# The Pascal Programming Language

(with material from tutorialspoint.com)

## Overview

- Background & History
- Features
- Hello, world!
- General Syntax
- Variables/Data Types
- Operators
- Conditional Statements
- Loops
- Functions and Procedures
- Arrays and Records

## Why Pascal?

- **well-structured, strongly typed**
  - explicit pass by value, pass by reference
- **imperative, object-oriented**
- **easy to learn**
  - originally developed as a learning language
  - surged in popularity in the 1980s
- **notable systems in Pascal**
  - Skype
  - TeX
  - embedded systems

## History

- **developed by Niklaus Wirth in the early 1970s**
  - developed for teaching programming with a general-purpose, high-level language
  - named for Blaise Pascal, French mathematician and pioneer in computer development
- **Algol-based**
  - Algol-60 is a subset of Pascal
  - block structure
- **used in early Mac development**
- **historically cited as**
  - easy to learn
  - structured
  - producing transparent, efficient, reliable programs
  - able to compile across multiple computer platforms

## Features of Pascal

- strongly typed
- extensive error checking
- arrays, records, files, and sets
- highly structured
- supports object-oriented programming

Source: xkcd.com/571

## Hello, world!

```
program HelloWorld (output);

{ main program }
begin
  writeln ('Hello, World!');
end.
```

- heading, declaration, execution parts
- { } comments
- writeln – with newline
- program ends with .

## General Syntax

- **comments**
  - { }
  - {* *} for multiline comments
    - {* this is a

      multiline comment *}
- **case insensitivity**
  - x and X are the same variable
  - reserved words: begin, Begin, and BEGIN all the same

