KAPITOLA 10

Vytváření funkcí

Funkce je procedura VBA, která vykonává nějaký výpočet a vrací určitou hodnotu. Funkce napsané v editoru Visual Basic lze použít v kódu VBA nebo ve vzorcích na pracovním listu. V této kapitole si přiblížíme:

- Rozdíl mezi procedurami deklarovanými jako `Sub` a `Function`
- Postup při vytváření vlastních funkcí
- Základní fakta o procedurách `Function` a parametrech funkcí
- Jak vytvořit funkci, která napodobuje funkci `SUMA` Excelu
- Postup při ladění funkcí, práce s dialogem Vložit funkci, použití doplňků pro ukládání vlastních funkcí
- Volání funkcí rozhraní API systému Windows pro vykonání jinak nemožných činností

Jazyk VBA dovoluje vytvářet procedury `Sub` a procedury `Function`. Procedurami `Sub` jsme se zabývali v minulé kapitole, předmětem této kapitoly jsou právě procedury `Function`, neboli *funkce*.

Odkaz: Kapitola 11 nabízí celou řadu užitečných a praktických příkladů funkcí. Mnohé použité postupy můžete použít ve vlastních programech.

Procedury (Sub) versus funkce (Function)

Na proceduru deklarovanou klíčovým slovem `Sub` můžete pohlížet jako na příkaz, který je spuštěn buď uživatelem, nebo voláním z jiné procedury. Oproti tomu procedury deklarované jako `Function` vždy vracejí jednu hodnotu (nebo pole proměnných), stejně jako funkce pracovních listů a vestavěné funkce VBA. A stejně jako u vestavěných funkcí mohou i vaše vlastní funkce používat parametry.

Procedury `Function` jsou vcelku univerzální a mohou být používány dvěma způsoby:

- Jako součásti výrazů v procedurách VBA,
- ve vzorcích, které vytváříte na pracovních listech.

Ve skutečnosti můžete použít proceduru `Function` kdekoli tam, kde lze použít funkce pracovního listu Excelu nebo vestavěnou funkci VBA. Jediným mně známým omezením je nemožnost použít funkci VBA ve vzorci ověřování dat.

Proč vytvářet vlastní funkce?

Zcela nepochybně jste se již seznámili s funkcemi pracovních listů v Excelu; dokonce i začátečníci vědí, jak používat nejběžnější funkce pro pracovní listy. Mezi tyto funkce patří třeba `SUMA`, `PRŮMĚR` a `KDYŽ`. Podle mých výpočtů Excel obsahuje více než 340 předdefinovaných funkcí pracovních listů. A pokud se vám to zdá málo, můžete si pomocí jazyka VBA vytvářet vlastní funkce.

Novinka: Excel 2007 zahrnuje všechny funkce obsažené v doplňku Analytické nástroje.

Máte-li k dispozici takový arzenál funkcí Excelu a VBA, naskýtá se otázka, proč ještě vyvíjet funkce vlastní? Odpověď zní: pro zjednodušení práce. Při vhodném návrhu najdou vaše vlastní funkce velké uplatnění ve vzorcích a procedurách VBA.

Často se vám například budou hodit vlastní funkce, které významně zkrátí zápis vzorců. Kratší vzorce se lépe čtou a snáze se s nimi pracuje. Musím však poznamenat, že vlastní funkce VBA budou ve vzorcích obvykle mnohem pomalejší než vestavěné funkce Excelu. A uživatel musí pochopitelně také aktivovat makra, aby mohl takové funkce využívat.

Při vytváření aplikací si můžete všimnout, že některé procedury ve svém kódu nějaké výpočty opakují. V takovém případě zvažte vytvoření vlastní funkce, která tyto výpočty bude provádět. Potom stačí jednoduše tyto funkce z procedur volat. Vlastní funkce tedy mohou eliminovat duplicitní kód, a tak i snížit výskyt chyb.

Také vaši spolupracovníci mohou mít ze speciálních funkcí užitek. Někdo třeba bude ochoten za vaše speciální funkce zaplatit. Ušetří totiž práci.

I když se mnozí nováčci při pomyšlení na vytváření vlastních funkcí pracovních listů vyděsí, postup jejich psaní není složitý. Já jsem se při tom spíše bavil. Zejména se mi líbí, když se pak moje funkce objeví v dialogu Vložit funkci spolu s vestavěnými funkcemi Excelu. Připadám si, jako kdybych se podílel přímo na vytváření Excelu.

V této kapitole vám vysvětlím vše, co potřebujete vědět, abyste mohli začít psát vlastní funkce. Nabídnu také spoustu příkladů.

Úvodní příklad funkce

Dost řečí, jdeme rovnou na to. V tomto oddílu najdete příklad procedury `Function` ve VBA.

Vlastní funkce

Následující vlastní funkce je definovaná v modulu VBA. Jmenuje se `OdstranitSamohlasky` a využívá jediný argument. Tento argument také vrací, ovšem s odstraněnými samohláskami.

```
Function OdstranitSamohlasky(Txt) As String
   ' Odstraňuje všechny samohlásky z textového argumentu
   Dim i As Long
   OdstranitSamohlasky = ""
   For i = 1 To Len(Txt)
      If Not UCase(Mid(Txt, i, 1)) Like "[AEIOU]" Then
         OdstranitSamohlasky = OdstranitSamohlasky & Mid(Txt, i, 1)
      End If
   Next i
End Function
```

Jistě se nejedná o nejužitečnější funkci, jakou jsem kdy napsal, zachycuje však některé zásadní principy práce s funkcemi. Její fungování vysvětlím později v části „Rozbor vlastní funkce".

Pozor: Když vytváříte vlastní funkce, které budou používány ve vzorcích na pracovních listech, musí být umístěny v normálních modulech VBA. Pokud byste vlastní funkce umístili do modulu kódu pro listy nebo pro sešit (`ThisWorkbook`), nedaly by se ve vzorcích použít.

Použití funkce v pracovním sešitu

Když do buňky zadáte vzorec, který používá funkci `OdstranitSamohlasky`, Excel spustí kód funkce a tak získá požadovanou návratovou hodnotu. Příklad použití této funkce ve vzorci vypadá takto:

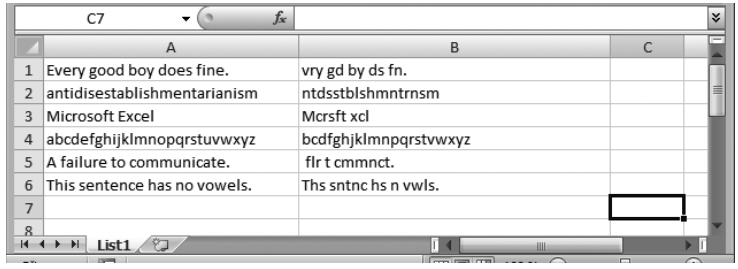
```
=OdstranitSamohlasky(A1)
```

Ukázku této funkce vidíte na obrázku 10.1. Vzorce jsou ve sloupci B, pracují s textem ve sloupci A. Jak je vidět, funkce vrací svůj parametr, ale s odstraněnými samohláskami.

Ve skutečnosti tato funkce pracuje úplně stejně jako každá jiná vestavěná funkce. Do vzorce ji můžete vložit pomocí příkazu Vzorce → Knihovna funkcí → Vložit funkci nebo klepnutím na průvodce vložením funkce vlevo na řádku vzorců. V dialogu Vložit funkci jsou vlastní funkce umístěny v kategorii Vlastní.

Ve vzorcích můžete vlastní funkce zanořovat do sebe a kombinovat je s dalšími prvky vzorců. Například další vzorec používá funkci `OdstranitSamohlasky` uvnitř funkce `VELKÁ` Excelu. Výsledkem je pak původní řetězec, ovšem bez samohlásek a hůlkovými znaky:

```
=VELKÁ(OdstranitSamohlasky(A1))
```



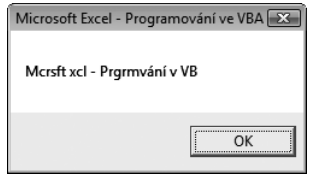
Obrázek 10.1: Použití vlastní funkce na pracovním listu

Použití funkce v proceduře VBA

Vlastní funkce lze používat nejen ve vzorcích na pracovních listech, ale také v jiných procedurách VBA. Následující procedura VBA, definována ve stejném modulu jako vlastní funkce `OdstranitSamohlasky`, nejprve zobrazí vstupní okno, ve kterém uživatel zadá nějaký text. Potom procedura pomocí vestavěné funkce VBA `MsgBox` zobrazí text zadaný uživatelem, který však již bude pomocí funkce `OdstranitSamohlasky` změněn (viz obrázek 10.2). Původní zadaný text se objeví jako titulek okna hlášení.

```
Sub BezSamohlasek()
   Dim VstupniRetezec As String
   VstupniRetezec = InputBox("Zadejte nějaký text:")
   MsgBox OdstranitSamohlasky(VstupniRetezec), , VstupniRetezec
End Sub
```

V příkladu na obrázku 10.2 byl do vstupního pole funkce `InputBox` zadán text *Microsoft Excel - Programování ve VBA*. Funkce `MsgBox` zobrazí tento text v podobě bez samohlásek.



Obrázek 10.2: Použití vlastní funkce v proceduře VBA

Rozbor vlastní funkce

Funkce mohou být tak složité, jak je třeba. Ve většině případů jsou spíše složitější a také mnohem užitečnější než naše ukázková funkce. Nicméně rozbor tohoto prvního příkladu vám může pomoci pochopit, co se vlastně ve funkci děje.

Zde je náš kód ještě jednou:

```
Function OdstranitSamohlasky(Txt) As String
    ' Odstraňuje všechny samohlásky z textového argumentu
    Dim i As Long
    OdstranitSamohlasky = ""
    For i = 1 To Len(Txt)
        If Not UCase(Mid(Txt, i, 1)) Like "[AEIOU]" Then
```

```
            OdstranitSamohlasky = OdstranitSamohlasky & Mid(Txt, i, 1)
        End If
    Next i
End Function
```

Všimněte si, že funkce začíná klíčovým slovem `Function`, nikoli `Sub`. Potom následuje název funkce (`OdstranitSamohlasky`). Tato vlastní funkce používá pouze jeden parametr (`Txt`), který je zapsán v závorkách. Závěrečná část `As String` definuje datový typ hodnoty, kterou funkce vrací. Pokud není datový typ vracené hodnoty určen, Excel použije standardně datový typ `Variant`.

