Wydział Zarządzania AGH
Katedra Informatyki Stosowanej

Procedury i funkcje

Programowanie w VBA

Procedury i funkcje

- **Wprowadzenie**
- **Budowa procedur i funkcji**
- **Przekazywanie parametrów**
- **Funkcje wbudowane**
- **Rekurencja**

Wprowadzenie

W czasie tworzenia programu zwykle wielokrotnie korzysta się z takich samych ciągów instrukcji. W języku VBA taki ciąg można zdefiniować osobno. Mając te definicje, programista - zamiast powtarzać takie same ciągi instrukcji - powołuje się na odpowiednią definicję. Definicje te nazywamy ***procedurami***, a powołania się na nie w programie - ***wywołaniami procedur.***

Wprowadzenie

- W języku VBA można opisywać procedury w dwóch postaciach:
  1. umożliwiającej podobne użycie procedury jak funkcji matematycznej; mówimy wtedy, że jest to procedura funkcyjna (funkcja);
  2. umożliwiającej wywołanie procedury jako osobnej instrukcji.
- Procedurą (funkcją) nazywamy wyodrębnioną część programu, stanowiącą pewną całość, mającą jednoznaczną nazwę i ustalony sposób wymiany danych z pozostałymi częściami programu.

Budowa procedury i funkcji

- Różnica między procedurą a funkcją polega na sposobie przekazywania wartości. Zadaniem procedury jest wykonanie pewnej sekwencji instrukcji w celu obliczenia jednej lub kilku wartości, natomiast funkcja służy do obliczenia jednej wartości.
- Zaletą funkcji jest możliwość bezpośredniego korzystania z nich w wyrażeniach (tak samo jak zmiennych).
- W przeciwieństwie do procedury funkcyjnej, która zwraca wartość, procedura **Sub** nie może być używana w wyrażeniach.

Budowa procedury i funkcji

- Procedura zaczyna się od instrukcji deklaracji procedury ***Sub*** i kończy instrukcją ***End Sub***.

  [Private | Public] Sub *nazwa_procedury* ([*argumenty*])

  [*instrukcje*]

  [**Exit Sub**]

  [*instrukcje*]

  End Sub

Budowa procedury i funkcji

- Funkcja zaczyna się od instrukcji deklaracji funkcji ***Function*** i kończy instrukcją ***End Function***.

  [Private | Public] Function *nazwa_funkcji* ([*argumenty*]) [**as** typ]

  [*instrukcje*]

  nazwa_funkcji =Wartość

  [**Exit Function**]

  [*instrukcje*]

  [*nazwa_funkcji =Wartość*]

  End Function

Budowa procedury i funkcji

- Instrukcja deklaracji procedury nadaje nazwę procedurze , wskazuje jej początek i koniec oraz wymienia wszystkie argumenty, jakie mogą być przekazywane pomiędzy procedurą a wołającym ją programem. Instrukcja ***End Sub*** wskazuje koniec procedury. Wszystko, co się znajduje między tymi dwiema instrukcjami stanowi treść procedury, odpowiedzialną za wykonanie zadania przypisanego tej procedurze. Lista argumentów (jeśli jest) stanowi połączenie między procedurą wołającą a procedurą wołaną.

Budowa procedury i funkcji

- Procedura **Private Sub** jest dostępna tylko w module, w którym została zadeklarowana. Procedura **Public Sub** jest dostępna we wszystkich procedurach i we wszystkich modułach w aplikacji.
- Cały wykonywalny kod musi być umieszczony wewnątrz procedur. Nie można stworzyć procedury **Sub** wewnątrz innej procedury **Sub.**
- Słowo kluczowe **Exit Sub** jest używane do bezwarunkowego zakończenia wykonywania procedury **Sub**. Wykonywanie programu jest kontynuowane od rozkazu następnego za wywołaniem procedury **Sub**. Dowolna liczba rozkazów **Exit Sub** może znajdować się w dowolnych miejscach wewnątrz procedury.

