Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra fyziky

Automatizace úkonů v tabulkovém kalkulátoru pomocí VBA

Bakalářská práce

Vedoucí práce: ing. Michal Šerý

Vypracoval: Jan Macháček

Automatizace úkonů v tabulkovém kalkulátoru pomocí VBA

V této práci jsem se zaměřil na popis jedné z částí tabulkových kalkulátorů, konkrétně na jejich makrojazyk – Visual Basic for Application – VBA. Je zde popsána struktura jazyka, jeho vlastnosti a charakteristika. V práci je uveden postup, jak vytvořit makro bez znalosti programování. Také zde popisuji postup pro vytvoření vlastních funkcí a procedur a jejich části. Zařadil jsem do práce rovněž svou vlastní aplikaci, která je využívána v praxi.

V současnosti není na trhu k dispozici pouze jediný nástroj, celá má práce se nicméně soustředí na jednu konkrétní variantu vybraného tabulkového kalkulátoru - program Excel, který je součástí kancelářského balíku Microsoft Office Professional verze 2002 (XP). Důvod, proč jsem si tento produkt k svému zkoumání zvolil, pramení z prostého faktu, že je mým každodenním pracovním nástrojem v zaměstnání.

V závěrečné části srovnávám vybraný produkt s jiným alternativním programem.

Klíčová slova:
Makro, tabulkový kalkulátor, procedura, funkce, objekt, VBA, Excel.

Task automation in spreadsheet program with the assistance of VBA

My thesis primarily deals with a part of spreadsheet program - their macro language - Visual Basic for Application – VBA. The work describes the structure, features and characteristics of the language. It also brings a procedure enabeling to create macro without the knowledge of programming, as well as the procedure how to create one´s own functions and procedures or their part. The work also includes a source code of my own application that is used in practice.

At present there are more instruments on the market, not just one. However, all my thesis (except the final evaluation) concentrates on one particular variation of this particular spreadsheet program – Excel, which is a part of Microsoft Office Professional Version 2002 (XP). The reason I chose this product simply comes from the fact that it is my daily work tool in my job.

In the final part of my thesis I compare the chosen software with another alternative software.

Keywords:

Macro, spreadsheet program, procedure, function, object, VBA, Excel.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne …… Jan Macháček

Obsah:

1 Historický vývoj maker ……...……...……...…...…7

2 VBA a objektové programování ...……...……...……...…9

2.1 Moduly ...…….……..... 9

2.2 Procedury a funkce …….……..... 9

2.3 Objekty …….……...…..… 9

2.4 Třídy objektů ……...…….……. 10

2.5 Kolekce ……...…....…….……. 11

2.6 Hierarchie objektů ……....……...…….. 11

2.7 Odkazy a na objekty v kódu ……....……...……... 14

2.8 Zkrácené odkazy na objekty ……....….....…… 15

2.9 Rozdíl mezi metodami a vlastnostmi v kódu ….....…….……. 16

2.10 Události ……...…….……. 17

2.10.1 Vznik události……....……..…… 17

2.10.2 Umístění procedur událostí ……...…….……... 18

2.10.3 Jak zabránit výskytu událostí ……...…….……..…… 20

2.10.4 Události sešitu ……...……..……... 22

2.10.5 Události listů ……...……...…..…… 22

2.10.6 Události grafů (listů typu graf) ……...……...……..…… 22

2.10.7 Události aplikace Excel …….....……. 22

3 Makroprogramování v aplikacích …….....….....…… 23

3.1 Makra …….....……. 23

3.2 Nahrávání maker pomocí záznamníku instrukcí ……...……. 24

3.3 Psaní maker v editoru VBA ……...……...…….. 31

3.3.1 Procedury a funkce ……..……...……. 31

3.3.2 Jednotlivé části podprogramů …….…… 33

3.3.3 Příkazy a jejich zápis …….……..…..……. 34

3.3.4 Proměnné ……...…….…… 34

3.3.5 Konstanty …….…..…..…… 35

3.3.6 Názvy objektů, proměnných a konstant ……...……..……. 35

3.3.7 Výrazy ……....…..…..…… 36

3.3.8 Operátory …….…..….....……. 36

3.3.9 Datové typy proměnných a konstant …….…...……..…… 37

3.3.10 Rozsah platnosti a životnost proměnných a konstant ……...……..….. 38
3.3.11 Deklarace proměnných ……..…….…… 40
3.3.12 Zápis hodnot do proměnných ……..……..…….. 41
3.3.13 Pole proměnných ……...……. 43
3.3.14 Parametry a jejich druhy …….……..…….. 44
3.3.15 Rozhodovací bloky …….……..….. 46
3.3.16 Cykly ……...…….……...……... 48
3.4 Spouštění procedur …….……...…….…....….. 50
3.4.1 Spouštění procedur z editoru VBA …….…….……...…… 50
3.4.2 Spouštění procedur z dialogového okna Makro … ……...……... 50
3.4.3 Spouštění procedur z jiných procedur a funkcí ……..……..…….. 51
3.4.4 Spouštění procedur z panelů nástrojů …….…….……..……. 51
3.4.5 Spouštění procedur z nabídky Excelu ……...…….....…… 53
3.4.6 Spouštění procedur pomocí příkazových tlačítek v listech ….……. 54
4 Ukázková aplikace …….…….…… 55
4.1 Popis činnosti aplikace ……..……..…….. 55
4.2 Kód aplikace …….…….. 56
5 Závěr ….……..……..…...……..…….. 62
5.1 Porovnání dostupných nástrojů …….……...…….. 62
5.2 Zhodnocení použitelnosti pro uživatele …….……....……. 65
Seznam literatury ……...……...……...….. 67

1 Historický vývoj maker

Elektronické tabulkové kalkulátory začaly vznikat roku 1961. Tehdy profesor Richard Mattesich vyslovil myšlenku o počítačovém tabulkovém kalkulátoru. Jeho představy a představy dalších vývojářů byly zpracovány v roce 1978 Danem Bricklinem, tehdejším studentem Harvard Business Schoul. Spolu s Bobem Frankstonem vyvinul program jménem VisiCalc. Tento program byl prvním počítačovým tabulkovým kalkulátorem. Byl vyvinutý pro Apple II.

Na začátku 80. let spolu s úspěšným zavedením IBM PC se trh s tabulkovými kalkulátory velmi rozšířil. Vývojáři VisiCalc byli příliš pomalí na to, aby byli schopni udržet krok. V roce 1983 se vedoucím produktem na trhu stal Lotus 1-2-3. Lotus 1-2-3 byl prvním programem, který uměl pracovat s tabulkami, grafy a databázovými funkcemi. Velkou výhodou programu Lotus 1-2-3 bylo, že měl implementovaná makra. Program umožnil vývojářům psát makra a testovat je. Běžným uživatelům byl k dispozici záznam maker, každodenní práce s tabulkovým kalkulátorem bylo možné částečně zautomatizovat. Říká se, že tato funkce byla do programu přidána na poslední chvíli, těsně před uvedením na trh. Díky tomu sklidil Lotus 1-2-3 velký úspěch a předběhl ostatní výrobce podobných programů.

První makra v programu Excel bylo možné vytvářet již od verze 2.1, bylo však nutné je vložit na speciální list pro makra. List s makry se ukládal do zvláštního souboru s příponou *.xml. Sešity Excel používaly příponu *.xls, kterou používají dodnes. Makra byla označována jako makra XLM nebo makra Excel4. Tímto způsobem bylo možné je podle pojmenování odlišit od kódu VBA uvedeného v programu Excel verze 5. Makrojazyk tabulkového kalkulátoru Excel byl daleko mocnějším nástrojem než makra z Lotus 1-2-3. Excel nabízel několik stovek funkcí, Lotus mu nemohl konkurovat.

Celý komplexní koncept maker měl svá pro a proti. Zkušení programátoři, kteří uměli využít možnosti tohoto nástroje, jej uvítali. Avšak rozmanitost funkcí byla na druhou stranu pro mnohé uživatele také jistou bariérou. Neexistovalo žádné jednoduché a pohodlné spojení manuální práce s programem Excel a programování maker. Pro obyčejného uživatele tabulkového kalkulátoru Excel bylo učení programovacího jazyka velmi náročné, což mnoho uživatelů odradilo. Další nevýhodou bylo vázání produktu na operační systém Windows, který byl známý pro své vysoké nároky na prostředky.

Velikou chybu ovšem udělal i Lotus, když doufal, že operační systém OS/2 nakonec nahradí Windows. Programátoři neplánovali vývoj Lotus 1-2-3 pro Windows. Velmi rychle bylo jasné, že Windows budou hrát rozhodující roli. První pokus přenést Lotus 1-2-3 přece jen na platformu Windows proběhl v roce 1991. Nová verze programu Lotus 1-2-3 však byla spíše aplikací pro DOS s vlastním grafickým rozhraním, než program pro Windows. Microsoft mezitím dosáhl s Windows a sadou Office prvenství na trhu. Bylo již pozdě snažit se udržet krok s tabulkovým kalkulátorem Excel a úspěch programu Lotus 1-2-3 byl nenávratně pryč. V roce 1993 Microsoft sjednotil programování jednotlivých aplikací ze sady Office. Vznikl tak programovací jazyk Visual Basic for Application – VBA. [1]

2 VBA a objektové programování

2.1 Moduly

Veškerý kód jazyka VBA je uložen v modulech. Moduly jsou fyzicky uloženy v sešitech (nikdy samostatně) a lze je spouštět jen z nich. V modulu jsou uloženy jednotlivé procedury a funkce, které jsou nejmenší spustitelnou jednotkou, obsahující programový kód. Dále zde může být úvodní část s deklaracemi proměnných a polí.

2.2 Procedury a funkce

Procedury a funkce obsahují příkazy, které jsou interpretem kódu prováděny. Jsou prováděny směrem od začátku do konce procedury či funkce. Tok provádění příkazů je možné změnit pomocí tzv. řídících struktur. Každý příkaz je obvykle psán na samostatný řádek. Jako příkaz lze uvnitř procedury či funkce volat vestavěný příkaz nebo funkci VBA, jehož význam je definován přímo v jazyku, ale také vlastní procedury či funkce, které programátor napíše, a které mohou plnit nejrůznější úlohy. Funkce na rozdíl od procedury vrací určitou hodnotu, kterou můžeme dále použít. Proceduru či funkci můžeme volat z jiné procedury či funkce jazyka VBA, funkce však lze navíc použít i ve vzorci na pracovní listu.

2.3 Objekty

Z vnějšího pohledu objekty nejvíce připomínají klasický „black box“ – černou skříňku. To je věc nebo struktura o jejichž vnitřku nevíme nic, ale když na takový objekt budeme určitým způsobem působit, dočkáme se určité reakce. První charakteristikou objektu je tedy fakt, že jde o uzavřenou strukturu.

Tyto uzavřené jednotky mají svůj vnitřní obsah a svou funkčnost. S obsahem objektu se pracuje pomocí jednotlivých funkcí, které jsou v něm uloženy. Protože však úplně uzavřená černá skříňka není vůbec přístupná, nedala by se vůbec k ničemu použít, umí objekty také komunikovat se svým okolím, tedy přijímat nějaké zprávy a reagovat na ně.

Pro vysvětlení pojmu objekt je nejlepší použít příklad z běžného života. Například pračku zná každý. Navenek je to uzavřená krabice, o jejím přesném obsahu mají ponětí jen pracovníci servisních středisek a výrobce. Pro nás jako uživatele je podstatné jen to, že pračka funguje. Z jejího obsahu nás normálně zajímá jen buben, do něhož se dává

prádlo. Pračka má na sobě několik ovládacích prvků, kterými řídíme její činnost. Některé prvky přímo nic nespouští, pouze mění určité parametry vlastního praní – teploty vody, počet otáček při ždímání, velikost dávky, typ pracího režimu. Také je na pračce tlačítko, kterým se spouští. Pračka tedy navenek vystupuje jako kompaktní celek, se kterým komunikujeme pomocí ovládacích tlačítek. Naše manipulace má vliv jednak na charakteristiky objektu, které mění jeho chování – teplota vody, počet otáček při ždímání apod., jednak můžeme spouštět výkonné příkazy – zapnout, vypnout.

V objektové terminologii se používají termíny vlastnosti a metody. Metoda je nějaká činnost, kterou objekt provede, zatímco vlastnosti popisují vzhled nebo stav objektu. Stisk tlačítka spouštějícího pračku je tedy ekvivalentní volání metody v kódu, zatímco přidání prádla do bubnu pračky je obdobou změny vlastnosti objektu (hmotnosti prádla v pračce). Komunikace mezi pračkou a okolím se provádí pomocí stisku tlačítek. Pračka na stisk tlačítka vždy určitým způsobem reaguje, i když se tato reakce často nijak neprojeví (když otočíme regulátorem teploty, nic neuvidíme, ani neuslyšíme). Samotná výměna obsahu bubnu nemá na pračku žádný praktický vliv – spustit ji můžeme i prázdnou.

Můžeme použít i poměrně jasnou ukázku z VBA. Pokud například budeme chtít zjistit počet objektů v kolekci Worksheets (objekt představující pracovní listy sešitu), použijeme vlastnost Count. Na výsledek se pak zeptáme kódem Worsheets.Count. Využití metody je transparentní například při zavírání aplikace. Pro tuto činnost poslouží kód Application.Quit. Některé objekty navíc mohou rozpoznávat tzv. Událost (Event), kdy kód proběhne na základě nějaké akce (klepnutí myší, změna obsahu, opuštění objektu apod.).

Pomocí jazyka VBA pracujeme především s objekty, které aplikace obsahuje a které vyjadřují její určitou část (objekty programu Excel).

2.4 Třídy objektů

Excel obsahuje více než 100 tříd objektů, reprezentujících ve VBA například sešit, pracovní list, oblast buněk, graf nebo nakreslený tvar. Jednotlivé třídy objektů dodržují určitou hierarchii, která je vyjádřena objektovým modelem. Každá třída se od ostatních nějak odlišuje. Každá třída má svou vlastní sadu vlastností a metod, kterou může použít. V praxi však v programovém kódu obvykle nepracujeme s třídou, ale s instancí. Třída sama je abstraktní. Konkrétním objektům dané třídy v paměti počítače

se správně říká instance. Instance mají všechny vlastnosti i metody, která má jejich třída.

2.5 Kolekce

Některé objekty jsou obsaženy v kolekcích. Pojem kolekce označuje skupinu objektů stejného typu. Samotné kolekce jsou v rámci objektového modulu chápány také jako objekty. Od jednotlivých objektů, které obsahuje, se dá odlišit svým názvem, ve kterém je oproti názvu těchto objektů obvykle přidané písmeno „s“ – CommandBars je kolekce všech objektů třídy CommandBar.

V kódu VBA můžeme pracovat buď s celou kolekcí jako s objektem, nebo s jejími jednotlivými členy. Záleží na tom, co chceme udělat. Kolekce většinou (někdy ne) mají metody pro přidání nebo odstranění svého člena. Například do kolekce Worksheets můžeme v makru přidat nový pracovní list metodou Add nebo ho z kolekce odstranit metodou Delete. V kolekci Workbooks však metoda Delete není – sešit z kolekce Workbooks odstraníme jen jeho uzavřením, pro které je ale určena jiná metoda, a to metoda sešitu, ne kolekce.

Kolekce výrazně zjednodušují celý objektový model. Umožňují jednoduchým způsobem zpracovat všechny objekty stejného druhu. Například všechny otevřené sešity uzavřeme tak, že projdeme všechny členy kolekce Workbooks a uzavřeme je. V každé kolekci jednoduše zjistíme počet členů. Pro přístup k jednotlivým členům kolekce slouží jejich pořadové číslo (tzv. ordinální index) nebo název.

2.6 Hierarchie objektů

Jestliže je celý tabulkový kalkulátor dostupný pomocí objektů, je žádoucí aby tyto objekty měly mezi sebou vazby. Vztahy mezi objekty jsou vyjádřeny pomocí hierarchického modelu, ve kterém jsou si objekty navzájem podřízeny. Na nejvyšším místě celého modelu obvykle stojí jeden subjekt, jenž je kontejnerem pro objekty na úrovni o jeden stupeň nižší. Ty zase v sobě obsahují další objekty. A tak to jde dále až k objektům nejnižší úrovně, které už kontejnery nejsou. Na nejvyšší úrovni v aplikaci je objekt Application. Například samotný Excel je vyjádřen objektem s názvem Application. Tento objekt Application obsahuje další objekty, které však existují jen tehdy, existuje-li sám objekt Application. (Excel musí být spuštěn, což je logické). Celkem je objektu Application podřízeno asi 50 (přesný počet je v každé verzi programu Excel jiný) tříd objektů a kolekcí. Nejzajímavější z nich jsou:

Workbooks – kolekce všech sešitů – objektů Woorkbook

Windows – kolekce všech objektů Window, neboli oken uvnitř hlavního okna Excelu

CommandBars – kolekce všech nabídek a panelů nástrojů

Dialogs – kolekce všech vestavěných dialogových oken

AddIns – kolekce všech objektů AddIn, neboli doplňků

Debut – objekt používaný při ladění programů

Každý z těchto objektů může opět obsahovat další objekty. Například kolekce Woorkbooks se skládá ze všech otevřených sešitů Workbook. Každý sešit Workbook obsahuje další objekty, opět uvedu jen některé:

Sheets – kolekce všech listů v sešitu

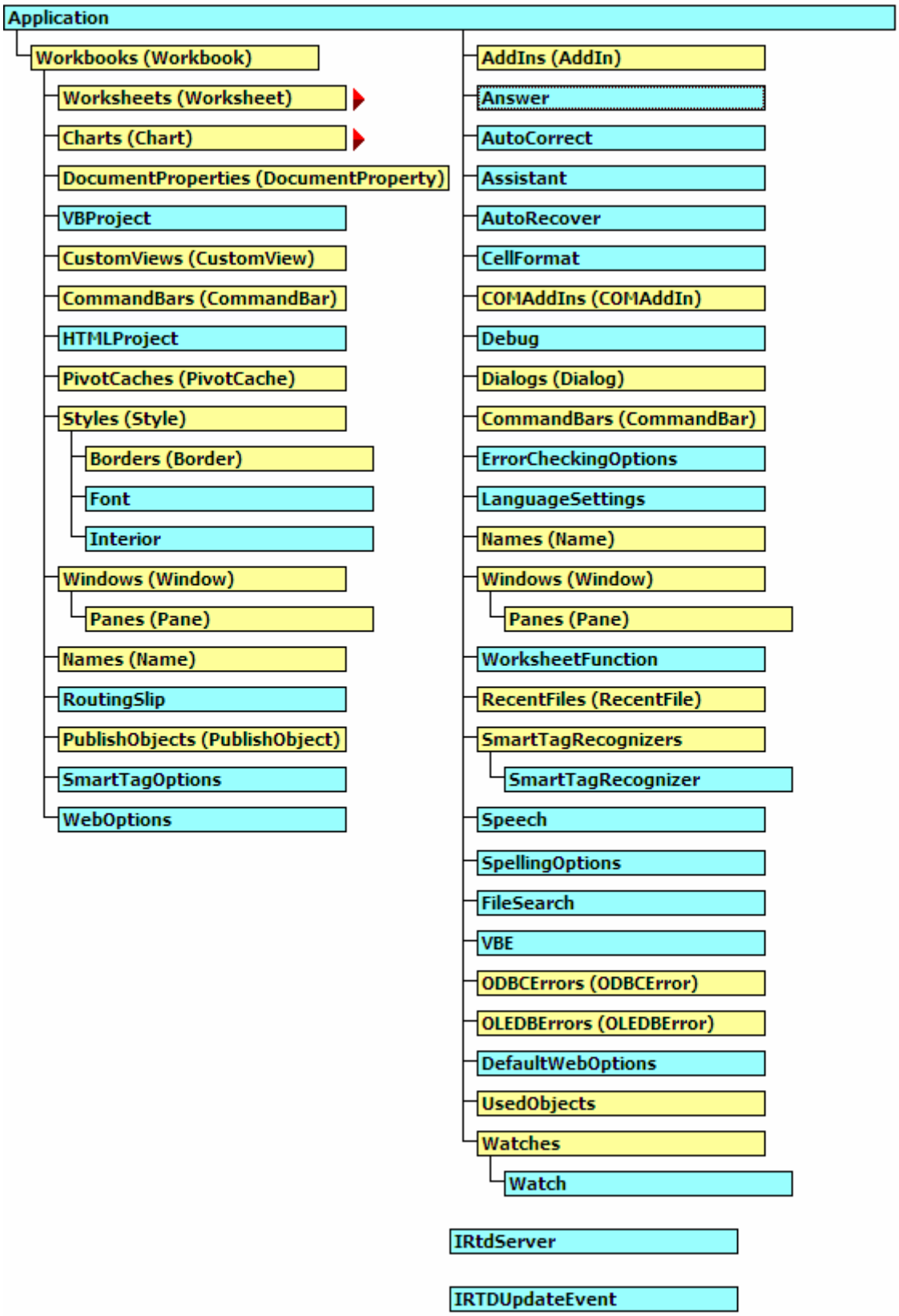
Worksheets – kolekce všech pracovních listů v sešitu, objektů Worksheet

Charts – kolekce listů typu graf – objektů Chart

Names – kolekce pojmenovaných názvů v listu – objektů Name

Styles – kolekce formátovacích stylů – objektů Style

Kompletní objektový model tabulkového kalkulátoru Excel je znázorněn formou diagramu v nápovědě programu.