Druhý řádek je komentářem (není samozřejmě povinný) popisujícím, co funkce vlastně dělá. Za ním následuje deklarace (`Dim`) proměnné ve funkci použité jako typu `Long`.

Všimněte si, že zde používám název funkce jako proměnnou. Když funkce skončí, bude vždy vracet aktuální hodnotu proměnné, jejíž název je stejný jako název funkce.

Dalších pět příkazů tvoří cyklus `For – Next`. Procedura prochází v cyklu všechny znaky ve vstupním řetězci a sestavuje výsledný řetězec. Příkaz uvnitř v cyklu používá vestavěnou funkci VBA `Mid`, která vrátí jeden znak ze vstupního řetězce.Tento znak převádí na velký (hůlkový). Pak jej porovnává se seznamem znaků s využitím operátoru `Like` Excelu. Jinými slovy, klauzule `If` je pravdivá, není-li aktuálním znakem A, E, I, O nebo U. V takovém případě se znak přidá na konec proměnné `OdstranitSamohlasky`. Když cyklus skončí, bude proměnná `OdstranitSamohlasky` obsahovat vstupní řetězec s odstraněnými samohláskami. Tento řetězec tvoří hodnotu, kterou potom funkce vrací.

Funkce skončí po dosažení příkazu `End Function`.

Pamatujte, že kód této funkce může mít řadu různých podob. Zde máme kupříkladu funkci, jež dosahuje stejného výsledku, ovšem pomocí odlišného kódu:

```
Function OdstranitSamohlasky2(txt) As String
    ' Odstraňuje všechny samohlásky z textového argumentu
    Dim i As Long
    Dim DocasRetezec As String
    DocasRetezec = ""
    For i = 1 To Len(txt)
        Select Case ucase(Mid(txt, i, 1))
            Case "A", "E", "I", "O", "U"
                ' Nedělat nic
            Case Else
                DocasRetezec = DocasRetezec & Mid(txt, i, 1)
        End Select
    Next i
    OdstranitSamohlasky2 = DocasRetezec
End Function
```

V této verzi jsem použil k ukládání řetězce bez samohlásek během jeho konstrukce novou řetězcovou proměnnou (`DocasRetezec`). Před koncem procedury jsem pak přiřadil obsah proměnné `DocasRetezec` názvu funkce. Tato verze také využívá konstrukci `Select Case` a nikoli konstrukci `If – Then`.

Na webu: Obě uvedené verze popisované funkce jsou na webu ke knize. Příslušný soubor nese název `OdstranitSamohlasky.xlsm`.

Co vlastní funkce pracovních listů nemohou dělat

Při psaní vlastních funkcí je důležité chápat klíčový rozdíl mezi funkcemi, které voláte z jiných procedur či funkcí VBA, a funkcemi používanými ve vzorcích na pracovních listech. Funkce používané ve vzorcích na listech musí být „pasivní": Kód uvnitř funkce nesmí například manipulovat s oblastmi. Vysvětlíme si to na příkladu.

Dejme tomu, že byste si chtěli napsat funkci pracovního listu, která změní formátování buňky. Půjde tedy o funkci, jež upraví barvu textu v buňce podle hodnoty buňky – to by se mohlo hodit. Dobrá, zkuste si to. Záhy zjistíte, že tuto funkci není možné napsat. Nezáleží na tom, jak to budete dělat; funkce list nezmění. Uvědomte si, že funkce prostě vrací hodnotu – nemůže provádět žádnou činnost s objekty.

Musím ovšem zmínit jednu významnou výjimku. Pomocí funkce VBA je možné změnit text komentáře buňky. Zde takovou funkci máme:

```
Function ZmenitKomentar(Bunka As Range, Koment As String)
   Bunka.Comment.Text Koment
End Function
```

Uveďme si ještě příklad použití této funkce ve vzorci. Následující vzorec nahradí komentář v buňce A1 novým textem. Funkce ovšem nebude fungovat, pokud buňka A1 žádný komentář neobsahuje.

```
=ZmenitKomentar(A1,"Hele, změnil jsem tento komentář!")
```

Funkční procedury

Vlastní procedura `Function` má mnoho společného s procedurou `Sub`. (Další informace o procedurách `Sub` najdete v kapitole 9.)

Deklarace funkce

Syntaxe deklarace funkce vypadá takto:

```
[Private | Public ] [Static] Function název [(parametry)] [As typ]
   [příkazy]
   [název = výraz]
   [Exit Function]
   [příkazy]
   [název = výraz]
End Function
```

Procedura `Function` zahrnuje následující elementy:

- `Public` (nepovinný) – stanoví, že funkce je dostupná všem procedurám ve všech modulech všech aktivních projektů VBA.
- `Private` (nepovinný) – stanoví, že funkce je přístupná pouze ostatním procedurám ve stejném modulu.
- `Static` (nepovinný) – určuje, že hodnoty proměnných deklarovaných ve funkci budou uchovány i po ukončení funkce.
- `Function` (povinný) – určuje začátek procedury, jež vrací hodnotu či jiná data.
- *`název`* (povinný) – představuje libovolný platný název funkce. Pro název platí stejná pravidla jako pro názvy proměnných.

- *parametry* (nepovinný) – představuje seznam jedné nebo více proměnných, uzavřených do závorek, které reprezentují parametry předávané funkci. Pro oddělování parametrů se používá čárka.
- *typ* (nepovinný) – datový typ hodnoty vracené funkcí.
- *příkazy* (nepovinný) – libovolný počet platných příkazů jazyka VBA.
- `Exit Function` (nepovinný) – příkaz, který způsobí okamžité ukončení funkce ještě před jejím řádným dokončením.
- `End Function` (povinný) – klíčové slovo, které určuje konec procedury `Function`.

Hlavní věc, kterou je třeba mít na paměti při psaní vlastních funkcí VBA, je to, že vrácená hodnota se musí přiřadit do názvu funkce alespoň jednou, zpravidla těsně před koncem funkce.

Vytvoření vlastní funkce začněte vložením nového modulu VBA. (Můžete použít i existující modul.) Zapište klíčové slovo `Function`, za ně název funkce a do závorek seznam parametrů (pokud nějaké jsou). Dále můžete deklarovat datový typ návratové hodnoty funkce, a to klíčovým slovem As a názvem datového typu (ale část s datovým typem je nepovinná). Dále zapište příkazy jazyka VBA, které provedou požadovanou činnost, a zkontrolujte, že do proměnné se stejným názvem, jako je název funkce, je v těle funkce alespoň jednou přiřazena návratová hodnota.

Funkci ukončete příkazem `End Function` (*pozn. překl.* – ten ovšem editor vloží sám, jakmile stisknete Enter po zadání názvu nové funkce).

```
='J21'(A1)
```

Názvy funkcí se řídí stejnými pravidly jako názvy proměnných. Pokud chcete používat funkci ve vzorcích na pracovním listu, není možné používat názvy, které vypadají jako názvy buněk (například funkce s názvem `J21` ve vzorci fungovat nebude). Vyhněte se také názvům, které jsou stejné jako názvy vestavěných funkcí Excelu. V případě konfliktu názvů dvou funkcí použije Excel vždy svou vestavěnou funkci.

Rozsah platnosti funkce

V kapitole 9 jsme probírali rozsah platnosti procedur (procedury veřejné nebo privátní). Totéž se týká i funkcí: rozsah platnosti funkce určuje, zda může být volána jinými procedurami v jiných modulech nebo v jiných pracovních listech.

V souvislosti s rozsahem platnosti funkcí si zapamatujte:

- Když rozsah platnosti funkce nedeklarujete, bude standardně veřejná (`Public`).
- Funkce deklarované jako privátní (`Private`) se neobjeví v dialogu Excelu Vložit funkci. Pokud tedy chcete vytvořit funkci, která má být používána pouze v procedurách VBA, měli byste ji deklarovat jako privátní, aby ji ostatní uživatelé nemohli (třeba i nechtěně) používat ve vzorcích.
- Pokud kód VBA potřebuje volat funkci definovanou v jiném sešitě, musíte nastavit odkaz do jiného sešitu pomocí příkazu editoru VB Tools → References.

Spouštění funkce

Na rozdíl od procedur `Sub`, které lze spouštět mnoha různými způsoby, je možné funkci spustit jen:

- voláním z jiné procedury,
- jejím použitím ve vzorci na pracovním listu.
- jejím voláním z okna Immediate editoru VB.

Volání funkce z jiné procedury

Vlastní funkci můžete volat z procedury stejným způsobem jako každou jinou vestavěnou funkci. Když si například definujete funkci s názvem `SUMAPole`, můžete potom napsat takovýto příkaz:

```
Celkem = SUMAPole(MojePole)
```

Tento příkaz spustí funkci `SUMAPole` a předá jí jako parametr proměnnou `MojePole`. Vrácená hodnota funkce se přiřadí do proměnné `Celkem`.

Použít můžete také metodu `Run` objektu `Application`. Příklad:

```
Celkem = Application.Run("SUMAPole", "MojePole")
```

První parametr metody `Run` je název funkce. Další parametry metody představují parametry předávané této volané funkci. Parametry pro metodu `Run` mohou mít formu literálových řetězců (jako v předchozím řádku kódu), čísel nebo proměnných.

Použití funkce ve vzorci pracovního listu

Použití vlastní funkce ve vzorci pracovního sešitu je stejné jako u vestavěných funkcí pracovního listu. Jediným rozdílem je to, že Excel musí konkrétní funkci najít – a to musíte zajistit vy. Je-li procedura `Function` deklarována ve stejném sešitu, nemusíte dělat nic speciálního. Je-li však v jiném sešitu, musíte Excelu sdělit, kde ji má hledat.