## General Syntax

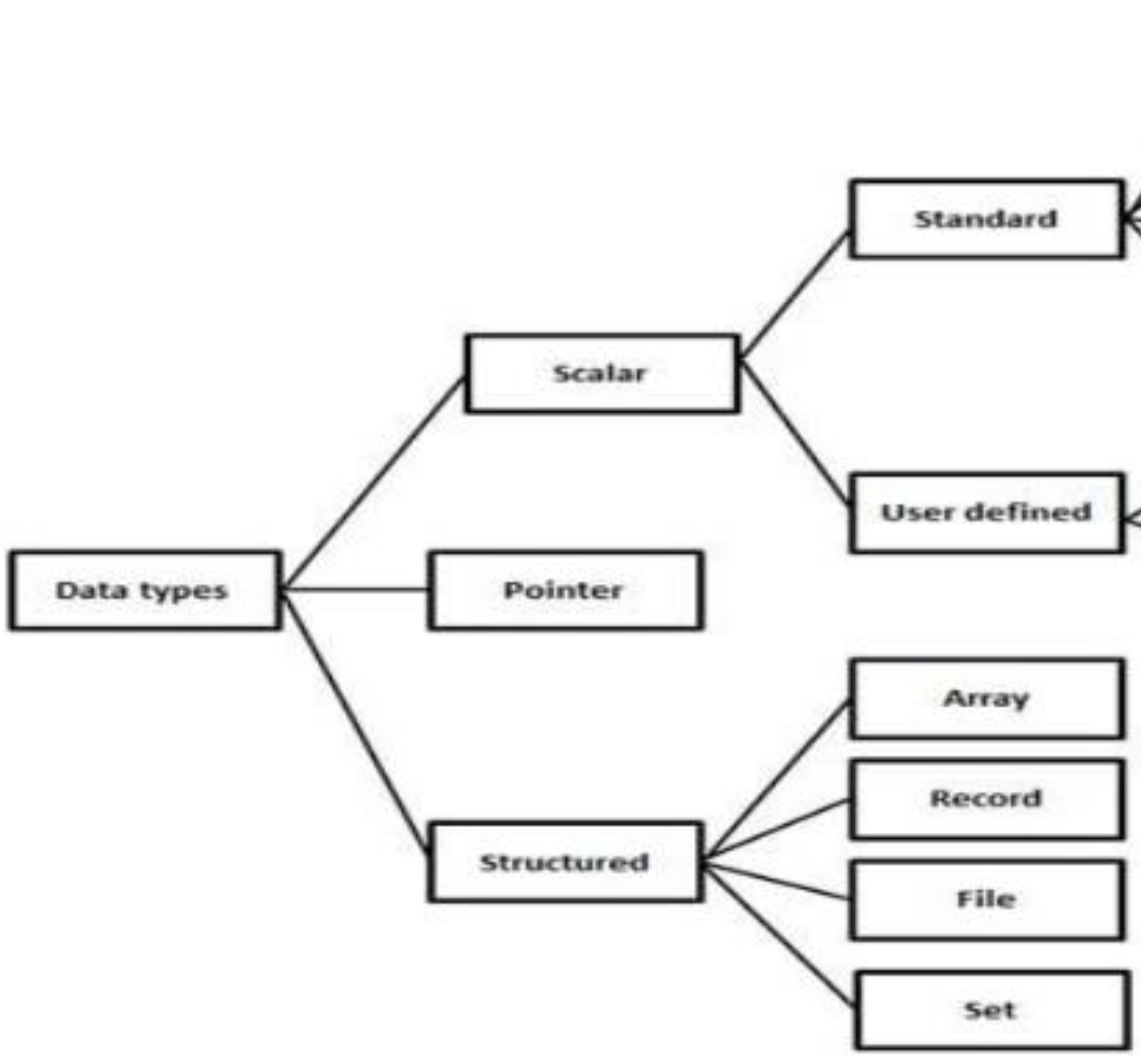
- **reserved words**

| | | | | |
|---|---|---|---|---|
| and | array | begin | case | const |
| div | do | downto | else | end |
| file | for | function | goto | if |
| in | label | mod | nil | not |
| of | or | packed | procedure | program |
| record | repeat | set | then | to |
| type | until | Var | while | with |

## Variables

- **var keyword**
  - beginning of variable declarations
  - before begin/end block
- **names**
  - letters or digits beginning with a letter
- **name1, name2 : type;**
- **examples**
  - x : integer;
  - r : real = 3.77;

## Data Types


Data types
Scalar
Pointer
Structured
Standard
User defined
Integer
Real
Character
Boolean
Enumerated
Subrange
Array
Record
File
Set

## Data Types

- **constants**
  - before var section
  - const

    DAYS_IN_WEEK = 7;

    NAME = 'Maria';

- **enumerated types**
  - order significant
  - type

    COLORS = (red, orange, yellow, green, blue, indigo, violet);

    MONTHS = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);

## Data Types

- **subranges**
  - subset of type within a certain range
    - grades on a test: 0..100
  - can appear in any section

```
type
   summer = (Jun..Sep);
var
   gr : 1..100;
```

- **user-defined types**
  - type

```
type
   days = integer;
var
   d : days;
```

## Example Program

```
program Welcome (input, output);

const
   intro = '***';

type
   name = string;

var
   firstname, lastname : name;

begin
  write ('Please enter your first name: ');
  readln (firstname);  writeln (firstname);
  write ('Please enter your last name: ');
  readln (lastname);   writeln (lastname);
  writeln;
  writeln (intro, 'Welcome, ', firstname, ' ', lastname);
end.
```

```
Please enter your first name: Christopher
Please enter your last name: Wren

***Welcome, Christopher Wren
```

## Example Program

```
program Circumference (input, output);

const
   PI = 3.14159;

var
   radius, diameter, circ: real;

begin
  write ('Enter the radius of the circle: ');
  readln (radius);  writeln (radius:4:2);

  diameter := 2 * radius;
  circ := PI * diameter;

  writeln ('The circumference is ', circ:7:2);
end.
```

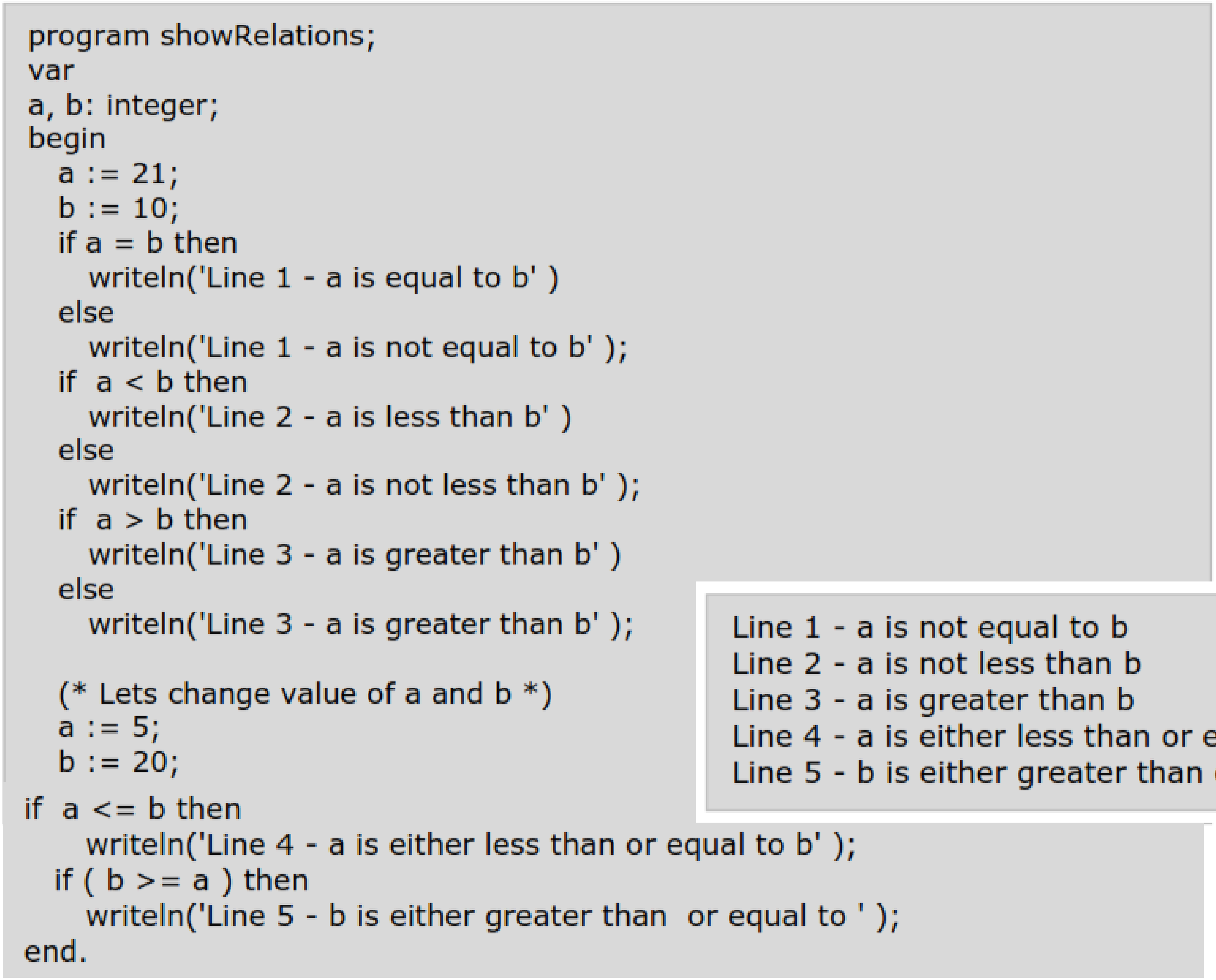
```
Enter the radius of the circle: 2.70
The circumference is   16.96
```