Application
Workbooks (Workbook)
Worksheets (Worksheet)
Charts (Chart)
DocumentProperties (DocumentProperty)
VBProject
CustomViews (CustomView)
CommandBars (CommandBar)
HTMLProject
PivotCaches (PivotCache)
Styles (Style)
Borders (Border)
Font
Interior
Windows (Window)
Panes (Pane)
Names (Name)
RoutingSlip
PublishObjects (PublishObject)
SmartTagOptions
WebOptions
AddIns (AddIn)
Answer
AutoCorrect
Assistant
AutoRecover
CellFormat
COMAddIns (COMAddIn)
Debug
Dialogs (Dialog)
CommandBars (CommandBar)
ErrorCheckingOptions
LanguageSettings
Names (Name)
Windows (Window)
Panes (Pane)
WorksheetFunction
RecentFiles (RecentFile)
SmartTagRecognizers
SmartTagRecognizer
Speech
SpellingOptions
FileSearch
VBE
ODBCErrors (ODBCError)
OLEDBErrors (OLEDBError)
DefaultWebOptions
UsedObjects
Watches
Watch
IRtdServer
IRTDUpdateEvent

2.7 Odkazy na objekty v kódu

Při práci s objekty v kódu musíme vždy zadat název objektu. To však nemusí stačit. Objekt se vyskytuje vždy v nějaké úrovni objektového modelu, a v kódu je proto nutné zadat i toto umístění. Jednotlivé úrovně objektového modelu se v kódu vyjadřují pomocí tečkové konvence, kdy je každá úroveň oddělena tečkou. Tečka se používá i k oddělení objektu od jeho vlastností či metody.

Na nejvyšší rovni hierarchie stojí objekt Application, který v úplném odkazu na objekt stojí vždy zcela vlevo. Úplný odkaz na kolekci sešitů Workbooks bude tedy vypadat takto:

```
Application.Workbooks
```

Je to zřejmé při pohledu do diagramu objektového modelu. Uvidíme tam kolekci Woorkbooks na úrovni o jeden stupeň níže než objekt Application. Chceme-li pracovat s nějakým objektem obsaženým v kolekci, vložíme název objektu do závorek za název kolekce:

```
Application.Workbooks("prvni.xls")          ' odkaz na sešit první.xls
```

Názvy objektů mohou mít různou podobu. Například název listu ve VBA odpovídá názvu na záložce listu. Název sešitu je vždy jen název souboru XLS, včetně přípony. Při odkazu na sešit nesmíme příponu vynechat a nesmíme zadat úplnou cestu. Následující dva odkazy na sešit jsou neplatné

```
Application.Workbooks("prvni")
        ' není zadána přípona
Application.Workbooks("c:\pokus\prvni.xls")
        ' zadána celá cesta k souboru
```

To je také důvodem, proč nemůžeme v programu Excel otevřít dva sešity se stejným názvem souboru, přestože jsou uloženy v různých složkách. Excel sešity rozlišuje jen podle názvu souboru. Dále můžeme používat odkazy pomocí pořadového čísla. Následující řádek je odkaz na prvního člena kolekce Workbooks:

```
Application.Workbooks(1)
```

Musíme ale dávat pozor na to, zda pořadové číslo prvního člena kolekce je jednička nebo nula. Obecně se dá říci, že u kolekcí programu Excel je to jednička, u některých kolekcí s původem mimo program Excel se však může začínat i nulou. Konkrétní údaje o způsobu číslování najdeme v nápovědě k dané kolekci.

Chceme-li zjistit nějaký údaj o celé kolekci nebo s ní něco udělat, odkážeme přímo na ni. Například se v kódu vypíše toto:

```
Application.Workbooks.Count          ' zjištění počtu otevřených sešitů
```

Kolekce pracovních listů obsažených v sešitu prvni.xls je vyjádřena odkazem:

```
Application.Workbooks("prvni.xls").Worksheets
```

Odkaz na první list v tomto sešitu bude v kódu zapsán takto:

```
Application.Workbooks("prvni.xls").Worksheets. ("List1")
```

2.8 Zkrácené odkazy na objekty

Úplný odkaz má několik nevýhod. Ve většině příkazů je takový zápis hodně dlouhý. Orientace v takovém kódu je značně ztížená. Z tohoto důvodu není nutné uvádět v odkazu na objekt všechny jeho nadřízené objekty. Jestliže však použijeme zkrácený zápis, musíme si pamatovat, že Excel si vynechané objekty do odkazu doplní sám, a to podle určitých pravidel. U kolekce Workbooks se není čeho obávat, protože nad ní je v odkazu už jen objekt Application, a ten není možné nahradit ničím jiným. Takže následující dva řádky jsou významově shodné:

```
Application.Workbooks("prvni.xls")
Workbooks("prvni.xls")
```

U jednotlivých sešitů – na které odkazujeme pomocí kolekce Workbooks – to už tak jednoduché není. Jestliže vynecháme určení sešitu, pak Excel předpokládá, že máme na mysli sešit aktuální, tedy ten, jehož okno je v programu Excel aktivní.
Například:

```
Workbooks("Sešit1"). Worksheets("List1")
    ' tímto řádkem odkazujeme na pracovní list List1 v sešitu Sešit1, i když
    aktivním sešitem může být jiný sešit
Worksheets("List1")
    ' tímto řádkem odkazujeme na list v aktivním sešitu, ať už se sešit
    jmenuje jakkoli
```

Pro odkaz na určitou oblast buněk (například na buňku A1) na listu s názvem List1 a v sešitu Sešit1 můžeme použít následující výraz:

```
Workbooks("Sešit1"). Worksheets("List1").Range("A1")
```

Tato forma zápisu připadá v úvahu vždy, když si nejsme jisti, který sešit je právě aktivní. Jestliže víme (bezpečně!), že aktivní je právě Sešit1, můžeme použít tento kratší zápis:

```
Worksheets("List1").Range("A1")
```

Pokud je aktivním listem List1 v sešitu Sešit1 můžeme použít ještě jednodušší zápis:

```
Range("A1")
```

Vynecháme-li tedy některé odkazy v hierarchii objektů, Excel dosadí ty, které jsou právě aktuální (aktivní).

Samotný odkaz na objekty, jak je tomu v těchto příkladech, nedělá nic. Aby se provedlo něco smysluplného, musíme změnit vlastnosti odkazovaného objektu nebo zadat metodu objektu, která se má provést.

2.9 Rozdíl mezi metodami a vlastnostmi v kódu

Vlastnost popisuje vzhled či stav objektu. Metoda provádí činnost. Z toho vyplývá, že vlastnost má vždy nějakou hodnotu. Například vlastnost Name obsahuje název objektu, a to v textové podobě. Vlastnost Visible říká, zda je objekt viditelný či ne. Tentokrát jde ovšem o logickou hodnotu (ano či ne – je vidět nebo vidět není). Vlastnost Column vrací číslo sloupce, v němž se nalézá určitá buňka.

Hodnoty vlastností zjišťujeme nebo je zadáváme. Proto se vlastnosti v kódu vyskytují vždy v přikazovacím výrazu nebo ve výrazu, který vrací hodnotu.

```
ActiveCell.Value = 20
    ' tento řádek přiřadí do vlastnosti Value hodnotu 20. Value je vlastnost
    objektu ActiveCell
```

Při zjišťování hodnoty vlastnosti ukládáme získaný údaj obvykle do proměnné. Například:

```
Pocet = Application.Workbooks.Count
    ' tímto řádkem přiřadíme do proměnné Pocet počet právě otevřených
    sešitů. Workbooks je objekt a Count je jeho vlastnost
```

Na rozdíl od vlastnosti provádí metoda nějakou akci.

```
Selection.Clear
```

Metoda Clear smaže obsah objektu, na kterém ji spustíme. Nepřiřazuje nikam žádnou hodnotu, proto u ní schází rovnítko.

Některé metody vracejí hodnotu (fungují zároveň jako funkce), takže se mohou vyskytnout i v přiřazovacím příkazu. Ovšem jen na jeho pravé straně!

```
Koef = Application.InputBox("Zadej číslo")
            ' v tomto případě je InputBox metoda objektu Application
```

Většina metod má také parametry pro další upřesnění jejich činnosti. Například:

```
Range("C5")Insert xlShiftDown
```

```
' v tomto příkladu je na místo buňky C5 vložena buňka a původní buňka je odsunuta směrem dolů
```

V tomto příkladu má metoda Insert jeden parametr, kterým je směr posunu původní buňky. Slovo xlShiftDown je zástupný výraz pro určitou hodnotu, neboli konstanta. [2]

2.10 Události

2.10.1 Vznik události

Kromě vlastností a metod se k objektům ještě nerozlučně váží události. Událost je akce vyvolaná uživatelem nebo systémem, na kterou objekt dokáže reagovat. Mezi takové akce patří například otevření sešitu, klepnutí či poklepání myší na objekt, změna hodnoty v buňce a podobně. Každý objekt má přesně definovanou sadu událostí, které umí rozeznat. Pro vlastní programování se obvykle použije jen jejich nepatrná část, ale programátor samozřejmě musí vědět, k čemu se dá určitá událost použít.

Každá událost má přiřazenou svou proceduru (říká se jí událostní procedura), v níž obvykle není vložen žádný kód. Pokud však nějaký kód do takové procedury napíšeme, bude spuštěn při každém výskytu dané události, tedy například při otevření nějakého sešitu nebo před jeho uzavřením, před přepočítáním sešitu, po změně hodnoty v buňce a podobně. Je jasné, že události představují značný potenciál, který je možné použít pro tvorbu robustních a výkonných aplikací.

Některé akce spouští více událostí za sebou. Například při otevírání sešitu proběhnou postupně tyto události sešitu:

Open
Activate
WindowActivate

Při uzavírání sešitu probíhají (minimálně) tyto události sešitu:

BeforeClose
WindowDeactivate
Deactivate

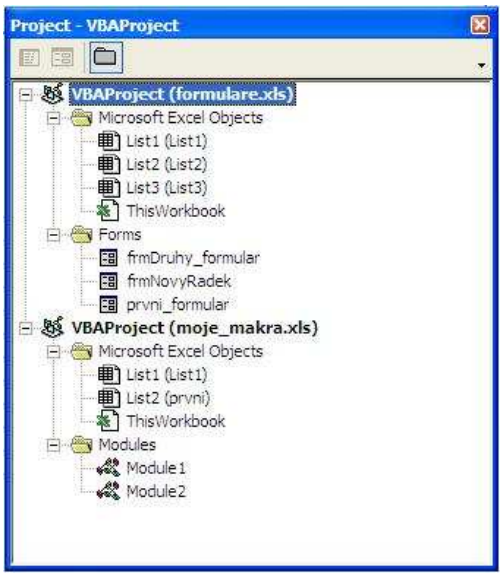
Posloupnosti událostí bývají mnohem složitější. V obou příkladech šlo navíc jen o ukázky na úrovni sešitu. Například při přidání nového listu do sešitu však proběhnou i události na úrovni listu a celé aplikace. Posloupnost událostí navíc nemusí být zcela logická – při přidání nového listu do sešitu proběhne nejdříve událost SheetActivate (aktivace listu) a až poté událost WorkbookNewSheet (nový list v sešitu).

Aplikace řízené událostmi jsou v systému Windows rozšířeny tak, že už si to ani neuvědomujeme. Každé dialogové okno je typickou ukázkou. Po klepnutí na tlačítko nebo po změně hodnoty určitého ovladače je okamžitě provedena nějaká operace. Klepnutí nebo poklepání myši, pohyb kurzorem nebo stisk klávesy jsou také událostmi, které je každý prvek v dialogovém okně schopen rozeznat.

2.10.2 Umístění procedur událostí

Událostní procedury je třeba vždy zapsat na správné místo. Jinak se stane, že procedury, které by měly při vzniku nějaké události proběhnout nejsou spuštěny.
V okně Project může mít každý sešit tyto části

- Objekty jednotlivých listů, jejich počet odpovídá počtu pracovních listů v sešitu
- Objekty jednotlivých listů typu graf, přidáme-li do sešitu graf na samostatném listu
- Objekt ThisWorkbook – představuje celý sešit
- Standardní moduly VBA ve složce Modules. Do těchto modulů nikdy neumisťujeme událostní procedury, jejich hlavičky se ve standardních modulech ani nevyskytují, musely bychom je tam ručně dopisovat
- Moduly tříd – umožňují vytvářet nové třídy objektů
- Formuláře – dialogová okna pro komunikaci mezi kódem VBA a uživatelem

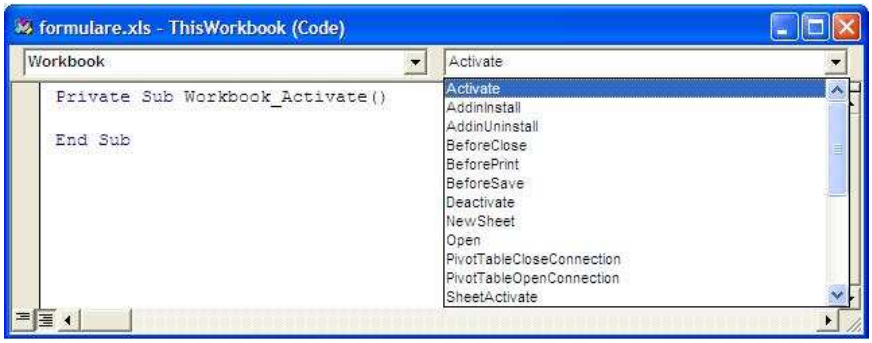


Když myší poklepeme na objekt typu list nebo objekt ThisWorkbook, otevře se okno kódu daného objektu. V jeho horní části jsou dva rozevírací seznamy. Levý seznam obsahuje řádek General a řádky Worksheet, Chart nebo Workbook. Část General slouží pro deklarování proměnných, které mají svůj rozsah platnosti na úrovni modulu, v praxi zde tedy budeme deklarovat proměnné, které budeme používat ve více událostech, probíhajících v daném modulu. Vybereme-li v levém seznamu druhý řádek (Worksheet, Chart nebo Workbook) objeví se v pravém seznamu všechny události, které můžeme použít pro daný list či sešit. Jakmile v pravém rozevíracím seznamu vybereme nějakou událost, vloží editor VBA do okna modulu hlavičku odpovídající (prázdné) procedury:

```
Private Sub Workbook_Activate ()

End Sub
```

Veškerý kód zapsaný mezi tyto dva řádky bude proveden při každém vyvolání dané události.

V pravém rozevíracím seznamu okna kódu vidíme události, které můžeme zachytávat na úrovni listu či sešitu



Některé událostní procedury obsahují v závorce za názvem procedury ještě seznam parametrů. Ty jsou vestavěny a my jejich deklarace v žádném případě nemůžeme měnit. Stejně tak nelze do seznamu parametrů zadávat parametry nové. Parametry událostí se vyskytují tam, kde jedna událost může proběhnout pro více různých objektů, nebo tam kde nám systém předává ještě dodatečné údaje k dané události. Například u událostí pohybu myši nad objektem (MouseMove) dostaneme předány souřadnice kurzoru. Při události stisknutí tlačítka myši (MouseDown) zase údaj o tom, které tlačítko bylo stisknuto. Dalším příkladem je událost SheetActivate (je na úrovni sešitu, proto ji najdeme v modulu ThisWorkbook) proběhne při aktivaci

některého listu v sešitu. Aby bylo možno určit, který list byl aktivován, je odkaz na aktivovaný list (tedy na objekt Worksheet nebo Chart, deklarovaný ovšem jako Object) předán jako parametr procedury:

```
Private Sub Workbook_SheetActivate (ByVal Sh As Object)
    MsgBox Sh.Name          ' zobrazí dialog s názvem listu
    MsgBox Sh.Rows.Count    ' dialog s počtem obsazených řádků v listu
End Sub
```

Deklarace parametru jako typ Object má v tomto případě určité nepříjemné důsledky. Musíme totiž rozlišovat, zda byl aktivován list typu graf nebo pracovní list. Kód uvedený jako příklad proběhne v pořádku, bude-li aktivovaný pracovní list. Při aktivaci listu typu graf by došlo k chybě za běhu programu ve druhém řádku, protože list typu graf neobsahuje žádné řádky a tudíž odkaz na kolekci Rows není platný.

V některých událostních procedurách se objevuje logický parametr Cancel. Například deklarace událostní procedury BeforeClose v modulu ThisWorkBook vypadá takto:

```
Private Sub Workbook_BeforeClose (Cancel As Boolean)


End Sub
```

Výchozí hodnota předaného parametru Cancel je False (nepravda). Jestliže však ve vlastním kódu procedury nastavíme tento parametr na True (pravda), nebude sešit uzavřen. Využití například při zakázání uzavření sešitu, zapomněl-li uživatel vytisknout denní přehled:

```
Private Sub Workbook_BeforeClose (Cancel As Boolean)
    dotaz =  "Vytisknul jste denní přehled změn?"
    odpoved = MsgBox (dotaz, vbYesNo, "Ukončení práce…")
    If odpoved = vbNo Then Cancel = True
End Sub
```

Procedura Workbook_BeforeClose je spuštěna po vydání příkazu k uzavření sešitu. Procedura nejdříve zobrazí okno hlášení a odpoví-li uživatel záporně klepnutím na tlačítko Ne, bude parametr nastaven na hodnotu True. Uzavření sešitu je poté zrušeno.

