Pascal

1. Prosty program

- Program wypisujący tekst na ekranie.

```
program helloConsole;
                      {słowo kluczowe program wraz z nagłówkiem
                      programu (taki sam jak nazwa projektu)}

{$APPTYPE CONSOLE}
                      {dyrektywa kompilatora, powoduje, ze
                      aplikacja będzie uruchamiana w konsoli}
uses
  SysUtils;
                      {lista modułów z których chcemy korzystać}

begin                          //blok instrukcji
  Writeln('Hello World');      //wyświetla napis
  Readln;
end.
```

Wyjście:

```
Hello World
```

- Inny przykładowy program

```
{wypisz zestawienie temperatur Fahrenheita-
Celsjusza dla f=0, 20, ..., 300}
program FahrCels;

{$APPTYPE CONSOLE}

uses
  SysUtils;

var                     {deklaracje zmiennych}
  fahr, celsius: Integer;
  lower, upper, step: Integer;

begin               //blok instrukcji
  lower := 0;       /dolna granica temperatur
  upper := 300;     //gorna granica
  step := 20;       //rozmiar kroku

  fahr := lower;

  while fahr <= upper do
  begin
    celsius := 5 * (fahr - 32) div 9;
    Writeln(fahr, #9, celsius);      //#9 - znak tabulacji
    fahr := fahr + step;
  end;
    Readln;
end.
```

Wyjście:

```
0       -17
20      -6
40      4
60      15
80      26
100     37
120     48
140     60
160     71
180     82
200     93
220     104
240     115
260     126
280     137
300     148
```

2. Język Object Pascal

2.1. Komentarze

- Nawiasy klamrowe { .. }, ,
- Ograniczniki typu „nawias-gwiazdka” (* ... *),
- Podwójny ukośnik // zapożyczony z języka C++, np.

```
//komentarz ograniczający 1 linię
```

- Mogą się rozciągać na wiele linii.
- Komentarze tej samej postaci nie mogą być zagnieżdżane.
- Można zagnieżdżać komentarze różnych typów np.:

```
(* Poniższy komentarz {jest
   zagnieżdżony} wskazuje
   instrukcję *)
```

- Uwaga! – jeśli po znaku komentarza (* lub { wystąpi znak $ to jest to interpretowane jako dyrektywa kompilatora.

2.2. Identyfikatory

- Ciąg liter (alfabetu angielskiego) i cyfr dziesiętnych.
- Pierwszym znakiem musi być litera (znak _ jest zaliczany do liter).
- Wielkie i małe litery nie są rozróżniane.
- Mogą być dowolnej długości (tylko 255 pierwszych znaków jest znaczących).

```
Alfa Alfa alfa //te same zmienne!!!
Cena_Produktu
_zmienna_systemowa
```

2.3. Słowa kluczowe i dyrektywy

Słowa kluczowe są to zastrzeżone słowa będące integralną częścią języka Object Pascal i nie mogą być definiowane przez programistę.
Rozpoznać je można po tym, że przy pisaniu w edytorze rozświetlają się lub zmieniają kolor np. begin, while etc.

And	**exports**	**library**	**set**
array	**file**	**mod**	**shl**
As	**finalization**	**nil**	**shr**
Asm	**finally**	**not**	**string**
begin	**for**	**object**	**then**
Case	**function**	**of**	**threadvar**
class	**goto**	**or**	**to**
const	**if**	**out**	**try**
constructor	**implementation**	**packed**	**type**
destructor	**in**	**procedure**	**unit**
dispinterface	**inherited**	**program**	**until**
div	**initialization**	**property**	**uses**
do	**inline**	**raise**	**var**
downto	**interface**	**record**	**while**
else	**is**	**repeat**	**with**
end	**label**	**resourcestring**	**xor**
except			

W odróżnieniu od słów kluczowych, dyrektywy języka (predefiniowane słowa) nie są zastrzeżone. Podane wyrazy mogą więc być identyfikatorami zdefiniowanymi przez programistę, co jednak nie jest zalecane.

absolute	**dynamic**	**message**	**private**	**resident**
abstract	**export**	**name**	**protected**	**safecall**
assembler	**external**	**near**	**public**	**stdcall**
automated	**far**	**nodefault**	**published**	**stored**
cdecl	**forward**	**overload**	**read**	**varargs**
contains	**implements**	**override**	**readonly**	**virtual**
default	**index**	**package**	**register**	**write**
deprecated	**library**	**pascal**	**reintroduce**	**writeonly**
dispid	**local**	**platform**	**requires**	

