L#6

Basics of Programming.
Procedures and functions

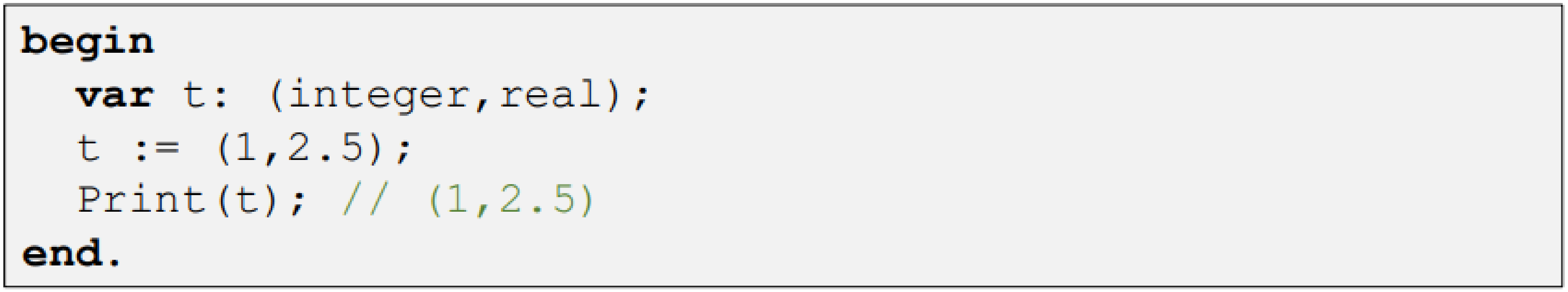
Course Basics of Programming Semester 1, FIIT

Mayer Svetlana Fyodorovna

Function and tuples

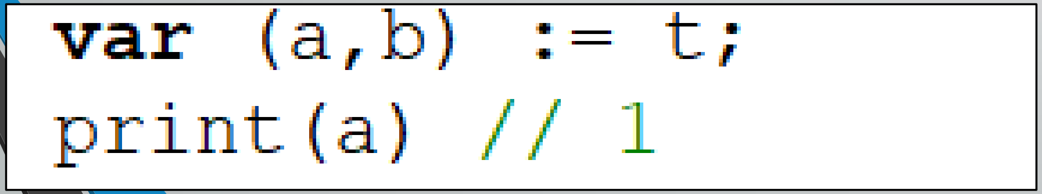
Types of tuples

- Let's assume we have a tuple (1, 2.5). What is the type of this tuple?
- Answer: (integer, real).
- We can define a variable of this type:



```
begin
  var t: (integer,real);
  t := (1,2.5);
  Print(t); // (1,2.5)
end.
```

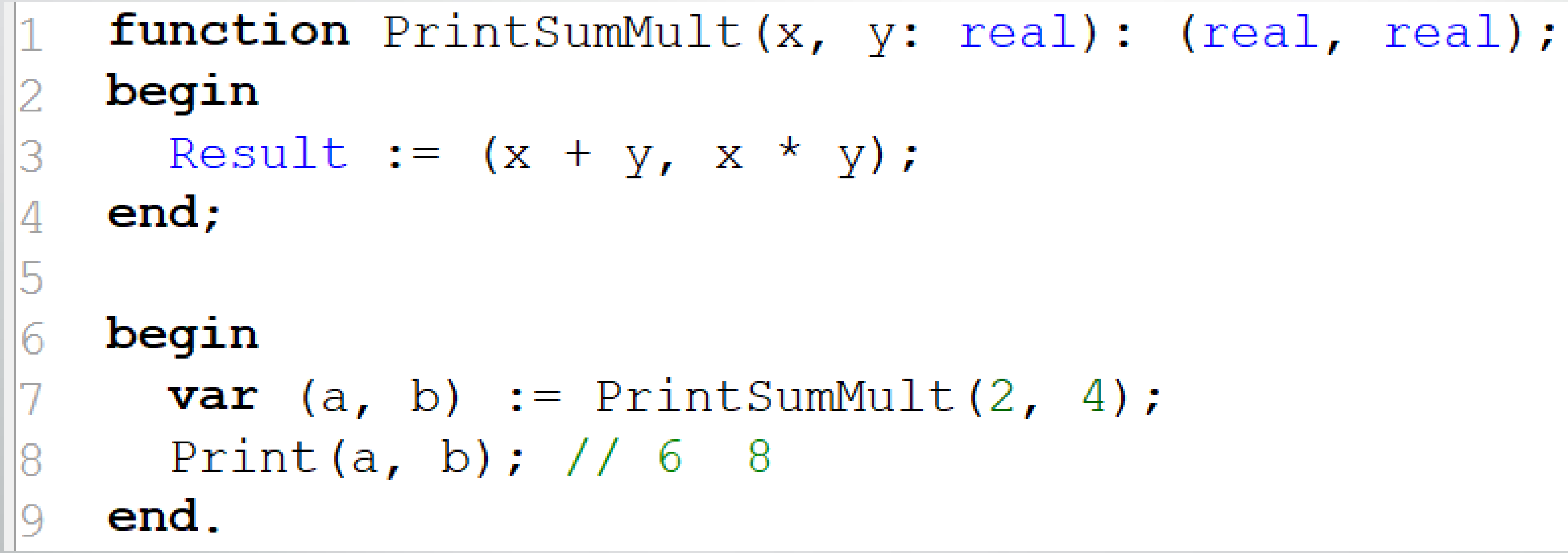
- Question. How to unpack values from a tuple?
- Answer. Using the unpack operation:



```
var (a,b) := t;
print(a) // 1
```

Functions that return tuples

- A tuple can be a function return type:

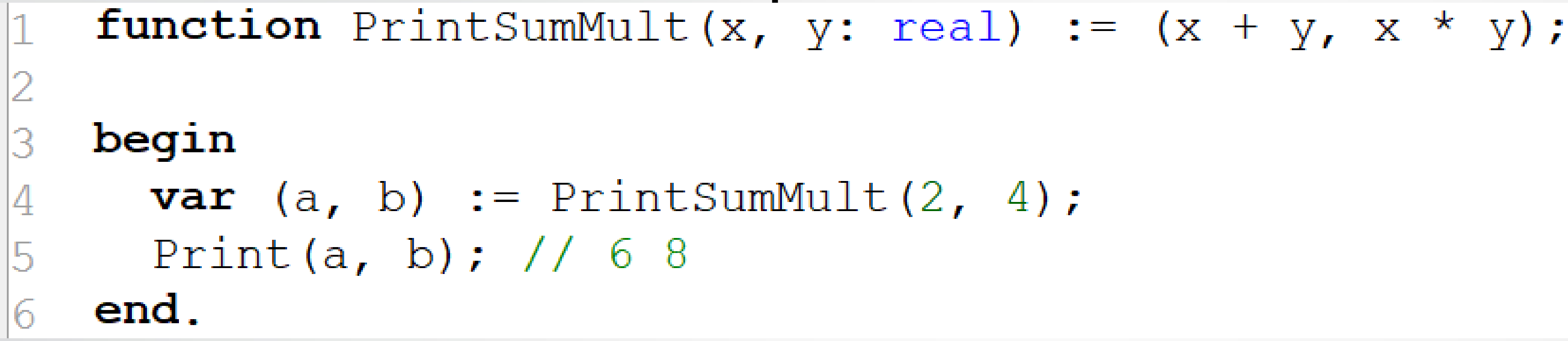


```
function PrintSumMult(x, y: real): (real, real);
begin
  Result := (x + y, x * y);
end;

begin
  var (a, b) := PrintSumMult(2, 4);
  Print(a, b); // 6  8
end.
```