2.10.3 Jak zabránit výskytu událostí

V praxi se setkáváme se situacemi, kdy máme pro určitou událost napsán kód, ale tento kód může být prováděn jen někdy. Například při otevírání sešitu nebude

z nějakého důvodu žádoucí, aby byl proveden kód události Activate. Při každé další aktivaci daného sešitu však už kód proveden být musí. Obdobně se můžeme setkat s případy, kdy v kódu událostní procedury provedeme něco, čím danou událost znovu spustíme. Změníme-li například v události listu Change hodnotu buňky v tomto listu, bude událost Change generována znovu. V takovém případě ovšem hrozí nekonečné zacyklení kódu.

Existuje dvojí možné řešení k takovéto situaci. Buď zachytávání událostí zcela vypneme, nebo použijeme logickou proměnnou, která ukončí provádění kódu. První řešení má výhodu, že událost není vůbec vyvolána, ovšem nevýhodou je fakt, že neproběhnou vůbec žádné události. Logická proměnná nic nepotlačuje, jen bezprostředně ukončí provádění kódu v událostní proceduře.

Zachytávání událostí vypneme takto:

```
Debug.Print "NazevProcedury"
```

Vlastnost EnableEvents má logické hodnoty True/False, za řádkem, který by vyvolal nežádoucí generování událostí, zase zachytávání zapneme.

Ukázka druhého přístupu pomocí logické proměnné potlačí generování události Activate při otevření sešitu:

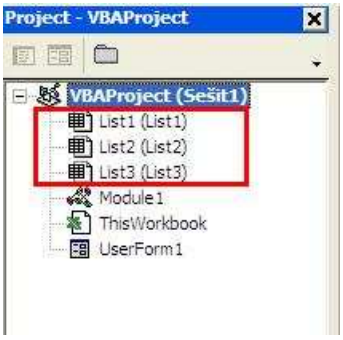
```
' proměnnou deklarujeme v modulu  ThisWorkbook – jinde ji nepotřebujeme
Private lPotlaceni As Boolean
    Private Sub Workbook_Activate ()
        If lPotlaceni Then
            ' při otevření sešitu proběhne tato část
            lPotlaceni = False
            Exit Sub
        Else
            ' zde bude kód, který proběhne při dalších aktivacích sešitu
        End If
    End Sub

Private Sub Workbook_Open ()
    lPotlaceni = True
End Sub
```

2.10.4 Události sešitu

Tyto události probíhají v rámci konkrétního (jednoho) sešitu. Jejich událostní procedury jsou uloženy v modulu kódu pro objekt ThisWorkbook. Jestliže potřebujeme zachytávat události pro libovolný otevřený sešit, musíme používat události na úrovni celé aplikace.

2.10.5 Události listů

Události objektů Worksheet. Každý list sešitu má svůj vlastní kód, který najdeme a otevřeme v okně projektu.



2.10.6 Události grafů (listu typu graf)

S událostmi grafů vložených na pracovních listech nemůžeme pracovat přímo, ale jen pomocí modulu třídy. Jinak jsou ale události obou typů grafů stejné.

2.10.7 Události aplikace Excel

Všechny události na úrovni aplikace, které můžeme pomocí modulu třídy zachytávat a využívat. Odkazy na sešity nebo listy, které danou událost vyvolaly, jsou vždy předány jako vestavěné parametry událostních procedur. [3]

3 Makroprogramování v aplikacích

3.1 Makra

Pro práci s počítačem existuje spousta programů. Tyto programy mohou být specializované a řešit pouze jednu určitou problematiku, jako jsou hry, prohlížeče internetu, nebo programy pro účetnictví. Nebo mohou být programy univerzální, které umožňují velmi široké použití. Příkladem je textový editor, nebo tabulkový kalkulátor. Každý z těchto typů má své přednosti i nedostatky. Specializované programy mohou být navrženy tak, aby maximálně usnadnily své používání. Jejich nevýhodou je většinou poměrně nízká flexibilita a dosti vysoké náklady spojené s pořízením. Univerzální programy umožňují řešit širokou škálu úkolů, jejich všestrannost bývá vykoupena větší pracností při řešení konkrétního úkolu. Zpracování specializovaných programů „na míru“ je dosti nákladné, a proto se v řadě případů musíme spokojit s některými z univerzálních programů. Velmi široce jsou používané tzv. kancelářské balíky, které zpravidla obsahují textový editor, tabulkový kalkulátor a prezentační program.

Univerzální programy prošly poměrně dlouhým vývojem a dnes již disponují širokou paletou nástrojů. Při obsluze programu z klávesnice nebo myši však musíme jednotlivé příkazy vždy znovu manuálně zadávat. Opakování většího množství příkazů je proto únavné a časově náročné. I u univerzálních programů však existuje možnost jejich modifikace tak, aby se přiblížily specializovaným programům. Touto možností je použití maker. Makra představují určitou nadstavbu nebo doplnění základního programu. Samotné slovo makro vzniklo jako zkratka slova makroinstrukce. Počítač pracuje na základě zadaných instrukcí. Instrukcí může být například stisk klávesy nebo volba určitého příkazu z panelu nabídek. Instrukcí může být také příkaz napsaný v programovacím jazyku. A soubor několika takových instrukcí tvoří makroinstrukci, tedy makro.

Makra mohou vznikat dvojím způsobem. Některé programy jsou vybaveny záznamníkem instrukcí. Při použití záznamníku postačuje spustit záznam a následně vykonat určitou činnost, například zadat údaje do buňky a změnit její formát. Po ukončení záznamu máme k dispozici nově vytvořené makro se záznamem provedených činností. Při každém spuštění makra se zaznamenané činnosti vždy znovu provedou. Dosažený výsledek závisí na dokonalosti záznamníku a jeho schopnosti zachytit všechny provedené činnosti.

Druhou možností je vytvoření makra ve vhodném programovacím jazyku. Vytvoření makra tímto způsobem je sice náročnější, ale výsledné makro může zajišťovat i činnosti, které jsou pro uživatele nedostupné. Příkladem může být použití dialogových oken pro obousměrnou komunikaci s počítačem nebo nastavení různé odezvy programu v závislosti na aktuální situaci. Makro může totiž reagovat různým způsobem, například podle toho, jaká je pozice aktivní buňky v listu. Oba způsoby lze i kombinovat a zaznamenané makro následně upravit a doplnit.

Každé makro má podobu psaného textu – procedury. Text se obecně skládá z několika skupin výrazů. Jednak jsou to výrazy ze slovníku programovacího jazyka, Basicu. Jedná se o výrazy, kterým počítač takříkajíc rozumí. Jsou to vybraná anglická slova nebo zkratky, které označují začátek a konec procedury, všechny příkazy, funkce a některé vybrané hodnoty. Slova ze slovníku Basic mají přesně definovanou funkci a užití, a proto je označujeme jako vyhrazená nebo klíčová slova. Do skupiny jednoznačně definovaných výrazů patří i operátory, například + nebo - , závorky a další pomocné znaky, jako je například tečka nebo znak dolaru. Druhou skupinu výrazů představují názvy, které sami vytváříme. Názvy vytváříme zejména k označování proměnných v proceduře. Třetí skupina zahrnuje data, tedy konkrétní číselné nebo textové hodnoty. Proměnné mají v makrech nezastupitelné místo, protože umožňují uchovávat informace. Proměnnou si můžeme představit například jako lístek papíru, nadepsaný číslem telefonu. Označení lístku by představovalo název proměnné a její hodnotou by bylo konkrétní zapsané číslo. [3]

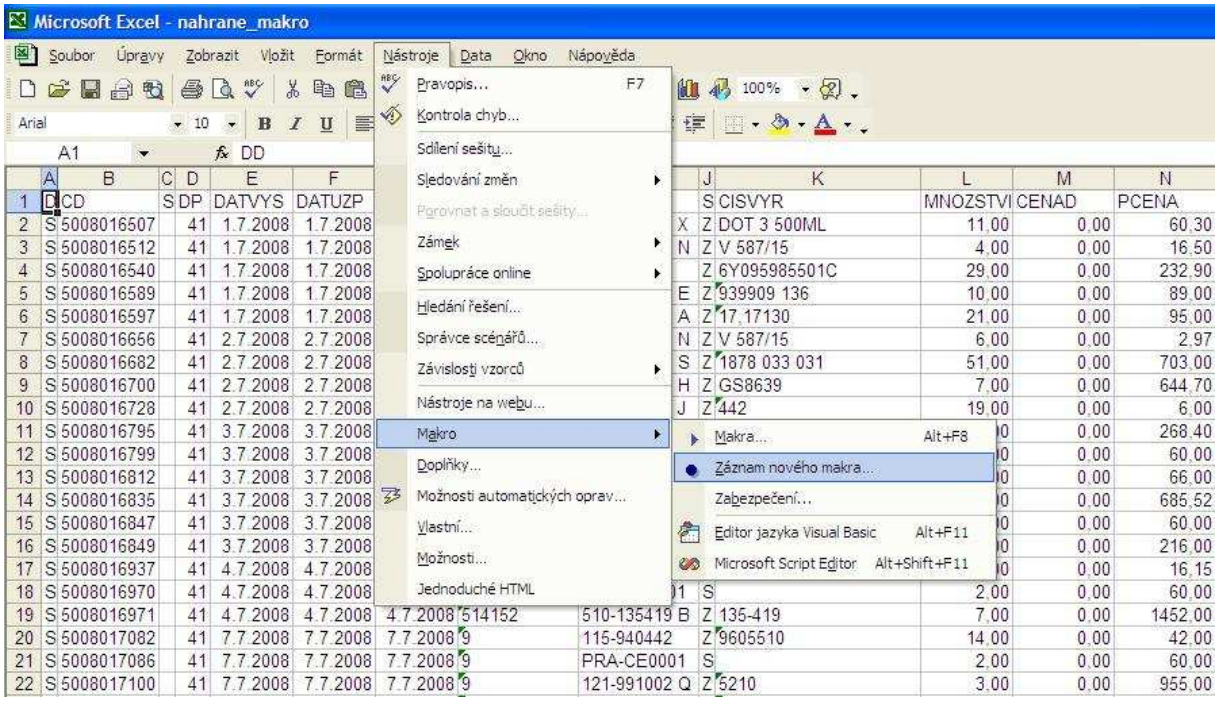
3.2 Nahrávání maker pomocí záznamníku instrukcí

Jedná se o nejrychlejší způsob vytvoření makra. Nahrávání je nezastupitelné během všech fází poznávání jednotlivých objektů programu Excel. Objekty mají velké množství různých vlastností, které jsou sice popsány v nápovědě, nicméně nahráním makra zjistíme nejrychleji, které z těchto vlastností skutečně musíme použít.

Nevýhodou použití záznamníku instrukcí je, že jednotlivé objekty (buňky, listy, grafy) vybírá. V nahraném makru proto vždy najdeme příkaz pro výběr – Select – pokud si ovšem požadovaný objekt nevybereme sami ještě před spuštěním záznamníku. Výběr objektů však ve skutečnosti není nutný a prodlužuje dobu provádění makra! Výběr objektů má ještě následující důsledky: buď si příslušný objekt (buňky, graf) označíme ještě před začátkem nahrávání makra – v takovém případě pak bude makro vždy zpracovávat aktuální výběr – nebo budeme jednotlivé objekty vybírat až při

vlastním nahrávání – pak ovšem bude nahrané makro pracovat vždy právě jen s těmito objekty. Nahrané makro vždy zaznamená jen přesně daný sled událostí, které jsme prováděli při jeho nahrávání. Není možné nahrát například výběr akce na základě nějaké podmínky, není možné žádnou akci na základě nějaké podmínky vynechat. Způsob jakým záznamník nahrává prováděné operace, se občas dá těžko pochopit. Kód, který bychom ručně napsali mnohem efektivněji, je někdy správně strukturován, někdy ne. Nahraný kód navíc ve výchozím nastavení pracuje s absolutními adresami buněk, což také nemusí být vždy to pravé.

Výhody i nevýhody nahraného makra se celkem dobře projeví na krátkém makru, které se dá často použít. Často se totiž vyskytuje situace, že soubor, který pomocí programu Excel otevřeme má v prvním řádku názvy sloupců, které je vhodné pro tisk nějak graficky zvýraznit. Po otevření takového souboru se proto často opakuje postup, kdy vybereme první řádek, buňkám v něm změníme řez písma na tučný a zarovnáme na střed buňky. Poté je třeba, aby všechny buňky měly viditelnou hodnotu (pokud je obsah buňky delší, než je nastavená délka buňky, zobrazí se pouze ###### přes celou buňku), to znamená přizpůsobit šířku sloupce a na závěr, aby byla hlavička souboru stále viditelná, musíme ukotvit první řádek. Všechny tyto úkony se dají dobře zaznamenat pomocí záznamníku maker. Záznamník najdeme v menu a spustíme.



Po klepnutí na řádek Záznam nového makra se objeví dialogové okno Záznam makra.

V tomto dialogu je nutné zapsat alespoň název makra. Výchozí názvy, které Excel pro nahrávaná makra nabízí (Makro1, Makro2 atd.) jsou v praxi nepoužitelné, protože neříkají nic o tom k čemu makro slouží. Proto do textového pole Název makra zapíšeme nějaký vlastní název například Prvni_tucny_radek.

I při vymýšlení názvu makra ale musíme dodržet určitá pravidla, jinak nás Excel upozorní na chybu a nedovolí nám pokračovat. Prvním znakem musí být vždy písmeno. Na dalších místech mohou být písmena, číslice nebo podtržítka. Název makra nesmí obsahovat mezery, jiné interpunkční znaky (například tečky, čárky, pomlčky) a některé další znaky (třeba #, %, @, &) Chceme-li název složený z více slov, oddělíme je pomocí znaku podtržítka (Prvni_tucny_radek), nebo napíšeme první písmeno každého slova velké (PrvniTucnyRadek). Maximální délka názvu makra je 255 znaků.

Název makra by měl odrážet jeho účel. Spodní textové pole Popis obsahuje standardní komentář, který bude zapsán na začátek makra. Komentář můžeme ponechat tak, jak je, nebo ho přepsat vlastním textem. Komentář je možné libovolně upravit i později. Excel XP toto pole vyplňuje aktuálním datumem a jménem uživatele.

Další pole v okně Záznam makra je Klávesová zkratka. V základním zobrazení nabízí pouze variantu CTRL+písmeno. Je ale možnost zadat i zkratku CTRL-SHIFT+písmeno, stačí klepnout do okna, kde se písmeno vypisuje a zadat tam velké písmeno (se stisknutou klávesou SHIFT).

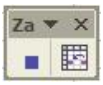
Poslední a velmi důležitá volba v tomto dialogovém okně se týká místa, kam bude makro uloženo. Jde vždy o nějaký sešit, jinak se makro uložit nedá. Excel implicitně nabízí uložení do aktivního sešitu („Tento sešit“) Vzhledem k tomu, že makro je k dispozici jen v případě, kdy je tento sešit otevřen, hodí se tato volba jen pro makra používaná v rámci daného sešitu. V případě, že není žádný sešit otevřen, není tato možnost dostupná. Druhým možným místem, kam se dá makro zapsat je Osobní

sešit maker. Uživatel se s ním při běžné práci nesetká – je určen výhradně pro ukládání maker (i když v případě zájmu můžeme ukládat i data do jeho listu). Excel tento sešit vytvoří až v okamžiku, kdy do něj chceme nějaké makro uložit.

Osobní sešit maker se jmenuje Personal.xls a je uložen ve složce XLStart, jejíž umístění je různé v závislosti na verzi Windows (ve Windows XP Professional CZ to je složka C:\Documnets and Settings\"jméno uživatele"\Data aplikací\Microsoft\Excel\XLStart). Sešity umístěné ve složce XLStart se otevírají automaticky při spuštění programu Excel, takže makra z tohoto sešitu jsou vždy k dispozici. Osobní sešit maker je od svého vytvoření označen jako skrytý, takže jeho obsah (jeden prázdný list) zobrazíme jedině příkazem nabídky Okno→Zobrazit. V okně Visual Basic se dají jeho makra zobrazit vždy. Kromě složky XLStart je možno definovat i další složku se stejným účelem (příkaz Nástroje→Možnosti, karta Obecné pole Umístění souborů otevřených při spuštění). Sešity uložené ve složce, kterou zadáme se budou rovněž automaticky otevírat při spuštění programu Excel.

Poslední možností pro uložení makra je Nový sešit – v tomto případě dojde k vytvoření nového sešitu, a makro bude do něj uloženo. Základní rozdíl v umístění maker tedy spočívá v tom, že makro používané jen v rámci jednoho sešitu ukládáme přímo do něj, zatímco makro obecně zaměřené, využitelné ve více různých souborech ukládáme do Osobního sešitu maker.

Po vyplnění všech možností a klepnutí na tlačítko OK v dialogovém okně Záznam makra se objeví nový panel nástrojů.

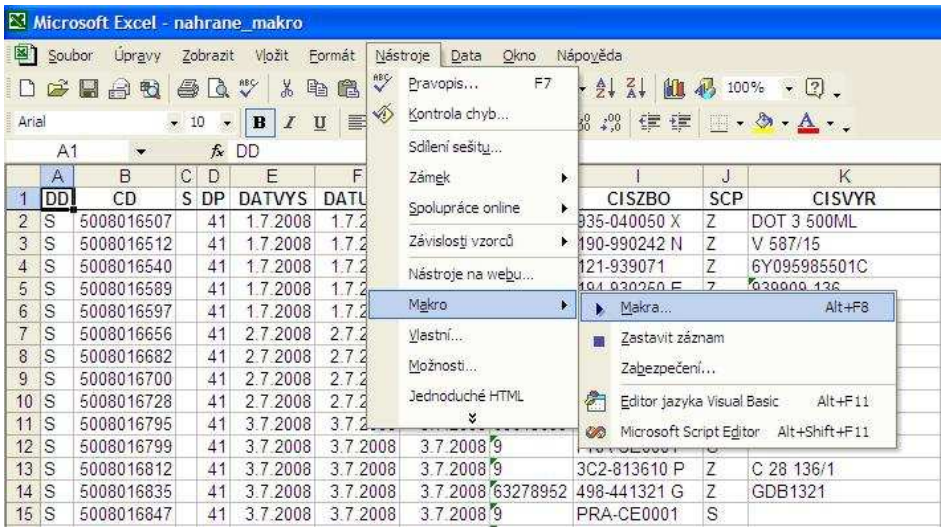


Tento panel obsahuje pouze dvě tlačítka. Tím prvním – Zastavit záznam, s ikonou modrého čtverečku – se celé nahrávání makra ukončí. Druhé tlačítko – Relativní odkaz – funguje jako přepínač. Je-li vypnuto, nahrává záznamník absolutní odkazy, po jeho zapnutí jsou odkazy nahrávány relativně.

Od okamžiku stisku tlačítka OK v dialogovém okně Záznam makra Excel nahrává všechny naše akce, včetně těch, které jsme udělat nechtěli! Je proto potřeba postupovat vždy s rozmyslem. Na druhé straně určitou ochranu znamená fakt, že Excel nenahraje akce zbytečné. Například když vybereme oblast buněk a ihned poté vybereme jinou oblast, bude záznam prvního výběru vypuštěn, protože jsme buňky v této oblasti nijak neměnili. Při nahrávání maker můžeme používat myš. Myší můžeme buňky

vybírat, přesouvat apod. Nahrávání makra, ale nemůžeme přerušit. Pokud tedy před začátkem nahrávání zapomeneme provést nějakou nezbytnou operaci, musíme makro nahrát znovu.