Můžete to udělat třemi způsoby:

- *Před název funkce uvést název souboru sešitu*. Pokud chcete například použít funkci s názvem `SpocitejNazvy`, která je definována v sešitu `MojeFunkce.xls`, můžete použít následující odkaz:

  ```
  =MojeFunkce.xlsm!SpocitejNazvy(A1:A1000)
  ```

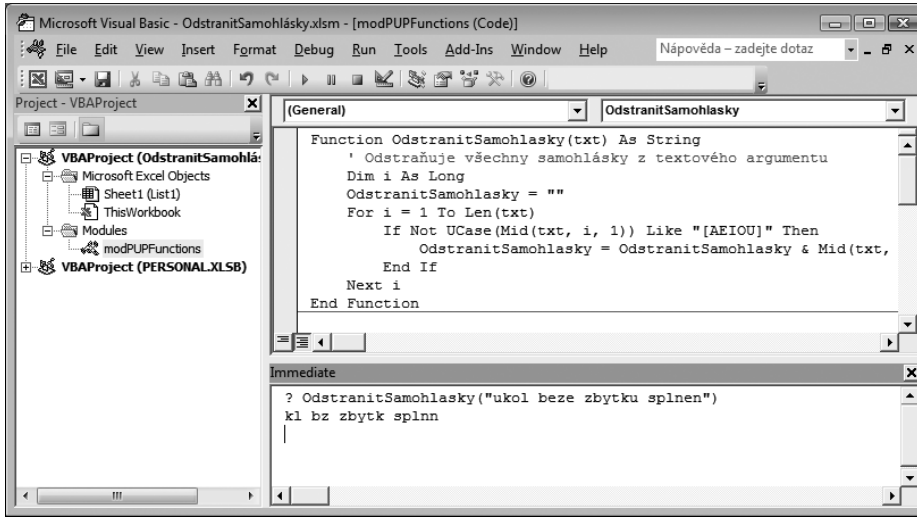
  Vložíte-li funkci prostřednictvím dialogu Vložit funkci, bude odkaz na sešit přidán automaticky.

- *Nastavit odkaz na sešit*. V editoru VB se odkaz (reference) nastaví volbou příkazu nabídky Tools → References. Je-li funkce umístěna v odkazovaném sešitu, nemusíte pak používat název sešitu. Dialog Vložit funkci však nadále přidává k funkci název sešitu, přestože to není nutné.
- *Vytvořit doplněk*. Když si ze sešitu, který obsahuje vaše procedury `Function`, vytvoříte doplněk, nebudete muset při volání libovolné v něm zahrnuté funkce používat ve vzorcích odkaz na soubor. Doplněk samozřejmě musí být nainstalován. O doplňcích bude řeč později v kapitole 21.

Časem si všimnete, že vaše procedury `Function` (funkce) se neobjeví v dialogu Makra (po klepnutí na příkaz nabídky Nástroje → Makro → Makra). Funkci také nelze spustit, když se kurzor nachází v jejím těle a vy klepnete na příkaz nabídky editoru VB Run → Run Sub/User Form (nebo stisknete klávesu F5). Místo toho se objeví dialog Makra, nabízející všechna makra, která je možné spustit. Z těchto důvodů je pro testování právě vytvářených funkcí nutná určitá změna oproti procedurám `Sub`. Vlastní funkci lze testovat z pomocné jednoduché procedury, která bude ve svém těle funkci volat. Je-li funkce navržena pro používání ve vzorcích na pracovním listu, bude třeba, abyste do listu vzorec s touto funkcí vložili a pomocí něj pak funkci testovali.

Z okna Immediate VBE

Poslední možností je volat funkční procedury z okna Immediate editoru VB. Tato metoda se většinou využívá pro účely testování. Příklad najdete na obrázku 10.3.



Obrázek 10.3: Volání funkční procedury z okna Immediate

Vynalézáme znovu kolo

Jen tak z legrace jsem si napsal vlastní verzi funkce Excelu `VELKÁ` (tato funkce převádí všechna písmena řetězce na velká) a nazval ji `HULKOVA`:

```
Function HULKOVA(VstupniRetezec As String) As String
    ' Převede všechna písmena svého parametru na velká
    Dim DelkaRetezce As Integer
    Dim i As Integer
    Dim HodnotaASCII As Integer
    Dim HodnotaChar As Integer

    DelkaRetezce = Len(VstupniRetezec)
    HULKOVA = VstupniRetezec
    For i = 1 To DelkaRetezce
       HodnotaASCII = Asc(Mid(VstupniRetezec, i, 1))
       HodnotaChar = 0
```

```
        If HodnotaASCII >= 97 And HodnotaASCII <= 122 Then
           HodnotaChar = -32
           Mid(HULKOVA, i, 1) = Chr(HodnotaASCII + HodnotaChar)
        End If
     Next i
  End Function
```

Tuto funkci najdete v sešitě nazvaném `VelkaPismena.xlsm` na webu ke knize. Samozřejmě se toto pracné řešení dá kdykoli nahradit vestavěnou funkcí VBA `UCase` – ale to by bylo moc snadné…

Byl jsem zvědavý, jak se bude tato moje vlastní funkce lišit od funkce vestavěné. Proto jsem si vytvořil pracovní list, který tuto funkci volal 20 000krát a předával jí náhodné názvy. Celé zpracování trvalo asi 40 sekund. Poté jsem nahradil svou funkci vestavěnou funkcí Excelu `VELKÁ` a test spustil znovu. Výpočet byl hotový téměř okamžitě.

Netvrdím, že moje funkce `HULKOVA` používá pro tento úkol optimální algoritmus, i tak je však zřejmé, že vlastní funkce napsané ve VBA nikdy nedosáhnou rychlosti vestavěných funkcí Excelu.

Parametry funkcí

Následující body se týkají parametrů funkcí. Dobře si je zapamatujte:

- Parametry mohou být proměnné (včetně polí proměnných), konstanty, literály nebo výrazy.
- Některé funkce nemají parametry.
- Některé funkce mají pevně daný počet povinných parametrů (může jich být 1 až 60).
- Některé funkce kombinují povinné a nepovinné parametry.

Poznámka: Pokud nějaký vzorec používá vlastní funkci a vrací chybu `#HODNOTA!`, znamená to, že ve vaší funkci je chyba. Potíž může být způsobena logickou chybou v kódu nebo předáním nesprávného parametru. Podívejte se kousek dál v této kapitole na odstavec „Ladění funkcí", kde najdete další informace.

Příklady funkcí

V tomto oddílu představuji celou řadu příkladů ukazujících, jak u funkcí efektivně používat parametry. Mimochodem, tento výklad se rovněž týká procedur `Sub`.

Funkce bez parametru

Stejně jako procedury `Sub` i procedury `Function` nemusí mít žádné parametry. Excel má také několik vestavěných funkcí, které parametry nepoužívají. Patří mezi ně funkce `NÁHČÍSLO`, `DNES` a `NYNÍ`. Podobné bezparametrové funkce si můžete napsat sami.

Na webu: Všechny příklady funkcí v tomto oddílu jsou na webu ke knize, konkrétně v souboru `BezArgumentu.xlsm`.

Následuje jednoduchý příklad funkce, která nepoužívá žádný parametr. Tato funkce vrací vlastnost `UserName` objektu `Application`. Toto jméno uživatele Excelu se normálně zobrazuje v dialogu Možnosti (na kartě Obecné) a je uloženo v registru Windows.

```
Function Uzivatel()
   ' Vrací jméno uživatele Excelu
   Uzivatel = Application.UserName
End Function
```

Když zadáte do buňky následující vzorec, buňka bude obsahovat jméno uživatele Excelu (za předpokladu, že je v registru Windows správně zadáno).

```
=Uzivatel()
```

Poznámka: Pokud použijete ve vzorci pracovního listu funkci bez parametrů, nesmíte v zápisu zapomenout prázdné závorky. Tento požadavek však není třeba dodržet, když voláte funkci z procedury VBA. Je ovšem dobrým zvykem závorky psát vždy, jelikož díky nim bude hned jasné, že se jedná o volání funkce.

Chcete-li tuto funkci použít v jiné proceduře, můžete její návratovou hodnotu přiřadit do proměnné, použít ji ve výrazu nebo ji využít jako parametr jiné funkce.

Následující příklad používá tuto funkci jako parametr funkce `MsgBox`. Spojovací operátor (`&`) spojí dohromady řetězcový literál s výsledkem funkce `Uzivatel`.

```
Sub ZobrazitUzivatele()
   MsgBox "Uživatel je " & Uzivatel()
End Sub
```

Příklad ukazuje, jak lze vytvořit obálkovou funkci, jež prostě vrací nějakou vlastnost nebo výsledek funkce VBA. Zde máme tři další obálkové funkce, které nepřebírají žádný argument.

```
Function AdresarExcelu() As String
   ' Vrátí adresář, v němž je instalován Excel
   AdresarExcelu = Application.Path
End Function

Function PocetListu()
   ' Vrátí počet listů v sešitu
   PocetListu = Application.Caller.Parent.Parent.Sheets.Count
End Function

Function NazevListu()
   ' Vrátí název listu
   NazevListu = Application.Caller.Parent.Name
End Function
```

Uveďme si další funkce bez parametru. Pro rychlé vyplnění oblasti buněk čísly jsem používal funkci Excelu `NÁHČÍSLO()`. Nelíbila se mi však skutečnost, že náhodná čísla se změnila, kdykoli se provedl přepočet pracovního listu. To jsem většinou napravoval převáděním vzorců na hodnoty.

Potom jsem si uvědomil, že bych si mohl napsat vlastní funkci vracející náhodná čísla, která by se však neměnila. Použil jsem vestavěnou funkci VBA `Rnd`, která vrací náhodné číslo v intervalu 0 až 1. Tato vlastní funkce vypadá následovně:

```
Function StatickeNahodne()
    ' Vrací náhodné číslo, které se
    ' nebude při přepočtu sešitu měnit
    StatickeNahodne = Rnd()
End Function
```

Chcete-li generovat posloupnost náhodných celých čísel v intervalu 0 až 1 000, můžete použít takovýto vzorec:

```
=CELÁ.ČÁST(StatickeNahodne() * 1000)
```

Hodnoty, které tento vzorec vytvoří, se nikdy nebudou měnit, když se list normálně přepočítá. Přepočet vzorců si však můžete vynutit stiskem Ctrl+Alt+F9.

Řízení přepočtu funkce

Když budete ve vzorci pracovního listu používat vlastní funkce, nabízí se otázka, kdy se budou přepočítávat.

Vlastní funkce se chovají jako vestavěné funkce pro pracovní listy Excelu. Normálně budou vlastní funkce přepočítávány pouze tehdy, je-li to nutné – neboli pouze tehdy, když se některý z parametrů funkce změní. Funkci však lze přinutit, aby se přepočítávala častěji. Přidání následujícího příkazu do procedury `Function` způsobí, že se funkce přepočte při každé změně libovolné buňky v pracovním listu.

```
    Application.Volatile True
```

Metoda `Volatile` objektu `Application` má pouze jeden parametr (s hodnotou `True` nebo `False`). Označíte-li funkci jako *volatilní* (*nestálou*), bude se funkce přepočítávat, kdykoli dojde k přepočtu kterékoli buňky pracovního listu.

