INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Makra v OpenOffice.org Calc

Petr Ponížil

Tento výukový materiál vznikl jako součást grantového projektu Gymnázia Kroměříž s názvem Beznákladové ICT pro učitele realizovaného v letech 2009–2012. Projekt je spolufinancován z Evropského sociálního fondu a státního rozpočtu České republiky.

Předmluva

Makra v OpenOffice.org (dále jen OOo) Calc jsou velmi silný nástroj rozšiřující možnosti této aplikace. Od uživatelů nejsou požadovány žádné znalosti objektového programování a ani se z tohoto textu objektové programováni nenaučí.Tento text je zaměřen na uživatele, kteří už mají nějaké základy programování a potřebují začít tvořit makra. Nejsou zde proto například podrobně probírány datové typy nebo struktury pro řízení běhu programu, ale jsou jen stručně popsány, aby například uživatel, který zná cyklus For z jazyka C, Pascalu, Fortranu nebo Basicu, dokázal po nahlédnutí tohoto textu napsat tento cyklus i v OOo Basicu. Na následujících několika stranách se pokusíme shrnout základní informace, které uživateli umožní začít psát svá vlastní makra pro OOo Calc.

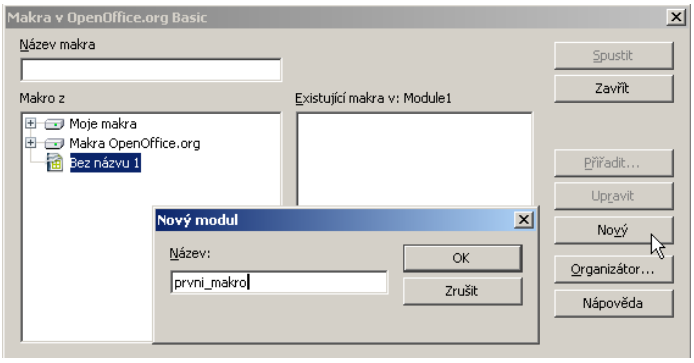
Další výhodou OOo Basicu je, že se uživatel nemusí trápit s programováním grafického rozhraní pro vstupy a výstupy programu. Načte nebo zadá vstupní data do formuláře v sešitu Calcu makro si data načte přímo z buněk sešitu a výsledkovou sestavu opět uloží do sešitu.

OOo Basic se někdy také nazývá StarBasic podle předchůdce současného OpenOffice.org, který se jmenoval Star Office.