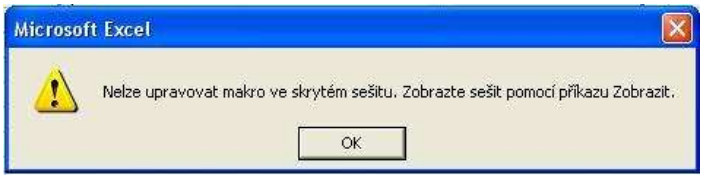
Podívejme se nyní na příklad nahraného makra. Po vyplnění názvu makra a klávesové zkratky jsem zmáčkl tlačítko OK. Poté jsem postupně v sešitu udělal potřebné změny (první řádek tučným písmem a buňky zarovnané na střed, přizpůsobení šířky všech sloupců a ukotvení prvního řádku) a ukončil nahrávání makra. Na kód takto nahraného makra se můžeme podívat v editoru VBA. Tento editor spustíme přes menu:



Klepnutím na řádek Makra… v tomto menu se objeví další tabulka, kterou je možno vyvolat také stiskem klávesové zkratky ALT+F8.

Z této tabulky můžeme buď klepnutím na tlačítko Spustit dané makro spustit nebo klepnutím na tlačítko Upravit můžeme otevřít editor VBA. Editor VBA je možno přímo vyvolat také stiskem klávesové zkratky ALT+F11. Pokud však editor jazyka VBA

vyvoláme přes toto menu stisknutím tlačítka Upravit, tak nám editor rovnou zobrazí požadované makro. Pokud se ale tlačítkem Upravit pokusíme přistoupit k makru, které je uloženo v osobním sešitu maker (Personal.xls), objeví se chybové hlášení:



Osobní sešit je totiž skrytý a z nějakého důvodu to programu Excel v okně Makra vadí. Přímo v editoru se přitom k těmto makrům dostaneme bez problémů. Nejdříve je nutné příkazem Okno→Zobrazit osobní sešit odkrýt, a pak už tlačítko Upravit bude použitelné. Máme-li otevřeno více sešitů najednou a makra nejsou uložena pouze v jednom z nich, je lépe začínat editaci z okna Makro, kde si najdeme název makra a klepneme na tlačítko Upravit. V editoru VBA totiž není možné vyhledávat název makra ve více sešitech najednou, takže musíme vědět, ve kterém sešitu makro je.

V editoru VBA se můžeme podívat, co záznamník maker zaznamenal.

```
Sub prvni_tucny_radek()
'
' Prvni_tucny_radek Makro
' Makro zaznamenané 30.1.2008, Jan Macháček
'
' Klávesová zkratka: Ctrl+Shift+B
'
    Rows("1:1").Select
    Selection.Font.Bold = True
    With Selection
      .HorizontalAlignment = xlCenter
      .VerticalAlignment = xlBottom
      .WrapText = False
      .Orientation = 0
      .AddIndent = False
      .IndentLevel = 0
      .ShrinkToFit = False
      .ReadingOrder = xlContext
```

```
        .MergeCells = False
    End With
    Cells.Select
    Cells.EntireColumn.AutoFit
    Rows("2:2").Select
    ActiveWindow.FreezePanes = True
    Range("A1").Select
End Sub
```

První řádek makra obsahuje klíčové slovo Sub, které celé makro uvádí a dále název makra. Na dalších řádcích jsou poznámky, které tam záznamník maker vypíše, tyto poznámky jsou z popisu makra, které jsme mohli změnit při jeho začátku v dialogovém okně Záznam makra a klávesová zkratka. Tyto řádky jsou uvedené apostrofem (') a všechen text uvedenýma tímto znakem je chápán jako poznámka a není VBA prováděn.

Takto vytvořené makro jistě bude fungovat pokaždé, když ho budeme chtít spustit, má však několik nevýhod. Užívá několikrát výrazu pro výběr Select. Tento výraz je ve většině případů zbytečný a zdržuje. Místo výběru oblasti můžeme s oblastí přímo pracovat. A tím zkrátit a zrychlit celý kód. Místo prvních dvou řádků

```
Rows("1:1").Select
Selection.Font.Bold = True
```

se tak dá napsat pouze řádek jeden

```
Rows("1:1").Font.Bold = True
```

Dále záznamník zapsal kód, ve kterém se formátuje celý blok najednou pomocí příkazu With Selection ….. End Selection. To je sice dobrý příkaz, pomocí kterého není potřeba se znova a znova odkazovat na jednu danou oblast, ale s oblastí buněk pracovat. Bohužel Excel při jedné úpravě formátování uvádí i další úpravy, které jen zopakují již existující formát. My jsme však chtěli pouze zarovnat text na střed buněk. Ne ještě text zarovnávat na spodek buněk, zakázat zalamování textu a další úpravy, které byly navíc doplněny. Pak bychom ani nepotřebovali opět vybírat buňky a ani používat příkaz With Selection…. End Selection. Stačil by nám pouze řádek

```
Rows("1:1").HorizontalAlignment = xlCenter
```

Další řádek za příkazem End Selection je úplně zbytečný. Sice příkaz na tomto řádku vybere všechny buňky, ale my s nimi opět hned můžeme pracovat, tak jak to udělá hned

následující řádek. Poslední čtyři řádky záznamu jsou použitelné tak, jak jsou. Opět se v nich dvakrát objevuje příkaz Select. Tentokrát je však jeho výskyt opodstatněný. Poprvé je potřeba oblast buněk opravdu vybrat, aby následující příkaz věděl, které buňky je třeba ukotvit. A předposlední řádek je v makru proto, aby po skončení makra byla opět vybrána pouze buňka A1. Poslední řádek je opět klíčové slovo oznamující konec makra. Po všech úpravách je teda možné makro přepsat do této podoby:

```
Sub prvni_tucny_radek_uprava()
    Rows("1:1").Font.Bold = True
    Rows("1:1").HorizontalAlignment = xlCenter
    Cells.EntireColumn.AutoFit
    Rows("2:2").Select
    ActiveWindow.FreezePanes = True
    Range("A1").Select
End Sub
```

Tento kratší zápis bude vykonávat stejné úkoly jako původní, ale mnohem efektivněji. Rychlost práce makra se při takto krátkém programu sice neprojeví, ale při mnohem delším programu by se jasně ukázala rychlost ovlivněná přebytečnými příkazy.

3.3 Psaní maker v editoru VBA

Nahrávání maker nám pomůže v začátcích, ale složitější procedury je nutné psát ručně buď celé nebo upravit nahrané makro. Obecně se dá říci, že standardní modul může obsahovat jednak procedury a funkce a dále deklarace proměnných. Deklarační část je na samém počátku modulu, procedury a funkce jsou umístěny za ní.

3.3.1 Procedury a funkce

Podprogram - je sled příkazů vykonávaný jako celek a tvořící uzavřenou jednotku. Běh spuštěného podprogramu je možné přerušit voláním jiného, po jehož vykonání se řízení vrací zpět do nedokončeného volajícího podprogramu. Běh může být rovněž přerušen v případě vzniku chyby, zásahem ze strany uživatele nebo použitím příkazu Stop.

Procedura – v jazyku se označuje klíčovým slovem Sub – jedná se o podprogram, který provede sérii příkazů (nebo jeden příkaz) a skončí. Tento sled

příkazů je mezi řádky Sub a End Sub. Makro je procedura bez parametrů, kterou můžeme nahrát, a kterou lze spustit z libovolného místa programu Excel. Každá procedura nemusí být makrem, ale každé makro je procedurou. Obecně můžeme říct, že makra jsou všechny procedury, které jsou vidět v dialogovém okně Makro.

Funkce – občas označované i jako funkční procedury, anglicky Function – je to podprogram, který kromě toho, že provede nějaké příkazy, také vrací určitou hodnotu. Tuto vrácenou hodnotu můžeme uložit do paměti počítače (tedy do proměnné) nebo ji zpracovat jiným způsobem, případně ji ignorovat. Funkce se v programech chovají stejně jako matematické funkce a stejně tak se i zapisují – za jejich názvem se píší závorky. Návratová hodnota funkce může obsahovat buď výsledek výpočtu nebo informuje o úspěšném či neúspěšném výsledku průběhu funkce. Příkazy funkce se píší mezi řádky Function a End Function. Funkce se z dialogového okna Makro spustit nedá, stejně tak procedura s parametry.

Makra nahráváme, vlastní procedury či funkce píšeme. Do jednotlivých modulů – modul musí být vždy aktivní – se procedury či funkce vkládají příkazem Insert→ Procedure. Tento příkaz zobrazí dialogové okno

V dialogovém okně Add Procedure je důležité v horním textovém poli zapsat název a ve středním rámečku zvolíme zda půjde o proceduru nebo funkci. Po potvrzení tlačítkem OK se do modulu vloží prázdná hlavička procedury či funkce, která bude mít tento tvar:

```
Public Sub název()
End Sub
```

nebo

```
Public Function název ()
```

```
End Function
```

Klíčové slovo Public určuje rozsah platnosti čili viditelnost procedury (funkce). Za názvem jsou vždy závorky, ty k názvu psát nemusíme, editor je doplní sám.

3.3.2 Jednotlivé části podprogramů

U nahraných maker to je jednoduché: za řádkem s názvem makra následuje několikařádkový komentář a pak už jen samotné příkazy. V proceduře či funkci se však mohou nacházet i další části. První z nich je část pro deklaraci proměnných. Některé proměnné se deklarují uvnitř procedur a některé na začátku modulu. Kde se které deklarují záleží na jejich použitelnosti, čili rozsahu platnosti a viditelnosti.

Jednodušší i složitější procedury by měly počítat také s možností nejrůznějších chyb, ke kterým během provádění může dojít. Nahrané makro například může odkazovat na buňku ležící o čtyři řádky nad buňkou aktuální. Ale co když toto makro spustíme v okamžiku, kdy aktuální buňkou bude například buňka C2? Čtyři řádky nad buňkou C2 už samozřejmě žádná buňka není, takže makro v tomto okamžiku ohlásí chybu. Z těchto důvodů by v procedurách vždy měly být tzv. chybové rutiny, tedy kód potřebný pro ošetření chyb, k nimž za běhu programu dojde.

Psané programy také mohou přebírat parametry, které modifikují jejich činnost. Parametry jsou uvedeny v hlavičce procedury, uvnitř závorek hned za jeho názvem. U nahraných maker se parametry nikdy nevyskytují, stejně jako proměnné.

```
Sub Priklad_parametru(Parametr1 as Range, Parametr2 As Integer)
    ' příklad hlavičky procedury se 2 parametry – první má hodnotu rozsahu, druhý hodnotu celého čísla
End Sub
```

Funkce se od procedury liší tím, že umí vrátit nějakou hodnotu. Při psaní musíme určit, co bude funkce vracet. V jazyku VBA se to udělá tak, že v kódu napíšeme příkaz ve tvaru:

```
Nazev_funkce = hodnota
```

Uvedeme se krátký příklad, který toto hodně ozřejmí. Použiji funkci pro výpočet rozdílu druhých mocnin dvou čísel:

```
Function RozdilMocnin(A as Double, B as Double)
    RozdilMocnin = A*A-B*B
End Function
```

Parametry jsou pojmenovány A a B. Při volání funkce v kódu pak budeme funkci předávat skutečné parametry, například takto:

```
X = RozdilMocnin(21.4, 18)          ' předáváme explicitní hodnoty
Y = RozdilMocnin(Vetsi, Mensi)      ' předáváme proměnné Vetsi a Mensi
```

Deklarační část a část pro ošetření chyb se pochopitelně ve vlastní funkci vyskytovat může také.

3.3.3 Příkazy a jejich zápis

Konvence říká, že na jeden řádek v kódu píšeme jeden příkaz (vestavěný příkaz jazyka, volání vlastní procedury, volání funkce a přiřazení návratové hodnoty apod.). Je to nejpřehlednější způsob a vylučuje některé chyby vzniklé zápisem.
Při zapnuté konsole správnosti zápisu kontroluje VBA po napsání každý řádek kódu a pokud je bez chyby, přeformátuje ho podle svých vnitřních konvencí (vloží mezery mezi operátory, malá písmeny na začátku příkazů převede na velká a podobně). Přitom se mohou také některé části kódu barevně odlišit. Toto rozlišení se nastavuje v okně možností editoru. Zde můžeme rovněž zapnout či vypnout zmíněnou automatickou kontrolu správnosti zápisu.

Je-li příkaz příliš dlouhý – obvykle tehdy, když má více parametrů – lze jej rozdělit na více řádků (není to nutné, dělá se to jen kvůli tomu, abychom ho celý viděli v okně modulu). Na konci každého řádku (kromě posledního) je nutné zapsat znak pokračování řádku, což je mezera následovaná podtržítkem.

Můžeme rovněž psát na jeden řádek více příkazů – přitom je oddělujeme dvojtečkou – ale je lepší se tomu vyhýbat, protože zápis pak ztrácí na přehlednosti.

3.3.4 Proměnné

Pokud použijeme nějakou funkci, pak výslednou hodnotu musíme někam uložit, pokud s ní chceme dále pracovat. Uložená výsledná hodnota se podobně jako vlastní běžící program nachází v operační paměti. Každé takové dočasné hodnotě, ať už číselné, textové či jiné, která je uložena na zvláštním místě v operační paměti, se říká proměnná. Jako proměnné můžeme v podstatě chápat i vlastnosti objektů, protože i ty mají nějakou hodnotu. Proměnná ovšem není na objektu nijak závislá.

Přiřazení se provádí operátorem = , u proměnných objektového typu vždy příkazem Set.

```
X = 15
```

```
Set ws = ActiveWorkbook.WorkSheets(2)
```

3.3.5 Konstanty

Konstanta je určitá hodnota, která se během programu nemění. Díky nim je možné zkrátit či zpřehlednit zápis kódu. Jestliže budeme například ve více makrech používat určitý koeficient, pak místo opakovaného psaní stejné hodnoty (čísla) je lepší toto číslo deklarovat jako konstantu, a v kódu pak psát její název. Zásadní výhodou konstant je i fakt, že v případě změny její hodnoty stačí zapsat nový údaj na jediném místě, není nutné vyhledávat a přepisovat všechny výskyty.

VBA rozlišuje konstanty podle místa jejich deklarace:

- Vestavěné konstanty – jsou deklarovány v objektových knihovnách samotného jazyka a také v objektových knihovnách jednotlivých ovladačů nebo objektů jiných aplikací. Vestavěné konstanty tedy můžeme okamžitě používat. Setkáme se s nimi u velkého množství různých metod i vlastností. Jejich výhodou jsou dobře navržené názvy, podle kterých většinou poznáme, co daná konstanta představuje.

```
Otazka = MsgBox ("Chcete opravdu skončit?", vbQuestion+vbYesNo) Then
    ' ukázka vestavěných konstant u funkce MsgBox
```

- Symbolické (uživatelsky definované) konstanty – vytváříme si je sami podle svých potřeb příkazem Const.

3.3.6 Názvy objektů, proměnných a konstant

Jazyk VBA se skládá z řady klíčových slov, jejichž názvy většinou nemůžeme použít k označení čehokoli jiného. A i kdybychom mohli použít nějaké klíčové slovo třeba pro název proměnné, není dobré to dělat. Není to k ničemu, jen se tím při psaní kódu zvyšuje možnost vzniku chyb. Při tvorbě názvů proměnných, konstant, procedur a funkcí musíme dodržovat ještě další pravidla.

- názvy musí začínat písmenem (nikoli číslicí!)
- názvy nesmí obsahovat tyto znaky: mezera, &, #, @, $, %, !, čárka, tečka
- názvy nesmí být delší než 255 znaků (u objektů 40 znaků). V praxi je lépe používat kratších názvů do 32 znaků, kód se jinak značně nepřehlední
- nemůžeme použít dva stejné názvy v rámci stejného rozsahu platnosti proměnné či objektu

VBA v názvech nerozlišuje mezi velkými a malými písmeny. Názvy nazev, NAZEV a Nazev jsou proto názvy stejné proměnné.

3.3.7 Výrazy

Pojem výraz označuje v programovacím jazyku jakoukoli kombinaci klíčových slov, proměnných, konstant a operátorů, jejímž výsledkem je logická hodnota, text, číslo nebo objekt. Výrazy můžeme používat pro testování dat, manipulaci s proměnnými nebo k provádění výpočtů. Je potřeba dát pozor na rozdíl mezi příkazy a výrazy. Příkazy provádí nějakou akci, výsledkem výrazu je hodnota.

3.3.8 Operátory

Operátory určují druh operace, která bude provedena na částech výrazu okolo daného parametru. Ve VBA pracujeme se čtyřmi základními druhy operátorů: aritmetické, srovnávací, spojovací a logické.

Aritmetické operátory provádějí matematické operace. Patří sem operátory sčítání, násobení (+), dělení (/), celočíselného dělení (\), odčítání (-), umocňování (^) a operátor Mod – výpočet zbytku dělení.

Srovnávací operátory – VBA podporuje stejné operátory, jaké se dají používat ve vzorcích programu Excel: rovná se (=), větší než (>), menší než (<), větší nebo rovno (>=), menší nebo rovno (<=) a nerovná se (<>). Operátor Is se používá pro porovnání dvou objektových proměnných a v příkazu Select Case.

Spojovací operátor & se používá pro spojování textových řetězců.

Logické operátory provádí logické operace. Do této skupiny patří operátory negace výrazu (Not), logický součin (And), logický součet (Or), neekvivalence (Xor), ekvivalence (Eqv) a implikace (Imp).

Pořadí vyhodnocení jednotlivých operátorů je dáno jejich příslušností do skupiny operátorů a jejich prioritou. Ta se dá v případě potřeby změnit uzavřením částí výrazu do závorek. Aritmetické operátory mají přednost před srovnávacími a ty zase před logickými. Srovnávací operátory jsou si navzájem rovny, tzn. jejich vyhodnocování probíhá vždy zleva doprava. Aritmetické operátory se vyhodnocují dle priority – pořadí je určeno: umocňování znaménko mínus, násobení a dělení, celočíselné dělení, zbytek po dělení, sčítání a odečítání, spojení textových řetězců. Operátor & - spojení textových řetězců – samozřejmě nepatří do skupiny aritmetických operátorů, ale v pořadí vyhodnocování se nachází právě mezi aritmetickými a srovnávacímu operátory. Logické operátory mají pořadí negace, logický součin, logický součet, neekvivalence, ekvivalence a implikace.

3.3.9 Datové typy proměnných a konstant

U proměnných rozeznáváme čtyři základní charakteristiky. Název, datový typ (vyjadřuje, jaký typ hodnoty můžeme do proměnné umístit – text, číslo, datum, apod. – a také kolik místa v paměti proměnná zabere), platnost (určuje, které části aplikace mohou s proměnnou pracovat, někdy se používá též pojem dostupnost) a životnost (specifikuje dobu existence proměnné – většinou souvisí s platností)

Datový typ stanoví, co všechno můžeme do proměnné uložit, a také způsob, jakým budou data uložena v paměti – tedy jako celá čísla, reálná čísla, textové řetězce.. Jestliže datový typ nezadáme, použije VBA výchozí typ Variant, který povoluje zadání všech možných hodnot. U konstant toto pravidlo neplatí – neurčíme-li při deklaraci konstanty datový typ, použije VBA ten, jenž nejlépe odpovídá hodnotě, kterou konstanta představuje.