Budowa procedury i funkcji

- Parametry (argumenty) umieszczone w nagłówku procedury (funkcji) nazywamy *parametrami formalnymi*, a parametry użyte w miejscu wywołania - *parametrami aktualnymi*.
- Liczba parametrów aktualnych musi być równa liczbie parametrów formalnych. Z reguły typ parametrów aktualnych musi być zgodny z typem odpowiednich parametrów formalnych (za wyjątkiem, gdy parametry przekazywane są przez wartość).
- Nazwy zmiennych w liście argumentów powinny być oddzielone od siebie przecinkami.

Budowa procedury i funkcji – lista argumentów

- [**Optional**] [**ByVal** | **ByRef**] *nazwa_arg* [**As** *typ*] [=*domyślna_wartość*]
- **Optional** (opjcjonalny); słowo kluczowe, określające, że argument nie jest wymagany. Jeżeli zostanie użyty, wszystkie argumenty listy muszą być również opcjonalne i zadeklarowane przy użyciu słowa **Optional**.
- **ByVal** (opcjonalne); określa, że wartość zostanie przekazana przez wartość. **ByRef** (opcjonalne, domyślne); określa, że wartość zostanie przekazana przez referencję.
- *domyślna_wartość* (opcjonalne); wartość domyślna argumentu. Może to być stała lub wyrażenie stałe, dotyczy tylko argumentów **Optional**.

Budowa procedury i funkcji

```
Sub Przyklad(k As Integer)  'początek procedury
  If k=0 Then  'sprawdzenie czy k=0
      Exit Sub  'jeżeli tak to opuść procedurę
  Else
      k=k*5  'jeżeli nie to wykonaj obliczenie
  End If
End Sub  'koniec procedury
```

```
Function Przyklad(k As Integer, Optional b As Byte = 3)
  If k=0 Then
      Przyklad = b
      Exit Function  ’opuść funkcję
  Else
      k = k * 7
      Przykład = k  'wykonaj obliczenie i zwróć wartość
  End If
End Function  'koniec funkcji
```

Budowa procedury i funkcji

- Zmienne używane wewnątrz procedur **Sub** dzielą się na dwie kategorie: lokalne - zadeklarowane wewnątrz procedury i zewnętrzne - zadeklarowana poza procedurą. Zmienne, które są zadeklarowane wewnątrz procedury są zawsze lokalne dla procedury. Zmienne, które są używane w procedurze, a nie są w niej zadeklarowane są również lokalne, z wyjątkiem tych, które zostały zadeklarowane poza procedurą (na wyższym poziomie).
- Może wtedy powstać konflikt nazw, jeżeli procedura odnosi się do niezadeklarowanej zmiennej, która ma taką samą nazwę jak inna procedura, zmienna lub stała. Zostanie przyjęte, że procedura odwołuje się do nazwy na poziomie modułu. Aby uniknąć takich konfliktów należy deklarować zmienne.

Budowa procedury i funkcji

- Aby wywołać procedurę wystarczy umieścić jej nazwę wewnątrz innej procedury w miejscu, w którym ma nastąpić jej wykonanie. Jeśli wołana procedura wymaga podania argumentów, należy umieścić je po nazwie procedury. Opcjonalnie można użyć **Call** *nazwa_proc*.
- W podanym poniżej przykładzie procedura PROC1 woła procedurę PROC2.

```
Sub Proc1()
    Proc2
    jakieś instrukcje
    Call Proc2
End Sub
Sub Proc2()
    treść procedury
End Sub
```

Przekazywanie parametrów

- Wywołanie procedury (funkcji) powoduje następujący ciąg akcji:
  - obliczane są wartości parametrów przekazywanych przez wartość; parametry przekazywane przez zmienną są zawsze nazwami zmiennych,
  - kolejnym parametrom formalnym z nagłówka procedury (funkcji) przypisywane są wartości lub nazwy kolejnych parametrów aktualnych,
  - dla tak określonych parametrów są wykonywane obliczenia zawarte w instrukcjach procedury (funkcji).

