Оператор yield в языке PascalABC.NET и его использование в курсе «Основы программирования»

Михалкович С.С., мехмат ЮФУ

Конференция «Современные информационные технологии: тенденции и перспективы развития 2017», Ростов-на-Дону

О курсе «Основы программирования»

- Курс читается студентам 1 курса направления «Фундаментальная информатика и информационные технологии» более 10 лет
- Основной язык программирования – PascalABC.NET
- PascalABC.NET включает все конструкции современных языков программирования: классы, интерфейсы, исключения, анонимные функции, обобщения, кортежи, срезы.
- Все эти конструкции активно используются в курсе с целью упрощения изложения
- Синтаксис ряда конструкций PascalABC.NET проще аналогичных в C#
- Возможности стандартной библиотеки PascalABC.NET ориентированы в первую очередь на академическое программирование и обучение

Последовательности в PascalABC.NET

- В 2015 году в PascalABC.NET был введен тип последовательности **sequence of** T
- Последовательности являются обобщениями массивов, списков, двусвязных списков, множеств
- Элементы последовательностей можно перебирать последовательно (на чтение) и выполнять над ними какие-то действия
- Основной цикл по последовательности – foreach
- Последовательности имеют множество методов расширения

Цикл foreach по последовательности

```
begin
  var a := Seq(1,3,4,7,2,15,4,8);
  foreach var x in a do
    Print(x)
end.
```

```
1 3 4 7 2 15 4 8
```

Вызов метода последовательности

```
begin
  var a := Seq(1,3,4,7,2,15,4,8);
  a.Println;
  Println(a.Max)
end.
```

```
1 3 4 7 2 15 4 8
15
```

Стандартные генераторы последовательностей

- Последовательность в общем случае не хранит значения
- Последовательность задаёт **алгоритм получения значений** и хранит исходные данные для этого алгоритма
- Имеется ряд стандартных генераторов последовательностей

Методы SeqGen, SeqRandomInteger

```
begin
  var a := SeqGen(10,1,x->x+2);
  a.Println;
  a := SeqGen(10,1,x->x*2);
  a.Println;
  a := SeqRandomInteger(10);
  a.Println;
end.
```

```
1 3 5 7 9 11 13 15 17 19
1 2 4 8 16 32 64 128 256 512
95 12 4 68 3 23 34 56 9 51
```

Метод Partition

```
begin
  var xx := Partition(1,2,10);
  foreach var x in xx do
    Println(x,x*x);
end.
```

```
1 1
1.1 1.21
1.2 1.44
1.3 1.69
1.4 1.96
1.5 2.25
1.6 2.56
1.7 2.89
1.8 3.24
1.9 3.61
2 4
```

Пользовательские генераторы последовательностей

- Для создания пользовательского генератора последовательности служит оператор yield

Генератор квадратов

```
function Squares(n:integer): sequence of integer;
begin
  for var i:=1 to n do
    yield i*i;
end;

begin
  Squares(10).Println
end.
```

```
1 4 9 16 25 36 49 64 81 100
```

Числа Фибоначчи

```
function Fib(n: integer): sequence of integer;
begin
  (var a, var b) := (1,1);
  yield a;
  for var i:=2 to n do
  begin
    (a,b) := (b,a+b);
    yield a;
  end;
end;

begin
  Fib(10).Println
end.
```

```
1 1 2 3 5 8 13 21 34 55
```

- Функция с yield сохраняет значения всех локальных переменных между вызовами, после чего продолжают работу с места последнего останова
- Реализуется с помощью конечного автомата

Реализация стандартных генераторов

- Все библиотечные функции стандартных генераторов были переписаны с использованием yield

Метод SeqGen

```
function SeqGen<T>(count: integer; x: T;
  f: T -> T): sequence of T;
begin
  for var i:=1 to n do
  begin
    yield x;
    x := f(x);
  end;
end;

begin
  var a := SeqGen(10,1,x->x+2);
  a.Println;
  SeqGen(10,1,x->x*2).Println;
end.
```

```
1 3 5 7 9 11 13 15 17 19
1 2 4 8 16 32 64 128 256 512
```

Метод RandomInfSeq

```
function RandomInfSeq: sequence of integer;
begin
  while True do
    yield Random(100);
end;

begin
  RandomInfSeq.Take(10).Println;
end.
```

```
26 6 28 64 66 29 90 34 9 89
```