Informaci o datovém typu jakékoli proměnné (s výjimkou uživatelsky definovaných typů) vrací funkce TypeName, která vypíše název datového typu jako text (řetězec). U základního typu Variant lze navíc konkrétní typ obsahu dané proměnné specifikovat pomocí speciální funkce VarType, která vrací číslo.

```
Pokus = “Nazdar“
MsgBox VarType (pokus)
```

Pokud by proměnná pokus z této ukázky neměla určen datový typ (takže by šlo o typ Variant), vrátí funkce VarType číslo 8, protože v proměnné je uložen textový řetězec. V praxi se funkce VarType používá k ověření typu hodnoty v proměnné typu Variant, abychom například měli jistotu, že v proměnné je číslo. Návratové hodnoty funkce VarType jsou uvedeny v nápovědě. Kromě toho existují další funkce, pomocí nichž můžeme ověřit, zda daná proměnná obsahuje určitý typ hodnoty, nebo zda se dá na určitý typ hodnoty konvertovat. Mezi tyto funkce patří IsNumeric (test na číselnou hodnotu), IsDate (test na datum), IsObject (testujeme, zda proměnná obsahuje odkaz na objekt), IsError (test chybové hodnoty). Datové typy se dají rozdělit do několika skupin – čísla, text, logické hodnoty (boolean), datum a objekt.

- **Byte** má velikost 1 bajt. Je to číslo 0 až 255, požívá se pro ukládání binárních dat.
- **Integer** má velikost 2 bajty. Jde o čísla v rozsahu -32 768 až 32767. Základní typ pro práci s celými čísly.
- **Long** má velikost 4 bajty. Ukládají se do něj čísla v rozsahu -2 147 483 648 až 2 147 483 647.

- **Single** má velikost 4 bajty. Jde o desetinná čísla s přesností na 6 desetinných míst, rozsah možných hodnot je uveden v nápovědě k programu.
- **Double** má velikost 8 bajtů. Desetinná čísla s dvojnásobnou přesností při výpočtu.
- **Currency** má velikost 8 bajtů. Číslo s pevným počtem 4 desetinných míst. Doporučeno pro operace s peněžními hodnotami. Výpočty s tímto typem probíhají rychleji než u typu Single či Double.
- **Decimal** má velikost 14 bajtů. Tento typ není možné deklarovat, proměnnou tohoto typu lze získat jen převodem proměnné typu Variant pomocí funkce CDec. Typ Decimal je určen pro čísla s extrémním počtem desetinných míst (až 28).
- **String** má různou velikost pro řetězce pevné a proměnné délky. Maximální rozsah má až $2*10^9$ znaků pro řetězce s proměnnou délkou nebo 64 kilobajtů pro řetězce s pevnou délkou.
- **Boolean** má velikost 2 bajty. Používá se pro logickou hodnotu True nebo False.
- **Date** má velikost 8 bajtů. Jedná se o datum v rozsahu od 1.ledna 100 až do 31. prosince 9999.
- **Object** má velikost 4 bajty. Jde o jakýkoli odkaz na objekt.
- **Variant** má různou velikost, nejméně 16 bajtů. Je to základní typ, může obsahovat speciální hodnoty Null, Error, jakoukoli numerickou hodnotu, jakýkoli text, odkaz na libovolný objekt nebo pole proměnných.

Dále mohou existovat ještě uživatelsky definované typy proměnných. Obecně platí, že nejlepší je používat takový datový typ, který zabere v paměti co nejméně místa, ale přitom dokáže obsáhnout všechny možné hodnoty, které mu můžeme přiřadit (pro desetinná čísla tedy nelze použít celočíselný typ Integer apod.). V praxi ovšem hraje roli i počet bajtů, které procesor umí zpracovat v jedné instrukci.

3.3.10 Rozsah platnosti a životnost proměnných a konstant

Každá proměnná a konstanta má svůj rozsah platnosti určující, které podprogramy jsou schopny s touto proměnnou pracovat.

Lokální proměnné mohou být použity pouze v tom podprogramu, v němž se vyskytují. Žádný jiný podprogram s nimi nemůže pracovat a jejich hodnoty tedy jinde

nemůžeme získat. Deklarace lokálních proměnných není povinná a používá se pro ni příkaz Dim, případně Static.

Modulové proměnné jsou dostupné v rámci celého modulu, a lze je proto použít třeba k uložení hodnoty, která má být dostupná někdy později v jiné proceduře. Modulové proměnné musí být deklarovány, a to v deklarační části modulu příkazem Dim nebo Private.

Veřejné (globální) proměnné mohou být použity ve všech modulech, neboli v celém (jednom) sešitu. Veřejné proměnné musí být deklarovány, a to v deklarační části modulu příkazem Public.

V horní části modulu jsou dva rozevírací seznamy. Když v levé části vybereme položku (General), v pravém seznamu najdeme položku (Declarations). To je právě ta deklarační část modulu. Deklarace veřejných proměnných musí být umístěny jen ve standardních modulech, modulech pro jednotlivé listy nebo v modulu This_Workbook, nikoli v modulech formulářů.

Rozsah platnosti se týká i vlastních procedur a funkcí! Ty jsou implicitně chápány jako veřejné, je tedy možné je volat procedurou, která je v jiném modulu (klíčové slovo Public není nutné uvádět, VBA ho předpokládá. Sub a Public Sub mají proto stejný význam. Pokud chceme z nějakých důvodů vytvořit proceduru nebo funkci, kterou lze spouštět jen v rámci modulu, kde je umístěna, doplníme do hlavičky klíčové slovo Private. „Privátní“ procedury se neobjeví v dialogovém okně Makro, je tedy takto možné i zabránit uživatelům v jejich nežádoucím spuštění. Veřejné funkce uložené ve standardních modulech se v uživatelském rozhraní programu Excel objeví v dialogovém okně Vložit funkci, a to v kategorii Vlastní.

S rozsahem proměnné úzce souvisí i její životnost, tedy délka existence proměnné v paměti. Lokální proměnné mají životnost omezenou na dobu provádění svého podprogramu. Naopak modulové a veřejné proměnné existují po celou dobu, kdy je daný sešit otevřen. Je-li třeba, aby ani lokální proměnná nezanikla po skončení podprogramu, ve kterém je použita, musí se uvnitř procedury deklarovat příkazem Static, případně můžeme příkaz Static přidat do hlavičky procedury, a pak budou všechny lokální proměnné deklarovány jako statické.

```
Static Cislo As Integer          ' deklarace statické proměnné
Public Static Sub Priklad()      ' všechny lokální proměnné jsou statické
```

Statické proměnné se používají například jako počítadla, když chceme kontrolovat, kolikrát byla daná procedura spuštěna. Při každém spuštění zvýšíme hodnotu počítadla o jedničku.

3.3.11 Deklarace proměnných

Deklarací rozumíme určení jména, typu a platnosti proměnné ještě předtím, než tuto proměnnou v kódu použijeme. Ve VBA, na rozdíl od jiných programovacích jazyků, nemusíme deklarovat lokální proměnné. V takovém případě se ale vystavujeme několika problémům:

- Pokud není proměnná deklarována, je jí při prvním požití automaticky přiřazen typ Variant. To znamená vyšší spotřebu místa v operační paměti.
- Kdykoli uděláme při zápisu jména proměnné chybu, vznikne nová proměnná, což se pak odrazí v podivném nebo zcela chybném chování programu.
- Nemůžeme použít některé pomůcky VBA, konkrétně automatické dokončování názvů proměnných.

Proto je mnohem výhodnější změnit nastavení editoru VBA tak, aby si deklaraci lokálních proměnných vynutil. Toto vynucení zajistíme zapnutím volby Require Variable Declaration, kterou najdeme v okně možností Tools →Options na kartě Editor. Při zapnutí volby se do deklarační části všech nových modulů přidá příslušný příkaz Option Explicit. Do již vytvořených modulů musíme tento příkaz přidat sami. Vlastní deklaraci proměnné zajistí příkaz Dim, případně Private nebo Public.

V jednom řádku je možné uvést více deklarací, navzájem oddělených čárkou. Pozor ovšem na to, že pokud chceme zadat jiný typ než výchozí Variant, musíme ho uvést u každé proměnné.

```
Dim i As Integer, j As Integer  ' obě proměnné i a j jsou typu Integer
Dim h, k                        ' obě proměnné h a k jsou typu Variant
Dim l, m As Integer             ' l je proměnná typu Variant, m je proměnná
                                typu Integer
```

Mimoto je možné deklarovat proměnné pomocí typové přípony, která se připojí na konec jména proměnné. Pak ovšem nesmíme zadávat typ pomocí klíčového slova As:

```
Integer     %
Long        &
```

```
Single          !
Double          #
Currency        @
String          $
Dim retez$, cislo%      ' retez je proměnná typu String, cislo je proměnná
                        typu Integer
```

Typové přípony se používají především u konstant. U proměnných jsou dnes již chápány jako zastaralý způsob, a navíc nejsou k dispozici všechny datové typy. Existence datových přípon je příčinou toho, že při psaní kódu nesmíme zapsat spojovací operátor & za název proměnné nebo číslo bez mezery, protože překladač bude tento znak v tomto případě považovat za typovou příponu.

Vlastní konstanty je vždy nutné deklarovat, jinak by překladač nepoznal, že něco chceme používat jako konstantu. Deklarace symbolických (vlastních) konstant se nijak zásadně neliší od deklarace proměnných, jen se místo příkazu Dim používá příkaz Const. V příkazu Const určujeme jméno konstanty a za rovnítkem uvedeme hodnotu, kterou bude konstanta představovat.

```
Const pozdrav = “Ahoj“              ' deklarace lokální konstanty
Public Const pozdrav2 = “Nazdar“    ' deklarace globální (veřejné) konstanty
```

Odlišné je však určení datového typu. Když sami nezadáme datový typ, použije překladač ten, který nejlépe vyhovuje výrazu vpravo od rovnítka. Pokud bychom chtěli datový typ explicitně určit, musíme ho přidat před rovnítko.

```
Const PI = 3.1415           ' překladač přiřadí konstantě typ Double
Const Pocet = 3             ' konstanta Pocet bude typu Integer
Const Pocet2 As Long = 4    ' konstanta Pocet2 je deklarována jako typ
                            Long
```

3.3.12 Zápis hodnot do proměnných

Textové pole zadávané „natvrdo“ v kódu (tzv. literály) se do proměnné vkládají uzavřené do uvozovek. Naopak text, který je zapsán třeba v buňce nebo na popisku ve formuláři, se do proměnné uloží přímo odkazem na příslušnou vlastnost.

```
jmeno = “Jan“
prijmeni = “Macháček“
obsah = ActiveCell.Value
```

Pro spojení několika textových řetězců použijeme zásadně jen operátor &, nikoli operátor +.

```
clovek = jmeno & " " & prijmeni
```

Operátor & lze použít i pro spojení textových řetězců s jiným typem údaje.

```
poradi = 1
misto = "Na " & poradi & ". místě"
    ' v proměnné misto bude uložen text „Na 1.místě"
```

Pokud je nutné uvnitř textového řetězce vyjádřit uvozovky, používá se funkce Chr, která vrací znak, jehož číslo ASCII zadáme jako parametr. Pro uvozovky vypadá volání funkce takto Chr(34). Pro vkládání dalších speciálních znaků (konec řádku, tabulátor) do textových řetězců můžeme používat buď funkci Chr nebo, což je lepší, vestavěné konstanty VBA. Například tabulátor se pomocí funkce Chr zapíše jako Chr(9), vestavěná konstanta má název VbTab.

Číselné hodnoty proměnných se zapisují bez uvozovek. U desetinných čísel je nutné i v českém národním prostředí používat desetinnou tečku, nikoli čárku! Pokud však do proměnné nepřiřazujeme hodnotu přímo, ale odkážeme se například na desetinné číslo zapsané v buňce listu, bude se VBA řídit místním nastavením Windows, a desetinný oddělovač (v českém prostředí čárku) přečte správně. To samé platí i v případě, kdy číslo zadáme jako textovou hodnotu, tedy uzavřené do uvozovek.

Datum nebo čas se zapisují různým způsobem. Jestliže jsme zvyklí na český formát, řadící jednotlivé části do posloupnosti den-měsíc-rok, můžeme uzavírat hodnoty do uvozovek. U anglického formátu měsíc-den-rok je nutné místo uvozovek používat znak #. Pro zápis data a času se také využívají převodní funkce. Jestliže datum zapisujeme v uvozovkách, můžeme použít všechny tvary, které nám Excel nabízí pro zápis kalendářního data do buněk. Pozor na to, že proměnná musí být deklarována jako typ Date, případně uložený řetězec převedeme na datum funkcí CDate. Proměnné typu Date zobrazují datum podle krátkého formátu data nastaveného v našem systému. Čas se bude zobrazovat také podle formátu času nastaveného v systému. Tato systémová nastavení můžeme změnit pomocí dialogového okna Místní nastavení v Ovládacích panelech Windows.

Do logických proměnných můžeme hodnotu Ano/Ne uložit buď přímým přiřazením, nebo vyhodnocením nějakého výrazu. Logické hodnoty True (Ano) a False (Ne) jsou odvozeny z číselného typu Integer, a lze je tedy používat i v číselných výrazech. Logická pravda - True - má číselnou hodnotu mínus 1, nepravda hodnotu 0.

Pokud je logické proměnné přiřazeno jakékoli nenulové číslo, zkonvertuje se na True. Počáteční hodnota proměnné typu Boolean je vždy False.

3.3.13 Pole proměnných

Pole proměnných (array) je skupina prvků stejného datového typu, které mají stejný název, přičemž na konkrétní prvek pole se odkazuje pomocí názvu pole a čísla indexu. Vytvořit můžeme například pole šest řetězcových proměnných, kde každá proměnná odpovídá názvu jedné pobočky podniku. Když toto pole pojmenujeme Pobocky, budeme odkazovat na první prvek tohoto pole zápisem Pobocky(0), na druhý prvek zápisem Pobocky(1) a na poslední Pobocky(5).

Praktické využití vyplývá už z definice – všechny hodnoty, které spolu nějakým způsobem souvisí a tvoří jednu skupinu je možné zpracovávat pomocí polí. Výhodou je například snadná manipulace se všemi prvky pole v cyklech. Každé pole musí být deklarováno příkazem Dim nebo Public. Pro rozsah pole platí stejná pravidla jako pro rozsah proměnné.

VBA dovoluje používat dva základní druhy polí:

- Statická – počet prvků pole je určen předem a nelze ho měnit
- Dynamická – počet prvků pole lze měnit za běhu programu

Pole proměnných s řetězcovými indexy (tzv. asociativní pole) ve VBA neexistuje, namísto něj se používají vlastní kolekce.

Při deklaraci statických polí se udává v závorkách velikost pole (rozsah indexů pole):

```
Dim Osoba(15)           ' indexy jsou v rozmezí 0 až 15
Dim Teplo(-100 To 100)  ' indexy jsou v rozmezí -100 až 100
```

VBA má implicitně nastaven dolní index na nulu. Nechceme-li začínat od nuly, musíme buď zadat rozsah indexů pomocí klauzule To (jako ve druhém příkladě), nebo v deklarační části modulu použít příkaz Option Base 1. Ten nastaví implicitní dolní index na jedničku. Deklarujeme-li pole pomocí klauzule To může být dolní index i záporný. Povolené rozmezí indexů je od -2^{31} do 2^{31} (datový typ Long).

U dynamických polí se neuvádí při deklaraci velikost.

```
Dim DynamickePole()
```

Takto deklarované pole však nelze hned používat, je nutné příkazem ReDim vymezit konkrétní rozsah indexů. U příkazu ReDim lze zadat rozsah i prostřednictvím nějaké již známé proměnné.

```
ReDim DynamickePole(1 To 100)
ReDim DynamickePole(začátek To konec)
```

Příkaz ReDim lze zavolat kdykoli (avšak jen v podprogramu, nikoli v deklarační části – jde o výkonný příkaz), rozsah indexů lze měnit vícekrát, podle potřeby.

Při každém použití příkazu ReDim se všechny proměnné v poli nastaví na hodnotu Empty (popř. nula nebo prázdný řetězec). Chceme-li, aby hodnoty proměnných zůstaly zachovány, musíme použít klauzuli Preserve:

```
ReDim Preserve DynamickePole(60)
```

V takovém případě však můžeme měnit jen horní hranici posledního rozměru pole, jinak dojde k chybě za běhu programu.

Pole mohou mít více dimenzí – lze tedy používat dvojrozměrné či vícerozměrné pole. Deklarace takového pole může vypadat například takto:

```
Dim ViceRozmeru (-20 To 20, 1 To 15, 1 To 30)
```

a v kódu by pak odkaz na prvek tohoto pole mohl být:

```
ViceRozmeru(10, 10, 28)
```

Přiřazení hodnoty do prvku pole není nijak složité. Na konkrétní prvek odkážeme názvem pole a uvedením indexu prvku. Například

```
Pobocky(1) = “Vimperk“
```

Pro naplnění všech prvků pole hodnotami se většinou používají cykly, ve kterých můžeme horní i dolní hranici rozsahu pole testovat funkcemi LBound (nejnižší index prvku pole) a UBound (nejvyšší index prvku pole)

```
For i = LBound(Pobocky) To UBound(Pobocky)
    ' zde by se něco dělo s polem Pobocky
Next i
```

3.3.14 Parametry a jejich druhy

Parametr představuje v programovacím jazyku konstanta, proměnná nebo výraz, který je předáván do procedury nebo funkce. U parametrů mohou rovněž být určeny jejich datové typy. Obecně se mohou vyskytnout tyto situace v souvislosti s parametry:

- Procedura (funkce) nemusí vyžadovat žádné parametry
- Procedura může přebírat pevně stanovený počet parametrů
- Procedura může přebírat předem neurčený počet parametrů
- Procedura může některé parametry vyžadovat a jiné zase mohou být nepovinné

- Procedura může mít všechny parametry nepovinné

Editor jazyka VBA nám při psaní kódu nabízí tzv. rychlou nápovědu. Po napsání názvu příkazu nebo názvu funkce (a levé závorky za jejím názvem) se u kurzoru objeví žlutý rámeček se jmény parametrů a jejich datovým typem. Parametr, který v daném okamžiku máme napsat, je v rychlé nápovědě zobrazen tučně. Jestliže je v okně rychlé nápovědy některý parametr uzavřen do hranatých závorek, znamená to, že je nepovinný. Pokud je parametrů více, oddělují se od sebe čárkami. Při zadávání parametrů máme dvě možnosti. Buď budeme přesně dodržovat pořadí parametrů, jak je dáno syntaxí konkrétního příkazu či funkce. Pokud je některý parametr nepovinný, můžeme ho vynechat, musíme však za takto vynechaným parametrem napsat také čárku. Nebo můžeme využít možnosti psát parametry „na přeskáčku“, kterou nabízí tzv. pojmenované parametry. Tyto parametry se vyskytují u těch procedur, které jsou definovány v objektových knihovnách, kde má každý parametr svůj přesný název. Pojmenované parametry zadáváme včetně jejich názvu, který je od vlastní hodnoty oddělen zvláštním operátorem := (dvojtečka a rovnítko). Zápis s využitím pojmenovaných parametrů se hodí v těch případech, kdy vynecháváme nepovinné parametry. Pořadí pojmenovaných parametrů totiž může být libovolné.

Budeme-li psát vlastní proceduru či funkci, které budou provádět kód v závislosti na určitých vstupních parametrech, musíme tyto parametry deklarovat v hlavičce procedury. K tomuto účelu jsou určeny závorky za názvem procedury či funkce.

```
Public Sub Vypocet (Koef As Double, Zacatek As Date)
```

Na příkladu vidíme, že u parametrů lze určit datové typy. Pokud datové typy nezadáme, bude použit typ Variant.

