
Object Pascal

Michalis Kamburelis

#####

1. #####: ### #### ## #####?	2
2. #####	4
2.1. ##### "Hello world!"	4
2.2. #####, #####, ##### #####	5
2.3. ##### ##### (if)	7
2.4. #####, #####, ##### ##### # ##### (#####) #####	9
2.5. ##### ##### ##### ## ##### (##### case)	10
2.6. ##### # #####, ##### # ##### #####	11
2.7. ##### (for, while, repeat, for .. in)	13
2.8. ##### # #####	16
2.9. ##### # ##### ##	17
3. ##### (Unit-#)	19
3.1. ##### ##### unit-###	21
3.2. ##### # ##### unit-#	22
3.3. ##### ##### unit-# #####	25
4. #####	26
4.1. #####	26
4.2. ##### (Inheritance), ##### (is), # ##### (as) ...	27
4.3. #####	30
4.4. #####	33
4.5. #####	34
4.6. ##### ##	35
5. #####	35
5.1. #####	35
5.2. #####	36
5.3. ##### #	37
5.4. Free notification	40

6. Run-time library	43
6.1. #####/##### # #####	43
6.2. #####	45
6.3. #####: TPersistent.Assign	45
7. #####	48
7.1. ##### (#####) #####	48
7.2. ##### — ## #, ## # ##### # #, ## # #####	49
7.3. Generic-#	52
7.4. Overload	54
7.5. #####	55
7.6. Record	58
7.7. ##### object	60
7.8. Pointer-#	60
7.9. #####	61
8. #####	65
8.1. ##### private # strict private	65
8.2. ##### ##### ##### #	66
8.3. Class method	67
8.4. ##### #	68
8.5. Class helper	70
8.6. ##### constructor-#, destructor-#	72
8.7. ##### # constructor-#	72
9. #####	74
9.1. ##### (CORBA) #####	74
9.2. CORBA # COM #####	76
9.3. GUID-# #####	78
9.4. ##### (COM) #####	79
9.5. ##### COM ##### # ##### reference- counting	82
9.6. ##### #####	84
10. #####	88

1. #####: ## # ## #?

#, #, # # #
"#####", # #, #, generic-# (#####) #
#.

2.

2.1. ##### "Hello world!"

"Hello world". ## ##### ## ##### ## ##:

```
// Эту строку необходимо использовать во всех современных программах
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
// Needed for console programs on Windows,
// otherwise (with Delphi) the default is GUI program without console.
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
program MyProgram; // Сохраните этот файл под названием myprogram.dpr
begin
  WriteLn('Hello world!');
end.
```

— #####, ##### # #####.

- ##### # ##### # ##### FPC, ##### # #####
myprogram.dpr # ##### # ##### fpc
myprogram.dpr .
- ##### Lazarus, ## ##### (#
#: Project → New Project → Simple Program). ##### ##
myprogram # ##### # #####. #####
Run → Compile # #####.
- ##### # #####, ## #, # #####

#####.

#####, #####
#####-##### # #####. #####
##-##### "#####", ##### # ##### GUI
Lazarus (Project → New Project → Application). #####! — #####
GUI # #####, # #####, #####
#####. Lazarus # Free Pascal Compiler #####
#####, GUI, ## # ##### # #####
(XML, json, #####...), ##### # #####,

```
#####  
##### # ##### Object Pascal ###  
#####
```

```
### #####  
Castle Game Engine, # ##### :)
```

2.2. #####, #####,

```
program MyProgram;  
  
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}  
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}  
  
procedure MyProcedure(const A: Integer);  
begin  
  WriteLn('A + 10 составляет: ', A + 10);  
end;  
  
function MyFunction(const S: string): string;  
begin  
  Result := S + 'строки управляются автоматически';  
end;  
  
var  
  X: Single;  
begin  
  WriteLn(MyFunction('примечание: '));  
  MyProcedure(5);  
  
  // деление с помощью оператора "/" всегда даёт результат с плавающей  
  // для целочисленного деления необходимо использовать "div"  
  X := 15 / 5;  
  WriteLn('X составляет: ', X); // отобразить в научном формате вида  
  3.0000000E+000  
  WriteLn('X составляет: ', X:1:2); // отобразить 2 знака после запятой  
end.
```

```
#####  
##### "#####" ##### Result # #####  
#####. ## ##### # #####  
##### # # #####
```

```
function MyFunction(const S: string): string;  
begin  
  Result := S + ' Добавим что-нибудь';
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
Result := Result + ' и ещё что-нибудь!';
Result := Result + ' И ещё немножко!';
end;
```

```
##### (MyFunction # #####) ###
#####, ##### ##### #####. ##
##### # ## #####, ##### # #####
"#####", ##### # #####, ##### # #####
#####. ##### ##### Result, # #####, #####
##### #####. #####
```

```
#####, ##### ##### #####. ## ###
##### # #####, ##### #####, ##### #
##### ##### ( ) (##### # ##### # #####
#####) - ##### ##### # #####
#####. #####:
```

```
function ReadIntegersUntilZero: string;
var
  I: Integer;
begin
  Readln(I);
  Result := IntToStr(I);
  if I <> 0 then
    Result := Result + ' ' + ReadIntegersUntilZero();
end;
```

```
##### Exit ##### #####
## ####, ### ##### end;. ##### Exit #####
### #####, ### #####, ##### Result.
### ## ##### Exit(X), #####
##### # ##### — ###
##### return X # -#####.
```

```
function AddName(const ExistingNames, NewName: string): string;
begin
  if ExistingNames = '' then
    Exit(NewName);
  Result := ExistingNames + ', ' + NewName;
end;
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
##### #####, ### ##### ##### ##### #####
"#####" (#####). ##### ##### ##### ##### #####
#####. ##### ### ##### ##### ##### #####
##### ##### (#####, ##### ##### #####), # ## #####
##### ##### #####. #####:
```

```
var
```

```
    Count: Integer;
    MyCount: Integer;
```

```
function CountMe: Integer;
```

```
begin
```

```
    Inc(Count);
    Result := Count;
```

```
end;
```

```
begin
```

```
    Count := 10;
    CountMe; // результат функции будет отброшен, однако функция
              выполняется, Count станет равен 11.
    MyCount := CountMe; // запоминаем результат выполнения функции будет,
    Count теперь 12.
```

```
end.
```

2.3. ##### (if)

```
##### if .. then ### if .. then .. else ##### #####
###, ##### ##### ##### #####. # ##### ## C-##### #####, #
##### ### ##### ##### ##### ##### # #####.
```

```
var
```

```
    A: Integer;
    B: boolean;
```

```
begin
```

```
    if A > 0 then
        DoSomething;
```

```
    if A > 0 then
```

```
        begin
```

```
            DoSomething;
            AndDoSomethingMore;
```

```
        end;
```

Object Pascal ###
#####

```
if A > 10 then
  DoSomething
else
  DoSomethingElse;
```

// идентично предыдущему примеру

```
B := A > 10;
if B then
  DoSomething
else
  DoSomethingElse;
end;
```

else ##### # ##### if. #####
##:

```
if A <> 0 then
  if B <> 0 then
    AIsNonzeroAndBToo
  else
    AIsNonzeroButBIsZero;
```

if ##### begin ... end;

if ##### else - ##### A ## B - #,

#####.

```
if A <> 0 then
begin
  if B <> 0 then
    AIsNonzeroAndBToo
  else
    AIsNonzeroButBIsZero;
end;
```


2.4. #####, ##### # ##### (#####)

and , or , not ,
xor . ## ##### # ##### # ## , ##### #
. ##### ## , ## ##### xor , #####
"#####". ## #####
(boolean), # ##### ##
boolean. ## ##### ## , # ## ,
(integer, byte ##), # ##### ##
.

(#####) - #####
#####: = , <> , > , < , <= , >= , ##### . #####
, ## # ##### #-##### , # #####
"#####" A = B (# ## , ##
A == B). ##### #
:= .

(##) ##### , ##
. ##### , #####

.

#####:

```
var
  A, B: Integer;
begin
  if A = 0 and B <> 0 then ... // так делать НЕЛЬЗЯ
```

, ## #####
and # ##### (0
and B) - ##### , #####
"#####" # ##### A =
(0 and B) . # # ##### type mismatch , #
A = (0 and B) #
0 .

#####:

Object Pascal ###
#####

var

A, B: Integer;

begin

if (A = 0) and (B <> 0) then ...

"##### (short-circuit evaluation)" - #####
#####, ##### ## #####, #####-#####
#####. #####:

if MyFunction(X) and MyOtherFunction(Y) then...

- ##### MyFunction(X) #####.
 - # ##### MyFunction(X) ##### false, ### #####, ## ##
- ##### ## ## #####, ###
false and что-нибудь ## ##### false. #####
MyOtherFunction(Y) ##### ## #####.
 - ##### # # ##### or. # #####, ##### ##
#####, ## ##### true ##### ##
true, ##### ## ##### # ##
#####.
 - ### ##### ##### ##### #####:
-

if (A <> nil) and A.IsValid then...

A
nil. ##### nil ## #####, ##### #
"#####" #####. ## #####
(null pointer).

2.5. ##### ## ##### (##### case)

case .. of .. end.

case SomeValue of

```
##### # # ##### Object Pascal ###
#####
```

```
0: DoSomething;
1: DoSomethingElse;
2: begin
    IfItsTwoThenDoThis;
    AndAlsoDoThis;
end;
3..10: DoSomethingInCaseItsInThisRange;
11, 21, 31: AndDoSomethingForTheseSpecialValues;
else DoSomethingInCaseOfUnexpectedValue;
end;
```

```
##### else ##### (# ##### default # C-#####).
# #####, ##### ##### ##### ##### ## #####
# ##### ## ##### # ## ##### else, ## #####
##### ## ##### case # #####.
```

```
##### #-##### ##### ##### case # #####
##### switch # #####. #####, ## case #
##### ## ##### #####, #.#.
### #####, #####
break # #####. #####
##### case # #####
#####.
```

2.6. ##### # #, ##### # #####

```
##### (enumerated) # #####
#####. #####, ## ##### # ## #####
### ##### # # ##### :)
```

```
type
    TAnimalKind = (akDuck, akCat, akDog);
```

```
#####, ## ##### #####
##### #, ##### ak = ##### "Animal Kind".
### #####, ## ## #####
# ##### unit-#. #####, #
##### ak ##### ## #####
#####.
```

Object Pascal ###
#####



```
##### # ##### ## ##### # #####  
#####. ##### ##### # ##### unit-## #####  
##### #####. #####, #####  
##### #####, ## ##, #####, ##### ## ##  
##### ## ##### # #####.
```



```
##### ##### ##### #### ##### #  
##### ##### # #####  
{ $scopedenums on }. # #####  
##### # ## ##### ## ##  
#####: TAnimalKind.akDuck. # #####  
##### # ##### ak, # #####  
##### Duck, Cat, Dog. #####  
##### ##, ## ##### # C#.
```

```
##### ## ##, ## ## ##  
# #####. ## ##, ## ##  
#####, ##### Ord(MyAnimalKind), #####  
##### # #####. #####  
## ##### TAnimalKind(MyInteger) # ##### #  
##### ##. # #####  
## #####, ## MyInteger ##### ## 0 ##  
Ord(High(TAnimalKind))).
```

```
##### ## ## ## ##  
##### ##:
```

type

```
TArrayOfTenStrings = array [0..9] of string;  
TArrayOfTenStrings1Based = array [1..10] of string;  
  
TMyNumber = 0..9;  
TAlsoArrayOfTenStrings = array [TMyNumber] of string;  
  
TAnimalKind = (akDuck, akCat, akDog);  
TAnimalNames = array [TAnimalKind] of string;
```

```
## ## ## ## ## (## ## set-#, ## ##  
##### ##):
```

type

```
TAnimalKind = (akDuck, akCat, akDog);
TAnimals = set of TAnimalKind;
var
  A: TAnimals;
begin
  A := [];
  A := [akDuck, akCat];
  A := A + [akDog];
  A := A * [akCat, akDog];
  Include(A, akDuck);
  Exclude(A, akDuck);
end;
```

2.7. ##### (for, while, repeat, for .. in)

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
{$R+} // включаем проверку на диапазон величин, очень полезно для отладки
```

```
var
  MyArray: array [0..9] of Integer;
  I: Integer;
begin
  // инициализация
  for I := 0 to 9 do
    MyArray[I] := I * I;

  // отображение
  for I := 0 to 9 do
    WriteLn('Квадрат составляет ', MyArray[I]);

  // делает то же самое, что и предыдущий вариант
  for I := Low(MyArray) to High(MyArray) do
    WriteLn('Квадрат составляет ', MyArray[I]);

  // делает то же самое
  I := 0;
  while I < 10 do
    begin
      WriteLn('Квадрат составляет ', MyArray[I]);
      I := I + 1; // это идентично "I += 1" или "Inc(I)"
    end;

  // делает то же самое
```

Object Pascal ###
#####

```
I := 0;
repeat
  WriteLn('Квадрат составляет ', MyArray[I]);
  Inc(I);
until I = 10;

// делает то же самое
// обратите внимание, тут переменная I перечисляет значения элементов
// массива, а не его индексы
for I in MyArray do
  WriteLn('Квадрат составляет ', I);
end.
```

#####:

- #####, ### ##### while # repeat ####
"#####" # #####, ### ##### "
#: # while .. do #####,
#####, # repeat .. until - #####,
#####. #####, ### ##### #: #
repeat, ##### # # #, # # #. #####
repeat ##### # # #.
- ##### for I := .. to .. do ... ##### # C-#####
for. ### #, ### #, #####
#####/##### # #. #
for ##### # # #
(### #). #####
downto ##### to, ##### #
#####.
- # #####, ### ##### #, #
#. #, ##### #
#, # #.
- ##### #, #####
(# # - I) #####
Break
Exit. ### for I in .. do .. ##### # # # foreach

#####:
- ## ##### (##. #####).