Example

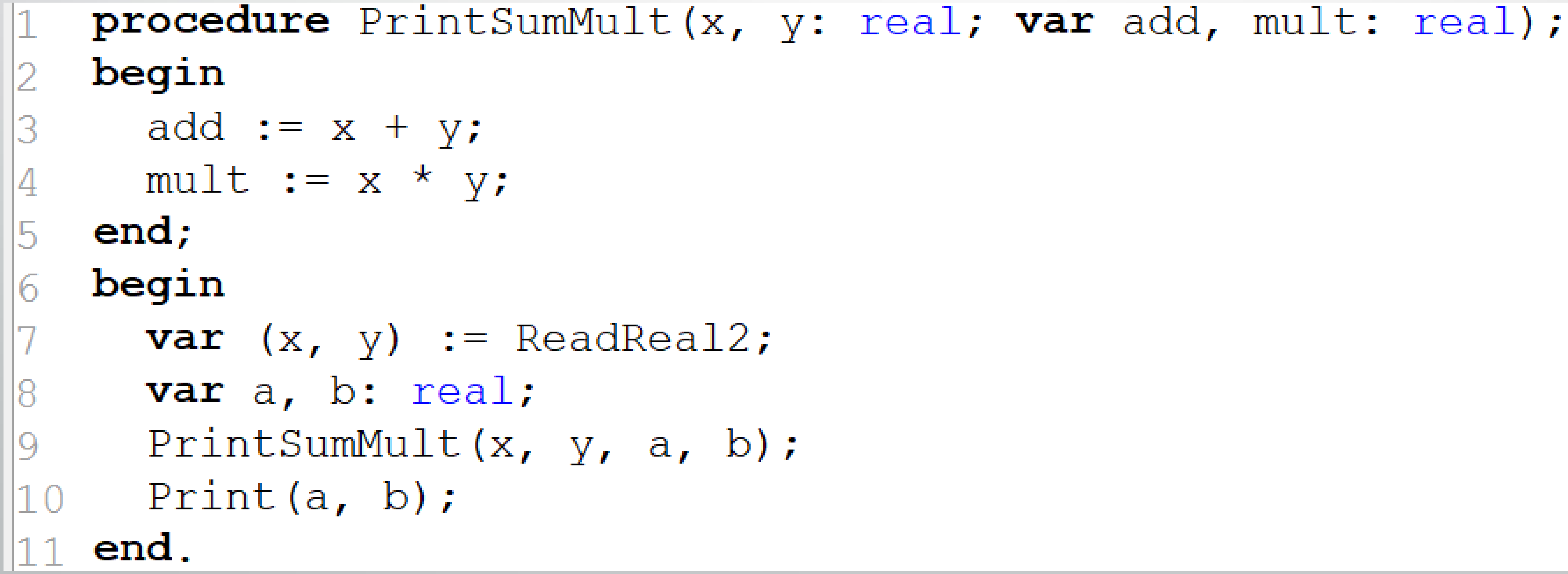
- Short function definitions + tuples as a function return value:



```
function PrintSumMult(x, y: real) := (x + y, x * y);

begin
  var (a, b) := PrintSumMult(2, 4);
  Print(a, b); // 6 8
end.
```

- An equivalent technique with procedures:



```
procedure PrintSumMult(x, y: real; var add, mult: real);
begin
  add := x + y;
  mult := x * y;
end;
begin
  var (x, y) := ReadReal2;
  var a, b: real;
  PrintSumMult(x, y, a, b);
  Print(a, b);
end.
```

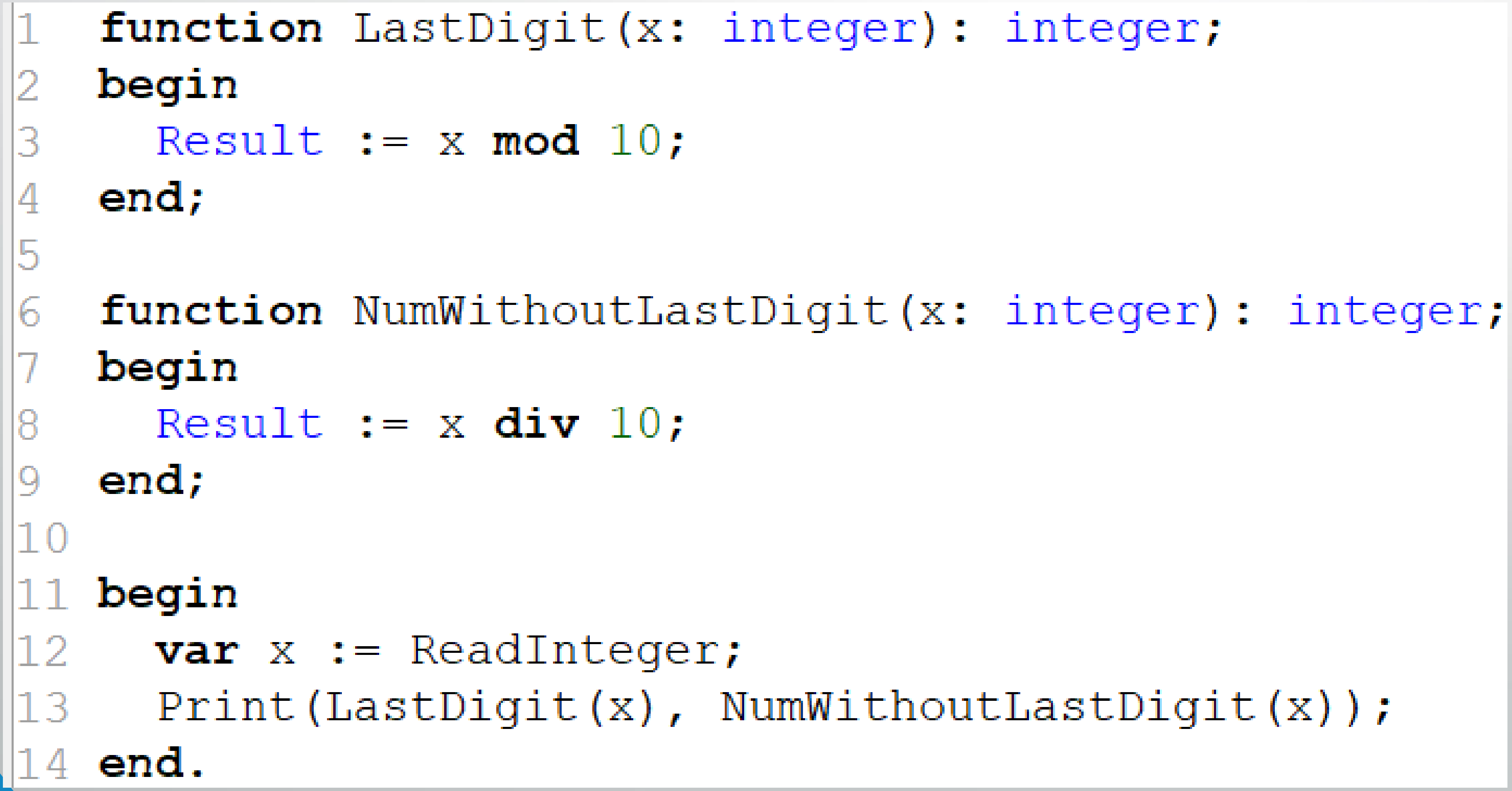
Tasks

- To do: Lesson # 11, Tasks 1, 2

Functions as wrappers for algorithms

Functions as wrappers for algorithms

- We can “wrap” our previous algorithms with functions. To do this, we must define the parameters of the function, their types and the return type of the function.
- Example 1:



