

Язык программирования PascalABC.NET 3.1 2015 – 2016

Обзор новых возможностей
(обновлено: март 2016 г.)

Новое за 2015-2016 гг.

- Case по строкам
- Русские идентификаторы
- Короткие определения функций
- Подпрограммы, методы и операции для символов и строк (частично 2016)
- Подпрограммы и методы для текстовых файлов
- Стандартный тип BigInteger
- Стандартные классы коллекций List<T>, Stack<T>, Queue<T>, HashSet<T>, SortedSet<T>, LinkedList<T>, Dictionary<T>, SortedDictionary<T>
- Короткие функции Arr, Seq, Lst, LLst, Dict, HSet, SSet (частично 2016)
- Операция in для стандартных коллекций (частично 2016)
- Write и Print выводят структурированные данные
- Кортежи, кортежное присваивание (2016)
- Срезы (2016)
- Лямбда-выражения
- Последовательности (частично 2016)
- Бесконечные последовательности (2016)
- Примеры работы с последовательностями
- Модуль ClientServer

Case по строкам, русские идентификаторы

В PascalABC.NET можно делать case по строкам. Это значительно удобнее старого стиля со вложенными if

Можно использовать русские идентификаторы. После небольших переопределений программа преобразуется. Использовать – дело вкуса

Case по строкам

```
begin
  var Country := ReadString;
  write('Столица: ');
  case Country of
    'Россия':   writeln('Москва');
    'Франция':  writeln('Париж');
    'Италия':   writeln('Рим');
    'Германия': writeln('Берлин');
    else        writeln('Нет в базе данных');
  end;
end.
```

Русские идентификаторы

```
type
  цел = integer;
  вещ = real;

procedure Вывод(число: вещ) := Println(число);

function Диапазон(нач, кон: цел) := Range(нач, кон);

begin
  var сумма: вещ := 0.0;
  var количество: цел := 10;

  for var i:=1 to количество do
    сумма += 1/i;

  Вывод(сумма);

  Вывод(Диапазон(1, количество).Sum(знач -> 1/знач));
end.
```

Короткие определения функций и процедур

В PascalABC.NET допускаются короткие определения для функций, задаваемых одним выражением. Тип возвращаемого значения можно не задавать – он выводится автоматически.

Также допускаются короткие определения процедур, задаваемых одним оператором

Коллекция коротких определений подпрограмм

```
function Sqr3(x: integer) := x*x*x;  
function CircleLen(r: real): real := 2 * Pi * r;  
function Hypot(a,b: real) := sqrt(a*a + b*b);  
function Len(x1,y1,x2,y2: real) := Hypot(x2-x1,y2-y1);  
function DigitCount(x: integer) := abs(x).ToString.Length;  
function Минимум(a,b,c: real): real := Min(Min(a,b),c);  
procedure Вывод<ЛюбойТип>(x: ЛюбойТип) := Println(x);  
begin  
    Println(Sqr3(2),CircleLen(1));  
    Println(Hypot(3,4),Len(1,1,3,4));  
    Println(DigitCount(-1234));  
    Вывод(Минимум(5,3,8));  
end.
```

Новые подпрограммы и методы для СИМВОЛОВ И СТРОК

Методы Ord и Chr теперь работают с Unicode-кодировкой (до февраля 2016 года они работали с кодировкой Windows).

Ряд методов расширения и операций добавлен в типы Char и String

Char

```
begin
  var i := Ord('Б'); // Код в Unicode
  var c := Chr(i);
  Println(i, c);

  c := 'z';
  Println(c.Code, c.Pred, c.Succ);
  Println(c.IsDigit, c.IsLetter);
  Println(c.IsLower, c.IsUpper);
  Println(c.ToLower, c.ToUpper);

  c := '5';
  Println(c.ToDigit);
end.
```

```
1041 Б
122 y {
False True
True False
z Z
5
```