Například vlastní funkce `StatickeNahodne` může být změněna tak, aby emulovala funkci Excelu `NÁH.ČÍSLO()`, a to následovně pomocí metody `Volatile`:

```
Function NeStatickeNahodne()
    ' Vrací náhodné číslo, které se
    ' změní při každém přepočtu listu
    Application.Volatile True
    NeStatickeNahodne = Rnd
End Function
```

Použijete-li pro metodu `Volatile` hodnotu `False`, bude se funkce přepočítávat pouze při změně jednoho či více jejích parametrů (pokud funkce nemá žádné parametry, tato metoda nemá žádný účinek).

Úplný přepočet sešitu, včetně nevolatilních vlastních funkcí, si vynutíte stiskem kombinace Ctrl+Alt+F9. Tato kombinace by vygenerovala nové náhodné číslo, vrácené funkcí `StaticRand`, kterou jsme viděli před chvílí.

Funkce s jedním parametrem

Tento oddíl popisuje funkci pro vedoucí prodeje, kteří potřebují spočítat celkové provize svých prodavačů. Výpočty v tomto příkladu vycházejí z následující tabulky:

Měsíční prodeje	Provize
0 – 9 999 korun	8,0 %
10 000 – 19 999 korun	10,5 %
20 000 – 39 999 korun	12,0 %
40 000 a více korun	14,0 %

Všimněte si, že míra provize není lineární a závisí na celkovém měsíčním prodeji. Zaměstnanci, kteří prodají více, dosahují také větší provize.

Existuje celá řada způsobů, jak vypočítat provize pro různé objemy prodeje zadané do pracovního listu. Pokud si celý postup řádně nepromyslíte, dospějete možná k takto dlouhému vzorci:

```
= KDYŽ(A(A1>=0, A1<=9999.99), A1 * 0.08,
  KDYŽ(A(A1>=10000, A1<=19999.99), A1 * 0.105,
  KDYŽ(A(A1>=20000, A1<=39999.99), A1 * 0.12,
  KDYŽ(A(A1>=40000, A1 * 0.14))))
```

To je špatný přístup, a to z několika důvodů. Za prvé, vzorec je příliš složitý a málokdo ho pochopí. Za druhé, hranice pásem prodeje jsou ve vzorci zapsány „natvrdo“, což znesnadňuje pozdější úpravu vzorců.

Lepším postupem (bez VBA) pro výpočet provizí je použít vyhledávací tabulku a funkci, která pro určitý objem prodeje vyhledá procento provize. Například následující vzorec používá funkci `SVYHLEDAT` pro načtení procenta provize z oblasti buněk Tabulka. Pak násobí získané procento hodnotou v buňce A1.

```
=SVYHLEDAT(A1, Tabulka, 2) * A1
```

Jiný postup (který nevyžaduje vyhledávací tabulku) představuje vytvoření vlastní funkce vypadající třeba takto:

```
Function Provize(Prodej)
   Const Uroven1 = 0.08
   Const Uroven2 = 0.105
   Const Uroven3 = 0.12
   Const Uroven4 = 0.14
   ' Vypočítá provize z prodejů
   Select Case Prodej
      Case 0 To 9999.99: Provize = Prodej * Uroven1
      Case 10000 To 19999.99: Provize = Prodej * Uroven2
      Case 20000 To 39999.99: Provize = Prodej * Uroven3
      Case Is >= 40000: Provize = Prodej * Uroven4
   End Select
End Function
```

Když tuto funkci zapíšete do modulu VBA, můžete ji používat ve vzorcích na pracovním listu nebo ji volat z jiné procedury VBA.

Po zapsání následujícího vzorce do buňky dostanete výsledek 3 000 (objem prodeje 25 000 korun spadá do kategorie 12% provize):

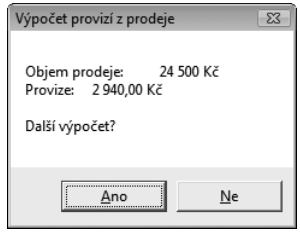
```
=Provize(25000)
```

I když nepotřebujete vlastní funkce na pracovních listech, mohou vám funkční procedury usnadnit alespoň psaní kódu ve VBA. Jestliže například procedura VBA vypočítává provize z prodejů, můžete použít znovu stejnou funkci a volat ji z procedury VBA. Následující krátká ukázka se zeptá uživatele na objem prodeje a potom použije funkci `Provize` pro výpočet částky provize:

```
Sub VypocetProdeje()
   Prodej = InputBox("Zadejte objem prodeje:")
   MsgBox "Provize je " & Provize(Prodej)
End Sub
```

Procedura `VypocetProdeje` začne zobrazením vstupního pole pro zadání objemu prodeje. Potom zobrazí okno s vypočtenou částkou provize z prodeje, která odpovídá zadané částce.

Tato procedura `Sub` sice funguje, je však ještě poněkud „syrová". V následujícím výpisu je její zdokonalená verze, která zobrazí formátované hodnoty a bude se opakovaně v cyklu uživatele ptát, zda chce počítat další provizi, dokud uživatel neklepne na tlačítko Ne (viz obrázek 10.4).



Obrázek 10.4: Použití funkce k zobrazení výsledku výpočtu

```
Sub VypocetProdeje()
    Dim Prodej As String
    Dim Zprava As String, Odpoved As String

    ' Vyzve k zadání objemu prodejů
    Prodej = Val(InputBox("Zadejte prodeje:", _
                "Kalkulátor pro výpočet provizí z prodeje"))

    ' Sestaví zprávu
    Zprava = "Objem prodeje:" & vbTab & Format(Prodej, "#,##0 Kč")
    Zprava = Zprava & vbCrLf & "Provize:" & vbTab
    Zprava = Zprava & Format(Provize(Prodej), "# ##0.00 Kč")
    Zprava = Zprava & vbCrLf & vbCrLf & "Další výpočet?"

    ' Zobrazí výsledek a zeptá se, zda se bude počítat znovu
    Odpoved = MsgBox(Zprava, vbYesNo, "Výpočet provizí z prodeje")
    If Odpoved = vbYes Then VypocetProdeje
End Sub
```

Tato funkce používá dvě vestavěné konstanty VBA: `vbTab` představuje tabulátor (pro odsazení výstupu) a konstanta `vbCrLf` určuje návrat kurzoru a odřádkování (neboli přechod na nový řádek). Funkce VBA `Format` zobrazí hodnotu ve zvoleném formátu (v tomto případě se symbolem korun a dvěma desetinnými místy).

V obou těchto příkladech musí být funkce `Provize` umístěna v aktivním sešitu; jinak Excel zobrazí zprávu s hlášením, že funkce není definována.

Používejte argumenty, nikoli odkazy na buňky

Všechny argumenty používané ve vlastní funkci byste měli předávat jako argumenty. Podívejte se na následující funkci vracející hodnotu v buňce A1 znásobenou dvěma:

```
Function DvojnasobekBunky()
    DvojnasobekBunky = Range("A1") * 2
End Function
```

Tato funkce sice funguje, ovšem v určitých situacích může vrátit nesprávný výsledek. Výpočtové jádro Excelu nemůže správně zpracovávat oblasti buněk, jež nejsou předávány jako argumenty. V určitých případech nemusí před vrácením hodnoty funkcí dojít k přepočtu všech předchůdcovských buněk. Funkce `DvojnasobekBunky` by měla být zapsána následujícím způsobem, který předává A1 jako argument:

```
Function DvojnasobekBunky(bunka)
    DvojnasobekBunky = bunka * 2
End Function
```

Funkce se dvěma parametry

Představte si, že již zmíněný pomyslný vedoucí prodeje přechází na novou obchodní politiku, která bude mít za úkol snížit fluktuaci: celkové vyplácené provize se budou zvyšovat o jedno procento s každým rokem, který prodavač pro společnost odpracuje.

Funkci `Provize` (je definována v předcházející sekci) jsem upravil tak, aby přebírala dva parametry. Nový parametr představuje počet let. Tuto novou funkci si nazvěme `Provize2`:

```
Function Provize2(Prodej, Roky)
    ' Vypočítává provize z prodeje na základě
    ' počtu let ve službě
    Const Uroven1 = 0.08
    Const Uroven2 = 0.105
    Const Uroven3 = 0.12
    Const Uroven4 = 0.14
    Select Case Prodej
        Case 0 To 9999.99: Provize2 = Prodej * Uroven1
        Case 10000 To 19999.99: Provize2 = Prodej * Uroven2
        Case 20000 To 39999.99: Provize2 = Prodej * Uroven3
        Case Is >= 40000: Provize2 = Prodej * Uroven4
    End Select
    Provize2 = Provize2 + (Provize2 * Roky / 100)
End Function
```

Docela jednoduché, že? Prostě jsem do deklarace `Function` přidal druhý parametr (`Roky`) a rovněž jsem doplnil výpočet upravující výši provize podle hodnoty tohoto parametru.

Následující ukázka předvádí zápis vzorce s touto funkcí (předpokládá se, že objem prodejů je zapsán v buňce A1 a počet odpracovaných let pracovníka obchodu je v buňce B1):

```
=Provize2(A1, B1)
```

Na webu: Všechny procedury související s prodejem najdete na webu ke knize v souboru s názvem `FunkceProvize.xlsm`.

Funkce s polem parametrů

Procedury `Function` mohou jako parametry také přebírat jedno nebo více polí proměnných, tato pole zpracovat a vrátit jednu hodnotu. Pole může být tvořeno také oblastí buněk.

Následující funkce očekává jako parametr pole proměnných a vrací součet prvků tohoto pole:

```
Function SUMAPole(Seznam) As Double
   Dim Polozka As Variant
   SUMAPole = 0
   For Each Polozka In Seznam
      If WorksheetFunction.IsNumber(Polozka) Then _
         SUMAPole = SUMAPole + Polozka
   Next Polozka
End Function
```

Funkce `IsNumber` u každého prvku pole ověřuje, zda obsahuje číslo. Teprve poté ho přičte k celkovému součtu. Přidáním tohoto jednoduchého příkazu vyloučíte chybu vzniklou nesouladem datových typů, ke které dojde, pokusíte-li se provádět aritmetické operace s řetězcem.