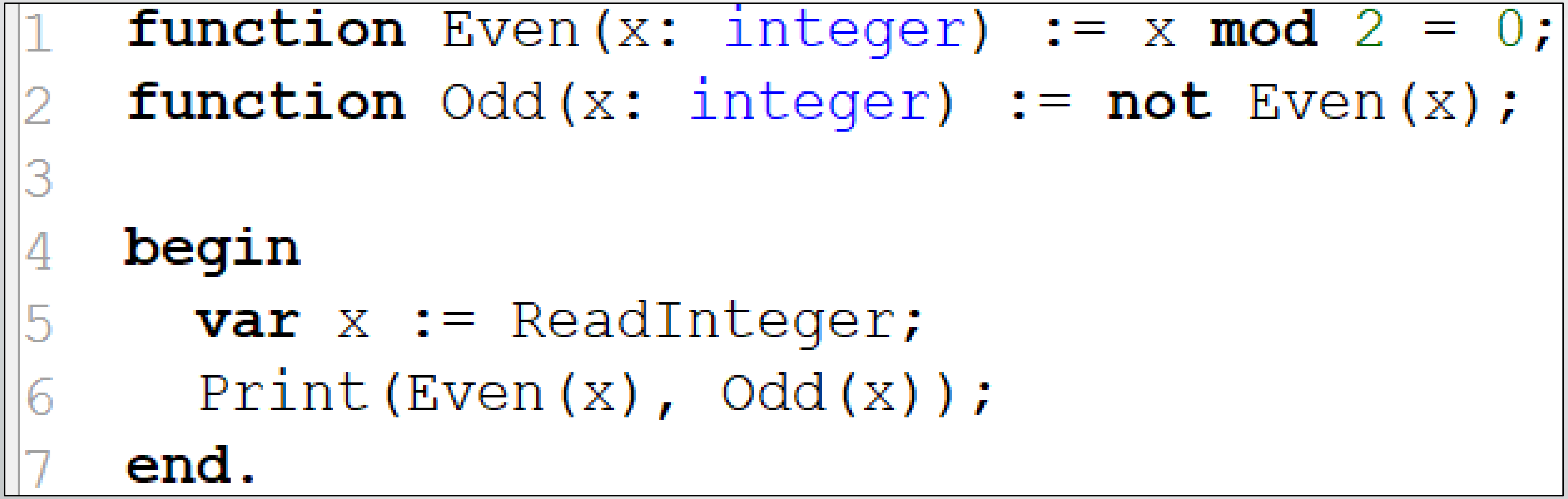
```
function LastDigit(x: integer): integer;
begin
  Result := x mod 10;
end;

function NumWithoutLastDigit(x: integer): integer;
begin
  Result := x div 10;
end;

begin
  var x := ReadInteger;
  Print(LastDigit(x), NumWithoutLastDigit(x));
end.
```

Functions as wrappers for algorithms

- Example 2:

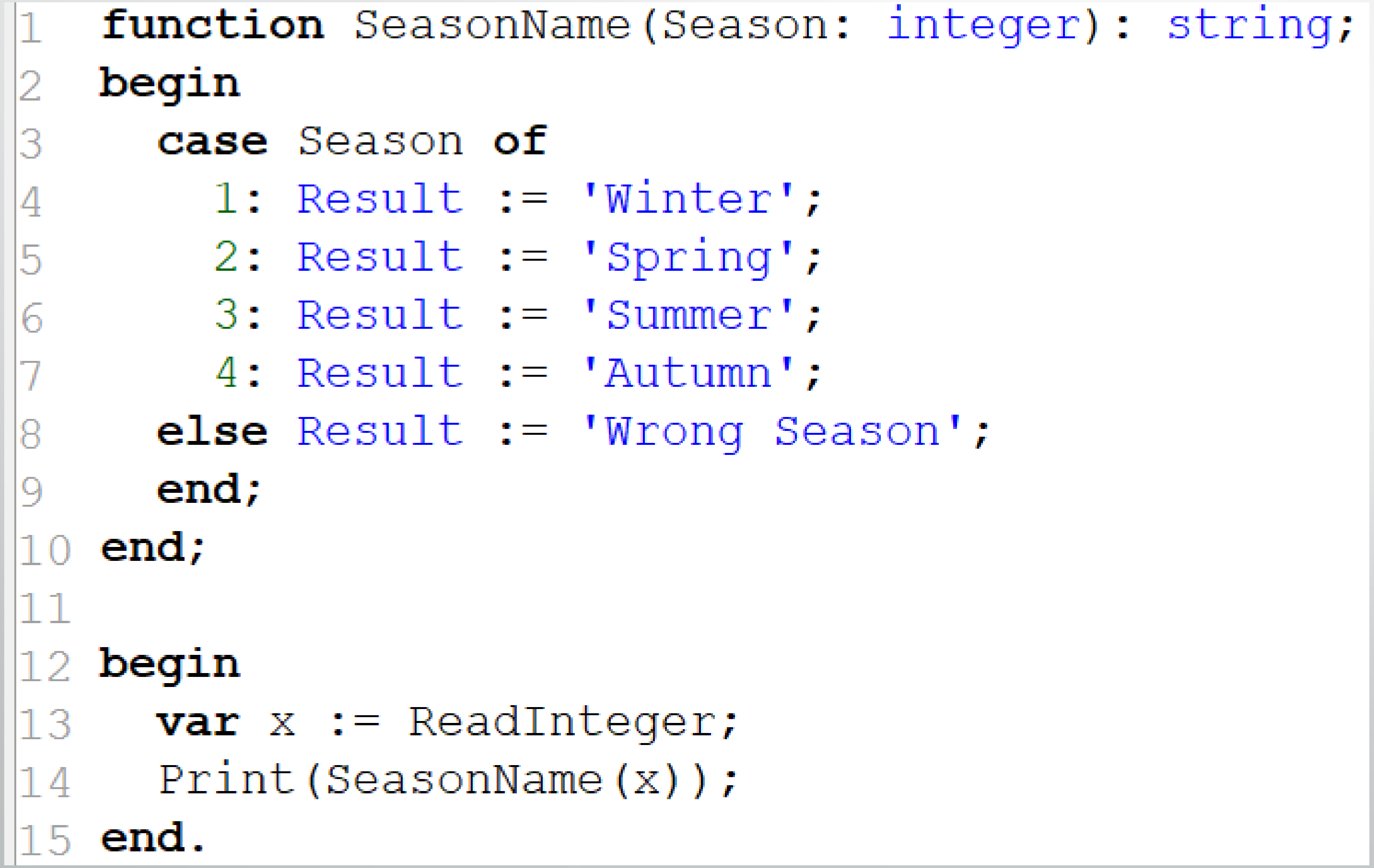


```
function Even(x: integer) := x mod 2 = 0;
function Odd(x: integer) := not Even(x);

begin
  var x := ReadInteger;
  Print(Even(x), Odd(x));
end.
```

Functions as wrappers for algorithms

- Example 3:

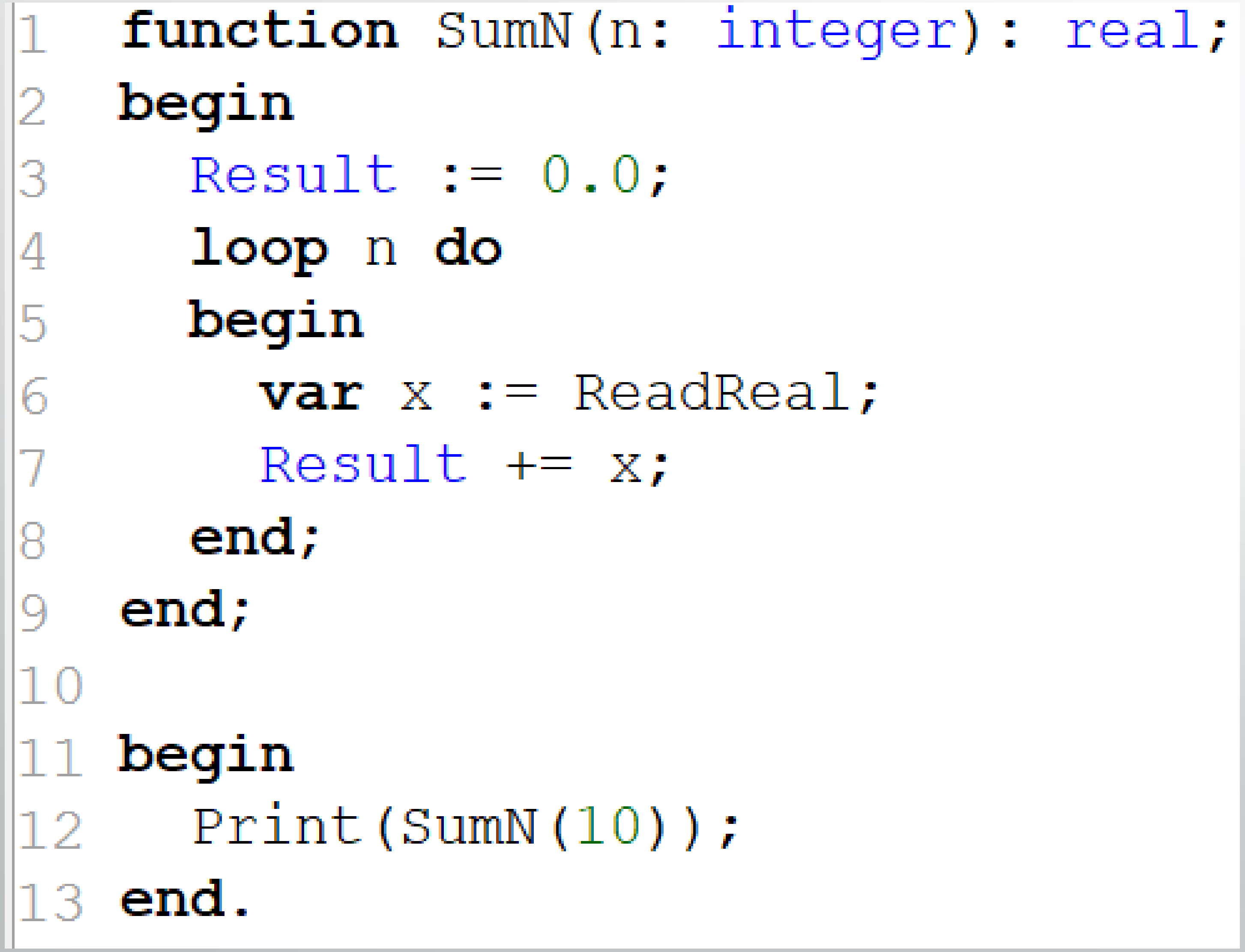


```
function SeasonName(Season: integer): string;
begin
  case Season of
    1: Result := 'Winter';
    2: Result := 'Spring';
    3: Result := 'Summer';
    4: Result := 'Autumn';
  else Result := 'Wrong Season';
  end;
end;

begin
  var x := ReadInteger;
  Print(SeasonName(x));
end.
```