```
##### ##### # ##### Object Pascal ###
#####
```

- ## ##### ##### ### ##### #####:

```
var
  AK: TAnimalKind;
begin
  for AK in TAnimalKind do...
```

```
# ## ##### ##### ### #####: 
```

```
var
  Animals: TAnimals;
  AK: TAnimalKind;
begin
  Animals := [akDog, akCat];
  for AK in Animals do ...
```

```
# # ### ## ##### ## #####, ##### generic-#,
#####, TObjectList ### TFPGObjectList.
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
uses
  SysUtils, FGL;
```

```
type
  TMyClass = class
    I, Square: Integer;
  end;
  TMyClassList = {$ifdef FPC}specialize{$endif}
  TFPGObjectList<TMyClass>;
```

```
var
  List: TMyClassList;
  C: TMyClass;
  I: Integer;
begin
  List := TMyClassList.Create(true); // значение true означает, что
  List владеет всеми дочерними объектами
  try
    for I := 0 to 9 do
      begin
        C := TMyClass.Create;
```

```
##### # ##### Object Pascal ###
#####
```

```
    C.I := I;
    C.Square := I * I;
    List.Add(C);
end;

for C in List do
    WriteLn('Квадрат ', C.I, ' составляет ', C.Square);
finally
    FreeAndNil(List);
end;
end.
```

```
## ## ## #####, #####
#####. ## ## #####
##### :)
```

2.8. ##### #

```
### ##### # ##### Write ###
WriteLn. ## ##### # #####
#####.
```

```
### "#####" ##### # #####, ### #####
#####, ##### ##### #####. ### ##
##### # #####. ##### ##,
##### ##### ##### ## #####
#####.
```

```
WriteLn('Hello world!');
WriteLn('Можно вывести целое число: ', 3 * 4);
WriteLn('Отформатировать его: ', 666:10);
WriteLn('А также вывести число с плавающей запятой: ', Pi:1:4);
```

```
#####
LineEnding (## ##### FPC RTL). (Castle Game Engine #####
##### NL). # ##### ## HTML # #####
##### # ##### (\) ## #####
##### # #####, #####:
```

```
WriteLn('Первая строка.\nВторая стока.');// НЕВЕРНЫЙ пример
```

```
##### # ##### # ##### Object Pascal ###
#####
```

```
#####. ##### ##### ##### #####: 'Моё целое
число ' + IntToStr(MyInt) + ', а значение числа пи составляет
' + FloatToStr(Pi).
```

```
# #####. ### #####. ##### #####
### XxxToStr # ##### (#####, FormatFloat), ### #####
#####
```

```
# #####. ### ##### ##### #####. #####
##### (#####, #####) # #####
##### StrToInt, StrToFloat # #####
(#####, StrToIntDef).
```

```
# #####. ##### ##### XxxToStr # #####
#####.
```

- ##### Format ##### # ##### Format('%d %f %s', [MyInt, MyFloat, MyString]). ### ##### sprintf # C-#####
#####. ### ##### # ##### placeholder-# #####
#####. ### placeholder-# #####
#####, ##### ## #####, ##### %.4f ### #####
4

```
# #####. ##### ## #####  
#####. ##### ##, #####, #####,  
#####
```

```
# #####. ## "#####"  
#####. #####  
#####  
##### (##  
##### array of const). #####  
##### Format, ## ##### #  
###, ###
```

```
# #####. ##### ##, #####-#####  
#####. ##### ## placeholder-# #  
EConvertError, #####, #####, ## segmentation  
fault (##### SIGSEGV).
```

- # ##### WriteStr(TargetString, ...) ##
Write(...), # ##### - #####
TargetString.

```
#####  #####  # ##### Object Pascal ###
#####
```

```
# #####. ### ##### ##### ### ##### Write, #
### ##### "#####" ##### ### #####,###
##### Pi:1:4.
# #####. ##### ##### ##### #####
"#####", #.#. ## ##### ##### #####.
##### ## ##### # #####, #####, #####
#### ##### MyStringFormatter(...) ##### ## #####
#####, ### Pi:1:4. ##### ## #####, # ##### #-# #####, ### ##
##### ##### ## ##### # ##### #####, #####
##### ## #####.
```

3. ##### (Unit-#)

```
Unit-# ##### ##### ##### ##### # ##### (#####
#####, ##### ##### #####), ### ##### unit-### #
#####. ### ##### # ##### # #####. ### #####
##### interface, ### ##### unit-## # #####
#####, ##### # #.#, # ##### implementation, ### #####, ### ##
#####. Unit MyUnit ##### myunit.pas (#####
##### ## ##### # ##### .pas).
```

```
unit MyUnit;
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
```

```
interface
```

```
procedure MyProcedure(const A: Integer);
function MyFunction(const S: string): string;
```

```
implementation
```

```
procedure MyProcedure(const A: Integer);
begin
  WriteLn('A + 10 составляет: ', A + 10);
end;

function MyFunction(const S: string): string;
begin
  Result := S + 'строки управляются автоматически';
end;
```

```
#####  #####  # ##### Object Pascal  ###  
#####
```

```
end.
```

```
#### ##### ##### ##### ##### ##### # #### ##### ####  
myprogram.lpr ( lpr = Lazarus program file; # Delphi ##### .dpr ).  
##### ##, ##### # ##### ##, ##, #####  
##### ##### .pas ### ##### ##. ###  
unit-## ##### ##### .pp. ##### # # #####  
##### .pas ### unit-## # .lpr ### FPC/Lazarus #####.
```

```
##### ##### ##### unit # ##### ##### ##### uses :
```

```
program MyProgram;
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}  
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
uses MyUnit;
```

```
begin
```

```
  WriteLn(MyFunction('Примечание: '));
```

```
  MyProcedure(5);
```

```
end.
```

```
Unit ##### ##### ##### initialization # finalization. # ###  
##### ##, ##### ## # ##### # #####  
#####, #####.
```

```
unit initialization_finalization;
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
```

```
interface
```

```
implementation
```

```
initialization
```

```
  WriteLn('Hello world!');
```

```
finalization
```

```
  WriteLn('Goodbye world!');
```

```
end.
```

3.1. ##### unit-###

unit-# ##### #
unit-#. ##### unit ##### # ##### interface ### ##### #
implementation. ##### #
(#####, ####...), ##### # unit-#.
- ##### unit
implementation, ## ##### #
implementation ##### unit-#.

unit AnotherUnit;

{ \$ifdef FPC } { \$mode objfpc } { \$H+ } { \$J- } { \$endif }

interface

uses

Classes;

{ Тип класса "TComponent" определён в unit-e Classes.

По этому необходимо использовать uses Classes, как видно выше. }

procedure DoSomethingWithComponent(**var** C: TComponent);

implementation

uses

SysUtils;

procedure DoSomethingWithComponent(**var** C: TComponent);

begin

{ Процедура FreeAndNil определена в unit-e SysUtils.

Поскольку мы лишь ссылаемся на её имя в разделе implementation,
вполне допустимо использовать SysUtils в секции "implementation". }

FreeAndNil(C);

end;

end.

(cyclic reference) #
interface. ##. ### unit-# ## ##### #
interface. ##### #
##, ###
##, ##### "#####" ##### interface unit-#, #####

```
#####  #####  # ##### Object Pascal  ###
#####
```

```
"#####" ### unit-#, ##### # uses # ##### interface. # ##### ###
##### #####, ### ##### #####
##### # ##### ##### #####
##### #####. # ##### #####
##### Makefile ### #####, # #####
##### #####, #####
#####.
```

```
##### unit-### #####
## ### "#####" ##### # implementation. ##### ### A
##### B # interface, # ##### unit B ##### A # implementation.
```

3.2. ##### # ##### unit-#

```
##### unit-# ##### #####. #####
##### ##### # #####, #####
#####, ## ## #####. # #####, #####
unit # ##### uses "##### ## ##", #.#. #####
##### # ##### #####
unit-### #####.
```

```
#####, ##### unit #####
#####, # ##### MyUnit.MyIdentifier. ###
#####, #####
MyUnit ##### unit-#. ### #####
##### unit-## # ##### uses, #####
## ## #####.
```

```
program showcolor;
```

```
{ $ifdef FPC } { $mode objfpc } { $H+ } { $J- } { $endif }
{ $ifdef MSWINDOWS } { $apptype CONSOLE } { $endif }
```

```
// unit-ы Graphics и GoogleMapsEngine определяют свои типы, которые
называются одинаково - TColor.
```

```
uses Graphics, GoogleMapsEngine;
```

```
var
```

```
{ Это сработает не так, как ожидается, поскольку TColor
определяется последним unit-ом в списке - GoogleMapsEngine. }
// Color: TColor;
{ А так будет правильно. }
```

```
#####  
##### # ##### Object Pascal ###  
#####
```

```
Color: Graphics.TColor;  
begin  
Color := clYellow;  
WriteLn(Red(Color), ' ', Green(Color), ' ', Blue(Color));  
end.
```

```
# ##### unit-## #####  
uses: ## - # ##### interface, ## - # implementation. #####  
##### # ##### #: "##### unit-# #####  
#####" # #####, ## # #####, ## unit-  
# ##### implementation #####  
unit-## ##### interface. ##### ##  
##### interface ##### unit-#  
unit-# ##### interface. ##  
#####  
#####  
#####:
```

```
unit UnitUsingColors;
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
```

```
// НЕВЕРНЫЙ пример
```

```
interface
```

```
uses Graphics;
```

```
procedure ShowColor(const Color: TColor);
```

```
implementation
```

```
uses GoogleMapsEngine;
```

```
procedure ShowColor(const Color: TColor);
```

```
begin
```

```
    // WriteLn(ColorToString(Color));
```

```
end;
```

```
end.
```

```
Unit Graphics (## ##### Lazarus LCL) ##### TColor.  
## ##### # unit-#, ##, ##
```

```
#####  #####  # ##### Object Pascal  ###  
#####
```

```
##### # ##### Interface ##### ShowColor ## #####. #####  
# ##, ## unit  GoogleMapsEngine ##### ##### ## TColor ,  
##### ##### # ##### implementation, #####  
## ##### TColor # ##### implementation. ##.  
##### ##### ## ##### ##:
```

```
unit UnitUsingColors;
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
```