Následující procedura předvádí volání výše uvedené funkce. Procedura vytvoří pole o sto prvcích a přiřadí každému prvku náhodné číslo. Potom funkce `MsgBox` zobrazí celkový součet hodnot prvků pole, a to pomocí volání funkce `SUMAPole`.

```
Sub VytvoritSeznam()
   Dim Cisla(1 To 100) As Double
   Dim i As Integer
   For i = 1 To 100
      Cisla( i ) = Rnd * 1000
   Next i
   MsgBox SUMAPole(Cisla)
End Sub
```

Všimněte si, že funkce `SUMAPole` nedeklaruje datový typ svého parametru (bude to tedy `Variant`). Jelikož není deklarován konkrétní číselný typ, lze tuto funkci použít také v takových vzorcích na pracovních listech, jejichž argumentem je objekt oblasti (`Range`). Následující vzorec vrací součet všech hodnot v oblasti A1:C10:

```
=SUMAPole(A1:C10)
```

Možná jste si všimli, že funkce `SUMAPole` ve vzorci pracovního listu pracuje téměř stejně jako funkce Excelu `SUMA`. Jediným rozdílem je skutečnost, že naše funkce `SUMAPole` neumí pracovat s více parametry. Upozorňuji, že tato funkce je určena jen pro ukázku, v praxi nemá ve vzorcích žádnou výhodu oproti funkci `SUMA` Excelu.

Na webu: Tento příklad najdete na webu ke knize v sešitu s názvem `ArgumentPole.xlsm`.

Funkce s nepovinnými parametry

Mnohé vestavěné funkce Excelu používají nepovinné parametry. Příkladem může být funkce ZLEVA, která vrací znaky z levé části řetězce. Její syntaxe je:

```
ZLEVA(text [, počet_znaků])
```

První parametr je povinný, druhý však uvádět nemusíte. Pokud je druhý parametr vynechán, Excel bude předpokládat, že má hodnotu 1. Následující dva vzorce tedy vrátí stejný výsledek:

```
=ZLEVA(A1, 1)
=ZLEVA(A1)
```

Také vlastní funkce, které napíšete v jazyce VBA, mohou mít nepovinné parametry. Parametr označíte za nepovinný tak, že před jeho název napíšete klíčové slovo Optional. V seznamu parametrů se nepovinné parametry musí objevit až za posledním povinným parametrem.

V dalším výpisu je ukázka jednoduché funkce, která vrací jméno uživatele. Jediný parametr funkce je nepovinný.

```
Function Uzivatel(Optional VelkaPismena As Variant)
   If IsMissing(VelkaPismena) Then _
      VelkaPismena = False
   Uzivatel = Application.UserName
   If VelkaPismena Then _
      Uzivatel = UCase(Uzivatel)
End Function
```

Jestliže má parametr hodnotu False nebo je vynechán, bude jméno uživatele vráceno tak, jak je zapsáno v dialogu Možnosti. Má-li však parametr hodnotu True, bude jméno převedeno na velká písmena (pomocí vestavěné funkce VBA UCase). První příkaz funkce používá funkci VBA IsMissing, která testuje, zda parametr byl, či nebyl předán. Jestliže parametr chybí, příkaz nastaví proměnnou VelkaPismena na False.

Všechny následující vzorce jsou platné (a první dva mají stejný smysl):

```
=Uzivatel()
=Uzivatel(False)
=Uzivatel(True)
```

Poznámka: Jestliže potřebujete zjistit, zda byl volitelný parametr funkci předán, či nikoli, musíte tento parametr deklarovat s datovým typem Variant. Jedině tak je možné pro kontrolu jeho přítomnosti použít funkci IsMissing, kterou ukázal poslední příklad.

Uvádím ještě další příklad vlastní funkce, která používá nepovinný parametr. Tato funkce náhodně vybere jednu buňku ze vstupní oblasti a vrátí obsah této buňky. Má-li druhý parametr hodnotu True, změní se vybraná hodnota pokaždé, když bude sešit přepočítán (funkce je označena jako *volatile* – nestálá). Pokud má druhý parametr hodnotu False (nebo je vynechán), funkce se nebude přepočítávat, dokud se alespoň jedna z buněk ve vstupní oblasti nezmění.

```
Function VyberBunku(Oblast As Variant, Optional Prepocet As Boolean = False)
   ' Vybere náhodně jednu buňku ze zadané oblasti
```

```
    ' Je-li Prepocet roven True, pak se funkce označí jako volatile
    Application.Volatile Prepocet

    ' Určí náhodně buňku
    VyberBunku = Oblast(Int((Oblast.Count) * Rnd + 1))
End Function
```

Všimněte si, že druhý parametr v hlavičce funkce obsahuje klíčové slovo `Optional`, spolu s implicitní hodnotou `False` (která bude použita, pokud parametr není zadán).

Všechny následující vzorce jsou platné a první dva mají stejný smysl:

```
=VyberBunku(A1:100)
=VyberBunku(A1:100, False)
=VyberBunku(A1:100, True)
```

Tato funkce se hodí například pro losování čísel loterie, vybírání jména vítěze ze seznamu jmen a podobně.

Na webu: Popisovanou funkci najdete na webu ke knize v souboru s názvem `Vyber.xlsm`.

Funkce, které vracejí pole

Jazyk VBA obsahuje užitečnou funkci s názvem `Array`. Funkce `Array` vrací proměnnou typu `Variant`, která obsahuje pole (tedy několik hodnot). Pokud znáte maticové vzorce v Excelu, pochopíte funkci VBA `Array` velmi rychle. Maticový vzorec zadáte do buňky stiskem kombinace Ctrl+Shift+Enter. Excel vloží kolem vzorce složené závorky symbolizující, že se jedná o maticový vzorec. Další podrobnosti o maticových vzorcích najdete ve třetí kapitole.

Poznámka: Je velmi důležité uvědomit si, že vrácené pole není stejné jako normální pole proměnných tvořené prvky s datovým typem `Variant`. Jinými slovy, proměnná pole typu `Variant` neodpovídá poli proměnných datového typu `Variant`.

Funkce `NazvyMesicu` (viz následující výpis kódu) je jednoduchou ukázkou použití funkce `Array` ve vlastní funkci:

```
Function NazvyMesicu()
    NazvyMesicu = Array("Leden", "Únor", "Březen", _
        "Duben", "Květen", "Červen", "Červenec", "Srpen", _
        "Září", "Říjen", "Listopad", "Prosinec")
End Function
```

Funkce vrací horizontální pole s názvy měsíců. Pomocí této funkce můžete vytvořit maticový vzorec vyplňující buňky názvy měsíců. Jak na to? Nejprve zkontrolujte, zda je kód této funkce ve standardním modulu VBA. Poté vyberte na listu více buněk v jednom řádku (začneme se 12 buňkami). Poté zapište následující vzorec (bez složených závorek) a stiskněte Ctrl+Shift+Enter:

```
{=NazvyMesicu()}
```

Co když budete chtít generovat svislý seznam názvů měsíců? Žádný problém, vyberte svislou oblast buněk a zapište následující vzorec (bez složených závorek), opět ukončený stiskem kombinace Ctrl+Shift+Enter:

```
{=TRANSPOZICE(NazvyMesicu())}
```

Tento vzorec používá funkci Excelu `TRANSPOZICE` pro převod vodorovného pole na svislé.

Další příklad je variantou funkce `NazvyMesicu`:

```
Function NazvyMesicu2(Optional MIndex)
   Dim VsechnyNazvy As Variant
   Dim HodnotaMesice As Integer
   VsechnyNazvy = Array("Leden", "Únor", "Březen", _
      "Duben", "Květen", "Červen", "Červenec", "Srpen", _
      "Září", "Říjen", "Listopad", "Prosinec")

   If IsMissing(MIndex) Then
      NazvyMesicu2 = VsechnyNazvy
   Else
      Select Case MIndex
         Case Is >= 1
            ' Určíme číslo měsíce (například 13=1)
            HodnotaMesice = ((MIndex - 1) Mod 12)
            NazvyMesicu2 = VsechnyNazvy(HodnotaMesice)
         Case Is <= 0 ' Svislé pole
            NazvyMesicu2 = Application.Transpose(VsechnyNazvy)
      End Select
   End If
End Function
```

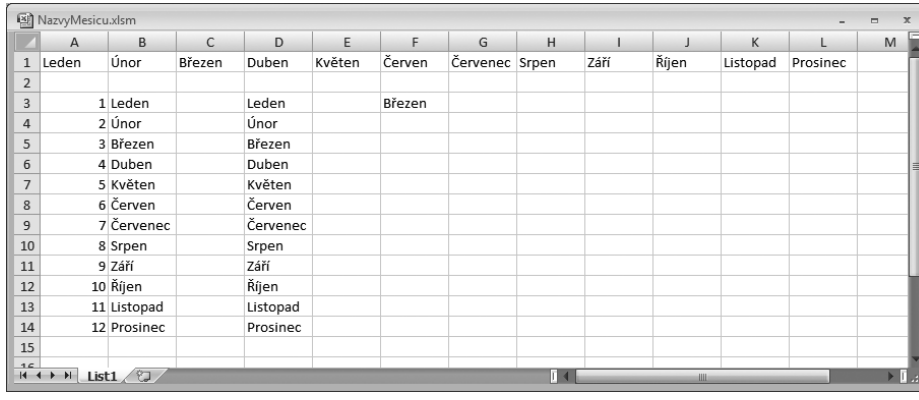
Všimněte si, že jsem použil funkci `IsMissing`, která testuje, zda nepovinný parametr byl, či nebyl zadán. V tomto případě není možné definovat výchozí hodnotu parametru už v hlavičce funkce, protože výchozí hodnota je definována přímo ve funkci. Funkci `IsMissing` můžete použít jen tehdy, je-li volitelný parametr typu `Variant`.

Tato rozšířená verze funkce má jeden nepovinný parametr, který se chová následovně:

- *Pokud tento parametr chybí*, funkce vrací vodorovné pole názvů měsíců.
- *Pokud je tento parametr menší nebo roven 0*, funkce vrací svislé pole názvů měsíců. Pro převod pole používá funkci `Transpose`.
- *Pokud je tento parametr zadán a je roven či větší než 1*, funkce vrací název měsíce, který odpovídá hodnotě parametru.

Poznámka: Tato funkce zjišťuje číslo měsíce pomocí operátoru `Mod`. Operátor `Mod` vrací zbytek po celočíselném dělení prvního operandu druhým operandem. Uvědomte si, že první prvek pole `VsechnyNazvy` má index 0, a tedy hodnoty indexů jsou v rozmezí 0 až 11. V příkazu s operátorem `Mod` je od parametru funkce odečtena jednička. Proto parametr s hodnotou `13` vrátí nulu (která odpovídá lednu) a parametr `24` vrátí `11` (neboli prosinec).

Tato funkce se dá použít mnoha způsoby, jak předvádí obrázek 10.5.