Functions as wrappers for algorithms

- Example 4: Sum of n numbers

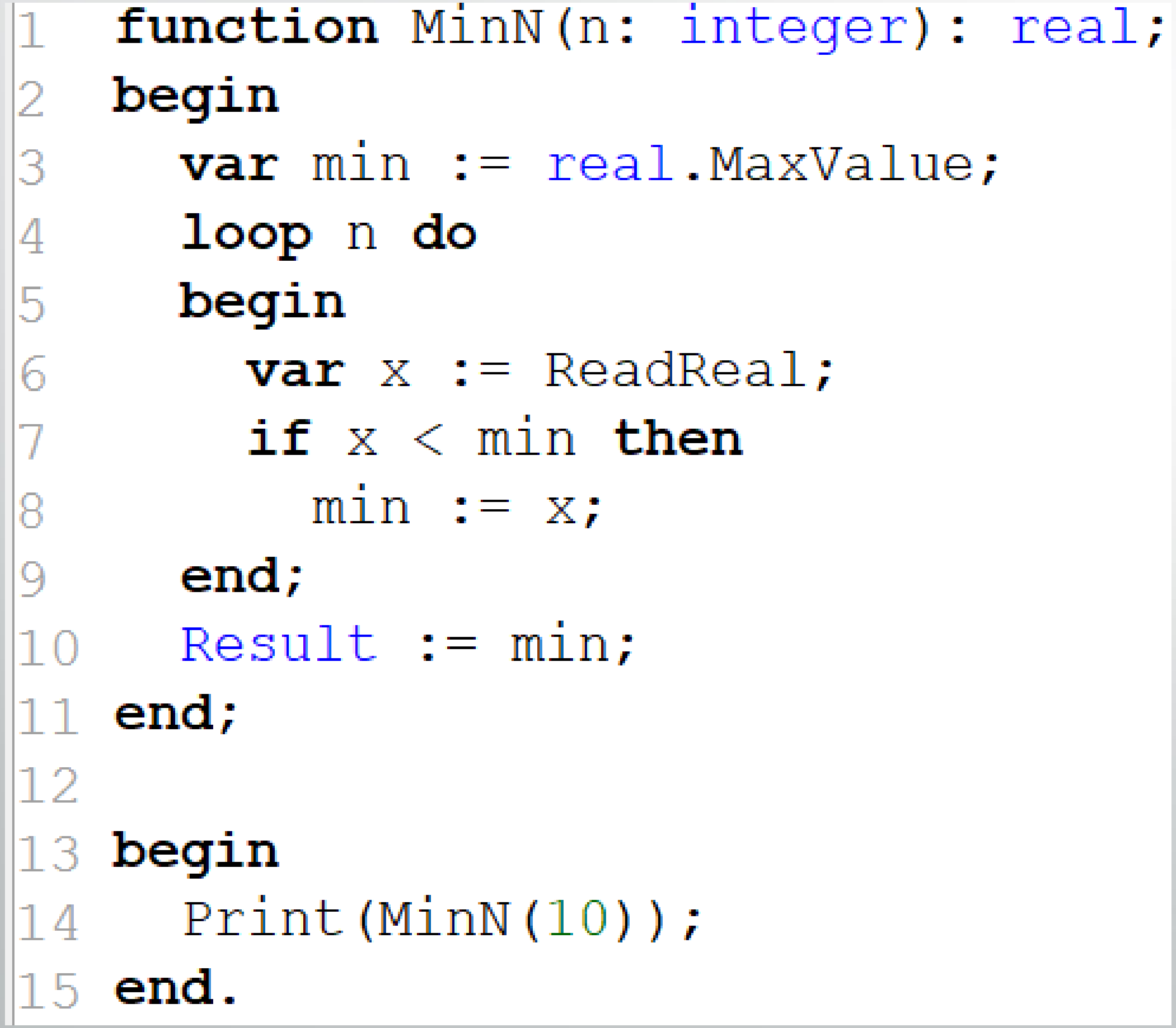


```
function SumN(n: integer): real;
begin
 Result := 0.0;
 loop n do
 begin
  var x := ReadReal;
  Result += x;
 end;
end;

begin
 Print(SumN(10));
end.
```

Functions as wrappers for algorithms

- Example 5: Minimal value among n numbers

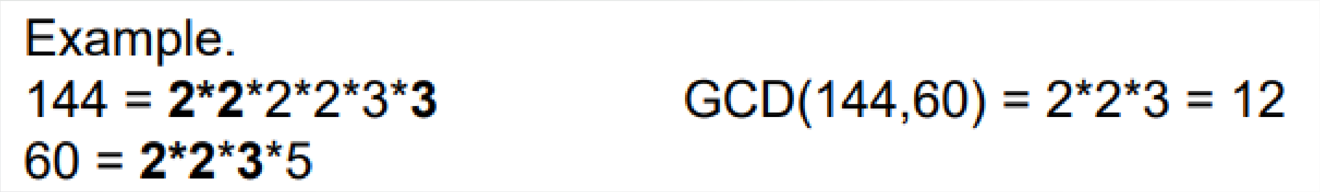


```
function MinN(n: integer): real;
begin
  var min := real.MaxValue;
  loop n do
  begin
    var x := ReadReal;
    if x < min then
      min := x;
  end;
  Result := min;
end;

begin
  Print(MinN(10));
end.
```

Functions as wrappers for algorithms

- Example 6: GCD

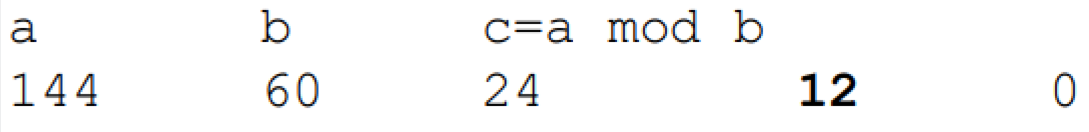


Example.

