Pascal Next manual

Nikita Kultin

Publisher: Nikita Kultin

9F9E5108-4263-4230-B5FF-9CB451C88A2C

03.12.2023

Contents

About this book

This book is a description of the new programming language Pascal Next.

The book is addressed to those who are familiar with the basics of programming, know any programming language, and have the skill of developing entry-level computer programs.

The purpose of the book is to show the capabilities of the Pascal Next programming language.

The author of the Pascal Next programming language, the developer of the Pascal Next compiler and the Pascal Next development environment is Nikita Kultin.

kultin.nikita@gmail.com

Pascal Next

Pascal Next is a compiled programming language and development environment for beginning programmers, focused on solving the problem of teaching the basics of programming.

Pascal Next language

The syntax of Pascal Next is based on the syntax of "classic" Pascal. At the same time, the syntax of selection and loop instructions was borrowed from Basic, which made it possible to eliminate redundant `begin` keywords.

The language allows you to work with integer and real numbers, strings, one-dimensional and two-dimensional arrays, and text files. The language has built-in mathematical functions (`Sqrt`, `Sin`, `Cos`, `Arctg`, `Trunc`, `Roung`, `Random`), type conversion functions (`IntToStr`, `StrToInt`, `FloatToStr`, `StrToFloat`), string manipulation functions (`Legth`, `Pos`, `Substr`, `UpCase`, `LowCase`) and dates (`getDay`, `getMonth`, `getYear`, `getDayOfWeek`, `getTime`); it is possible to initialize the array in the declaration instruction; both multi-line and single-line comments can be included in the program text.

Development environment and compiler

The Pascal Next development environment, which is a Win32 application, runs on operating systems from Microsoft Windows XP to Microsoft Windows 10/11. The development environment includes a code editor, compiler and help system. The interface language of the Pascal Next development environment is English or Russian (for Russian localization of the operating system).

The Pascal Next programming language compiler, included in the Pascal Next development environment, is a Win32 application. The compiler creates a Win32 executable file. Compiler messages about errors in the program are in English/Russian.

The Pascal Next development environment runs on operating systems from Microsoft Windows XP to Microsoft Windows 10/11.

- The interface language of the Pascal Next development environment is English or Russian (for Russian localization of the operating system).
- Compiler error messages are in English/Russian.
- Built-in programming language reference.
- Easy installation process
- Minimum computer resource requirements
- Distribution volume (installer) – 2.09 MB

WWW.pascal-next.ru/en

Pascal Next can be downloaded for free from www.pascal-next.ru/en

You can also download code examples and documentation from the site.

Program structure

The Pascal Next program is a set of procedures and functions.

The main procedure, with instructions from which program execution begins, is designated by the `program` keyword. All other procedures are designated by the `procedure` keyword, functions - by the `function` keyword.

The simplest program is a single procedure `program` and in general looks like this:

```
program Name()
var
    // here are variable declarations
begin
    // here are the instructions to be executed
end.
```

Example

```
// convert weight from pounds to kilograms
program p1()
var
     fnt: float; // weight in pounds
     kg:float; // weight in kilograms
begin
     write('Weight in pounds >');
     readln(fnt);
     kg := fnt * 0.495; // 1 kg = 495 g

     writeln(fnt:6:2, ' lb. = ', kg:6:3, 'kg');

     writeln('Press <Enter>');
     readln;
end.
```

Before the `var` section there may be a `const` section (named constant declaration section) in which the programmer can place declarations of constants used in the program.

Example

```
// convert weight from pounds to kilograms
program p1()
const
     K = 0.495; // coefficient conversion from pounds to kg
var
     fnt: float; // weight in pounds
     kg:float; // weight in kilograms
begin
     write('Weight in pounds >');
     readln(fnt);
     kg := fnt * K;

     writeln(fnt:6:2, ' lb. = ', kg:6:3, 'kg');
```

```
    writeln('Press <Enter>');
    readln;
end.
```

Comments

A Pascal Next program can contain both multi-line and single-line comments. A multiline comment is text enclosed in curly braces. Single-line comment – text from the next two consecutive slash characters to the end of the line.

Example:

```
{ this is
  a multiline
  comment }

 { this is also a multi-line comment }

  // this is a one-line comment
```

Data types

Pascal Next supports integer, real and string data types.

integer – integers in range -2 147 483 648 ... 2 147 483 647

float - positive and negative real numbers ranging from 1.5x10⁻³⁸ to 3.4x10³⁸

string – character string up to 128 characters long

Variables

All program variables must be declared in the `var` section of the procedure or function in which they are used.

Numerical variable

The instruction for declaring a numeric variable of an integer or real type generally looks like this:

```
    name: type;
```

where:

name – variable name;

type – date type.

Example:

```
    sum: float;
    k: integer;
```

It is possible to declare several variables of the same type with one instruction.

For example:

```
a,b,c: float;
```

Strings variable

The instruction for declaring a string variable in general looks like this:

```
name: string[length];
```

Where:

length – the maximum number of characters that the variable can hold.

The maximum allowed value for the length parameter when declaring a string is 128.

Example:

```
name: string[25];
```

It is possible to declare several variables of the same type with one instruction.

For example:

```
firstName, lastName: string[12];
```

When declaring a string variable, you can use an entire named constant.

For example, if an entire named `LN` constant is declared in the `const` section, then the declaration of the `firstName` and `lastName` variables could be like this:

```
firstName, lastName: string[LN]; // LN is an integer named constant
```

Variable name

As a variable name, you can use any sequence of characters starting with a letter and consisting of letters and numbers. In addition to letters and numbers, a variable name can contain underscore characters.

Example:

```
amount: integer;
x1: float;
month_salary: float;
annual_income: float;
first_name: string[20];
```

The Pascal Next compiler does not distinguish between uppercase and lowercase letters, i.e. is **case-insensitive** when writing identifiers. Thus, for example, the identifiers `first_name`, `FIRST_NAME` and `First_Name` refer to the same object (variable).

Reserved words of the programming language, as well as names of built-in procedures and functions, cannot be used as names of variables (and other program objects).

Constants

Numeric constants are written in the usual way.

Example of integer constants:

```
123
```

```
-45
0
```

Example of real constants:

```
5.0
27542.15
25.7
-34.05
0.0
```

A string constant is a sequence of any characters enclosed in single quotes.

Examples of string constants:

```
'Hello, World!'
'Bart Simpson'
'(C) Nikita Kultin, 2021'
' '
''
'100'
'99.5'
```

Named constants

Named constants must be declared in the `const` section of the program, procedure, or function in which they are used.

The declaration of a named constant looks like this:

```
name = value;
```

Examples:

```
const
    Copyright ='(c) Nikita Kultin, 2021'; // named string constant
    PI = 3.1415925; // real named constant
    HB = 7; // integer named constant
    NL = 25; // integer named constant
```

Once declared, a named constant can be used in a program as an ordinary constant, including in the variable declaration section.

Examples of using named constants when declaring variables:

```
matrix array[1..HB,1..HB] of float; // HB is a named constant
students array[1..HB] of string[NL]; // HB, NL - named constants
name: string[NL]; // NL - named constant
```

Examples of using named constants in code:

```
sq := PI*r*r; // PI - named constant

for i:=1 to HB do // HB is a named constant
    for j:=1 to HB do
       matrix[i,j]:=0;
    end;
end;
```

Logical (boolean) data type

Pascal Next does not have a logical (boolean) data type, however, it can be easily simulated by defining in the program integer named constants `TRUE` and `FALSE` with the values 1 and 0, respectively. After this, instead of variables of the logical type, you can use variables of the integer type, treating them as logical.

Example

```
// pseudo-boolean type
program p()
const
     // "logical" constants
     TRUE = 1;
     FALSE = 0;

     HB = 10;
var
     a:array[1..HB] of integer; // array of numbers
     r:integer; // number to be found in the array
     found: integer; // flag that the number is in the array (found)
     i:integer;
begin
     for i:= 1 to HB do
         a[i] := Random(HB);
     end;

     write('Number list: ');
     for i:= 1 to HB-1 do
         a[i] := Random(HB);
         write(a[i]:3,',');
     end;

     write(a[HB]:3);

     r:= Random(HB);
     writeln('Search: ',r);

     found := FALSE; // let the number not be found
     i:= 1;
     repeat
         if a[i] = r then
             found := TRUE; // number found
         else
             i:=i+1;
         end;
     until( found = TRUE) OR (i > HB);

     writeln('i=',i);

     if found = TRUE then
         writeln('Found!');
```

```
    else
        writeln('Not found!');
    end;

    write('Press <Enter>');
    readln;
end.
```

Output to console window

Information is displayed on the screen (in the console window) using the `write` and `writeln` instructions.