## Operators

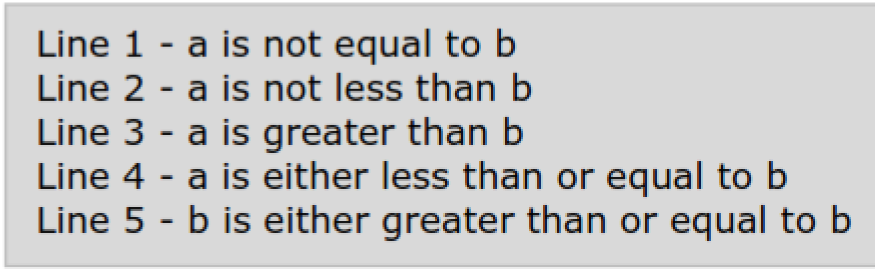
```
program calculator (input, output);

var
  a, b, c: integer;
  d      : real;

begin
  a := 21;
  b := 10;

  c := a + b;
  writeln ('Line 1 - Value of c is ', c);

  c := a - b;
  writeln ('Line 2 - Value of c is ', c);

  c := a * b;
  writeln ('Line 3 - Value of c is ', c);

  d := a / b;
  writeln ('Line 4 - Value of d is ', d:3:2);

  c := a mod b;
  writeln ('Line 5 - Value of c is ', c);

  c := a div b;
  writeln ('Line 6 - Value of c is ', c);
end.
```

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of d is 2.10
Line 5 - Value of c is 1
Line 6 - Value of c is 2
```

# Relational Operators



```
program showRelations;
var
a, b: integer;
begin
   a := 21;
   b := 10;
   if a = b then
      writeln('Line 1 - a is equal to b' )
   else
      writeln('Line 1 - a is not equal to b' );
   if  a < b then
      writeln('Line 2 - a is less than b' )
   else
      writeln('Line 2 - a is not less than b' );
   if  a > b then
      writeln('Line 3 - a is greater than b' )
   else
      writeln('Line 3 - a is greater than b' );

   (* Lets change value of a and b *)
   a := 5;
   b := 20;
if  a <= b then
      writeln('Line 4 - a is either less than or equal to b' );
   if ( b >= a ) then
      writeln('Line 5 - b is either greater than  or equal to ' );
end.
```



```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to b
Line 5 - b is either greater than or equal to b
```

## Logical Operators

```
program beLogical;
var
a, b: boolean;
begin
   a := true;
   b := false;

   if (a and b) then
      writeln('Line 1 - Condition is true' )
   else
      writeln('Line 1 - Condition is not true');
   if  (a or b) then
      writeln('Line 2 - Condition is true' );

   (* lets change the value of a and b *)
   a := false;
   b := true;
   if  (a and b) then
      writeln('Line 3 - Condition is true' )
   else
      writeln('Line 3 - Condition is not true' );
   if not (a and b) then
   writeln('Line 4 - Condition is true' );
end.
```

```
Line 1 - Condition is not true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