2.4. Stałe

- Są synonimami konkretnych wartości występujących w programie.
- Dopuszczalne są tylko systemy dziesiętny (domyślny) i heksadecymalny (liczba poprzedzona znakiem *$* np. `$29A, -$29A`).
- Deklaracja stałych jest poprzedzona słowem *const*.
- Object Pascal nie wymaga deklarowania typów stałych, kompilator sam ustala typ stałej na podstawie przypisywanej wartości.

```
const
  ANumber = 3.14;                    //typ ustalony Extended
  i = 20;                            //typ ustalony 1 .. 127
  InfoString = 'Blad wykonania';     //typ ustalony AnsiString
```

- Kompilator ustalając typ stałej, wybiera typy o możliwie najmniejszej liczebności.
  Automatycznie określane typy stało liczbowe.

Wartość	Typ
od -2^63 do - 2.147.483.649	Int64
od -2.147.483.648 do -32.769	Longint
od -32.768 do -129	Smallint
od -128 do -1	Shortint
od 0 do 127	1 .. 127
od 128 do 255	Byte
od 256 do 32.767	0 .. 32.767
od 32.768 do 65.535	Word
od 65.536 do 2.147.483.647	0 .. 1.147.483.647
od 2.147.483.648 do 4.294.967.295	Cardinal
od 4.294.967.296 do 2^63-1	Int64

Inne automatycznie określane typy dla stałych

- stałe o wartościach rzeczywistych są stałymi typu *Extended*.
  Stała zmiennopozycyjna składa się z: części całkowitej, kropki dziesiętnej, części ułamkowej, litery *e* lub *E*, opcjonalnego znaku + lub – oraz wykładnika potęgi. Część ułamkową z kropką można pominąć.
  Przykłady:
  `0.21e-50, 0.21E-50, 2e50`

- dla zbiorów liczbowych i znakowych zakres wynika bezpośrednio z ich postaci.
- stałe łańcuchowe, zależnie od ustawienia przełącznika kompilacji `$H`, są albo łańcuchami typu *ShortString* (dla `{$H-}`), albo łańcuchami typu *AnsiString* (dla `{$H+}`, domyślnie).

- Można jawnie wskazać typ deklarowanej stałej.
  Uwaga! - Przy włączonym przełączniku kompilatora `{$J+}` tak zadeklarowana stała zachowuje się w programie jak zwykła zmienna.
  Domyślnie kompilator zabrania modyfikowania tak zadeklarowanej stałej (ustawione `{$J-}`). Takie ustawienie jest zalecane.

```
const
  ANumber : Double = 3.14;
  i : Integer = 20;
  InfoString : String = 'Blad wykonania';
```

- Dopuszczalne jest rzutowanie typów np.

```
const
      LargeConst = Int64(1);
```

- Znaki sterujące poprzedzane są symbolem # za którym powinna być liczba z zakresu 0-255 (dziesiętna lub szesnastkowa) określająca kod ASCII symbolu np. łańcuch:
  `#89#111#117` równoważny `#$59#$6F#$75` równoważny 'You'

2.5. Typy danych

Podział typów:

- Proste
  - Porządkowe
    - Całkowite
    - Znakowe
    - Boolowskie
    - Wyliczeniowe
    - Okrojone
  - Rzeczywiste
- Łańcuchowe
- Strukturalne
  - Zbiory
  - Tablice
  - Rekordy
  - Pliki
  - Klasy
  - Reference klas
  - Interfejsy
  - Wskaźniki
- Proceduralne
- Variant

Zestawienie typów

Typ zmiennej	***Zakres***	***Format***
ShortInt	-128..127	całkowity 8-bitowy ze znakiem
Byte	0..255	całkowity 8-bitowy bez znaku
SmallInt	-32768..32767	całkowity 16-bitowy ze znakiem
Word	0..65535	całkowity 16-bitowy bez znaku
Integer, LongInt	-2147483648..2147483647	całkowity 32-bitowy ze znakiem
Cardinal, LongWord	0..4294967295	całkowity 32-bitowy bez znaku
Int64	-2^{63}..2.147.483.649	całkowity 64-bitowy ze znakiem
Single	$1{,}5 \times 10^{-45}$..$1{,}7 \times 10^{38}$	zmiennoprzecinkowy 4-bajtowy
Real48	$2{,}9 \times 10^{-39}$..$1{,}7 \times 10^{38}$	zmiennoprzecinkowy 6-bajtowy
Double	$5{,}0 \times 10^{-324}$..$1{,}7 \times 10^{308}$	zmiennoprzecinkowy 8-bajtowy
Extended	$3{,}6 \times 10^{-4951}$..$1{,}1 \times 10^{4932}$	zmiennoprzecinkowy 10-bajtowy
Currency	–922337203685477.5808 ..922337203685477.5807	stałoprzecinkowy 64-bitowy
ShortString	255 znaków	łańcuch znaków o ustalonej maksymalnej długości
AnsiString	~2^{31} znaków	dynamiczny łańcuch znaków 1-bajtowych
WideString	~2^{30} znaków	dynamiczny łańcuch znaków dwubajtowych