String

```
begin
  var s := 'Целое = '+5;
  Println(s);
  s := 'Вещественное = '+3.14;
  Println(s);

  Println('x'*10); // строка из 10 символов 'x'
  Println('abc'*3);

  s := 'No yes yes no yes No';
  s.ToWords.Println(','); // разбиение строки на слова
  // Matches - все вхождения с помощью регулярных выражений
  foreach var m in s.Matches('no', RegexOptions.IgnoreCase) do
    Println(m, m.Index); // подстрока 'no' и её позиция в s
end.
```

```
Целое = 5
Вещественное = 3.14
xxxxxxxxxxx
abcabcabc
No, yes, yes, no, yes, No
No 0
no 11
No 18
```

Методы для текстовых файлов

Функции `OpenRead`, `OpenWrite`, `OpenAppend` позволяют описывать и открывать файл одной строкой
Определен ряд методов: `f.ReadInteger`, `f.ReadString`, ..., `f.Eof`, `f.Close`, `f.Write(...)`, `f.Name`

Функция `ReadLines` возвращает **последовательность** строк. При ее вызове файл открывается. По последовательности мы можем совершить цикл `foreach`, по окончании которого файл закрывается.
`ReadLines` не считывает файл в память: в каждый момент в памяти находится лишь одна строка!

Описание задачи

Задача. Дан текстовый файл `freqs.txt` формата

```
201 3.18 аккомпанемент noun
202 1.71 аккомпанировать verb
203 4.84 аккорд noun
204 2.88 аккордеон noun
205 1.16 аккумулировать verb
206 39.85 аккумулятор noun
207 3.12 аккумуляторный adj
208 3.31 аккуратненький adj
209 48.48 аккуратно adv
210 2.94 аккуратность noun
211 23.38 аккуратный adj
...
```

Вывести все глаголы verb

Решение 1

```
begin
  var f := OpenRead('freqs.txt');
  while not f.Eof do
    begin
      var ss := f.ReadlnString.ToWords;
      if ss[3] = 'verb' then
        writeln(ss[2]);
      end;
    f.Close;
  end.
```

Решение 2

```
begin
  foreach var s in ReadLines('freqs.txt') do
    begin
      var ss := s.ToWords;
      if ss[3] = 'verb' then
        writeln(ss[2]);
      end;
    end.
```


Стандартные классы коллекций

В стандартном модуле определены стандартные классы коллекций .NET:

Класс коллекции	Описание	Короткая функция для создания
List<T>	Список (расширяемый динамический массив)	Lst(1,5,3)
Stack<T>	Стек	
Queue<T>	Очередь	
HashSet<T>	Множество на базе хеш-таблицы	HSet(1,5,3)
SortedSet<T>	Множество на базе	SSet(1,5,3)
LinkedList<T>	Двусвязный список	LLst(1,5,3)
Dictionary<T>	Словарь на базе хеш-таблицы	Dict(KV('слон',2), KV('бегемот',3))
SortedDictionary<T>	Словарь на базе бинарного дерева поиска	

Операция `in` для стандартных коллекций

Для всех стандартных коллекций, для статических массивов и встроенных множеств определена универсальная операция `in`.

Всё, для чего работает `in`

```
begin
  var s := 'abcde слово';
  var sq := Seq(1,3,4);
  var a := Arr(1,3,4);
  var l := Lst(1,3,4);
  var ll := LLst(1,3,4);
  var h := HSet(1,3,4);
  var st := SSet(1,3,4);
  var d := Dict(KV('бегемот',3),KV('зебра',2));
  var aa: array [1..3] of integer := (3,2,1);
  var ss: set of integer := [1,3,4];

  Println('f' in s, 'слово' in s);
  Println(3 in sq);
  Println(1 in a, 1 in l, 1 in ll);
  Println(2 in h, 2 in st);
  Println('зебра' in d); //есть ли ключ в словаре
  Println(0 in aa);
  Println(0 in st);
end.
```

Окно вывода:

```
False True
True
True True True
False False
True
False
False
```

Write и Print для структурированных данных

Стандартные процедуры Write и Print выводят значения любых структурированных данных

Массивы и последовательности выводятся в []

Записи и кортежи выводятся в ()