NazvyMesicu.xlsm

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Leden	Únor	Březen	Duben	Květen	Červen	Červenec	Srpen	Září	Říjen	Listopad	Prosinec	
2													
3	1	Leden		Leden		Březen							
4	2	Únor		Únor									
5	3	Březen		Březen									
6	4	Duben		Duben									
7	5	Květen		Květen									
8	6	Červen		Červen									
9	7	Červenec		Červenec									
10	8	Srpen		Srpen									
11	9	Září		Září									
12	10	Říjen		Říjen									
13	11	Listopad		Listopad									
14	12	Prosinec		Prosinec									
15													

List1

Obrázek 10.5: Různé způsoby předání pole nebo jediné hodnoty na pracovním listu

Oblast A1:L1 obsahuje vzorec zadaný jako pole. Nejprve vyberte oblast A1:L1, zadejte uvedený vzorec (bez složených závorek) a nakonec stiskněte Ctrl+Shift+Enter.

```
{=NazvyMesicu2()}
```

Oblast A3:A14 obsahuje celá čísla od 1 do 12. Buňka B13 obsahuje následující vzorec (není maticový), který je zkopírován do 11 buněk pod touto buňkou:

```
=NazvyMesicu(A3)
```

Oblast D3:D14 obsahuje následující maticový vzorec:

```
=NazvyMesicu2(-1)
```

Oblast F3 obsahuje následující (nikoli maticový) vzorec:

```
= NazvyMesicu2(3)
```

Připomeňme, že pro zadání maticového vzorce musíte stisknout kombinaci Ctrl+Shift+Enter.

Poznámka: Dolní hranice pole vytvořeného funkcí `Array` je vždy určena dolní hranicí definovanou příkazem `Option Base` v deklarační (horní) části modulu. Není-li tento příkaz použit, je dolní hranicí nula.

Na webu: Sešit ukazující funkce `NazvyMesicu` na webu ke knize a nese název `NazvyMesicu.xslm`.

Funkce vracející chybovou hodnotu

V některých případech bude třeba, aby vlastní funkce vracela chybovou hodnotu. Podívejme se na funkci `OdstranitSamohlasky`, kterou jsme si představili již dříve v této kapitole.

```
Function OdstranitSamohlasky(txt) As String
    ' Odstraňuje všechny samohlásky z textového argumentu
    Dim i As Long
    OdstranitSamohlasky = ""
    For i = 1 To Len(txt)
```

```
        If Not UCase(Mid(txt, i, 1)) Like "[AEIOU]" Then
            OdstranitSamohlasky = OdstranitSamohlasky & Mid(txt, i, 1)
        End If
    Next i
End Function
```

Když ji použijete ve vzorci v pracovním listu, tato funkce odstraní samohlásky svého parametru (jedné buňky). Jedná-li se o číselnou hodnotu, vrátí funkce tuto hodnotu jako řetězec. Předpokládejme, že chcete, aby v případě zadání čísla funkce vracela chybovou hodnotu (`#N/A`) a nikoli stejný údaj převedený na řetězec.

Nejdříve vás možná napadne přiřadit řetězec, který vypadá jako chybová hodnota Excelu. Například takto:

```
OdstranitSamohlasky = "#N/A"
```

Ačkoli řetězec *vypadá* jako chybová hodnota, ostatní vzorce, které se na tuto buňku budou odkazovat, ji jako chybovou hodnotu chápat nebudou. Aby funkce vracela *skutečnou* chybovou hodnotu, musíte použít funkci `CVErr` jazyka VBA, která převádí číslo chyby na skutečnou chybu.

Naštěstí má jazyk VBA vestavěné konstanty pro chyby, které můžete ze svých funkcí vracet. Tyto chyby představují hodnoty chyb vzorců Excelu, ne hodnoty chyb běhu programu VBA. Jsou to následující konstanty:

- `xlErrDiv0` (pro chybu #DIV/0!),
- `xlErrNA` (pro chybu #N/A),
- `xlErrName` (pro chybu #NÁZEV!),
- `xlErrNull` (pro chybu #NULL!),
- `xlErrNum` (pro chybu #NUM!),
- `xlErrRef` (pro chybu #REF!),
- `xlErrValue` (pro chybu #HODNOTA!).

Má-li tedy vlastní funkce vrátit chybu #N/A, použijte následující příkaz:

```
OdstranitSamohlasky = CVErr(xlErrNa)
```

Upravenou verzi funkce `OdstranitSamohlasky` najdete v dalším výpisu. Nová funkce používá konstrukci `If – Then` provádějící jinnou činnost, když argumentem není text. Využívá funkci pracovního listu Excelu `JE.TEXT`, která zjistí, zda zadaný parametr obsahuje text. Pokud ano, funkce pokračuje normálně. Pokud ale buňka text neobsahuje, vrátí chybu `#N/A`.

```
Function OdstranitSamohlasky(txt) As Variant
    ' Odstraňuje všechny samohlásky z textového argumentu
    ' Vrací #VALUE, pokud není txt řetězcem
    Dim i As Long
    OdstranitSamohlasky = ""
    If Application.WorksheetFunction.IsText(txt) Then
        For i = 1 To Len(txt)
            If Not UCase(Mid(txt, i, 1)) Like "[AEIOU]" Then
                OdstranitSamohlasky = OdstranitSamohlasky & Mid(txt, i, 1)
            End If
        Next i
```

```
    Else
        OdstranitSamohlasky = CVErr(xlErrNA)
    End If
End Function
```

Poznámka: Změnil jsem také datový typ hodnoty vracené funkcí. Protože tato funkce nyní může vracet i něco jiného než jen řetězec, změnil jsem datový typ na `Variant`.

Funkce s neurčitým počtem parametrů

Některé funkce pracovních listů v Excelu přebírají předem neurčený počet parametrů. Příkladem může být dobře známá funkce `SUMA`, která má následující syntaxi:

```
SUMA(číslo1, číslo2…)
```

První parametr je povinný, funkce však může mít v Excelu 2007 dalších až 254 parametrů. Ukázka funkce `SUMA` se čtyřmi parametry (jedná se o oblasti) vypadá takto:

```
=SUMA(A1:A5, C1:C5, E1:E5, G1:G5)
```

Typy parametrů se dokonce dají kombinovat. V následující ukázce jsou tři parametry – první je oblast, druhý je hodnota, třetí je výraz:

```
=SUMA(A1:A5, 12, 24*3)
```

Proceduru `Function`, která bude přebírat předem nespecifikovaný počet parametrů, můžete vytvořit i ve VBA. Celý trik spočívá v tom, že posledním (eventuálně jediným) parametrem funkce bude pole proměnných, před nímž bude klíčové slovo `ParamArray`.

Poznámka: `ParamArray` může být použito pouze u *posledního* parametru v seznamu parametrů funkce. Použit musí být vždy datový typ `Variant` a jedná se vždy o volitelný parametr (i když tu chybí klíčové slovo `Optional`).

V dalším výpisu kódu je zachycena funkce, která může mít libovolný počet parametrů (jednoduchých hodnot, neumí pracovat s oblastmi dvou a více buněk). Tato funkce vrací součet parametrů.

```
Function JednoduchaSuma(ParamArray seznamarg() As Variant) As Double
   For Each arg In seznamarg
      JednoduchaSuma = JednoduchaSuma + arg
   Next arg
End Function
```

Chcete-li uvedenou funkci změnit tak, aby pracovala s argumenty zahrnujícími více buněk, musíte doplnit další cyklus, který bude zpracovávat jednotlivé buňky ve všech argumentech:

```
Function JednoduchaSuma2(ParamArray seznamarg() As Variant) As Double
    Dim bunka As Range
    For Each arg In seznamarg
        For Each bunka In arg
            JednoduchaSuma2 = JednoduchaSuma2 + bunka
        Next bunka
    Next arg
End Function
```

Funkce `JednoduchaSuma` se podobá funkci `SUMA` v Excelu, není ovšem zdaleka tak flexibilní. Vyzkoušejte si tuto funkci s různými typy parametrů a uvidíte, že funkce selže, pokud bude některá z buněk obsahovat nečíselnou hodnotu.

Emulace funkce SUMA Excelu

V tomto oddílu vám představím vlastní funkci pojmenovanou `MojeSuma`. Na rozdíl od funkce `JednoduchaSuma` z předcházejícího oddílu napodobuje funkce `MojeSuma` funkci Excelu `SUMA` (téměř) perfektně.

Ještě předtím, než se podíváte na kód funkce `MojeSuma`, řekneme si něco o funkci `SUMA` (zajisté jste s ní již v Excelu pracovali). Tato funkce je opravdu univerzální. Může mít až 255 parametrů (dokonce i „chybějící" parametry) a jako parametry lze předávat číselné hodnoty, buňky, oblasti, textové vyjádření čísel, logické hodnoty, a dokonce i vložené funkce. Podívejte se například na tento vzorec:

```
=SUMA(B1,5,"6",,TRUE,SQRT(4),A1:A5,D:D,C2*C3)
```

Toto je korektní zápis obsahující všechny následující typy parametrů, které jsou v seznamu seřazeny podle pořadí v příkladu:

- Odkaz na jednu buňku,
- literál,
- řetězec představující číslo,
- chybějící parametr,
- logická hodnota TRUE,
- výraz, který používá jinou funkci,
- jednoduchý odkaz na oblast,
- odkaz na oblast zahrnující celý sloupec,
- výraz vypočítávající násobek dvou buněk.

Funkce `MojeSuma` (viz výpis 10.1) umí pracovat se všemi uvedenými typy parametrů.

Na webu: Sešit, jehož součástí je funkce `MojeSuma`, najdete na webu ke knize.