Parametry uvedené v deklarační hlavičce jsou považovány za formální. Při volání funkce jí totiž budeme předávat skutečné parametry, jejichž názvy budou odpovídat názvům konstant či proměnných. Takže funkci Vypocet použitou v příkladu budeme volat například takto:

```
Koeficient = 5.67
Od = “15.2.2008“
Vysledek = Vypocet (Koeficient, Od)
```

V tomto případě jsou skutečnými parametry proměnné Koeficient a Od. Procedury Vypocet použije proměnnou Koeficient všude tam, kde má ve svém kódu formální parametr Koef a podobně s parametrem Zacatek a proměnnou Od.

Jako parametr můžeme předávat proměnné, konstanty, pole proměnných, odkazy na objekty a textové řetězce. Parametry můžeme proceduře předat dvěma způsoby – odkazem a hodnotou. Výchozí je předávání odkazem, kdy VBA předá do procedury adresu proměnné v paměti počítače. Volaná procedura pak může parametr nejen přečíst, ale také ho měnit. Při předávání parametru hodnotou procedura pouze obdrží kopii původní proměnné, která nemůže být volanou procedurou nijak ovlivněna. Z toho vyplývá, že předávání hodnotou je nutné použít v těch případech, kdy uvnitř procedury provedeme nějaké změny hodnoty parametru, ale po ukončení procedury musí parametr mít zase svoji původní hodnotu. Protože se při předávání hodnotou předává pouze kopie hodnoty, nemusí v některých případech být dodržen datový typ parametru, specifikovaný v hlavičce volaného podprogramu. Pokud je formální parametr deklarován jako typ Variant, provede interpretr VBA sám převod a k žádné chybě nedojde.

Předávání parametru hodnotou zajistíme tak, že před jeho název do hlavičky procedury zapíšeme ByVal

```
Public Sub Přepočet(ByVal Zaklad As Double)
```

Některé typy proměnných nemůžeme předávat hodnotou – uživatelsky definované typy. U proměnných, které zabírají více místa v paměti, trvá předání hodnotou mnohem déle. Proto bychom neměli předávat hodnotou proměnné typu Variant a String.

3.3.15 Rozhodovací bloky

V mnoha případech je nutné vykonávat v kódu různé věci v závislosti na určité podmínce. Příkladů je možné uvést mnoho, počínaje klasickou potřebou provést dvě různé operace v závislosti na hodnotě určité buňky nebo zpracováním určitého souboru ve složce a nutností vyzvat uživatele k zadání názvu či umístění souboru, pokud soubor nebude mít očekávané jméno souboru nebo bude uložen jinde, než kde ho makro hledá. Jde o jednu z nejobvyklejších situací, s nimiž se setkáme. Jakmile se kód musí větvit podle toho, zda nějaká podmínka je nebo není splněna, přicházejí ke slovu řídící struktury, kterým se běžně říká rozhodovací bloky. Jazyk VBA poskytuje tyto dvě řídící struktury If…Then…Else a Select Case. Druhá z nich nabízí jednodušší zápis jednotlivých větví.

Příkaz Select Case. V tomto příkazu vyhodnotíme určitý výraz a v závislosti na jeho hodnotě poté spustíme jednu z několika možných větví kódu. Velkou výhodou příkazu Select Case je fakt, že u jednotlivých větví můžeme zadat několik možných

hodnot nebo rozmezí výsledných hodnot výrazu. Výraz, který testujeme píšeme do prvního řádku bloku, přímo za klíčová slova Select Case. Může zde být porovnání hodnot, výpočet nebo jen proměnná.

```
Select Case a > b              ' výraz vracející True nebo False
Select Case Left(jmeno, 1)     ' výraz vracející znak
Select Case x                  ' proměnná
```

Proměnná může nabývat různých hodnot, proto větví programu může být rovněž neurčitý počet. Jednotlivé větve jsou uvozeny klíčovým slovem Case, za nímž se nacházejí možné hodnoty testovaného výrazu (proměnné). Větev Case Else bude provedena v těch případech, kdy hodnota proměnné neodpovídá žádné jiné větvi – slova Case Else si tedy lze přeložit do češtiny jako „ve všech ostatních případech“. V kódu ji uvádět nemusíme, je nepovinná. Příklad syntaxe:

```
Dim Hodnota As Integer
Select Case Hodnota
    Case 0, 1, 2  ' první větev pro případ, že Hodnota bude mít hodnotu 0, 1
                  nebo 2
        ' zde bude jedna skupina příkazů
    Case 3        ' druhá větev – Hodnota bude 3
        ' zde bude druhá skupina příkazů
    Case Else     ' třetí větev „pro všechny ostatní případy“
        ' zde bude třetí skupina příkazů
```

Konkrétní počet bloků Case závisí samozřejmě na konkrétní situaci. Navíc je možné hodnoty výrazu zadávat v určitém rozsahu či skupině.

```
Case 7 To 10          ' pro číselné hodnoty v rozsahu  7 až 10
Case “A“ To “C“       ' pro textové hodnoty v rozsahu A až C, slovo Auto
                      vyhovuje, slovo Dveře již ne
Case Is > 30          ' pro číselné hodnoty větší než 30
```

Při porovnávání typu „větší než“, „menší než“ musíme použít nejen znak > nebo <, ale i operátor Is. Pokud ho nenapíšeme, editor ho naštěstí doplní za nás.

Rozhodovací blok If…Then…Else je historicky starší, než Select Case. Oproti struktuře Select Case má jednu zásadní výhodu a jednu zásadní nevýhodu. U příkazu If…Then…Else se vždy testuje, zda podmínka platí nebo neplatí. Není tu tedy možné větvení podle různých možných hodnot výrazu či proměnné – pro každou podmínku jsou větve vždy jen dvě. Na druhé straně uvnitř jediného příkazu

If…Then…Else můžeme testovat více různých podmínek, což Select Case neumí. Nejjednodušší podoba tohoto příkazu má jeden jediný řádek.

```
If nějaký_výraz_platí Then něco_provedu [Else provedu_něco_jiného]
```

Tato varianta otestuje nějaký výraz a je-li platný, provede určitý příkaz. Pokud výraz neplatí, můžeme volitelně provést něco jiného, co je uvedeno v části Else. Víceřádková varianta příkazu If…Then…Else se hodí v případech, kdy chceme provést v některé z větví více řádků kódu nebo v případech, kdy potřebujeme testovat více podmínek.

```
If nějaká_podmínka Then
    ' zde může být neomezeně řádků kódu
Else
    ' zde může být jiný kód, který se provede při nesplnění podmínky
End If
```

Přibyl jeden řádek End If čili uzavírací řádek celého rozhodovacího bloku. U víceřádkových bloků If…Then…Else ho musíme vždy uvést.

Jestliže testujeme více podmínek, dá se celý blok vyjádřit tímto schématem:

```
If vyrazA Then              ' pokud platí výraz vyrazA
    prikazA                 ' proveď příkaz prikazA
ElseIf vyrazB               ' pokud platí výraz vyrazB
    prikazB                 ' proveď příkaz prikazB
Else                        ' pokud neplatí žádný předchozí příkaz
    prikazC                 ' proveď příkaz prikazC
End If
```

Části Else a ElseIf jsou nepovinné. Jakmile je splněna některá z uvedených podmínek, nejsou již další podmínky testovány a kód pokračuje až řádkem následujícím za řádkem End If. Toto je vlastně obdoba příkazu Select Case, testuje proměnnou na různé hodnoty a podle toho se provedou různé bloky kódu.

3.3.16 Cykly

Potřebujeme-li určitou část kódu provádět opakovaně, uděláme to nejlépe pomocí cyklu. Příkazy pro cyklus mají svojí hlavičku a ukončení (podobně jako If…Then…Else). Mezi ně se píše blok příkazů, které nebudou opakovat. Ve VBA můžeme použít dva typy cyklů – u jednoho známe počet opakování, u druhého ne. Ve druhém případě místo přesného počtu iterací (opakování) zadáváme podmínku, která se bude při každém opakování testovat – buď před nebo po provedení bloku příkazů.

Cyklus For…Next použijeme, pokud známe počet opakování. Syntaxe je

```
For Pocitadlo = “od“ To “do“ [Step prirustek]
    ' hodnota prirustek se uvádí pouze tehdy, je-li přírůstek různý od 1
    ' blok příkazů
Next
```

Jako počítadlo vystupuje pomocná proměnná Pocitadlo, která se při každém opakování automaticky zvětšuje o jedničku (nezadáme-li za klauzulí Step jiný přírůstek,třeba i záporný). Cykly je možné vnořovat, neboli v rámci jednoho cyklu můžeme spouštět další cyklus.

Cyklus Do…While | Until použijeme, pokud počet opakování neznáme předem a nelze ho určit ani nějakou proměnnou. Jsou dvě možné syntaxe příkazu

```
Do {While | Until} podmínka     ' podminka je podmínka pro opakování cyklu
    [blok příkazů]
    [Exit Do]
    [blok příkazů]
Loop
```

nebo

```
Do
    [blok příkazů]
    [Exit Do]
    [blok příkazů]
Loop  {While | Until} podminka ' podminka je podmínka pro opakování cyklu
```

Podmínka opakování může být uvedena buď v hlavičce (za klíčovým slovem Do), nebo na posledním řádku (za klíčovým slovem Loop). V prvním případě je podmínka testována ještě před provedením cyklu – nebude-li splněna, cyklus se neprovede vůbec. Ve druhém případě proběhne test podmínky až po provedení bloku příkazů, blok příkazů uvnitř cyklu tedy bude spuštěn minimálně jednou. Jako podmínku lze uvést jakýkoli výraz,jehož výslednou hodnotou je True nebo False.

Klíčová slova While a Until stanoví, zda bude cyklus vykonán, dokud podmínka bude platit (While), nebo naopak,dokud platit nezačne (Until). Z toho vyplývá, že Until = While Not (logické opaky).

Cyklus lze předčasně ukončit příkazem Exit Do. Tento příkaz je povinný, pokud v cyklu nezadáme podmínku a klauzuli While | Until. V takovém případě jde o

nekonečný cyklus (někdy velmi užitečný), musíme jej však pomocí nějaké vnořené podmínky ukončit.

Cyklus For Each…Next lze použít u kolekcí objektů a polí proměnných. Tento příkaz provede nějaké činnosti se všemi členy kolekce,nezávisle na jejich počtu. Namísto počítadla, jako u příkazu For…Next, je tu proměnná typu Variant (nebo objektového typu), kterou používáme jako odkaz na jednoho člena kolekce.

```
Dim x                          ' nebo Dim x As Workbook
For Each x In Application.Workbooks
    Debug.Print x.Name
Next
```

V tomto případě se do okna Immediate vypíšou názvy právě otevřených sešitů. Pomocí proměnné x můžeme pracovat s jednotlivými sešity.

3.4 Spouštění procedur

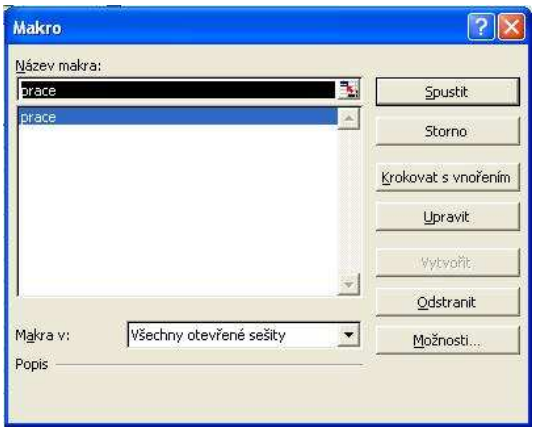
3.4.1 Spouštění procedur z editoru VBA

Jestliže je procedura zobrazena v okně kódu a kurzor je kdekoli uvnitř procedury, můžeme makro spustit klávesou F5. Ekvivalent této klávesy v nabídce editoru je znázorněn na obrázku.

Pokud se kurzor nenachází uvnitř žádné procedury při stisku klávesy F5 nebo tlačítka Run Sub/User Form v menu, zobrazí editor své dialogové okno Makra. Zde si pak můžeme vybrat proceduru, kterou chceme provést. Tento postup se nedá použít u funkcí a procedur s povinnými parametry.

3.4.2 Spouštění procedur z dialogového okna Makro

Pokud se máme otevřený sešit s nějakými makry, můžeme makra spustit přes nabídku menu Nástroje → Makro. Ekvivalentem tohoto tlačítka je klávesová zkratka Alt+F8. V obou případech se zobrazí dialogové okno Makro:



V tomto okně vybereme makro, které chceme spustit a stiskneme tlačítko Spustit. Opět platí, že tento postup se nedá použít u funkcí a procedur s povinnými parametry. To je ovšem zajištěno už tím, že tyto procedury a funkce se ke spuštění ani nenabídnou.

3.4.3 Spouštění procedur z jiných procedur a funkcí

Jde o jeden z nejběžnějších případů, protože v praxi bychom neměli psát příliš dlouhé procedury či funkce. Mnohem lepší je kód rozdělit tak, abychom určité pasáže nemuseli zbytečně opakovat na více místech jednoho podprogramu.Celkem existují tři možnosti, jak z jedné procedury zavolat jinou:

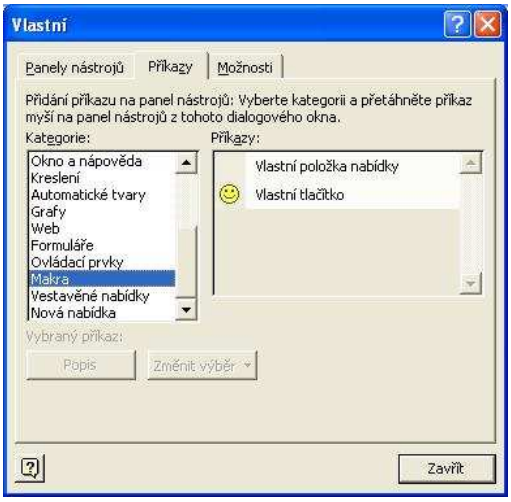
- Zapíšeme název procedury, za kterým následují případné parametry oddělené čárkami bez závorek
- Použijeme příkaz Call, následovaný názvem procedury a případnými parametry zapsanými v závorkách a oddělenými pomocí čárek
- Použijeme metodu Run objektu Application. Tuto metodu můžeme použít také pro spuštění maker uložených v jiném sešitu nebo napsaných v prastarém jazyku XLM. Metoda Run je také užitečná, pokud potřebujeme spustit proceduru, jejíž název je obsažen v proměnné. Potom můžeme metodě Run tuto proměnnou předat jako parametr.

Druhý způsob se v praxi moc nepoužívá, ale jeho výhodou je vyšší srozumitelnost kódu – je jasné, že jde o volání jiné procedury, a ne o volání nějakého vestavěného příkazu VBA.

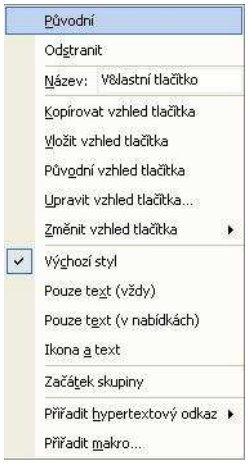
3.4.4 Spouštění procedur z panelů nástrojů

Panely nástrojů jsou dnes zcela běžným rysem uživatelského rozhraní a všichni milovníci myší s nimi pracují velmi rádi. Samotné vytvoření nového tlačítka a přiřazení

makra není nic složitého. Nejdříve musíme zobrazit dialogové okno pro přidání nového tlačítka do panelů nástrojů. Uděláme to tak, že na libovolném panelu nástrojů klepneme pravým tlačítkem myši a z místní nabídky vybereme příkaz Vlastní. Objeví se dialogové okno, ve kterém se přepneme na kartu Příkazy a v levém seznamu označíme kategorii Makra.



V pravém panelu pak uvidíme řádek vlastní tlačítko. Tento řádek uchopíme myší, přesuneme ho do některého panelu nástrojů, a jakmile se svislý kurzor dostane na místo, kam chceme tlačítko zařadit, tlačítko myši uvolníme. Pro tlačítko dále musíme zadat název makra, které se pomocí něj bude spouštět, a také případně upravit jeho vlastnosti – především změnit jeho ikonu, protože výchozí žlutý smajlík nám asi vyhovovat nebude. Místo ikony je také možné zapsat určitý text, například název makra. Všechny tyto volby lze velmi pohodlně provést přes místní nabídku tlačítka, ovšem musíme mít stále zobrazeno dialogové okno Vlastní (jinak pravým tlačítkem myši otevřeme místní nabídku pro zobrazování panelů). Jednotlivé příkazy místní nabídky vidíme na obrázku:



- Název – umožňuje zadat text popisku na tlačítku
- Výchozí styl, Pouze Text, Ikona a text – mění podobu tlačítka. Výchozí styl znamená „jen ikona“.
- Změnit vzhled tlačítka – otevře vedlejší seznam dalších možných ikon, ze kterých si můžeme vybrat
- Upravit vzhled tlačítka – pokud nám nevyhovuje žádná z vestavěných ikon, můžeme si přes tento příkaz vyvolat Editor tlačítek a v něm ikonu upravit bod po bodu. V jeho střední části najdeme paletu 16 barev, v dolní části jsou tlačítka pro posun celého obrázku a podobu tlačítka po dokončení.
- Začátek skupiny – po zapnutí bude vlevo od tlačítka přidán svislý pruh jako oddělovač nové skupiny tlačítek s příbuzným významem
- Přiřadit makro – význam je jasný. Makro si vybereme v dialogovém okně, které se po zadání tohoto příkazu objeví
- Přiřadit hypertextový odkaz – díky tomuto tlačítku si můžeme k tlačítku přiřadit odkaz na jiný soubor, určité místo v jiném souboru nebo na webovou stránku.

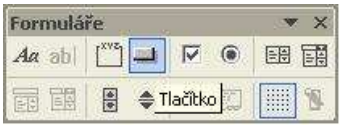
Dialogové okno Vlastní můžeme použít i pro vytváření zcela nových panelů nástrojů a jejich úpravu. Tlačítka z panelů nástrojů lze mazat dvěma možnými způsoby – buď máme zobrazeno dialogové okno Vlastní, a pak stačí tlačítko uchopit, stáhnout ho někam mimo panel (nad buňky) a upustit. Nebo dialogové okno Vlastní zobrazeno není, pak je nutné nejdříve podržet klávesu Alt a poté stejným způsobem myší odstranit.

3.4.5 Spouštění procedur z nabídky programu Excel

Systém nabídek je jedním ze základních součástí uživatelského rozhraní a setkáme se s ním téměř ve všech aplikacích Windows. Přidávání nových příkazů do nabídek se nijak neliší od přidávání nových tlačítek do panelů nástrojů. Ve starších verzích je to trochu horší, s výjimkou nabídky nástroje. V dialogovém okně Vlastní stačí vybrat kategorii Makro v levém seznamu a v pravém panelu to je tentokrát řádek Vlastní položka nabídky. Tento řádek myší přetáhneme na požadované místo v nabídce a tam pustíme. Poté už zbývá jen změnit zobrazený text a přiřadit nové položce makro. To provedeme v místní nabídce, která vypadá stejně jako u tlačítek na panelech. Jinak je celý postup shodný s předchozí možností popsanou v části Spouštění maker z panelů nástrojů.