## Operator Precedence

```
program opPrecedence;
var
a, b, c, d : integer;
e: real;
begin
   a := 20;
   b := 10;
   c := 15;
   d := 5;
   e := (a + b) * c / d;      (* ( 30 * 15 ) / 5 *)
   writeln('Value of (a + b) * c / d is : ',  e:3:1 );

   e := ((a + b) * c) / d;    (* (30 * 15 ) / 5  *)
   writeln('Value of ((a + b) * c) / d is  : ' ,  e:3:1 );

   e := (a + b) * (c / d);   (*  (30) * (15/5)  *)
   writeln('Value of (a + b) * (c / d) is  : ',  e:3:1);

   e := a + (b * c) / d;     (*  20 + (150/5)  *)
   writeln('Value of a + (b * c) / d is  : ' ,  e:3:1 );
end.
```

```
Value of (a + b) * c / d is : 90.0
Value of ((a + b) * c) / d is  : 90.0
Value of (a + b) * (c / d) is  : 90.0
Value of a + (b * c) / d is  : 50.0
```

## Conditional Statements

- **if-then**

```
if (a <= 20) then
   c:= c+1;
```

- **if-then-else**

```
if color = red then
   writeln('You have chosen a red car')
else
   writeln('Please choose a color for your car');
```

## Conditional Statements

```
program ifelse_ifelseChecking;
var
   { local variable definition }
   a : integer;
begin
   a := 100;
  (* check the boolean condition *)
   if (a = 10)  then
      (* if condition is true then print the following *)
      writeln('Value of a is 10' )
   else if ( a = 20 ) then
      (* if else if condition is true *)
      writeln('Value of a is 20' )
   else if( a = 30 ) then
      (* if else if condition is true  *)
      writeln('Value of a is 30' )
   else
      (* if none of the conditions is true *)
      writeln('None of the values is matching' );
    writeln('Exact value of a is: ', a );
end.
```

```
None of the values is matching
Exact value of a is: 100
```

## Conditional Statements

- use begin/end blocks, if necessary

```
if( boolean_expression 1) then
begin
   if(boolean_expression 2)then
      S1
   else
      S2;
end;
```

- different from above

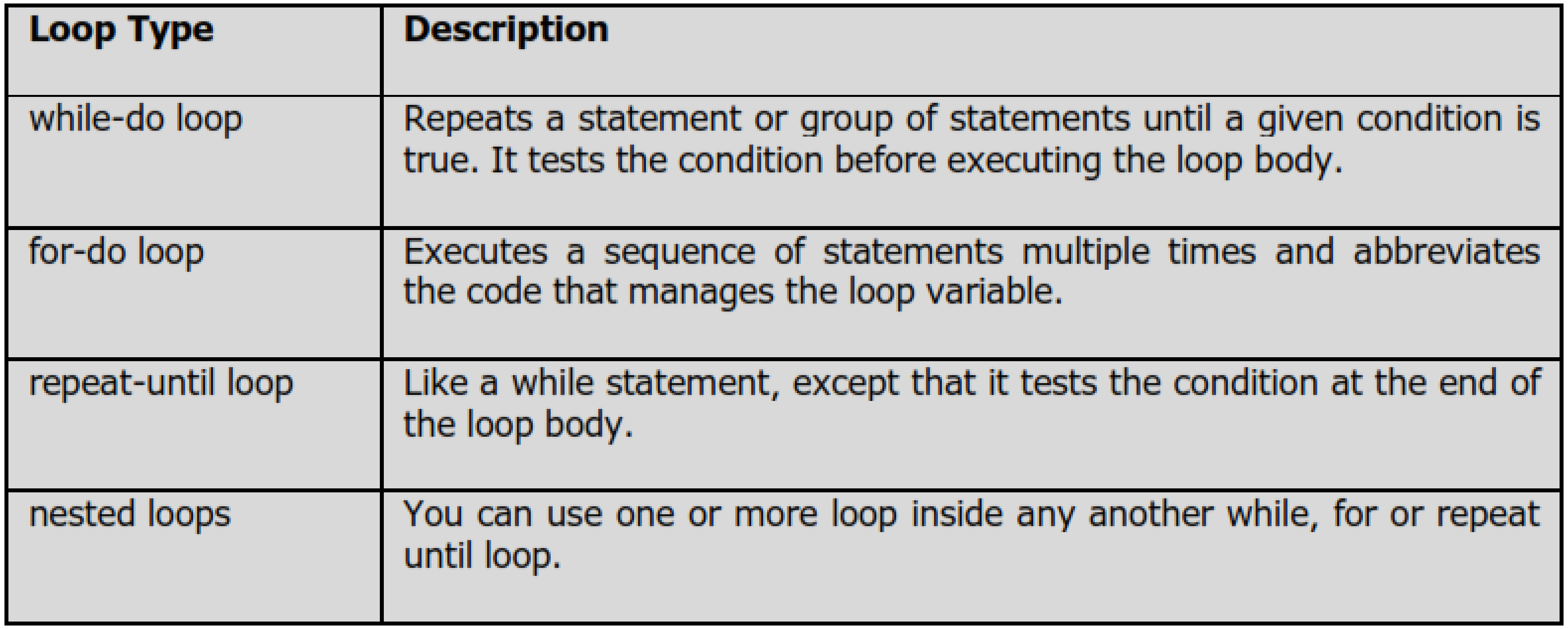
```
if ( boolean_expression 1) then
begin
   if exp2 then
      S1
end;
   else
      S2;
```