```
// НЕВЕРНЫЙ пример
```

```
// демонстрирующий, как предыдущий пример "видит" компилятор
```

```
interface
```

```
uses Graphics;
```

```
procedure ShowColor(const Color: Graphics.TColor);
```

```
implementation
```

```
uses GoogleMapsEngine;
```

```
procedure ShowColor(const Color: GoogleMapsEngine.TColor);
```

```
begin
```

```
    // WriteLn(ColorToString(Color));
```

```
end;
```

```
end.
```

```
# ##### ##### ##### #####: ##### #####  
implementation, ##### ##### TColor ## unit-# Graphics. ##  
##### #####, ##### GoogleMapsEngine # ##### interface ##  
unit-# Graphics. #####, ## ##### # #####  
unit-# UnitUsingColors, ## ## ##### ##.
```

```
unit UnitUsingColors;
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
```

```
interface
```

```
uses Graphics;
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
procedure ShowColor(const Color: TColor);

implementation

uses GoogleMapsEngine;

procedure ShowColor(const Color: Graphics.TColor);
begin
    // WriteLn(ColorToString(Color));
end;

end.
```

3.3. ##### unit-#

```
##### ##### ##### ##### ## ##### unit-# #
##### ## ##### unit. #. # ##### ##### unit-#
##### ##### ##### ##### # #####.
```

```
# ##### ##### ##-## ##### #####
##### # ##### unit-#. # ##### #####
"#####" #####-#### unit ### #####.
```

```
### ##### ##### ##### # #####
##### # ##### unit-#.
```

```
unit MyUnit;
```

```
{ $ifdef FPC } { $mode objfpc } { $H+ } { $J- } { $endif }
```

```
interface
```

```
uses Graphics;
```

```
type
```

```
{ Используем TColor из unit-a Graphics для определения TMyColor. }
TMyColor = TColor;
```

```
{ Как вариант, можно переопределить его под тем же именем.
  В таком варианте необходимо будет явно указать наименование unit-a,
  иначе получится несогласованное определение "TColor = TColor". }
TColor = Graphics.TColor;
```


Object Pascal ###
#####

```
TMyClass = class
  MyInt: Integer; // это "поле"
  property MyIntProperty: Integer read MyInt write MyInt; // это
  "свойство"
  procedure MyMethod; // это "метод"
end;

procedure TMyClass.MyMethod;
begin
  WriteLn(MyInt + 10);
end;
```

4.2. ##### (Inheritance), ##### (is), # ##### ##### (as)

#####

```
program MyProgram;

{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}

uses
  SysUtils;

type
  TMyClass = class
    MyInt: Integer;
    procedure MyVirtualMethod; virtual;
  end;

  TMyClassDescendant = class(TMyClass)
    procedure MyVirtualMethod; override;
  end;

procedure TMyClass.MyVirtualMethod;
begin
  WriteLn('TMyClass отображает MyInt + 10: ', MyInt + 10);
end;

procedure TMyClassDescendant.MyVirtualMethod;
begin
```

Object Pascal ###
#####

```
WriteLn('TMyClassDescendant отображает MyInt + 20: ', MyInt + 20);  
end;
```

var

```
C: TMyClass;
```

begin

```
C := TMyClass.Create;
```

try

```
C.MyVirtualMethod;
```

finally

```
FreeAndNil(C);
```

end;

```
C := TMyClassDescendant.Create;
```

try

```
C.MyVirtualMethod;
```

finally

```
FreeAndNil(C);
```

end;

end.

virtual.#####

override, #####
reintroduce, #####, #####
#####, #####.

is. #####

as.

program is_as;

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
```

```
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

uses

```
SysUtils;
```

type

```
TMyClass = class
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
    procedure MyMethod;
end;

TMyClassDescendant = class(TMyClass)
    procedure MyMethodInDescendant;
end;

procedure TMyClass.MyMethod;
begin
    WriteLn('Это MyMethod')
end;

procedure TMyClassDescendant.MyMethodInDescendant;
begin
    WriteLn('Это MyMethodInDescendant')
end;

var
    Descendant: TMyClassDescendant;
    C: TMyClass;
begin
    Descendant := TMyClassDescendant.Create;
    try
        Descendant.MyMethod;
        Descendant.MyMethodInDescendant;

        { производные классы сохраняют все функции родительского класса
          TMyClass, по этому можно таким образом создавать ссылку }
        C := Descendant;
        C.MyMethod;

        { так не сработает, поскольку в TMyClass не определён этот метод }
        //C.MyMethodInDescendant;
        { правильно записать следующим образом: }
        if C is TMyClassDescendant then
            (C as TMyClassDescendant).MyMethodInDescendant;

    finally
        FreeAndNil(Descendant);
    end;
end.
```

```
##### ##### ##### # ##### X as TMyClass, ##### #####
##### ##### ##### # ##### ##### TMyClass(X). #####
### ##### #####-##### ## ##### # #####
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
##### # ##### ##### X ## ##### ##### TMyClass. ## #####
##### TMyClass(X) ##### ## ##### ##### ## #####, #####
##### #####, ## X ##### ##### ##### TMyClass,
#####, ## ## ##### ## ##### ## ##### # ##### ##### is:
```

```
if A is TMyClass then
  (A as TMyClass).CallSomeMethodOfMyClass;
// вариант ниже - работает незначительно быстрее
if A is TMyClass then
  TMyClass(A).CallSomeMethodOfMyClass;
```

4.3.

```
##### (Properties) ##### "#####" (####. #####. syntax
sugar - #####, ##### #####, ##### #####
## ##### ## #####, ## ##### #####
##### ## #####) ##### ##### # #####:
```

1. ##### #-# ##### ##### ## #### (##### ##### #
#####), ## ##### ##### #####
(getter) # ##### (setter). ##### #####
- ##### #####
###, ##### #####.
 2. ##### #-# ##### ##### ## ####, ## ##### ##### -
#####, ##### ## ##### ## #####.
-

type

```
TWebPage = class
private
  FURL: string;
  FColor: TColor;
  function SetColor(const Value: TColor);
public
  { Значение URL невозможно установить напрямую.
    Следует вызвать метод вроде Load('http://www.freepascal.org/'),
    для загрузки страницы и установки значения этого свойства. }
  property URL: string read FURL;
  procedure Load(const AnURL: string);
  property Color: TColor read FColor write SetColor;
end;
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
procedure TWebPage.Load(const AnURL: string);
begin
    FURL := AnURL;
    NetworkingComponent.LoadWebPage(AnURL);
end;

function TWebPage.SetColor(const Value: TColor);
begin
    if FColor <> Value then
    begin
        FColor := Value;
        { Например, требовать обновления класса, каждый раз,
          когда изменяется значение его цвета:
        Repaint;
        { Ещё пример: обеспечить чтобы нечто изменялось синхронно
          с установкой цвета, например }
        RenderingComponent.Color := Value;
    end;
end;
```

```
##### #####, ### #####, ##### #####
### #####, ##### #####/##### (#####
##### private # ##### ##### property #
##### f (## field - ####) #####
### #####. # #####, ##### Color #####
setter-##### SetColor. #####, ### ##### Color
##### ## private #### FColor. ##### ##
####, #####, #####, ### ##### getter ###
setter - ### # #####, ### # # #####
#####.
```

```
### #####:
```

1. ##### ## #### #####, # ### (# #####, ### #
getter).
2. ##### ## #### #####, # ### (# #####, ###
setter).

```
##### ## ##, ##### # #####  
##### # ##### # #####. #####, #####  
##### Integer ##### Integer, ###  
##### (#####), ##### Integer.
```

#, ##### "getter" # "setter" - ##### # ##
(#####).
#, #####
#####-##### :

- ##### getter ## ##### (#####,
/ # #####). ##
(#####-
:)) - ##### #

#####.

#####, ## ##### getter #####
#####-
#. ##
#####, ## # "getter".

- ##### setter #####, #####
getter #####. ## #####
"setter", # # #, #####
exception. #####
#####. ##### # #, #####
MyClass.MyProperty := 123; ##### #
#####, ## MyClass.MyProperty = 123.

- #####, ##### ## #####
#####. #####
####, ##### - ##### ##, ##

#####. #####
#####-##, #####, # ##.

- ####, # ##### private, ##### - ##### #
##.

- #####, #####, #####
#####, ## #. #####
:)



#, ##
unit-#. ## : ##


```
##### ##### # ##### Object Pascal ###
#####
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

uses

SysUtils;

type

TMyClass = **class**
 procedure MyMethod;
end;

procedure TMyClass.MyMethod;

begin

if Random > 0.5 **then**
 raise Exception.Create('Вызываем exception!');

end;

var

C: TMyClass;

begin

 Randomize;
 C := TMyClass.Create;
 try
 C.MyMethod;

finally
 FreeAndNil(C)

end;

end.

```
##### #####, ### ##### finally ##### ##### # #####, ####
##### ##### ##### Exit (## #####, ##### ###
#####), ##### Break ### Continue (##### #).

```

4.5.

```
### # # ##### #####-##### #, # #####
##### ##### # "#####" ##### /
##### / #####.

```

```
##### #####:

```

public

```
##### ## ##### #, ##### # # ##### unit-##.

```

Object Pascal ###
#####

private

#####.

protected

#####.

private # protected, ##### ##, ## #####
#####. ## # ##### unit-# ##### #####, # #####
private # protected. ##### ## #####, #####
#####. # ##### ## #####
strict private ## strict protected ##
#####, ##### ## #####
#####. ##### ## ##### # ##### [8.1](#),
«##### private # strict private».

#####, ##### ## ##### ##, ## ## #####
public. #####, ##### ##
{ $\$M+$ }, #####, #####
{ $\$M+$ }, ## # ##### TPersistent, #####
TComponent, ##### TPersistent. ##
published, #####
public, ##### # #####
(stream) #####.

#####, ## ##### ## ## ## # #####
published - ## ##### ## ## ## (stream), # ##
#####, ##### ##, #####. ## ##
##, ## ## ##-##
##, ## ##### public.

4.6.

#####, ## ## class
TObject.

5.

5.1.

##, ## #####
#####. #####

Object Pascal ###
#####

FPC -gl -gh, ##### (#####
##. https://castle-engine.io/manual_optimization.php#section_memory).

#, ## ##
(raised exceptions). ## ##, ## ##
(# ## ##, ## ##
##), ## ##
#####.

5.2.

FreeAndNil(A) ## unit-# SysUtils #####
#####. ## ##, ## A ##### nil, # ## —
(destructor), # ##### A # nil. #####
##.

#####:

```
if A <> nil then  
begin  
  A.Destroy;  
  A := nil;  
end;
```

#####, ## ##, ##### FreeAndNil
##, ##### A ##### nil
destructor #####. ##
##, ## "#####" ## ##
##.

#, ## A.Free #####
#####:

```
if A <> nil then  
  A.Destroy;
```

##. ##### A, ## ## nil.