3.4.6 Spouštění procedur pomocí příkazových tlačítek v listech

Jedná se o velmi vděčný způsob, hlavně pro méně zkušené uživatele. Přímo na listu jsou tlačítka, na které stačí klepnout myší, aby se provedla nějaká procedura. Výhodou je snadná přístupnost tlačítek a jejich textový popis, ze kterého uživatel rychle pozná, k čemu daný prvek slouží. Nejdříve je nutné tlačítko do listu nakreslit. Libovolným způsobem zobrazíme panel nástrojů Formuláře, který vidíme na obrázku.



V tomto panelu nástrojů jsou ikony různých ovladačů, které můžeme po jejich vybrání myší vykreslit přímo do listu. Na obrázku je upravený panel, který je obvykle orientován na výšku. Protože nepatří mezi standardně zobrazené panely, musíme ho nejdříve vyvolat Zobrazit → Panely nástrojů → Formuláře. Myší klepneme na ikonu tlačítka a poté nakreslíme (se stisknutým tlačítkem myši) někam do volného prostoru na listu ovladač. Na tlačítku uvidíme úchyty, pomocí kterých můžeme měnit jeho velikost. Protože se jedná o grafický objekt, stejně jako například čáry nebo části grafů, můžeme ho myší také přesouvat do jiných pozic. Po nakreslení tlačítka se automaticky zobrazí dialogové okno Přiřadit makro, ve kterém vybereme požadovanou proceduru a je hotovo. Jestliže jsme k tlačítku ještě nepřiřadili žádné makro, bude tlačítko po klepnutí myši vybráno a můžeme upravovat jeho vlastnosti, případně přepsat jeho popisek. Jakmile však makro přiřadíme, není možné pro výběr tlačítka použít levé tlačítko myši (tím spustíme makro), ale pravé tlačítko. Poté je opět možné měnit vlastnosti tlačítka.

Pozor na podobnost panelů nástrojů Formuláře a Ovládací prvky. Na obou jsou ikony ovladačů, které sice vypadají stejně, ale funkčně se přece jen liší. Panel nástrojů Ovládací prvky se používá při tvorbě UserForms (formulářů), můžeme je však umísťovat i do listů. [2]

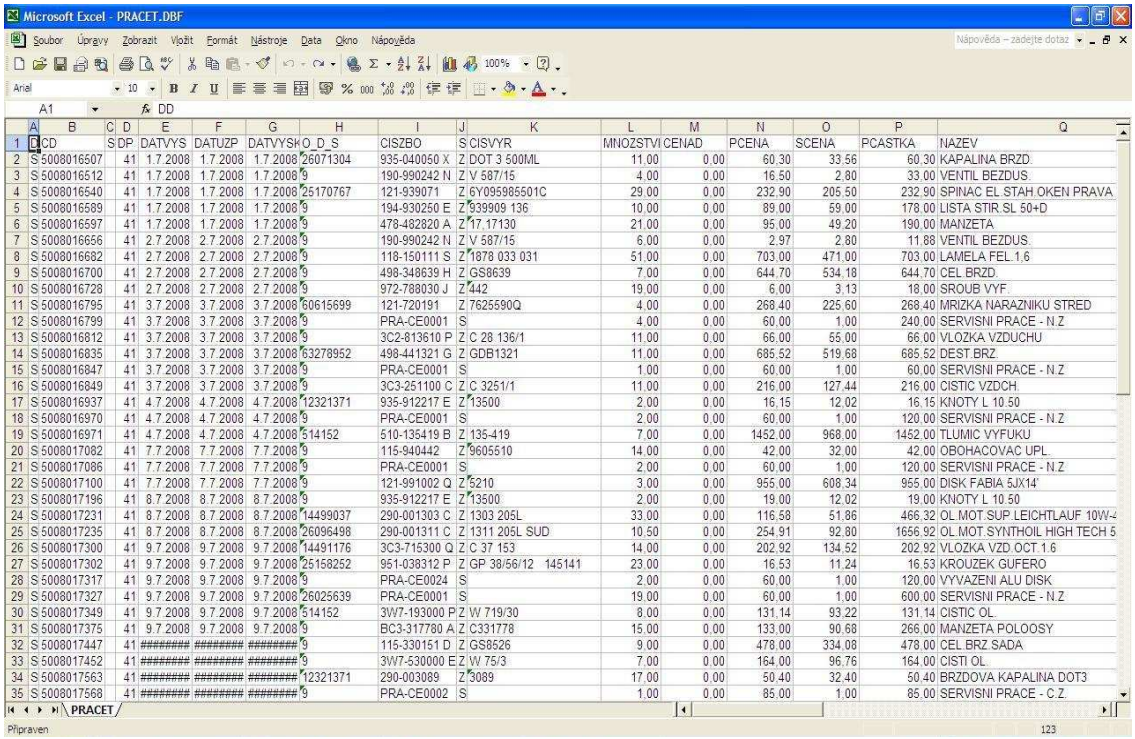
4 Ukázková aplikace

4.1 Popis činnosti aplikace

Aplikace byla vytvořena pro použití ve firmě, ve které jsem zaměstnán. Ve firmě používáme program Office XP, proto je aplikace vytvořena pro tento program. Impulsem pro vytvoření aplikace bylo pravidelné upravování souboru, který vznikal jako výstup z programu, který je součástí skladového hospodářství firmy. Tento program pouze vytvoří soubor obsahující potřebná data. Soubor má pokaždé stejný formát, záhlaví, pořadí a počet sloupců. Proměnlivý je počet řádků. Vytvořený soubor je ale potřeba upravit, některá data dopočítat a naformátovat pro tisk. Výstupem z aplikace je vytištěný dokument a zároveň soubor připravený pro odeslání elektronickou formou.

Konkrétně se jedná o report, který sumarizuje měsíční zisky z provozu střediska servisu automobilů. Je potřeba zjistit nejen jaký byl zisk čistě za práci servisních techniků – tyto čísla jsou snadno k dosažení pomocí jiného podprogramu skladového hospodářství, ale jaký byl zisk za práci i náhradní díly a ostatní materiál prodaný přes středisko servisu. Program skladového hospodářství, který tento soubor vytváří projde soubor všech faktur vydaných firmou a označí všechny faktury, na kterých se fakturovala servisní práce. Poté vygeneruje požadovaný soubor, kde na každém řádku je součet za každou jednu fakturu, na které se fakturovala nějaká servisní práce.

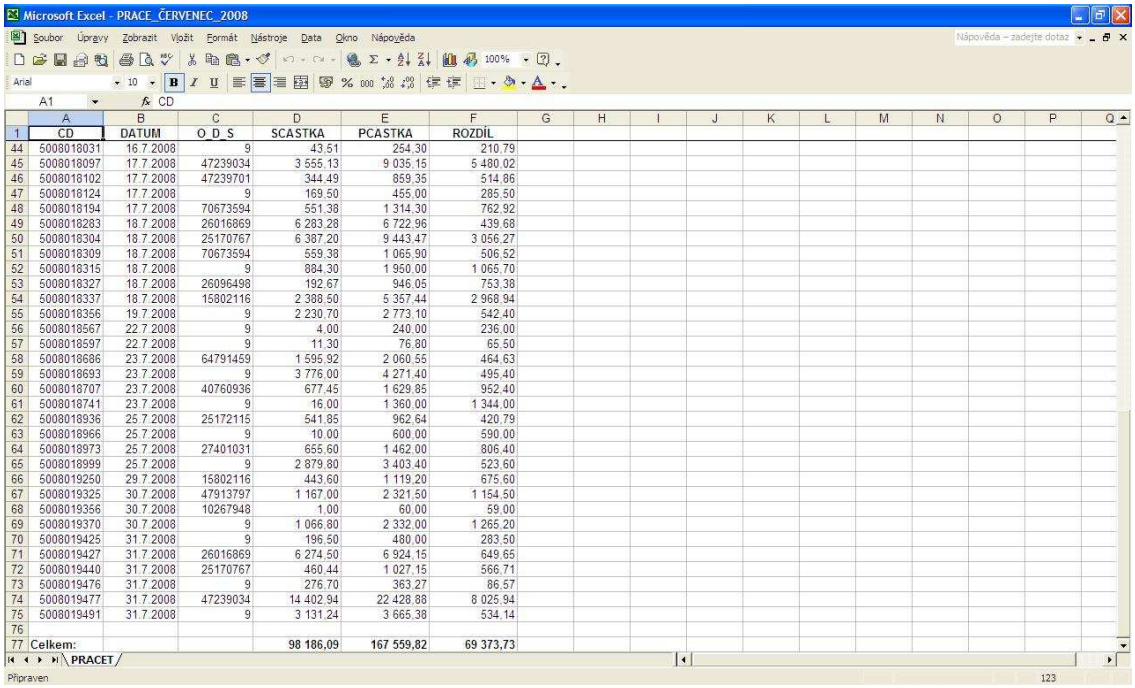
Takto například vypadá soubor po výstupu z programu skladového hospodářství:



Microsoft Excel - PRACET.DBF

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	D	CD	S	DP	DATVYS	DATUZP	DATVYSK	O_D_S	CISZBO	S	CISVYR	MNOZSTVI	CENAD	PCENA	SCENA	PCASTKA	NAZEV
2	S	5008016507		41	1.7.2008	1.7.2008	1.7.2008	26071304	935-040050 X	Z	DOT 3 500ML	11.00	0.00	60,30	33,56	60,30	KAPALINA BRZD.
3	S	5008016512		41	1.7.2008	1.7.2008	1.7.2008	9	190-990242 N	Z	V 587/15	4.00	0.00	16,50	2,80	33,00	VENTIL BEZDUS.
4	S	5008016540		41	1.7.2008	1.7.2008	1.7.2008	25170767	121-939071	Z	6Y095985501C	29.00	0.00	232,90	205,50	232,90	SPINAC EL.STAH.OKEN PRAVA
5	S	5008016589		41	1.7.2008	1.7.2008	1.7.2008	9	194-930250 E	Z	939909 136	10.00	0.00	89,00	59,00	178,00	LISTA STIR.SL 50+D
6	S	5008016597		41	1.7.2008	1.7.2008	1.7.2008	9	478-482820 A	Z	17.17130	21.00	0.00	95,00	49,20	190,00	MANZETA
7	S	5008016656		41	2.7.2008	2.7.2008	2.7.2008	9	190-990242 N	Z	V 587/15	6.00	0.00	2,97	2,80	11,88	VENTIL BEZDUS.
8	S	5008016682		41	2.7.2008	2.7.2008	2.7.2008	9	118-150111 S	Z	1878 033 031	51.00	0.00	703,00	471,00	703,00	LAMELA FEL.1,6
9	S	5008016700		41	2.7.2008	2.7.2008	2.7.2008	9	498-348639 H	Z	GS8639	7.00	0.00	644,70	534,18	644,70	CEL.BRZD.
10	S	5008016728		41	2.7.2008	2.7.2008	2.7.2008	9	972-788030 J	Z	442	19.00	0.00	6,00	3,13	18,00	SROUB VYF.
11	S	5008016795		41	3.7.2008	3.7.2008	3.7.2008	60615699	121-720191	Z	7625590Q	4.00	0.00	268,40	225,60	268,40	MRIZKA NARAZNIKU STRED
12	S	5008016799		41	3.7.2008	3.7.2008	3.7.2008	9	PRA-CE0001	S		4.00	0.00	60,00	1,00	240,00	SERVISNI PRACE - N.Z
13	S	5008016812		41	3.7.2008	3.7.2008	3.7.2008	9	3C2-813610 P	Z	C 28 136/1	11.00	0.00	66,00	55,00	66,00	VLOZKA VZDUCHU
14	S	5008016835		41	3.7.2008	3.7.2008	3.7.2008	63278952	498-441321 G	Z	GDB1321	11.00	0.00	685,52	519,68	685,52	DEST.BRZ.
15	S	5008016847		41	3.7.2008	3.7.2008	3.7.2008	9	PRA-CE0001	S		1.00	0.00	60,00	1,00	60,00	SERVISNI PRACE - N.Z
16	S	5008016849		41	3.7.2008	3.7.2008	3.7.2008	9	3C3-251100 C	Z	C 3251/1	11.00	0.00	216,00	127,44	216,00	CISTIC VZDCH.
17	S	5008016937		41	4.7.2008	4.7.2008	4.7.2008	12321371	935-912217 E	Z	13500	2.00	0.00	16,15	12,02	16,15	KNOTY L 10.50
18	S	5008016970		41	4.7.2008	4.7.2008	4.7.2008	9	PRA-CE0001	S		2.00	0.00	60,00	1,00	120,00	SERVISNI PRACE - N.Z
19	S	5008016971		41	4.7.2008	4.7.2008	4.7.2008	514152	510-135419 B	Z	135-419	7.00	0.00	1452,00	968,00	1452,00	TLUMIC VYFUKU
20	S	5008017082		41	7.7.2008	7.7.2008	7.7.2008	9	116-940442	Z	9605510	14.00	0.00	42,00	32,00	42,00	OBOHACOVAC UPL.
21	S	5008017086		41	7.7.2008	7.7.2008	7.7.2008	9	PRA-CE0001	S		2.00	0.00	60,00	1,00	120,00	SERVISNI PRACE - N.Z
22	S	5008017100		41	7.7.2008	7.7.2008	7.7.2008	9	121-991002 Q	Z	5210	3.00	0.00	955,00	608,34	955,00	DISK FABIA 5JX14"
23	S	5008017196		41	8.7.2008	8.7.2008	8.7.2008	9	935-912217 E	Z	13500	2.00	0.00	19,00	12,02	19,00	KNOTY L 10.50
24	S	5008017231		41	8.7.2008	8.7.2008	8.7.2008	14499037	290-001303 C	Z	1303 205L	33.00	0.00	116,58	51,86	466,32	OL.MOT.SUP.LEICHTLAUF 10W-4
25	S	5008017235		41	8.7.2008	8.7.2008	8.7.2008	26096498	290-001311 C	Z	1311 205L SUD	10.50	0.00	254,91	92,80	1656,92	OL.MOT.SYNTHOIL HIGH TECH 5
26	S	5008017300		41	9.7.2008	9.7.2008	9.7.2008	14491176	3C3-715300 Q	Z	C 37 153	14.00	0.00	202,92	134,52	202,92	VLOZKA VZD.OCT.1.6
27	S	5008017302		41	9.7.2008	9.7.2008	9.7.2008	25158252	951-038312 P	Z	GP 38/56/12 145141	23.00	0.00	16,53	11,24	16,53	KROUZEK GUFERO
28	S	5008017317		41	9.7.2008	9.7.2008	9.7.2008	9	PRA-CE0024	S		2.00	0.00	60,00	1,00	120,00	VYVAZENI ALU DISK
29	S	5008017327		41	9.7.2008	9.7.2008	9.7.2008	26025639	PRA-CE0001	S		19.00	0.00	60,00	1,00	600,00	SERVISNI PRACE - N.Z
30	S	5008017349		41	9.7.2008	9.7.2008	9.7.2008	514152	3W7-193000 P	Z	W 719/30	8.00	0.00	131,14	93,22	131,14	CISTIC OL.
31	S	5008017375		41	9.7.2008	9.7.2008	9.7.2008	9	BC3-317780 A	Z	C331778	15.00	0.00	133,00	90,68	266,00	MANZETA POLOOSY
32	S	5008017447		41	########	########	########	9	115-330151 D	Z	GS8526	9.00	0.00	478,00	334,08	478,00	CEL.BRZ.SADA
33	S	5008017452		41	########	########	########	9	3W7-530000 E	Z	W 75/3	7.00	0.00	164,00	96,76	164,00	CISTI OL.
34	S	5008017563		41	########	########	########	12321371	290-003089	Z	3089	17.00	0.00	50,40	32,40	50,40	BRZDOVA KAPALINA DOT3
35	S	5008017568		41	########	########	########	9	PRA-CE0002	S		1.00	0.00	85,00	1,00	85,00	SERVISNI PRACE - C.Z.

Takto vytvořený soubor obsahuje mnoho dat navíc, které nejsou potřeba. Pro vytvoření požadovaného reportu stačí pouze 5 sloupců a sice číslo dokladu „CD“, datum vystavení „DATVYS“, odběratel „O_D_S“, částka za kterou bylo zboží na faktuře nakoupeno „PCASTKABD“, který aplikace přejmenuje na „SCASTKA“ a částka za kterou bylo zboží a servisní práce na faktuře prodáno „PCASTKA“. Šestý sloupec aplikace dopočítá, jedná se o rozdíl mezi sloupcem „PCASTKA“ a „SCASTKA“. Za posledním obsazeným řádkem aplikace pro zvýraznění jeden řádek vynechá a na další řádek dopočítá součet sloupců „PCASTKA“, „SCASTKA“ a „ROZDIL“. Na závěr aplikace nabídne možnost vytištění nově upraveného souboru a jeho uložení. Soubor je uložen do stejného adresáře, kde se nacházel původní soubor, což je požadováno a zároveň je soubor vytištěn na výchozí tiskárně a poté uzavřen. Po všech úpravách vypadá soubor takto:



Microsoft Excel - PRACE_ČERVENEC_2008

	A	B	C	D	E	F
1	CD	DATUM	O_D_S	SCASTKA	PCASTKA	ROZDÍL
44	5008018031	16.7.2008	9	43,51	254,30	210,79
45	5008018097	17.7.2008	47239034	3 555,13	9 035,15	5 480,02
46	5008018102	17.7.2008	47239701	344,49	859,35	514,86
47	5008018124	17.7.2008	9	169,50	455,00	285,50
48	5008018194	17.7.2008	70673594	551,38	1 314,30	762,92
49	5008018283	18.7.2008	26016869	6 283,28	6 722,96	439,68
50	5008018304	18.7.2008	25170767	6 387,20	9 443,47	3 056,27
51	5008018309	18.7.2008	70673594	559,38	1 065,90	506,52
52	5008018315	18.7.2008	9	884,30	1 950,00	1 065,70
53	5008018327	18.7.2008	26096498	192,67	946,05	753,38
54	5008018337	18.7.2008	15802116	2 388,50	5 357,44	2 968,94
55	5008018356	19.7.2008	9	2 230,70	2 773,10	542,40
56	5008018567	22.7.2008	9	4,00	240,00	236,00
57	5008018597	22.7.2008	9	11,30	76,80	65,50
58	5008018686	23.7.2008	64791459	1 595,92	2 060,55	464,63
59	5008018693	23.7.2008	9	3 776,00	4 271,40	495,40
60	5008018707	23.7.2008	40760936	677,45	1 629,85	952,40
61	5008018741	23.7.2008	9	16,00	1 360,00	1 344,00
62	5008018936	25.7.2008	25172115	541,85	962,64	420,79
63	5008018966	25.7.2008	9	10,00	600,00	590,00
64	5008018973	25.7.2008	27401031	655,60	1 462,00	806,40
65	5008018999	25.7.2008	9	2 879,80	3 403,40	523,60
66	5008019250	29.7.2008	15802116	443,60	1 119,20	675,60
67	5008019325	30.7.2008	47913797	1 167,00	2 321,50	1 154,50
68	5008019356	30.7.2008	10267948	1,00	60,00	59,00
69	5008019370	30.7.2008	9	1 066,80	2 332,00	1 265,20
70	5008019425	31.7.2008	9	196,50	480,00	283,50
71	5008019427	31.7.2008	26016869	6 274,50	6 924,15	649,65
72	5008019440	31.7.2008	25170767	460,44	1 027,15	566,71
73	5008019476	31.7.2008	9	276,70	363,27	86,57
74	5008019477	31.7.2008	47239034	14 402,94	22 428,88	8 025,94
75	5008019491	31.7.2008	9	3 131,24	3 665,38	534,14
76						
77	Celkem:			98 186,09	167 559,82	69 373,73

