L#7

Basics of Programming.

Procedures and functions

Course Basics of Programming Semester 1, FIIT

Mayer Svetlana Fyodorovna

Filling an array with generated sequence. lambda expressions

```
var a := Arr(1,3,5,7,9);    // [1,3,5,7,9]

var a := ArrFill(5,555);  // [555,555,555,555,555]

var a := Arr(55) * 4 + Arr(77) * 6;  // [55,55,55,55,77,77,77,77,77,77]

var a := ArrGen(10,i->i);    // [0,1,2,3,4,5,6,7,8,9]

var a := ArrGen(10,i->i,1);  // [1,2,3,4,5,6,7,8,9,10]

var a := ArrGen(10,1,x->x+2);  // [1,3,5,7,9,11,13,15,17,19]

var a := ArrGen(10,1,x->x*2);  // [1,2,4,8,16,32,64,128,256,512]

var a := ArrGen(10,1,1,(x,y)->x+y);  // [1,1,2,3,5,8,13,21,34,55]
```

ArrGen

```
ArrGen<T>(count:integer; gen:integer->T):array of T;
returns the array of count elements filled with gen(i) values

ArrGen<T>(count:integer; gen:integer->T; from:integer):array of T;
returns the array of count elements filled with gen(i) values beginning with
i=from

ArrGen<T>(count:integer; first: T; next: T->T):array of T;
returns the array of count elements beginning with first, and with a next
function to move from the previous element to the next

ArrGen<T>(count:integer; first: T; second: T; next:(T,T)->T):array of T;
returns the array of count elements beginning with first and second, and with a
next function of two previous to the next
```

Tasks

- To do: Lesson # 13, Tasks 1, 2, 3, 4

Array input

```
var a:=ReadArrInteger(5);
var a:=ReadArrReal(5);
```

Randomly generated array

```
var a:=new integer[10];
a:=arrRandomInteger(10);
```

Params keyword to pass a varying number of parameters

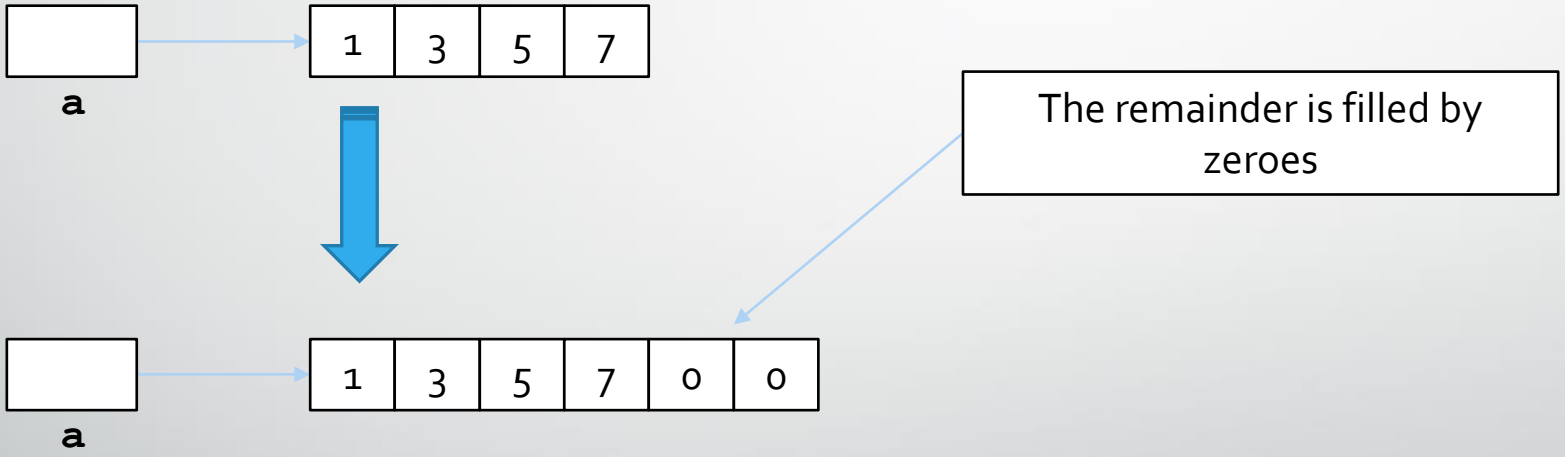
```
function Sum(params a: array of integer): integer;
begin
  Result := 0;
  foreach var x in a do
    Result += x;
end;

begin
  Print(Sum(1,3,5));
  Print(Sum(1,4,10,22,44));
end.
```