## Case Statements

```
program checkCase;
var
  grade: char;
begin
  grade := 'F';
  case (grade) of
    'A' : writeln('Excellent!' );
    'B', 'C': writeln('Well done' );
    'D' : writeln('You passed' );
  else
    writeln('You really did not study right!' );
   end;
  writeln('Your grade is  ', grade );
end.
```

```
You really did not study right!
Your grade is F
```

## Loops



| Loop Type | Description |
|---|---|
| while-do loop | Repeats a statement or group of statements until a given condition is true. It tests the condition before executing the loop body. |
| for-do loop | Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| repeat-until loop | Like a while statement, except that it tests the condition at the end of the loop body. |
| nested loops | You can use one or more loop inside any another while, for or repeat until loop. |

## Loops

- **while-do**

```
while (condition) do S;
```

```
while number>0 do
begin
   sum := sum + number;
   number := number - 2;
end;
```

- **break**
- **continue**

## Loops

- **while-do**

```
program whileLoop;
var
   a: integer;
begin
   a := 10;
   while  a < 20  do
   begin
      writeln('value of a: ', a);
      a := a + 1;
   end;
end.
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

## Loops

- for-do

```
for i:= 1 to 10 do writeln(i);
```

```
program forLoop;
var
   a: integer;
begin
   for a := 10  to 20 do
   begin
      writeln('value of a: ', a);
   end;
end.
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

## Loops

- **repeat-until**

```
repeat
   sum := sum + number;
   number := number - 2;
until number = 0;
```

```
program repeatUntilLoop;
var
   a: integer;
begin
   a := 10;
   (* repeat until loop execution *)
   repeat
      writeln('value of a: ', a);
      a := a + 1
   until a = 20;
end.
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

## Loops

- **nested loops**

```
program nestedPrime;
var
   i, j:integer;
begin
   for i := 2 to 50 do
   begin
      for j := 2 to i do
         if (i mod j)=0  then
            break; {* if factor found, not prime *}
      if(j = i) then
         writeln(i , ' is prime' );
   end;
end.
```

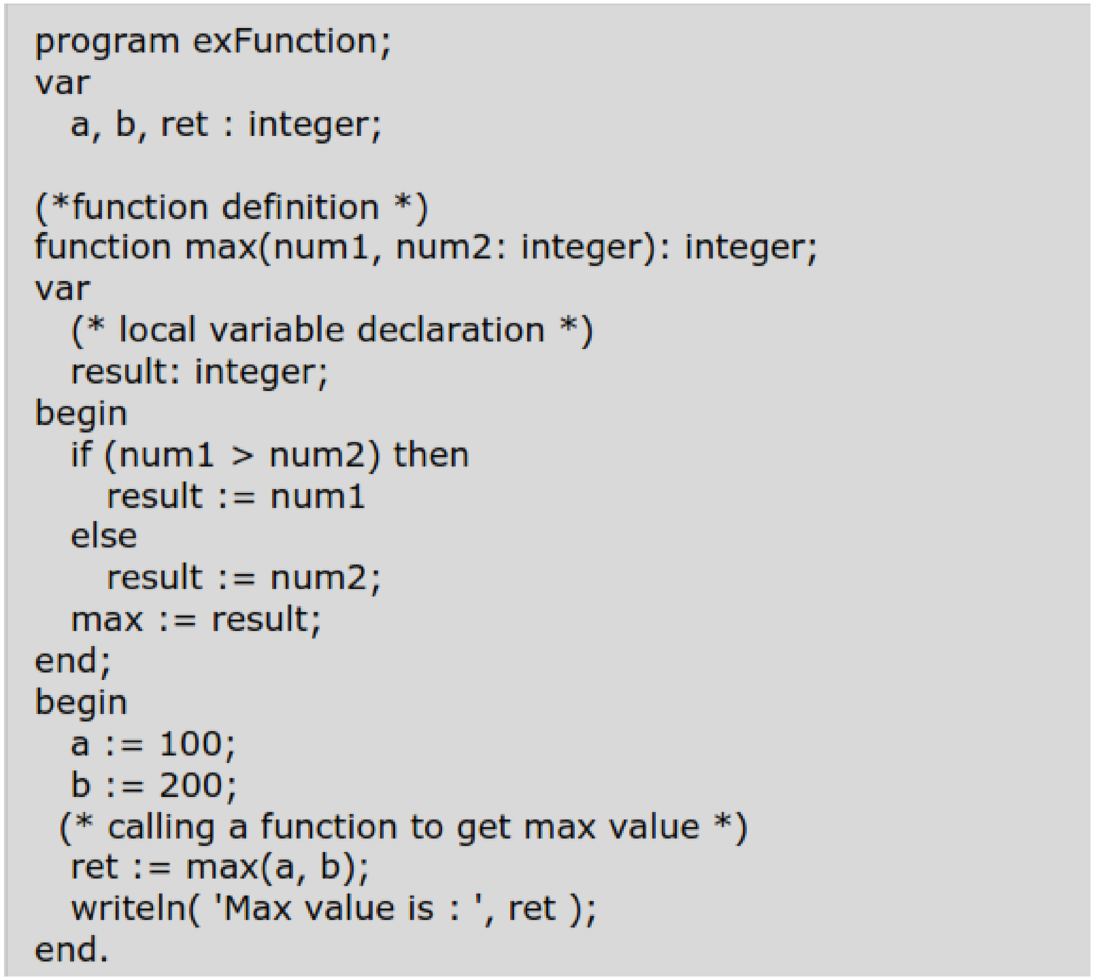
```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
```

## Functions and Procedures

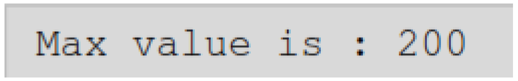
- **Pascal has explicit differentiation between functions and procedures**
  - different reserved words
  - functions must return a value
  - procedures do not return a value
- **recursion allowed**

# Functions

- please don’t write code formatted like this



