# ОПЕРАТОР YIELD В ЯЗЫКЕ PASCALABC.NET И ЕГО ИСПОЛЬЗОВАНИЕ В КУРСЕ «ОСНОВЫ ПРОГРАММИРОВАНИЯ»

**Михалкович С.С.**
*ФГОАУ ВПО «Южный федеральный университет», Институт математики, механики и компьютерных наук им. И.И. Воровича*
*E-mail: miks@sfedu.ru*

Курс «Основы программирования» для студентов 1 курса направления «Фундаментальная информатика и информационные технологии» читается более 10 лет. Курс основан на системе программирования PascalABC.NET [1–3], которая постоянно развивается, пополняясь новыми конструкциями и библиотечными средствами. Расширение языка PascalABC.NET и его библиотек подчинено двум задачам: удобство преподавания современного языка программирования и введение языковых средств, появляющихся в других современных языках программирования. Одним из таких современных средств является работа с последовательностями, представляющими собой обобщение массива, списка, двусвязного списка или множества.

Последовательность в PascalABC.NET имеет тип **sequence of** T, ряд встроенных методов (унаследованных от платформы .NET и собственных расширений), а также возможность использовать для перебора элементов последовательностей цикл **foreach**. Не останавливаясь подробно на многоплановой теме использования последовательностей в указанном курсе, рассмотрим в настоящей работе вопрос, связанный с генераторами последовательностей, т.е. функциями, возвращающими **sequence of** T. Следует отметить, что в отличие от массива, списка и т.д. последовательность вообще говоря не хранится в памяти, а представляет собой алгоритм получения элементов. Данный алгоритм легко задавать с помощью высокоуровневого оператора **yield**, появившегося сравнительно недавно вязыке C# и ряде других современных языков, а в PascalABC.NET – в 2016 году. Рассмотрим пример:

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

Данная функция возвращает последовательность квадратов элементов от 1 до n, которая может использоваться в дальнейших вычислениях. В частности, можно перебрать все элементы x этой последовательности с помощью цикла **foreach** и выполнить с ними какое-то действие:

```
foreach var x in Squares(10) do
  Действие (x)
```

Кроме того, можно вызвать любой метод последовательности, например:

```
Squares(10).Println.
```

Функции с оператором **yield** интересны тем, что всякий раз возвращают новое значение последовательности с помощью **yield**, сохраняя значения всех локальных переменных между вызовами, после чего продолжают работу с места последнего останова. Введение оператора **yield** в язык PascalABC.NET позволило реализовать многие стандартные функции существенно проще, ввести ряд новых стандартных функций, а также упростить изложение и реализацию ряда алгоритмов курса.

К числу стандартных функций, использующих в реализации **yield**, относится например генератор последовательности на основе рекуррентного соотношения:

```
var q := SeqGen(10,1,x->x*2);
```

В данном примере функция SeqGen имеет следующую реализацию:

```
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;
```

После введения оператора **yield** появилась возможность генерировать бесконечные последовательности, например:

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

Использовать такую последовательность можно только, обрезая её в какой-то момент до конечной:

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

Наконец, наиболее интересно использование оператора **yield** в алгоритмах обхода структур данных. Рассмотрим один из самых показательных примеров: обход бинарного дерева в инфиксном порядке:

```
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;
```

Важно отметить, что благодаря использованию оператора **yield** мы не выполняем внутри функции никаких действий с возвращаемыми в определенном порядке элементами дерева: этим занимается вызывающий алгоритм. Таким образом, использование **yield** делает учебный код более чистым, отделяя обход данных от собственно действий над данными. Отметим также, что фактически каждый цикл здесь возвращает подпоследовательность. Именно с этой целью в PascalABC.NET введен еще один оператор – **yield sequence**, используя который, можно записать последнюю функцию максимально компактно:

```
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;
```

*Литература*


1. *Бондарев И.В., Белякова Ю.В., Михалкович С.С.* Система программирования PascalABC.NET: 10 лет развития / Труды XX Научно-методической конференции «Современные информационные технологии: тенденции и перспективы развития». Ростов н/Д: Изд-во ЮФУ, 2013. С. 69–71.
2. *Бондарев И.В., Михалкович С.С.* Система программирования PascalABC.NET: новые возможности 2015-16 гг. / Труды XXIII Научно-методической конференции «Современные информационные технологии: тенденции и перспективы развития». Ростов н/Д: Изд-во ЮФУ, 2016. С. 69–71.
3. *Абрамян М.Э., Михалкович С.С.* Веб-среда разработки и обучения // Открытые системы. СУБД. 2012, № 10. С. 56–59.