Отделение получения данных от их обработки

- Важнейший методический эффект от введения **yield** и генераторов последовательностей – возможность отделить алгоритм получения данных от обработки этих данных

Способ отцов и дедов

```
begin
  var sum := 0;
  for var i:=1 to 10 do
  begin
    var x := ReadInteger;
    sum += x;
  end;
  Print(sum);
end.
```

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

Современный способ

```
begin
  var seq := ReadSeqInteger(10);

  var sum := 0;
  foreach var x in seq do
    sum += x;

  Print(sum);
end.
```

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

Отделение получения данных от их обработки

- Важнейший методический эффект от введения **yield** и генераторов последовательностей – возможность отделить алгоритм получения данных от обработки этих данных

Способ отцов и дедов

```
begin
  var sum := 0;
  for var i:=1 to 10 do
  begin
    var x := ReadInteger;
    sum += x;
  end;
  Print(sum);
end.
```

Чтение и обработка данных – всё в кучу

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

Современный способ

```
begin
  var seq := ReadSeqInteger(10);

  var sum := 0;
  foreach var x in seq do
    sum += x;

  Print(sum);
end.
```

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

Отделение получения данных от их обработки

- Важнейший методический эффект от введения **yield** и генераторов последовательностей – возможность отделить алгоритм получения данных от обработки этих данных

Способ отцов и дедов

```
begin
  var sum := 0;
  for var i:=1 to 10 do
  begin
    var x := ReadInteger;
    sum += x;
  end;
  Print(sum);
end.
```

Чтение и обработка данных – всё в кучу

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

Современный способ

```
begin
  var seq := ReadSeqInteger(10);

  var sum := 0;
  foreach var x in seq do
    sum += x;

  Print(sum);
end.
```

Чтение данных

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

Отделение получения данных от их обработки

- Важнейший методический эффект от введения **yield** и генераторов последовательностей – возможность отделить алгоритм получения данных от обработки этих данных

Способ отцов и дедов

```
begin
  var sum := 0;
  for var i:=1 to 10 do
  begin
    var x := ReadInteger;
    sum += x;
  end;
  Print(sum);
end.
```

Чтение и обработка данных – всё в кучу

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

Современный способ

```
begin
  var seq := ReadSeqInteger(10);

  var sum := 0;
  foreach var x in seq do
    sum += x;

  Print(sum);
end.
```

Чтение данных

Обработка данных

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

Получение значений из файла

Функция Elements открывает файл, получает из него значения, возвращает их в виде последовательности, после чего закрывает файл

Функция, возвращающая последовательность символов из файла

```
function Elements<T>(name: string): sequence of T;
begin
  var f := OpenFile&<T>(name);
  while not f.Eof do
    yield f.ReadElement;
  f.Close;
end;

begin
  Elements&<char>('a1.pas').Print;
end.
```


```
function Elements<T>(name: string): sequence of T;
begin
  var f := OpenFile&<T>(name);
  while not f.Eof do
    yield f.ReadElement;
  f.Close;
end;

begin
  Elements&<char>('a1.pas').Print;
end.
```

Обход бинарного дерева

- Обход бинарного дерева в инфиксном порядке
- Обработка значений отделена от обхода дерева и получения значений: обход делается в функции InfixPrintTree, а вычисление суммы полученных из дерева данных – в основной программе

Функция InfixPrintTree

```
function InfixTraverseTree<T>(root: Node<T>): sequence of T;
begin
  if root = nil then exit;
  foreach var x in InfixTraverseTree(root.left) do
    yield x;
  yield root.data;
  foreach var x in InfixTraverseTree(root.right) do
    yield x;
end;

begin
  var root := CreateTree(20);
  Println(InfixTraverseTree(root).Sum);
end.
```

```
124
```

Оператор yield sequence

- В функции InfixPrintTree в циклах foreach возвращается подпоследовательность
- Для возврата подпоследовательности имеется специальный оператор **yield sequence**:

Функция InfixPrintTree – использование yield sequence

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

begin
  var root := CreateTree(20);
  Println(InfixTraverseTree(root).Sum);
end.
```

```
124
```

Выводы

Использование оператора yield в курсе «Основы программирования» позволяет:

- отделять получение данных от их обработки
- формировать более чистый стиль программирования
- компактнее излагать алгоритмы и структуры данных, не отвлекаясь на технические детали