! In the case of more than one parameter, params should be the last one in the list

Dynamic array expansion.

SetLength procedure

```
begin
  var a := Arr(1,3,5,7);
  SetLength(a,6);
  Print(a);
end.
```



Standard **SetLength** function allocates the necessary memory to contain the array elements

SetLength procedure

Problem: Fill array **b** with positive numbers of array **a**

```
function MakeArr (params a: array of integer): array of integer;
begin
  var b := new integer[a.length];
  var j := 0;
  for var i := 0 to a.length - 1 do
    if a[i] > 0 then
    begin
      b[j] := a[i];
      j += 1;
    end;
  SetLength(b, j);
  result := b;
end;

begin
  var a := arrRandomInteger(5, -5, 10);
  println('array a', a);
  println('result of function, array b ', MakeArr(a));
end.
```

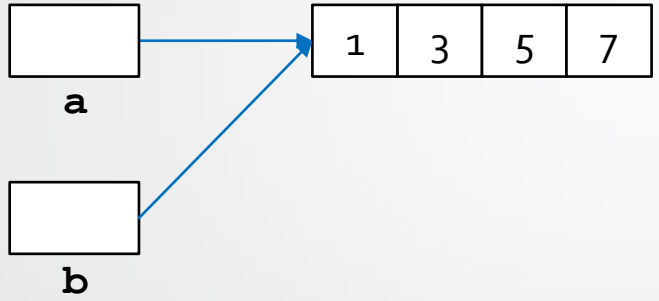
```
array a [-1,9,9,7,-3,0,-2,1,1,-1]
result of function, array b  [9,9,7,1,1]
```

Tasks

- To do: Lesson # 13, Tasks 5, 6, 7

Reassignment:

```
var a: array of integer := (1,3,5,7);
var b:=a; // [1,3,5,7]
```



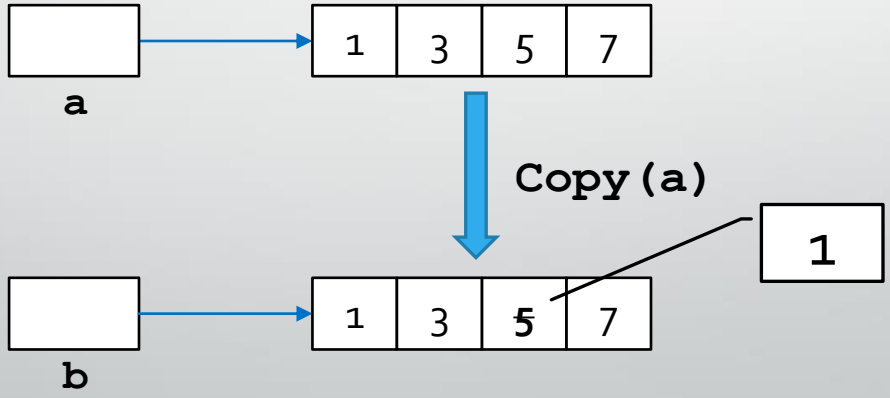
But!
If we now reassign the values of the elements of the array b, then the array a will also change:

```
var a: array of integer := (1,3,5,7);
var b:=a; b[2]:=1;
print(a); //[1,3,1,7]
```