```
program exFunction;
var
   a, b, ret : integer;

(*function definition *)
function max(num1, num2: integer): integer;
var
   (* local variable declaration *)
   result: integer;
begin
   if (num1 > num2) then
      result := num1
   else
      result := num2;
   max := result;
end;
begin
   a := 100;
   b := 200;
  (* calling a function to get max value *)
   ret := max(a, b);
   writeln( 'Max value is : ', ret );
end.
```



```
Max value is : 200
```

## Procedures

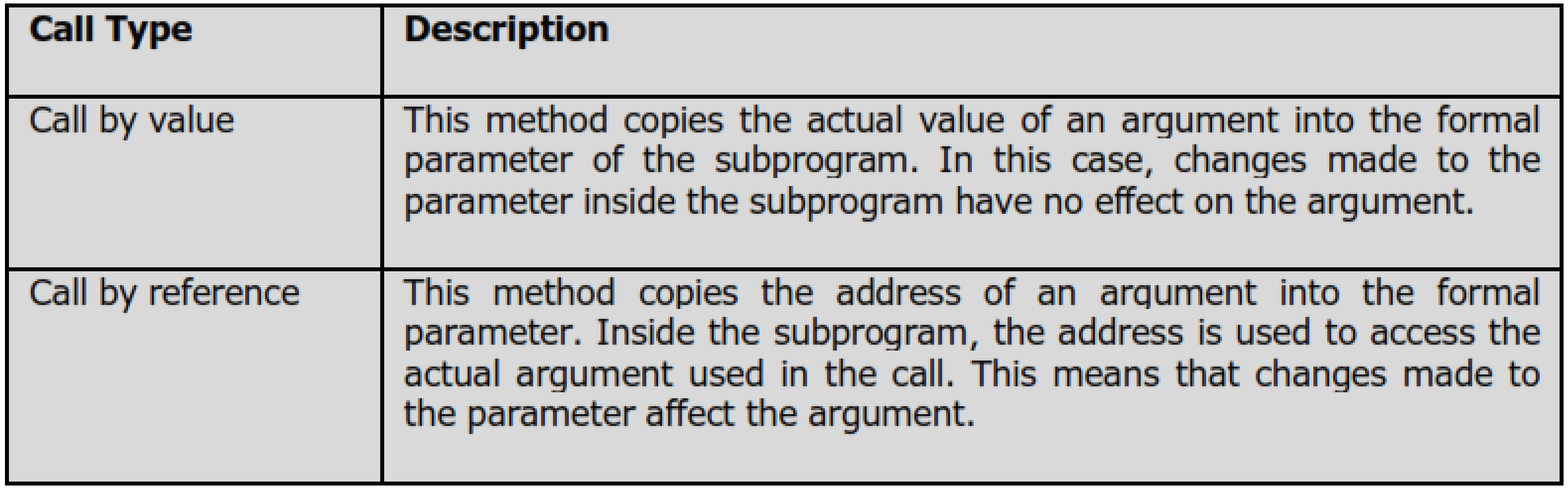
- **please don’t write code formatted like this, either**

```
program exProcedure;
var
   a, b, c,  min: integer;
procedure findMin(x, y, z: integer; var m: integer);
(* Finds the minimum of the 3 values *)
begin
   if x < y then
      m:= x
   else
      m:= y;
   if z < m then
      m:= z;
end; { end of procedure findMin }
begin
   writeln(' Enter three numbers: ');
   readln( a, b, c);
   findMin(a, b, c, min); (* Procedure call *)
   writeln(' Minimum: ', min);
end.
```