#, ## ## ##
#####, ##### ## nil. ## A.Free #####
##, ##### A

```
#####  #####  # ##### Object Pascal ###
#####
```

```
##### nil. #####, ##### Free ##### ##### ## ##### #####.
### ##### "#####"—# #####, ##### ##### Self
<> nil. ##### ##### ##### ##### ##-##### ##### (#.#. #
#####, ##### ## ##### ##### ##### # ## ##### ##### # #####
#####).
```

```
## ##### ##### ##### ##### FreeAndNil(A), ### #####, #
##### ## ##### ##### Free ### ##### Destroy.
##### ##### Castle Game Engine. ### ##### #####
#####, ### ## ##### ##### nil, ##### ## #####
##### #####.
```

5.3. ##### # #####

```
## #####, ##### ##### ## ##### ##-##
#####. ##### destructor, ### ##### constructor-#, #
##### ##, ##### ## ##### # constructor-# (#,
#####, # ##### ##### #####). ##### ##
###, ##### ##### ##### ##### #####. ##### #####
##### ##### ##### ##### nil, # #####
##### ##, ##### FreeAndNil(A).
```

```
#####:
```

```
uses SysUtils;
```

```
type
```

```
  TGun = class
```

```
  end;
```

```
  TPlayer = class
```

```
    Gun1, Gun2: TGun;
```

```
    constructor Create;
```

```
    destructor Destroy; override;
```

```
  end;
```

```
constructor TPlayer.Create;
```

```
begin
```

```
  inherited;
```

```
  Gun1 := TGun.Create;
```

```
  Gun2 := TGun.Create;
```

```
end;
```

```
destructor TPlayer.Destroy;
begin
  FreeAndNil(Gun1);
  FreeAndNil(Gun2);
  inherited;
end;
```

```
#####
##### TComponent,
##### "owner" (#####). #####
##### owner(#####) ##### owner-##. #####
##### # #####, #####, #.#.
### ##### # #####, #####.
##### # #####, #####.
#####:
```

```
uses SysUtils, Classes;
```

```
type
```

```
  TGun = class(TComponent)
  end;
```

```
  TPlayer = class(TComponent)
    Gun1, Gun2: TGun;
    constructor Create(AOwner: TComponent); override;
  end;
```

```
constructor TPlayer.Create(AOwner: TComponent);
```

```
begin
```

```
  inherited;
  Gun1 := TGun.Create(Self);
  Gun2 := TGun.Create(Self);
```

```
end;
```

```
##### override #####
constructor ## TComponent. ##, # #####, ##
##### constructor-#. #####, ##-### —#####
##### constructor # ##### reintroduce. #####
#####, ## ## #####, streaming,
### ##### constructor, ##
##### ## #####
#####.
```

```
##### # ##### # ##### Object Pascal ###
#####
```

```
##### #, ### ##### ##### nil # ##### owner-#.
##### # # ##### # owner-##### # # #
##### #. ### ##### #, #####
##### # # ##### TComponent, ##### # #
##### # #. ##### # # #
##### # #: ManualGun := TGun.Create(nil);.
```

```
### # ##### # #####
##### — ##### OwnsObjects (##### # # true)
#####-#####, #, ### TFPGObjectList ### TObjectList. #. #
##### # #:
```

```
uses SysUtils, Classes, FGL;
```

```
type
```

```
  TGun = class
  end;
```

```
  TGunList = {$ifdef FPC}specialize{$endif} TFPGObjectList<TGun>;
```

```
  TPlayer = class
    Guns: TGunList;
    Gun1, Gun2: TGun;
    constructor Create;
    destructor Destroy; override;
  end;
```

```
constructor TPlayer.Create;
```

```
begin
```

```
  inherited;
  // Вообще говоря, параметр OwnsObjects и так true по умолчанию
  Guns := TGunList.Create(true);
  Gun1 := TGun.Create;
  Guns.Add(Gun1);
  Gun2 := TGun.Create;
  Guns.Add(Gun2);
```

```
end;
```

```
destructor TPlayer.Destroy;
```

```
begin
```

```
  { Здесь достаточно освободить сам список.
    Он сам автоматически освободит всё содержимое. }
  FreeAndNil(Guns);
```

```
#####  #####  # ##### Object Pascal ###
#####
```

{ Таким образом нет нужды освобождать Gun1, Gun2 отдельно. Правда, хорошей

практикой будет теперь установить значение "nil" соответствующим значениям

ссылок на них, поскольку мы знаем, что они освобождены.

В этом простом классе с простым destructor-ом, очевидно, что к ним не произойдёт доступа, однако в случае сложных destructor-ов это может оказаться полезно.

Альтернативно, можно избежать объявления Gun1 и Gun2 отдельно и использовать напрямую Guns[0] и Guns[1] в коде.

Можно также создать метод Gun1, который возвращает ссылку на Guns[0]. }

```
Gun1 := nil;
Gun2 := nil;
inherited;
end;
```

```
#####, ### ##### owner-## #####-##### ##### (### #####
#####) #, # ##### ##### ##### # #####
##### #####, #####. #####
###-#### ## ##### ### ##### ##### Extract, #####
### #####, ### # ##### ##### #####.
```

```
#####, # Castle Game Engine ### ##### TX3DNode
##### TX3DNode # #####
children. ##### X3DNode ##### TX3DRootNode #, # #####,
##### owner-## ##### TCastleSceneCore. #####
##### owner-# — #####
##### OwnsXxx.
```

5.4. Free notification

```
#### ## #####, ###,
### ## ##### # ## # #####. #####
#### ##, ##### "##### pointer-##. #####
##### # #####, #####. ###
##### # runtime #####, #####
("#####") — # #####, #####
#####.
```



```
##### # ##### Object Pascal ###
#####
```

```
# ##### # ##### FreeAndNil #####
##### nil #### — #####
##### # #####. #####:
```

```
var
```

```
Obj1, Obj2: TObject;
```

```
begin
```

```
Obj1 := TObject.Create;
```

```
Obj2 := Obj1;
```

```
FreeAndNil(Obj1);
```

```
// что произойдёт, если попытаться получить доступ к классу Obj1 или
Obj2?
```

```
end;
```

```
1. # ##### Obj1 ##### nil. ####
##### # ####, ####
##### if Obj1 <> nil then ... #####
##### ####, #####:
```

```
if Obj1 <> nil then
```

```
WriteLn(Obj1.ClassName);
```

```
##### # ##### nil ## ##### #
#####. #####, #### #
##### Obj1 <> nil, # ##### # Obj1,
#####.
```

```
## ## ##### # ####, ##
##-##### ##### #
#####.
```

```
2. ##### # Obj2 —####. ## nil, #####
#####. ##### #- nil #####
##### # —### #
##### access violation, # #####-##.
```

```
# #####:
```

- ##### - ##### #. ##
#####, ####

Object Pascal ###
#####

#####. ##### # ##### TCar ##### wheel, ##### ## #####
Twheel, ## ##### ## wheel #####, #####
car, # ## car ##### ## wheel #####
destructor. ## ## ##### #####. # #####
#####, # ##### ##### ##### # ##, ## #
#####.

- # ##### ##, ##### ##### ##### ##### Obj1, #####
Obj2 ##### nil. # #####
#####.
- #####, ##### ##### ##### #####
TComponent ## "free notification" (#####
##). ##### ##### ##### #####
#####, # ##### ##### ##
nil.

#####. ##### ##
#####, #####, ##### ##
#####, # ##, ##### ##### # #####.

TComponent.
FreeNotification ,
RemoveFreeNotification, # override Notification.

constructor-## / destructor-## # setter-##. ##### ##
#####, ## ##### #####, #####
#####.

type

TControl = **class**(TComponent)
end;

TContainer = **class**(TComponent)

private

FSomeSpecialControl: TControl;

procedure SetSomeSpecialControl(**const** Value: TControl);

protected

procedure Notification(AComponent: TComponent; Operation:
TOperation); **override**;

public

destructor Destroy; **override**;

```
#####  #####  # ##### Object Pascal ###  
#####
```

```
    property SomeSpecialControl: TControl  
        read FSomeSpecialControl write SetSomeSpecialControl;  
end;  
  
implementation  
  
procedure TContainer.Notification(AComponent: TComponent; Operation:  
    TOperation);  
begin  
    inherited;  
    if (Operation = opRemove) and (AComponent = FSomeSpecialControl) then  
        { установить значение nil для SetSomeSpecialControl чтобы всё  
        аккуратно подчистить }  
        SomeSpecialControl := nil;  
end;  
  
procedure TContainer.SetSomeSpecialControl(const Value: TControl);  
begin  
    if FSomeSpecialControl <> Value then  
        begin  
            if FSomeSpecialControl <> nil then  
                FSomeSpecialControl.RemoveFreeNotification(Self);  
            FSomeSpecialControl := Value;  
            if FSomeSpecialControl <> nil then  
                FSomeSpecialControl.FreeNotification(Self);  
        end;  
end;  
  
destructor TContainer.Destroy;  
begin  
    { Установить значение nil для SetSomeSpecialControl, чтобы запустить  
    notification про освобождение памяти }  
    SomeSpecialControl := nil;  
    inherited;  
end;
```

6. Run-time library

6.1. ##### / ##### #

```
### ##### / ##### # ##### ##### ## ##### #####  
TStream. # ##### ##### ##### ##### #####  
### TFileStream, TMemoryStream, TStringStream # #.
```

```
#####  #####  # ##### Object Pascal ###
#####
```

```
{ $ifdef FPC } { $mode objfpc } { $H+ } { $J- } { $endif }
{ $ifdef MSWINDOWS } { $apptype CONSOLE } { $endif }
```

```
uses SysUtils, Classes;
```

```
var
```

```
    S: TStream;
```

```
    InputInt, OutputInt: Integer;
```

```
begin
```

```
    InputInt := 666;
```

```
    S := TFileStream.Create('my_binary_file.data', fmCreate);
```

```
    try
```

```
        S.WriteBuffer(InputInt, SizeOf(InputInt));
```

```
    finally
```

```
        FreeAndNil(S);
```

```
    end;
```

```
    S := TFileStream.Create('my_binary_file.data', fmOpenRead);
```

```
    try
```

```
        S.ReadBuffer(OutputInt, SizeOf(OutputInt));
```

```
    finally
```

```
        FreeAndNil(S);
```

```
    end;
```

```
    WriteLn('Из файла прочитано целое число: ', OutputInt);
```

```
end.
```

```
# Castle Game Engine ##### Download ### #####
#####, ##### ##### # ##### URL. ### #####
#####, #####, ##### ## HTTP # HTTPS, Android assets # #####
#####. #####, ### ##### ##### ##### ##
##### ##### (##### # ##### data) #####
##### ApplicationData ### ##### URL castle-data:/xxx URL.
#####:
```

```
EnableNetwork := true;
```

```
S := Download('https://castle-engine.io/latest.zip');
```

```
S := Download('file:///home/michalis/my_binary_file.data');
```

```
##### # ##### Object Pascal ###  
#####
```

```
##### ##### ##### ##### ##### #####  
TCastleTextReader ## CastleClassUtils.## #####  
API, ## ##### TStream. URL #####  
##### TCastleTextReader, ##### TStream  
#####.
```

```
Text := TCastleTextReader.Create('castle-data:/my_data.txt');  
while not Text.Eof do  
  WriteLnLog('NextLine', Text.ReadLine);
```

6.2.

```
## #####  
##### generic ##### ## unit-# FGL. ##### TFPGList  
## ##### (##### record-# ## ##### object-#), #  
TFPGObjectList ## #####. # Castle Game Engine:  
##### TGenericStructList ## CastleGenericLists  
## ##### record-## ## ##### object-##. ## #####  
##### override ## ##### FPC.
```

```
##### ##### ##### ##### ##### ## #####:  
## #####-##### # ## API ##### #####, #####  
#####, #####, ##### # #.#. #####, ## #####  
##### (array of X, SetLength(X, ...)) #####  
## API ##### - ##### SetLength. #####  
##### TList ## TObjectList #####  
##### TObject # #####  
#####.
```

