Petr Špiroch

OpenOffice.org – programování maker

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta Katedra informatiky

OpenOffice.org – programování maker

bakalářská práce

Autor: Petr Špiroch

Vedoucí bakalářské práce: Mgr. Jiří Pech, Ph.D.

České Budějovice 2007

Název práce: OpenOffice.org – programování maker
Autor: PETR ŠPIROCH

Katedra: Katedra Informatiky

Vedoucí diplomové práce: Mgr. Jiří Pech, Ph.D.

e-mail vedoucího: pechj@pf.jcu.cz

Abstrakt:

Cílem této bakalářské práce je vytvořit uživatelskou příručku pro tvorbu maker v OpenOffice.org. V práci je kladen důraz na uvedení do problematiky tvorby maker v aplikaci OpenOffice.org, na podrobný popis jazyka OpenOffice.org Basic a jeho jednotlivých prvků. Na konci práce je stručně probrán i aspekt objektového programování v tvorbě maker.

Vše je názorně předvedeno a popsáno na jednoduchých názorných příkladech (programech), které jsou spolu s kompletní instalací OpenOffice.org a dalšími dostupnými výukovými materiály přiloženy v příloze na CD.

Klíčová slova: Příručka, tvorba, Basic, programování, příklady

Title: OpenOffice.org - programming of macros

Author: PETR ŠPIROCH

Department: Katedra Informatiky

Supervisit: Mgr. Jiří Pech, Ph.D.

Supervisor´s e-mail address: pechj@pf.jcu.cz

Abstract:

The main aim of this B. A. Thesis is to write the user‘s guide for the creation of the macros in OpenOffice.org. The stress is laid mainly on the introduction to the problems of the macro creation in the OpenOffice.org application, and on the detailed description of OpenOffice.org Basic, the programming language, and its individual components. The last part of the work consists of the brief analysis of the utilization of the object programming in the macro creation.

Everything is clearly described and demonstrated by easy and illuminating examples (programs) that are included on the enclosed CD. It contains also the complete installation of OpenOffice.org and some other available educational materials.

Keywords: guide, creation, Basic, programming, examples

Poděkování

Děkuji Mgr. Jiřímu Pechovi, Ph.D. za informace, trpělivost, rady a materiály, které mi poskytl během vypracovávání bakalářské práce.

Prohlašuji, že jsem předloženou bakalářskou práci vypracoval samostatně a použil jen pramenů, které cituji a uvádím v seznamu použité 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 Č.Budějovicích 27.4. 2007 .....

Podpis

Obsah

Seznam obrázků

Seznam tabulek

Seznam příkladů

Úvod

Cílem této bakalářské práce je vytvořit uživatelskou příručku, která čtenáři přiblíží, co jsou to makra a popíše práci s nimi v kancelářském balíku OpenOffice.org. Příručka by měla být určena pro čtenáře, který nemá s programováním maker zkušenosti, avšak ovládá základní dovednosti v OpenOffice.org. Čtenář se v této práci postupně dozví, jak si nahrát a posléze i naprogramovat makra, základy pro práci s jazykem Basic, a nakonec i použití maker v dokumentech. Všechny kapitoly by měly být doplněny o malé samostatné programy s popisem jednotlivých prvků programu.

V první části bude vysvětleno co je to OpenOffice.org a makro. Budou zde všeobecné informace o makrech a aplikacích OpenOffice.org.

V druhé, konkrétnější části by se měl čtenář dozvědět jakým způsobem se tvoří a ukládají makra v OpenOffice.org, jak lze makra spouštět a přiřadit do ovládacích prvků OpenOffice.org. Budou zde popsány postupy krok za krokem, aby se i čtenář, který se pohybuje v OpenOffice.org poprvé, dovedl orientovat.

Ve třetí části, která se bude věnovat podrobně jazyku Basic, se dozví čtenář zásady a pravidla pro tvorbu maker. Budou zde rozebrány jednotlivé základní prvky jazyka, jako jsou proměnné, funkce a příkazy. Všechny podkapitoly budou doplněny o názorné příklady, na kterých budou prakticky vysvětleny jednotlivé prvky jazyka Basic.

Poslední čtvrtá část této bakalářské práce bude pojednávat o objektovém programování. Tato oblast programování maker je velmi rozsáhlá a sama o sobě by vydala na další dvě podobné práce, proto se omezím na nejpoužívanější metody pro práci s objekty a na popis, kde lze nalézt další informace.

Celá práce by měla být doplněna o přílohu ve formě CD, kde budou k dispozici nejenom všechny zde použité programy, ale i nejnovější kompletní instalace OpenOffice.org a další dostupné materiály pro studium programování maker v OpenOffice.org.

1.0 Co to jsou makra a OpenOffice.org

V dnešní době existuje pro práci s počítačem velká řada programů, které mohou mít nejrůznější určení. Může se jednat o **specializované programy**, které se většinou zaměřují na určitou problematiku, nebo se může jednat o **univerzální programy**, které jsou určeny pro široké použití. U specializovaných programů je výhodou, že oproti univerzálním programům dokáží řešit problémy, které univerzální neumí – jsou tzv. „šité na míru“. Z toho ovšem vyplývá i jejich velká nevýhoda – a to je právě jejich jednostrannost - nelze je použít k ničemu jinému a navíc náklady na jejich pořízení bývají dosti vysoké. Oproti nim stojí právě zmiňované univerzální programy, které mají velmi široké použití. Toto je ale vykoupeno právě tím, že zde nenajdeme vždy ty funkce, které bychom pro řešení svého problému potřebovali. Velmi široce používané jsou tzv. kancelářské balíky, které zpravidla ve svém základu obsahují textový editor, tabulkový procesor a prezentační program. Jedním z takových balíků je i sada OpenOffice.org. Její hlavní výhodou je nejspíše to, že je volně přístupná a tudíž uživatel nemusí vynaložit žádné finanční prostředky na její pořízení.

Univerzální programy mají docela velkou historii a prošly dost velkým vývojem. Proto disponují již velmi slušnou paletou nástrojů. Při řešení konkrétního (můžeme říci „specializovaného“) problému ovšem musíme jednotlivé příkazy zadávat znovu a znovu (klikem myší, či stisknutím kláves). Z toho plyne, že rutinní operace, které se často opakují, se mohou stát únavné a dosti časově náročné. Ale i na toto je u univerzálních programů myšleno a existuje u nich možnost jejich modifikace. Taková modifikace spočívá v tom, aby se univerzální program co nejvíce přiblížil specializovaným programům. Touto možností je právě **použití maker**.

Nejprve je třeba si říci, co to makra vlastně jsou. Slovo **makro** vzniklo ze slova **makroinstrukce**. Počítač pracuje na základě zadaných instrukcí. Instrukcí může být stisk klávesy, tlačítka myši, nebo výběr z panelu nástrojů. Instrukcí může být také příkaz napsaný v programovacím jazyku. A soubor takovýchto instrukcí tvoří makroinstrukci – tedy naše makro. **Makra umožňují automatizovat rutinní operace**, usnadňují zpracování a vyhodnocování dat. Uživatel ocení význam maker hlavně při práci s rozsáhlejšími datovými soubory. Pokud bychom to přeložili do řeči

naprostého laika, tak makra v kancelářském balíku OpenOffice.org jsou programy, které mají za úkol usnadnit a zjednodušit uživateli práci a rozšiřují možnosti kancelářských programů nad jejich základní funkce. Při tom i obyčejný uživatel si může s trochou znalostí sám makro nahrát či naprogramovat. Tyto programy mohou být maličké, jako je například vložení datumu a času, nebo mohou být i dosti rozsáhlé. Příkladem rozsáhlého makra může být francouzský makromodul Dmaths [www.dmaths.com], pomocí kterého můžeme mimo jiné vkládat do textového dokumentu grafy funkcí.