Множества и словари выводятся в {}

Println для всех контейнеров

```
begin
  var aa: array [1..3] of integer := (1,3,4);
  var sq := Seq(1,3,4);
  var a := Arr(1,3,4);
  var r := (1,3,4);
  var l := Lst(1,3,4);
  var ll := LLst(1,3,4);
  var h := HSet(1,3,4);
  var st := SSet(1,3,4);
  var ss: set of integer := [1,3,4];
  var d := Dict(KV('бегемот',3),KV('зебра',2));

  Println(aa);
  Println(sq);
  Println(a);
  Println(l);
  Println(ll);
  Println(r);
  Println(h);
  Println(st);
  Println(ss);
  Println(d);
end.
```

Окно вывода:

```
[1,3,4]
[1,3,4]
[1,3,4]
[1,3,4]
[1,3,4]
(1,3,4)
{1,3,4}
{1,3,4}
{4,3,1}
{(бегемот,3),(зебра,2)}
```

Write и Print для структурированных данных

Процедуры Write и Print успешно справляются с рекурсивными структурами такими как связные списки и деревья

Вывод связного списка и массива кортежей

```
type
  Node<T> = auto class
    data: integer;
    next: Node<T>;
end;

begin
  var p: Node<integer> := nil;
  for var i:=1 to 10 do
    p := new Node<integer>(Random(100),p);
    Writeln(p);

    var a := Arr(('Иванов',23),
                 ('Попов',20),
                 ('Козлова',21));
    Writeln(a);
end.
```

```
(33, (33, (55, (42, (93, (86, (47, (53, (92, (75, nil))))))))))
[ (Иванов, 23), (Попов, 20), (Козлова, 21) ]
```

Вывод бинарного дерева

```
type
  Node<T> = auto class
    data: integer;
    left, right: Node<T>;
end;

function CreateTree(n: integer): Node<integer>;
begin
  Result := n=0 ? nil :
    new Node<integer>(Random(100),
                     CreateTree((n-1) div 2),
                     CreateTree(n-1-(n-1) div 2));
end;

begin
  var root := CreateTree(10);
  Writeln(root);
end.
```

```
(10, (27, (14, nil, nil), (30, nil, (10, nil, nil))), (76, (51, nil, (64, nil, nil)), (80, nil, (67, nil, nil))))
```

Кортежи

- Кортеж (Tuple) – безымянная запись, создаваемая «на лету»
- Тип кортежа: (string,integer), значение типа кортеж: ('Иванов',23)
- Максимальное количество компонент = 7 (ограничение .NET)
- Доступ к компонентам осуществляется по индексу: t[0], t[1].
Индекс должен быть константой.

Пример кортежа

```
begin
  var t: (string,integer);

  // Упаковка данных в кортеж
  t := ('Иванов',23);

  Println(t);
  Println(t[0],t[1]);

  var name: string;
  var age: integer;

  // Распаковка кортежа в переменные или
  // кортежное присваивание
  (name,age) := t;

  Println(name,age);
end.
```

Окно вывода:

```
(Иванов,23)
Иванов 23
Иванов 23
```

Примеры использования кортежей

Кортежи кардинально меняют стиль решения ряда задач

Вместо Swar

```
begin
  var a := 3;
  var b := 5;
  Println(a,b);
  (a,b) := (b,a);
  Println(a,b);

  var c := 7;
  Println(a,b,c);
  (a,b,c) := (b,c,a);
  Println(a,b,c);

  (a,b) := (1,1,1); // справа - более длинное!
end.
```

```
3 5
5 3
5 3 7
3 7 5
```

Кортежи как возвращаемые значения функций

```
function SP(a,b: real) := (a*b,2*(a+b));

function DivMod(a,b: integer) := (a div b,a mod b);

begin
  var S,P: real;
  (S,P) := SP(2,3); // распаковка в переменные
  Println(S,P);
  (S,P) := SP(3,4);
  Println(S,P);

  Println(DivMod(20,7));
end.
```

```
6 10
12 14
(2,6)
```

Примеры использования кортежей 2

Кортежи радикально меняют стиль решения ряда задач

Нахождение НОД

```
begin
  var a := 126;
  var b := 72;

  while b>0 do
    (a,b) := (b, a mod b);

  Println(a);
end.
```