Copy function

To avoid this situation, you need to create array **b** as a copy of array **a**:

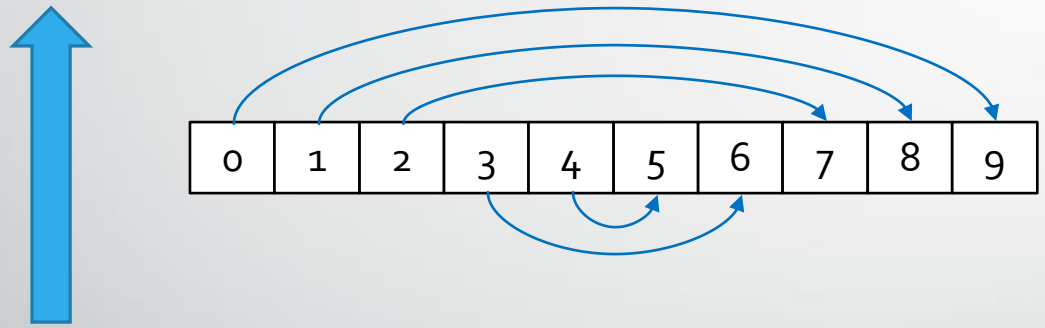
```
var a: array of integer := (1,3,5,7);
var b:=Copy(a);
b[2]:=1;

print(a); //[1,3,5,7]
```



Standard procedures and methods

Reverse of an array

```
procedure Reverse<T>(a: array of T);
begin
  var n := a.Length;
  for var i:=0 to n div 2 - 1 do
    Swap(a[i],a[n-i-1]);
end;
```



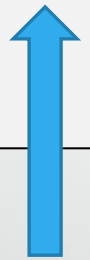

```
begin
var a:=new integer[10];
a:=arrRandomInteger(10);
print(a);// [41,81,84,63,12,26,88,25,36,72]
Reverse(a);
print(a) //[72,36,25,88,26,12,63,84,81,41]
end.
```

A standard **Reverse(a)** procedure has this algorithm. Thus, we don't need to create it in our program, it's possible just to use it.

We can use slices: a := a[::-1]

Linear search algorithm

```
function IndexOf<T>(a: array of T; x: T): integer;
begin
  Result := -1;
  for var i := 0 to a.Length - 1 do
    if a[i] = x then
    begin
      Result := i;
      break;
    end;
end;
```



```
begin
  var a := new integer[10];
  a := arrRandomInteger(5,0,5); //[1,3,5,4,5]
  print(a.IndexOf(3)) // 1
end.
```

There are standard methods **a.IndexOf(x)** and **a.LastIndexOf(x)**

For checking the presence of an element in an array:

1. **a.Contains(x)**
2. x **in** a

Linear search algorithm with some condition

```
function FindIndex<T>(a: array of T; cond: T->boolean): integer;
begin
  Result := -1;
  for var i := 0 to a.High do
    if cond(a[i]) then
    begin
      Result := i;
      break;
    end;
end;
```

```
begin
  var a := new integer[10];
  a := arrRandomInteger(5); //[13,53,15,73,22]
  print(a.FindLastIndex(a->odd(a))) // 3
end.
```

There are standard methods **a.FindIndex(condition)** and **a.FindLastIndex(condition)**

Search algorithm without break

We can create our own function to search **x** in the array. The function returns **-1** if it is not found.

```
function IndexOfW<T>(a: array of T; x: T): integer;
begin
  var n := a.Length;
  var i := 0;
  while (i < n) and (a[i] <> x) do
    i += 1;
  Result := i = n ? -1 : i;
end;

begin
  var a := new integer[10];
  a := arrRandomInteger(10, 0, 10);
  print(a);
  print(indexOfW(a, 2))
end.
```

ternary operator

Transformation of an array elements

Problem: transform elements using a rule **x -> f(x)**

```
procedure Transform<T>(a: array of T; f: T -> T);
begin
  for var i:=0 to a.Length-1 do
    a[i] := f(a[i]);
end;
```

```
begin
  var a := new integer[5];
  a := arrRandomInteger(5); // [4,36,93,36,29]
  a.Transform(a,a -> a mod 2 = 0 ? a-1 : a+1);
  print(a) // [3,35,94,35,30]
end.
```