Protože práce pojednává o programování maker v kancelářském balíku **OpenOffice.org**, tak bychom měli o tomto balíku také něco vědět:

Podle prohlášení o hlavním cíli celého projektu se OpenOffice.org snaží „*Za spolupráce v komunitě vytvořit přední mezinárodní kancelářskou sadu nástrojů, která bude pracovat na všech hlavních platformách a poskytovat přístup k veškerým funkcím a datům pomocí API, založeného na otevřených komponentách a souborového formátu, založeného na XML*.“ [1]

OpenOffice.org se snaží konkurovat kancelářskému balíku Microsoft Office a napodobit způsob práce s ním tam, kde je to vhodné. Jedná se o **Open Source program** , který je poskytován zcela **zdarma** a na vývoji tohoto programu se může podílet v podstatě kdokoliv, kdo bude přínosem. Dokáže zapisovat a číst většinu formátů používaných v Microsoft Office a mnoha jiných aplikacích, což je nezbytná funkce pro mnoho uživatelů. Plně podporuje dokumenty ve formátu OpenDokument – což jsou standardizované otevřené formáty (.odt, .ods, .odg, .odp, …), které nejsou vázány na žádného výrobce. Také dokáže dokumenty exportovat rovnou do formátu PDF.

Určitým nedostatkem OpenOffice.org je, že nemohou spouštět makra vytvořená v Microsoft Office. MS Office používá jazyk VBA (Visual Basic for Applications) a OpenOffice.org používá jazyk Basic založený na prostředí OpenOffice.org API (aplikační programové rozhraní). I když je základní programovací jazyk stejný (Basic), objekty a metody se od sebe navzájem liší. Vezmeme-li to z druhé strany pohledu, tak je nekompatibilita nejen nevýhodou, ale i určitou výhodou – a to z hlediska náchylnosti na viry. MS Office jsou známy svojí náchylností na tzv. makroviry. Oproti tomu stojí OpenOffice.org. Do nedávné doby

nebyl znám žádný makrovirus, který by napadal dokumenty OpenOffice.org. Až na konci roku 2006 byl objeven první a doufejme že na dlouhou dobu poslední makrovirus, který by mohl napadnout i dokumenty tvořené v OpenOffice.org.

Součástí balíku OpenOffice.org jsou aplikace:

- Writer – textový procesor
- Calc – tabulkový procesor
- Impress – prezentační nástroj
- Draw – grafický editor
- Base – databázový frontend
- Math – nástroj pro vytváření matematických vzorců
- Quickstarter – program pro MS Windows, který nahrává knihovny a důležité kusy kódu do paměti předem, aby se díky tomu spustil program rychleji.

„Předchůdce tohoto balíku, kancelářský balík StarOffice vyvíjí od roku 1994 německá firma StarDivision. Od verze 4.2 je vyvíjen na platformově nezávislé knihovny jazyka C++ StarView. V roce 1999 jej koupila společnost Sun Microsystems, výrobce počítačů s procesorem SPARC a s operačním systémem Solaris. Sun poté zveřejnil téměř celý zdrojový kód StarOffice a ten je nyní vyvíjen jako OpenOffice.org komunitou nezávislých vývojářů a poskytován zdarma pod licencí z rodiny GPL podobně jako například aplikace projektu Mozilla. StarOffice nadále existují a obsahují kromě kódu OpenOffice.org i některá další rozšíření chráněná autorskými právy. Cílem kroků Sunu bylo vytvořit srovnatelnou alternativu k Microsoft Office.“ [1]

[1] *Wikipedie* [online]. 2001-2007 [cit. 2007-01-15].
Dostupný z WWW: http://cs.wikipedia.org/wiki/Openoffice

2.0 Záznam a uložení makra v OpenOffice.org

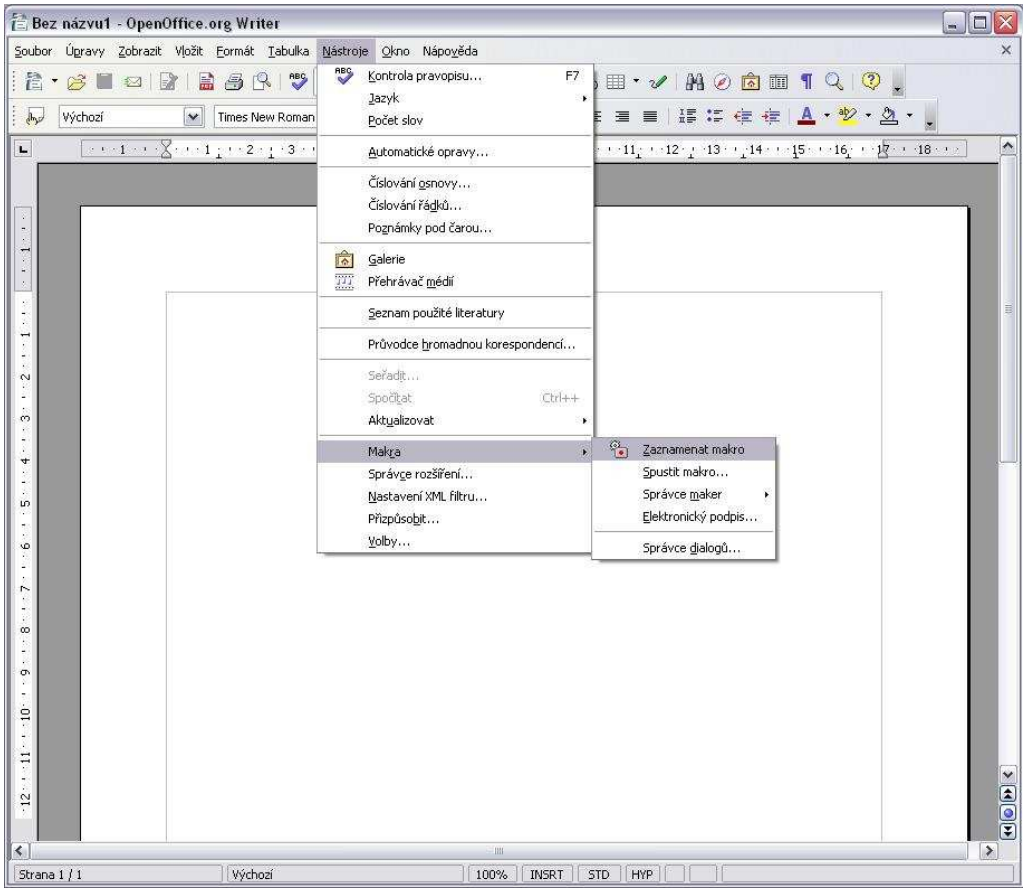
2.1 Záznam makra v záznamníku maker

Jak jsme se v úvodu dozvěděli, tak makro je vlastně soubor instrukcí. Makro (makroinstrukce) může tvořit soubor příkazů napsaných v programovacím jazyce. Instrukcí může být také pouze stisknutí klávesy či výběr z panelu nástrojů. Proto existují v OpenOffice.org dvě základní možnosti tvorby maker – a to je přímo naprogramování samotného makra příkaz po příkazu, nebo nejjednodušším způsobem je vytvoření makra s využitím záznamníku maker. Zde se dají tvořit malá a jednoduchá makra, která nám pomáhají při těch nejrutinnějších operacích.

V malé ukázce si vytvoříme makro, které nám v textovém editoru **Writer** napíše text MOJE PRVNÍ MAKRO a vycentruje ho doprostřed řádky. Nejdříve spustíme textový editor balíku OpenOffice.org. Po instalaci kancelářského balíku se vytvořily ikony v nabídce **START** – zde vybereme **OpenOffice.org → Writer**. Vytvoří se nám textový dokument s názvem **Bez názvu1**. V panelu nabídek vybereme příkaz **Nástroje → Makra** a v rozbaleném menu stiskneme příkaz **zaznamenat makro** (viz obrázek 2.1).