6.3. #####: TPersistent.Assign

```
#####  
##### TPersistent # ##### override ##  
Assign. #####, ##### Assign  
#####.
```

```
#####, #####  
##### Assign, #####  
#####, # # # #.
```

```
{ $ifdef FPC } { $mode objfpc } { $H+ } { $J- } { $endif }
{ $ifdef MSWINDOWS } { $apptype CONSOLE } { $endif }

uses
    SysUtils, Classes;

type
    TMyClass = class(TPersistent)
    public
        MyInt: Integer;
        procedure Assign(Source: TPersistent); override;
    end;

    TMyClassDescendant = class(TMyClass)
    public
        MyString: string;
        procedure Assign(Source: TPersistent); override;
    end;

procedure TMyClass.Assign(Source: TPersistent);
var
    SourceMyClass: TMyClass;
begin
    if Source is TMyClass then
    begin
        SourceMyClass := TMyClass(Source);
        MyInt := SourceMyClass.MyInt;
        // Xxx := SourceMyClass.Xxx; // добавить необходимые поля здесь
    end else
    { Вызываем inherited ТОЛЬКО если не получается вручную обработать
    Source }
        inherited Assign(Source);
    end;

procedure TMyClassDescendant.Assign(Source: TPersistent);
var
    SourceMyClassDescendant: TMyClassDescendant;
begin
    if Source is TMyClassDescendant then
    begin
        SourceMyClassDescendant := TMyClassDescendant(Source);
        MyString := SourceMyClassDescendant.MyString;
        // Xxx := SourceMyClassDescendant.Xxx; // добавить необходимые поля
        здесь
    end;
end;
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
{ ВСЕГДА вызываем inherited, чтобы TMyClass.Assign сама обработала
  все остающиеся поля. }
inherited Assign(Source);
end;

var
  C1, C2: TMyClass;
  CD1, CD2: TMyClassDescendant;
begin
  // тестируем TMyClass.Assign
  C1 := TMyClass.Create;
  C2 := TMyClass.Create;
  try
    C1.MyInt := 666;
    C2.Assign(C1);
    WriteLn('C2 состояние: ', C2.MyInt);
  finally
    FreeAndNil(C1);
    FreeAndNil(C2);
  end;

  // тестируем TMyClassDescendant.Assign
  CD1 := TMyClassDescendant.Create;
  CD2 := TMyClassDescendant.Create;
  try
    CD1.MyInt := 44;
    CD1.MyString := 'что-нибудь';
    CD2.Assign(CD1);
    WriteLn('CD2 состояние: ', CD2.MyInt, ' ', CD2.MyString);
  finally
    FreeAndNil(CD1);
    FreeAndNil(CD2);
  end;
end.
```

```
##### ##### ##### ##### override ##### AssignTo #
##### # ## ##### override ##### Assign # #####, # #####
#####.
```

```
##### ##### ##### # inherited ### ##### Assign. Inherited ##
TPersistent.Assign ##### ##### # #####, ##### ##
## ##### ##### ##### # ##### (### #####
##### AssignTo # ##### ## #####, # #####,
```

```
#####  #####  # ##### Object Pascal ###
#####
```

```
#### ##### ## ##### #####). # ##### #####, #### #####
##### ##### ##### ## #####, # ##### #### ##### Assign, ## #
##### ##### ##### ##### inherited ## TMyClass.Assign.
##. ##### #####.
```



```
##### #####, ### ##### ##### TPersistent ##
##### ##### ##### published, #####
##### ##### ##### # ##### (streaming) #####
## TPersistent #####. #####, ## ### ##### #
##### ##### # ##### published. #####
##### # ##### # ### ##### #####
##### # #####, ##### ##### #####
## public. ##### ##. ##### 4.5, «#####
#####».
```

7.

7.1. ##### (#####)

```
##### ##### (### ##### #####, #####, ##### # #.)
##### ##### (#####) ###-#####.
```

```
##### ##### ##### #####-##### ##### ##### (##### #
#####) ##### #####, # ##### ####, #####
#####, ### ###. ### #####, #####
##### ## ##### ##### #####
##### ##### # #####. #####, ## #####
#####. # #####, ##### (#####)
##### #, ### #####, ##### #####, #
##### #####, ##### ### #####.
```

```
##### ## ##### #####:
```

```
function SumOfSquares(const N: Integer): Integer;
```

```
function Square(const Value: Integer): Integer;
begin
    Result := Value * Value;
end;
```


Object Pascal ###
#####

```
var
  I: Integer;
begin
  Result := 0;
  for I := 0 to N do
    Result := Result + Square(I);
end;
```

.....
#####, # ##### Square ##### # ##### I:

.....

```
function SumOfSquares(const N: Integer): Integer;
var
  I: Integer;

  function Square: Integer;
  begin
    Result := I * I;
  end;

begin
  Result := 0;
  for I := 0 to N do
    Result := Result + Square;
end;
```

.....
—###

#. #####, ## #####, ##### #

#####.

7.2. ##### —### ## #####, ## ## ##### ## #####,
#####

#####

#####

- #####, ## ## #####, ## ## #####
(#.#. ## ## #####)

```
##### ##### # ##### Object Pascal ###
#####
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
function Add(const A, B: Integer): Integer;
begin
    Result := A + B;
end;
```

```
function Multiply(const A, B: Integer): Integer;
begin
    Result := A * B;
end;
```

```
type
    TMyFunction = function (const A, B: Integer): Integer;
```

```
function ProcessTheList(const F: TMyFunction): Integer;
var
    I: Integer;
begin
    Result := 1;
    for I := 2 to 10 do
        Result := F(Result, I);
    end;
```

```
var
    SomeFunction: TMyFunction;
begin
    SomeFunction := @Add;
    WriteLn('1 + 2 + 3 ... + 10 = ', ProcessTheList(SomeFunction));

    SomeFunction := @Multiply;
    WriteLn('1 * 2 * 3 ... * 10 = ', ProcessTheList(SomeFunction));
end.
```

-
- ##### ## #####: ### ##### # ##### ##### of object .
-

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
uses
    SysUtils;
```

```
type
```

```
##### ##### # ##### Object Pascal ###  
#####
```

```
TMyMethod = procedure (const A: Integer) of object;  
  
TMyClass = class  
  CurrentValue: Integer;  
  procedure Add(const A: Integer);  
  procedure Multiply(const A: Integer);  
  procedure ProcessTheList(const M: TMyMethod);  
end;  
  
procedure TMyClass.Add(const A: Integer);  
begin  
  CurrentValue := CurrentValue + A;  
end;  
  
procedure TMyClass.Multiply(const A: Integer);  
begin  
  CurrentValue := CurrentValue * A;  
end;  
  
procedure TMyClass.ProcessTheList(const M: TMyMethod);  
var  
  I: Integer;  
begin  
  CurrentValue := 1;  
  for I := 2 to 10 do  
    M(I);  
end;  
  
var  
  C: TMyClass;  
begin  
  C := TMyClass.Create;  
  try  
    C.ProcessTheList({$ifdef FPC}@{$endif} C.Add);  
    WriteLn('1 + 2 + 3 ... + 10 = ', C.CurrentValue);  
  
    C.ProcessTheList({$ifdef FPC}@{$endif} C.Multiply);  
    WriteLn('1 * 2 * 3 ... * 10 = ', C.CurrentValue);  
  finally  
    FreeAndNil(C);  
  end;  
end.
```

```
##### ##, ### #####  
### #####. ### #####. #### ### #####
```

```
##### # ##### # ##### Object Pascal ###  
#####
```

of object, ## ## ##### #####-##### ## ## ##, ####
8.3, «Class method» # ##### ##.

type

```
TMyMethod = function (const A, B: Integer): Integer of object;
```

```
TMyClass = class
```

```
  class function Add(const A, B: Integer): Integer
```

```
  class function Multiply(const A, B: Integer): Integer
```

```
end;
```

var

```
  M: TMyMethod;
```

begin

```
  M := @TMyClass(nil).Add;
```

```
  M := @TMyClass(nil).Multiply;
```

end;

```
# #####, # ##### ##### ##### #####  
@TMyClass(nil).Add, # ## ##### @TMyClass.Add.
```

- ##### ##### ## #####: ## ##### ## #####
is nested # #####, # ##### ##### { \$modeswitch
nestedprocvars} ## ##### #####. ##### ##. ##### 7.1,
«##### (#####) #####».

7.3. Generic-#

Generic-# - ## ##### ##### #####-#####. #####-#####
(##### - #####) ##### #####. #####
(#####,
#####, #####, #####...): ##### T, # ##### specialize
(#####) ## ##### ##### integer, ##### ##
string, ##### ## TMyRecord # #.

Generic-# # ##### generic-## # C++. ##
#####, ## ## "##### specialize, ##
(#####, ##### ## "#####",
#####, # #####
"#####-##### # generic ##### ##
"#####"). ##### ## #####
(##### ## #####) # #####

```
#####  #####  # ##### Object Pascal ###  
#####
```

#####. ##### ##### (integer, float) # ##### record,
class # #.#. ### specialize ##### generic-#.

```
uses SysUtils;
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}  
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
type
```

```
generic TMyCalculator<T> = class  
    Value: T;  
    procedure Add(const A: T);  
end;
```

```
procedure TMyCalculator.Add(const A: T);  
begin  
    Value := Value + A;  
end;
```

```
type
```

```
TMyFloatCalculator = {$ifdef FPC}specialize{$endif}  
TMyCalculator<Single>;  
TMyStringCalculator = {$ifdef FPC}specialize{$endif}  
TMyCalculator<string>;
```

```
var
```

```
FloatCalc: TMyFloatCalculator;  
StringCalc: TMyStringCalculator;
```

```
begin
```

```
FloatCalc := TMyFloatCalculator.Create;
```

```
try
```

```
FloatCalc.Add(3.14);
```

```
FloatCalc.Add(1);
```

```
WriteLn('Сложение величин типа Float: ', FloatCalc.Value:1:2);
```

```
finally
```

```
FreeAndNil(FloatCalc);
```

```
end;
```

```
StringCalc := TMyStringCalculator.Create;
```

```
try
```

```
StringCalc.Add('что-нибудь');
```

```
StringCalc.Add(' ещё');
```

```
WriteLn('Сложение величин типа String: ', StringCalc.Value);
```

```
finally
```

```
FreeAndNil(StringCalc);
```

```
#####  #####  # ##### Object Pascal  ###
#####
```

```
end;
end.
```

```
##### generic-## ## #####, ##### generic
##### # #####:
```

```
uses SysUtils;
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
{ Примечание: этот пример требует FPC 3.1.1 и не скомпилируется в FPC
3.0.0 или более ранних версиях. }
```

```
generic function Min<T>(const A, B: T): T;
```

```
begin
```

```
  if A < B then
```

```
    Result := A
```

```
  else
```

```
    Result := B;
```

```
end;
```

```
begin
```

```
  WriteLn('Min (1, 0): ', specialize Min<Integer>(1, 0));
```

```
  WriteLn('Min (3.14, 5): ', specialize Min<Single>(3.14, 5):1:2);
```

```
  WriteLn('Min (''a'', ''b''): ', specialize Min<string>('a', 'b'));
```

```
end.
```

7.4. Overload

```
##### (# #####, ##### # #####) # #####
##### ##, ## # #####. ##
##### #, ##### #, ##### #
##### # #####.
```

```
## ##### overload ##### FPC. ##, ##
## # ##### (# ##### # unit-) #, #
## # # # # #.
#####, ##### # # Foo(Integer) # Foo(string),
# ## # # # # Foo(Float), ##
##### Foo(Float) #, #####
##### # #####.
```

```
#####  #####  # ##### Object Pascal  ###
#####
```

```
#####  #####, ### #####  #####  # #####  #####  #####
#####  ##### overload.
```

7.5.

```
#####  #####  #####  #####  #####  #####  #####:
```

- ##### ## ##### ##### ##### (#####, ##### ## #####
#####, ### ##### ##### ##### #####),
- ##### ##### ##### ##### ##### #####,
- ### ##### ##### ##### #####.

```
#####  #####, #####  # #####  #####. # #####, #####
#####  #####  #####  #####  #####, #####  ## #####
#####  #####  #####  #####  #####  #####, ### #####, ### ##
#####  ##### "#####" #####  #####  #####  #####. ### #####, ## #####
"#####".
```

```
.....
unit PreprocessorStuff;
```

```
{ $ifdef FPC } { $mode objfpc } { $H+ } { $J- } { $endif }
```

```
interface
```

```
{ $ifdef FPC }
```

```
{ всё что идёт внутри данного условия ifdef определено только для FPC, а  
не других компиляторов (например, Delphi). }
```

```
procedure Foo;
```

```
{ $endif }
```

```
{ Определить константу NewLine. Это пример того, как "нормальный"  
синтаксис Паскаля "поломан" директивами предобработки.  
Если компилировать на Unix-системах (включая Linux, Android, Mac OS X),  
компилятор увидит следующее:
```

```
const NewLine = #10;
```

```
Если компилировать на Windows, компилятор увидит так:
```

```
const NewLine = #13#10;
```

```
Однако, на других операционных системах, код не скомпилируется,  
поскольку компилятор увидит следующее:
```



```
##### # ##### # ##### Object Pascal ###
#####
```

```
##### ## ##### #####, ## ##### ##### ## ## #####
{$ifdef UNIX}, {$ifdef MSWINDOWS} ##### # #####
##### (#####, #####). ##### #
#####. ## ##### ##### #####,
##### ##### -Fi ## ##### FPC, #####
##### ##### ##### #####. #####
##### ##### include ##### { $I my
platform_specific_implementation.inc}. ## #####
#####.
```

7.6. Record

Record #####. ##
#####, #####. # record-## ## ##
#####. ## ##### structure # C-#####.

