменные с короткими именами вроде **a**, **b**, **c** и забывая объявлять их локально, программист имеет потенциальный источник неприятностей.

В небольших программах, вероятно, проще делать все переменные *глобальными*. В этом случае *побочные эффекты* даже полезны. Однако, бесконтрольное или небрежное использование *побочных эффектов* в большой программе вряд ли допустимо. В некоторых случаях целесообразно определить в программе несколько *глобальных* переменных с понятными и не часто встречающимися именами.

Функции

Процедуры и функции, являющиеся двумя разновидностями подпрограмм в Турбо Паскале, имеют много общего. Большинство введенных ранее

понятий, таких как формальные и фактические параметры и их соответствие, параметры-значения и параметры-переменные, передача в подпрограмму массивов и символьных строк, локализация имен и побочные эффекты, в равной степени относится и к использованию функций.

Как и процедура, подпрограмма - функция оформляется в виде самостоятельного фрагмента программы и помещается в раздел описаний основной программы или другой подпрограммы. Описание функции также содержит заголовок и тело (блок) функции. Заголовок функции имеет вид:

FUNCTION **имя** (список *формальных параметров*)**: тип ;**

где **Function** - зарезервированное (служебное) слово,
имя - любой допустимый идентификатор;
тип - тип возвращаемого функцией результата.

В Турбо Паскале нет функции вычисления тангенса угла, заданного в радианах. Вот пример определения такой функции:

```
Function TAN ( X: Real ): Real;
  Begin
    Tan := Sin(X)/Cos(X);
  End;
```

Функция **Tan** после определения может использоваться в программе наряду с другими стандартными функциями **Sin**, **Exp**, **Sqrt** и т. д. При ее вызове необходимо задавать один фактический параметр вещественного типа (**Real**), возвращаемый функцией результат также имеет вещественный тип.

Таким образом, смысл функции заключается в задании алгоритма вычисления некоторого *единственного значения* и организации возврата (передачи) этого значения в точку вызова. Возврат вычисленного значения организуется следующим образом. Среди операторов, образующих тело функции, должен быть хотя бы один оператор присваивания, в левой части которого содержится *имя функции*, а в правой - выражение того же типа, что и тип возвращаемого функцией результата. Таких операторов может быть несколько, важно лишь, чтобы хотя бы один из них срабатывал в процессе выполнения функции. В противном случае - результат функции будет *неопределенным*.

Рекурсия

Рекурсия - это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов вызывает "саму себя".

Рекурсия достаточно широко применяется в программировании, что основано на рекурсивной природе многих математических алгоритмов.

В качестве примера приведем популярный алгоритм вычисления факториала от неотрицательного целого числа, определяемого так:

$$0! = 1 ;$$
$$1! = 1 ;$$
$$2! = 1 \cdot 2 ;$$
$$\ldots$$
$$N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (N-1) \cdot N .$$

Алгоритм основан на очевидном соотношении:

$$N! = (N-1)! \cdot N ,$$

что позволяет для вычисления факториала использовать результат точно такого же вычисления для предыдущего числа. Пример рекурсивной функции:

```
Function Fact ( N: Integer ): LongInt;
  Begin
    If N = 0
       Then Fact := 1
       Else Fact := N * Fact (N-1);
  End;
```

Вызов функции **Fact** с фактическим значением, больше нуля, приведет к тому, что в теле функции рекурсивно вызовется сама функция **Fact** с аргументом **N - 1**, та, в свою очередь, может вызвать функцию со значением, уменьшенным на единицу, и т. д. Цепочка рекурсивных вызовов будет продолжаться до тех пор, пока не будет вызвана последняя функция **Fact** с фактическим параметром, равным нулю. Результат этой функции будет возвращен в вызвавшую ее подпрограмму, та, в свою очередь, возвратит результат в предыдущую и т. д. Например, для того чтобы вычислить факториал числа 12, функция **Fact** создаст цепочку из двенадцати последовательных вызовов, которая после выполнения (в последней подпрограмме) условия **N = 0** начнет раскручиваться в обратном порядке.

Рекурсивная форма организации алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы, но при выполнении она, как правило, медленнее и может вызвать переполнение программного стека, поскольку каждый очередной рекурсивный вызов приводит к образованию новых *локальных объектов* подпрограммы, которые существуют независимо друг от друга до тех пор, пока активизированы незавершенные рекурсивные вызовы.

Опережающее описание

Достаточно часто встречается ситуация, когда подпрограмма вызывает другую подпрограмму, описание которой располагается после вызывающей.