There is a standard method **a.Transform**(x -> x*x)

For transformation of some elements by condition:

a.Transform(x -> x **mod** 2 = 0 ? x – 1 : x + 1)

Number of elements by condition

```
function Count<T>(a: array of T; cond: T->boolean): integer;
begin
  Result := 0;
  foreach var x in a do
    if cond(x) then
      Result += 1;
end;
```

The standard **a.Count(condition)** procedure

```
begin
  var a := new integer[5];
  a := arrRandomInteger(5); // [18,10,91,47,35]
  print(a);
  print(a.Count(a->odd(a))) // 3
end.
```

Minimal element and its index

Two solutions:

```
function MinElemAndIndex(a: array of real): (real,integer);
begin
  var (min, minind) := (a[0], 0);
  for var i:=1 to a.Length-1 do
    if a[i]<min then
      (min, minind) := (a[i], i);
  Result := (min, minind)
end;
```

There are standard **a.Min**, **a.IndexMin**

```
begin
  var a := new integer[5];
  a := arrRandomInteger(5); // [86,37,41,45,76]
  print(a.Min,a.IndexMin); // 37  1
end.
```

```
function MinElemAndIndex(a: array of real): (real,integer);
begin
  var (min, minind) := (real.MaxValue, 0);
  for var i:=0 to a.Length-1 do
    if a[i]<min then
      (min, minind) := (a[i], i);
  Result := (min, minind)
end;
```

Conditional minimum

```
function MinElemAndIndexCond(a: array of real: cond: real -> boolean):
  (real,integer);
begin
  var (min, minind) := (real.MaxValue, 0);
  for var i:=0 to a.Length-1 do
    if (a[i]<min) and cond(a[i]) then
      (min, minind) := (a[i], i);
  Result := (min, minind)
end;
```

Tasks

- To do: Lesson # 13, Tasks 8

Loop over some indices

```
begin
  var a := new integer[10];
  a := arrGen(10,i->i); //[0,1,2,3,4,5,6,7,8,9]
  foreach var i in a.Indices(x -> x.InRange(10,20)) do
    a[i] += 1;
  print(a); // [0,1,2,3,4,6,7,8,9,10]
end.
```

```
begin
  var a := new integer[10];
  a := arrGen(10,i->i); //[0,1,2,3,4,5,6,7,8,9]
  foreach var i in a.Indices((x,i) -> (i mod 2 = 0) and (x > 0)) do
    a[i] += 1;
  print(a); // [0,1,3,3,5,5,7,7,9,9]
end.
```

Tasks

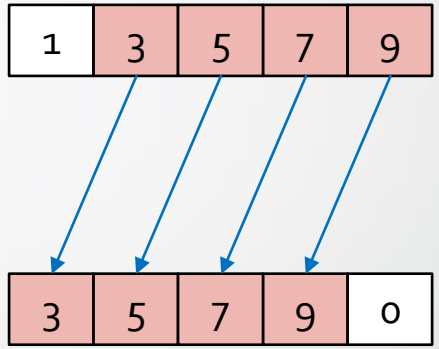
- To do: Lesson # 13, Tasks 9

Elements’ shift

Shift to the left

Problem: Create the procedure to shift the elements to the left

```
procedure ShiftLeft<T>(a: array of T);
begin
  for var i := 0 to a.Length - 2 do
    a[i] := a[i + 1];
  a[a.Length - 1] := default(T);
end;

begin
  var a := new integer[5];
  a := arrRandomInteger(5); // [56,28,33,57,25]
  shiftLeft(a);
  print(a)// [28,33,57,25,0]
end.
```

