PascalABC.NET

Последовательности

- Тип sequence of T
- Генераторы последовательностей
- Ленивость
- Методы последовательностей
- Последовательности и массивы
- Операторы yield и yield sequence
- Бесконечные последовательности

Обновлено 25.01.18

Оглавление

К оглавлению

Что такое последовательность

- Последовательность **sequence of** T – это набор элементов, получаемых последовательно один за другим
- Элементы последовательности генерируются с помощью специальных функций – **генераторов последовательностей**. Простейшими генераторами являются функции Seq и Range(a,b)
- Генератор Range(a,b,h) возвращает последовательность целых в диапазоне от a до b с шагом h
- Для перебора элементов последовательности используется цикл **foreach**
- Вывести элементы последовательности можно также с помощью метода Println

Цикл foreach и метод Println

```
begin
  var sq: sequence of integer;
// Seq генерирует последовательность, перечисляя значения
  sq := Seq(1,2,3,4,5);
// Для перебора элементов последовательности используется
// цикл foreach
  foreach var x in sq do
    Print(x);
  Println;
  var sq1 := Seq(1,3,5,7,9);
// Метод Println выводит элементы через пробел
  sq1.Println;
  Seq(2,4,6,8,10).Println
end.
```

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

Генерация с помощью Range

```
begin
// Если a,b - целые, то Range(a,b) генерирует
// последовательность целых в диапазоне от a до b с шагом 1
  var sq: sequence of integer := Range(1,5);
  foreach var x in sq do
    Print(x);
  Println;
// Range(a,b,h) генерирует последовательность целых
// в диапазоне от a до b с шагом h
  var sq1 := Range(1,9,2); // шаг 2
  sq1.Println;
// Range может генерировать последовательность символов
// в заданном диапазоне
  Range('a','z').Println;
end.
```

```
1 2 3 4 5
1 3 5 7 9
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
```

Операции с последовательностями

- При сложении последовательностей s1 и s2 образуется последовательность, в которой после элементов последовательности s1 идут элементы последовательности s2
- При умножении последовательности на число n образуется последовательность, полученная сложением n штук исходных последовательностей
- Для поиска элемента в последовательности используется операция in

Сложение, операция in

```
begin
  var s := Seq(1,2,3) + Seq(4,5,6);
  s.Print;
  Println(4 in s);
  Println(8 in s);
end.
```

```
1 2 3 4 5 6
True
False
```

Умножение на число

```
begin
  var sq := Seq(1,2,3) * 3;
  sq.Print;
end.
```

```
1 2 3 1 2 3 1 2 3
```

Генератор Range и цикл for

- С помощью генератора Range(1,n) генерируется та же последовательность, что и в цикле for var i:=1 to n do
- Это означает, что Range можно использовать как замену цикла for – либо в комбинации с циклом foreach, либо в комбинации с использованием методов последовательностей
- Огромное преимущество генератра Range в том, что можно применять методы последовательностей, что позволяет записать решение **одной строкой**

Использование for

```
begin
  var n := 1000000;
// Вычисление суммы квадратных корней чисел от 1 до n
// с помощью цикла for.
// Цикл for генерирует последовательность чисел от 1 до n,
// которые последовательно записываются в переменную i
  var s := 0.0;
  for var i:=1 to n do
    s += Sqrt(i);
  s.Println;
end.
```

```
666667166.458842
```

Использование Range

```
begin
  var n := 1000000;
  var s := 0.0;
// Вычисление суммы квадратных корней чисел от 1 до n
// с помощью цикла foreach и генератора Range.
// Range генерирует последовательность чисел от 1 до n,
// которые цикл foreach последовательно записывает в i
  foreach var i in Range(1,n) do
    s += Sqrt(i);
  s.Println;
// Другой способ вычисления суммы - с помощью метода Sum
// для последовательности. Лямбда-выражение x->Sqrt(x)
// проектирует все элементы x на Sqrt(x) и затем они
// суммируются
  Range(1,n).Sum(x->Sqrt(x)).Println
end.
```

```
666667166.458842
666667166.458842
```

Генераторы ReadSeqInteger и ReadSeqReal

- С помощью генератора последовательности ReadSeqInteger(n) можно получить последовательность n целых, введённых с клавиатуры
- Аналогично ReadSeqReal(n) – последовательность n вещественных, введённых с клавиатуры
- Использование ReadSeqInteger и ReadSeqReal позволяет отделить получение значений от их обработки

