L#7

Basics of Programming. Procedures and functions

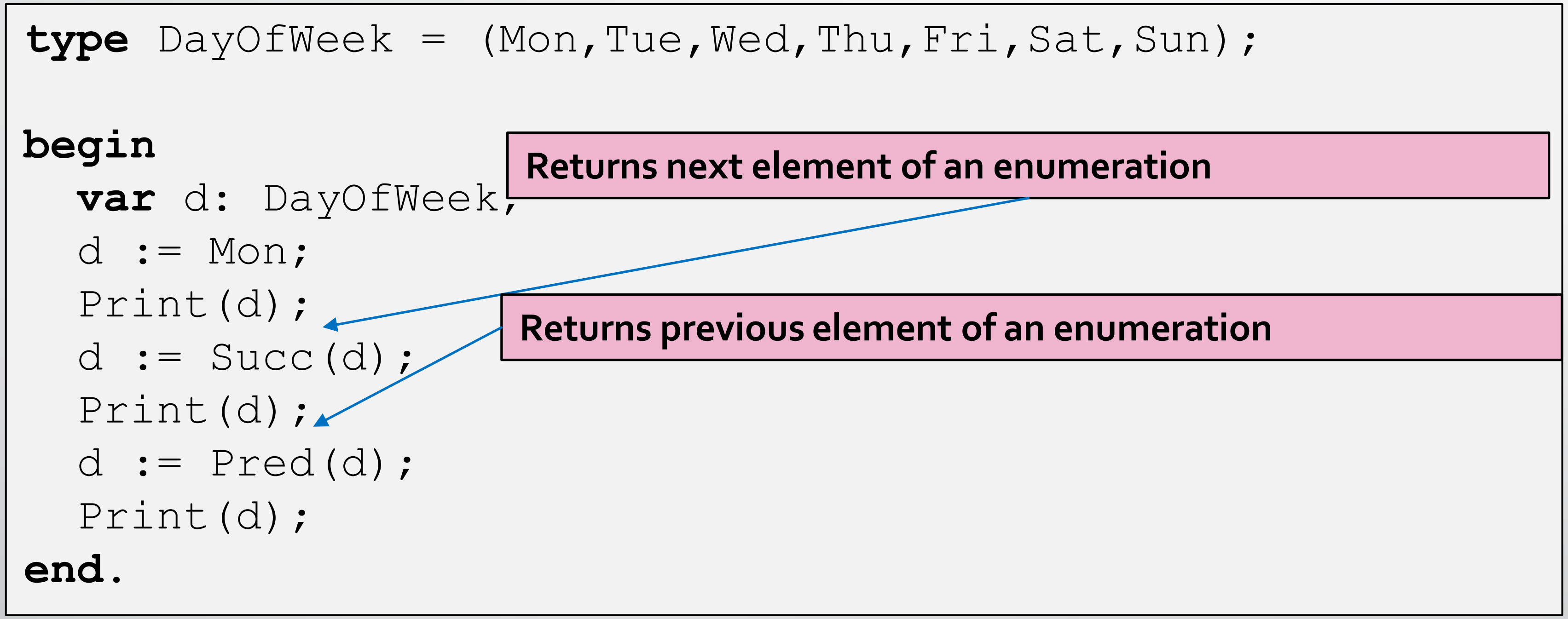
Course Basics of Programming Semester 1, FIIT

Mayer Svetlana Fyodorovna

Enumeration type. Arrays

Enumeration type

User can define their own type by listing a fixed set of possible values :



```
type DayOfWeek = (Mon,Tue,Wed,Thu,Fri,Sat,Sun);

begin
  var d: DayOfWeek;
  d := Mon;
  Print(d);
  d := Succ(d);
  Print(d);
  d := Pred(d);
  Print(d);
end.
```

Returns next element of an enumeration

Returns previous element of an enumeration

Pred & Succ functions do not control the situation outside the bounds

Enumeration type (2)

We can use **for** loop and **case** statement with an Enumeration type:

```
type DayOfWeek = (Mon,Tue,Wed,Thu,Fri,Sat,Sun);

begin
  for var d:= Mon to Fri do
    case d of
      Mon: Print('Monday');
      Tue: Print('Tuesday');
      Wed: Print('Wednesday');
      Thu: Print('Thursday');
      Fri: Print('Friday');
    end;
end.
```

Enumeration type (3)

boolean is an Enumeration type. It is defined as follows:

```
type boolean = (False,True);
```

Enumeration type

It's more correct to use the variable of the created Enumeration type:

```
type Season = (Spring, Summer, Autumn, Winter);
begin
  var d: Season; // the variable of enum type
  d := Season.Spring; // using the typeName.value notation
  print(d); // Spring
  d:=Autumn; // Set an enum variable by name
  if d = Season.Summer then  // false
    …
  if Season.Spring < Season.Autumn then  // true
    …
end.
```

Tasks

- To do: Lesson # 12, Task 1

Arrays

Array definition

Array is a set of elements of the same type sequentially located in memory. Each element has its own index.