In general, instructions for outputting information to the console window are written as follows:

```
write(output_list);
writeln(output_list);
```

where:

output_list – comma-separated variable names, string constants, functions and expressions.

Example:

```
write(sum);
write('Press <Enter>');
writeln('x1=', x1, ' x2=', x2);
writeln(pound,' pounds =', pound*0.453, ' kg');
writeln(Random(100));
```

Formatted output

In the output line after the name of the variable or expression, you can specify the format for displaying the value, separated by a colon.

For integer and string values, the format specifies the width of the output field - the number of positions on the screen that is reserved for displaying the value of the variable.

In general, formatted output of integer and string values is specified as follows:

```
expression:n
```

where:

expression – expression (in the simplest case, the name of a variable), the value of which should be displayed;

n – output field width (integer constant).

Formatted output of real values in general is specified as follows:

```
expression:n:m
```

where:

expression – expression (in the simplest case, the name of a variable), the value of which should be displayed;

n – output field width (integer constant).

m - number of digits of the fractional part (integer constant).

Example:

```
// variables x1 and x2 are of real type
writeln('x1=', x1:9:3, 'x2=', x2:9:3);

// variable name is string, salary is real
writeln(name:15, salary:12:3);

// expression pound*0.453 of real type
writeln(pound:5:2,' pounds =', pound*0.453:6:3, ' kg.');
```

Keyboard input

Keyboard input is provided by the readln instruction, which is generally written as follows:

```
readln(name);
```

where:

name – the name of the variable whose value must be obtained from the user while the program is running.

Example:

```
readln(weight);
readln(salary);
readln(FirstName);
```

ATTENTION! When entering real values, you must use a period as the decimal separator. If, when entering a real value, a comma is entered instead of a period, an error (exception) will not occur, but the fractional part will be discarded.

Assignment statement

The assignment statement looks like this:

```
name := expression;
```

where:

name – the name of the variable or array element;

expression – an expression whose value is assigned to a variable or array element.

An expression consists of operands and operators. Operands are objects on which an action is performed, operators are symbols denoting actions.

Constants, variables, array elements, and functions can be used as the operand of an expression.

Example:

```
k := 0;
```

```
    x:=x1;
    a[i] = 0;
    x:=x + dx;
    x:=x + 0.05;
    n := Round((x1-x2)/dx);
    a[i] := Random(6);
    sum := sum + b[i,j];
```

Arithmetic operators

Arithmetic operators:

Operator	Action	Operand type	Epression type
+	addition	integer, float	integer – if both operands are integers; float – if one of the operands is float
-	subtraction	integer, float	integer – if both operands are integers; float – if one of the operands is float
*	multiplication	integer, float	integer – if both operands are integers; float – if one of the operands is float
/	division	integer, float	float
DIV	quotient	integer, integer	integer
MOD	remainder	integer, integer	integer

The + operator applies to string operands. The result of applying the addition operator to string operands is the concatenation (union) of string operands.

Example:

```
    name := 'Bart' + ' ' + 'Simpson';
    name := FirstName + ' ' + LastName;
```

The above instructions assume that the variables name, FirstName, and LastName are of type string.

Operators precedence

The value of an expression is evaluated from left to right, and note that multiplication and division operators have higher precedence than addition and subtraction operators.

To specify the desired sequence for calculating the value of an expression, use parentheses.

Example:

```
r := 1/((r1 + r2)/(r1 * r2));
x1 := (-b + Sqrt(d))/(2*a);
```

If statement

The choice of action depending on the fulfillment of some condition is implemented using the `if` statement.

The instruction to select one of two possible action options is written as follows:

```
    if condition then
        // here are the instructions that must be executed,
        // if the condition is true
    else
        // here are the instructions that must be executed,
        // if the condition is NOT true (false)
    end;
```

Example:

```
    if t = 1 then
        r := r1+r2;
    else
        r := r1*r2/(r1+r2);
    end;
```

If, when a condition is met, some action must be performed, and if the condition is not met, this action must be skipped and go to the next program statement, then the `if` statement is written like this:

```
    if condition then
        // here are the instructions that will be executed,
        // if the condition is true
    end;
```

Example:

```
    if a[i] < a[i+1] then
        b:=a[i];
        a[i]:=a[i+1];
        a[i+1]:=b;
    end;
```

Multiple choice

Multiple choice (choosing one action from several possible ones) is done using nested `if` statements.

The instructions below show how you can implement choosing one action from four possible options.

```
if condition1 then
      // here are the instructions that will be executed,
      // if condition1 is true
else
    if condition2 then
        // here are the instructions that will be executed,
        // if condition1 is false and condition2 is true
```

```
    else
        if condition3 then
             // here are the instructions that will be executed,
             // if conditions condition1 and condition2 are false,
             // and condition3 is true
        else
            // here are the instructions that will be executed,
            // if none of the conditions is condition1,
            // condition2 or condition3 are NOT true
        end;
    end;
end;
```

Example:

```
    // n is material number
    if n = 1 then
        material :='Aluminium';
        density := 2.7;
    else
        if n = 2 then
            material :='Copper';
            density := 8.9;
        else
            if n = 3 then
                material :='Steel';
                density := 7.856;
            else
                material :='Plastic';
                density := 1.9;
            end;
        end;
    end;
```

Condition

A condition is a boolean expression that can take one of two values: `TRUE` or `FALSE`.

There are simple and complex conditions.

Simple condition

A simple condition in general form is written as follows:

```
op1 comparison_operator op2
```

where:

op1 and *op2* are the operands being compared, which can be constants, variables, functions or expressions.

Comparison operators:

Operator	Operator name
=	equals
>	more
>=	more or equal
<	less
<=	less or equal
!=	not equal

Example:

```
a[i+1] < a[i]
d != 0
pos(' ', st) = 1
name = 'simpson'
```

Complex condition

A complex condition in general form is written as follows:

```
condition1 logical_operator condition2
```

where:

condition1 and *condition2* are expressions of logical type, which can be simple or complex conditions.

Logical operators:

- AND
- OR
- NOT

Example:

```
x >= x1 AND x <= x2
NOT((x < x1) OR (x > x2))
sum >=1000 and sum <10000
name = 'Bart' OR name = 'Homer'
```

Loops

There three types loops in Pascal Next:

- For loop
- While loop
- Repeat loop

For loop

The `For` loop statement is generally written as follows:

```
for count := start to finish do
```

```
        // instructions that need to be executed several times
    end;
```

where:

count – cycle counter (integer type variable);

start and *finish* are expressions of integer type (in the simplest case, integer constants) that determine, respectively, the initial and final value of the loop counter.

Example:

```
    for i:=1 to 10 do
        writeln(i:2, ' Hello, World!');
    end;
```

```
    for i:=1 to n do
        writeln(i:2, ' Hello, World!');
    end;
```

While loop

The `While` loop instruction (loop with a precondition) is generally written as follows:

```
    while condition do
        // here are instructions that will be executed until
        // while the condition is true
    end;
```

where:

condition – a simple or complex condition for executing instructions located between the words do and end.

Example:

```
    i := 1;
    while i <= 10 do
        writeln(i:2, ' Hello, World!');
        i := i + 1;
    end;
```

Repeat loop

The `Repeat` loop instruction (loop with a postcondition) is generally written as follows:

```
    repeat
        // here are instructions that will be executed until
        // while the condition is false
    until condition;
```

where:

condition – a simple or complex condition for ending the loop (stopping the execution of instructions located between the words repeat and until).

Example:

```
i := 1;
repeat
    writeln(i:2, ' Hello, World!');
    i := i + 1;
until i > 10;
```

Goto instruction

The `goto` instruction (unconditional jump) is generally written as follows:

```
goto label
```

where:

label – identifier of the instruction to which you want to jump.

A label is any string that begins with a letter and consists of letters and numbers.

The label is written before the instruction to which the jump must be made, and is separated from this instruction by a colon.

The label must be declared in the label declaration section of the procedure or function in which it is used. The beginning of the label declaration section is marked with the keyword `label`.

The label declaration section precedes the constant declaration section or, if there is no const section, the variable declaration section.