Ввод значений в цикле

```
begin
  var n := 10;
// Вычисление суммы n вводимых значений
// Значение вводится и тут же обрабатывается
// Недостаток - механизмы ввода и обработки перемешиваются
  var s := 0;
  loop n do
  begin
    var x := ReadInteger;
    s += x
  end;
  Print(x);
end.
```

```
1 2 3 4 5 6 7 8 9 10
55
```

Ввод при помощи ReadSeq

```
begin
  var n := 10;
// Вначале ReadSeqInteger формирует последовательность
// вводимых данных. Таким образом, часть, связанная с
// вводом, отделена от основного алгоритма
// Тип sq автоматически выводится как sequence of integer
  var sq := ReadSeqInteger(n);
// Далее идёт алгоритм вычисления суммы
// Он не содержит операторов, связанных с вводом данных
  var s := 0;
  foreach var x in sq do
    s += x
  Print(x);
end.
```

```
1 2 3 4 5 6 7 8 9 10
55
```

Генераторы SeqGen

- Генераторы SeqGen позволяют задавать последовательности, используя различные алгоритмы:
  - SeqGen(n, i->f(i)) – i-тый элемент определяется формулой f(i), элементы индексируются с 0
  - SeqGen(n, first, x->next(x)) – первый элемент равен first, а каждый следующий определяется по предыдущему x формулой next(x)
  - SeqGen(n, a, b, (a,b)->next(a,b)) – первые два элемента равны a и b, а каждый следующий определяется по двум предыдущим формулой next(a,b)
- Параметр n задаёт количество элементов в последовательности

SeqGen – пример 1

```
begin
  var n := 10;
// Последовательность квадратов значений начиная с нуля
  var q := SeqGen(n,i->i*i);
  foreach var x in q do
    Print(x);
  Println;
// Последовательность кубов значений начиная с 1
  var q1 := SeqGen(n,i->i*i*i,1);
  q1.Println;
end.
```

```
0 1 4 9 16 25 36 49 64 81
1 8 27 64 125 216 343 512 729 1000
```

SeqGen – пример 2

```
begin
  var n := 12;
// Арифметическая прогрессия
  var q := SeqGen(n,1,x->x+2);
  q.Println;
// Геометрическая прогрессия
  var q1 := SeqGen(n,1,x->x*2);
  q1.Println;
// Числа Фибоначчи: первые два равны 1,
// а каждое следующее - сумме двух предыдущих
  var q2 := SeqGen(n,1,1,(x,y)->x+y);
  q2.Println;
end.
```

```
1 3 5 7 9 11 13 15 17 19 21 23
1 2 4 8 16 32 64 128 256 512 1024 2048
1 1 2 3 5 8 13 21 34 55 89 144
```

«Ленивость» вычислений

- Элементы последовательности **не хранятся в памяти одновременно**, а вычисляются и возвращаются функцией-генератором по одному элементу
- Принцип получения элементов по мере необходимости в тот момент, когда они становятся нужны, называется **ленивостью вычислений**

Сохранение последовательности в переменной

```
begin
// В памяти не хранятся элементы последовательности!
  var sq := Range(1,100000);
// В этот момент элементы ещё не начали вычисляться!
  var sum := 0;
// Элементы последовательности начинают вычисляться по
// одному в момент прохода по ним циклом for.
// В каждый момент в памяти хранится только текущий элемент
  foreach var x in sq do
    sum += x;
  sum.Println;
end.
```

```
705082704
```

Последовательность как алгоритм

```
begin
  var n := 1000000;
// В переменной sq хранится алгоритм получения n членов
// геометрической прогрессии и значение текущего элемента
// Размер памяти, занимаемой переменной sq,
// мал и не зависит от n
  var sq := SeqGen(n,1.0,t->t/2);

  var sum := 0.0;
// На каждом шаге по предыдущему элементу t вычисляется
// следующий по правилу t := t/2
  foreach var x in sq do
    sum += x;
  sum.Println;
end.
```

```
2
```

Генераторы SeqRandom

- Генераторы SeqRandomInteger(n) и SeqRandomReal(n) возвращают последовательность n случайных элементов соответствующего типа
- В вызовах SeqRandomInteger(n,a,b) и SeqRandomReal(n,a,b) дополнительные параметры a и b задают диапазон генерации случайных чисел [a,b]
- Из-за ленивости повторный проход по последовательности, созданной с помощью генераторов SeqRandom, возвращает **новые** значения(!)