Typ zmiennej	*Format*
Boolean, ByteBool	boolowski 1-bajtowy (wartość True/False)
WordBool	boolowski 2-bajtowy
BOOL, LongBool	boolowski 4-bajtowy
Char	znak 1-bajtowy
WideChar	znak 2-bajtowy
PChar	łańcuch znaków jednobajtowych z zerowym ogranicznikiem
PWideChar	łańcuch znaków dwubajtowych z zerowym ogranicznikiem
TDateTime	data/czas 8-bajtowy
Variant, Olevariant, TVarData	wariantowy 16-bajtowy

- Typy dzielą się na ogólne (zależne od platformy) i podstawowe (niezależne od platformy).
  Uwaga! Operacje na typach ogólnych (np. *Char*, *Integer*, *Cardinal*) są szybsze, ale mogą się one zmieniać w zależności od procesora i systemu operacyjnego (np. *Integer* w systemach 16-bitowych ma wartość maksymalną 32767).
- Ilość miejsca zajmowanego przez dany typ można sprawdzić za pomocą funkcji `SizeOf`.
- Typy porządkowe to uporządkowane zbiory, w których każdy element oprócz pierwszego ma swojego poprzednika i każdy element oprócz ostatniego ma swojego następcę.
  Każdy element ma swój numer porządkowy:
  - dla liczb typu całkowitego jest to wartość tej liczby
  - dla typów okrojonych są to wartości odpowiadających im typów bazowych
  - dla typów wyliczeniowych jest to numer elementu zbioru zaczynając od 0

Funkcje operujące na typach porządkowych:

Ord	zwraca wartość porządkową (nie działa na typ Int64)
Pred	zwraca wartość poprzednią
Succ	zwraca wartość następną
High	zwraca wartość największą
Low	zwraca wartość najmniejszą

Typy boolowskie reprezentują wartości "prawda" (True) i "fałsz" (False).

```
Ord(False) = 0; Ord(True) = 1;
```

Typy wyliczeniowe są zbiorem dowolnych identyfikatorów.
Jeżeli elementów jest mniej niż 256 to typ wyliczeniowy zajmuje 1 bajt. Jeżeli więcej to tym zajmuje 2 bajty (może być maksymalnie 65536 elementów).

```
type Days = (Monday, Tuesday, Wednesday, Thursday, Friday);
```

Identyfikatorom tym przyporządkowywane są po kolei wartości od 0:

```
Ord(Monday) = 0; Ord(Wednesday) = 2;
```

Można zmienić domyślne przyporządkowanie wartości:

```
type Days = (Monday=1, Tuesday=2, Wednesday=Monday+Tuesday, Thursday,
Friday=0)

Ord(Monday) = 1; Ord(Wednesday) = 3;
```

Typy okrojone to typy postaci:

```
type TypOkrojony = NajmniejszaWartosc..NajwiekszaWartosc;
```

Typ okrojony zajmuje tyle samo miejsca w pamięci co typ bazowy, z którego powstał. Np. **type** Cyfry = 0..9; zajmuje 1 bajt. Typem bazowym może być dowolny typ porządkowy. Typy okrojone są wygodne przy określaniu zakresu danych, które może przyjąć funkcja/procedura.