```
Enter three numbers:
89 45 67
Minimum: 45
```

## Parameter Passing

- **call by value and call by reference**
  - explicitly differentiated through var keyword



| Call Type | Description |
|---|---|
| Call by value | This method copies the actual value of an argument into the formal parameter of the subprogram. In this case, changes made to the parameter inside the subprogram have no effect on the argument. |
| Call by reference | This method copies the address of an argument into the formal parameter. Inside the subprogram, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument. |

## Parameter Passing: Call by Value

```
program exCallbyValue;
var
   a, b : integer;
(*procedure definition *)
procedure swap(x, y: integer);
var
   temp: integer;
begin
   temp := x;
   x:= y;
   y := temp;
end;
begin
   a := 100;
   b := 200;
   writeln('Before swap, value of a : ', a );
   writeln('Before swap, value of b : ', b );
(* calling the procedure swap  by value   *)
   swap(a, b);
   writeln('After swap, value of a : ', a );
   writeln('After swap, value of b : ', b );
end.
```

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200
```

# Parameter Passing: Call by Reference

```
program exCallbyRef;
var
   a, b : integer;
(*procedure definition *)
procedure swap(var x, y: integer);
var
   temp: integer;
begin
   temp := x;
   x:= y;
   y := temp;
end;

begin
   a := 100;
   b := 200;
   writeln('Before swap, value of a : ', a );
   writeln('Before swap, value of b : ', b );
   (* calling the procedure swap  by value   *)
   swap(a, b);
   writeln('After swap, value of a : ', a );
   writeln('After swap, value of b : ', b );
end.
```

```
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of b : 100
```

## Arrays

- aggregate of like types
- contiguous memory
- examples

```
type
   vector = array [ 1..25] of real;
var
   velocity: vector;
```

- different types of subscripts allowed

```
type
   temperature = array [-10 .. 50] of real;
var
   day_temp, night_temp: temperature;
```

- packed arrays store data, such as chars, side by side instead of along the default 4-byte boundary

## Arrays

- **example**

```
program exArrays;
var
   n: array [1..10] of integer;   (* n is an array of 10 integers *)
   i, j: integer;
begin
   (* initialize elements of array n to 0 *)
   for i := 1 to 10 do
       n[ i ] := i + 100;   (* set element at location i to i + 100 *)
    (* output each array element's value *)
   for j:= 1 to 10 do
      writeln('Element[', j, '] = ', n[j] );
end.
```

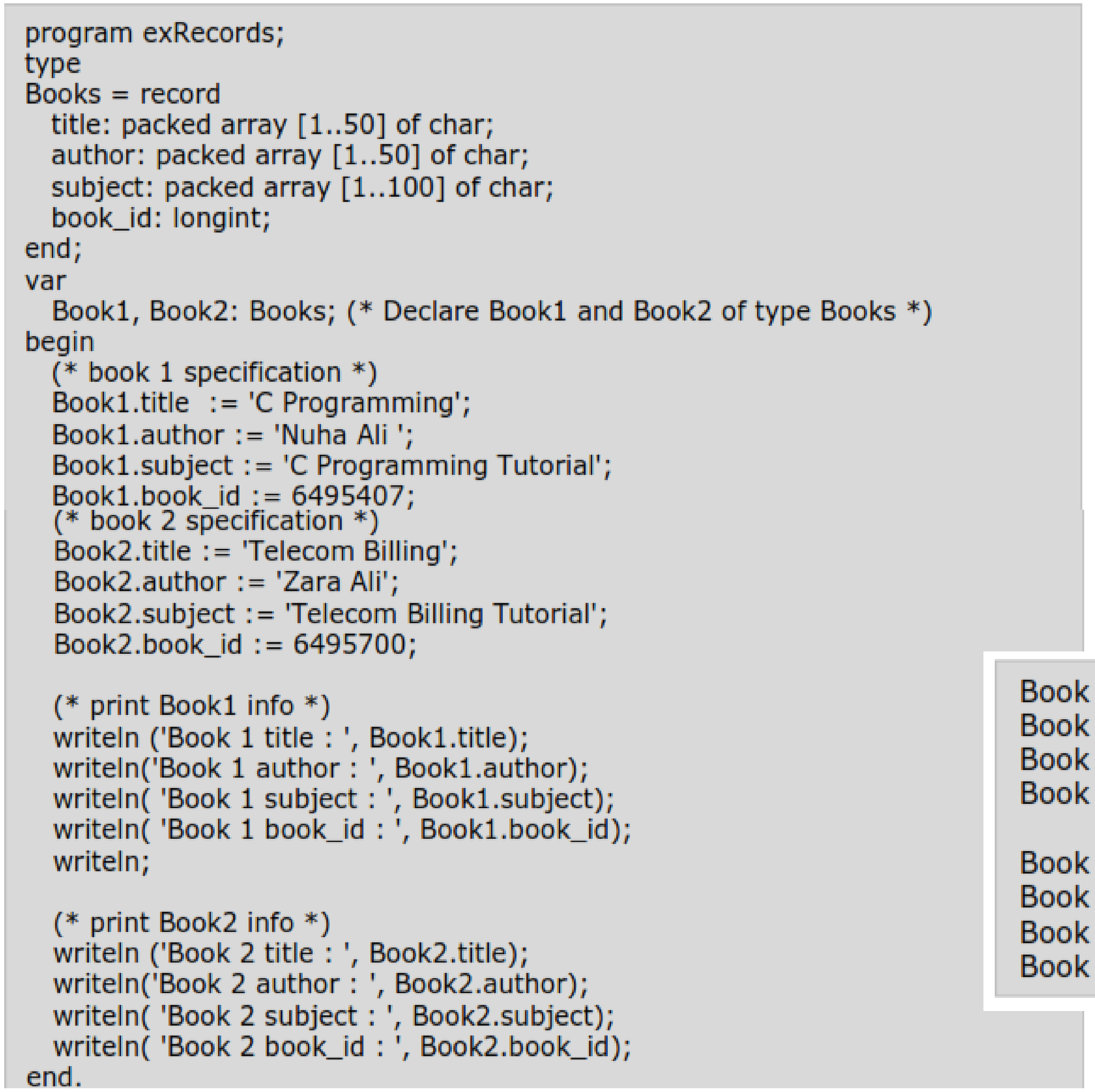
```
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
Element[10] = 110
```