Example:

```
// calculation of gcd - greatest common
// divisor of two positive integers
program p1()
  label // label declaration section
      m1,m2; // labels
  var
      a,b: integer; // numbers
      n:integer; // GCD
  begin
      a:=12;
      b:=18;
      writeln('a=',a:2,' b=',b:2);

 m1:  if a = b then
         n:=a;
         goto m2;
      end;
      if a > b then
        a:= a-b;
        goto m1;
      else
        b:= b-a;
        goto m1;
      end;
```

```
m2:  writeln('Greatest common divisor:', n);
     write('Press <Enter>');
     readln;
 end.
```

Arrays

One-dimensional array

The declaration of a one-dimensional array in general looks like this:

```
name: array[1..HB] of type;
```

where:

name – array name

HB – upper limit of the array index range (number of array elements) - integer or integer named constant

type – type of array elements (array type)

Attention! The maximum allowed number of elements of a one-dimensional array is 255

Example:

```
Salary: array[1..15] of float; // array of real numbers
nPacients: array[1..31] of integer; // array of integers
Students: array[1..25] of strings[15]; // array of strings
```

It is possible to declare several arrays of the same type and size with one instruction, for example:

```
gold, silver, bronze: array[1 ..10] of integer; // three arrays of integers
students_1, students_2: array[1 .. 30] of string[25]; // two arrays of strings
```

When declaring a one-dimensional array, you can use an integer named constant as the upper bound of the index range.

For example, if integer named constants `HB` and `NL` are declared in the const section, then the declaration of the students array could be like this:

```
Students: array[1..HB] of strings[NL]; // HB and NL are named integer constants
```

Named constants used in an array declaration statement can be conveniently used in array processing instructions, for example:

```
for i:=1 to HB do
   writeln(Students[i]);
end;
```

Attention! When working with a large number of arrays or with large-dimensional arrays, it should be taken into account that the total size of data (memory occupied by program variables, including arrays) and program code cannot exceed 64K.

Two-dimensional array

A declaration of a two-dimensional array in general looks like this:

```
    name: array[1..NR,1..NC] of type;
```

where:

name – array name;

NR – number of rows - the upper limit of the array row index range;

NC – number of columns – upper limit of the array column index range;

type – type of array elements (array type).

Attention! The maximum allowed number of rows and number of columns in a two-dimensional array is 255.

Example declarations of two-dimensional arrays:

```
    // two-dimensional array of integers (25 rows, 12 columns)
    Salary: array[1..25,1..12] of integer;

    // two-dimensional array of real numbers (5 rows, 8 columns)
    Matrix: array[1..5,1..8] of float;

    // two-dimensional array of strings (100 rows, 2 columns)
    dictionary: array[1..100,1..2] of string[20];
```

It is possible to declare several arrays of the same type and size with one instruction, for example:

```
    Matrix1, Matrix2: array[1..5,1..8] of float; // two two-dimensional arrays
```

When declaring a two-dimensional array, you can use integer named constants as the upper bounds of the index ranges.

For example, if integer named constants `NR` and `NC` are declared in the const section, then the matrix array declaration could be like this:

```
    Matrix: array[1..NR,1..NC] of float; // NR and NC are named integer constants
```

Named constants used in an array declaration statement can be conveniently used in array processing instructions, for example:

```
  for i:=1 to NR do
     for j:=1 to NC do
          matrix[i,j]:=0;
     end;
  end;
```

Attention! When working with a large number of arrays or with large-dimensional arrays, it should be taken into account that the total size of data (memory occupied by program variables, including arrays) and program code cannot exceed 64K.

Initializing an Array

At the beginning of the program, elements of numeric arrays have a zero value, elements of string arrays have the value "empty string".

If a program needs an array whose element values differ from the default values, it is necessary to initialize the array.

Initializing an array is assigning the required values to all elements of the array.

You can initialize an array by specifying an initialization list in the array declaration statement.

Initializing a one-dimensional array

The instruction for declaring and initializing a one-dimensional array looks like this:

```
    name: array[1 .. N] of type = list;
```

where:

name – array name;

N – number of array elements (integer or integer named constant);

type – array type (integer, float or string);

list – list of constants. The type of constants must correspond to the type of the array, and their number must correspond to the size of the array.

Example:

```
material: array[1..4] of string[10] = 'Aluminum','Cooper','Gold','Steel';
density: array[1..4] of float = 2.71, 8.94, 19.32, 7.86;
```

When initializing a float array, you can use integer constants in the initialization list.

When initializing a character array, if the length of a string constant in the list is greater than the length of the string specified in the array declaration instruction, the corresponding array element will be initialized with the "truncated" string.

For example, if the array declaration statement looks like this

```
material: array[1..4] of string[6] = 'Aluminum','Cooper','Gold','Steel';
```

then the material[1] element will be initialized to 'Alumin'.

Initializing a two-dimensional array

The instruction for declaring and initializing a two-dimensional array looks like this:

```
name: array[1..NR, 1..NC] of type = list;
```

where:

name – array name;

NR and *NC* – number of rows and columns of the array (integers or integer named constants);

type – array type (integer, float or string);

list – list of constants. The type of constants must correspond to the type of the array, and their number must correspond to the number of array elements. In the initialization list, the values for the first row of the array are first specified, then for the second, and so on.

Example:

```
phrase: array[1..4, 1..3] of string[12] =
                  'buongirno','ciao','grazie',
                  'good day','good by','thank you',
                  'Добрый день', 'До свидания', 'Спасибо',
                  'Buenos dias','adios', 'gracias';

matrix: array[1..3, 1..4] of float =
                    1.5, 2.5, 3.0, 0.0,
                    3.7, 2.0, 6.2, 1.7,
                    0.0, 0.0, 0.0, 0.0;
```

When initializing a float array, you can use integer constants in the initialization list.

When initializing a character array, if the length of the string constant is greater than the length of the string specified in the array declaration instruction, then the corresponding array element will be initialized with a "truncated" string.

User defined function

A declaration of user defined function looks like this:

```
function Name (parameters): type
var
      // here are local variable declarations
begin
     // here are the instructions
    return value;
end;
```

where:

Name – function name;

parameters – declaration of function parameters;

type – type of function value;

value – the value of the function.

The declaration of each function parameter looks like this:

```
parameter_name: type
```

Example:

```
// The function CylinderVolume  calculates the volume of the cylinder
```

```
Function CylinderVolume(d: integer, len: integer): float
const
    PI=3.1415926;
var
    v: float;
begin
    v := PI*(d/2)*(d/2);
    return v;
end;
```

The parameters that are specified in the function call instruction are called actual. Parameters are passed to the function by value. The actual parameter can be an expression whose type matches the type of the formal parameter. In the simplest case, a constant or a variable can be used as the actual parameter. If the formal parameter of a function is of real type, then an expression of either real or integer type can be used as the actual parameter.

Examples of function calls:

```
volume := CylinderVolume(20, 550); // parameters are integer constants
volume := CylnderVolume(D1, L1); // parameters - variables
```

In order for a function to become available to another function or procedure, including the main procedure of a program, its declaration (text) must be placed in the program text before the procedure or function that uses it.

Example

```
// The user defined function CylinderVolume - volume of the cylinder.
Function CylinderVolume(d: integer, len: integer):float
const
    PI = 3.1415926;
begin
    return PI*(d/2)*(d/2)*len;
end;

// cylinder volume calculation program
Program P1()
var
    diam:integer;  // diameter
    len:integer;   // length
    volume: float; // volume
begin
    writeln('Cylinder volume');
    write('Diameter, mm >');
    readln(diam);
    write('Length, mm >');
    readln(len);

    volume := CylinderVolume(diam,len // volume in mm cubic.
    volume := volume / 1000; // volume in cm cubic.

    writeln('Cylinder volume', volume:9:2, ' cm.cub.');

    writeln;
```

```
        write('Press <Enter>');
        readln;
    end.
```

User defined procedure

A declaration of user defined procedure looks like this:

```
    procedure Name(parameters)
    var
        // here are local variable declarations
    begin
        // here are the instructions
    end;
```

Where:

Name – procedure name;

parameters – declaration of procedure parameters.

The declaration of each parameter looks like this:

```
    parameter_name: type
```

Example

```
// The Line procedure displays the string st in the console window n times
procedure Line(n:integer, st: string)
var
     i:integer;
begin
     for i:=1 to n do
         write(st);
     end;
     writeln;
end;
```

Procedure call example

```
     Line(k,'-');
     Line(25, ch);
```

The parameters that are specified in the procedure call statement are called actual parameters. Parameters are passed to the procedure by value. The actual parameter can be an expression whose type matches the type of the formal parameter. In the simplest case, a constant or a variable can be used as the actual parameter. If the formal parameter of a function is of real type, then an expression of either real or integer type can be used as the actual parameter.