Przekazywanie parametrów

- W zależności od sposobu przekazywania wartości do procedury (funkcji) tzn. sposobu zastępowania parametrów formalnych argumentami, wyróżniamy dwa rodzaje parametrów:
  1. parametry przekazywane przez wartość,
  2. parametry przekazywane przez zmienną określonego typu.
- Deklaracje tych parametrów mają postać:
  - **ByVal** *NazwaArg* [**As** *typ*]
  - **ByRef** *NazwaArg* [**As** *typ*]

Przekazywanie parametrów

- Parametry przekazywane przez wartość
  Jeżeli parametr przekazywany jest przez wartość, to w wywołaniu procedury (funkcji) odpowiadający mu argument musi być wyrażeniem.
- W chwili wywołania procedury (funkcji) parametrom formalnym przypisywane są, obliczone właśnie w tym momencie, wartości odpowiednich parametrów aktualnych. Jeśli parametrem jest zmienna i w treści procedury zmienimy jej wartość, to po wyjściu z procedury wartość tej zmiennej nie ulegnie zmianie tj. będzie taka sama jak przed wywołaniem procedury.

Przekazywanie parametrów

- Parametry przekazywane przez wartość

  Jako że parametrem aktualnym musi być wyrażenie, w tym przypadku wywołanie procedury zmiana(*y*) mogłoby mieć postać:

  - zmiana(3+x*2);
  - zmiana(3);
- W pierwszym przypadku, w momencie wywołania procedury ***y*** ma wartość 3+x*2, w drugim 3.

Przekazywanie parametrów

- Parametry przekazywane przez zmienną
  Argumenty odpowiadające parametrom przekazywanym przez zmienną muszą być zmiennymi, przy czym typ argumentu musi być **identyczny** z typem parametru formalnego.
- Po wyjściu z procedury (funkcji) parametr aktualny będzie miał wartość taką jak odpowiadający mu parametr formalny w chwili zakończenia działania procedury. Jeśli więc chcemy przekazać na zewnątrz procedury wartość parametru, który zmienia się w trakcie działania procedury, to może się to odbyć tylko wtedy, gdy parametr przekazywany jest przez zmienną.

Przekazywanie parametrów

- Procedury bez parametrów

  Gdy w programie pewien ciąg instrukcji powtarza się często lub chcemy zwiększyć czytelność programu, można ten ciąg zastąpić procedurą bez parametrów. Procedurę taką wywołujemy przez podanie nazwy. W procedurach bez parametrów można wykonywać działania na zmiennych (stałych) globalnych oraz na zmiennych i stałych lokalnych w danej procedurze. Po wykonaniu procedury bez parametrów zmienne globalne mają wartości takie, jakie zostały im nadane w tej procedurze, natomiast zmienne lokalne nie są dostępne poza tą procedurą.

Przekazywanie parametrów

- Przy definiowaniu i wywoływaniu procedur (funkcji) należy przestrzegać poniższych zasad:
  - Zmienne mające w danej procedurze charakter lokalny, należy deklarować jako obiekty lokalne.
  - W definicji procedury wskazane jest umieszczanie (najlepiej bezpośrednio po nagłówku) komentarzy objaśniających cel procedury, jej parametry oraz zmienne lokalne i nielokalne.

Przekazywanie parametrów

- Wybór rodzaju parametrów (tj. sposobu ich przekazywania) powinien być dokonany wg następujących reguł:
  - Jeśli parametr służy do przekazywania otoczeniu procedury efektów jej działania, to parametr ten musi być przekazany przez zmienną, a co za tym idzie, odpowiadający mu argument musi być zmienną,
  - W pozostałych przypadkach stosujemy przekazywanie parametru przez wartość. Jest ono zalecane jako bezpieczniejsze, gdyż chroni zmienne nielokalne przed niepożądaną zmianą wewnątrz procedury.
  - Unikamy przenoszenia przez wartość dużych struktur danych, gdyż przy każdym wywołaniu procedury rezerwowana jest pamięć, w której przechowywana jest wartość argumentu.