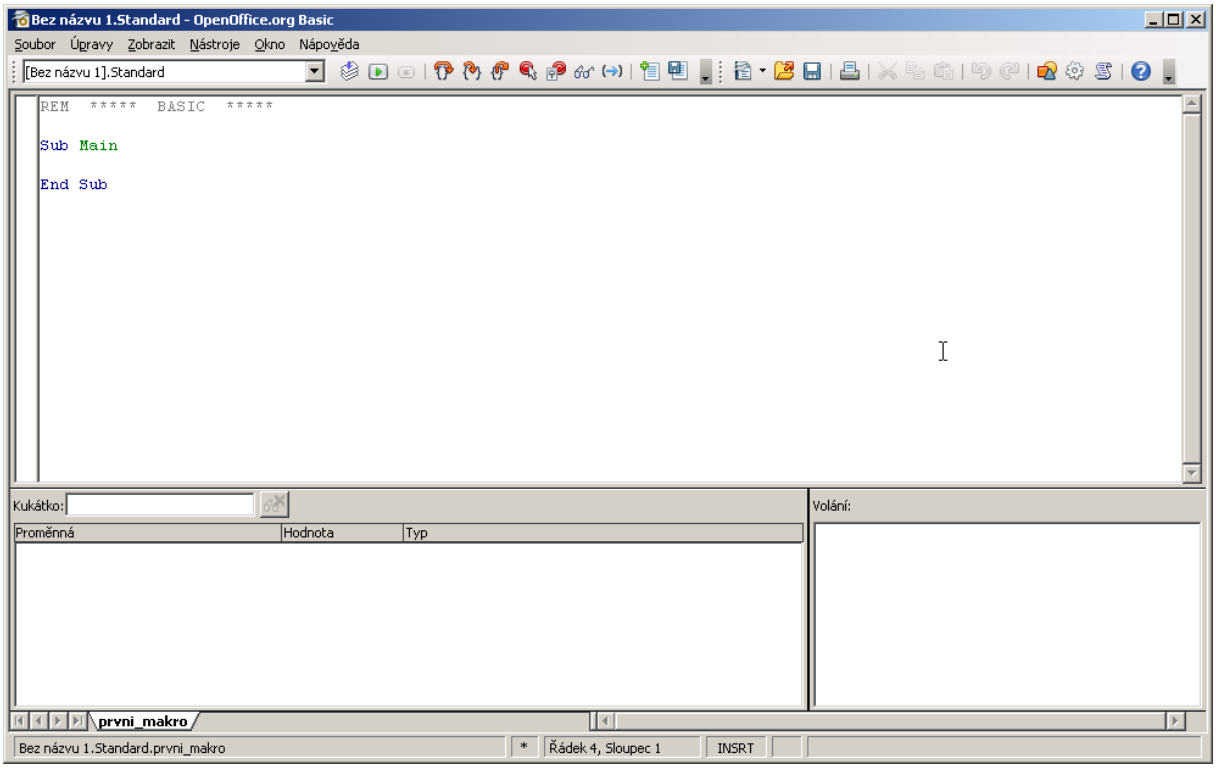
Práce s makry

Vytvoření makra

Otevřte nebo vytvořte nový OOo dokument. Dialog pro správu maker (obr. 1) otevřete tak, že v menu klikněte na **Nástroje > Makra > Správce maker > OpenOffice.org Basic**. Na levé straně dialogu klikněte na jméno otevřeného souboru (v našem případě „Bez názvu 1“) a na pravé straně po kliknulím na tlačítko Nový... zadejte jméno nového modulu (v našem případě „prvni_makro“). Po kliknutí na OK se otevře vývojové prostředí OOo Basic IDE (obr 2). Práci s vývojovým prostředím se budeme později zabývat důkladněji.



Obr. 1: Dialog pro správu maker - vytvoření nového makra



Obr. 2: OOo Basic IDE

Vývojové prostředí

V tomto vývojovém prostředí už je předchystán základ prvního makra `Sub Main` – hlavička makra a `End Sub` – jeho konec. Makro upravíme do podoby:

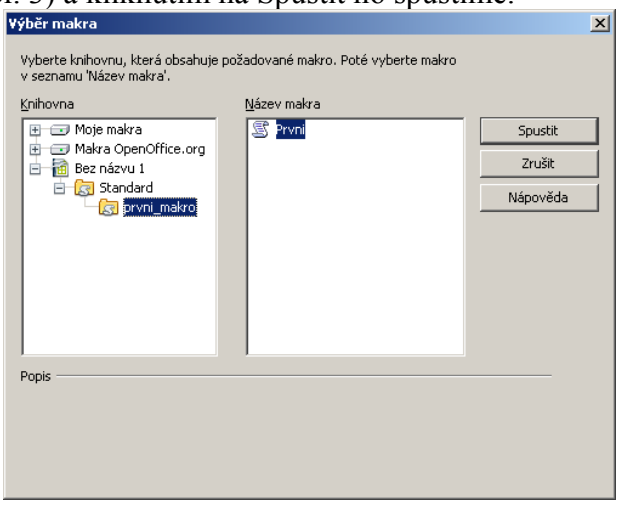
```
Sub Prvni
   print "Hello World"
End Sub
```

Nyní můžeme zavřít vývojové prostředí nebo jen přepnout do okna OOo Calcu a spustit makro.

Spuštění makra

Spuštění z menu

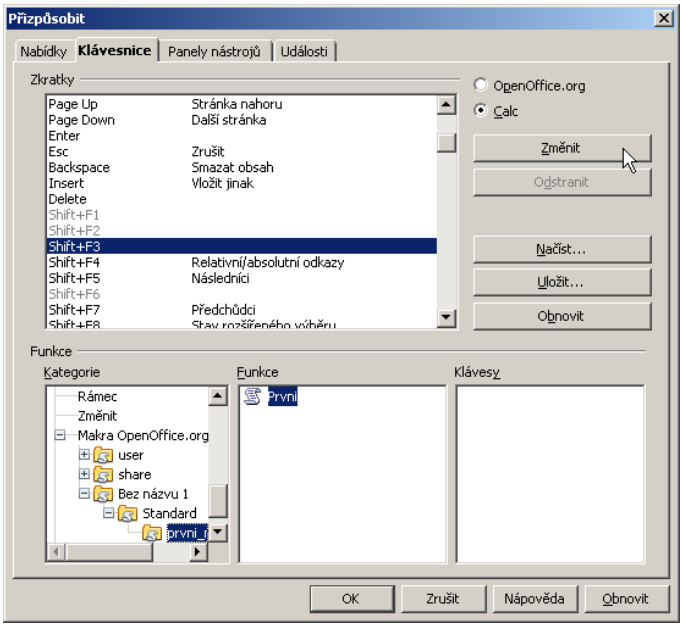
Hotové makro lze spustit několika způsoby. Základní metodou je spouštění z menu **Nástroje > Makra > Spustit makro**. Makro nalezneme v knihovně Bez názvu 1\Standard\první_makro\Prvni (obr. 3) a kliknutím na Spustit ho spustíme.



Obr. 3: Výběr makra pro spuštění

Spuštění klávesovou zkratkou

Spouštění makra přes Správce maker je docela zdlouhavé. Proto existuje možnost přiřadit makru klávesovou zkratku. Ta se nepřiřazuje ve Správci maker, jak by se dalo očekávat třeba podle Excelu (toto umístění ale má svou logiku – všechny klávesové zkratky se v OOo nastavují na jednom místě), ale v Nástroje | Přizpůsobit... v dialogu přizpůsobit přejdeme na záložku Klávesnice (obr. 4). v horní části dialogu v okně Zkratky je seznam klávesových zkratek a jim přiřazené akce.