Пример 1

```
begin
  var n := 5;
// Последовательность случайных целых от 1 до 100
  var sq := SeqRandomInteger(n);
  sq.Println;
// Последовательность случайных целых от 2 до 5
  SeqRandomInteger(n,2,5).Println;
// Последовательность случайных вещественных от 0 до 10
  SeqRandomReal(n).Println;
// Последовательность случайных вещественных от 1 до 2
  SeqRandomReal(n,1,2).Println;
end.
```

```
41 75 69 0 48
4 2 2 4 5
9.12771103863032 6.95180140759414 6.96582832232389
4.62824112019885 2.76186555286956
1.49661216768278 1.34373120467352 1.83351196573745
1.52549836715939 1.52268430382138
```

Пример 2

```
begin
  var n := 10;
// Последовательность случайных целых от 1 до 100
  var sq := SeqRandomInteger(n);
  sq.Println;
// При повторном использовании - другие случайные значения!
  sq.Println;
// Чтобы при повторном использовании была та же
// последовательность, её надо превратить в массив
  var sq1 := SeqRandomInteger(n,1,10).ToArray;
  sq1.Println;
  sq1.Println;
end.
```

```
54 70 7 61 93 66 68 74 100 71
15 49 41 19 74 66 10 51 63 17
5 5 9 6 10 5 7 10 7 2
5 5 9 6 10 5 7 10 7 2
```

Простейшие методы последовательностей

- Методы Count, Min, Max, Sum, Average вычисляют соответственно количество, минимальное и максимальное значение, сумму и среднее элементов последовательности
- Метод Count может вызываться с параметром-лямбдой, задающей условие отбора, а методы Min, Max, Sum, Average – с параметром-лямбдой, задающей проекцию
- Метод Take(n) вернёт последовательность из n первых элементов исходной последовательности, метод Skip(n) – последовательность после пропуска первых n элементов,
  метод TakeLast(n) – последовательность из n последних элементов исходной последовательности,
  метод SkipLast(n) – последовательность без n последних элементов,
  методы First и Last возвращают первый и последний элемент последовательности

Count, Min, Max, Sum, Average

```
begin
  var s := Seq(12,5,7,13,1,22,3,4,9,7,7,9);
  s.Println;
  Println('Количество =',s.Count);
  Println('Количество чётных =',s.Count(x->x mod 2=0));
  Println('Минимум =',s.Min);
  Println('Максимум =',s.Max);
  Println('Сумма =',s.Sum);
  Println('Сумма квадратов =',s.Sum(x->x*x));
  Println('Среднее =',s.Average);
end.
```

```
12 5 7 13 1 22 3 4 9 7 7 9
Количество = 12
Количество чётных = 3
Минимум = 1
Максимум = 22
Сумма = 99
Сумма квадратов = 1157
Среднее = 8.25
```

First, Last, Take, Skip

```
begin
  var s := Range(1,9);
  s.Println;
  Println(s.First,s.Last);
  s.Take(3).Println;
  s.Skip(3).Println;
  s.TakeLast(3).Println;
  s.SkipLast(3).Println;
end.
```

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

Фильтрация, проекция и сортировка

Важнейшими методами последовательностей являются:

- Select(x -> f(x)) – проекция (трансформация) элементов x на элементы f(x)
- Where(x -> filter(x)) – фильтрация (отбор) элементов, удовлтворяющих условию filter
- OrderBy(x -> x) – сортировка по возрастанию, OrderBy(x -> x.поле) – сортировка по заданному полю по возрастанию, OrderByDescending – сортировка по убыванию

Эти методы преобразуют последовательность в другую последовательность и могут вызываться **по цепочке**: s.Where(x -> x>3).OrderBy(x -> x).Select(x -> x*x).Println и т.д.