Przekazywanie parametrów

```
Function Przyklad(ByVal k As Integer, Optional b As Byte = 3)
If k = 0 Then
    Przyklad = b
    Exit Function 'opuść funkcję (bez Exit też jest OK)
Else
    k = k * 7
    Przyklad = k
    'wykonaj obliczenie i zwróć wartość
End If
End Function

Sub przekazywanie_argumentow()
Dim x As Integer
Dim y As Single
x = InputBox("Podaj x")
y = Przyklad(x) + 10
MsgBox "x= " & Str(x) & "  y= " & Str(y)
y = Przyklad(x,10) + 10
MsgBox "x= " & Str(x) & "  y= " & Str(y)
End Sub
```

Funkcje wbudowane

- Funkcje matematyczne
- Funkcje daty i czasu
- Funkcje tekstowe
- Funkcje formatujące
- Funkcje wejścia-wyjścia
- Funkcje konwersji danych
- Funkcje testujące dane
- Inne funkcje

Funkcje wbudowane - matematyczne

Rnd	liczba losowa <0, 1)
Abs(X)	wartość bezwzględna
Sgn(X)	znak liczby
Fix(X)	część całkowita (po usunięciu części ułamkowej)
Int(X)	część całkowita (po zaokrągleniu w dół)
Log(X)	logarytm naturalny
Exp(X)	*e* do potęgi
Sqr(X)	pierwiastek kwadratowy
Sin(X)	sinus
Cos(X)	cosinus
Tan(X)	tangens

Funkcje wbudowane - matematyczne

Fix(X) część całkowita (po usunięciu części ułamkowej)

```
Dim varLiczba As Variant
varLiczba=Fix(99.2)  'varLiczba=99
varLiczba=Fix(99.8)  'varLiczba=99
varLiczba=Fix(-99.2)  'varLiczba=-99
varLiczba=Fix(-99.8)  'varLiczba=-99
```

Int(X) część całkowita (po zaokrągleniu w dół)

```
Dim varLiczba As Variant
varLiczba=Int(99.2)  'varLiczba=99
varLiczba=Int(99.8)  'varLiczba=99
varLiczba=Int(-99.2)  'varLiczba=-100
varLiczba=Int(-99.8)  'varLiczba=-100
```

Funkcje wbudowane - tekstowe

Format	formatowanie łańcuchów
Len	zwraca długość łańcucha
Trim	usuwa początkowe i końcowe spacje
Left	zwraca część łańcucha od lewej strony
Right	zwraca część łańcucha od prawej strony
Space	zwraca łańcuch wypełniony spacjami
String	zwraca łańcuch wypełniony znakami

Funkcje wbudowane - tekstowe

Format(*Wyrażenie*, *Format*) formatowanie łańcuchów

varStr = Format(5459.4, "##,##0.00") 'zwraca "5,459.40"
varStr = Format(334.9, "###0.00") 'zwraca "334.90"
varStr = Format(5, "0.00%") 'zwraca "500.00%"
varStr = Format("HELLO", "<") 'zwraca "hello"
varStr = Format("To jest TO", ">") 'zwraca "TO JEST TO"

Len*(Łańcuch)* zwraca długość łańcucha

strString=„abcdef"
varDlugosc=Len(strString)

Trim*(Łańcuch)* usuwa początkowe i końcowe spacje

Dim varNapis As Variant
varNapis=Trim(" - tekst - ") 'varNapis="- tekst -"

Funkcje wbudowane - tekstowe

Left(*Łańcuch, Długość)* zwraca część łańcucha od lewej strony

```
Dim varNapis As Variant
varNapis=Left("Jasio idzie do kina",7)  'varNapis="Jasio i"
varNapis=Left("Jasio idzie do kina",60)  'varNapis="Jasio idzie do kina"
```