4.2 Kód aplikace

```
Sub prace()
' prace Makro
' Makro zaznamenané 20.1.2008, Jan Macháček
' Klávesová zkratka: Ctrl+Shift+P
```

```
Dim pocetr As Integer
    ' deklarace proměnných požívaných při běhu aplikace
Dim sloupec As String
Dim radek As String
Dim rozsah As String
Dim posun As Integer
Dim bunka As String
Dim vzorec As String
Dim mesic_cislo As Integer
Dim mesic(1 To 12)
Dim rok As String
Dim datum As Date
Dim nazev As String

mesic(1) = "LEDEN"
    ' deklarace hodnot pole mesic
mesic(2) = "ÚNOR"
mesic(3) = "BŘEZEN"
mesic(4) = "DUBEN"
mesic(5) = "KVĚTEN"
mesic(6) = "ČERVEN"
mesic(7) = "ČERVENEC"
mesic(8) = "SRPEN"
mesic(9) = "ZÁŘÍ"
mesic(10) = "ŘÍJEN"
mesic(11) = "LISTOPAD"
mesic(12) = "PROSINEC"

Rows("2:2").Select
    ' vybere druhý řádek
ActiveWindow.FreezePanes = True
    ' ukotví řádky nad předchozím výběrem
Cells.Select
```

```
    ' výběr všech buněk
Cells.EntireColumn.AutoFit
    ' všechny sloupce přizpůsobí šířku
Range("A:A,C:F,I:AR,AU:BQ").Delete Shift:=xlToLeft
    ' smazání sloupců a posun buněk doleva
Range("B1") = "DATUM"
    ' přepsání obsahu buňky na DATUM
Range("D1") = "SCASTKA"
    ' přepsání obsahu buňky na SCASTKA
Range("E1") = "PCASTKA"
    ' přepsání obsahu buňky na PCASTKA
Range("F1") = "ROZDÍL"
    ' přepsání obsahu buňky na ROZDÍL
Columns("A:C").ColumnWidth = 11.43
    ' určení šířky prvních tří sloupců
Columns("D:F").ColumnWidth = 13.57
    ' určení šířky druhé trojice sloupců
pocetr = Range("E2").CurrentRegion.Rows.Count
    ' spočítání obsazené řádky kolem zadané buňky
For radky = 2 To pocetr
  bunk = "C" & radky
  hodnota = Range(bunk).Value
  cislo = CLng(hodnota)
  Range(bunk) = cislo
  Range(bunk).NumberFormat = "@"
Next radky
    ' cyklus, který pro celý sloupec C od prvního do posledního obsazeného
    řádku převede číslo uložené jako text na číslo  a dá všem těmto číslům
    jednotný formát

Columns("C:C").HorizontalAlignment = xlRight
    ' zarovnání třetího sloupce doprava
Rows("1:1").Font.Bold = True
    ' první řádek tučným písmem
```

```
    Rows("1:1").HorizontalAlignment = xlCenter
        ' první řádek zarovnat na střed
    Range("F2").FormulaR1C1 = "=SUM(RC[-1],-RC[-2])"
        ' spočítání rozdílu mezi sloupci SCASTKA a PCASTKA v druhé řádce
    radek = CStr(pocetr)
        ' převede číslo posledního obsazeného řádku na řetězec, aby bylo
        možno číslo přiřadit do vzorce pro zadání adresy buňky
    sloupec = "F"
        ' název sloupce MUSÍ být zadán jako řetězec
rozsah = sloupec & "2" & ":" & sloupec & radek
        ' adresa buňky MUSÍ být jako řetězec, tento řetězec určí rozsah ve
        sloupci F od druhého řádku do posledního obsazeného řádku
    Range("F2").AutoFill Destination:=Range(rozsah), Type:=xlFillDefault
        ' vyplní automaticky všechny buňky v rozsahu zadaném hodnotou
        „rozsah“ podle druhé řádky
    posun = pocetr + 2
        ' odsazení prázdným řádkem a spočítání adresy řádky, pro výpočet
        celkových hodnot
    radek = CStr(posun)
        ' převedení hodnoty na řetězec
    bunka = "A" & radek
        ' určení adresy buňky pro zápis
    Range(bunka) = "Celkem:"
        ' zápis hodnoty do buňky na určené adrese
    bunka = "D" & radek
        ' určení adresy buňky pro zápis
    vzorec = "=SUM(R[-" & pocetr & "]C:R[-1]C)"
        ' vytvoření vzorce pro součet hodnot v zadaném rozsahu buněk
    Range(bunka).FormulaR1C1 = vzorec
        ' vypsání vzorce do buňky na adresu určenou hodnotou „bunka“
    rozsah = bunka & ":" & sloupec & radek
        ' určení rozsahu oblasti, která bude dále použita
    Range(bunka).AutoFill Destination:=Range(rozsah), Type:=xlFillDefault
```

```
        ' oblast určená proměnnou „rozsah“ bude vyplněna stejně, jako buňka
        určená proměnnou „bunka“
    rozsah = "D2:" & sloupec & radek
        ' určení rozsahu oblasti, která bude dále použita
    Range(rozsah).NumberFormat = "#,##0.00"
        ' nastavení formátu čísel v použité oblasti
    rozsah = radek & ":" & radek
        ' určení rozsahu oblasti, která bude dále použita
    Rows(rozsah).Font.Bold = True
        '  nastavení tučného písma v použité oblasti
    datum = Range("B2").Value
        ' zjištění hodnoty buňky B2 a její přiřazení do proměnné „datum“
    mesic_cislo = Month(datum)
        ' přiřazení čísla měsíce do proměnné „mesic_cislo“
    rok = Year(datum)
        ' přiřazení roku do proměnné „rok“
    nazev = "PRÁCE_" & mesic(mesic_cislo) & "_" & rok
        ' vytvoření názvu souboru sloučením řetězců „PRÁCE_“ , měsíc a rok
If vbYes = MsgBox("Soubor bude uložen jako: " & nazev & vbNewLine &
"Chcete soubor vytisknout a zavřít?", vbYesNo) Then
        ' větvení aplikace, o kterém rozhodne uživatel. Tento informační box se
        zeptá a nabídne možnost, zda uživatel chce takto upravený soubor uložit
        a vytisknout. Pokud ano, aplikace vykoná následující řádky.
     ActiveWorkbook.SaveAs "C:\pok\prace\" & nazev, xlWorkbookNormal
        ' uložení tohoto sešitu s názvem daným proměnnou „nazev“ do adresáře
        C:\pok\prace, kde potřebuji, aby se daný soubor ukládal, ve formátu
        definovaným hodnotou xlWorkbookNormal
    With ActiveSheet.PageSetup
        ' pro aktivní list nastaví parametry tisku - vzhledu stránky
      .PrintTitleRows = "$1:$1"
        ' nahoře opakovat řádek 1
      .CenterFooter = "&F"
        ' zápatí - ve středu zobrazit název souboru
      .RightFooter = "Stránka &P"
```

```
        ' zápatí - vpravo zobrazit číslo stránky
      .CenterHorizontally = True
        ' vycentrovat stránku horizontálně
      .Orientation = xlPortrait
        ' otočit stránku na výšku
      .FirstPageNumber = xlAutomatic
        ' číslo první stránky automaticky
    End With
    ActiveSheet.PrintOut copies:=1
        ' vytištění souboru, 1 kopie
    ActiveWorkbook.Close SaveChanges:=True
        ' uzavření sešitu a uložení provedených změn
    Exit Sub
        ' ukončení aplikace
  Else
        ' pokud uživatel při větvení programu stiskl volbu, že nechce soubor
        tisknout a ukládat, provede se následující část kódu
    ActiveWorkbook.Close SaveChanges:=False
        ' uzavření sešitu bez uložení provedených změn
    Exit Sub
        ' ukončení aplikace
   End If
        ' ukončení větvení kódu
End Sub
        ' klíčové slovo označující konec kódu
```

5 Závěr

5.1 Porovnání dostupných nástrojů

Poprvé se jazyk VBA objevil v programu Excel verze 5 a byl lokalizován (například do německého jazyka). Programovací jazyk se však změnil už s příchodem další verze Excel 7. Ten byl prodávaný pod názvem Excel 95. Bylo sice stále možné používat lokalizované příkazy, avšak v plánu bylo sjednocení příkazů zpět do angličtiny. Od verze 5 podporuje Excel oba makro jazyky, jak XLM, tak VBA. Jak to bude s těmito jazyky vypadat v budoucnu je nejasné. Office 2007 stále ještě podporuje oba.

V roce 1997 byl na trh uveden Excel 8 s obchodním názvem Excel 97. Tato verze je značně pozměněná, bylo zavedeno podmíněné formátování a změněn datový formát sešitu. Starší verze programu Excel neuměly s novým formátem pracovat. VBA navíc používalo upravené rozhraní a také objektový model byl mírně upraven. Makra již nebyla nadále středem pozornosti, byla potlačena do pozadí. Moduly a listy byly přesunuty do projektů VBA, kód lze upravovat pouze pomocí Visual Basic Editoru. Kromě standardních modulů byly zavedeny také třídy, které umožňují vytváření nových objektů a událostí. Přibylo velké množství událostí, bylo možné integrovat ActiveX komponenty. Uvolněna byla také rozšiřující knihovna umožňující změnu vývojového prostředí VBE a úpravy kódu VBA pomocí VBA.

V roce 1999 vydal Microsoft Excel verze 9 pod označením Excel 2000. Změny předchozí verze zůstaly zachovány. Tato verze programu Excel nabídla navíc další nové funkce. Poprvé byly k dispozici kontingenční grafy. Další velký krok směřoval k Internetu. Nyní bylo možné uložit tabulku ve formátu HTML, tabulka mohla být navíc interaktivní. To znamená, že ve webovém dokumentu vytvořeném Excesem bylo možné tabulku upravovat online. Byla také zavedena funkce „Spolupráce online“ v nabídce „Nástroje“. Pro uživatele VBA byla v uživatelských formulářích (UserForms) zavedena vlastnost modeless, která umožňuje tvorbu modálních formulářů. Modální formulář je takový, který umožní při otevřeném dialogu změnit obsah listu. Před verzí 2000 nebylo toto možné. Nejprve musel být dialog uzavřen a teprve poté bylo možné obsah listu upravit, nebo přepnout list či sešit.

Excel verze 10 se začal prodávat v roce 2001. Obchodní název měl sice Excel 2002, ale častěji byl označován jako Excel XP. Microsoft opět zabudoval další funkce

ubírající se směrem k webu a Internetu. Podpora XML (eXtensible Markup Language) byla vylepšena. Byly zavedeny funkce jako SmartTags, RTD (Real Time Data) a nový objektový model týkající se ochrany listu.

SmartTags, chytré značky, jsou malé symboly v listě, které se objeví teprve, pokud nastane jistá událost. Chytrá značka se například zobrazí při zadání chybného vzorce. Po klepnutí na chytrou značku se zobrazí rozbalovací menu se speciálními příkazy pro odstranění chyby. Chytrých značek existuje mnoho typů.

V programu Excel verze XP mají vývojáři možnost používat pomocí RTD automatizační server COM (Component Object Model) pro získání dat v reálném čase. V předchozích verzích se používala technologie DDE (Dynamic Data Exchange).

Funkce zamykání listu se výrazně vylepšila (NÁSTROJE→ZÁMEK→ ZAMKNOUT LIST). Nyní existuje řada objektů, které lze při aktivaci zámku listu nastavit. Tímto způsobem lze chránit celý list proti změnám. Změnu struktury a formátu listu, sloupců, řádků a buněk lze nastavit individuálně. U formátování je nyní možné nastavit barvu posuvníků.

Objevily se také novinky v uživatelském rozhraní a interakci s uživatelem. Konečně lze vyhledávat mezi jednotlivými listy. Funkce řazení byla výrazně vylepšena. Záhlaví a zápatí lze nyní lépe upravovat, lze vkládat také obrázky. Do záhlaví, příp. zápatí lze také vložit cestu a název souboru. To bylo dříve možné pouze pomocí VBA.

V roce 2003 se na trhu objeli Excel 11. V jeho názvu se opět sjednotil název s rokem vydání. Excel 2003 se kromě zmodernizovaného vzhledu oproti očekávání příliš nezměnil. Zaměření je především směrem ke XML, tedy opět směrem na web. Velmi příjemnou novinkou je přítomnost seznamů, které lze automaticky vytvořit, popř. Excel při jejich vytvoření nabídne pomoc. Klepnutím pravého tlačítka myši v buňce nebo označené oblasti se zobrazí místní nabídka s novým příkazem VYTVOŘIT SEZNAM. Stejný příkaz nalezneme také v nabídce DATA\SEZNAM. Oba příkazy vedou ke stejnému výsledku. Vytvořený seznam ihned uvidíme. Položky seznamu lze například automaticky filtrovat. Navíc se zobrazí vlastní panel nástrojů SEZNAM. Například klepnutím na tlačítko IMPORTOVAT DATA XML lze automaticky importovat odpovídající data. Při aktivovaném filtru je jako nejvyšší kritérium použita funkce řazení.

Zatím poslední vydání nese název Excel 2007. Zásadním rozdílem oproti předchozím verzím je odlišné uživatelské rozhraní. Dalším výrazným zlepšením je

maximální počet řádků. Zatímco předchozí verze dovolovaly použití maximálně 65.536 řádků na jednom listu, v této verzi lze použít i milion řádků.

Podstatné nevýhody má Excel (a vůbec celý kancelářský balík Microsoft Office) dvě. První nevýhodou je fakt, že je použitelný pouze pro operační systém Microsoft Windows. Tuto nevýhodu hodně stírá fakt, že operační systém Windows je celosvětově nejrozšířenější operační systém. Druhou nevýhodou je cena. Kancelářský balík Microsoft Office 2007, obsahující program Excel 2007, se v nejlevnější variantě prodává pod označením Home and Student a jeho cena se pohybuje v současnosti kolem jednoho tisíce korun. Tato varianta je ovšem pouze pro domácí a nekomerční použití. Varianty určené pro komerční účely jsou mnohem dražší.

Stejně jako ve většině oblastí programového vybavení osobních počítačů, i zde existuje freewarová alternativa. Je jím kancelářský balík OpenOffice.org a jeho tabulkový kalkulátor Calc. Volně ke stažení je tento program například na stránkách www.openoffice.cz. Aktuální verze nese označení OpenOffice.org 3.0. Tento program je vyvíjen jako tzv. opensource software. Podle informací z webových stránek klub.openoffice.cz kancelářský balík OpenOffice.org podporuje tyto operační systémy:

- Microsoft Windows
- GNU / LINUX (x86, PowerPC)
- Solaris (SPARC)
- MacOS X
- OS/2 (komerční verze)

Je vytvořen tak, aby v malých firmách, domácnostech, ve školách a všude tam, kde potřebujeme pracovat s kancelářskými aplikacemi, ale nemáme dostatek prostředků na nákup drahého softwaru, dokázal nahradit dosud používané drahé kancelářské balíky, zejména Microsoft Office. K tomu nabízí ještě další výhodu – dokumenty Microsoft Office se dají do formátů OpenOffice.org snadno převést. Je třeba upozornit, že jsou určité hranice, na nichž převod havaruje. Jedná se zejména o převod maker, s nimiž si OpenOffice.org příliš nerozumí. Některé detaily pak ztvárňují dokumenty OpenOffice.org trochu jinak – například grafy programu Excel ze samostatných listů.

VBA v programu Excel však nabízí uživatelům více funkcí. Pro užívání programu Excel také hovoří větší počet uživatelů a tím pádem větší pravděpodobnost správného zobrazení předávaných dokumentů a převoditelnost vytvořených funkcí a procedur. Bohužel funkce a procedury vytvořené v každém systému jsou na druhý téměř nepřevoditelné kvůli odlišné syntaxi, pojmenování příkazů a objektů.

5.2 Zhodnocení použitelnosti pro uživatele

Užití maker je velmi široké. Od jednoduchých maker, která zachycují několik příkazů zadaných z klávesnice, až po rozsáhlé soubory maker, které mohou zvýšit výkonnost univerzálního programu na úroveň blížící se jednoúčelovým programům. Možnosti programovacích jazyků jsou dnes natolik bohaté, že budeme těžko hledat zadání, které by nebylo možno makrem zajistit. Velikost maker není prakticky omezena. Přesto je potřeba se snažit, aby byla přehledná a ne příliš rozsáhlá, v maximální míře využít skutečnost, že makra mohou navzájem spolupracovat. Složitější úkol je vždy možno rozdělit na několik dílčích úkolů a pro každý dílčí úkol vytvořit samostatné makro. Tento postup má několik výhod. Při dělení složitého úkolu na dílčí úlohy si zpravidla můžeme ověřit, jestli opravdu víme, jakou činnost a za jakých podmínek má makro zajistit. Napsání a ověření správné funkce několika jednodušších maker bývá zpravidla snazší a jednodušší, než u jednoho rozsáhlého makra.

Jednoduchá makra zajišťují určitou činnost, představují jakési stavební kameny, které můžeme opakovaně používat. Proto je výhodné ke každému novému skriptu připojit textový komentář, v němž je popsána činnost a způsob, jakým pracuje. Podrobnější popis oceníme zvláště, když se k makru vracíme po delší době.

Použití maker v praxi je hodně závislé na konkrétním uživateli a jeho programátorských schopnostech. Pokud si jako běžného uživatele tabulkového kalkulátoru představíme například sekretářku zpracovávající data pro nadřízeného, nejpravděpodobněji ze všech možností makroprogramování využije pouze Záznamník maker. Ale i toto využití opravdu laickému uživateli bude stačit a hodně mu ulehčí jeho každodenní práci. Příklad, který jsem popsal pro využití Záznamníku maker je v praktickém životě naší firmy využíván opravdu denně. Pro pohodlné a často opakované upravování dokumentů nyní stačí jednou kliknout na nově přidanou ikonku v panelu nástrojů.

Pokud bude uživatelem někdo s více programátorskými zkušenostmi, ulehčí si život samozřejmě více. Při vytváření různých dokumentů nemusí používat pouze vestavěné funkce, ale vytvářet si svoje vlastní. VBA umožňuje i automatizaci při vytváření grafů. Velmi zkušený programátor by pomocí VBA jistě dokázal naprogramovat i celý informační podnikový systém. Tabulkový kalkulátor pomocí

formulářů a seznamů ve VBA dokáže nahradit databázový systém. Pro zpracování databází v současné době existuje velké množství programů. Součástí Microsoft Office ve verzi Professional (a vyšších) je Microsoft Access. Jedná se o solidní aplikaci pro práci se stolními databázemi, tedy databázemi menšího rozsahu. U skutečně miniaturních databází je však zbytečné používat Access – v programu Excel najdeme všechny základní nástroje pro zpracování databází, ukládaných přímo na listech sešitů.

Seznam použité literatury:

[1] Weber M., Breden M.: Excel VBA – Velká kniha řešení, Computer Press, 2007
[2] Černý J.: Excel 2000 – 2007 záznam, úprava a programování maker – 2., aktualizované vydání, Grada 2008
[3] Černý J.: Programování v Excelu 2000, 2002, 2003, Grada, 2005
[4] Sobek M.: OpenOffice.org tipy a triky pro záznam a úpravu maker, Grada, 2006

Internet:

[5] www.openoffice.cz
[6] www.excelplus.net