- Konwersja typów

  Funkcje konwersji, przedstawione w poniższej tabeli zadeklarowane są w module SysUtils:

Nazwa	Opis
IntToStr	Konwertuje typ Integer na String
StrToInt	Konwertuje typ String na Integer
CurrToStr	Konwertuje typ Currency na String
StrToCurr	Konwertuje typ String na Currency
DateTimeToStr	Konwertuje typ TDateTime na String
StrToDateTime	Konwertuje typ String na TDateTime
DateToStr	Konwertuje typ TDate na String
StrToDate	Konwertuje typ String na TDate
TimeToStr	Konwertuje typ TTime na String
TimeStrToTimeToStr	Konwertuje typ String na TTime
FloatToStr	Konwertuje typ Extended na String
StroToFloat	Konwertuje typ String na Extended.
IntToHex	Konwertuje typ Integer do postaci heksydemalnej.
StrPas	Konwertuje typ String na PChar.
String	Konwertuje typ PChar na String.
PChar	Konwertuje typ String na PChar.
StrToBool	Konwertuje typ String na Boolean.
StrToInt64	Konwertuje typ String na Int64.

- Rzutowanie
  - Postać *identyfikatorTypu(wyrażenie)*
  - Stanowi jeden ze sposobów osłabienia rygorystycznej kontroli typów wykonywanej przez kompilator.
  - Warunkiem koniecznym (lecz nie wystarczającym) wykonalności rzutowania typów jest identyczny rozmiar zmiennej podlegającej rzutowaniu i typu docelowego.

Przykład:

```
var
  c : Char;
  b1, b2 : Byte;
  s : Single;
  k : Integer;

begin
  ...
  c := 'j';
  b1 := c;          //błąd kompilacji
  b2 := byte(c);    //poprawne
  ...
  s := 4.8;
  k := Trunc(s);    // przekształcenie jawne, k = 4
  k := Round(s);    // przekształcenie jawne, k = 5
    .....
  k := 5;
  s := k;           // przekształcenie ukryte, s = 5.0
```

2.6. Zmienne

- Deklaracja zmiennej powinna być umieszczona przed blokiem *begin* (danej procedury, funkcji lub programu głównego).
- Możliwe jest inicjowanie zmiennych podczas deklaracji (tylko dla zmiennych globalnych).

```
var
  n, i : Integer;   //deklaracja kilku zmiennych tego samego typu
  x1: String;
  e: Extended;
  c: Integer = 8;   //deklaracja zmiennej z przydzieleniem wartości
  s1, s2: String = 'Oto wartość 1'; {blad! - nie można przydzielić
                                     wartości kilku zmiennym naraz}

begin
end;
```

2.7. Ważniejsze operatory

Operatory przypisania, porównania i operatory logiczne:

Operator	*Pascal*
Przypisania	:=
Równości	=
Nierówności	<>
Mniejszości	<
Większości	>
Niewiększości	<=
Niemniejszości	>=
Logiczne „i”	and
Logiczne „lub”	or
Zaprzeczenie	not

Operator konkatenacji łańcuchów: +

```
Np. Nazwa := 'Object ' + 'Pascal';
```

Operatory arytmetyczne:

Operator	***Pascal***
Dodawania	`+`
Odejmowania	`-`
Mnożenia	`*`
Dzielenia rzeczywistego	`/`
Dzielenia całkowitego	`div`
Reszty z dzielenia (modulo)	`mod`
Potęgowania	brak

Operatory zwiększania i zmniejszania:

Operator	***Pascal***
Zwiększania	`Inc()`
Zmniejszania	`Dec()`

Uwaga! – Wykonując dzielenie używaj operatorów stosownych do operandów i oczekiwanego wyniku.

```
var
       i: Integer;
       e: Extended;
begin
       i := 4/3;  {błąd kompilacji – próba przypisania zmiennej całkowitej
                   wyniku dzielenia rzeczywistego}
       e  :=  3.4 div 2.3  {błąd – próba dzielenia całkowitego operandów, z
                            których co najmniej jeden nie jest liczbą
                            całkowitą}
       i := Trunc(4/3);    //ta linia jest poprawna
       e := 3.4 / 2.3;     //ta linia jest poprawna

       Inc(i);             //powiększ i o 1
       Inc(i, 20);         //powiększ i o 20
       i := i + 1;         //powiększ i o 1
end;
```

2.8. Tablice statyczne

Tablice stanowią uporządkowany ciąg zmiennych tego samego typu. Typem elementu tablicy może być dowolny typ, również zdefiniowany przez użytkownika. Poniższa deklaracja definiuje tablicę ośmiu liczb całkowitych:

```
Type
  MyArray = array [0 .. 7] of Integer;
```

Od tej chwili typ `MyArray` staje się pełnoprawnym typem danych, a więc możliwe jest definiowanie zmiennych tego typu:

```
var
  A : MyArray;
```

Powyższa deklaracja równoważna jest następującej:

```
var
  A : array [0 .. 7] of Integer;
```

W przedstawionej deklaracji poszczególne elementy tablicy identyfikowane są kolejnymi liczbami, począwszy od zera — `A[0]`, `A[1]` itd. — lecz minimalna wartość indeksu tablicy może mieć w Pascalu dowolną wartość.