18

Кролики Фибоначчи

```
begin
  var n := 15;

  var a := 1;
  var b := 1;
  Print(a);
  for var i := 3 to n do
    begin
      Print(b);
      (a,b) := (b,a+b);
    end;
end.
```

1 1 2 3 5 8 13 21 34 55 89 144 233 377

Срезы

- Срез – это подмножество элементов коллекции в заданном диапазоне с заданным шагом
- Срезы имеют вид **a[from : to]** или **a[from : to : step]**
- Диапазон **[from : to]**. Элемент **to** не входит в диапазон
- Срезы с пропуском значений a[f :], a[: t], a[f :: s], a[: t : s], a[:: s]
- Срезы реализованы для строк, динамических массивов и списков List. Срез для строки – это строка, для динамического массива – динамический массив, для списка - список

Разные виды срезов

```
begin
  var a := Arr(0,1,2,3,4,5,6,7,8,9);
  var l := Lst(0,1,2,3,4,5,6,7,8,9);
  var s := 'PascalABC.NET 2016';
  Println(a[2:5]);
  Println(s[7:14]);
  Println(l[2:9:2]);
  Println(l[8:1:-2]);
  Println('-'*20);
  Println(a[2:]);
  Println(a[:5]);
  Println('-'*20);
  Println(l[0:l.Count:3]);
  Println(l[::3]);
  Println('-'*20);
  Println(a[a.Length-1:-1:-1]); // инверсия
  Println(a[::-1]); // инверсия
  Println('-'*20);
  Println(s[::-1]); // инверсия
  Println(a[0::2]+a[1::2]);
end.
```

Окно вывода:

```
[2,3,4]
ABC.NET
[2,4,6,8]
[8,6,4,2]
-----
[2,3,4,5,6,7,8,9]
[0,1,2,3,4]
-----
[0,3,6,9]
[0,3,6,9]
-----
[9,8,7,6,5,4,3,2,1,0]
[9,8,7,6,5,4,3,2,1,0]
-----
6102 TEN.CBAlacsaP
[0,2,4,6,8,1,3,5,7,9]
```

Лямбда-выражения

Лямбда-выражения представляют собой функции, создаваемые «на лету». Они облегчают написание и восприятие текста программы. Лямбда-выражения присутствуют практически во всех современных распространенных языках программирования.

Типы простейших лямбда-выражений:

- $T1 \rightarrow T2$ – функция с одним параметром типа $T1$, возвращающая значение типа $T2$
- $(\text{real}, \text{real}) \rightarrow \text{boolean}$ – функция с двумя вещественными параметрами, возвращающая `boolean`

Пример 1

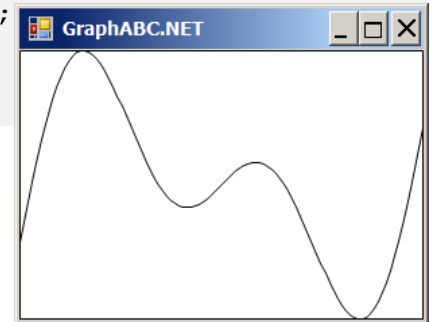
```
var f: real->real;  
    less: (real,real) -> boolean;  
begin  
    f := x -> 3*x-2;  
    writeln(f(2));  
    less := (a,b) -> a<b;  
    writeln(less(3,2));  
end.
```

```
4  
False
```

Использование лямбда-выражений позволяет не описывать множество маленьких функций, которые используются в программе один раз

Пример 2

```
uses GraphABC;  
  
begin  
    Window.SetSize(300,200);  
    Draw(x->x*cos(x));  
end.
```



Лямбда-выражения активно используются как параметры стандартных подпрограмм. Например, процедура `Draw` рисует график функции на полное графическое окно

Последовательности

Последовательность **sequence of T** представляет собой набор элементов, получаемых последовательно один за другим.

Элементы последовательности **не хранятся одновременно в памяти**, а генерируются с помощью некоторого алгоритма. Таким образом, в памяти в каждый момент времени хранится лишь один элемент.