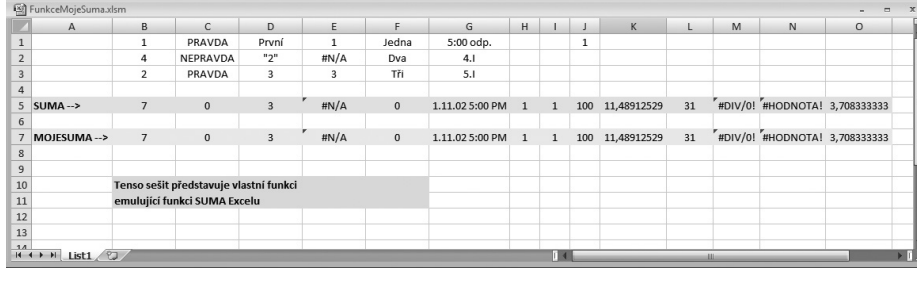
Výpis 10.1: Funkce MojeSuma

```
Function MojeSuma(ParamArray argumenty() As Variant) As Variant
    ' Emuluje funkci Excelu SUMA
    ' Deklarace proměnných
    Dim i As Variant
    Dim DocasnaOblast As Range, bunka As Range
    Dim ECode As String
    Dim m, n
    MojeSuma = 0

    ' Zpracujeme každý parametr
    For i = LBound(argumenty) To UBound(argumenty)
        ' Přeskočíme scházející parametry
        If Not IsMissing(argumenty(i)) Then
            ' Jaký datový typ to je?
```

```
            Select Case TypeName(argumenty(i))
                Case "Range"
                    ' Vytvoříme dočasnou oblast pro celé řádky či sloupce
                    Set DocasnaOblast = _
                       Intersect(argumenty(i).Parent.UsedRange, argumenty(i))
                    For Each bunka In DocasnaOblast
                        If IsError(bunka) Then
                            MojeSuma = bunka ' Vrátíme tuto chybu
                            Exit Function
                        End If
                        If bunka = True Or bunka = False Then
                            MojeSuma = MojeSuma + 0
                        Else
                            If IsNumeric(bunka) Or IsDate(bunka) Then _
                                MojeSuma = MojeSuma + bunka
                        End If
                    Next bunka
                Case "Variant()"
                    n = argumenty(i)
                    For m = LBound(n) To UBound(n)
                      MojeSuma = MojeSuma(MojeSuma, n(m)) ' Rekurzivní volání
                    Next m
                Case "Null"  ' Ignorujeme
                Case "Error" ' Vrátíme tuto chybu
                    MojeSuma = argumenty(i)
                    Exit Function
                Case "Boolean"
                    ' Zjistit zadání literálu TRUE a případně zpracovat
                    If argumenty(i) = "True" Then MojeSuma = MojeSuma + 1
                Case "Date"
                    MojeSuma = MojeSuma + argumenty(i)
                Case Else
                    MojeSuma = MojeSuma + argumenty(i)
            End Select
        End If
    Next i
End Function
```

Obrázek 10.6 zachycuje sešit s různými vzorci využívajícími funkce SUMA a `MojeSuma`. Je zřejmé, že obě funkce vracejí shodné výsledky.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		1	PRAVDA	První	1	Jedna	5:00 odp.			1					
2		4	NEPRAVDA	"2"	#N/A	Dva	4.I								
3		2	PRAVDA	3	3	Tři	5.I								
4															
5	SUMA -->	7	0	3	#N/A	0	1.11.02 5:00 PM	1	1	100	11,48912529	31	#DIV/0!	#HODNOTA!	3,708333333
6															
7	MOJESUMA -->	7	0	3	#N/A	0	1.11.02 5:00 PM	1	1	100	11,48912529	31	#DIV/0!	#HODNOTA!	3,708333333

Obrázek 10.6: Porovnání funkce SUMA s funkcí MojeSuma

Chcete-li se blíže seznámit s fungováním uvedené funkce, vytvořte nějaký vzorec, který s ní pracuje. Pak si nastavte v kódu bod přerušení a krokujte příkazy po jednotlivých řádcích. (Viz „Ladění funkcí“ dále v této kapitole.) Vyzkoušejte si to pro několik odlišných

typů argumentů a brzy budete velmi dobře rozumět fungování celé funkce. Při studiu kódu funkce `MojeSuma` si zapamatujte následující poznatky:

- Scházející parametry (určené pomocí funkce `IsMissing`) jsou prostě ignorovány.
- Procedura používá pro určení typu parametru (`Range`, `Error` atd.) funkci VBA `TypeName`. S každým typem parametru se totiž zachází odlišně.
- U parametru typu oblast (`Range`) funkce prochází každou buňku oblasti a jejich hodnoty přidává do mezisoučtu.
- Návratový typ funkce je `Variant`, protože funkce potřebuje vracet také chybu, pokud některý z jejích parametrů obsahuje chybovou hodnotu.
- Pokud parametr obsahuje chybu (například `#DIV0!`), funkce `MojeSuma` jednoduše také vrátí chybu – stejně jako funkce `SUMA` v Excelu.
- Funkce Excelu `SUMA` považuje textový řetězec za nulu, pokud se nevyskytuje jako literál (literál je skutečná hodnota, ne proměnná). Proto i funkce `MojeSuma` přičítá hodnotu buňky pouze tehdy, je-li možné ji vyhodnotit jako číslo (pro tento účel používám funkci `IsNumeric` jazyka VBA).
- U parametrů typu oblast používám funkci `Intersect`, pomocí které vytvářím dočasnou oblast. Ta bude obsahovat průnik zadané oblasti a oblasti využívané na pracovním listu. Toto opatření je zde pro případ, kdy oblast zadaná jako parametr obsahuje celý řádek nebo sloupec. Vyhodnotit celý sloupec či řádek by trvalo věčnost.

Možná jste zvědaví na poměr rychlostí funkcí SUMA a `MojeSuma`. Funkce `MojeSuma` je samozřejmě mnohem pomalejší. Poměr rychlostí však záleží na konkrétní rychlosti vašeho počítače a také na samotných vzorcích. Na mém počítači se pracovní list s 1 000 vzorců `SUMA` přepočítal okamžitě. Když jsem funkce `SUMA` nahradil funkcemi `MojeSuma`, celý přepočet trval asi 8 sekund. Ano, funkce `MojeSuma` by mohla být ještě trochu vylepšena, nikdy se však svou rychlostí ani zdaleka nepřiblíží rychlosti funkce `SUMA`.

Podstatou tohoto příkladu však není vytvořit novou funkci `SUMA`, to jste (doufám) pochopili. Příklad má demonstrovat postup tvorby vlastní funkce pracovního listu, která vypadá a pracuje jako stejná vestavěná funkce Excelu.

Ladění funkcí

Pokud ve svém pracovním listu používáte pro testování procedur `Function` vzorce, nebudou se chyby za běhu programu zobrazovat v dobře známém okně s chybovým hlášením. Dojde-li k chybě, vzorec jen vrátí chybovou hodnotu (`#HODNOTA!`). Naštěstí to ale nepředstavuje při ladění funkcí žádný problém, protože máte k dispozici celou řadu možných náhradních postupů:

- *Umístěte na důležitá místa kódu funkce funkci `MsgBox`, kterou budete sledovat hodnoty určitých proměnných.* Okna hlášení se naštěstí u funkcí pracovního listu při vyhodnocování zobrazují. Ujistěte se však, že ve svém pracovním sešitu máte jen jeden vzorec s tuto funkcí. Jinak se vám okno hlášení objeví pro každý vzorec, který bude vyhodnocován – a to vám určitě rychle začne vadit.
- *Testujte si funkci voláním z procedury `Sub` namísto volání ze vzorce na pracovním listu.* Chyby za běhu programu se v tomto případě budou zobrazovat normálním způso-

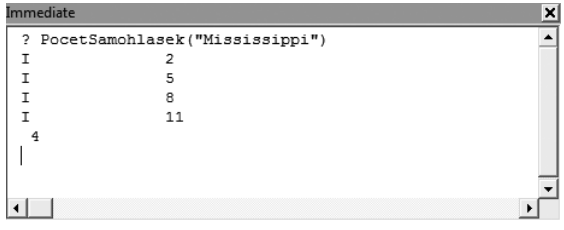
bem a vy tak můžete problém buď hned odstranit (pokud znáte příčinu), anebo se přepnout do debuggeru.

- *Vložte si do těla funkce zarážku (breakpoint) a funkcí potom krokujte.* V tomto případě budete mít k dispozici všechny standardní ladicí nástroje. Zarážka (bod přerušení) se nastaví tak, že kurzor umístíte do řádku funkce, na kterém se má provádění kódu zastavit, a z nabídky Debug vyberete příkaz Toggle Breakpoint (nebo stisknete F9). Jakmile se funkce vykonává, postupujte po jednotlivých řádcích stiskem F8.
- *Použijte v kódu jeden nebo více dočasných příkazů* `Debug.Print` *pro výpis hodnot do okna Immediate v editoru VB.* Chcete-li například sledovat hodnotu uvnitř cyklu, použijte podobný postup jako u následující rutiny:

```
Function PocetSamohlasek(r) As Long
   Dim Pocet As Long
   Dim i As Long
   Dim Znak As String * 1

   ' Spočítá samohlásky v zadaném řetězci
   Pocet = 0
   For i = 1 To Len(r)
      Znak = UCase(Mid(r, i, 1))
      If Znak Like "[AEIOU]" Then
         Pocet = Pocet + 1
         Debug.Print Znak, i
      End if
   Next i
   PocetSamohlasek = Pocet
End Function
```

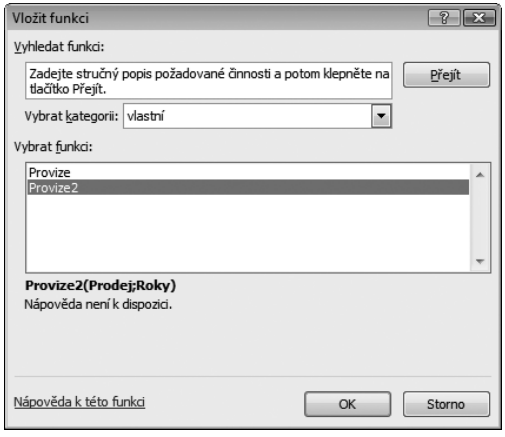
V tomto případě budou hodnoty dvou proměnných `Znak` a `i` tištěny do okna Immediate při každém spuštění příkazu `Debug.Print`. Obrázek 10.7 předvádí výsledek v případě, kdy byl funkci předán parametr.



Obrázek 10.7: Pomocí okna Immediate si můžete zobrazovat výsledky během vykonávání funkce

Práce s dialogem Vložit funkci

Dialog Vložit funkci Excelu je velmi užitečný nástroj. Při vytváření vzorce na pracovním listu totiž umožňuje vybrat určitou funkci ze seznamu (viz obrázek 10.8). Funkce pracovního listu jsou v tomto okně roztříděny do různých kategorií, aby se požadovaná funkce dala rychleji najít. Dialog Vložit funkci zobrazuje také vlastní funkce pracovního listu a dialogové okno Argumenty funkce vás vyzve k zadání parametrů funkce.



Obrázek 10.8: Vkládání vlastní funkce do vzorce

Poznámka: Vlastní funkce deklarované s klíčovým slovem `Private` se v dialogu Vložit funkci neobjeví. Pokud píšete funkci, která má být používána výhradně jinými procedurami VBA, deklarujte ji právě pomocí klíčového slova `Private`. Deklarováním funkce za soukromou ovšem nezamezíte jejímu použití ve vzorci na listu. Daná funkce se jen nezobrazí v dialogovém okně Vložit funkci.