We will use so called **dynamic** arrays. This means that they allocate memory dynamically - during program execution

```
begin
  var a: array of integer;
  a := new integer[3];
  a[0] := 5;
  a[1] := 2;
  a[2] := 3;
end.
```

We can combine definition and memory allocation

```
var a := new integer[3] (5,2,3);
```

Output:

```
begin
  var a: array of integer;
  a := new integer[5];
  a[0] := 1;  a[1] := 3; a[2] := 5; a[2] := 7; a[2] := 9;
```

```
Println(a); // [1,3,5,7,9]
a.Println;  // 1 3 5 7 9
a.Println(';'); // 1;3;5;7;9
Println(a[2]); // 5
```

Arrays as a reference types

We say that the array variable references to the memory allocated by the new operation

```
var a := new integer[3];
```

To return memory allocated by array we can assign a special **nil** value to an array variable:

```
var a := nil;
```

Loop on array

To iterate through the array elements by their indexes we can use **for** loop:

```
for var i:=0 to a.Length-1 do
  a[i] += 1;
```

For read-only access (without changing the values) we can use **foreach** loop:

```
foreach var x in a do
  Print(x)
```

Generic type

- In a **generic** type or function/procedure definition, a type parameter is a placeholder for a specific type that a client specifies when they create an instance of the generic type.

<T> means any type

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

Tasks

- To do: Lesson # 12, Tasks 2,3,4,5,6

Multi-file layout

- To make a Multi-file layout first you need to create a unit-file (*.pas*). Unit files consists of:
- **Unit name**:

```
unit DynArrs;
```

- **Interface section:** defines the interface to functions, that is declaration(s) of the function(s) or procedures:

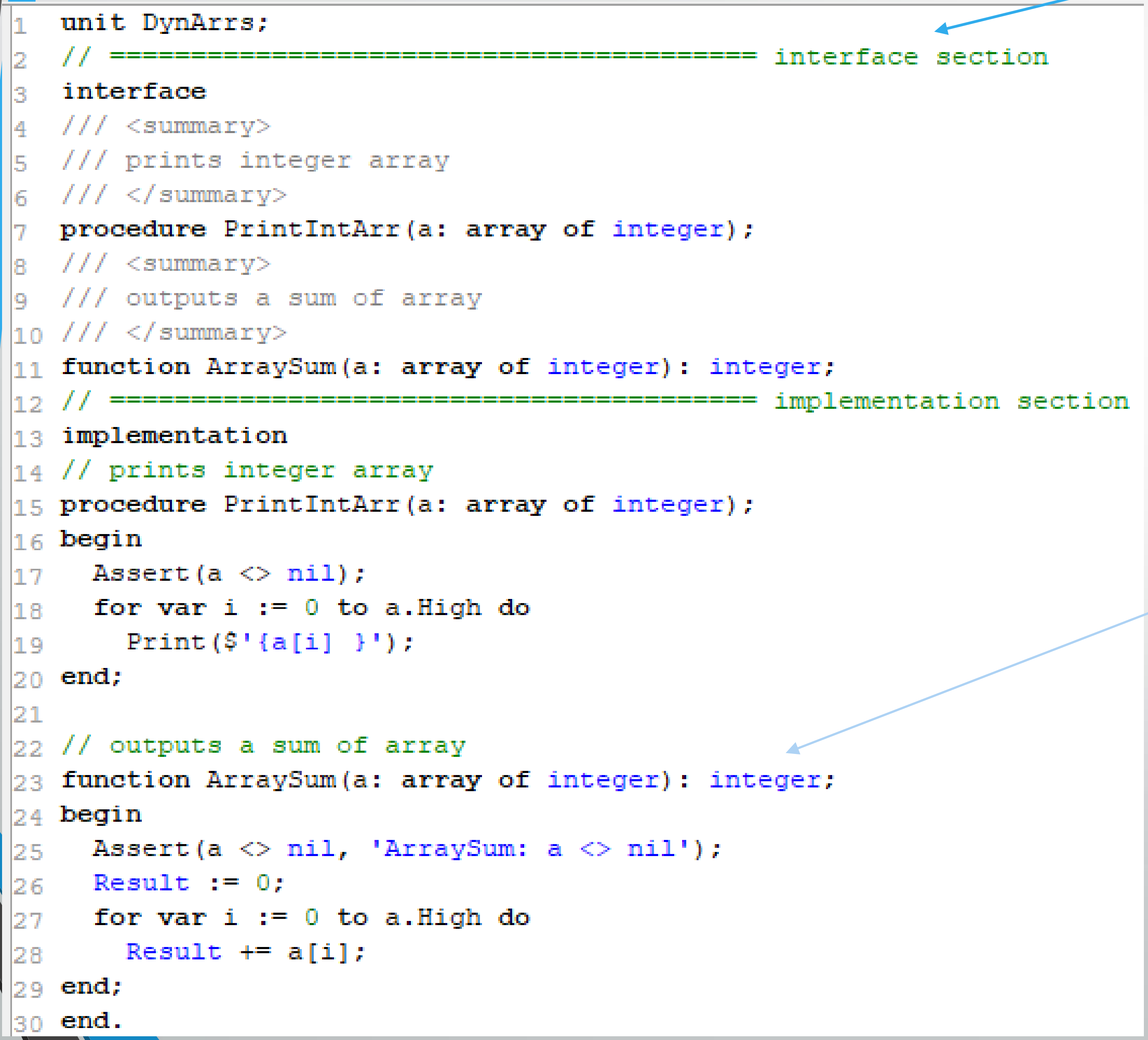
```
// ======================== interface section
Interface
procedure PrintIntArr(a: array of integer);
function ArraySum(a: array of integer): integer;
```