Dolną i górną wartość graniczną indeksu tablicy wymiarowej zwracają funkcje `Low()` i `High()`.

Przykład:

```
var
  X : array [ 19 .. 30 ] of Integer;
  i: Integer;

begin
  { TODO -oUser -cConsole Main : Insert code here }
  for i := Low(X) to High(X) do
    X[i] := i;
  for i := Low(X) to High(X) do
    Writeln(X[i]);
  Readln;
end.
```

Możliwe jest deklarowanie tablic wielowymiarowych, np.

```
var
      TMatrix : array[1..10, 4..123] of Integer;
```

Powyższa deklaracja równoważna jest następującej:

```
var
      TMatrix : array[1..10] of array[4..123] of Integer;
```

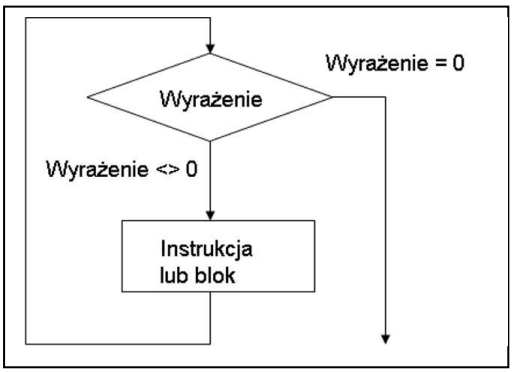
2.9. Pętla while..do

```
while wyrażenie do
  Instrukcja;

albo

while wyrażenie do
begin
   CiągInstrukcji
end
```

- Warunek jest testowany przed wejściem do pętli.
- Ciało pętli może nie zostać ani razu wykonane.
- Jeśli zawartość pętli *while..do* to kilka instrukcji, to grupujemy je za pomocą *begin..end*.
- Dopóki wyrażenie jest prawdziwe wykonuj instrukcję lub blok instrukcji.



Przykład:

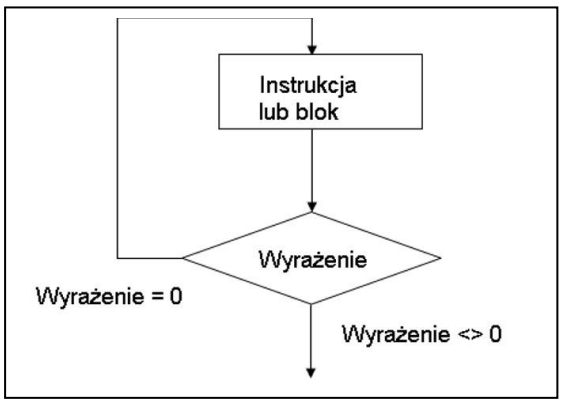
```
n := 10;
while n <> 0 do
begin
  Write(n, #9);
  Dec(n);
end;
Readln;
```

Wyjście:

```
10      9       8       7       6       5       4       3       2       1
```

2.10. Pętla repeat..until

- Warunek jest testowany po wykonaniu ciała pętli.
- Ciało pętli musi być choć raz wykonane.
- Powtarzaj do czasu zajścia warunku.



Przykład:

```
n := 10;
repeat
  Write(n, #9);
  Dec(n);
until n = 0;
Readln;
```

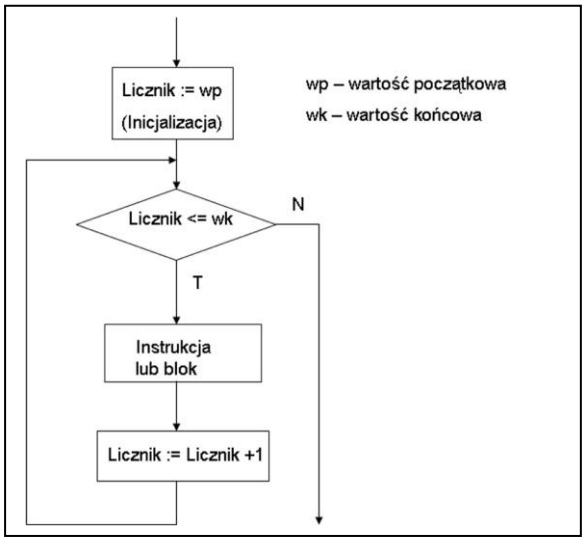
Wyjście:

```
10      9       8       7       6       5       4       3       2       1
```