Элементы последовательности можно перебрать с помощью цикла **foreach** и вывести с помощью метода `Println`

Генерация с помощью Range

```
begin
  var sq: sequence of integer := Range(1,10);

  foreach var x in sq do
    Print(x);
  Println;

  var sq1 := Range(1,20,2); // шаг 2
  sq1.Println(', ');

  Range('a','z').Println;
  Range(1.0,2.0,10).Println;
end.
```

```
1 2 3 4 5 6 7 8 9 10
1,3,5,7,9,11,13,15,17,19
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2
```

Генерация с помощью Seq...

```
begin
  var s := Seq(1,5,3);
  s.Println;
  SeqFill(10,77).Println;
  SeqRandom(10).Println;
  SeqGen(10,i->i*i,1).Println;
  SeqGen(10,1,x->x*2).Println; // степени 2
  SeqGen(10,1,1,(x,y)->x+y).Println; //Фибоначчи
end.
```

```
1 5 3
77 77 77 77 77 77 77 77 77 77
30 14 86 45 79 3 8 33 66 18
1 4 9 16 25 36 49 64 81 100
1 2 4 8 16 32 64 128 256 512
1 1 2 3 5 8 13 21 34 55
```

Методы последовательностей

Последовательности содержат ряд методов, входящих в технологию **LINQ**. Основные:

- Print, Println – вывод
- Min, Max, Sum, Average – минимум, максимум, сумма, среднее
- Select – проекция, Where – фильтрация, OrderBy – сортировка

Методы последовательностей можно применять к **массивам, спискам, множествам**

Min, Max, Sum, Average

```
begin
    var s := Seq(12,5,7,13,1,22,3,4,9,7,7,9);
    s.Println;
    Println('Минимум =',s.Min);
    Println('Максимум =',s.Max);
    Println('Сумма =',s.Sum);
    Println('Среднее =',s.Average);
end.
```

```
12 5 7 13 1 22 3 4 9 7 7 9
Минимум = 1
Максимум = 22
Сумма = 99
Среднее = 8.25
```

Select, Where, OrderBy

```
begin
    var s := Seq(('Умнова',16), ('Иванов',23),
        ('Попова',17), ('Козлов',24));
    s.Where(x -> x[1] >= 18).Println;
    Println('Сортировка по фамилии:');
    s.OrderBy(x -> x[0]).Println;
    Println('Сортировка по возрасту:');
    s.OrderBy(x -> x[1]).Println;
    var a := Seq(1,5,3,8,7,6,4);
    a.Println;
    // Возвести все элементы в квадрат
    a.Select(x -> x*x).Println;
    // Отфильтровать элементы < 7 и увеличить их на 10
    a.Where(x -> x<7).Select(x -> x + 10).Println;
end.
```

```
(Иванов,23) (Козлов,24)
Сортировка по фамилии:
(Иванов,23) (Козлов,24) (Попова,17) (Умнова,16)
Сортировка по возрасту:
(Умнова,16) (Попова,17) (Иванов,23) (Козлов,24)
1 5 3 8 7 6 4
1 25 9 64 49 36 16
11 15 13 16 14
```

Новые методы генерации последовательностей

Конечные последовательности:

- To, Downto, Times - альтернатива Range

Бесконечные последовательности:

- Step для целых и вещественных – бесконечная арифметическая прогрессия
- Iterate – бесконечная последовательность, заданная рекуррентным соотношением
- Repeat – бесконечная последовательность повторяющихся элементов
- Cycle - бесконечная последовательность повторяющихся последовательностей

Генерация конечных последовательностей

```
begin
  2.To(10).Println;
  10.Downto(2).Println;
  5.Times.Println;

  foreach var x in 2.To(10) do
    Print(x);
  end.
```

```
2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2
0 1 2 3 4
2 3 4 5 6 7 8 9 10
```

Генерация бесконечных последовательностей

```
begin
  var inf := 1.0.Iterate(x->x+0.1);
  inf.Take(10).Println; // Take обрезает до 10 эл-тов

  var inf2 := 5.Step;
  inf2.Take(10).Println;
  var inf3 := 5.0.Step(0.2); //начиная с 5 с шагом 0.2
  inf3.Take(10).Println;

  var inf4 := 666.Repeat;
  inf4.Take(10).Println;
  var inf5 := Seq(1,2,3).Cycle;
  inf5.Take(10).Println;
end.
```

```
1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9
5 6 7 8 9 10 11 12 13 14
5 5.2 5.4 5.6 5.8 6 6.2 6.4 6.6 6.8
666 666 666 666 666 666 666 666 666 666
1 2 3 1 2 3 1 2 3 1
```