- **Implementation section**: contains all of the logic of the functions or procedures.

```
Implementation
// prints integer array
procedure PrintIntArr(a: array of integer);
begin
  for var i := 0 to a.High do
    Print($'{a[i]} ');
end;
// outputs a sum of array
function ArraySum(a: array of integer): integer;
begin
  // …
end;
```

Multi-file layout

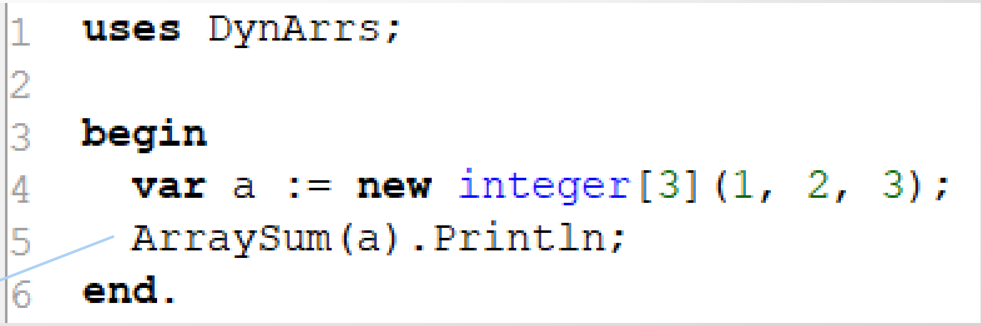
DynArrs.pas



```
unit DynArrs;
// ======================================== interface section
interface
/// <summary>
/// prints integer array
/// </summary>
procedure PrintIntArr(a: array of integer);
/// <summary>
/// outputs a sum of array
/// </summary>
function ArraySum(a: array of integer): integer;
// ======================================== implementation section
implementation
// prints integer array
procedure PrintIntArr(a: array of integer);
begin
  Assert(a <> nil);
  for var i := 0 to a.High do
    Print($'{a[i]} ');
end;

// outputs a sum of array
function ArraySum(a: array of integer): integer;
begin
  Assert(a <> nil, 'ArraySum: a <> nil');
  Result := 0;
  for var i := 0 to a.High do
    Result += a[i];
end;
end.
```

Task-01.pas



```
uses DynArrs;

begin
  var a := new integer[3](1, 2, 3);
  ArraySum(a).Println;
end.
```

Tasks

- To do: Lesson # 12, Tasks 7, 8, 9, 10

Q & A