2.11. Pętla for

for *licznik := wartość początkowa* **to** *wartość końcowa* **do** *instrukcja;*
for *licznik := wartość początkowa* **downto** *wartość końcowa* **do** *instrukcja;*

- Stosuje się ją wtedy, gdy dokładnie wiadomo ile wykonań (iteracji) danej czynności chcemy zastosować.
- Wartości *licznik* nie należy modyfikować wewnątrz pętli.
- Zmienna *licznik* jest obliczana tylko raz, w chwili rozpoczęcia wykonywania pętli.
- Działanie: dla *licznika* zmieniającego się od wartości początkowej do wartości końcowej z krokiem 1 (z krokiem -1 w przypadku **downto**) wykonuj instrukcję lub blok instrukcji.



Przykład:

```
var
  i : Integer;

begin
  for i := 1 to 5 do
    Write('[', i, ']', #9);
  Readln;
end.
```

```
var
  i : Integer;

begin
  for i := 5 downto 1 do
    Write('[', i, ']', #9);
  Readln;
end.
```

Wyjście:

```
[1]    [2]    [3]    [4]    [5]
```

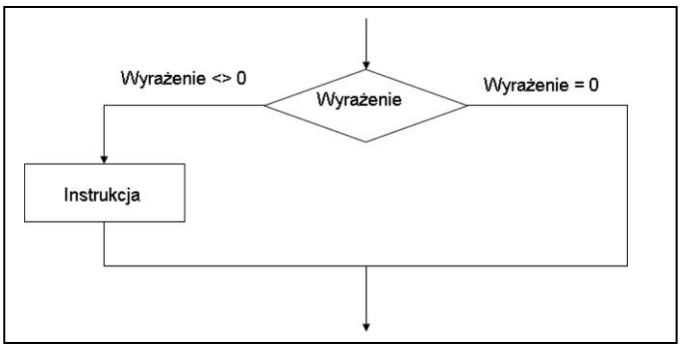
```
[5]    [4]    [3]    [2]    [1]
```

2.12. Instrukcja warunkowa if

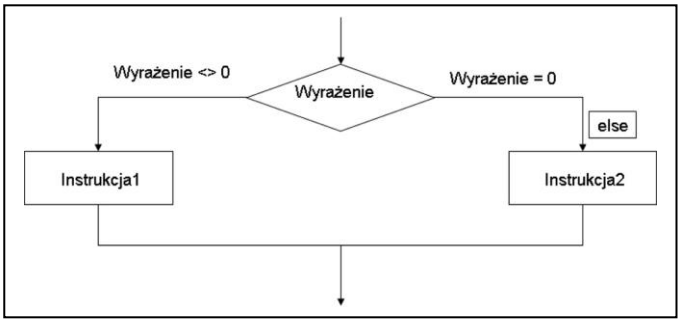
if *wyrażenie* **then**
Instrukcja;

albo

if *wyrażenie* **then**
begin
CiągInstrukcji;
end;




```
if wyrażenie then
begin
  CiągInstrukcji1;
end
else
begin
  CiągInstrukcji2
end;
```



Przykład:

```
if (n mod 2) = 0 then
  Writeln('parzysta')
else
  Writeln('nieparzysta');
```

2.13. Instrukcja warunkowa case of

- Często stosowana gdy do sprawdzenia jest więcej warunków.
- Testowana jest wartość wyrażenia (lub zmiennej) przez porównanie ze stałymi c1, c2 etc. Jeżeli nie zostanie znaleziona to nastąpi przejście do frazy else.
  Jeżeli wyrażenie = c1 to wykonaj Instrukcja1, jeżeli wyrażenie = c2 to Instrukcja2, itd. Jeżeli wyrażenia nie ma na liście wykonaj Inne.
- Po wykonaniu wskazanej instrukcji następuje wyjście za instrukcję wyboru.
- Brak możliwości porównywania danych tekstowych.

```
case wyrażenie of
  c1:  Instrukcja1;
  c2:  Instrukcja2;
  c3:  Instrukcja3;
  ...
  else    Inne
end
```

Przykład:

```
var
      n: Integer;
begin
  Writeln('Podaj liczbe...');
  Readln(n);
  case n of
    1..5: Writeln('Low');
    6..9: Writeln('High');
    0, 10..99: Writeln('Out of range');
    else Writeln('');
  end;
  Readln;
end;
```

2.14. Procedura break i continue

Wywołanie *break* wewnątrz pętli (*while*, *for*, *repeat*) powoduje jej natychmiastowe zakończenie. Jeżeli pętle są zagnieżdżone, następuje zakończenie najbardziej wewnętrznej pętli zawierającej wywołanie *break*.