## Records

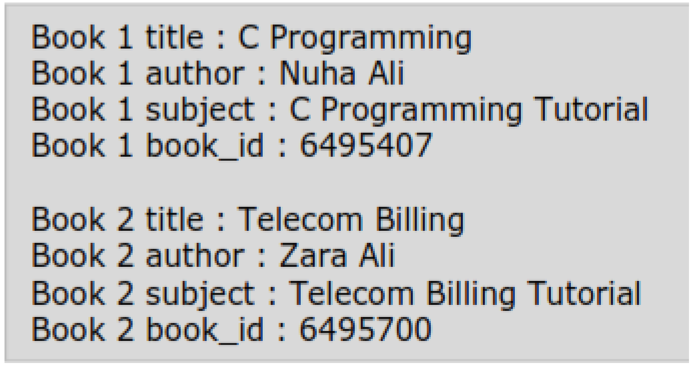
- aggregate with differing types
- must use type declaration
- example

```
type
Books = record
   title: packed array [1..50] of char;
   author: packed array [1..50] of char;
   subject: packed array [1..100] of char;
   book_id: integer;
end;
```

# Records



```
program exRecords;
type
Books = record
   title: packed array [1..50] of char;
   author: packed array [1..50] of char;
   subject: packed array [1..100] of char;
   book_id: longint;
end;
var
   Book1, Book2: Books; (* Declare Book1 and Book2 of type Books *)
begin
   (* book 1 specification *)
   Book1.title  := 'C Programming';
   Book1.author := 'Nuha Ali ';
   Book1.subject := 'C Programming Tutorial';
   Book1.book_id := 6495407;
   (* book 2 specification *)
   Book2.title := 'Telecom Billing';
   Book2.author := 'Zara Ali';
   Book2.subject := 'Telecom Billing Tutorial';
   Book2.book_id := 6495700;

   (* print Book1 info *)
   writeln ('Book 1 title : ', Book1.title);
   writeln('Book 1 author : ', Book1.author);
   writeln( 'Book 1 subject : ', Book1.subject);
   writeln( 'Book 1 book_id : ', Book1.book_id);
   writeln;

   (* print Book2 info *)
   writeln ('Book 2 title : ', Book2.title);
   writeln('Book 2 author : ', Book2.author);
   writeln( 'Book 2 subject : ', Book2.subject);
   writeln( 'Book 2 book_id : ', Book2.book_id);
end.
```



```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407

Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

# Records

```
program exRecords;
type
Books = record
   title: packed array [1..50] of char;
   author: packed array [1..50] of char;
   subject: packed array [1..100] of char;
   book_id: longint;
end;
var
   Book1, Book2: Books; (* Declare Book1 and Book2 of type Books *)
```

```
(* procedure declaration *)
procedure printBook( var book: Books );
begin
   (* print Book info *)
   writeln ('Book  title : ', book.title);
   writeln('Book  author : ', book.author);
   writeln( 'Book  subject : ', book.subject);
   writeln( 'Book book_id : ', book.book_id);
end;

begin
   (* book 1 specification *)
   Book1.title  := 'C Programming';
   Book1.author := 'Nuha Ali ';
   Book1.subject := 'C Programming Tutorial';
   Book1.book_id := 6495407;

  (* book 2 specification *)
   Book2.title := 'Telecom Billing';
   Book2.author := 'Zara Ali';
   Book2.subject := 'Telecom Billing Tutorial';
   Book2.book_id := 6495700;

  (* print Book1 info *)
   printbook(Book1);
   writeln;

   (* print Book2 info *)
   printbook(Book2);
end.
```

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407

Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

# Other Topics

- **pointers**
- **sets**
- **variants**
  - like unions in C/C++
- **strings**
- **file I/O**
- **memory management**
- **classes**