Standardně jsou vlastní funkce zobrazovány v kategorii *vlastní*. V případě potřeby je však možné zařadit je do jiné kategorie. Funkci se také dá přiřadit popisný text, který vysvětluje její účel (doporučuji, abyste funkci tento text přiřadili vždy).

Poznámka: Dialogové okno Vložit funkci vám umožňuje hledat požadovanou funkci podle klíčových slov. Tento vyhledávací nástroj však bohužel nelze použít pro nalezení vlastních funkcí napsaných ve VBA.

Jak funkci přiřadit kategorii

Excel kupodivu nenabízí žádný přímý způsob pro zařazení funkce do určité kategorie. Pokud chcete, aby se vlastní funkce zobrazovala v jiné kategorii, než je kategorie *vlastní*, musíte napsat a vykonat určitý kód VBA.

Následující příkaz přiřadí funkci s názvem `Provize` do kategorie *finanční* (kategorie s číslem 1).

```
Application.MacroOptions Macro:="Provize", Category := 1
```

Poznámka: Tento příkaz je zapotřebí spustit pouze jednou (ne při každém otevření sešitu). Po jeho provedení se funkce objeví v zadané kategorii při každém otevření sešitu, který tuto funkci obsahuje.

Tabulka 10.1 obsahuje přehled čísel kategorií, které můžete použít. Všimněte si, že některé z kategorií (10 až 13) se normálně v dialogu Vložit funkci nezobrazují. Pokud ale funkci do některé z těchto kategorií zařadíte, kategorie se v dialogu objeví.

Tabulka 10.1: Kategorie funkcí

Číslo kategorie	Název kategorie
0	Žádná (funkce se objeví jen v seznamu všech funkcí)
1	Finanční
2	Datum a čas
3	Matematické
4	Statistické
5	Vyhledávací
6	Databáze
7	Text
8	Logické
9	Informační
10	Příkazy
11	Přizpůsobení
12	Ovládání maker
13	Externí/DDE
14	Vlastní
15	Inženýrská analýza
16	Krychle (novinka v Excelu 2007)

Přidání popisu funkce

Když v dialogu Vložit funkci nějakou funkci vyberete, objeví se její stručný popis. Popis vlastní funkce se dá určit dvěma způsoby: Buď použít dialog Makro, nebo napsat kód VBA.

Poznámka: Pokud nedefinujete popis své vlastní funkce, dialog Vložit funkci zobrazí následující text: „Nápověda není k dispozici".

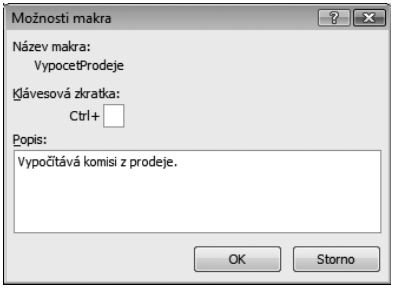
Vytvoření popisu vlastní funkce v dialogu Makro

Při vytváření popisu vlastní funkce se řiďte těmito kroky:

1. Vytvořte funkci v editoru VB.
2. Aktivujte Excel a přesvědčte se, že je sešit obsahující příslušnou funkci aktivním sešitem.
3. Zadejte příkaz Vývojář → Kód → Makra (nebo stiskněte kombinaci Alt+F8).

Dialog Makro vypíše seznam všech dostupných procedur. Vaše funkce tu však uvedeny nebudou.

4. Do pole Název makra zapište jméno vlastní funkce.
5. Klepněte na tlačítko Možnosti. Objeví se dialog Možnosti makra.
6. Do políčka Popis napište stručný popis funkce (viz obrázek 10.9). Políčko Klávesová zkratka nemá u funkcí žádný význam.



Obrázek 10.9: Zadání popisu vlastní funkce v dialogu Možnosti makra

7. Klepněte na tlačítko OK a potom na tlačítko Storno.

Po provedení předcházejících kroků bude dialog Vložit funkci zobrazovat při výběru vlastní funkce popis, který jste zadali v kroku 6.

Odkaz: Informace o vytváření vlastních témat nápovědy, která budou dostupná z průvodce funkcí, najdete v kapitole 24.

Když pro vložení vlastní funkce použijete dialog Vložit funkci, otevře se po klepnutí na tlačítko OK dialog s parametry funkce. U vestavěných funkcí Excelu tento dialog zobrazí popis každého z parametrů této funkce. Pro parametry vlastních funkcí si bohužel tyto nápovědy definovat nemůžete.

Vytvoření popisu vlastní funkce pomocí kódu VBA

Další způsob zadání popisu pro vlastní funkci nabízí kód VBA. Následující příkaz přiřazuje popis k funkci s názvem `Provize`.

```
Application.MacroOptions _
   Macro:="Provize", _
   Description := "Výpočet provizí z prodeje"
```

Tento příkaz stačí provést jen jednou, nikoli při každém otevření sešitu s danou funkcí.

Použití doplňků pro ukládání vlastních funkcí

Často používané vlastní funkce je vhodné uložit do souboru doplňku (*add-in*). Základní výhodou je tu skutečnost, že funkce pak můžete používat ve vzorcích všech sešitů. Navíc lze funkce používat bez zadání kvalifikátoru s názvem souboru.

Předpokládejme, že máte vlastní funkci s názvem `OdstranMezery` a že je uložena v souboru `MojeFunkce.xlsm`. Chcete-li tuto funkci použít ve vzorci v jiném sešitu, než je sešit `MojeFunkce.xlsm`, musíte vzorec zapsat v tomto tvaru:

```
=MojeFunkce.xlsm!OdstranMezery(A1:C12)
```

Když si ale ze souboru `MojeFunkce.xlsm` vytvoříte doplněk, a tento je zapnut, můžete odkaz na sešit vynechat a zadat vzorec takto:

```
=OdstranMezery(A1:C12)
```

Odkaz: Doplňkům je věnována kapitola 21.

Pozor: Potenciální riziko používání doplňků k ukládání vlastní funkcí tkví v tom, že váš sešit je pak na daném doplňku závislý. Potřebujete-li svůj sešit sdílet s kolegou, musíte s ním sdílet rovněž kopii doplňku, jenž danou funkci zahrnuje.

Používání funkcí API Windows

VBA si může „vypůjčit" metody z jiných souborů, které s Excelem ani s VBA nemají co do činění – například ze souborů DLL (dynamicky připojované knihovny), jež používá systém Windows i jiné programy. Díky tomu lze ve VBA provádět takové věci, které by jinak byly mimo možnosti tohoto jazyka.

Rozhraní API (*Application Programming Interface*) systému Windows je množina funkcí, které mají programátoři Windows k dispozici. Když budete volat funkci Windows z VBA, budete pracovat s API Windows. Řada prostředků systému Windows, které programátoři využívají, je k dispozici právě v knihovnách DLL. Tyto knihovny obsahují programy a funkce připojované k aplikaci až v době běhu programu, nikoli již v době překladu.

Excel samotný používá celou řadu knihoven DLL. Kód mnoha těchto knihoven DLL by mohl být přeložen přímo do spustitelného souboru `excel.exe`, vývojáři jej však raději ukládají do knihoven DLL, protože ty se do paměti zavádějí jen v případě potřeby. Toto oddělení málo používané funkčnosti přináší menší velikost hlavního spustitelného souboru Excelu. Kromě toho se snižuje i spotřeba operační paměti (zopakujme, že knihovny DLL se do paměti nahrávají jen v případě potřeby).

Knihovny DLL se rovněž používají pro sdílení kódu. Většina programů pro Windows používá například shodné dialogy pro otevírání a ukládání souborů. Systém Windows proto nabízí knihovnu DLL, která obsahuje kód pro vytvoření řady standardních dialogů. Programátoři pak mohou tuto knihovnu DLL volat namísto toho, aby si psali své vlastní rutiny.

Pokud programujete v jazyce C, můžete si napsat vlastní knihovnu DLL a potom ji ve VBA používat. Jazyk Microsoft Visual Basic má rovněž podporu pro vytváření souborů DLL, které je možné volat z Excelu. (Ovšem pozor, Visual Basic tu neznamená VBA!)

Příklady práce s Windows API

Předtím, než některou funkci Windows API použijete, musíte ji nejdříve *deklarovat*, a to na začátku standardního modulu VBA, ještě před první procedurou. V modulech kódu pro objekty `ThisWorkbook` a v modulech kódů pro pracovní listy musíte funkce API deklarovat slovem `Private`.

Deklarace funkce API musí být absolutně přesná. Příkaz deklarace VBA říká interpreteru VBA následující fakta:

- Kterou funkci API budete používat,
- ve které knihovně je tato funkce API umístěna,
- jaké parametry funkce API používá.

Deklarovanou funkci API můžete pak v kódu VBA libovolně používat.

Zjištění adresáře Windows

Následuje příklad deklarace funkce API:

```
Declare Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Tato funkce se dvěma parametry vrací název složky, ve které je nainstalován systém Windows (tento údaj ve VBA jinak nezjistíte). Po zavolání uvedené funkce bude název složky Windows uložen do proměnné `lpBuffer` a délka řetězce s názvem adresáře bude uložena v proměnné `nSize`.

Po přidání příkazu `Declare` na začátek modulu můžete tuto funkci volat jako funkci `GetWindowsDirectoryA`. Další výpis předvádí volání funkce a zobrazení výsledku v okně hlášení.

```
Sub ZobrazitAdresarWindows()
   Dim CestaWindows As String
   Dim AdresarWindows As String
   CestaWindows = Space(255)
   AdresarWindows = Left(CestaWindows, GetWindowsDirectoryA _
      (CestaWindows, Len(CestaWindows)))
   MsgBox AdresarWindows, vbInformation, "Adresář Windows"
End Sub
```

Vykonání procedury `ZobrazitAdresarWindows` způsobí zobrazení dialogového okna hlášení s názvem adresáře Windows.

Často si programátoři vytváří pro funkce API tzv. *obálky* (wrapper). Obálku tvoří vlastní funkce, která uvnitř sebe používá funkci API. Takto se dá práce s funkcemi API zásadně zjednodušit. Podívejte se na příklad takovéto „obálkové“ funkce VBA:

```
Function AdresarWindows() As String
   ' Vrací adresář, kde je instalován systém Windows
   Dim CestaWindows As String
   CestaWindows = Space(255)
   AdresarWindows = Left(CestaWindows, GetWindowsDirectoryA _
      (CestaWindows, Len(CestaWindows)))
End Function
```