In order for a procedure to become available to another function or procedure, including the main procedure of a program, its declaration (text) must be placed in the program text before the procedure or function that uses it.

Example - Table of trigonometric functions sin and cos

```
// The Line procedure draws a line
procedure Line(n:integer, ch: string)
var
  i: integer;
begin
  for i:=1 to n do
     write(ch);
  end;
  writeln;
end;

// Displays a table of values of the sin and cos functions
program main()
var
   g1,g2: float; // range of angle changes in degrees
   dg:float;     // step of changing the angle in degrees
   g:float;      // current angle value in degrees
   r:float;      // angle in radians
   k:integer;    // width (in characters) of the table on the screen
begin
  k:= 43;
  writeln;
  writeln(' Sin and Cos');
  writeln;
  Line(k,'_');
  writeln(' deg':7, '      rad':12, '     Sin':12,  '       Cos':12);
  Line(k,'-');
  g1 := 0;
  g2 := 360;
  dg := 15;
  g :=g 1;
  while g <= g2 do
    r := 3.14/180*g;
    writeln(g:7:2, r:12:6, sin(r):12:6, cos(r):12:6 );
    g:= g + dg;
  end;

  Line(k,'=');
  writeln;

  write('Press <Enter>');
  readln;
end.
```

Recursion

Pascal Next supports recursive **functions**.

Example

```
    // the Factorial function calculates the factorial of n
    function Factorial(n: integer): integer
    var
        f: integer;
    begin
        if n = 1 then
            f:=1;
        else
            f:= n*Factorial(n-1);
        end;

        return f;
    end;

    // factorial values of numbers from 1 to 12
    program p28()
    var
        i: integer;
    begin
        for i:=1 to 12 do
           writeln(i:2, ' - ', Factorial(i));
      end;

      write('Press <Enter>');
      readln;
    end.
```

Global Variables

Global or general are variables and data structures (arrays) that are declared outside a procedure or function, but to which the procedure or function has access.

Global variables are typically used to provide procedures and functions with access to common data.

In order for a procedure or function to gain access to program variables, its declaration must be placed in the program declaration, after the variable declaration section.

Attention! You can only nest procedures and functions in a program (in the program procedure). You cannot place another procedure or function inside a procedure or function.

Example:

```
program p29()
  const
    HB = 10;
  var
    c: array[1..HB] of integer;
    sum: integer;
```

```
  i: integer;

function SumArr(m: integer, n: integer): integer
  var
    i: integer;
    sum: integer;
  begin
    sum := 0; // local variable
    for i := m to n do
      sum := sum + c[i];
    end;
    return sum;
  end;

 // Initializing an array with random numbers
 procedure InitArr()
   var
     i: integer;
   begin
     for i := 1 to HB do
       c[i] := Random(10);
     end;
  end;

// main program
begin
  InitArr(); // array initialization
  sum := SumArr(1,HB);

  for i:=1 to HB do
    write(c[i]:4);
  end;
  writeln;

  writeln('sum=',sum);

  write('Press <Enter>');
  readln;
end.
```

File operations

There are three main file operations:

- Reading lines from a text file
- Writing strings to a text file
- Adding lines to a text file

When reading strings from a text file, you should use the `StrToInt` or `StrToFloat` functions to convert a string to an integer or real value.

When writing numeric value to a text file, you should use the `IntToStr` or `FloatToStr` functions to convert a numeric value to a string.

Access to a text file is provided by a file descriptor - an object of type Text.

File functions are described in the following table:

Function	Description	Example
reset(*file_name*)	Opens a text file for reading. Returns a file descriptor or -1 if the filename is incorrect (there is no file in the specified folder, the filename is incorrect).	`DataFile:='C:\Temp\dat.txt';` `f:=reset(DataFile);`
readstring(*descriptor*)	Reads a line from a text file. To convert a string to a number you should use the StrToInt or StrToFloat function.	`name:=readstring(f);`

eof(*descriptor*)	End of file. Returns -1 if end of file is reached.	`while eof(f) != -1 do`
		`  // action`
		`end;`

rewrite(*file_name*)	Opens a text file for overwriting. Returns a file descriptor (if the file does not exist, it will be created).	`DataFile:='C:\Temp\dat.txt';` `f:=rewrite(DataFile);`
append(*file_name*)	Opens a text file for appending. Returns a file handle, or -1 if the specified file does not exist.	`DataFile:='C:\Temp\dat.txt';` `f:=append(DataFile);`

writestring(*descriptor*, *string*);	Writes a string of characters to a text file. To convert a numeric value to a string, use the IntToStr or FloatToStr function.	`n:=100; k:=32.5;`
		`writestring(f,IntToStr(n));`
		`writestring(f,FloatToStr(k));`
		`writestring(f, name);`

close(*descriptor*)	Closes an open file.	`close(f);`

The file name can be full (including the path to the file) or short. The short file name assumes that the data file is located in the same directory as the program's executable file.

When running a program from the development environment, you must specify the full file name.

The path to the file located in the Documents folder should be written like this:

```
c:\users\user_name\documents
```

where:

user_name – user name in the operating system (Windows Username).

For example, if the username is `nikita`, then the full name of the data.txt file located in the Documents folder looks like this:

```
c:\users\nikita\documents\data.txt
```

If the file is on the desktop, then its name should be written like this:

```
c:\users\user_name\desktop\fille
```

For example, if the username is `nikita`, then the full name of the data.txt file located on the desktop looks like this:

```
c:\users\nikita\desktop\data.txt
```

Mathematical functions

Mathematical functions:

Abs(x)	absolute value of x
Arctg(x)	arctangent, x – tangent
Cos(r)	cosine, r – angle in radians
Random(n)	random number in the range 1..n
Round(x)	rounding
Sqrt(x)	square root
Sin(r)	sine, r – angle in radians
Tg(r)	tangent, r – angle in radians
Trunc(x)	integer part of real

String functions

Length

Length(string) – length of the string

The `Length` function returns the number of characters of the string specified as a parameter.

Example:

```
name:= 'Bart Simpson';
k := Length(name);
```

Pos

Pos(substring, string) – position of the substring in the string

The `Pos` function returns the position of the first occurrence of a substring in the specified string. If the string does not contain the specified substring, the function returns zero.

Example:

```
    name := 'Bart Simpson';
    p := Pos('Simpson', name);
```

Subsrt

Substr(string, start, len) – substring

The `Substr` function returns a substring of the specified string. The `start` parameter specifies the position of the first character of the required substring (the beginning of the substring), the `len` parameter specifies the number of characters of the substring. If the length of the string is less than the value of the `start` parameter, then the function returns an empty string. If the length of the string is such that it is impossible to obtain a substring of the specified the `len` parameter, then the function returns the right side of the string, starting from the character with the `start` parameter.

Example of using the string functions Length, Pos and Substr

```
program p1()
var
    name: string[15];

    lastName: string[15];
    firstName: string[15];

    p: integer;   // position of space in line
begin
    name := 'Bart Simpson';
    writeln('Name: ', name);
    p:= Pos(' ', name);

    if p !=0 then
        firstName := Substr(name, 1, p-1);
        lastName:= Substr(name, p+1, Length(name)-p);
    else
        firstName := name;
        lastName:='';
    end;
    writeln('First Name: ', firstName, ' Last name: ', lastName);
    readln;
end.
```

UpCase

UpCase(string) – string converted to upper case. The `UpCase` function returns a string converted to uppercase.

Example:

```
    name := UpCase(name);
```

LowCase

LowCase(string) – string converted to lowercase

The `LowCase` function returns a string converted to lowercase.

Example:

```
    name := LowCase(name);
```

Convertsion functions

IntToStr

The `IntToStr` function returns a string representation of an expression of the integer type specified as its parameter.

StrToInt

The `StrToInt(st)` function converts an image string of an integer into a number corresponding to the parameter string. If the parameter string is not a valid representation of an integer (contains characters other than digits), then the function returns 0.

FloatToStr

The `FloatToStr(x)` function returns the string representation of the real-type expression specified as its parameter.

StrToFloat

The `StrToFloat(st)` function converts the image string of a real number into the number corresponding to the parameter string. If the parameter string is not a valid representation of a real number (contains characters other than digits or a decimal separator), then the function returns 0.