Obr. 4: Přiřazení klávesové zkratky

Makro můžeme přiřadit pouze klávesové zkratce, která není šedá (jako je šedá např. Shift+F1 na obr. 4) – zvolme Shift+F3. v dolní části dialogu v okně Kategorie vybereme Makra OpenOffice.org a nalezneme knihovnu do které jsme uložili makro. v okně Funkce je seznam maker ve zvolené knihovně, vybereme požadované makro a po kliknutí na tlačítko Změnit se v okně Klávesy objeví zvolená klávesová zkratka. Po přiřazení klávesových zkratek makrům tkačítkem OK zavřeme dialog Přizpůsobit. Od tohoto okamžiku lze makro Prvni spouštět pomocí klávesové zkratky Shift+F3.

Spuštění tlačítkem

Makro lze přiřadit také tlačítku. Tlačítko lze vytvořit tak, že zobrazíme panel pro ovládací prvky formuláře - **Zobrazit > Panely nástrojů > Ovládací prvky formuláře**. Důležité je tlačítko v prvním řádku vpravo, obsahující tužku a pravítko. Tímto tlačítkem zapínáme a vypínáme režim návrhu formuláře. Je-li tento režim zapnut, můžeme prvky formuláře editovat, ale nejsou zatím funkční. Používat prvky formuláře lze až po vypnutí režimu návrhu. Zapněme tedy režim návrhu formuláře (tlačítka panelu jsou aktivní). Klikněme na ikonu tlačítka ve čtvrtém řádku vpravo a levým tlačítkem myši vyznačme na listu Calcu polohu a velikost tlačítka. Vzniklo tlačítko popsané „Tlačítko“. Klikněme na tlačítko pravým tlačítkem myši a v kontextovém menu vyberme položku Ovládací prvek. Na kartě „Obecné“ změňme Popisek na Makro První a případně lze ještě definovat písmo. Na kartě „Události“ klikněme na Stisknuto tlačítko myši a v dialogu „Přiřadit akci nejdříve v levém části zkontrolujme, že je vybrána položka Stisknuto tlačítko myši a tlačítkem Makro v pravé části dialogu vyberme makro, které se má spustit po stisknutí tlačítka myší. Kliknutím na ikonu režim návrhu formuláře vypněme tento režim a tlačítko je připraveno k použití.

Obr. 5: Panel nástrojů „Ovládací prvky formuláře“

Popis jazyka OOo Basic

Základní dovednosti v OOo Basicu

Komentáře

Komentář je text který interpret OOo Basicu ignoruje a slouží pro autora nebo jeho pokračovatele k vysvětlení kódu. Komentář začíná klíčovým slovem REM, které může být kdekoliv na řádku. Za komentář se považuje text za tímto klíčovým slovem. Druhou možností jak označit komentář he znak apostrofu - '. Opět je za komentář považováno vše ze tímto znakem.
Příklady komentářů:

```
REM Komentář zahrnuje celý řádek
x = 0  REM x = 0 je příkaz, text za REM je komentář
' Komentář zahrnuje celý řádek
x = 0  ' x = 0 je příkaz, text za REM je komentář
```

Příkazy

OOo Basic osahuje zpravidla každý příkaz na novém řádku a znak nového řádku je pak oddělovačem příkazů. Je-li příkaz příliš dlouhý, může být rozdělen na několik řádků pomocí znaku podtržítka - _.
Příklad:

```
suma = prvni + druha + treti + ctvrta + pata + sesta + _
       sedma + osma + devata
```

Na jednom řádku může být i několik příkazů oddělených dvojtečkou + :.
Příklad:

```
a = 0 :  b = 0 : c = 0 : d = 0
```

Proměnné a, b, c, d jsou vynulovány.

Jména

Jméno proměnné, konstanty, funkce nebo procedury může mít délku do 255 znaků.
První znak jména musí být malé nebo velké písmeno. OOo Basic nerozlišuje malá a velká písmena, proto jména PROMENNA, Promenna a promenna odkazují na stejnou proměnnou.
Jméno může obsahovat číslice a znak podtržítka (_), ale ne jako první znak.

Datové typy

datový typ	velikost [byte]	počáteční hodnota	poznámka
Boolean	1	False	True nebo False
Integer	2	0	celé číslo od -32768 do 32767
Long	4	0	celé číslo od -2147483648 do 2147483647
Date	0	00:00:00	datum a čas
Currency	8	0.0000	měna na 4 desetinná místa
Single	4	0.0	desetinné číslo v intervalu +/-3.402823 x 10E38
Double	8	0.0	desetinné číslo v intervalu +/-1.79769313486232 x 10E308
String	-	""	text do 65 535 znaků
Object	-	Null	objekt
Variant	-	Empty	jakýkoliv datový typ

Datový typ Date je v OOo Basicu reprezentován hodnotou Double odpovídající počtu dní od 30.12.1899 00:00:00 (toto datum je 0.0). Celá část je počet dnů od uvedeného data, desetinná část je zlomek dne a odpovídá hodinám, minutám a sekundám.

