Delphi – Laboratorium 3

1. 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 Delphi 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 := I + 1; //poprawne
end;
…
var M: Integer;
begin
   M := 12;
   MyFunc(M);
   Writeln(M); //M ma nadal wartość 12
   …
end;
```

2. Zabezpieczanie programów przed błędami

Warunkiem i (lub) stanem wyjątkowym nazywamy wystąpienie błędu lub innego zdarzenia, które przerywa normalne wykonywanie programu.

W języku Delphi warunki i stany wyjątkowe są reprezentowane przez typy klasowe (wszystkie są potomkami typu *Exception*).

Obsługę błędów, które normalnie powodują przerwanie wykonywania programu (np. brak pamięci, naruszenie jej ochrony, dzielenie przez zero etc.) zapewnia moduł *System.SysUtils.*

a. Procedura *Val* (bez generowania wyjątków)

```
procedure Val (S: string; var liczba; var kod: Integer);
```

Procedura powoduje przekształcenie łańcucha *S* (pierwszy parametr) na liczbę i zapamiętuje wynik pod zmienną *liczba* podaną jako drugi argument. Łańcuch musi być poprawną liczbą całkowitą lub rzeczywistą. Jeśli łańcuch nie jest poprawną liczbą

to pod zmienną *kod* znajdzie się pozycja pierwszego błędnego znaku. Jeśli wszystko przekształcenie przebiegnie pomyślnie to pod zmienną *kod* zostanie podstawione 0.

```
var s1 : string;
         i, kod : Integer;
begin
s1 := '1234';
      Val(s1, i, kod); //kod=0, zamiana przebiegła pomyślnie
      s1 := '234m6';
      Val(s1, i, kod); //kod=4, błąd
end;
```

b. Instrukcja wywoływania stanów wyjątkowych *raise*

Służy do wywoływania warunków lub stanów wyjątkowych predefiniowanych lub zdefiniowanych przez programistę.

Instrukcja może mieć takie postacie:

```
//pozwala na generowanie wyjątków poza blokiem try..except
raise stan-wyjatkowy
//może zostać użyta tylko w bloku except instrukcji try…except
//powoduje wyświetlenie domyślnego komunikatu o błędzie
raise
```

Stan wyjątkowy to wyrażenie typu klasowego (obiekt).

Przykład (liczba jest zmienną typu *Integer*):

```
if (liczba < -10) or (lizcba > 10)
   then raise EIntOverflow.Create('Wartość poza zakresem');
```

Jeśli wartość zmiennej *liczba* będzie poza zakresem to zostanie wyświetlone okienko z napisem.

c. Instrukcja obsługi warunków *try…except*

Podstawowa instrukcja obsługi warunków i stanów wyjątkowych.

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('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('Błędna postać liczby!', mtError, [mbOK], 0) ;
  end;
end;
end.
```

d. Instrukcja obsługi warunków *try…finally*

W odróżnieniu od bloku *except* w instrukcji *try…except* kod znajdujący się po słowie *finally* będzie wykonywany zawsze, niezależnie od tego czy wystąpi wyjątek. Instrukcji tej używa się np. w przypadku gdy konieczne jest zwolnienie pamięci czy zamknięcie pliku, a nie jesteśmy pewni, czy podczas operacji nie wystąpi żaden błąd.

```
try
  <instrukcje programu>
finally
  <instrukcje programu>
end;
```

Przykład:

```
Reset (plik);     //otwarcie pliku
try
      try
       …          //przetwarzanie pliku
      finally
            CloseFile(plik); //zamknięcie pliku
      end;
except
      on EInOutError do
      … //obsługa błędów przetwarzania pliku (błędów we-wy)
end;
```