Date and time functions

getDay

The `getDay()` function returns the serial number of the day of the current month.

getMonth

The `getMonth()` function returns the serial number of the month of the year.

The first month of the year is January, numbered starting from one.

Example

```
program p1()
var
     day: integer;
     month: integer;
```

```
     monthName: array[1..12] of string[10] =
            'January', 'February', 'March', 'April', 'May', 'June',
            'July','August','September','October','November','December';
begin
    day := getDay();
    month := getMonth();

    writeln('Today is ', day, ' ', monthName[month] );

    write('Press <Enter>');
    readln;
end.
```

getYear

The `getYear()` function returns the year number.

Example

```
program p1()
var
    day: integer;
    month: integer;
    year: integer;

begin
     day := getDay();
     month := getMonth();
     year := getYear();

     writeln('Today is ', day, '-', month, '-', year);

    write('Press <Enter>');
    readln;
end.
```

getDayOfWeek

The `getDayOfWeek()` function returns the ordinal number of the day of the week. The first day of the week is sunday. Days of the week are numbered from zero.

Example

```
program p1()
var
    dayOfweek: integer;

    weekDay: array[1..7] of string[11] =
           'Sunday', 'Monday', 'Tuesday', 'Wednesday',
           'Thursday', Friday', 'Saturday';
begin
    dayOfweek:= getDayOfWeek();
    writeln('Today is ', weekDay[dayOfWeek + 1]);
```

```
    write('Press <Enter>');
    readln;
end.
```

getTime

The `getTime()` function returns the number of seconds since the beginning of the current day.

Example

```
program p1()
var
   time: integer;

        hour: integer;
   min: integer;
   sec: integer;

begin
    time := getTime();
    Writeln('Seconds from the beginning of the day: ', time);

    hour:= time div 60 div 60;
    min:=  (time - hour*3600)  div 60;
    sec:=  time- hour*3600 - min*60;

    writeln('It is ', hour, ':', min, ':', sec, ' now');

    writeln('Press <Enter>');
    readln;
end.
```

Reserved words

Here's a list of reserved words of the Pascal Net:

and	array	begin
const	div	do
else	end	float
for	function	goto
if	integer	label
mod	not	of
or	procedure	program
repeat	return	string
then	to	until

var while

Pascal and Pascal Next

The main differences between Pascal Next syntax and "classic" Pascal described in the following table:

	Pascal	**Pascal Next**
Comment	multiline comment	multiline comment one line comment
Comparison operator "not equal"	`<>`	`!=`

If statement	`if condition then`
	`  begin`
	`  // action_1`
	`  end`
	`else`
	`  begin`
	`  action_2`
	`  end;`
		`if condition then`
		`  // action_1`
		`else`
		`  // action_2`
		`end;`

For loop	`for i:=start to fin do`
	`begin`
	`  // action`
	`end;`
		`for i:=start to fin do`
		`  // action`
		`end;`

While loop	`while condition do`
	`begin`
	`  // action`
	`end;`
		`while condition do`
		`  // action`
		`end;`

Repeat loop	`repeat`
	`  // action`
	`until condition;`
		`repeat`
		`  // action`
		`until condition;`

Ability to declare a user type	yes	no
Ability to initialize an array in its declaration instructions	no	yes

Properties of the .exe file

When you compile a program for the first time, the Pascal Next compiler creates a text file (RS) and places in it a template of information about the program (program name, program version, copyright, etc.). During the compilation process, information from the RS file is placed in the `rsrc` section of the EXE file. This gives the user the opportunity to obtain information about the program (the contents of the EXE file), see the name of the program, who the developer is, who owns the copyright, and the version number.

Information about the contents of the EXE file is displayed in the **Properties** window, which appears on the screen by right-clicking on the file name and selecting the appropriate command from the context menu.

Example of the RS file generated by the compiler:

```
Company=
Description=
Version=1.0.0.3
Copyright=(C) , 2023
ProductName=
ProductVersion=1.0.0.1
```

The programmer can change the contents of the RC file so that the information placed in the EXE file matches the program being developed.

An example of an RS file after making changes:

```
Company=MyCompany
Description=Pound to gram converter
Version=1.0.0.1
Copyright=(C) Nikita Kultin , 2023
ProductName=Weight converter
ProductVersion=1.0.0.1
```

Code examples

This chapter contains sample Pascal Next programs that demonstrate the syntax and capabilities of the programming language.

Hollow cylinder volume calculator

```
// Hollow cylinder volume calculator
Program P1()
const
   PI = 3.1415926;
var
    diam:integer;  // diameter
    wal:integer;   // wall thickness
    len:integer;   // length
    volume: float; // volume
begin
    writeln(' Hollow cylinder volume calculator');
    write('Diameter, mm >');
    readln(diam);
    write('Wall thickness, mm >');
    readln(wal);
    write('Length, mm >');
    readln(len);

    volume := PI*diam*diam/4*len - PI*(diam -2*wal)*(diam -2*wal)/4*len;
    volume := volume / 1000; // volume in cm. cubed

    writeln('Hollow cylinder volume', volume:9:2, ' cm cubed');

    writeln;
    write('Press <Enter>');
    readln;
end.
```

Weight converter from pounds to grams/kilograms

```
// Weight converter from pounds to grams/kilograms
program p1()
const
   K = 453.59237;
var
    Pounds: float; // weight in pounds
    Grams: integer; // weight in grams
    Kilograms: float; // weight in kilograms

    // weight as kg + g
```

```
   KG: integer;
   GR: integer;

begin

   writeln('Pounds to grams/kilograms converter');
   writeln;

   write('Pounds>');
   readln(Pounds);

   Grams := Round(Pounds * K);

   if grams < 1000 then
       writeln(Pounds:6:2, ' lb = ', Grams, ' g');
   else
       Kilograms := Grams / 1000;
       KG := Grams DIV 1000;
       GR := Grams mod 1000;

       write(Pounds:6:2, ' lb = ', Kilograms:6:3);
       writeln(' kg = ',KG,' kg ', GR:3, ' g');
   end;

   writeln;
   write('Press <Enter>');
   readln;
end.
```

Current in a circuit consisting of two resistors

```
// Current in a circuit consisting of two resistors, which can be connected in series or
in parallel
program p()
var
      R1,R2: float; // resistance values, Ohm
      T: integer;   // connection type: 1 - serial; 2 - parallel
      U: float;     // voltage

      R: float; // circuit resistance
      I: float; // current in the circuit
begin
    writeln('Current in a circuit consisting of two resistors');
    writeln;
    write('R1, Ohm >');
    readln(R1);
    write('R2, Ohm >');
    readln(R2);
    write('Connection type (1 - serial; 2 – parallel) >');
    readln(T);
```

```
    write('U, Volt >');
    readln(U);

    if T = 1 OR T = 2 then
         if T = 1 then
             R := R1 + R2;
         else
             R := R1*R2/(R1+R2);
         end;

         writeln('Circuit resistance: ',R:6:2, ' Ohm');

         I := U/R;

         write('I = ');
         if I < 0.1 then
             I := I * 1000;
             writeln(Round(I),' mA');
         else
             writeln(I:6:3,' A');
         end;
      else
         writeln('Error! The connection type is incorrect.');
      end;

    write('Press <Enter>');
    readln;
end.
```

Mass of a hollow rod

```
// mass of a hollow rod (pipe)
Program P1()
const
    PI = 3.1415926;
var
    diam:integer; // diameter
    wal:integer;  // wall thickness
    len:integer;  // length
    n:integer;    // material number

    material:string[15]; //material
    density: float;      //material density, gr/cm3

    volume: float; // volume
    mas: float;    // mass, gr
begin
    writeln('Mass of a hollow rod (pipe)');
    write('Diameter, mm >');
    readln(diam);
```

```
    write('Wall thickness, mm>');
    readln(wal);
    write('Length, mm >');
    readln(len);

    writeln('Select material');
    writeln('1. Aluminum');
    writeln('2. Copper');
    writeln('3. Steel');
    writeln('4. Plastic');
    write('>');
    readln(n);

    if ( n < 1 OR n > 4 ) then
        writeln('Error! The material number is incorrect.');
    else
        if n = 1 then
            material :='Aluminium';
            density := 2.7;
        else
            if n = 2 then
                material :='Copper';
                density := 8.9;
            else
                if n = 3 then
                    material :='Steel';
                    density := 7.856;
                else
                    material :='Plastic';
                    density := 1.9;
                end;
            end;
         end;

        writeln('');
        // volume in mm3
        volume := PI*diam*diam/4*len –
                  PI*(diam -2*wal)*(diam -2*wal)/4*len;

        // volume in cm cubic.
        volume := volume / 1000;

        mas := volume * density;

        writeln('Material: ', material, '(',density:6:3,'gr/cm3)');
        writeln('Volume:', volume:9:2, ' cm3');
        writeln('Mass:', mas:6:2);
    end;

    writeln;
    write('Press <Enter>');
```

```
    readln;
end.
```