Příklad:

```
Dim datum As Date
datum = Now()    REM funkce Now() vrací současný okamžik typu Date
datum = datum + 1 REM Teď je v datum zitra, ve stejnou hodinu
datum = datum + 1.0/24 REM Teď je v datum zítra o hodinu později
```

Číslo Date může být i záporné a pak odpovídá datu před 30.12.1899 00:00:00.

Deklarace proměnných

Proměnná je vyhrazené pojmenované místo v paměti obsahující nějakou hodnotu. OOo Basic nevyžaduje deklaraci proměnných před jejich použitím. Proměnná může být deklarována i jejím použitím. Využívání této možnosti se důrazně nedoporučuje. Dopustíme-li se totiž překlepu ve jméně proměnné, interpret ji bere jako jinou proměnnou a to vede k těžko odhalitelným chybám.
Příklad:

```
a1 = 5       REM Pětku jsme přiřadili do a_jedna.
a2 = 7
b = a1 + a2  REM Ale teď jsme místo a_jedna použili AL.
             REM Najít takovou chybu je k zešílení.
```

Vložíme-li volbu Option Explicit na začátek každého modulu, vyvolá použití nedeklarované proměnné chybu.
Proměnné deklarujeme klíčovým slovem DIM.
Příklady:

```
Dim pocet As Integer  REM proměnná počet je typu integer
Dim pocet             REM není-li specifikován typ, je proměnná
                      REM typu variant
Dim a, b, c As Single REM tři proměnné typu single
Dim i As Integer, x As Single  REM dvě proměnné různých typů
                               REM deklarované na jednom řádku
```

Přiřazení hodnoty proměnné

Hodnota se do proměnné přiřazuje znaménkem „rovná se“

Příklady:

```
pi = 3.12415926
pocet = 6
Let pocet = 6  REM totéž jako předchozí řádek, nepoužívá se
```

Jak uvidíme později, stejné znaménko „=“ se používá i pro operátor „rovno“. Zatímco v Pascalu se přiřazuje a:=5 a porovnává a=5, v C/C++ se přiřazuje a=5 a porovnává a==5, v OOo Basicu se porovnává i přiřazuje a = 5.

Vlastní datové typy

OOo Basic umožňuje pracovat i se složitějšími datovými typy definovanými uživatelem. Předpokládejme, že budeme pracovat s databází osob, kde každá má definováno jméno, příjmení a mzdu. v následujícím příkladu vytvoříme datový typ, který uschová všechny tyto veličiny v jedné struktuře.

Příklad:

```
Type TypOsoba
  jmeno As String
  prijmeni As String
  mzda as Single
End Type

Sub VytvorOsobu
  Dim Osoba As TypOsoba
  Osoba.jmeno = "Jan"
  Osoba.prijmeni = "Novák"
  Osoba.mzda = 23712
  print Osoba.jmeno
End Sub
```

Pole

Pole (array) je datová struktura, která sdružuje daný vždy konečný počet prvků (čísel, textových řetězců, …) stejného datového typu. K jednotlivým prvkům pole se přistupuje pomocí jejich indexu (celého čísla, označujícího pořadí prvku). Na rozdíl od proměnných musí být pole před použitím vždy deklarováno a to i v případě, že není použito „Option Explicit“. Index prvku pole se v OOo Basicu uvádí v kulatých závorkách (na rozdíl od modernějších jazyků jako je Pascal, Java, C, kde se pro pole používají hranaté závorky). Máme-li tedy proměnnou `a` a pole `b`, přiřadíme prvek pole `b` do proměnné `a` příkazem `a = b(7)` a není možné bez znalosti jiných částí kódu poznat, jestli jsme do `a` přiřadili prvek pole `b` a nebo hodnotu funkce `b` s argumentem 7.

Příklady deklarace polí:

```
Dim a(5) As Integer                 REM 6 prvků od 0 do 5 včetně
Dim b(-5 to 5) As String            REM 11 prvků od -5 do 5 včetně
Dim c(5 to 10, 20 to 25) As Long    REM dvojrozměrné pole 6x6
Dim d()                             REM nějaké pole
d = Array(5,6,7,8,9)                REM přiřadil do pole 5 prvků
```

Funkce Join a Split používané pro koncerzi mezi textovým řetězcem a polem hodnot jsou popsány v kapitole o řetězcích.

`LBound()` a `Ubound()` - rozsah indexů pole

Fukce `LBound(arrayname[,Dimension])` a `UBound(arrayname[,Dimension])` vrací rozsah indexů pole, `LBound()` nejnižší index a `UBound()` nejvyšší index. Například vytvoříme-li pole `Dim pole(-6 to 5) as Integer` vrátí `LBound(pole)` číslo −6 a `UBound(pole)` číslo 5. Obě funkce mají volitelný druhý parametr upřesňující dimenzi, ke které se funkce vztahuje. Například pro pole `Dim pole2d(-3 to 4, 12 do 15) as Integer`, `UBound(pole2d,1)` vrací horní mez 1. indexu. a `LBound(pole2d,2)` vrací dolní mez 2. indexu.