Новые методы последовательностей

- Cartesian – декартово произведение
- SplitAt – разбиение на 2 последовательности
- Interleave – чередование элементов последовательностей
- Partition – разбиение на подпоследовательности по условию
- Batch – разбиение на подпоследовательности одинаковой длины

Cartesian

```
begin
  var s1 := Range(1,3);
  var s2 := Range('a','c');
  s1.Cartesian(s2).Println;
  s1.Cartesian(s2, (x,y)->x+y).Println;
end.
```

```
(1,a) (1,b) (1,c) (2,a) (2,b) (2,c) (3,a) (3,b) (3,c)
1a 1b 1c 2a 2b 2c 3a 3b 3c
```

SplitAt, Interleave, Partition, Batch

```
begin
  var sq := Range(1,10);
  Println(sq.SplitAt(5));

  var sq1,sq2: sequence of integer;
  (sq1,sq2) := sq.SplitAt(5);
  sq1.Interleave(sq2).Println;

  Println(SeqRandom(10).Partition(x->x.IsEven));
  sq.Batch(3).Println;
  sq.SelectMany(x->x).Println;
  sq.Batch(3,seq->seq.Sum).Println;
end.
```

```
([1,2,3,4,5],[6,7,8,9,10])
1 6 2 7 3 8 4 9 5 10
([24,90],[65,27,85,55,39,21,79,65])
[1,2,3] [4,5,6] [7,8,9] [10]
1 2 3 4 5 6 7 8 9 10
6 15 24 10
```

Новые методы последовательностей 2

- `Tabulate` – табулирование функции
- `Pairwise` – разбиение на кортежи из соседних пар
- `Numerate` – нумерация последовательности
- `ZipTuple` – объединение двух последовательностей в последовательность кортежей
- `UnzipTuple` – разъединение последовательности кортежей на две последовательности

Tabulate, Pairwise, Numerate

```
begin
  Range(1,5).Tabulate(x->x*x).Println;
  Range(1,6).Pairwise.Println;
  Arr('Иванов','Петров','Сидоров').Numerate
    .Println;
  'ABCDEF'.Numerate.Println;
end.
```

```
(1,1) (2,4) (3,9) (4,16) (5,25)
(1,2) (2,3) (3,4) (4,5) (5,6)
(1,Иванов) (2,Петров) (3,Сидоров)
(1,A) (2,B) (3,C) (4,D) (5,E) (6,F)
```

ZipTuple, UnzipTuple

```
begin
  var sq1 := Range('a','e');
  var sq2 := 1.Step;
  var sq3 := 2.5.Step(0.5);

  var q1 := sq1.ZipTuple(sq2).Println;
  writeln(q1.UnzipTuple);
  var q2 := sq1.ZipTuple(sq2,sq3).Println;
  writeln(q2.UnzipTuple);
end.
```

```
(a,1) (b,2) (c,3) (d,4) (e,5)
([a,b,c,d,e],[1,2,3,4,5])
(a,1,2.5) (b,2,3) (c,3,3.5) (d,4,4) (e,5,4.5)
([a,b,c,d,e],[1,2,3,4,5],[2.5,3,3.5,4,4.5])
```

Новые методы последовательностей 3

- MinBy, MaxBy – минимум-максимум по полю
- TakeLast(n) – последние n элементов
- ToArray, ToList, ToLinkedList, ToHashSet, ToSortedSet, AsEnumerable – преобразования из одного типа последовательности в другой

MinBy, MaxBy, TakeLast

```
begin
  var a := Seq(('Ivanov',18), ('Avilov',15),
              ('Popov',17));
  Println(a.MinBy(x->x[1]));
  Println(a.MaxBy(x->x[0]));

  Range(1,9).TakeLast(3).Println;
end.
```

```
(Avilov,15)
(Popov,17)
7 8 9
```