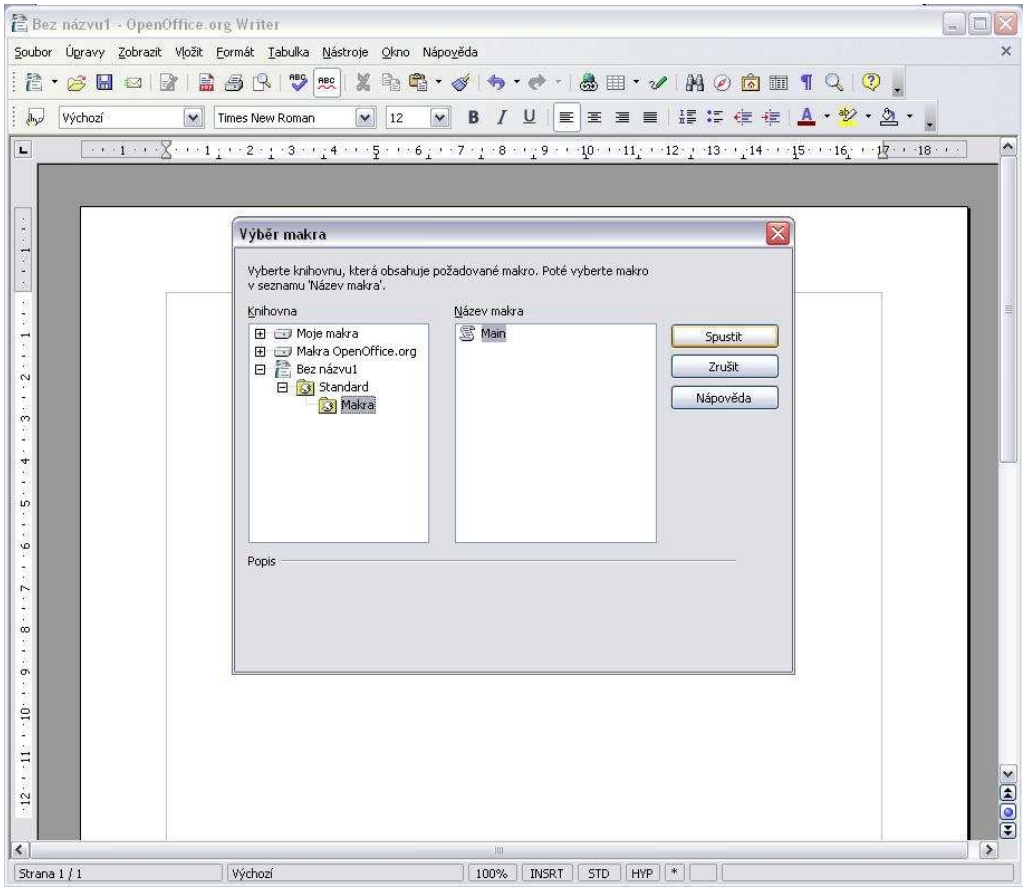
Zobrazí se malé dialogové okno **Zaznamenat…** s jediným tlačítkem zastavit nahrávání. Od této chvíle nahráváme naše akce. Nezaznamenává se čas, ale pouze akce, které vykonáme a jejich posloupnost. V nabídce vybereme **Formát → Zarovnání → Na střed**. Kurzor v dokumentu se přesune doprostřed řádky. Nyní na klávesnici stiskneme klávesu **Caps Lock**, čímž nastavíme psaní velkých písmen a napíšeme text **MOJE PRVNÍ MAKRO**. Záznam makra ukončíme stisknutím tlačítka **Zastavit nahrávání** v malém dialogovém okně. Zobrazí se dialogové okno s názvem **Makra v OpenOffice.org**. V tomto okně určíme, kam se zaznamenané makro uloží a jeho název. V levém dolním okně s názvem **Uložit makra do** poklikáním otevřeme soubor **Bez názvu1** a vybereme řádek **Standard**. Dále klepneme na tlačítko **Nový modul** a otevře se nám stejnojmenné dialogové okno. Místo názvu **Module1**, který je nám nabídnut, napíšeme text např. „**Makra**“ a potvrdíme tlačítkem **OK**. Nyní nám

pod řádkem Standard přibyl nový řádek s právě napsaným názvem Makra. Uložení makra dokončíme klepnutím na tlačítko **Uložit**. Na dotaz, zdali chceme přepsat již existující makro Main, odpovíme stisknutím tlačítka **ANO**.



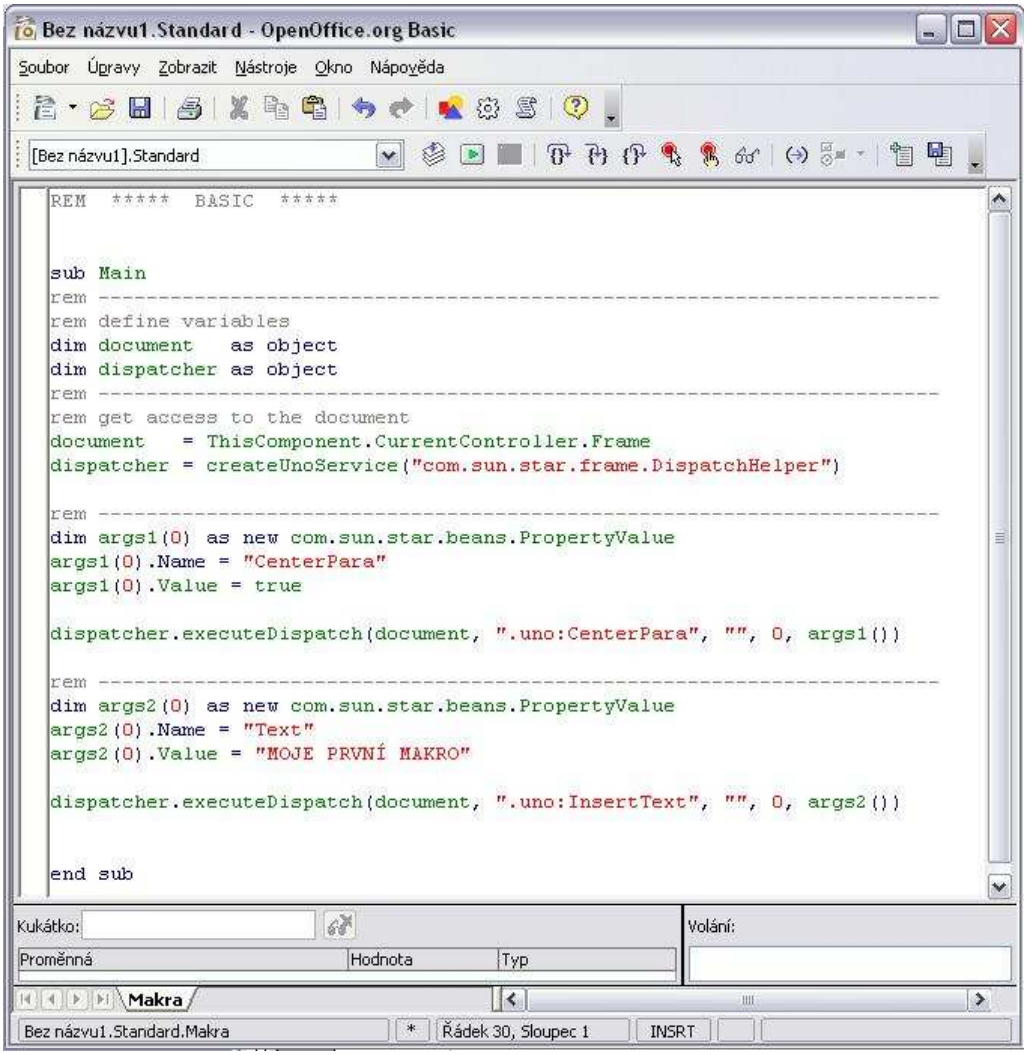
Obr. 2.1: Zaznamenání makra

Nyní jsme úspěšně vytvořili a uložili své první makro v OpenOffice.org. Zkusíme, zda makro opravdu funguje. Nejdříve smažeme vše, co jsme právě vytvořili. Kurzor máme tedy na začátku první řádky a list dokumentu by měl být úplně prázdný. S využitím příkazu **Nástroje → Makra → Spustit makro…** zobrazíme dialogové okno **Výběr makra**. V levém okně **Knihovna** vybereme soubor **Bez názvu1** a klepnutím na řádek **Standard** zobrazíme řádek **Makra**. V pravém okně se zobrazí řádek s názvem **Main**. Po klepnutí na tlačítko **Spustit** se makro spustí a provedou se stejné akce, jaké jsme nahráli při nahrávání.