Kopírování pole

Při kopírování proměnných s kopíruje jejich hodnota. Máme-li např. proměnné `a` a `b`, tak po provedení kódu `a = 5 : b = a : a = 3`, bude v proměnné `a` hodnota tři a v proměnné `b` hodnota pět. Při kopírování pole se příkazem `b() = a()` zkopíruje do `b()` odkaz, tedy adresu v paměti, kde je uložené pole `a()` a obě pole budou ukazovat na stejné místo a tedy i totožné proměnné. Lépe to pochopíne na následujícím příkladu:

```
Sub Pole
  Dim a()
  Dim b()
  a = Array(11,12,13,14)
  b = Array(21,22,23,24)
  b() = a()
  print "1. b(3): " + b(3)  REM Vypíše 14 - hodnotu a(3) i b(3)
  a(3) = 33                 REM Změníme pole a(), hodnotu a(3)
  print "2. b(3): " + b(3)  REM Vypíše b(3):33; a(3) a b(3) je stejná proměnná
  b(3) = 77                 REM Změníme pole b()
  print "3. a(3): " + a(3)  REM a změnila se i odpovídající hodnota pole a()
                            REM Pole a(),b() jsou po přiřazení b()=a() totožná
End Sub
```

Řetězce

Text je v řetězci uložen jako posloupnost šestnáctibitových Unicode znaků. Retězce je možno spojovat buď pomocí operátoru + nebo operátoru &. Chování těchto dvou operátorů se poněkud liší. Operátor + provede součet dvou operandů podle toho jakého jsou typu. Je-li první operand číslo, předpokládá operátor +, že se jedná o součet čísel a převede případné řetězce na čísla.

```
Print 123 + "4"      REM 127
```

První operand je číslo 123, proto se druhý operand, řetězec "4", převede na číslo 4 a přičte k číslu 123.

Je-li první operand řetězec, předpokládá operátor +, že se jedná o spojení řetězců a převede případná čísla na řetězce.

```
Print "4" + 123      REM 4123
```

První operand je řetězec "4", proto se druhý operand, číslo 123 převede na řetězec "123" a připojí za řetězec "4".

Operátor & vždy spojuje řetězce, proto

```
Print 123 & "4"      REM 1234
Print "4" & 123      REM 4123
```

Ale pozor, vložíme-li operátor & mezi dvě čísla, vrátí prázdný řetězec.

```
Print 123 & 4        REM
```

OOo Basic také obsahuje velké množství funkcí pro práci s řetězci. V následující tabulce uvedeme výběr nejdůležitějších z nich.

funkce	popis
`InStr([i,] seno, jehla)`	Vrací pozici řetězce jehla v řetězci seno. Nenalezne-li, vrací 0. Prohledává řetězec od i-tého znaku.
`Len(retezec)`	Vrací délku řetězce.
`Mid(retezec,pocatek,delka)`	Vrací část řetězce retezec o délce delka od pozice pocatek.
`Mid(retezec,pocatek,delka,nahrad)`	Nahradí část řetězce retezec o délce delka od pozice pocatek retezcem nahrad. Náhrada nesmí zvěšit délku řetězce.
`Join(pole(),oddelovac)`	Vrací prvky pole spojené do jednoho řetězce oddělené znakem oddělovač. Není-li oddělovač uveden, oddělí prvky jednou mezerou.
`Replace(seno, jehla, nahrad)`	V řetězci seno nahradí všechny řetězce jehla řetězcem nahrad.
`Split(retezec, oddelovac)`	Převede retezec na pole položek, oddělovačem je řetězec oddelovac.
`Format(objekt, vzor)`	Převede objekt (zpravidla číslo) na řetězec, jehož formát je definovaný řetězcem vzor.

`Funkce Format()`

`Format(objekt, vzor)` převede objekt (zpravidla číslo) na řetězec, jehož formát je definovaný řetězcem vzor. Předveďme funkci `Format()` na několika příkladech.

Formátování čísel – je-li v řetězci vzor znak 0, nahradí se odpovídající číslicí, není-li na daném místě číslice, doplní se nula.

```
Format(1234, "00.00")                    '1234.00
Format(12.345, "###0.00")                '12.35
Format(12.345, "0000.00")                '0012.35
Format(123.45678, "##E-####")            '12E1
MsgBox Format(.0012345678, "0.0E-####")  '1.2E-003
MsgBox Format(123.45678, "#.e-###")      '1.e002
MsgBox Format(.0012345678, "#.e-###")    '1.e-003
```

Formátování řetězců – je-li vzor znak < převede řetězec na malá písmena, je-li vzor znak > převede řetězec na velká písmena.

```
Format("Petr", "<")                          'petr
Format("Petr", ">")                      'PETR
```

Operátory

Seznam operátorů podporovaných v OOo Basicu

priorita	operátor	typ	popis
1	`NOT`	unární	Logické NE
1	-	unární	Unární minus
1	+	unární	Unární plus