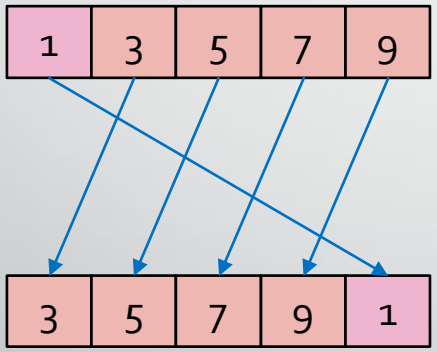


With slices:

a := a[1:] + Arr(0);

Circular shift left

```
procedure CircularShiftLeft<T>(a: array of T);
begin
  var v := a[0];
  for var i:=0 to a.Length-2 do
    a[i] := a[i+1];
  a[a.Length-1] := v;
end;
```

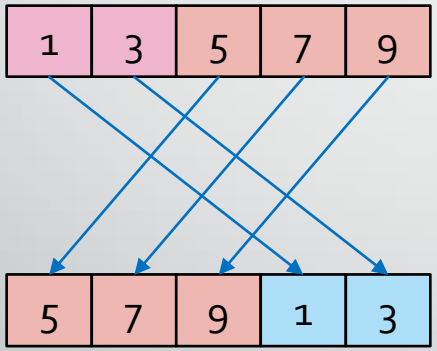


With slices:

a := a[1:] + a[:1];

Circular shift left by k

```
1. loop k do
     CircularShiftLeft(a); // ineffective

2. With second array

3. With partial reverse
```



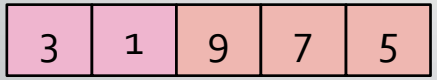
k = 2

Using slices:

a := a[k:] + a[:k];

Circular shift left by k – using partial Reverse

```
var k:=2;
var a := arr(1,3,5,7,9);
Reverse(a,0,k); // [3,1,5,7,9]
Reverse(a,k,a.Length-k); // [3,1,9,7,5]
Reverse(a); // [5,7,9,1,3]
```



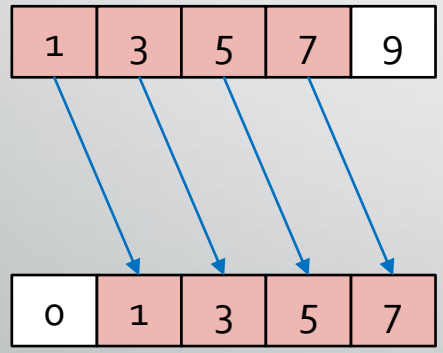
Reversing of 1-st and 2-nd part

Reversing of a whole array

Number of operations $= \frac{3}{2}n + \frac{3}{2}n = 3n$

Shift right

```
procedure ShiftRight<T>(a: array of T);
begin
  for var i:=a.Length-1 downto 1 do
    a[i] := a[i-1];
  a[0] := default(T);
end;
```

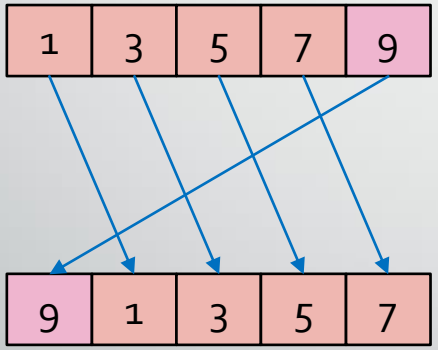


Using slices:

a := Arr(0) + a[:a.Length-1];

Circular shift to the right

```
procedure CircularShiftRight<T>(a: array of T);
begin
  var v := a[a.Length-1];
  for var i:=a.Length-1 downto 1 do
    a[i] := a[i-1];
  a[0] := v;
end;
```



Using slices:

```
var m := a.Length-1;
a := a[m:] + a[:m];
```

Tasks

- To do: Lesson # 13, Tasks 10

Q & A