Obr.2.2: Výběr makra pro spuštění

Potvrdili jsme si funkčnost našeho makra a nyní se podíváme, jak vlastně vypadá jeho zdrojový kód. Příkazem **Nástroje → Makra → Správce maker → OpenOffice.org Basic…** zobrazíme nám již známé okno **Makra**. V levé části opět vybereme soubor **Bez názvu1**, výběrem řádku **Standard** zobrazíme řádek **Makra**. V pravém okně se zobrazí řádek **Main**. Tentokrát klepneme na tlačítko **Upravit**. Zobrazí se jednoduchý textový editor s kódem makra, jak je vidět na obrázku 2.3.



Obr.2.3: Zobrazení kódu makra

Tak takto vypadá naše jednoduché makro v jazyce počítače - konkrétně v jazyku OpenOffice.org Basic. Možná překvapí délka textu, potřebná k napsání pouze tří slov. Tímto se nenechme odradit, protože jsme právě vytvořili své první jednoduché makro a i takováto jednoduchá makra nám mohou značně ušetřit práci. Makra lze vytvářet nejen v textovém editoru, ale také například v tabulkovém procesoru **Calc** balíku OpenOffice.org, kde si praktické použití dovedeme asi lépe představit. Postup záznamu je obdobný jako u textového editoru.

V textu programu na obr.2.3 si všimněme, že zápis makra začíná slovíčkem **Sub**, za nímž následuje **název makra Main** a končí slovy **End Sub**. Slovo **Sub** je

zkratkou anglického slova subroutine, což v češtině znamená slovo podprocedura. **Sub značí začátek podprocedury** a **End Sub značí konec podprocedury**. Protože jazyk OpenOffice.org Basic nerozlišuje pojmy jako procedura a podprocedura, slovo podprocedura se většinou nepoužívá a hovoříme o nich jako o procedurách.

2.2 Konfigurace OpenOffice.org

V této podkapitole si ukážeme možnosti zpřístupnění již hotového makra, tak abychom nemuseli místo uložení hojně používaného makra složitě vyhledávat a neustále spouštět z tohoto místa. Mimo tohoto způsobu lze makro v OpenOffice.org zpřístupnit prakticky odkudkoliv. Tomuto účelu je určená volba v menu **Nástroje → Přizpůsobit**. Otevře se dialogové okno se stejným názvem. Okno je tvořeno čtyřmi záložkami (kartami), ve kterých můžeme měnit vzhled a některé funkce celého OpenOffice.org. Zde se samozřejmě budeme zaobírat pouze zpřístupněním a zrušením makra v jednotlivých možnostech.

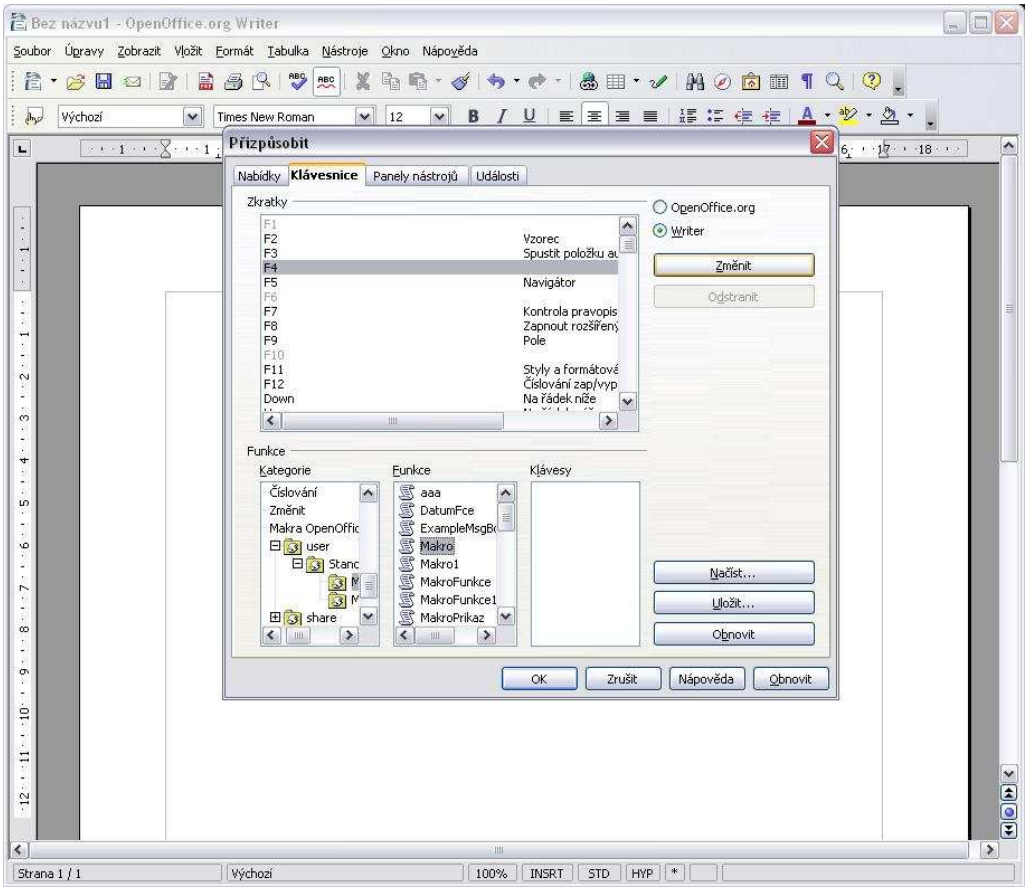
2.2.1 Přiřazení makra do nabídky (do menu)

Záložka **Nabídky** se skládá za dvou seznamů. V prvním si vybíráme nabídku, do které chceme umístit odkaz na naše makro, v druhém se nám zobrazuje vlastní obsah nabídky. Přejdeme do prvního seznamu na místo, kam chceme vložit odkaz na naše makro (např. Nástroje | Makra) a stiskneme tlačítko **Přidat**. Zobrazí se další dialogové okno **Přidat příkazy,** kde v jeho levém okně „Kategorie“ sjedeme posuvníkem až dolu a poklikáme na řádek „Makra OpenOffice.org“. Zde vybereme nám již známé umístění makra „Bez názvu1 → Standard → Makra“ a v pravém okně makro „Main“. Stiskneme tlačítko **Přidat.** Nyní můžeme okno zavřít. V okně „**Přizpůsobit**“ se nám v druhém spodním seznamu odkaz na naše makro objevil. Okno tlačítkem **OK** zavřeme a můžeme vyzkoušet, zda se nám umístění odkazu povedlo. V nabídce vybereme **Nástroje → Makra** a v otevřeném okně opravdu vidíme, že se zde naše makro Main opravdu objevilo. Po stisknutí řádku by se mělo makro spustit a vypsat námi nahraný text MOJE PRVNÍ MAKRO. Odstranění tohoto odkazu z panelu nabídek dosáhneme takto: **Nástroje → Přizpůsobit**. V horním okně vybereme nabídku, do které jsme makro umístili (v našem případě Nástroje | Makra) a

v dolním okně klikneme na řádku s názvem makra (Main). Stiskneme tlačítko **Upravit** a v rozbalené podnabídce řádku **Odstranit**. Dialogové okno zavřeme tlačítkem OK a je hotovo.

2.2.2 Přiřazení makra klávese

Na začátek je nutno uvést, že klávesové zkratce lze makro přiřadit pouze pokud je makro tzv. globální – tzn. pokud je uloženo pro celý OpenOffice.org. Protože jsme si své první makro uložili pouze pro náš otevřený dokument Bez názvu1, tak toto makro v nabídce pro klávesové zkratky nenajdeme. Ale makro již zaznamenat umíme, tak si své „globální“ makro vytvoříme. Jediným rozdílem bude, že toto makro v okně „Makra v OpenOffice.org“ neuložíme do „Bez názvu1 → Standard“, ale do „**Moje makra → Standard → Module1**“ a pojmenujeme ho např. Makro1.