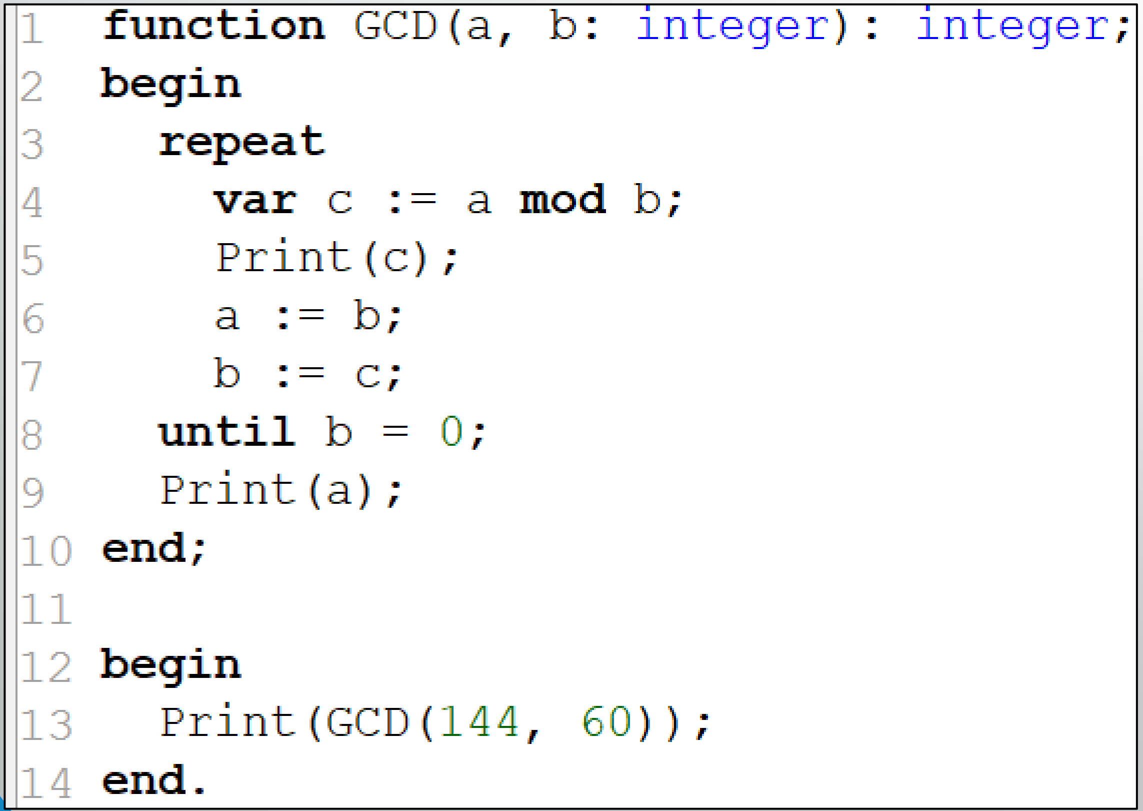
144 = 2*2*2*2*3*3   GCD(144,60) = 2*2*3 = 12

60 = 2*2*3*5

- The Euclidean Algorithm (3 century BC):



a	b	c=a mod b		
144	60	24	12	0

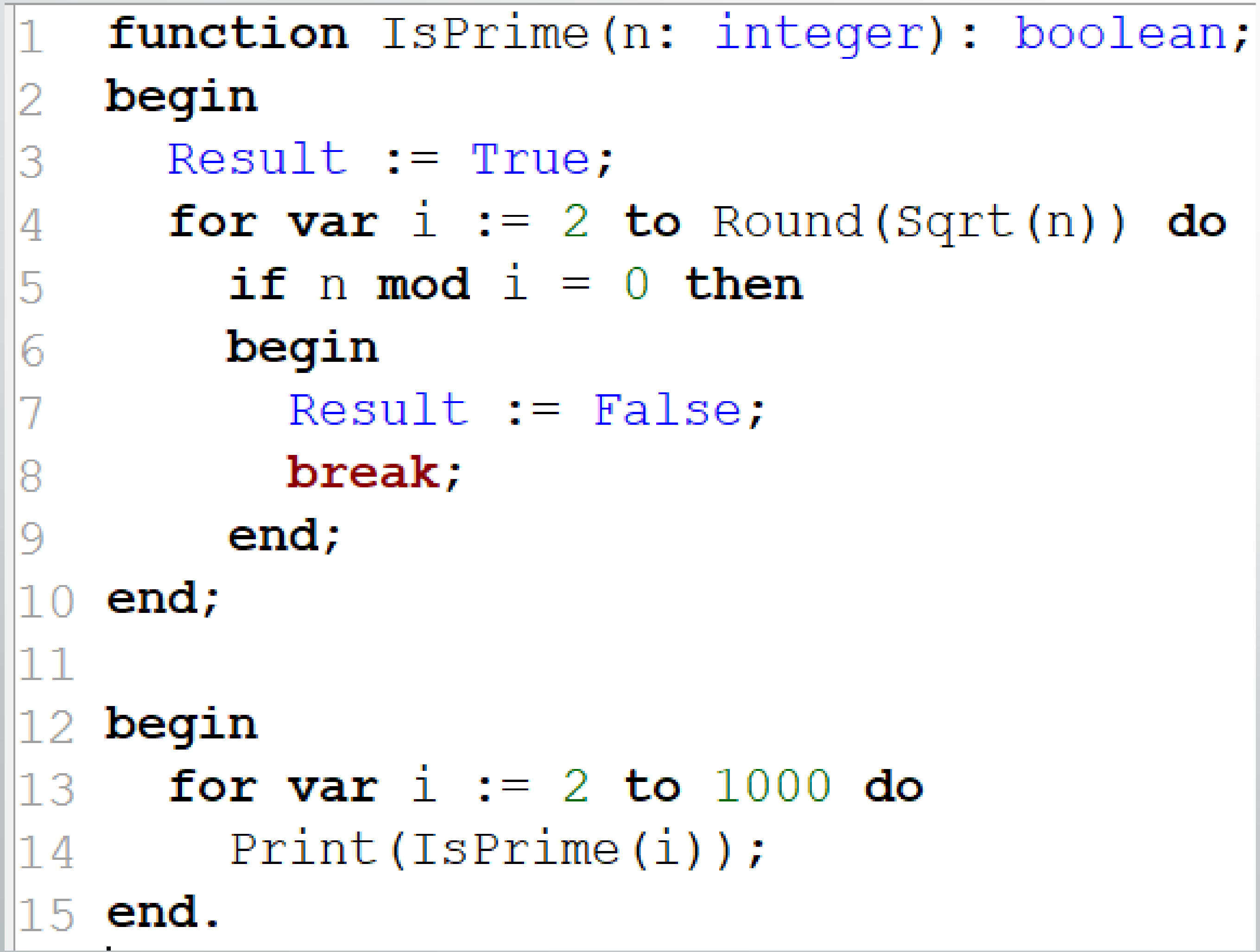


```
function GCD(a, b: integer): integer;
begin
  repeat
    var c := a mod b;
    Print(c);
    a := b;
    b := c;
  until b = 0;
  Print(a);
end;

begin
  Print(GCD(144, 60));
end.
```

Functions as wrappers for algorithms

- Example 7: Print all prime numbers in the range[2;1000]
- Solution. n is a prime number if it can only be divided by 1 and itself. If n is divisible by 2 .. n-1, then this is a composite number