```
##### { $modeswitch advancedrecords}, #####
##### # ##### # record # #####
#####, #####, ##### ## ##### #
##### ## record.
```

```
{ $ifdef FPC}
  { $mode objfpc} { $H+} { $J-}
  { $modeswitch advancedrecords}
{ $endif}
{ $ifdef MSWINDOWS} { $apptype CONSOLE} { $endif}
```

type

```
TMyRecord = record
public
  I, Square: Integer;
  procedure WriteLnDescription;
end;
```

procedure TMyRecord.WriteLnDescription;

begin

```
  WriteLn('Квадрат числа ', I, ' равен ', Square);
```

end;

var

```
A: array [0..9] of TMyRecord;
```

```
R: TMyRecord;
```

```
I: Integer;  
begin  
  for I := 0 to 9 do  
    begin  
      A[I].I := I;  
      A[I].Square := I * I;  
    end;  
  
    for R in A do  
      R.WriteLineDescription;  
    end.
```

```
# #####, # ##### ##### class, #  
## record, ##### ##### #####  
#####, ## ##### # #####. #####, record-# ## ##  
##### ##### ##### ##### ## #####  
#####:
```

- Record-## ## ##### constructor ## destructor. ## ##### ##
#####. ## ##### # ##### (## ##### "#####
#####"), ##### ##### ##### #####, ##### ##
string, ##### #####. ##### #####
#####, ##### #####
#####.
- #####, ##### ## record-## ##### # #####
#####.
- ##### (##### # ##### #####) ## record-##
#####: ##### C layout ##
packed record. ## ##### # #####
#####:

```
# ## #####, ##### # #####  
##### # ##### API, ##### ##  
##### ## record,  
  
# ## ##### ## ##,  
  
# ## ##### "#####" ##### (##  
##### # #####, ## ##  
#####).
```

- # record-## ##### ##### case, #####
union # C-##### # #####

```
#####  #####  # ##### Object Pascal  ###
#####
```

```
## ## #####, ## #####, # ##### ##
#####. ##### ##### #####
##### # #####. # ##, #####
##### "#####"
```

7.7. ##### object

```
#####-#####, # Turbo Pascal ### ##### # #####,
##### ##, ##### object. ## ##, ##
- ##### record # class.
```

- ##### object-# ##### / #####, # # #####
constructor / destructor.
- ## ## ##### # ##### record. #####
record ## object ## (pointer-##) ##
###-###, ##, #####. ## ##
#####, #####
##.
- ##### object-# ##### # #####, ##,
##. #####
#####: ##### object, #####
#####, ## ## constructor-#, #####.

```
# ##### objects-#
#. ##### class-# #####. # #
##### record-#
(##### advanced records). #####, ## object-#.
```

7.8. Pointer-#

```
# ##### pointer(#####) ## #####. #####
## ## TMyRecord # ^TMyRecord, # ##
##### pointer-# PMyRecord. # #####
##### # record:
```

type

```
PMyRecord = ^TMyRecord;
TMyRecord = record
```

```
#####  #####  # ##### Object Pascal  ###
#####
```

```
Value: Integer;
Next: PMyRecord;
end;
```

```
#####  #####  #####, ###  #####  #####  #####
#####  (###  PMyRecord  #####  #  #####  TMyRecord, #
TMyRecord  #####  #  #####  #####  PMyRecord). #####  #####
pointer ##  ###, #####  ###  ##  ###  #####, #  ###  #####, #####  ##  #####
#####  #  ###  ##  #####  #####  type.
```

```
#####  #####  #####  #####  #####  #####  #####  #####
New / Dispose, ###  (#####  #####, ##  ##  #####)
#####  GetMem / FreeMem. ###  #####  #  #####, ##  #####
pointer  #####, #####  #####  #####  #####  #####  ^  (#  #####
MyInteger := MyPointerToInteger^). #####  #####  (#####  pointer
##  #####  #####) #####  #  #####  #####-#####  @
(#####, MyPointerToInteger := @MyInteger).
```

```
#####  #####  #####  #####  ###  Pointer, #####  ##  #####  ##
#####  ###  #####  #  #####  void*  # C-#####  #####. ##  #####
##  #####, #  ###  #####  #####  ###  #  #####  #####  ##  pointer-#.
```

```
#####  #####, ###  #####  #####  #####  #####  pointer-##, #####  ##
#####  #  ###  #  ##  #####  #####  #####  #####  ^  #  @.
#####  #####  #####  #####  #  ###  #####, #  #  #####  #####  ###
#####  #####  ###  #####:
```

```
type
  TMyClass = class
    Value: Integer;
    Next: TMyClass;
  end;
```

7.9.

```
#  #####  #####  #####  "#####  " (overload)  #####
#####  #####  #####, #####  #####  #####  #  #####
#####  #####  #####. #####  #####  #####:
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
```

```
##### ##### # ##### Object Pascal ###  
#####
```

```
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
uses StrUtils;
```

```
operator* (const S: string; const A: Integer): string;
```

```
begin
```

```
    Result := DupeString(S, A);
```

```
end;
```

```
begin
```

```
    WriteLn('повтор' * 10);
```

```
end.
```

```
##### ##### ##### ##### ### #####. ##### ##, ### ##### #  
##### #####-##### ##### ##### #####, ##### ###  
##### ##### ## ##### #####.
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
```

```
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

```
{$ifndef FPC}
```

```
    {$message warn 'Delphi does not support global operator overloading'}
```

```
    begin end.
```

```
{$endif}
```

```
uses
```

```
    SysUtils;
```

```
type
```

```
    TMyClass = class
```

```
        MyInt: Integer;
```

```
    end;
```

```
operator* (const C1, C2: TMyClass): TMyClass;
```

```
begin
```

```
    Result := TMyClass.Create;
```

```
    Result.MyInt := C1.MyInt * C2.MyInt;
```

```
end;
```

```
var
```

```
    C1, C2: TMyClass;
```

```
begin
```

```
    C1 := TMyClass.Create;
```

```
    try
```


Object Pascal ###
#####

```
C1.MyInt := 12;  
C2 := C1 * C1;  
try  
  WriteLn('12 * 12 = ', C2.MyInt);  
finally  
  FreeAndNil(C2);  
end;  
finally  
  FreeAndNil(C1);  
end;  
end.
```

record-###. #####

#####

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}  
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}  
  
{$ifndef FPC}  
  {$message warn 'Delphi does not support global operator overloading'}  
  begin end.  
{$endif}
```

```
uses SysUtils;
```

```
type  
  TMyRecord = record  
    MyInt: Integer;  
  end;
```

```
operator* (const C1, C2: TMyRecord): TMyRecord;  
begin  
  Result.MyInt := C1.MyInt * C2.MyInt;  
end;
```

```
var  
  R1, R2: TMyRecord;  
begin  
  R1.MyInt := 12;  
  R2 := R1 * R1;  
  WriteLn('12 * 12 = ', R2.MyInt);  
end.
```

```
#####  #####  # ##### Object Pascal  ###  
#####
```

```
###  #####  # record-###  #####  #####  #####  {$modeswitch  
advancedrecords} # #####  #####  # class operator  
#####  #####  record-#. #####  #####  #####  #####  generic  
#####  #####  #####  ## #####  #####-#####  #####  (#####  
TFPGList, #####  #####  ## #####  #####  #####  #####). #  
#####  #####  "#####  #####  #####  (##  #####  
#####  #####  record) ##  #####  (#####  ##  ##  #####  ##,  
#####  #####  # TFPGList), # ##  #####  specialize #####  #  #####  
specialize TFPGList<TMyRecord>.
```

```
{$ifdef FPC}  
  {$mode objfpc}{$H+}{$J-}  
  {$modeswitch advancedrecords}  
{$endif}  
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}  
  
{$ifndef FPC}  
  {$message warn 'Delphi does not have FGL unit'}  
  begin end.  
{$endif}
```

uses

SysUtils, FGL;

type

```
TMyRecord = record  
  MyInt: Integer;  
  class operator+ (const C1, C2: TMyRecord): TMyRecord;  
  class operator= (const C1, C2: TMyRecord): boolean;  
end;
```

```
class operator TMyRecord.+ (const C1, C2: TMyRecord): TMyRecord;  
begin  
  Result.MyInt := C1.MyInt + C2.MyInt;  
end;
```

```
class operator TMyRecord.= (const C1, C2: TMyRecord): boolean;  
begin  
  Result := C1.MyInt = C2.MyInt;  
end;
```

type

```
TMyRecordList = {$ifdef FPC}specialize{$endif} TFPGList<TMyRecord>;
```



```
var
  R, ListItem: TMyRecord;
  L: TMyRecordList;
begin
  L := TMyRecordList.Create;
  try
    R.MyInt := 1;   L.Add(R);
    R.MyInt := 10;  L.Add(R);
    R.MyInt := 100; L.Add(R);

    R.MyInt := 0;
    for ListItem in L do
      R := ListItem + R;

      WriteLn('1 + 10 + 100 = ', R.MyInt);
    finally
      FreeAndNil(L);
    end;
  end.
```

8.

8.1. ##### private # strict private

private ##### ## #####
#####. ##### ## #####: #####
unit-# ##### ## #####
private ##### # #####. ##### ## C++ ##### ##, ## # ##### ##
unit-# ##### "#####". ##### ## #####.

#####, ## ##### unit-## ##### ## #####

strict private. ## #####, ## #####
#####. #####. #####.

protected (#####
"#####" # ## # unit-) # strict
protected (##### # #####
#####).

8.2. #####
#####

```
##### (const) ##  
#### (type). #####  
#####  
#####  
##### private (## "#####"), #####.
```

```
#####  
##### var:
```

type

```
TMyClass = class
```

private

type

```
TInternalClass = class
```

```
Velocity: Single;
```

```
procedure DoSomething;
```

end;

var

```
FInternalClass: TInternalClass;
```

public

const

```
DefaultVelocity = 100.0;
```

```
constructor Create;
```

```
destructor Destroy; override;
```

end;

```
constructor TMyClass.Create;
```

begin

inherited;

```
FInternalClass := TInternalClass.Create;
```

```
FInternalClass.Velocity := DefaultVelocity;
```

```
FInternalClass.DoSomething;
```

end;

```
destructor TMyClass.Destroy;
```

begin

```
FreeAndNil(FInternalClass);
```

inherited;

end;

{ Обратите внимание на префикс "TMyClass.TInternalClass." }

```
##### ##### # ##### Object Pascal ###
#####
```

```
procedure TMyClass.TInternalClass.DoSomething;
begin
end;
```

8.3. Class method

```
##### ##, ##### ##### ## ##### ( TMyClass ), ##
##### ## ### #####.
```

```
type
  TEnemy = class
    procedure Kill;
    class procedure KillAll;
  end;
```

```
var
  E: TEnemy;
begin
  E := TEnemy.Create;
  try
    E.Kill;
  finally
    FreeAndNil(E);
  end;
  TEnemy.KillAll;
end;
```

```
##### ##, ### ## ##### — ##### ##
##### ##, #####, ##### # ##### 8.4,
«##### ## ».
```

```
##### ##, ### constructor #####, ### class method #####
##### "#####" ( MyInstance := TMyClass.Create( ...
); ). #####, ##### # #####
##### ## —# ## # #####
##### "#####" constructor-##, #####
##### constructor (#####, #####
#####) #####, # ##### constructor
(#####).
```

8.4. ##### ##

class method-# ## constructor-# ## #####
#####. ##### ## #####
#####: class of TMyClass.

type

```
TMyClass = class(TComponent)
end;
```

```
TMyClass1 = class(TMyClass)
end;
```

```
TMyClass2 = class(TMyClass)
end;
```

```
TMyClassRef = class of TMyClass;
```

var

```
C: TMyClass;
ClassRef: TMyClassRef;
```

begin

```
// Можно сделать так:
```

```
C := TMyClass.Create(nil); FreeAndNil(C);
C := TMyClass1.Create(nil); FreeAndNil(C);
C := TMyClass2.Create(nil); FreeAndNil(C);
```

```
// А с помощью ссылки на класс можно сделать следующим образом:
```

```
ClassRef := TMyClass;
C := ClassRef.Create(nil); FreeAndNil(C);
```

```
ClassRef := TMyClass1;
C := ClassRef.Create(nil); FreeAndNil(C);
```

```
ClassRef := TMyClass2;
C := ClassRef.Create(nil); FreeAndNil(C);
```

end;

class method-###.
—