Obr.2.4. Přiřazení makra klávese

Nyní můžeme přejít k přidání tohoto makra klávesové zkratce. V dialogovém okně „**Přizpůsobit**“ (otevřeme ho Nástroje → Přizpůsobit) se přesuneme o záložku doleva k záložce **Klávesnice**. Zde se v horním okně ukázaly klávesové zkratky (v levé části okna) a jejich funkce, jsou-li k nim přiřazeny (v pravé části okna). Pokud napravo od klávesové zkratky není nic uvedeno, pak je tato klávesová zkratka volná. V dolní části se zobrazily tři okénka. V levém sjedeme posuvníkem dolu a vybereme řádku **Makra OpenOffice.org**, poklikáme na řádku **User**, dále na **Standard** a nakonec na **Module1**. V prostředním okénku se nám objevil námi uložený název „globálního“ makra **Makro1.** Klikneme na něj a v horním okně s názvem „Zkratky“ si vybereme volnou klávesu (např. F4). Klikneme na tlačítko **Změnit** a v okně vidíme, že se makro ke zkratce přiřadilo. Přiřazení ke zkratce můžeme odstranit tak, že vybereme zkratku a stiskneme tlačítko **Odstranit**. Před smazáním si ale funkci zkratky ověříme. Okno zavřeme tlačítkem **OK** a po stisku klávesy **F4** zjistíme, že makro je opravdu spuštěno. Nyní můžeme přiřazení ke zkratce smazat.

2.2.3 Přiřazení makra do panelu nástrojů

Přiřazení do panelu nástrojů je obdobné jako přiřazení do nabídky (do menu). Také zde jsou dva seznamy, v prvním si vybereme, kam chceme makro přiřadit a v druhém (spodním) se nám ukazuje stav. Zkusíme si makro přiřadit do panelu nástrojů Standardní. Stiskneme tlačítko **Přidat**. Otevře se okno „**Přidat příkazy**“. Zde v levém okně sjedeme posuvníkem dolu, vybereme **Makra OpenOffice.org** – zde najdeme své makro a stiskneme **Přidat**. Nyní okno zavřeme. V dolním okně se nám naše makro objevilo. Můžeme k němu také přiřadit ikonku – to uděláme tak, že klikneme na naše makro v dolním okně, vpravo stiskneme tlačítko **Upravit** a v podnabídce vybereme řádek **Změnit ikonu**. V novém okně si vybereme ikonku, která nám nejvíce vyhovuje a stiskneme **OK**. Okno „**Přizpůsobit**“ zavřeme tlačítkem **OK**. Nyní se v dokumentu pod nabídkou menu objevila námi vybraná ikonka, na kterou klikneme-li, spustí naše makro. Smazání přiřazení makra k panelu nástrojů je stejné jako u přiřazení k nabídce.

2.2.4 Přiřazení makra události

Znovu spustíme dialogové okno „**Přizpůsobit**“ a přesuneme se do záložky **Události**. Zde máme vypsané události, kterým lze makro přiřadit. V praxi si můžeme

představit přiřazení makra, které vloží záhlaví a zápatí s číslováním stránek při spuštění události vytvořit dokument. Makro přiřadíme tak, že si vybereme událost, které chceme makro přiřadit a po stisknutí tlačítka **Makro…** vybereme příslušné makro. Po potvrzení výběru se nám příslušná akce – makro zobrazí vlevo vedle události, ke které jsme ji přiřadili. Smazat takovouto akci lze pomocí tlačítka **Odstranit**.

2.3 Zápis a uložení makra

Makra jsou tedy, jak jsme se již dověděli, vlastně procedury (podprocedury). Procedury se ukládají jako jiná data. Pro přehlednost byly vytvořeny tři úrovně ukládání – všechna **makra se zapisují do modulů, moduly se ukládají do knihoven a knihovny do knihovních kontejnerů**. Knihovní kontejner se vytváří automaticky pro každý nový dokument. Svůj vlastní knihovní kontejner má i samotná aplikace OpenOffice.org. Počet procedur v modulu, počet modulů v knihovně a počet knihoven v kontejnerech není omezen. Názvy maker v modulech, modulů v knihovnách a knihoven v kontejnerech by měly být jednoznačné, abychom předešli chybám.

2.3.1 Pravidla pro ukládání do kontajnerů

Při každém vytvoření dokumentu se automaticky vytváří spolu s dokumentem i **kontejner** stejného názvu a je přiřazen příslušnému dokumentu. **Knihovny, moduly** a samozřejmě i **makra** uložené v tomto kontejneru jsou tedy dostupná a funkční pouze z tohoto dokumentu a v jiných dokumentech či aplikacích balíku OpenOffice.org nejsou přístupná. Tento **kontejner** obsahuje základní **knihovnu** s názvem **Standard**. **Knihovnu Standard** zpravidla nepoužíváme. Vytváříme si vlastní knihovny, protože knihovna se stejným názvem je obsažena v každém kontejneru a použití stejně označených knihoven by mohlo vést k chybám.

Svůj knihovní kontejner má i aplikace OpenOffice.org. Kontejner se nazývá **Moje makra a dialogy**. Pokud makro uložíme do tohoto aplikačního kontejneru, bude toto makro přístupné ze všech dokumentů dané aplikace. V předchozí kapitole jsme si pro představu takovéto makro nazvali „globálním“makrem. Pokud ale přeneseme příslušný dokument z počítače do jiného, makra se s ním nepřenesou, protože nebudou v tomto dokumentu a jeho kontejneru uloženy. To povede k chybám, protože makra

nebudou v dokumentu k dispozici. Tomu předejdeme uložením maker přímo do kontejneru příslušného dokumentu. Potom budou makra i po přenesení dokumentu jinam přístupné, protože spolu s dokumentem se přenáší i jeho knihovní kontejner. Nevýhodou tohoto postupu je zase naopak to, že makra nejsou přístupná pro jiné dokumenty. Proto si musíme při ukládání makra dobře rozmyslet, pro jaký účel a rozsah použití je naše procedura určená.

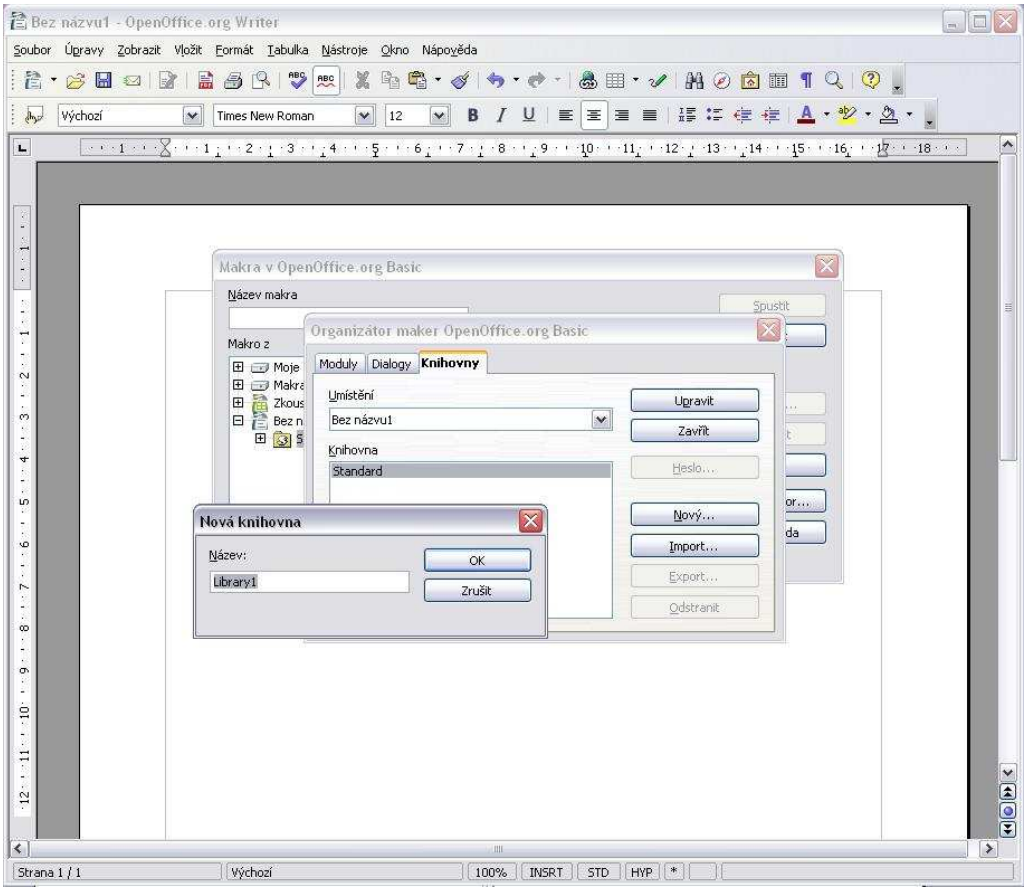
2.3.2 Vytváření, kopírování a mazání knihoven a modulů.