Wywołanie *continue* wewnątrz pętli (*while*, *for*, *repeat*) powoduje porzucenie bieżącego „cyklu" pętli i natychmiastowe przejście do sprawdzania jej warunku.

2.15. Pomiar czasu w Delphi

Pomiar czasu w Delphi, z dokładnością do milisekund, można zrealizować m. in. na dwa poniższe proste sposoby:

- Wykorzystując strukturę `TDateTime` oraz funkcję `MiliSecondsBetween`

  `function MiliSecondsBetween(const ANow, AThen: TDateTime): Int64;`
  Funkcja zwraca różnicę milisekund pomiędzy dwoma datami typu `TDateTime`, przekazanymi w parametrach `ANow` i `AThen`

  Uwaga: Do listy modułów należy dopisać `DateUtlis`

  Przykład:

```
uses
      Windows, Messages, SysUtils, Variants, Classes, Graphics,
      Controls, Forms, Dialogs, StdCtrls, DateUtils;
…
var
      start, stop : TDateTime;
       czasWynikowy: Int64;
begin
      start := Now;
      //wykonanie algorytmu
      stop := Now;
      czasWynikowy := MilliSecondsBetween(start, stop);
end;
```

- Wykorzystując funkcję `GetTickCount`
  `function GetTickCount: DWORD; stdcall;` - df

  Przykład:

```
var
      start, stop, czasWynikowy : Integer;
begin
      start := GetTickCount;
      //wykonanie algorytmu
      stop := GetTickCount;
      czasWynikowy := start - stop;
end;
```

Uzyskany wynik (dowolnym z powyższych sposobów) można przekonwertować do łańcucha za pomocą funkcji: `function IntToStr(Value: Integer): string; overload;`

```
var
      s: string;
begin
      ...
      s := IntToStr(czasWynikowy);
      ...
end;
```

2.16. Obsługa wyjątków: try … except … end;

Składnia:

```
try w połączeniu z except
try
  <instrukcje programu>
except
  on <typ wyjątku> do begin
  <obsługa wyjątku>
 end;
end;
```

Słowo kluczowe ***try*** wyznacza początek bloku obsługi wyjątków ***try…except***. W jego ramach wykonywane są **<*instrukcje programu*>**. Jeżeli wykonywanie którejś instrukcji spowoduje wystąpienie wyjątku wykonywana jest sekcja rozpoczynająca się od słowa ***except***, natomiast w przypadku bezbłędnego wykonania **<*instrukcji programu*>** sekcja ***except*** pozostaje niewykorzystana. Dyrektywy ***on*** w sekcji ***except*** służą do zróżnicowanej obsługi poszczególnych typów wyjątków; jeżeli nie zostanie użyta żadna dyrektywa ***on***, wszystkie wyjątki obsługiwane będą w taki sam sposób.

Przykład programu konsolowego:

```
Program Obsluga;

{$APPTYPE CONSOLE}
```

```
Var
  R1, R2 : Double;
begin
  While True do
  begin
    try
      Write('Podaj liczbę rzeczywistą:');
      Readln(R1);
      Write('Podaj inną liczbę rzeczywistą:');
      Readln(R2);
      Writeln ('Teraz spróbuję podzielić wprowadzone liczby ...');
      Writeln ('Iloraz wynosi ', (R1/R2));
    except
      on EZeroDivide do
         Writeln('Próba dzielenia przez zero!');
      on EInOutError do
         Writeln('Nieprawidłowa postać liczby!');
    end;
  end;
end;
```

Inny przykład (fragment aplikacji - procedura):

```
procedure TForm1.Button1Click(Sender: TObject);
var
  liczba, liczba1, wynik: Integer;
begin
  try
    liczba := StrToInt(Edit1.Text);
    liczba1 := StrToInt(Edit2.Text);
    wynik  := liczba div liczba1;
  except
    on EDivByZero do
        MessageDlg('Can not divide by zero!', mtError, [mbOK], 0) ;
     on EInOutError do
         MessageDlg('Nieprawidłowa postać liczby!', mtError, [mbOK], 0) ;
 end;
end;
end.
```

2.17. Procedury i funkcje

Funkcja jest to wydzielony blok kodu, który wykonuje określoną czynność i zwraca wynik.
Procedura jest to wydzielony blok kodu, który wykonuje określoną czynność, lecz nie zwraca wyniku.
Parametr jest wartością przekazywaną do funkcji albo procedury, który modyfikuje lub określa zakres jej działania.