```
#####  #####  # ##### Object Pascal ###  
#####
```

```
#####  #####  #####  #####  #####  ### # #####  #####  
#####.
```

type

```
TMyClass = class(TComponent)  
  class procedure DoSomething; virtual; abstract;  
end;
```

```
TMyClass1 = class(TMyClass)  
  class procedure DoSomething; override;  
end;
```

```
TMyClass2 = class(TMyClass)  
  class procedure DoSomething; override;  
end;
```

```
TMyClassRef = class of TMyClass;
```

var

```
C: TMyClass;  
ClassRef: TMyClassRef;
```

begin

```
ClassRef := TMyClass1;  
ClassRef.DoSomething;
```

```
ClassRef := TMyClass2;  
ClassRef.DoSomething;
```

```
{ А следующая строка приведёт к ошибке выполнения,  
  поскольку DoSomething является abstract в TMyClass. }
```

```
ClassRef := TMyClass;  
ClassRef.DoSomething;
```

end;

```
####  ####  #####  #####  # #####  #####  #####  ##  ####  #####  (##  
##  #####-####  #####  #####, # ##  #####  #####, #####  ###  
#####  ###  #####  #####  #####  #####), #####  #####  
#####  ClassType. #####  #####, #####  ###  ClassType  #####  
TClass, #####  #  ####  #####  #####  class of TObject. ###  ####  
#####  ##  ###  ####  #####  #####, #####  #####  #####, ###  #####  
#####  #####  #####.
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
ClassType ##### ##### ##### #####,
##### ##### constructor-#. ##### ##### #####
#####, ### Clone ##### ##### ##### #####
##### # ##### ##### #####. ##### #####
##### # Assign (##. ##### 6.3, «##### TPersistent.Assign»)
### ####, ##### ##### #####, ##### # ##### #####
#####.
```

```
##### #####, ### ##### ##### #####
constructor ##### #####. #####, ### #####
# ##### TComponent, ##### ###
##### override ##### constructor-# TComponent.Create(AOwner:
TComponent).
```

type

```
TMyClass = class(TComponent)
    procedure Assign(Source: TPersistent); override;
    function Clone(AOwner: TComponent): TMyClass;
end;
```

```
TMyClassRef = class of TMyClass;
```

```
function TMyClass.Clone(AOwner: TComponent): TMyClass;
```

begin

```
// Таким образом будет создан класс конкретного типа TMyClass:
// Result := TMyClass.Create(AOwner);
// А такой подход может создать класс как типа TMyClass, так и его
наследников:
Result := TMyClassRef(ClassType).Create(AOwner);
Result.Assign(Self);
```

end;

8.5. Class helper

```
##### ##### ##### ##### #####. #####
## ##### # ##### ##### MyInstance.MyMethod(...
). # ##### ##### #####, ### #####
##### X, ##### X.Action(...).
```

```
#####, ##### ##### #####-####, ###
##### ##### ## ##### TMyClass, ##### ###
##### TMyClass. ##### #####. #####,
```

Object Pascal ###
#####

#, #####, ##### ##
#. ##### #
— ##### Render # ##### TMy3DObject #####
#, #, #, ##### TMy3DObject
#? # #####
"#####" #, # #
#.

- #####,
TMy3DObject #####.

```
procedure Render(const Obj1: TMy3DObject; const Color: TColor);
var
  I: Integer;
begin
  for I := 0 to Obj1.ShapesCount - 1 do
    RenderMesh(Obj1.Shape[I].Mesh, Color);
end;
```

#####. #, ##### - #
#. ##### #
X.Action(...), # #
Render(X, ...). ##### X.Render(...), #
Render # unit-#, # TMy3DObject .

class helper-#, #/

"#####"
— "#####" TMy3DObject .

```
type
  TMy3DObjectHelper = class helper for TMy3DObject
    procedure Render(const Color: TColor);
  end;

procedure TMy3DObjectHelper.Render(const Color: TColor);
var
  I: Integer;
begin
  { Обратите внимание, мы получаем доступ к ShapesCount, Shape без
    дополнительных указаний типа TMy3DObject.ShapesCount }
```

```
##### # ##### Object Pascal ###  
#####
```

```
for I := 0 to ShapesCount - 1 do  
  RenderMesh(Shape[I].Mesh, Color);  
end;
```



```
##### "type helper", #####  
##### #####  
# ##### integer. #####  
##### "record helper" #####... ##, ##  
##### ##. #####: http://lists.freepascal.org/fpc-announce/2013-February/000587.html.
```

8.6. ##### constructor-#, destructor-#

```
### destructor-# ##### Destroy, # ## ##### virtual  
(##### ## #####)   
# #####.
```

```
# ##### constructor-# ##### Create.  
##### # ##, #####  
##### — #####, ##### CreateMy, ##### # Create, #####  
constructor Create ##### ## #####  
##### # ##### CreateMy #####.
```

```
# ##### TObject constructor ## #####, # ##  
##### ##. ##### constructor #####  
constructor ##### (#####: ## overload, #  
##### ## ##).
```

```
# ##### TComponent, ##### constructor  
Create(AOwner: TComponent); override; . ## #####  
##### , # #####  
## #####, ##### constructor-# #####  
(##. ##### 8.4, «#####» ##).
```

8.7. ##### constructor-#

```
### ##### # ##### constructor-# #####  
#####
```

```
X := TMyClass.Create;
```

Object Pascal ###
#####

#####, # X ## ##### #####-#####
#####. ## ##### ##### ##### #####
#####?

#####. ## #####
constructor-#, ## ##### destructor. ##### ##
destructor ##### "#####", #.#. ##### # #####,
#####. ##### ## ## ##, ##
#####, #####, # ##### FreeAndNil.

##, ## ##### constructor-# ##
#####. ##### ## ## ##
nil, # ##### 0 # #.#.

#.#. ##### ## ## ## ##:

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}  
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

uses

SysUtils;

type

TGun = **class**

end;

TPlayer = **class**

Gun1, Gun2: TGun;

constructor Create;

destructor Destroy; **override**;

end;

constructor TPlayer.Create;

begin

inherited;

Gun1 := TGun.Create;

raise Exception.Create('Вызываем exception из constructor-a!');

Gun2 := TGun.Create;

end;

destructor TPlayer.Destroy;

begin

{ В данном случае в результате ошибки в constructor-е, у нас
может оказаться Gun1 <> nil и Gun2 = nil. Смириться. }


```
{ $message warn 'Delphi does not support CORBA interfaces, only COM, that
change how memory is managed. This example is not valid in Delphi.' }
begin end.
{$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}

uses
    SysUtils, Classes;

type
    IMyInterface = interface
    [ '{79352612-668B-4E8C-910A-26975E103CAC}' ]
        procedure Shoot;
    end;

    TMyClass1 = class(IMyInterface)
        procedure Shoot;
    end;

    TMyClass2 = class(IMyInterface)
        procedure Shoot;
    end;

    TMyClass3 = class
        procedure Shoot;
    end;

procedure TMyClass1.Shoot;
begin
    WriteLn('TMyClass1.Shoot');
end;

procedure TMyClass2.Shoot;
begin
    WriteLn('TMyClass2.Shoot');
end;

procedure TMyClass3.Shoot;
begin
    WriteLn('TMyClass3.Shoot');
end;

procedure UseThroughInterface(I: IMyInterface);
begin
    Write('Стреляем... ');
```

```
I.Shoot;  
end;  
  
var  
  C1: TMyClass1;  
  C2: TMyClass2;  
  C3: TMyClass3;  
begin  
  C1 := TMyClass1.Create;  
  C2 := TMyClass2.Create;  
  C3 := TMyClass3.Create;  
  try  
    if C1 is IMyInterface then  
      UseThroughInterface(C1 as IMyInterface);  
    if C2 is IMyInterface then  
      UseThroughInterface(C2 as IMyInterface);  
    if C3 is IMyInterface then  
      UseThroughInterface(C3 as IMyInterface);  
  finally  
    FreeAndNil(C1);  
    FreeAndNil(C2);  
    FreeAndNil(C3);  
  end;  
end.
```

9.2. CORBA # COM

"CORBA"?

CORBA #####. ##### ## #####
#####. ##### "#####". ##
, #####
, # ##
API.

, ## ## #####
CORBA (Common Object Request Broker Architecture) (##.
https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture), ##
#####.

{\$interfaces corba} ?

, ##### ## ##### COM #####. #####
{\$interfaces com} , ## ##
, ## ## ## #####.

Object Pascal ###
#####

COM #####. ##### # #####, ####
#####. #####
CORBA # ##### — ### ##### ##, ### ##### ## #####, ###
C# # Java. ### ##### COM #####
"#####", #####, #####, ## #####.

##, ### {\$interfaces xxx} #####
##, ##### # ## #
(### ##### # ## interface, # ##
interface(ISomeAncestor)). ### ##### # ##
#####, ## ##### ## ##, ### # ##### # ##
{\$interfaces xxx}.

COM #####?

COM ##### # ##### # #####,
IUnknown. ##### #
IUnknown:

- #####, ##### # ##### # _AddRef # _ReleaseRef.

(reference-counting).
- ##### # QueryInterface.
- ##### # ##### COM (#####
#####).

COM #####?

##, ##### COM ##### # ####
#####. ## ##### — ##### reference-
counting #####. ##### #
(##### # ##, ##### # "#####"),
##, ##### # #####. # ##### #
#####.

- ##### # ##### # ##### # ##### (## #
#####) ##### # #####.
- ##### # ##### # ##### # ##### # ##### #
reference-counting.
- ##### # #####-##### # ## # ##### COM #####.

- ##### # ##### # #####. ##### # #
##, #####. # ##

```
#####          # ##### Object Pascal ###
#####
```

```
#####, ##### ##### ##### #: ##### reference-
counting ##### COM, #### # #####
##### _AddRef # _ReleaseRef, ### ##### #
#####. ##### ##### # #####, ##### ##
##### ##### ## ##### #####.
#### ##### ## #####.
```

```
##### ## #####, ### #####: ##### ##### # ##### CORBA
# ##### {${interfaces corba} ##### # ##### ##,
##### # #####. ## ## #####, COM ##### ##
##### "#####" #####.
```

```
#####, ##### #####, ##### ##### ##### ##### # # COM
#####.
```

reference-counting ##### # ##### CORBA?

```
#####. ##### ##### ##### ##### _AddRef / _ReleaseRef .
### ##### ##### ##### ##### IUnknown . #####, #
##### #####, ##### ##### ##### #####
reference-counting # #####, ##### ##### ##### COM
#####.
```

9.3. GUID-#

```
GUID-# ### ## ##### ##### ##### ##### ##### ##### #
##### ['{ABCD1234-...}'] ##### ##### ##### ### ##### #####
#####. # #####, ### # #####. ##, # #####,
# ### #####. ##### ##### # ### ## #####
(#### ## ##### # ##### COM ### CORBA). #####,
### ##### ##### ### #####. # ##### #
#####. #####, ###, ##### ##### ### GUID-##,
```

```
### ##### (#####) GUID-#, ### ##### #####  
### ##### is. #####, ## ##### true ####  
##### ##### ##### ## #####. "#####"  
##### Supports(ObjectInstance, IMyInterface) #####  
##### # #####, ##### ##### #####  
### GUID. ### ##### # CORBA. # COM #####. ### ##### FPC 3.0.0.
```

```
##### , ##### ##### ##### GUID ### #####
#####. ##### ##### ##### GUID-## Lazarus GUID
```

```
#####  #####  # ##### Object Pascal ###
#####
```

```
generator(##### Ctrl + Shift + G # #####).####
##### ##-####, ##### https://www.guidgenerator.com/.
```

```
# ##### ##### ##### #####, ##### CreateGUID #
GUIDToString ##### ## RTL. #####, #####:
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}

uses
  SysUtils;
var
  MyGuid: TGUID;
begin
  Randomize;
  CreateGUID(MyGuid);
  WriteLn(['' + GUIDToString(MyGuid) + '']);
end.
```

9.4. ##### (COM)

```
##### COM ##### ##
```

1. ##### # COM (#####, #####
Windows, ##### # Unix-##### XPCOM, #####
Mozilla),
2. ##### - reference counting (###

##).