S moduly a knihovnami se pracuje stejně jako se složkami a soubory ve Windows. Můžeme je vytvářet, mazat, kopírovat a přesouvat.

Pokud chceme knihovnu nebo modul tvořit, mazat nebo kopírovat mezi dvěmi dokumenty, budeme používat dialogové okno **Organizátor maker OpenOffice.org**. K tomuto oknu se dostaneme z námi již známého okna **Makra v OpenOffice.org Basic** (Nástroje → Makra → Správce maker → OpenOffice.org Basic). K našemu cíli se dostaneme přes tlačítko **Organizátor…**. Nyní máme otevřené naše cílové okno. Zde vidíme, že obsahuje tři záložky (karty) – „**Moduly**“, „**Dialogy**“ a „**Knihovny**“. Nás bude zajímat pouze první a poslední záložka, záložku „Dialogy“ pro svou práci potřebovat nebudeme. Pokud se podíváme do záložky „**Knihovny**“, tak nahoře vidíme rozbalovací okno „**Umístění**“. Rozbalíme-li si ho, zobrazí se seznam možných umístění knihoven – vlastně se zobrazí dostupné kontajnery. Je zde nám již známý aplikační kontejner **Moje makra a dialogy**. Dále zde určitě najdeme náš dokumentový kontejner **Bez názvu1**, pokud máme momentálně otevřeny v OpenOffice.org i jiné dokumenty, tak se zde zobrazí i jejich kontejnery. Najdeme zde také umístění **Makra a dialogy OpenOffice.org** – toto umístění obsahuje již hotová makra, která se nám nainstalovala spolu s instalací OpenOffice.org. Tohoto umístění si nebudeme všímat. Pod oknem umístění se v okně „Knihovna“ zobrazují aktuální knihovny, které se v daném umístění vyskytují.

Pro vytvoření knihovny nám slouží tlačítko **Nový…**. Po stisknutí tohoto tlačítka pouze zadáme do okna, které se objeví, název nové knihovny a knihovna se v daném umístění vytvoří. Knihovnu můžeme posléze také mazat – mazanou knihovnu vlevo vybereme a stiskneme tlačítko **Odstranit…**. Potvrdíme a knihovna je smazána. Ještě zde existuje tlačítko **Import…**. Tímto tlačítkem můžeme importovat knihovnu z

dokumentu do dokumentu. Správně řečeno z kontejneru jednoho dokumentu do kontejneru druhého dokumentu. Jak jsme se již dozvěděli, tak kontejner dokumentu je vždy připojen k dokumentu samotnému. Po stisknutí tlačítka **Import…** tedy stačí pouze najít cestu k dokumentu, z kterého chceme knihovny překopírovat a potvrdit. Pozor na jednoznačnost názvů knihoven! – Pokud bude mít knihovna, kterou chceme přenést, stejný název jako některá z knihoven v našem dokumentu, nastane chyba a knihovna se nám nepřekopíruje. Tlačítko **Export…** slouží k exportování knihovny z našeho dokumentu do jiného. Platí zde obdobný postup jako u importu.



Obr.2.5: Vytvoření nové knihovny

Pro práci s moduly přejdeme do záložky „Moduly“. Zde můžeme vytvářet nebo odstraňovat moduly v knihovnách. V okně vybereme cestu, kde chceme s moduly pracovat a pomocí dvou tlačítek **Nový…** a **Odstranit…** moduly buďto vytváříme či mažeme.

2.3.3 Kopírování knihovny mezi dokumentem a aplikačním knihovním kontajnerem

Kopírování knihovny z dokumentu do aplikačního knihovního kontejneru probíhá stejně jako kopírování mezi dokumenty přes tlačítko **Import…**, pouze jako cílový knihovní kontejner vybereme **Moje makra**.

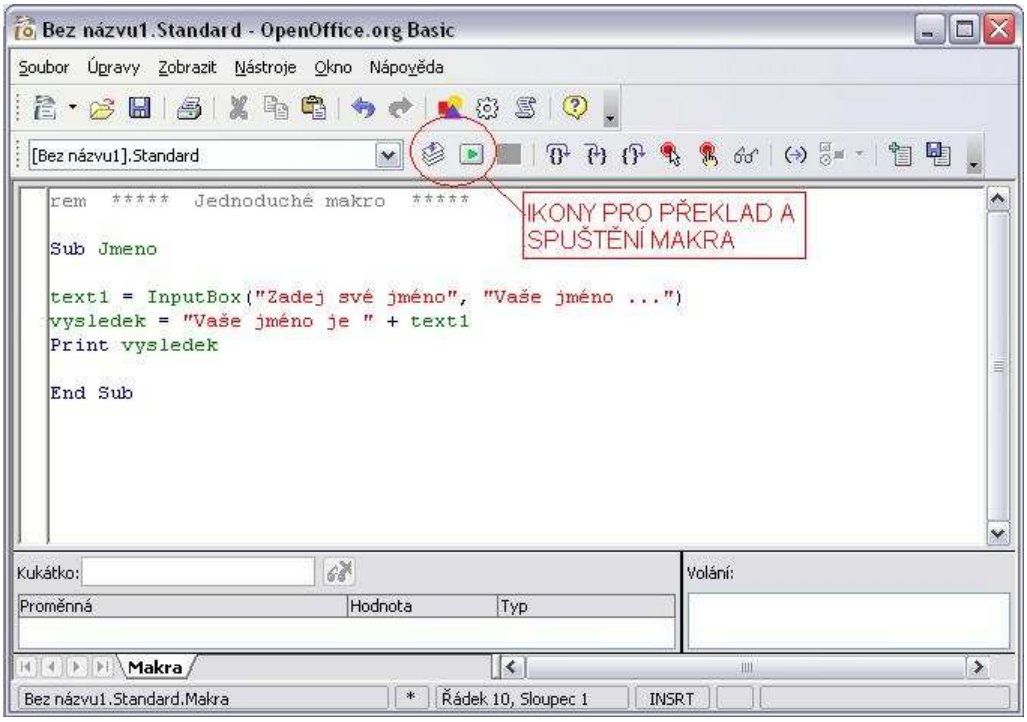
Kopírování knihovny z aplikačního knihovního kontejneru do dokumentu je v podstatě také stejné, pouze musíme znát umístění aplikačního knihovního kontejneru v počítači. Po standardní instalaci OpenOffice.org ve verzi 2.2 do Windows XP je cesta ke složce obsahující kontejner **C:\Documents and Settings** ***(zde bude jméno uživatele)*** **\Data aplikací\OpenOffice.org2\user\basic.** Tato složka **basic,** vytvořená již při instalaci programu, je právě námi hledaný aplikační knihovní kontejner. Když se do této složky podíváme, tak zde najdeme dva soubory – skript.xlc a dialog.xlc a dále složky nesoucí názvy knihoven vytvořených uživatelem. Soubor **script.xlc** obsahuje přehled všech dostupných aplikačních knihoven, v nichž jsou uloženy data. Soubor **dialog.xlc** slouží pro evidenci dialogů. Když otevřeme složku některé z knihoven, najdeme zde také tyto dva soubory, akorát s jinou příponou - ***.xlb**. Soubor, který nás zajímá je soubor **script.xlc** ve složce **basic**.