Right(*Łańcuch, Długość)* zwraca część łańcucha od prawej strony

Space*(Długość)* zwraca łańcuch wypełniony spacjami

```
varNapis=„Witaj" & Space(5) & „Jasiu"
  'varNapis=„Witaj     Jasiu"
```

String*(Długość, Znak)* zwraca łańcuch wypełniony znakami

```
Dim varNapis As Variant
varNapis=String(7,42)          'varNapis="*****"
varNapis=String(4,"Jasio")     'varNapis="JJJJ"
```

Funkcje wbudowane – konwersji danych

Chr*(KodZnaku)* kod ASCII na znak

```
Dim varZnak As String
varZnak=Chr(65)  ' varZnak="A"
varZnak=Chr(97)  ' varZnak="a"
```

Str*(Wartość)* liczba na tekst

```
Dim varString As String
varString=Str(459)  'varString=" 459"
varString=Str(-459.65)  'varString="-459.65"
varString=Str(459.001)  'varString=" 459.001"
```

Funkcje wbudowane – konwersji danych

Val(*Łańcuch*) tekst na liczbę

```
Dim varLiczba As Single
varLiczba=Val("459")  'varLiczba=459
varLiczba=Val("45 9. 65")  'varLiczba=459.65
varLiczba=Val("459 i 001")  'varLiczba=459
```

CInt(*Wyrażenie*) wartość na Integer

Argument *Wyrażenie* jest wymagany. Może to być dowolne poprawne wyrażenie numeryczne w zakresie -32,768 do 32,767; część ułamkowa jest zaokrąglana.

```
Dim varDouble As Double, varInt As Integer
varDouble=2125.5678
varInt=CInt(varDouble)  'varInt=2126
```

Rekurencja

a) tradycyjnie (iteracyjnie):

```
Function silnia1 (n As Byte) As Double
 dim i As Byte
 dim s As Double
 if n>170 then
    silnia1 = 0
 else
    s=1
     for i=2 to n
           s = s * i
    next i
    silnia1 = s
end if
End Function
```

Rekurencja

b) rekurencyjnie:

```
Function silnia2 (n As Byte) As Double
   if n>170 then
         silnia2=0
   else
         if n=0 then
               silnia2=1
         else
               silnia2=n*silnia2(n-1)
         end if
    end if
End Function
```

Rekurencja

- Każdą procedurę (funkcję), która jest zapisana rekurencyjnie można zapisać iteracyjnie. Wybór sposobu zapisu procedury zależy od programisty. Warto się jednak zawsze zastanowić, który sposób jest efektywniejszy.
  Z reguły zapis iteracyjny jest naturalny i zastąpienie go zapisem rekurencyjnym jest kłopotliwe.
- Porównując funkcje z dwóch poprzednich przykładów widzimy, że funkcja ***silnia2*** jest w zapisie dużo krótsza niż ***silnia1***, ale obliczenia będą dłużej wykonywane, gdyż każdorazowo do obliczenia wartości kolejnej *silni* są obliczane wartości wszystkich poprzednich *silni*. Natomiast przy zapisie iteracyjnym do obliczenia wartości *silni* wykorzystujemy wartości poprzedniej *silni* zapamiętanej w zmiennej ***s***.

Przykład

Korzystając z rozwinięcia w szereg Taylora napisać funkcję:

$$\cosh(x) = \sum_{n=0}^{\infty} \frac{1}{(2n)!} x^{2n}$$

$$\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \cdots$$

Funkcje Excela w VBA

W kodzie VBA możemy wykorzystywać wbudowane funkcje arkusza. Zapis:

Application.WorksheetFunction.*NazwaFunkcji*([argumenty])

Np.

```
Pi = Application.WorksheetFunction.PI()
Si = Application.WorksheetFunction.FactDouble(120)
x =  Application.WorksheetFunction.NormInv(Rnd, xSt, odch)
```