2	^	binární	Umocnění. Např. 2^3 odpovídá matematickému zápisu 2^3.
3	*	binární	Numerické násobení.
3	/	binární	Numerické dělení.
4	MOD	binární	Zbytek po celočíselném dělení. Např. 13 MOD 5 vrátí 3.
5	\	binární	Celočíselné dělení. Např. 13 \ 5 vrátí 2.
6	-	binární	Numerické odčítání
6	+	binární	Numerické sčítání a spojování řetězců.
7	&	binární	Spojování řetězců.
8	IS	binární	Odkazují oba operandy na stejný objekt?
8	=	binární	Rovno.
8	<	binární	Menší než.
8	>	binární	Větší než.
8	<=	binární	Menší nebo rovno.
8	>=	binární	Větší nebo rovno.
8	<>	binární	Nerovno.
9	AND	binární	Logické A.
9	OR	binární	Logické NEBO.
9	XOR	binární	Exclusiv OR.
9	EQV	binární	Ekvivalence.
9	IMP	binární	Implikace.

Řízení běhu programu

If Then Else

Podmínka `If Then Else` provádí blok příkazů v závislosti na splnění podmínky. Syntaxe:

```
If podmínka Then
   blok příkazů 1
Else
   blok přílazů 2
End If
```

V případě, že je podmínka pravda, provede se blok příkazů 1, když podmínka není pravda, provede se blok příkazů 2.

Např.:

```
Dim x,y, vetsi as Integer
x = 7
y = 5
If x < y Then
   vetsi = y
Else
   vetsi = x
End If
```

V proměnné vetsi je větší z čísel x a y.

IIf

IIf je takzvané Immediate If (Přímé If).

Syntaxe:

```
object = IIf (Podmínka, VýrazKdyžPravda, VýrazKdyžNepravda)
```

Vyhodnotí se podmínka; je li podmínka True vrátí hodnotu výrazu VýrazKdyžPravda, není-li podmínka splněna vrátí výraz VýrazKdyžNepravda.

Příklad:

```
c = IIf(a<b, a, b)  REM Vrátí menší z hodnot a, b.
```

Do ... Loop

Cyklus Do ... Loop může existovat ve třech modifikacích:

Do While ... Loop – blok příkazů se provádí tak dlouho dokud je podmínka True. Není-li podmínka na začátku splněna, neprovede se ani jednou.

```
Do While podminka
   blok
Loop
```

Do Until ... Loop – blok příkazů se provádí tak dlouho dokud je podmínka False. Je-li podmínka na začátku splněna, neprovede se ani jednou.

```
Do Until podminka
   blok
Loop
```

Do ... Loop While – blok příkazů se provádí tak dlouho dokud je podmínka True. Blok říkazů se provede nejméně jednou (podmínka se vyhodnocuje až po prvním provedení bloku příkazů).

```
Do
  blok
Loop While podminka
```

Do ... Loop Until – blok příkazů se provádí tak dlouho dokud je podmínka False. Blok říkazů se provede nejméně jednou (podmínka se vyhodnocuje až po prvním provedení bloku příkazů).

```
Do
  blok
Loop Until podminka
```

For....Next

Cyklus `For....Next` slouží k opakování bloku programu.

Příklad:

```
soucet = 0
For i = 3 To 17 Step 2
   soucet = soucet + i
Next i
```

sečte všechna lichá čísla od 3 do 17. Není-li uveden `Step`, příčítá se 1.

Cyklus `For ... Next` nemá v OOo Basicu variantu jako je `For Each ... Next` ve Visual Basicu v Excelu. Toto omezení tze snadno obejít pomocí funkcí `LBound()` a `Ubound()`.

Příklad:

```
Sub PrikladForEach
  Dim a()
  Dim i
```

```
  Dim soucet As Integer

  soucet = 0
  a = Array(27, 51, 16, 33)    REM Vytvořeno pole obsahující čísla
  For i = LBound(a) To UBound(a)
    soucet = soucet + a(i)
  Next i
  print soucet
End Sub
```

Procedury a funkce

Funkce je část programu, kterou je možné opakovaně volat z různých míst kódu a vrací hodnotu. Funkce, která nevrací hodnotu se nazývá procedura. Procedura je deklarována klíčovým slovem Sub a ukončena End Sub. Makro tak, jak jsme jej dosud používali, je vlastně procedura.
Příklad:

```
Sub Tisk
  print "V procedure Tisk."
End Sub

Sub Main
  Call Tisk    REM Proceduru lze volat příkazem Call,
  Tisk         REM ale i pouhým jménem procedury.
End Sub
```

Spustíme-li makro (proceduru) Main, spustí dvakrát proceduru Tisk. Spouštěná procedura může být deklarována před místem, kde je volána (jako je to např. v C/C++) nebo i za tímto místem.