Nyní, když známe cestu k aplikačnímu knihovnímu kontejneru, můžeme kopírovat. V dialogovém okně „Organizátor maker OpenOffice.org“ opět vybereme umístění kam chceme kopírovat a tlačítkem **Import…** vyvoláme okno pro cestu ke knihovnám. Nyní nalezneme náš soubor **C:\Documents and Settings** ***(zde bude jméno uživatele)*** **\Data aplikací\OpenOffice.org2\user\basic\script.xlc**. Potvrdíme, z nabídky vybereme knihovnu, kterou chceme kopírovat a znovu potvrdíme. Vrátíme se do dialogového okna „Organizátor maker“ a zde v přehledu se nová knihovna zobrazí. Kopírování bylo tedy úspěšné.

3.0 Programování v OpenOffice.org Basic

V předchozí kapitole jsme si ukázali, jak vytvořit makro pomocí záznamníku maker. Krom tohoto jednoduchého způsobu ještě existuje způsob přímé tvorby procedur a tím je práce v **integrovaném vývojovém prostředí IDE**. OpenOffice.org již má takovéto prostředí v sobě integrováno. V podstatě jsme již v předchozí kapitole nevědomky do tohoto prostředí nahlédli a to v případě, kdy jsme si záznam našeho nahraného makra prohlíželi.

Znovu si vytvoříme dokument OpenOffice.org s názvem Bez názvu1, tentokrát ale jako tabulkový dokument. V nabídce Start nalezneme OpenOffice.org Calc a spustíme tabulkový procesor. Na panelu nabídek vybereme příkaz **Nástroje → Makra → Správce maker → OpenOffice.org Basic**, otevře se dialogové okno **Makra**. V levé části otevřeme soubor **Bez názvu1**, vybereme řádek **Standard** a klepnutím na tlačítko **Nový…** zobrazíme okno **Nový modul**. Zde napíšeme název nového modulu např. **Makra** a potvrdíme tlačítkem **OK**. Zobrazí se nám vývojové prostředí IDE s modulem Makra. Název modulu je uveden ve spodní záložce. V tomto prostředí je možno makra tvořit, ověřovat, upravovat, ukládat a spouštět. K tomu potřebujeme ovšem znát něco o jazyku Basic, který toto IDE prostředí využívá. Základy jazyka Basic se dozvíme v následujících kapitolách. Nejprve ale zkusíme napsat naše první makro v prostředí IDE – v textovém okně vymažeme všechen text a místo něj zapíšeme text příkladu z obrázku 3.1. Nyní je třeba příklad zkompilovat (přeložit pro počítač). K tomu slouží ikonka tří listů papíru se šipkou nad. Najdeme ji v panelu nástrojů hned vedle rozbalovacího okna. Stisknutím přeložíme a ikonou hned vedle (zelený trojúhelník ve čtverci (play)) makro spustíme. Objeví se okno, které vyzývá k zadání jména. Napíšeme text a stiskneme OK. Objeví se další okno s textem Vaše jméno je …. Makro tedy funguje. Základní popis programu, který po spuštění vyzve uživatele k zadání jména do dialogového okna a po potvrzení toto jméno vypíše do jiného dialogového okna, je uveden pod obrázkem 3.1.



Obr.3.1: Překlad a spuštění makra

Popis příkladu z obrázku 3.1 :

řádky začínající slůvkem **rem** jazyk Basic vynechává – slouží nám jako komentáře, většinou popisující program.
Procedura opět začíná slovem **Sub**, za kterým je jméno procedury a končí slovy **End Sub**. Mezi těmito klíčovými slovy se nachází tělo procedury.
V těle procedury vidíme proměnnou s názvem **text1** a jí je přiřazen výsledek funkce **InputBox()** – tato funkce čte vstup z klávesnice.
Dále je zde další proměnná **vysledek**, které je přiřazen text v uvozovkách + hodnota proměnné **text1** a na poslední řádce těla je příkaz **Print vysledek**, který nám vrátí hodnotu proměnné **vysledek** na obrazovku ve formě dialogového okna.

3.1 Zásady pro tvorbu maker

Makra mohou být jednoduchá, ale také velmi složitá. Jsou to vlastně různě velké bloky instrukcí, zajišťující určité činnosti. Proto mohou být velmi rozsáhlá a nepřehledná. Tomuto se jako programátoři snažíme předejít a hojně využíváme vlastnosti maker a to je větvení. Znamená to, že makra mohou navzájem spolupracovat. Proto se snažíme psát ne příliš rozsáhlá makra a složitější úkol rozdělit

na několik dílčích úkolů, které navzájem propojíme. Pro každý takový dílčí úkol tvoříme samostatné makro. Pro přehlednost bývá zvykem takovéto makro okomentovat – připojit ke kódu programu text, který v krátkosti popisuje, co a jakým způsobem vlastně jednotlivé instrukce dělají.

Jak jsme se již dozvěděli, tak každé makro je vlastně procedurou, jejichž text se skládá z několika segmentů. První skupinou jsou **jednoznačné výrazy** (např.klíčová slova), kterým počítač rozumí. Jsou to vybraná anglická slova nebo zkratky označující začátek a konec procedury, všechny příkazy a funkce. Patří sem i operátory, jako je + nebo -, závorky a další znaky např. $. Druhou skupinou jsou **názvy**, které si uživatel (programátor) sám vytváří. Názvy bychom měli volit krátké a popisující činnost, pro kterou jsou vytvářeny. Jazyk Basic nerozlišuje velká a malá písmena, jejich vhodné použití pouze přispívá ke zpřehlednění zápisu. Názvy vytváříme hlavně jako označení proměnných v proceduře. Třetí skupinou jsou **data**, což jsou konkrétní číselné a datové hodnoty.

3.2 Základní pravidla pro zápis procedur

Procedura vlastně obsahuje řádky textu, jehož součástí jsou v podstatě instrukce pro počítač. Po spuštění makra se tyto instrukce postupně plní až do úspěšného konce.

Procedura jasně začíná slovíčkem **Sub** a končí slovy **End sub**. Prostor mezi těmito řádky je vymezen pro text. Psaní procedur má svá pravidla, která je nutno dodržet.

3.2.1 Řádkování a mezery

V textu programu (naší procedury) musí být každý příkaz na svém vlastním řádku. V případě dlouhých příkazů lze řádek pro přehlednost rozdělit tak, že v místě rozdělení napíšeme mezeru a podtržítko _ . Poté můžeme pokračovat v psaní příkazu na dalším řádku. Rozdělení není možné uprostřed slova, pouze v místě mezery.

Basic nerozlišuje mezi mezerou a tabulátorem a také ignoruje prázdné řádky. Pro přehlednost programu tedy můžeme jednotlivé řádky odsazovat a oddělovat dle libosti.

3.2.2 Komentáře

V našem příkladu jsme se setkali s klíčovým slovem **rem**. Toto pro Basic znamená, že vše, co je na řádce za tímto slovem napsáno, ignoruje. Toto využíváme pro psaní komentářů, které slouží pouze nám, a usnadňují nám později práci. Psaní komentářů je v programování zcela samozřejmé a oceníme ho zvláště při psaní dlouhých programů. Nemusíme totiž později, když kód programu procházíme, zjišťovat co jaký řádek dělá – máme to totiž popsané právě v komentářích. Stejně jako slovo **rem** má stejnou funkci i znak apostrof ‘ .