```
## ##, ##### #  
## ##### ##, ##  
#####: ##### # API, ##  
##### ## COM  
##### #  
# COM.
```

```
## #####:
```

- # ##### "#####" ##### _AddRef, _Release
QueryInterface. ###, ###
#####

```
##### # ##### Object Pascal ###  
#####
```

```
##### COM #####, ### reference-counting. #####,  
##### ## ##### — ##. #####.
```

```
# ##### TInterfacedObject ##### ##  
##### # #####, ##### reference-counting.
```

```
# ##### TComponent ##### ##  
# #####, ##### reference-counting. # Castle  
Game Engine #####, ##  
#####: TNonRefCountedInterfacedObject  
# TNonRefCountedInterfacedPersistent ### ##, ##.  
#####: https://github.com/castle-engine/castle-engine/  
blob/0519585abc13e8386cdae5f7dfef6f9659dc9b57/src/base/  
castleinterfaces.pas.
```

- ##### ##
#####.
(##### ##
reference-counting, ##### _AddRef
#####), ##
##-## ##, ##-####
##. ##. ##### "7.7 Reference counting" #
FPC (<http://freepascal.org/docs-html/ref/refse47.html>).

```
##### COM #####
```

- #####, ## # ## reference-counting,
- ##### TInterfacedObject
- # ##### # #####, #####
reference-counting #####.

```
#### #####:
```

```
.....  
{ $ifdef FPC }  
  { $mode objfpc } { $H+ } { $J- }  
  { $interfaces com }  
{ $endif }  
{ $ifdef MSWINDOWS } { $apptype CONSOLE } { $endif }
```

```
uses  
  SysUtils, Classes;
```


type

IMyInterface = **interface**

['{3075FFCD-8EFB-4E98-B157-261448B8D92E}']

procedure Shoot;

end;

TMyClass1 = **class**(TInterfacedObject, IMyInterface)

procedure Shoot;

end;

TMyClass2 = **class**(TInterfacedObject, IMyInterface)

procedure Shoot;

end;

TMyClass3 = **class**(TInterfacedObject)

procedure Shoot;

end;

procedure TMyClass1.Shoot;

begin

WriteLn('TMyClass1.Shoot');

end;

procedure TMyClass2.Shoot;

begin

WriteLn('TMyClass2.Shoot');

end;

procedure TMyClass3.Shoot;

begin

WriteLn('TMyClass3.Shoot');

end;

procedure UseThroughInterface(I: IMyInterface);

begin

Write('Стреляем... ');

I.Shoot;

end;

var

C1: IMyInterface; // COM управляет освобождением памяти

C2: IMyInterface; // COM управляет освобождением памяти

C3: TMyClass3; // Здесь управлять освобождением памяти придётся ВМ

begin

```
##### ##### # ##### Object Pascal ###
#####
```

```
C1 := TMyClass1.Create as IMyInterface;
C2 := TMyClass2.Create as IMyInterface;
C3 := TMyClass3.Create;
try
  UseThroughInterface(C1); // Нет необходимости в операторе "as"
  UseThroughInterface(C2);
  if C3 is IMyInterface then
    UseThroughInterface(C3 as IMyInterface); // Так не работает
finally
  { Переменные C1 и C2 выходят из поля зрения
    и будут автоматически уничтожены сейчас.

    а переменная C3 является экземпляром класса
    и не управляется интерфейсом,
    по этому её необходимо совободить вручную. }
  FreeAndNil(C3);
end;
end.
```

9.5. ##### COM ##### # ##### reference-counting

```
###   ###   #####   #####   #   #####   #####,   #   #####,
#####   TComponent   (#####   #####,   #####   ###
TNonRefCountedInterfacedObject   #
TNonRefCountedInterfacedPersistent ) ##### reference-counting ###
COM #####.   ###   #####   #####   COM   #####   #   ###   ###
#####   #####   #####   #####   #####.
```

```
###   ##,   #####   #####   #####   #   #####   #####,   #####   ##   #####
#####,   #####   #####-#####   #####   #####   #####   ##   #####   #   ##
#####,   ###   #####   #####   #####   #####   #####   #   #####   Cx as IMyInterface
#####   #####   #####   #####   #####   #####,   #####   #####   #####
#####   ##   #####   #####   #####.   #####   #####   #####   #####
UseInterfaces,   #####   #####   #####   ###   #####   #####   (#####   ##
#####   #####   #####,   ###   #####   #####   #####   #####   ##   #####
#####   #####).
```

```
#####   #####   #####   #####,   #####   #####   #####   CORBA,
#####   #   #####   #####   ###   #####   #####   #####   #####
reference-counting   #   #####.
```

```
{$ifdef FPC}  
  {$mode objfpc}{$H+}{$J-}  
  {$interfaces com}  
{$endif}  
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
```

uses

SysUtils, Classes;

type

IMyInterface = **interface**
 ['{3075FFCD-8EFB-4E98-B157-261448B8D92E}']
 procedure Shoot;
end;

TMyClass1 = **class**(TComponent, IMyInterface)
 procedure Shoot;
end;

TMyClass2 = **class**(TComponent, IMyInterface)
 procedure Shoot;
end;

TMyClass3 = **class**(TComponent)
 procedure Shoot;
end;

procedure TMyClass1.Shoot;
begin
 WriteLn('TMyClass1.Shoot');
end;

procedure TMyClass2.Shoot;
begin
 WriteLn('TMyClass2.Shoot');
end;

procedure TMyClass3.Shoot;
begin
 WriteLn('TMyClass3.Shoot');
end;

procedure UseThroughInterface(I: IMyInterface);
begin
 Write('Стреляем... ');

```
I.Shoot;
end;

var
  C1: TMyClass1;
  C2: TMyClass2;
  C3: TMyClass3;

procedure UseInterfaces;
begin
  if C1 is IMyInterface then
    //if Supports(C1, IMyInterface) then // эта строчка идентична проверке
    "is" выше
    UseThroughInterface(C1 as IMyInterface);
  if C2 is IMyInterface then
    UseThroughInterface(C2 as IMyInterface);
  if C3 is IMyInterface then
    UseThroughInterface(C3 as IMyInterface);
end;

begin
  C1 := TMyClass1.Create(nil);
  C2 := TMyClass2.Create(nil);
  C3 := TMyClass3.Create(nil);
  try
    UseInterfaces;
  finally
    FreeAndNil(C1);
    FreeAndNil(C2);
    FreeAndNil(C3);
  end;
end.
```

9.6.

CORBA, ## # COM #####. ####, ####
CORBA #####.

1. ##### as ##### #
#####.

```
UseThroughInterface(Cx as IMyInterface);
```

```
##### # ##### Object Pascal ###
#####
```

```
#### ##### C1, C2, C3 ##
##### # #####. ### #####
##### C3, # ##### IMyInterface.
```

```
##### as ##### ## ##,
##### Cx ### ##### (#####, TMyClass2) ### ##
##### (#####, IMyInterface2).
```

However, it is not allowed for CORBA interfaces.

2. #####, ##### ##### ## ##### #####
#####:

```
UseThroughInterface(Cx);
```

```
# ##### ##### # #####. #####
### ##### C1 # C2 (#####, ## ##,
##### IMyInterface), ## ## C3.
```

```
## #####, ##### ##### ## ##
#####. # #####, ##### TMyClass, #####
##### TMyClass, #####
##### TMyClass ##### # ##
#####. ## ##### # ##### #
#####.
```

3. ##### IMyInterface(Cx) #####
#####:

```
UseThroughInterface(IMyInterface(Cx));
```

```
#####, ##### ##### ##
#####. ##### ## #####,
##### ##, ## # ##### # #####,
### _#####_# #####.
```

```
##### #: Cx ##### class (#####
TMyClass2), ## ##### ## ## ##
#####. #####, ##### # #####
##### - ##### (#####).
```

```
#####  #####  # #####  Object Pascal  ###
#####
```

```
#####  ###  ###  #####, #####  ##  #####  #####:
```

```
{$ifdef FPC} {$mode objfpc}{$H+}{$J-} {$endif}
{$ifdef MSWINDOWS} {$apptype CONSOLE} {$endif}
// {$interfaces corba} // обратите внимание, что приведение типа с помощью
"as" для интерфейсов типа CORBA не скомпилируется
```

```
uses Classes;
```

```
type
```

```
IMyInterface = interface
[ '{7FC754BC-9CA7-4399-B947-D37DD30BA90A}' ]
    procedure One;
end;
```

```
IMyInterface2 = interface(IMyInterface)
[ '{A72B7008-3F90-45C1-8F4C-E77C4302AA3E}' ]
    procedure Two;
end;
```

```
IMyInterface3 = interface(IMyInterface2)
[ '{924BFB98-B049-4945-AF17-1DB08DB1C0C5}' ]
    procedure Three;
end;
```

```
TMyClass = class(TComponent, IMyInterface)
    procedure One;
end;
```

```
TMyClass2 = class(TMyClass, IMyInterface, IMyInterface2)
    procedure One;
    procedure Two;
end;
```

```
procedure TMyClass.One;
begin
    Writeln('TMyClass.One');
end;
```

```
procedure TMyClass2.One;
begin
    Writeln('TMyClass2.One');
end;
```

```
procedure TMyClass2.Two;
```

```
begin
  Writeln('TMyClass2.Two');
end;

procedure UseInterface2(const I: IMyInterface2);
begin
  I.One;
  I.Two;
end;

procedure UseInterface3(const I: IMyInterface3);
begin
  I.One;
  I.Two;
  I.Three;
end;

var
  My: IMyInterface;
  MyClass: TMyClass;
begin
  My := TMyClass2.Create(nil);
  MyClass := TMyClass2.Create(nil);

  // Следующий код не скомпилируется, так как в момент компиляции
  // неизвестно является ли My интерфейсом IMyInterface2.
  // UseInterface2(My);
  // UseInterface2(MyClass);

  // Это скомпилируется и работает правильно.
  UseInterface2(IMyInterface2(My));
  // А это не скомпилируется. Приведение типа InterfaceType(ClassType)
  // проверяется в момент компиляции.
  // UseInterface2(IMyInterface2(MyClass));

  // Это скомпилируется и работает правильно.
  UseInterface2(My as IMyInterface2);
  // Это скомпилируется и работает правильно.
  UseInterface2(MyClass as IMyInterface2);

  // Это скомпилируется, но приведёт к непонятной ошибке "Access
  // violation" при выполнении программы.
  // UseInterface3(IMyInterface3(My));
  // Это не скомпилируется. Приведение типа InterfaceType(ClassType)
  // проверяется в момент компиляции.
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
// UseInterface3(Interface3(MyClass));

// Это скомпилируется, но приведёт к понятному сообщению об ошибке
"EInvalidCast: Invalid type cast" и укажет на проблему.
// UseInterface3(My as Interface3);
// Это скомпилируется, но приведёт к понятному сообщению об ошибке
"EInvalidCast: Invalid type cast" и укажет на проблему.
// UseInterface3(MyClass as Interface3);

writeln('Готово');
end.
```

10.

Copyright Michalis Kamburelis.

```
##### ##### ##### ##### # ##### AsciiDoc ##### ##
#####: https://github.com/modern-pascal/modern-pascal-introduction. #####
### #####, #####, #####, ##### # pull
request-## :). # ##### ##### ##### ##### GitHub ##### # e-mail:
michalis@castle-engine.io1. #####: https://michalis.xyz/.
```

```
##### ##### ##### ##### # #####
##### ##### Wikipedia, ##. https://en.wikipedia.org/wiki/
Wikipedia:Copyrights:
```

- *Creative Commons Attribution-ShareAlike 3.0 Unported License (CC BY-SA)*

```
####
```

- *GNU Free Documentation License (GFDL) (unversioned, with no invariant sections, front-cover texts, or back-cover texts)*.

```
##### ## #####!
```

```
##### ## #####: ##### # ##### 2016-2019
```

```
##### ## ##### # ##### # #####: vitality #
pupsik @ freepascal.ru
```

¹ <mailto:michalis@castle-engine.io>