Funkce je deklarována klíčovým slovem Function a ukončena End Function. Funkce vrací hodnotu, která je v těle funkce přiřazena jejímu jménu.
Příklad:

```
Function Pi as double    REM Vytvoříme funkci Pi, která bude
                         REM vracet Ludolfovo číslo jako double
   Pi = 3.1415926535897  REM Hodnota, kterou má funkce vrátit se
                         REM přiřadí jménu funkce
End Function

Sub Main
  print Pi
End Sub
```

Argumenty

Proceduře nebo funkci můžeme při volání předat argumenty. Seznam argumentů se uvádí při deklaraci za jménem procedury zpravidla v závorce. Argumenty musí být deklarovány stejným způsobem jako se deklarují proměnné. Vytvořme proceduru, které předáme tři čísla typu double a ona vypíše jejich součet:

```
Sub Soucet(prvni as double, druhy as double, treti as double)
  print prvni + druhy + treti
End Sub
```

Makro lze spustit buď s argumenty v závorce nebo bez závorky:

```
Soucet(2,3,4)  REM Vypíše 9
Soucet 3,4,5   REM Vypíše 12
```

Argument procedury nebo funkce lze předávat **odkazem** (předá se adresa místa v paměti, kde je proměnná uložena a při změně proměnné uvnitř volané funkce se změna projeví i na hodnotě proměnné volající procedury) nebo **hodnotou** (předá se jen hodnota, kterou si funkce uloží do jiného místa v paměti a její změna uvnitř volané funkce se ve volající proceduře neprojeví). Argument předáme odkazem použijeme-li v hlavičce funkce klíčové slovo ByRef, naopak

použijeme-li v hlavičce funkce klíčové slovo ByVal, předáváme argument hodnotou. Defaultně, není-li specifikován způsob předání, předávají se argumenty odkazem.

```
function Soucet(s1 as double, ByRef s2 as double, ByVal s3 as double) as double
  soucet = s1 + s2 + s3
  s1=1
  s2=2
  s3=3
End function

Sub Spustit
  Dim scitanec1 as double, scitanec2 as double, scitanec3 as double

  scitanec1 = 11
  scitanec2 = 12
  scitanec3 = 13
  print soucet(scitanec1,scitanec2,scitanec3)        REM Vypíše 36
  print ""+scitanec1+", "+scitanec2+", "+scitanec3   REM Vypíše 1, 2, 13
End Sub
```

Podle očekávání se první argument předal odkazem, protože způsob předání nebyl specifikován, druhý argument také odkazem (bylo uvedeno klíčové slovo ByRef) a třetí hodnotou (bylo uvedeno klíčové slovo ByRef). Defaultní chování se liší od konvence používané v Pascalu nebo C/C++, kde se parametry, u nichž není způsob předání specifikován, předávají hodnotou.
Uživatelem definované funkce lze používat standardním způsobem i v sešitu Calcu. V tom případě se argumenty předávají hodnotou bez ohledu na to, jak je to bastaveno v hlavičce funkce. To znamená, že zadáme-li do nějaké buňky sešitu Calcu funkci např: `=SOUCET(A1;B1;C1)`, tak přesto, že první dva argumenty jsou předávány odkazem, hodnotu buněk A1, B1 a C1 se nezmění (ono je to logické, kdyby se změnily, musela by se buňka znovu přepočítat, tím by se znovu změnily vstupní hodnoty atd.)

Přístup k sešitu Calcu

Má-li makro dělat něco rozumného, musí komunikovat se sešitem, to znamená zejména načítat vstupní data z buněk sešitu a ukládat do buněk výsledky výpočtu. Přístup k buňkám je řešen jinak, o něco složitěji, než je tomu u MS Excelu.

Přístup ke konkrétní buňce

Chceme-li například vložit v MS Excelu v aktuálním sešitu na aktuální stránce do buňky na 3. řádku a v 5. sloupci číslo 7, poslouží nám příkaz:

```
Cells(3,5).Value=7
```

V OOo Calc:

```
thisComponent.Sheets(0).getCellByPosition(4,2).Value=7
```

při tom thisComponent odkazuje na aktuální sešit, `Sheets(0)` je první list tohoto sešitu a `getCellByPosition(4,2)` je odkaz na buňku v 5. sloupci a 3. řádku. Všimněme si, že číslo sloupce a řádku se zadává v opačném pořadí, než v MS Excelu. Navíc jsou v OOo Calc řádky a sloupce číslovány od nuly. Chceme-li pracovat s aktivním listem, lze místo `Sheets(0)` použít `ActiveSheet`. v makru se lze vyhnout zdlouhavému vypisování odkazů na stále stejný sešit a list s použitím proměnných:

```
Dim List
Dim Bunka
List= thisComponent.Sheets(0)
Bunka = List.getCellByPosition(4,2)
Bunka.Value = 7
```

Na list lze odkazovat indexem, jak jsme předvedli v minulém případě nebo jeho jménem. Místo `Sheets(0)` bychom mohli napsat i `Sheets("List1")`, pokud se první list aktuálního sešitu jmenuje List1. Podobně na konkrétní buňku lze místo sloupce a řádku odkázat jejím jménem
`getCellRangeByName("E3").`
Kromě numerické hodnoty lze do buňky zapsat i vzorec:
```
Bunka.Formula = "=SUM(A1:C4)"
```
nebo textový řetězec
```
Bunka.String = "můj řetězec"
```

