Lection #9

Lesson # 16

Basics of Programming.

Sequences

Course Basics of Programming Semester 1, FIIT

Mayer Svetlana Fyodorovna

Sequences

Sequence is a set of elements that can be handled one by another

Sequence:

1. Doesn't store all its elements in memory
2. Seq is an algorithm for getting values one by one
3. We only have one element in memory at a time.
4. The main loop – **foreach**

Type of sequence

Sequence has the type **sequence of** T.

Arrays and lists are kinds of sequences.

It means that:

- We can assign an array or a list to the sequence
- Sequence methods are also ones for arrays and lists (and another kinds of sequences)

```
begin
  var s: sequence of integer;
  var a: array of integer := Arr(1,2,3);
  s := a;
  var l: List<integer> := new List<integer>(a);
  s := l;
  s.Println;
end.
```

Println is a sequence method !

Sequence input and further processing

- ReadSeqInteger
- ReadSeqReal

accumulate the sequence, the q variable stores it

```
begin
var q:=ReadSeqReal(10);
var s:=0.0;
foreach var x in q do
  s+=x;
print(s)
end.
```

The substitution principle

The substitution principle. We can use any kind of object instead of the basic object without loss of operability

```
function SumSquares(s: sequence of integer): integer;
begin
  Result := 0;
  foreach var x in s do
    Result += x*x;
end;

begin
  SumSquares(Arr(1,2,3)).Print; // 1*1 + 2*2 + 3*3
  SumSquares(Lst(2,3,4)).Print;
end.
```

```
14 29
```

Standard sequence generators

```
Seq(1,3,5,7,9) .Print; // 1 3 5 7 9

print(Range(1,10)); // [1 2 3 4 5 6 7 8 9 10]

print(Range(1,10,2)); // [1 3 5 7 9]
print(Range('a','k')); // [a,b,c,d,e,f,g,h,i,j,k]  sequence of char

var q:=SeqRandomInteger(5,10,20);
print(q); // [12,18,16,14,16]

var q:=SeqRandomReal(3, 1.0, 5.0);
print(q); // [4.98996168374548,2.22339218166815,2.81110574389394]

print(SeqFill(10,55)); // 55 55 55 55 55 55 55 55 55 55

print(Partition(0.0, 6.0, 4));
// divide equally into 4 parts [0, 1.5, 3, 4.5, 6]
```

In all cases the elements of a sequence are not stored in a memory at once. They are calculated as needed

Tasks

- Task 1,2

Sequence methods we know

```
s.Sum
s.Average
s.Min
s.Max
s.Count
s.Count(condition)
s.All(condition)
s.Any(condition)
s.Print
s.Println
s.PrintLines
```

Example

To do: Create a function that searches in a sequence for the number of maximum elements.

```
function findCountMax(a: sequence  of integer):integer;
begin
var k:=a.Max();
  foreach var i in a do
    if i=k then result+=1;
end;
begin
  var c:=Seq(1,5,2,10,1,10);
  c.Println();
  println('number of max elements ',findCountMax(c));
end.
```

```
1 5 2 10 1 10
number of max elements  2
```

Example

To do: Define a sequence using Seq method. Calculate a quantity of entered number within the sequence. You should use count (condition) method.

```
begin
  var c:=Seq(-1,2,3,-5,2,-7,8,2,11);
  c.Println();
  var x:=readinteger('enter x:');
  var n:=c.count(c->c=x);
  println($'there is {x} within the seq {n} times');
end.
```

```
-1 2 3 -5 2 -7 8 2 11
enter x: 2
there is 2 within the seq 3 times
```

Tasks

- Task 3,4,5

Sequence generators with lambda-expressions

```
SeqGen(10,i->i); // 0 1 2 3 4 5 6 7 8 9

SeqGen(10,i->i,1); // 1 2 3 4 5 6 7 8 9 10
```

- SeqGen(count: integer; f: integer -> T): sequence of T;

```
begin
var sq:=SeqGen(5,x->x+1);
sq.println; // 1 2 3 4 5
end.
```

- SeqGen(count: integer; first: T; next: T -> T): sequence of T;

```
begin
var sq:=SeqGen(5, 4, x->x+1);
sq.println; // 4 5 6 7 8
end.
```

In all cases the elements of a sequence are not stored in a memory at once. They are calculated as needed

Sequence generators with lambda-expressions

- SeqGen(count: integer; first, second: T; next: (T,T) -> T): sequence of T;

```
begin
var sq:=SeqGen(5, 1, 3, (x,y)->x+y);
sq.println; // 1 3 4 7 11
end.
```

- SeqWhile(first: T; next: T -> T; pred: T -> boolean): sequence of T;

Condition is added:

```
begin
var sq:=seqWhile(2,x->x*2,x->x<=1024);
sq.println; // 2 4 8 16 32 64 128 256 512 1024
end.
```

- SeqWhile(first,second: T; next: (T,T) -> T; pred: T -> boolean): sequence of T;

In all cases the elements of a sequence are not stored in a memory at once. They are calculated as needed

Example

- **To do:** Create a sequence of N Fibonacci numbers

```
begin
var n:=readInteger('Enter n');
var sq:=SeqGen(n,1,1,(x,y)->x+y);
sq.println();
end.
```

```
Enter n 8
1 1 2 3 5 8 13 21
```

Example

- **To do:** Create a sequence of N numbers generated by the iterative process: $a_1=2,\ a_k=(a_{k-1}+1)*a_{k-1},\ k = 2,3,\ldots$

```
begin
var n:=readInteger('Enter n');
var sq:=SeqGen(n,2,x->(x+1)*x);
sq.println();
end.
```

```
Enter n 9
2 6 42 1806 3263442 -1461943274 -757910022 5287454 1232959906
```

Infinite sequence generators

- Cycle()

Repeating a sequence block

Take is used to restrict

```
Seq(1,2,10,5).Cycle().Take(15).Println; // 1 2 10 5 1 2 10 5 1 2 10 5 1 2 10
```

- Repeat

Infinite sequence of numbers

```
var q:=55.Repeat.Take(10).Println; // 55 55 55 55 55 55 55 55 55 55
```

- Step

Generating an infinite sequence with a step

```
var q:=5.Step(2).Take(10).Println; // 5 7 9 11 13 15 17 19 21 23
```

- Iterate

Generating an infinite sequence using a lambda-function

```
var q:=10.Iterate(x->x-2).Take(10).Println; // 10 8 6 4 2 0 -2 -4 -6 -8
```

Tasks

- Task 6,7,8,9,10,11,12

Q & A