Każda funkcja w Object Pascalu ma predefiniowaną zmienną lokalną o nazwie **Result**. Używana jest ona do przechowywania wyniku zwracanego przez funkcję. Wystarczy więc przypisać zmiennej Result wartość, którą funkcja ma zwrócić jako wynik.

Deklaracja funkcji i procedury ma następującą postać:

```
procedure nazwa (lista parametrów);
deklaracje zmiennych lokalnych (nie są widoczne poza ciałem procedury)
begin
      blok instrukcji;
end;
```

```
function nazwa (lista parametrów):typ zwracanej wartości;
deklaracje zmiennych lokalnych (nie są widoczne poza ciałem funkcji)
begin
      blok instrukcji;
end;
```

Przykład:

```
procedure Up20 (I: Integer);
begin
      if I > 20 then
       writeln('Wiecej niż 20');
end;

function MyFunc: Integer;
var
 I: Integer;
Begin
      I := 22;
      MyFunc := Inc(I);          //do zmiennej wynikowej przypisane 23
      Result := Result * 2;      // do zmiennej wynikowej przypisane 46
      MyFunc := Result + 1;      //funkcja zwroci 47
end;
```

Wywołanie powyższej funkcji i procedury np.:

```
I = MyFunc;
Up20(I);
```

Parametry do funkcji/procedur mogą być przekazywane przez:

- Wartość

  Powoduje to automatyczne utworzenie lokalnej kopii tak przekazywanego parametru i wykonywanie na nim wszystkich operacji w treści procedury/funkcji. Oryginalny parametr nie zostaje naruszony, jego wartość nie ulega zmianie.

  Przykład:

  ```
  procedure MyFunc(S: String);
  ```

- Referencję (inaczej nazywane przez adres lub przez zmienną)

  Powoduje, że w treści procedury/funkcji pod nazwą parametru kryje się rzeczywisty parametr aktualny i wszystkie operacje wykonywane są bezpośrednio na nim. Jest to sposób na przekazanie przez procedurę wartości zwrotnej do programu wywołującego – po zakończeniu procedury/funkcji wartość parametru odzwierciedla wszystkie wykonywane na nim operacje. Deklaracja parametru przekazywanego przez referencję polega na poprzedzeniu jego nazwy słowem kluczowym `var`.

  Przykład:

  ```
  procedure MyFunc(var I: Integer);
  begin
        I := Inc(I);
  end;
  ```

```
…
var
 M: Integer;
begin
      M := 12;
      MyFunc(M);
      Writeln(M); //M ma teraz wartość 13
      …
end;
```

- Stałą

  Łączy zalety dwóch powyższych sposobów. Z jednej strony parametr podlega tym samym regułom co parametr przekazywany przez wartość, z drugiej jednak strony na stosie odkładany jest adres parametru (nie jego kopia) przy czym kompilator nie dopuści aby parametr ten znalazł się po lewej stronie przypisania. Deklaracja parametru przekazywanego przez stałą polega na poprzedzeniu jego nazwy słowem kluczowym `const`.
  Przekazanie parametru przez stałą jest wskazane w przypadku, gdy parametr jest tylko parametrem wejściowym i nie zamierzamy wykorzystywać jego kopii lokalnej jako obszaru roboczego.

  Przykład:

```
procedure MyFunc(const I: Integer);
var
 N: Integer;
Begin
      N := 1;
      I := Inc(N); //ta instrukcja spowoduje blad kompilacji
      N := Inc(I); //poprawne
end;
…
var
 M: Integer;
begin
      M := 12;
      MyFunc(M);
      Writeln(M); //M ma teraz wartość 13
      …
end;
```

3. Materiały

[1] Delphi Language Guide
[2] http://4programmers.net/
[3] http://www.marcocantu.com/ (Essential Pascal, Essential Delphi)
[3] Delphi 6. Vademecum profesjonalisty. Tom 1.

Dodatkowe strony:

http://delphi.cartall.com.pl/
http://delphi.icm.edu.pl/
http://cpw.net.pl/