Table of values of the functions sin and cos

```
// Table of trigonometric functions sin and cos.
// Demonstrates the use of the sin and cos functions,
// use of procedures, formatted output.

// Draws a line
procedure line(n:integer, ch: string)
var
    i: integer;
begin
    for i:=1 to n do
           write(ch);
    end;
    writeln;
end;

// Displays a table of syn and cos values
// p1 - start angle, p2 – end angel, p3 – step
procedure tabsin(p1: float, p2: float, p3: float)
var
    g:float; // angle in degrees
    r:float; // angle in radians
    k:integer; // line length (parameter f-i line)
begin
    k:= 43;
    Line(k,'_');
    writeln(' Deg':7, '     Rad':12, '     Sin':12,  '       Cos':12);
    Line(k,'-');
    g := p1;
    while g <= p2 do
        r := 3.14/180*g;
        writeln(g:7:2, r:12:6, sin(r):12:6, cos(r):12:6 );
        g:= g + p3;
    end;

    Line(k,'=');
    writeln;
end;

program main()
begin

    TabSin(0.0, 360.0, 15.0); // from 0 to 360 by step 15

    write('Press <Enter>');
    readln;
```

```
end.
```

Maximum element of an array

```
// Maximum element of an array
program p5()
const
    ARS = 5; // array size
var
    a: array[1 .. ARS] of integer;
    max: integer;

    i,;
begin
    a[1] := 8;
    a[2] :=-8;
    a[3] := -17;
    a[4] := 25;
    a[5] := 6;

    write('Source array: ');
    for i := 1 to ARS do
       write(a[i]:4);
    end;
    writeln;

    max:=1; // let the first element be maximum

    for i := 2 to ARS  do
       if (a[i] > a[max]) then
            max:=i;
       end;
    end;

    writeln('Max element: ', a[max], 'Inex of element:', max);

    write('Press <Enter>');
    readln;
end.
```

Sorting an array using the exchange method (Bubble sort)

```
// Sorting an array using the exchange method
program p5()
const
    ARS = 5; // array size
var
    a: array[1 .. ARS] of integer;
    b: integer;
```

```
    i,j: integer;
begin
    a[1] := 8;
    a[2] :=-8;
    a[3] := -17;
    a[4] := 25;
    a[5] := 6;

    write('Source array: ');
    for i := 1 to ARS do
       write(a[i]:4);
    end;
    writeln;

    // sorting
    for i:= 1 to ARS do
        for j:=1 to ARS do
            if (a[j+1] < a[j]) then
                // exchange
                b:= a[j];
                a[j] := a[j+1];
                a[j+1] := b;
            end;
        end;
    end;

    // display sorted array
    write('Sorted array: ');
    for i:=1 to ARS do
        write(a[i]:4);
    end;
    writeln;

    write('Press <Enter>');
    readln;
end.
```

Number to words converter

```
// Generates a value in Word for an integer in the range from 1 to 999
function Prop(n: integer): string
var
    st: string[64]; // an integer in words
    d1: array[1..19] of string[10]=
          'one','two','three','four','five','six','seven',
          'eight','nine','ten','eleven','twelve','thirteen',
          'fourteen','fifteen','sixteen','seventeen','eighteen',
          'nineteen';
```

```
    d2: array[1..9] of string[10] =
          '','twenty','thirty', 'forty', 'fifty', 'sixty',
          'seventy', 'eighty', 'ninety';

    // hundreds
    d3: array[1..9] of string[10] =
           'one','two','three','four','five',
           'six','seven','eight','nine';

    hund:integer; // number of hundreds
    tens:integer; // number of tens if the number is greater than 19
    un:integer;   // number of units

begin
    st:='';

    hund := n div 100;
    if hund != 0 then
        st := d3[hund] + ' hundreds ';
        n := n - hund *100;
    end;

    if N !=0 then
        if (n <=19) then
            st := st + d1[n];
        else
            tens := n div 10;
            st:= st + d2[tens];
            un := n - tens *10;
            if (un !=0 ) then
                st := st + ' ' + d1[un];
            end;
        end;
    end;
    return st;
end;


// converts the first letter of a string to uppercase
function Capital(st: string):string
var
    res: string[128];
begin
    res := UpCase(substr(st,1,1)) + LowCase(substr(st,2,length(st)-1));
    return res;
end;


Program p()
var
    n: integer;
```

```
    st: string[128]; // an integer in words

    i:integer;

begin
    writeln('Type number from 1 to 999 and press <Enter>');
    writeln('To stop type 0 or press <Enter>');
    repeat
        writeln;
        write('>');
        readln(n);
        if (n >= 0) AND (n < 1000) then
            st := Capital(Prop(n));
            writeln(st);
        end;
    until(n = 0);

    writeln;

    for i:=1 to 25 do
        n:= Random(999);
        writeln(n:3, ' - ', Capital(Prop(n)));
    end;

    write('Press <Enter>');
    readln;
end.
```

Sorting a two-dimensional array

```
// Sorting a two-dimensional array
program TwoDimArrSorting ()
const
    NR=6;
    NC=16;
var
    a: array[1..NR, 1..NC] of integer;
    i,j: integer;
    key: integer; // key column index
    m: integer;   // min rows element index
    c: integer;   // row index in
    b: integer;

begin
    // random array for sorting
    for i:= 1 to NR do
        for j:=1 to NC do
            a[i,j,Random(100);
        end;
    end;
```

```
    writeln('Source array:');
    for i:= 1 to NR do
        for j:=1 to NC do
            write(a[i,j]: 4);
        end;
        writeln;
    end;

    key:=1; // key column

    for i:=1 to NR do // repeat as many times as there are rows in the array
        // find the minimum element in the key column of the array
        // from the j-th element
        m:=i;
        for j:=i+1 to NR do
            if a[j,key] < a[m,key] then
                m:=j;
            end;
        end;

        if m != i then
            // exchange the i-th and m-th rows of the array
            for c:=1 to NC do
                b:=a[i,c];
                a[i,c]:=a[m,c];
                a[m,c]:=b;
            end;
        end;
    end;

    writeln;
    writeln('Key column: ', key);

    writeln('Sorted array:');
    for i:= 1 to NR do
        for j:=1 to NC do
            write(a[i,j]: 4);
        end;
        writeln;
    end;

    writeln;
    write('Press <Enter>');
    readln;
end.
```

User function CylinderVolume

```
// User function the Volume – volume of a cylinder
```

```
Function CylinderVolume(d: integer, len: integer):float
const
    PI = 3.1415926;
begin
    return PI*(d/2)*(d/2)*len;
end;



// Volume of a hollow cylinder
Program P1()
var
    diam:integer; // diameter
    wal:integer; // wall thickness
    len:integer; // length
    volume: float; // volume
begin
   writeln('Volume of a hollow cylinder');
   write('Diameter, mm >');
    readln(diam);
    write('Wall thickness, mm>');
    readln(wal);
    write('Length, mm >');
    readln(len);

    // volume in mm3
    volume := CylinderVolume(diam,len) - CylinderVolume(diam-2*wal,len);
    volume := volume / 1000; // volume in cm3

    writeln('Volume of a hollow cylinder', volume:9:2, 'cm3');
    writeln;
    write('Press <Enter>');
    readln;
end.
```