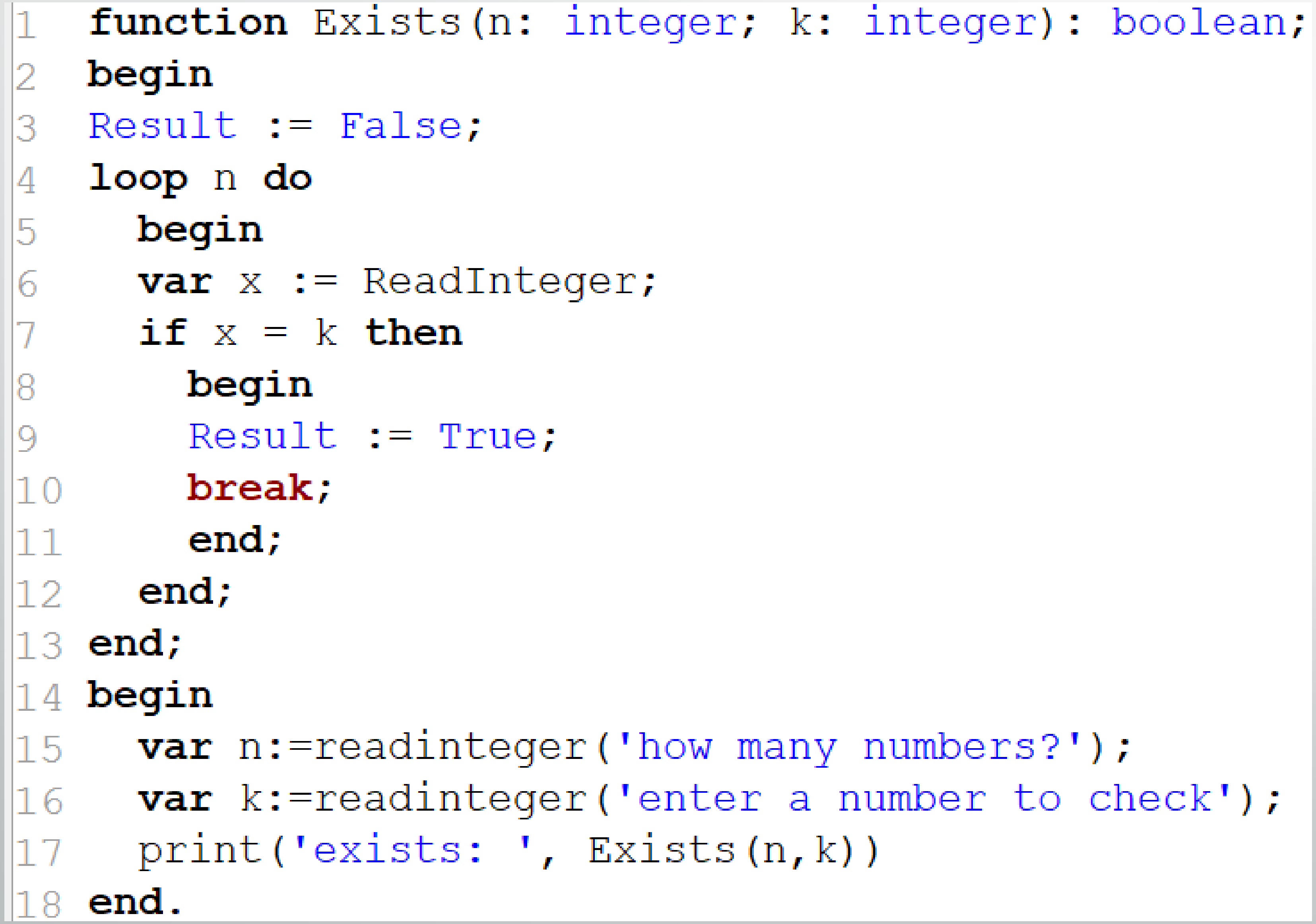


```
function IsPrime(n: integer): boolean;
begin
  Result := True;
  for var i := 2 to Round(Sqrt(n)) do
    if n mod i = 0 then
    begin
      Result := False;
      break;
    end;
end;

begin
  for var i := 2 to 1000 do
    Print(IsPrime(i));
end.
```

Functions of Boolean type

- Example 8: Print if k number is among the sequence of n numbers.



```
function Exists(n: integer; k: integer): boolean;
begin
Result := False;
loop n do
  begin
  var x := ReadInteger;
  if x = k then
    begin
    Result := True;
    break;
    end;
  end;
end;
begin
  var n:=readinteger('how many numbers?');
  var k:=readinteger('enter a number to check');
  print('exists: ', Exists(n,k))
end.
```

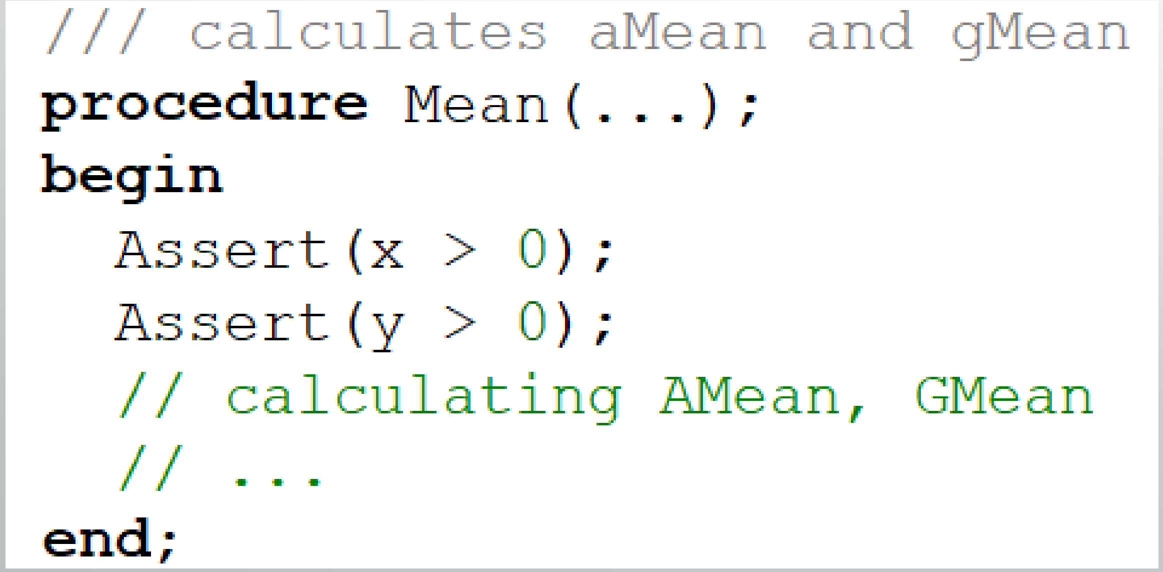
Tasks

- To do: Lesson # 11, Tasks 3,4,5,6

Assert

Input validation: Assert statement

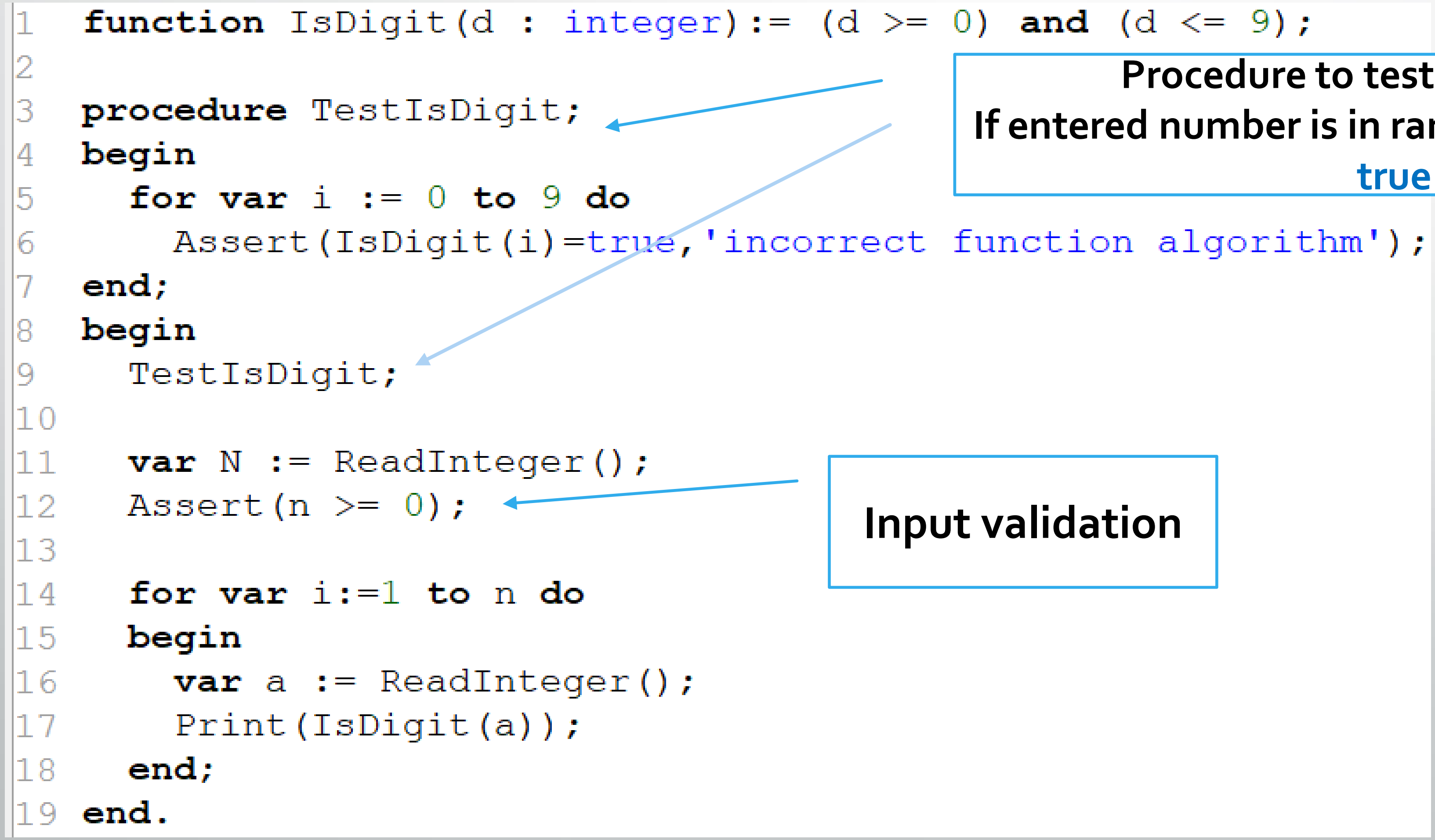
- Problem: Create the Mean (X, Y, AMean, GMean) procedure that calculates the arithmetic mean AMean = (X + Y) / 2 and the geometric mean GMean = (X Y) 1/2 of two positive numbers X and Y (X and Y are entered , AMean and GMean are real output parameters).
- The programmer must "secure" the program against incorrect input data.
- The positivity of the parameters X and Y is necessary for calculating the geometric mean, which occurs inside the Mean procedure. So, the check must be inside the Mean procedure:



```
/// calculates aMean and gMean
procedure Mean(...);
begin
  Assert(x > 0);
  Assert(y > 0);
  // calculating AMean, GMean
  // ...
end;
```