Select, Where, OrderBy

```
begin
  var s := Seq(1,5,3,8,7,6,4);
  s.Println;
// Возвести все элементы в квадрат
  s.Select(x -> x*x).Println;
// Отфильтровать элементы <7 и увеличить их на 10
  s.Where(x -> x<7).Select(x -> x + 10).Println;
// Отфильтровать элементы >3 и отсортировать по возрастанию
  s.Where(x -> x>3).OrderBy(x -> x).Println;
end.
```

```
1 5 3 8 7 6 4
1 25 9 64 49 36 16
11 15 13 16 14
4 5 6 7 8
```

Последовательность объектов

```
type Pupil = auto class
  Имя: string;
  Возраст: integer;
end;

function Pup(n: string; a: integer) := new Pupil(n,a);

begin
  var s := Seq(Pup('Умнова',16),Pup('Иванов',23),
    Pup('Попова',17),Pup('Козлов',24));
  s.Where(x -> x.Возраст >= 18).Println;
  Println('Сортировка по фамилии:');
  s.OrderBy(x -> x.Имя).Println;
  Println('Сортировка по возрасту по убыванию:');
  s.OrderByDescending (x -> x.Возраст).Println;
end.
```

```
(Иванов,23) (Козлов,24)
Сортировка по фамилии:
(Иванов,23) (Козлов,24) (Попова,17) (Умнова,16)
Сортировка по возрасту по убыванию:
(Козлов,24) (Иванов,23) (Попова,17) (Умнова,16)
```

Последовательности и массивы

- Массив является последовательностью. Это значит, что массив можно присвоить переменной типа «последовательность», а также, что к массивам можно применять все методы, которые имеются у последовательностей
- Метод последовательности, применённый к массиву (например, Select или Where), возвращает последовательность (не массив!)
- Последовательность не является массивом, для ее явного преобразования к массиву надо использовать метод ToArray

Массив как последовательность

```
begin
  var a := Arr(7,5,9,3,6,4,1,10);
  var s: sequence of integer := a; // OK
  a.Println;
  Println('Минимум =',a.Min);
  Println('Максимум =',a.Max);
  Println('Сумма =',a.Sum);
  Println('Сумма квадратов =',a.Sum(x->x*x));
  Println('Первый чётный =',a.First(x->x mod 2 = 0));
end.
```

```
7 5 9 3 6 4 1 10
Минимум = 1
Максимум = 10
Сумма = 45
Сумма квадратов = 317
Первый чётный = 6
```

Метод ToArray

```
begin
  var a := Arr(7,5,9,3,6,4,1,10);
// a.Select возвращает последовательность
// Чтобы присвоить результат массиву, надо  преобразовать
// последовательность в массив методом ToArray
  a := a.Select(x->x*x).ToArray;
  a.Println;
// Вновь необходимо вызвать метод ToArray
  a := a.Where(x -> x mod 2 = 0).ToArray;
  a.Println;
end.
```

```
49 25 81 9 36 16 1 100
36 16 100
```

Оператор yield

- Чтобы написать функцию-генератор последовательности, используется оператор **yield**
- Оператор **yield** можно использовать только в функциях, возвращающих **sequence of** T
- Запрос одного значения у функции-генератора приводит к её выполнению до ближайшего вызова оператора **yield** x, после чего работа функции приостанавливается, запоминаются значения всех локальных переменных и функция возвращает значение x. Последующие запросы значения приведут к продолжению работы функции с приостановленного места.

Арифметическая прогрессия

```
// Функция-генератор арифметической прогрессии
function Arithm(n,a,d: integer):
  sequence of integer;
begin
  loop n do
  begin
    yield a;
    a += d;
  end;
end;

begin
  Arithm(10,1,3).Println
end.
```

```
1 4 7 10 13 16 19 22 25 28
```

Фильтрация элементов

```
// Функция-генератор, фильтрующая элементы
// последовательности по условию, задаваемому фильтром
function MyWhere<T>(s: sequence of T;
  filter: T->boolean): sequence of T;
begin
  foreach var x in s do
    if filter(x) then
      yield x;
end;

begin
  var s := Range(1,9);
  MyWhere(s, x->x mod 2 <> 0).Println
end.
```

```
1 3 5 7 9
```

Оператор yield sequence

- Оператор **yield sequence** позволяет вернуть подпоследовательность в функции-генераторе последовательностей
- Оператор **yield sequence** по существу заменяется на цикл **foreach** по внутренней последовательности с возвратом всех значений с помощью **yield**