ToArray, ToList, ToLinkedList, ToHashSet, ToSortedSet, AsEnumerable

```
begin
  var s := Seq(1,3,5,2,3,2,1,5,3);
  Println(s.ToHashSet);
  Println(s.ToSortedSet);

  var a: array of integer := s.Select(x->x+1).ToArray;
  var l: List<integer> := a.ToList;
  var ll: LinkedList<integer> := a.ToLinkedList;
  var s: sequence of integer := a.AsEnumerable;
end.
```

```
{1,3,5,2}
{1,2,3,5}
```

Модуль ClientServer

- Простой модуль для клиент-серверного взаимодействия
- Может использоваться для школьников в теме Сетевое программирование

Сервер

```
uses ClientServer;
uses ABCObjects, GraphABC;

var obj: ObjectABC;
    x := 240; y := 300;

procedure ProcessCommand(команда: string);
const d = 5;
begin
    case команда.ToLower of
'left': begin x += d; obj.MoveOn(d,0); end;
'right': begin x -= d; obj.MoveOn(-d,0); end;
'up': begin y -= d; obj.MoveOn(0,-d); end;
'down': begin y += d; obj.MoveOn(0,d); end;
    end;
    LineTo(x,y);
    Window.Title := 'Команда: '+команда;
end;

begin
    Window.Title := 'Сервер: ждет команды';
    obj := new CircleABC(x,y,20,Color.Yellow);
    MoveTo(x,y);
    RunServer(ProcessCommand);
end.
```

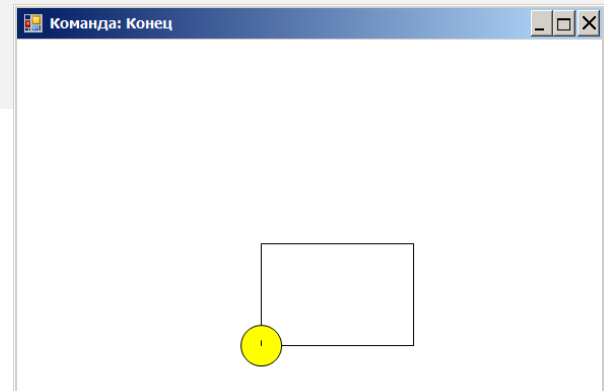
Клиент

```
uses ClientServer;

begin
    servername := '127.0.0.1';

    var команды := Seq('Left')*30 + Seq('Up')*20+
        Seq('Right')*30 + Seq('Down')*20 + 'Конец';

    foreach var команда in команды do
        begin
            SendCommand(команда);
            Sleep(100);
        end;
end.
```



Заключение

- [PascalABC.NET 2015-2016](#) меняет стиль решения ряда задач. Код становится более простым, понятным, легче пишется и меняется.
- Богатый набор стандартных классов-коллекций существенно расширяет набор простых задач, которые можно решать на PascalABC.NET.
- Методы, встроенные в символы, строки, текстовые файлы, позволяют в несколько раз сократить решения, связанные с обработкой текстовой информации.
- Тип [BigInteger](#) открывает возможность простого решения ряда задач с числами, ранее предлагавшихся на олимпиадах по программированию.
- Процедуры Write и Print работают не только с элементарными, но и со структурными данными, выводя их содержимое. Нет необходимости тратить время на написание цикла для вывода массива или словаря.
- Наличие в языке [последовательностей](#), методов последовательностей и [лямбда-выражений](#) кардинально меняет стиль решения задач: решения даются [в функциональном стиле](#), не содержат циклов и состоят из запросов и цепочек запросов к последовательностям.
- Методы последовательностей можно использовать для строк, динамических массивов, классов списков, множеств и словарей.
- Ряд методов последовательностей был добавлен в стандартную библиотеку PascalABC.NET в 2016 году и отсутствует в стандартных .NET-библиотеках.
- [Кортежи](#) и [срезы](#) появились в PascalABC.NET раньше чем в C# и позволяют манипулировать при решении задач новыми высокоуровневыми сущностями, улучшая стиль решения.