Testing the procedures and functions

- Problem: Create an IsDigit(D) function, which returns true if entered integer D is a digit (that is D is in the range [0,9]). In the main program output the results of this function for N entered numbers.



```
function IsDigit(d : integer):= (d >= 0) and (d <= 9);

procedure TestIsDigit;
begin
  for var i := 0 to 9 do
    Assert(IsDigit(i)=true,'incorrect function algorithm');
end;
begin
  TestIsDigit;

  var N := ReadInteger();
  Assert(n >= 0);

  for var i:=1 to n do
  begin
    var a := ReadInteger();
    Print(IsDigit(a));
  end;
end.
```

Procedure to test the function
If entered number is in range[1;9], must return
true

Input validation

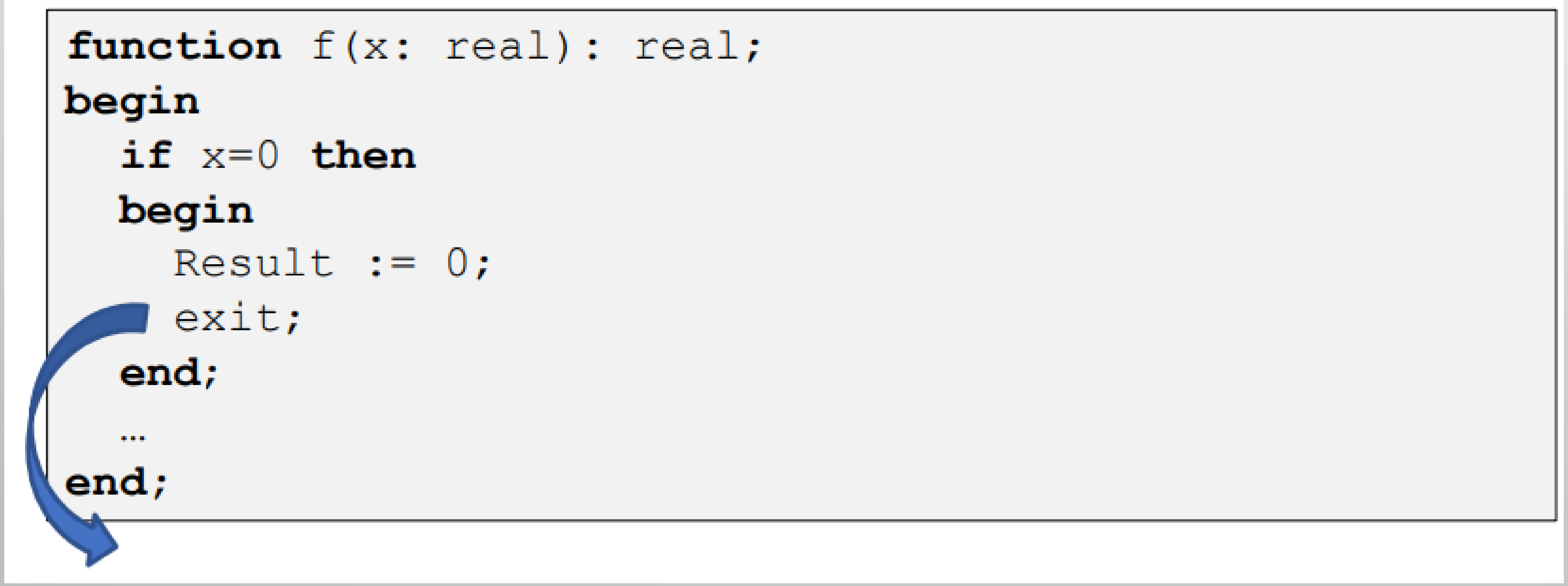
Tasks

- To do: Lesson # 11, Tasks 7,8,9,10

Exit statement

Exit statement

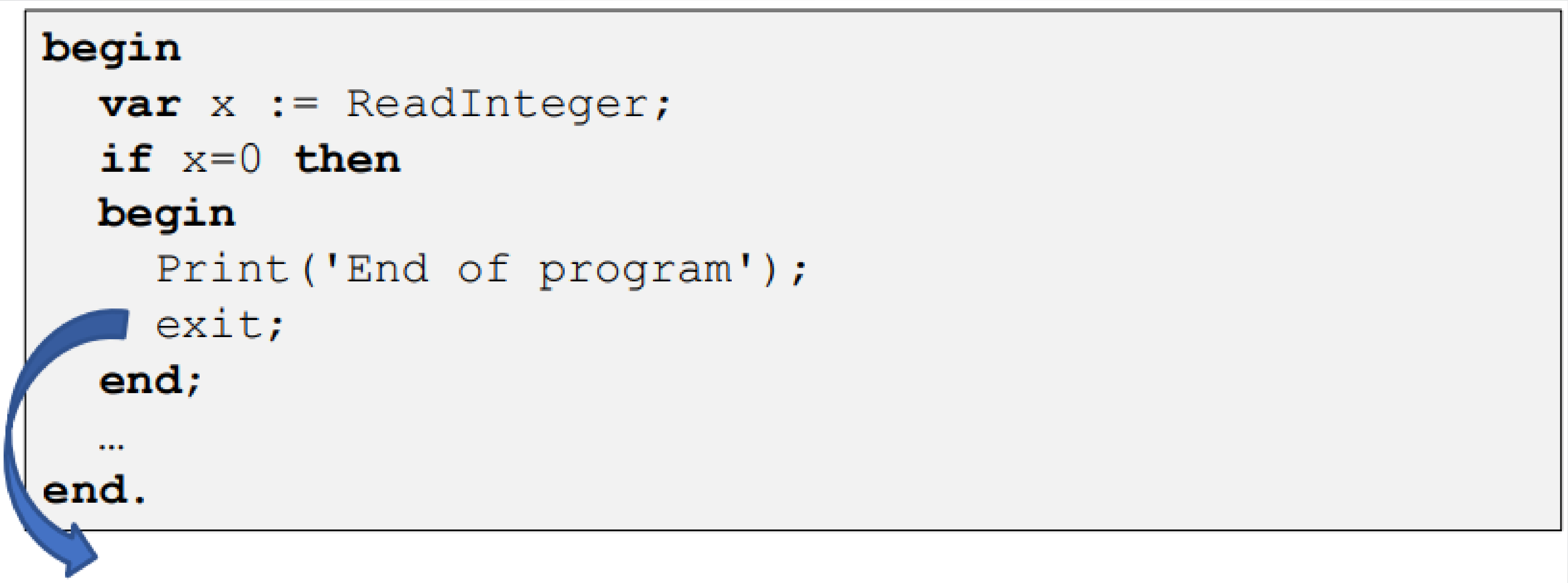
- Exit statement terminates an execution of a procedure or function.
- Exit statement is similar to break statement when it is inside function.



```
function f(x: real): real;
begin
  if x=0 then
  begin
    Result := 0;
    exit;
  end;
  …
end;
```