Слияние последовательностей

```
// Вначале возвращаются элементы первой
// последовательности, затем второй
function Seq12: sequence of integer;
begin
  yield sequence Seq(1,3,5);
  yield sequence Seq(2,4,6);
end;

begin
  Seq12.Println
end.
```

```
1 3 5 2 4 6
```

Инфиксный обход бинарного дерева

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

function CNode<T>(x: T; l: Node<T> := nil;
  r: Node<T> := nil): Node<T> := new Node<T>(x,l,r);

function Infix<T>(root: Node<T>): sequence of T;
begin
  if root = nil then exit;
  yield sequence Infix(root.left);
  yield root.data;
  yield sequence Infix(root.right);
end;

begin
  var root := CNode(1,CNode(2,CNode(3),CNode(4)),CNode(5));
  Infix(root).Print;
end.
```

```
3 2 4 1 5
```

Бесконечные последовательности

- Последовательность может быть бесконечной. Это возможно в силу ленивости последовательностей
- К бесконечной последовательности нельзя применять методы, которые проходятся по всей последовательности, во избежание зацикливания
- Чаще всего бесконечная последовательность обрезается до конечной применением метода Take(n)
- Имеется ряд стандартных методов, генерирующих бесконечные последовательности: Cycle, Iterate, Step (см. их использование в приводимом ниже примере)

Генераторы бесконечных последовательностей

```
// Генератор бесконечной последовательности значений n
function Inf(n: integer): sequence of integer;
begin
  while True do
    yield n;
end;

begin
// В q хранится бесконечная последовательность
  var q := Inf(5);
  foreach var x in q.Take(10) do
    Print(x);
  Println;
// В q1 хранится конечная последовательность!
  var q1 := q.Take(3);
  q1.Println;
end.
```

```
5 5 5 5 5 5 5 5 5 5
5 5 5
```

Методы с бесконечными последовательностями

```
begin
// Cycle повторяет последовательность бесконечное число раз
  Seq(1,2,3).Cycle.Take(15).Println;
// a.Step(d) генерирует бесконечную арифметическую
// прогрессию от a с шагом h
  2.0.Step(0.5).Take(10).Println;
// a.Iterate(x->f(x)) генерирует бесконечную рекуррентную
// последовательность с первым элементом, равным a,
// и рекуррентным соотношением x_{n+1} = f(x_n)
  1.0.Iterate(x->x/2).Take(7).Println;
end.
```

```
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
2 2.5 3 3.5 4 4.5 5 5.5 6 6.5
1 0.5 0.25 0.125 0.0625 0.03125 0.015625
```

Самое важное

- Последовательности – важнейший тип данных PascalABC.NET. Они используются даже в простейших программах для начинающих. Последовательности являются обобщением массивов, списков, множеств и описываются типом **sequence of** T
- Цикл **foreach** специально вводился в язык как цикл по последовательности. Поскольку массивы, списки, множества и словари являются последовательностями, по ним также возможен цикл **foreach**
- Последовательность может не хранить свои элементы в памяти, а задаваться алгоритмом получения элементов. При использовании цикла **foreach** в каждый момент времени требуется только одна ячейка – для текущего элемента последовательности
- Последовательности могут быть бесконечными. Чтобы работать с ними, необходимы методы, которые берут часть элементов такой последовательности: например sq.Take(n)
- Последовательности возвращаются специальными функциями – **генераторами последовательностей**. В таких функциях используется оператор **yield**, генерирующий текущий элемент. Переменную Result в этих функциях использовать запрещено
- Для последовательностей существует огромное количество методов. Все они доступны также и для разновидностей последовательностей: массивов, списков, множеств и словарей
- Каждый метод последовательности представляет собой простейший алгоритм. В качестве параметров таких алгоритмов повсеместно используются лямбды – они позволяют передавать в методы действия (процедуры) и преобразования (функции).
- Методы последовательностей можно **комбинировать** для получения желаемого результата, используя точечную нотацию: **sq.Where(x -> x>3).OrderBy(x -> x).Select(x -> x*x).Println**. Подобная запись понятна, состоит из простых сменных модулей, просто модифицируется для решения родственных задач. Кроме того, алгоритм получения последовательности sq **отделён** от алгоритма её обработки