User-defined procedure and function

```
// User-defined procedure and function

// displays a greeting in Russian, Italian, Spanish or English
procedure hi(lang: string)
begin
    if lang = 'ru' then
        writeln('Привет!');
    else
        if lang = 'it' then
            writeln('Ciao!');
        else
            if lang = 'es' then
                writeln('Ola!');
```

```
            else
                writeln('Hi!');
            end;
        end;
    end;
end;


// returns a greeting string in Russian, Italian. Spanish or English
function GoodDay(lang: string): string
var
    msg: string[15];
begin
    if lowcase(lang) = 'ru' then
        msg:='Добрый день!';
    else
        if lowcase(lang) = 'it' then
            msg:='Buongiorno!';
        else
            if lowcase(lang) = 'es' then
                msg:='Buones dias!';
            else
                msg:='Good day!';
            end;
        end;
    end;
    return msg;
end;


program p()
var
    LangID: string[10]; // language identifier:
                        // ru - Russian
                        // en - English
                        // it - Italian
                        // es – Spanish
begin
    repeat
        write('Language ID (ru, en, it, es)>');
        readln(LangID);
        if length(LangID) !=0 then
            hi(lowcase(LangID));
            writeln(GoodDay(LangID));
            else
              hi(lowcase('ru'));
            writeln(GoodDay('ru'));
        end;
    until length(LangID) = 0;

    write('Press <Enter>');
```

```
    readln;
end.
```

Recursive function Factorial and table of factorials

```
// Recursive function Factorial and table of factorials from 1 to 12

// Function Factorial
function fac(n: integer): integer
var
    f: integer;
begin
    if n = 1 then
          f:=1;
    else
          f:= n*fac(n-1);
    end;

    return f;
end;


// Table of factorials from 1 to 12
program p28()
var
        i: integer;
begin
        for i:=1 to 12 do
             writeln(i:2, ' - ', fac(i));
        end;

        write('Press <Enter>');
        readln;
end.
```

Recursion. Finding routes between two points in a graph

```
// Recursion. Finding routes between two points in a graph
program findRoad()

var
    n:integer; //number of graph vertices

    // map (graph): map[i,j] not 0 if points i and j are connected
    map:array[1..7,1..7] of integer;

    road:array[1..7] of integer; // road - map point numbers

    incl:array[1..7] of integer; // incl[i]=1, if point
```

```
                                    // number i is included in road

    start: integer; // starting point (from)
    finish:integer; // end point (to)

    i,j: integer;

    r: integer;

function step(s: integer,f: integer,p:integer): integer
    // s - the point from which the step is taken
    // f - the point where you need to get (final)
    // p - number of the required route point
var
    c:integer;// Number of the point at which the next step is taken
    r: integer;
begin
    if s = f then  // The current point is where you need to go
       // Display found route
        write('Route: ');
        for i:=1 to p-1 do
            write(road[i],' ');
        end;
        writeln;

    else
        // Selecting the next point
        for c:=1 to N do // Check all vertices
            if(map[s,c] !=0) and (incl[c]=0)
                // The point C is connected to the S poin,
                // but yet not included to the route              then
                road[p]:=c; // Add a point to the route
                incl[c]:=1; // and mark it as included

                r:=step(c,f,p+1); // find next point

                incl[c]:=0;
                road[p]:=0;
            end;
        end;
    end;
end; // end of the Step function


// main program
begin
    N:=7;

    for i:=1 to N do
       for j:=1 to N do
           map[i,j]:=0;
```

```
    end;
end;

// The Map. map[i,j] = 1 if point i is connected to point j
map[1,2]:=1;
map[1,3]:=1;
map[1,4]:=1;

map[2,1]:=1;

map[3,1]:=1;
map[3,4]:=1;
map[3,7]:=1;

map[4,1]:=1;
map[4,3]:=1;
map[4,6]:=1;

map[5,6]:=1;
map[5,7]:=1;

map[6,4]:=1;
map[6,5]:=1;
map[6,7]:=1;

map[7,3]:=1;
map[7,5]:=1;
map[7,6]:=1;

// Display the map
for i:=1 to N do
    for j:=1 to N do
       write(map[i,j]:3);
    end;
    writeln;
end;

repeat
    for i:=1 to N do
        road[i]:=0;  // no route
        incl[i]:=0;  // no points included
    end;

    writeln('Search for route');
    write('Starting point ->');
    readln(start);

    if start != 0 then
        write('End point ->');
        readln(finish);
        writeln;
```

```
            road[1]:= start; // add the start point to the route
           incl[start]:=1;   // and mark it as included in the route
           r:=step(start,finish,2); // find the second waypoint
        end;
    until start = 0;

    writeln;
    write('Press <Enter>');
    readln;
end.
```

String processing. Trim and Capital functions

```
// String processing. Trim and Capital functions

// Converts the first letter of a string to uppercase
function Capital(st: string):string
var
    res: string[128];
begin
    res := UpCase(substr(st,1,1)) + LowCase(substr(st,2,length(st)-1));
    return res;
end;


// removes leading and trailing spaces
function Trim(st: string):string
var
     trs: string[64]; // string without leading or trailing spaces
     p:integer;       // pointer to space at the beginning of the line
     lch: string[1];  // last character of the line
begin
   trs:=st;

    // remove leading spaces
    p:= pos(' ',trs);
    while p = 1 do
      trs:=substr(trs,2,length(trs)-1);
       p:= pos(' ',trs);
    end;

    // remove trailing spaces
    lch := substr(trs,length(trs),1);
    while lch = ' ' do
       trs:=substr(trs,1,length(trs)-1);
       lch := substr(trs,length(trs),1);
    end;

    return trs;
```

```
end;


program p1()
var
    name: string[25];

    lastName: string[15];
    firstName: string[15];

    p: integer; //space position between first name and last name

begin
    repeat
        writeln;
        write('name>');
        readln(name);

        if (Length(name) != 0) then
            name := Trim(name); // remove leading and trailing spaces
            p:= Pos(' ', name);
            if p !=0 then
              firstName := Capital(LowCase(Substr(name, 1, p-1)));
              lastName:=
                 Capital(LowCase(Trim(Substr(name, p+1, length(name)-p))));
            else
              firstName := Capital(LowCase(name));
              lastName:='';
            end;

            writeln('Name:', name);
            writeln('First Name:', firstName, ': Last name:', lastName, ':');
            name := firstname + ' ' + lastname;
            writeln('Name:', name, ':');
        end;
    until (length(name) = 0);

    write('Press <Enter>');
    readln;
end.
```

Cryptographer

```
// Cryptographer. Encrypts/decrypts text

program crypto()
var
    alf: string[64]; // alphabet

    src: string[128]; // original text
```

```
    dst: string[128]; // ciphertext
    rst: string[128]; // decoded text

    key: string[32]; // key
    n:integer; // number of the key letter used for
               // encoding/decoding the current message character
               // "character code" is the serial number
               // of the character in the alphabet alf
     pk:integer; // key symbol code
     ps:integer; // character code of the original message
     pd:integer; // character code that replaces the message character

    i: integer;

begin
    alf:= 'abcdefghijklnmopqrstuwvxyz0123456789 .,!?$';

    // the codeword must consist of alphabetic characters
    key := 'bartsimpson';

    src := 'Hello, James Bond!  Die Another Day... $100.00';

    src := LowCase(src);


   // encrypt
    n:=1;
    for i:=1 to length(src) do

        ps:= pos(substr(src,i,1),alf);
        if ( ps !=0) then // is the symbol in the alphabet?
            // encrypt
            pk:= pos(substr(key,n,1),alf);

            ps:= ps+pk;
            if  ps > length(alf)  then
                ps:= ps-length(alf);
            end;

            dst:=dst+substr(alf,ps,1);
        else
            // not encrypt
            dst:=dst+ substr(src,i,1);
        end;

        n := n + 1;
        if n > length(key) then
            n := 1;
        end;
    end;
```

```
    // decrypt
    n:=1;
    for i:=1 to length(dst) do

        ps:= pos(substr(dst,i,1),alf);
        if ( ps !=0) then
            pk:= pos(substr(key,n,1),alf);

            ps:= ps-pk;
            if  ps < 1  then
                ps:= ps+length(alf);
            end;

            rst:=rst+substr(alf,ps,1);
        else
            rst:= rst+ substr(dst,i,1);
        end;

        n := n + 1;
        if n > length(key) then
            n := 1;
        end;
    end;

    writeln('   Source message:', src);
    writeln('Encrypted message:', dst);
    writeln;
    writeln('Decrypted message:', rst);

    write('Press <Enter>');
    readln;
end.
```