Přístup k oblasti

Práce s vybranou oblastí

Na vybranou oblast sešitu ukazuje ThisComponent.CurrentSelection. Můžeme si ji opět pro zjednodušení práce přiřadit do proměnné
```
Vyber = ThisComponent.CurrentSelection
Vybraná oblast se skládá z řádků
Radky = Vyber.Rows
```
a sloupců
```
Sloupce = Vyber.Columns
```
Počet řádků a sloupců oblasti obsahuje vlastnost `getCount.`
Počet řádků tedy získáme `Radky.getCount` a analogicky počet sloupců `Sloupce.getCount`.
Odkaz na konkrétní buňku vybrané oblasti pak je
```
Bunka = Vyber.getCellByPosition (sloupec,radek),
```
při tom řádky a sloupce ve vybrané oblasti s číslují od nuly.
Předveďme si nyní získané poznatky na makru, které do vybrané oblasti vloží do každé buňky nulu.
```
Sub Vynuluj
   Vyber = ThisComponent.CurrentSelection
   Sloupce = Vyber.Columns
   Radky=Vyber.Rows
   For sloupec= 0 To Sloupce.getCount-1
      For radek = 0 To Radky.getCount-1
         Bunka=Vyber.getCellByPosition (sloupec,radek)
         Bunka.Value = 0
      Next radek
   Next sloupec
End Sub
```

Výběr oblasti v makru

Nahrazení obsahu buněk

Hledání a nahrazování obsahu buněk není možné v celám sešitu, ale pouze po jednotlivých listech. K nahrazování slouží ReplaceDescriptor, ve kterém nejdřív nastavíme parametry hledání a pak teprve se hledání provede. Postup je zřejmý z následujícího příkladu:

```
Sub HledejNahrad
  Dim List         'Sheet in which to replace
  Dim Deskriptor   'Replace descriptor
  Dim Pocet        'počet náhrad

  List = ThisComponent.Sheets(1)
  Deskriptor = List.createReplaceDescriptor()  'Vytvořil ReplaceDescriprot
  Deskriptor.setSearchString("Hledat")
  Deskriptor.setReplaceString("Nahradit za")
```

```
  Deskriptor.SearchWords = False   'je-li True, nahrazuje jen celé buňky
  Pocet = List.replaceAll(Deckriptor)
  Print "Nahrazeno: " & Pocet
End Sub
```

Přiřazení oblasti listu do pole

Metoda getDataArray umožňuje velmi pohodlné a jednoduché zkopírování oblasti buněk do pole. Opačný proces, zkopírování pole do oblasti buněk, umožňuje příkaz setDataArray. Použití osvětlí následující příklad, který načte čísla z oblasti `A1:C12`, vydělí je dvěma a zkopíruje do oblasti `E1:G12`:

```
Sub GetAndSetData
  Dim Oblast          'Oblast buňek
  Dim List            'List sešitu
  Dim pole            'pole obsahující data
  Dim i,j As Integer  'index pro pohyb v poli

  List = ThisComponent.Sheets(0)               'Vybral první list sešitu
  Oblast = List.getCellRangeByName("A1:C12")   'Vybral oblast

  pole = Oblast.getDataArray()                 'Data jsou ve dvojrozměrném poli
  print pole(1)(2)
  For i = 0 To UBound(pole)                    'Index i jde přes řádky oblasti
    For j = 0 To UBound(pole(1))               'Index j jde přes sloupce
      pole(i)(j)=pole(i)(j)/2
    Next
  Next
  Oblast = List.getCellRangeByName("E1:G12")
  Oblast.setDataArray(pole)
End Sub
```

Všimněte si, že na pole se neodkazuje, jak je v OOo Basicu zvykem pole(i,j), ale pole(i)(j). Prvek pole(i) je pole obsahující sloupce i-tého řádku.

Formátování buněk a oblastí

Nejdříve přiřadíme do proměnných odkaz na aktivní list a konkrétníbuňku v něm:

```
List= thisComponent.Sheets(0)
Bunka = List.getCellByPosition(4,2)
```

Vybrali jsme první list a v něm buňku D2.
Nyní můžeme formátovat zvolené oblasti

Font

```
List.CharFontName = "Courier"
Bunka.CharFontName = "Arial"
```

Font celého listu byl nastaven na Courier, font buňky D2 na Arial

Velikost písma

```
Sloupec.CharHeight = 12
Sloupec.CharHeight = 12
```

Barva písma

```
cell.CharColor = RGB (255, 0, 0)
```

Barva buňky

```
Bunka.CellBackColor = RGB(255,0,0)
```

Výška řádku a šířka sloupce

```
Sheet.Rows(0).OptimalHeight = True 'přizpůsobí výšku řádku textu
Sheet.Rows(1).Height = 1000        'nastaví výšku řádku na 10 mm
```