Exit statement and main program

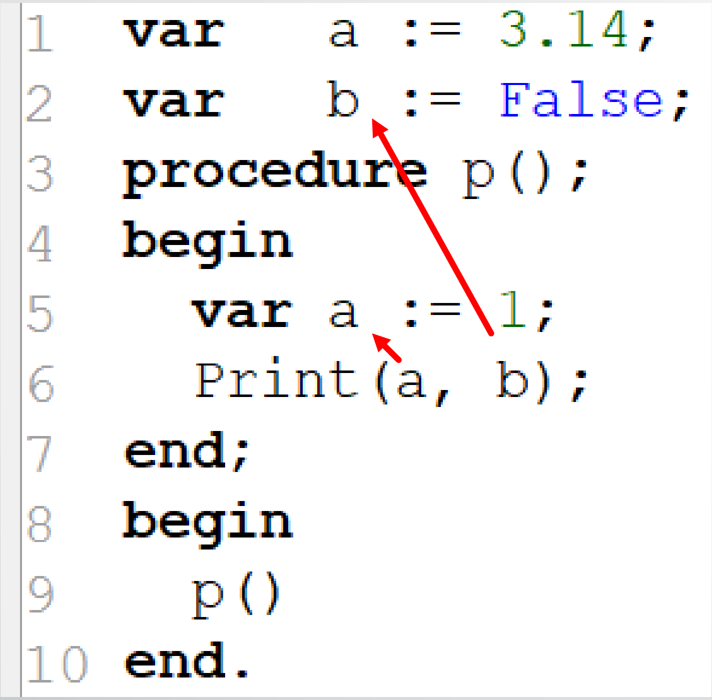
- If exit statement is used in a main program, it terminates the program.



```
begin
  var x := ReadInteger;
  if x=0 then
  begin
    Print('End of program');
    exit;
  end;
  …
end.
```

Local and global variables

- The variable outside a procedure or function is called a global variable.
- The variable within a procedure or function is called a local variable.

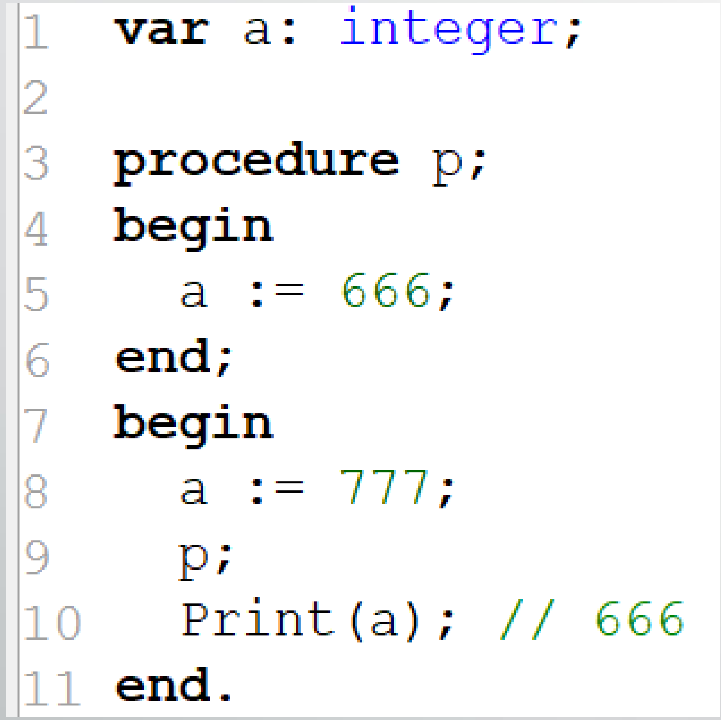


```
var    a := 3.14;
var    b := False;
procedure p();
begin
  var a := 1;
  Print(a, b);
end;
begin
  p()
end.
```

- How does the compiler find the variable?
- First, it finds the variable in the local scope.
- If this process fails, then it finds this variable in the global scope.

Side effect

- A side effect is that the value of a global variable is changed inside the procedure.
- As a rule it's bad practice.



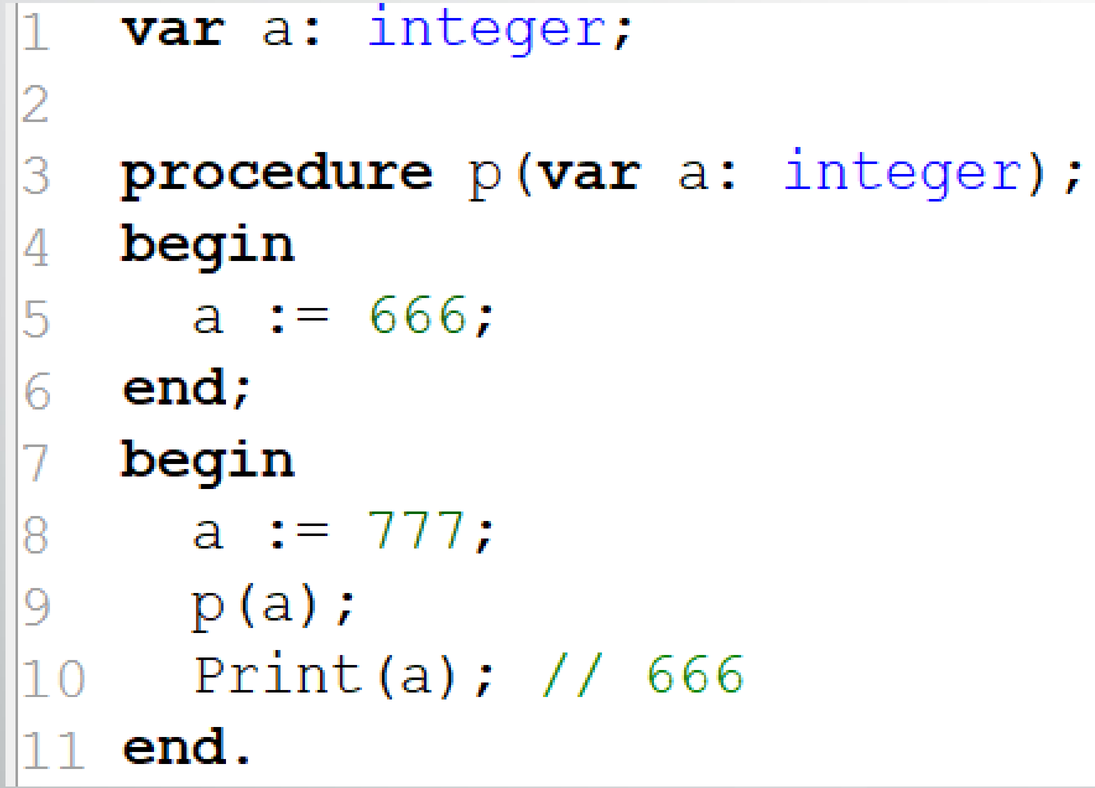
```
var a: integer;

procedure p;
begin
  a := 666;
end;
begin
  a := 777;
  p;
  Print(a); // 666
end.
```

- This is unexpected behavior.
- How to write a program correctly?

Without side effect

- To fix side effect behavior we must pass in a as a reference (var parameter):



```
var a: integer;

procedure p(var a: integer);
begin
  a := 666;
end;
begin
  a := 777;
  p(a);
  Print(a); // 666
end.
```

- It is a predictable behavior.

Q & A