Password generator

```
// Password generator

program PWGen()
const
    PWLEN = 10; // password length
    N=5;        // number of password options
var
    pw: string[PWLEN]; // password
    alp: string[64];   // alphabet

    r:integer; // random number - alphabet character number
    i:integer; // number of the generated password character

    up:integer; // 1 - convert the letter to lowercase; 2 - leave as is
```

```
    j: integer;

begin
    // character set
    alp := 'abcdefghijklmnopqrstuwvxyz0123456789!$?#_';

    for j:=1 to N do
        // generate password
        pw := '';
        for i:= 1 to PWLEN do
            r := Random(length(alp));
            up := Random(2);
            if ( up = 1) then
                pw:=pw + Upcase(substr(alp,r,1));
             else
                 pw:=pw + substr(alp,r,1);
             end;
        end;

        writeln(j:3, '. ', pw);
    end;

    writeln;
    write('Press <Enter>');
    readln;
end.
```

Writing to a file

```
// Convert integers to a string and write to a file
program p23()
const
    K = 10; // amount of numbers
var
     f:text;         // text file
     fn: string[64]; // file name

     i:integer;
begin
    fn:= 'c:\temp\data.dat';
    writeln('Data file: ',fn);

    f:=rewrite(fn); // open the file to rewriting

    for i:=1 to K do
        writestring(f, IntToStr( random(100)) );
    end;

    close(f);
```

```
    writeln(K, ' numbers written to file:',fn);

    write('Press <Enter>');
    readln;
end.
```

Reading from file

```
// Reading lines from a file and converting them to an integer.
program p23()
var
     f:text;          // text file
     fn: string[64]; // file name

     st: string[15]; // line read from file

     k:integer;     // number
     sum: integer; // sum of numbers
     n:integer;     // count of numbers
     med: float;    // average
begin
    fn:= 'c:\temp\data.dat';
    writeln('Data file: ', fn);

    f:= reset(fn); // open the file to reading
    if (f = -1) then
        writeln('Path/File access error!');
    else
        // read numbers from the file
        n:=0;
        sum:=0;
        while (eof(f) != 1) do
            n:=n+1;
            k:= StrToInt(readstring(f));
            sum:= sum + k;
            writeln(k:4);
        end;

        close(f);

        if n != 0 then
            med:= sum/n;
        end;

        writeln('Numbers in file:',n:5);
        writeln('Sum of numbers:',sum:5);
        writeln('Arithmetic mean:',med:6:2);
    end;

    write('Press <Enter>');
```

```
    readln;
end.
```

Display the contents of text file

```
// Display the contents of text file

// Draw a line
procedure Line(ch: string, n: integer)
var
    i: integer;
begin
    for i := 1 to n do
        write(ch);
    end;
    writeln;
end;

program p20()
var
    f:text; // text file
    fn:string[64]; // file name

    st: string[128]; // line read from file
    n:integer; // number of lines
begin
    fn:= 'C:\Users\nikita\documents\pas\p20.pas';

    f:=reset(fn); // Open file for reading.
                  // The Reset function returns -1 if for some reason
                  // access to the file was not obtained
    if f != -1 then
        writeln(fn);
        Line('-',length(fn));
        n:=0;

        while (eof(f) != 1) do  // while not end of file
            n:=n+1;
            st := readstring(f);  // read line from file
            writeln(n:3, '  ', st);
        end;
        Line('-',length(fn));
    else
        writeln('File access error: ', fn);
        writeln('Invalid name/path or file is in use by another application');

    end;

    writeln;
```

```
    write('Press <Enter>');
    readln;
end.
```

Date and time

```
// Demonstrates the use of the GetTime, GetDay,
// GetMonth, GetYear, DayOfWeek functions

// Returns time in hh:mm:ss format
function TimeToStr(time: integer):string
var
    st: string[8];
    hour: integer;
    min: integer;
    sec: integer;
begin
    hour:= Trunc(time/60/60);
    min:= Trunc((time - hour*3600)/60);
    sec:= time- hour*3600 -min*60;

    st:='';
    if hour < 9 then
        st:= '0';
    end;
    st:= st + IntToStr(hour) +':';

    if min < 9 then
        st:= st + '0';
    end;
    st:= st + IntToStr(min) + ':';

    if sec < 9 then
        st:= st+ '0';
    end;

    st:= st + IntToStr(sec);

    return st;
end;

// returns date in dd/mm/yyyy format
function ShortDate(day: integer, month: integer, year: integer): string
var
    st: string[10];
begin
    st:='';
    day := getDay();
    if day < 10 then
        st:='0';
```

```
    end;
    st:= st + IntToStr(day)+'/';

    month := getMonth();
    if month < 10 then
        st:=st+'0';
    end;
    st:= st+IntToStr(month)+'/';

    year := getYear();
    st:=st+ IntToStr(year);

    return st;
end;


program p1()
var
    day: integer;
    month: integer;
    year: integer;
    dayOfweek: integer;

    weekDay: array[1..7] of string[10] =
               'Sunday', 'Monday', 'Tuesday', 'Wednesday',
               'Thursday Friday Saturday';
    monthName: array[1..12] of string[10] =
               'January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', ,'December';
    hour: integer;
    min: integer;
    sec: integer;

    time: integer;
    time2: integer;
    dtime: integer;

    i: integer;

begin
     writeln('Today is ', getDay(), ' ', monthName[getMonth()],
             getYear():5, ', ' , weekDay[getDayOfWeek()+1]);

     day := getDay();
     dayOfweek:= getDayOfWeek();
     month := getMonth();
     year := getYear();

    writeln('Today is ', day, ' ', monthName[month],
                        year:5, ', ' , weekDay[dayOfweek+1]);
     writeln('Today is ', day, ' ', monthName[month] , year:5);
```

```
     writeln('Today is ', day, '-', month, '-', year);

    time := getTime();
    hour:= time div 60 div 60;
    min:=  (time - hour*3600)  div 60;
    sec:=  time- hour*3600 - min*60;

    writeln('It is ', hour, ':', min, ':', sec, 'now');
    writeln;
    writeln;

    for i:=1 to 7 do
        writeln(i-1:2,' - ', weekDay[i]);
    end;

    writeln;

    day := getDay();
    month := getMonth();
    year := getYear();
    dayOfweek := getDayOfWeek();
    time := getTime();

    writeln('Today ', ShortDate(day, month, year), ' ',
             weekDay[dayOfWeek+1], ' (',dayOfWeek, ')');

    writeln;
    writeln('Now ', TimeToStr(time));

    write('Wait 15 sec and press <Enter>');
    readln;

    time2:=getTime();
    writeln('Now ', TimeToStr(time2));

    dtime:= time2- time;
    writeln('You where waiting ', TimeToStr(dtime));

        write('Press <Enter>');
        readln;
end.
```

Hangman game

```
// Hangman game
Program HangmanGame()
const
      NW = 5; // number of words
      LW = 15; // maximum number of letters in a word
```

```
    TRUE = 1;
     FALSE = 0;
var
    words: array[1..NW] of string[15]  =
            'hangman', 'apple', 'pascal', 'spaceman', 'italia';

    secretWord:string[LW];
    userWord:string[LW];

    ch: string[1]; // letter entered by the player
    k:integer; // number of letters entered by the player
    misses:string[15]; //missing characters (8 letters + 7 commas)

    st2: string[LW];

    found:integer;

    i,j: integer;

    debug: integer;

begin
    writeln;
    writeln('Welcome to the Hangman game!');
    writeln;
    secretWord := words[Random(NW)+1];

    for i := 1 to Length(secretWord) do
          userWord := userWord + '-';
    end;

    //WriteLn('Secret word:',secretWord);
    //WriteLn('User word:',userWord);

    k := 0;
    repeat
        writeln;
        WriteLn('Word:', UpCase(userWord));
        WriteLn('Misses:', UpCase(misses));
        Write('Guess:');

        ReadLn(ch);; // the first character of the entered line

        found := false;
        for i := 1 to Length(secretWord) do
            if substr(secretWord,i,1) = ch then
                //the player guessed the letter
                found := TRUE;

                // replace the current character of the userWord string
                // with the ch character
```

```
                t2 := '';
                for j := 1 to Length(secretWord) do
                    if j = i then
                        st2 := st2 + ch;
                    else
                        st2 := st2 + Substr(userWord,j,1);
                    end;
                end;
                userWord := st2;
            end;
        end;

        if NOT found then
            if Length(misses) = 0 then
                misses := misses + ch;
            else
                misses := misses +',' + ch;
            end;
        end;

        k := k + 1; // number of letters entered

    until  (k = 8) OR (userWord = secretWord);

    writeln;
    if (userWord = secretWord) then
        WriteLn('You are win!');
    else
        WriteLn('You are lost!');
    end;

    WriteLn('The seecret word is ', UpCase(secretWord));

    Write('Press <Enter>');
    Readln;
end.
```