Например:

```
Procedure A ( Area, C1: Real; Gamma: Integer );
  Begin
    . . .
    B ( Area );    <------ Ошибка! Неизвестный идентификатор.
    . . .
  end;
Procedure B ( X: Real );
  . . .
```

Как отмечалось ранее, подпрограмма (как и любой другой объект) должна быть описана перед тем, как она будет использована. Поэтому из подпрограммы **A** нельзя вызвать подпрограмму **B**, поскольку эта процедура описывается ниже подпрограммы **A** и информацией о ней компилятор еще не располагает. В данном случае можно просто поменять местами описания подпрограмм. Однако в реальных программах может возникнуть необходимость взаимного вызова подпрограмм, когда простая перестановка строк не даст требуемого результата. В этом случае вводится так называемое *опережающее описание* подпрограммы. Для рассматриваемого примера это будет выглядеть так:

```
Procedure B ( X: Real ); Forward;
Procedure A ( Area, C1: Real; Gamma: Integer );
  Begin
    . . .
    B ( Area );
    . . .
  end;
Procedure B ;
  . . .
```

Как видно, *опережающее описание* заключается в том, что объявляется лишь заголовок процедуры **B**, а ее тело заменяется стандартной директивой **FORWARD** (полное описание - *впереди*). Поэтому *опережающее описание* подпрограммы часто называют *ссылкой вперед*. Теперь в процедуре **A** можно использовать обращение к процедуре **B** - ведь она уже частично описана, точнее известны ее формальные параметры, и компилятор может правильным образом организовать ее вызов. Полное определение подпрограммы можно расположить в любом месте программы (до раздела операторов основной программы). В заголовке полного описания можно не указывать описанные ранее формальные параметры.

Модули

Для того чтобы написать большую программу, вовсе необязательно создавать и отлаживать ее целиком, сосредоточив в одном файле исходные тексты всех определений и подпрограмм. Разработка крупного проекта вообще невозможна без разделения на несколько составных частей, которые разрабатываются самостоятельно. В данном разделе даются общие понятия, связанные с модульным принципом организации программ.

Модуль - это совокупность (библиотека) программных объектов, предназначенных для *использования* другими модулями и программами. Каждый модуль имеет определенное внешнее сходство с отдельной программой, однако сам по себе не является выполняемым.

В Турбо Паскале версии 6.0 имеется восемь стандартных модулей, в которых содержится большое число разнообразных типов, констант, процедур и функций. Этими модулями являются **System**, **DOS**, **Crt**, **Printer**, **Graph**, **Overlay**, **Turbo3** и **Graph3**. Модули **Graph**, **Turbo3** и **Graph3** содержатся в одноименных **TPU**-файлах, остальные входят в состав библиотечного файла **TURBO.TPL**. Лишь один модуль **System** подключается к любой программе автоматически, все другие становятся доступны только после указания их имен в списке, следующем за служебным словом **USES**. Использование объектов стандартных модулей позволяет наряду со средствами классического Паскаля использовать дополнительные возможности, такие, например, как доступ к средствам дисковой операционной системы MS-DOS, управление графическим и текстовым режимом дисплея, генерация звука и создание временных задержек, вывод результатов работы программ на принтер, создание громоздких программ с перекрытиями, совместимость с программами ранней версии 3.0.

Турбо Паскаль версии 7.0 содержит дополнительно шесть новых модулей, которые обеспечивают работу с новым стандартным типом **PChar**, а также обеспечивают создание программ, работающих в среде Windows.

Целые наборы модулей представляют объектно-ориентированные библиотеки, такие, например, как Turbo Vision, Object Professional или Object Windows.

И, наконец, пользователь может сам создавать библиотеки подпрограмм и других определений, организуя их в виде собственных модулей и используя в своих программах наряду со стандартными. Однако этот вопрос выходит за рамки данных методических указаний и мы отсылаем интересующихся к соответствующей литературе.

РЕКОМЕНДУЕМЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Епанешников А., Епанешников В. Программирование в среде Turbo Pascal 7.0. М.: ДИАЛОГ - МИФИ, 1993. 288 с.

Фаронов В. В. Турбо Паскаль (в 3-х книгах). Книга 1. Основы Турбо Паскаля. М.: Учебно-инженерный центр МВТУ-ФЕСТО ДИДАКТИК, 1992. 304 с.

Зуев Е. А. Система программирования Turbo Pascal. М.: Радио и связь, 1991. 288 с.

Использование подпрограмм в Турбо Паскале

Составители: Семенов Валерий Владимирович
Макаренко Татьяна Васильевна

Редактор Т. И. Кузнецова

Техн. редактор Н. М. Каленюк

Подписано в печать 23.04.96 Формат 60 x 84 1/16
Бумага офсетная. Печать офсетная.
Усл.печ.л. 0,93 . Уч.-изд.л. 1,0. Усл.кр.-отт. 1,05
Тираж 200 экз. Заказ 143.

Самарский государственный аэрокосмический университет имени академика С. П. Королева.
443086 г. Самара, Московское шоссе, 34.

Издательство Самарского государственного аэрокосмического университета.
443001 г. Самара, ул. Ульяновская, 18.