3.2.3 Názvy proměnných a procedur

Názvy proměnných a procedur si většinou určuje programátor sám. Nejsou pro ně žádná psaná pravidla, ale pro přehlednost by měly názvy vystihovat to, co popisují - např. vysledek, celkovaCena, Sirka. Všimněme si, že je jedno jestli název začíná velkým či malým písmenem. Basic totiž velká a malá písmena nerozlišuje a nechává psaní tudíž na nás a našem citu. Zažitou praxí je psát názvy s počátečním velkým písmenem a obsahuje-li název více slov tak tzv. velbloudím stylem (CelkováCena). POZOR! V žádném případě nesmí být dvouslovný název rozdělen mezerou – to Basic přečte jako dva názvy a vyhodí chybu. Nesmí být taktéž stejný jako klíčová slova (např. Sub, End, Print, …). Název může obsahovat 255 znaků, musí začínat písmenem, za kterým může být sled písmen a číslic. Je možno použít též podtržítko _ , které se chová jako písmeno. Též je vhodné nepoužívat českou diakritiku, ta může totiž při překladu vyvolat nemalé problémy.

3.2.4 Příkazy a funkce

S příkazy jsme se v našem programu také již setkali – např. příkaz **End sub** ukončuje proceduru. Slovo **Print** je také příkazem. Tento příkaz zajistí zobrazení dialogového okna. Pokud je za příkazem **Print** mezera a pak text v uvozovkách, zobrazí v tomto okně tento text. V našem příkladu jsme měli za tímto příkazem proměnnou Vysledek, ale této proměnné byl o řádek výše přiřazen text. Obecně lze tedy říci, že příkaz se skládá z názvu příkazu, za kterým následují parametry oddělené čárkou. Parametry mohou být povinné a nepovinné, přičemž pokud obsahuje příkaz obojí, tak na prvním místě je vždy parametr povinný. Příkaz Print má nepovinný

parametr, napíšeme-li tedy pouze Print, zobrazí se dialogové okno bez jakéhokoliv textu.

Funkci jsme již v příkladu také použili – konkrétně ve funkci InputBox(). Tato funkce zajistí zobrazení dialogového okna s textovým polem a po potvrzení tlačítkem OK, přečtení textu z tohoto pole. V našem příkladu jsme tento text přiřadili proměnné Text1 jako jeho hodnotu. Obecně vypadají funkce jako příkazy, ovšem s tím rozdílem, že parametry musejí být uzavřeny v závorkách.

3.2.5 Operátory

Žádný program se neobejde bez operátorů. Jsou to jedny ze základních prvků písma a matematiky všeobecně. I jazyk Basic je obsahuje a využívá je jako všechny ostatní jazyky pro práci s proměnnými. Operátory se dělí na tři skupiny:

3.2.5.1 Matematické operátory

+	Sčítání čísel a datumových údajů, spojování řetězců
-	Odčítání čísel a datumových údajů
*	Násobení čísel
/	Dělení čísel
\	Celočíselné dělení, tzn. zaokrouhlení výsledku na celé číslo
^	Mocnina
MOD	Zbytek po dělení

3.2.5.2 Logické operátory

AND	Logický součin (a)
OR	Logický součet (nebo)
XOR	Exkluzivní logický součet
NOT	Negace
IMP	Implikace
EQV	Ekvivalence

3.2.5.3 Porovnávací operátory

=	Rovnost čísel, řetězců nebo datumových údajů
<>	Nerovnost čísel, řetězců nebo datumových údajů
>	Větší než druhé číslo, řetězec či datumový údaj
>=	Větší nebo roven než druhé číslo, řetězec či datumový údaj

<	Menší než druhé číslo, řetězec nebo datumový údaj
<=	Menší nebo roven než druhé číslo, řetězec či datumový údaj

Použití matematických, logických a porovnávacích operátorů je většinou intuitivní a uvidíme ho v následujících kapitolách ve všech příkladech.

3.3 Základní prvky jazyka Basic

Do teď jsme si vyjmenovali zásady a základní pravidla pro psaní programů (maker) v jazyce Basic. Byly to informace všeobecné a bez příkladů, na kterých bychom viděli konkrétní použití. Nyní se ale dostáváme konečně k samotné kostře programů. Tato kostra se skládá ze základních stavebních prvků, jako jsou proměnné, funkce a příkazy, textové řetězce a operátory, které vše spojují. V neposlední řadě nesmíme zapomenout také na klíčová slova, které program ohraničují a doplňují.

3.3.1 Proměnné

Bez proměnných se neobejdeme v žádném programovacím jazyce. Proměnnou si můžeme představit jako jakousi schránku, která má svůj název a obsahuje nějakou informaci. V běžném životě se také s proměnnými setkáváme – schránku nám např. může představovat cenovka na zboží v obchodě s potravinami a obsah schránky konkrétní cena na cenovce. Cena na cenovce se v průběhu prodeje zboží může měnit. Stejné je to i v programovacím jazyce. Je zde proměnná s názvem „Cenovka“, které je přiřazena nějaká hodnota např. „100“.

3.3.1.1 Typy proměnných

Kromě názvu a hodnoty ještě proměnná obsahuje informaci o typu proměnné a o tom, kde je proměnná uložena. Typ proměnné je v našem případě číselný a nic jiného než číslo se zde objevit nesmí. V našem případě by to bylo i nelogické. Kdybychom proměnné s názvem „Cenovka“ přiřadili např. barvu zboží, tak by se již určitě nejednalo o cenovku. Stručně si typy proměnných probereme. V tabulce 3.1 jsou uvedeny všechny druhy proměnných, jejich klíčová slova a stručný popis. Detailnější popis je uveden v textu za tabulkou.

Typ	Klíčové slovo	Typový znak	Popis
celá čísla	Integer	%	celá čísla v rozsahu -32 768 až 32 767
celá čísla s větším rozsahem	Long	&	celá čísla v rozsahu -2 147 483 648 až 2 147 483 647
racionální čísla	Single	!	čísla s desetinnou čárkou v rozsahu +/-1,4E-45 až +/-3,4E38
racionální čísla s větším rozsahem	Double	#	čísla s desetinnou čárkou v rozsahu +/-4,9E-324 až +/-1,7E308
měna	Currency	@	čísla se čtyřmi desetinnými místy v rozsahu -9E14 až 9E14
text	String	$	sled písmen číslic a znaků zapsaných v uvozovkách
datum a čas	Date		datum a čas
logické hodnoty	Boolean		hodnoty True (pravda) a False (nepravda)
všechny typy	Variant		zahrnuje všechny uvedené typy

Tab.3.1: Základní typy proměnných

3.3.1.1.1 Celočíselné typy

Celočíselné typy máme dva – **Integer** a **Long** a liší se od sebe pouze rozsahem hodnot. Jak je již z názvu patrné mohou nabývat pouze celočíselných hodnot. Rozdíl v použití těchto dvou typů je v objemu paměti, který si pro sebe tyto proměnné vymezují. Long, protože má mnohem větší rozsah, zabírá v paměti mnohem více místa než Integer. Hlavní rozdíl v použití je v rychlostech výpočtu. Integer je rychlejší, protože má menší nároky na paměť počítače…

3.3.1.1.2 Reálná čísla

Reálná čísla mají ze stejného důvodu jako celočíselné typy opět dva typy – **Single** a **Double**. Tyto typy mohou nabývat hodnot reálných čísel – desetinných čísel, např. 3,14 ; 0,0054 ; 154,1 ; …

3.3.1.1.3 Měnový typ

Číselný typ **Currency** se používá pro práci s měnovými údaji – je omezen na čísla se čtyřmi místy za desetinnou čárkou.

3.3.1.1.4 Datumový typ

Typ s klíčovým slovem **Date** slouží pro práci s datem a časem. Tento typ je ve formátu reálného čísla, které značí dobu uběhnutou od tzv. bodu nula až do času, který požadujeme. Bodem nula byla určena půlnoc z 29. na 30. prosince roku 1899. Číslo 35000,5 v datovém typu Date tedy značí 35000 dnů od tohoto data a číslo za desetinnou čárkou značí část dne – v tomto případě tedy polovinu dne – což připadá na 12. hodinu polední.

3.3.1.1.5 Textový typ

Typ **String** může nabývat hodnot ve formě textu, který je uzavřen do uvozovek. Tento text může obsahovat jakékoliv znaky, čísla a písmena, tzn. že je možné použít i text s diakritikou.

3.3.1.1.6 Logický typ

Logický typ **Boolean** může nabývat pouze dvou hodnot, představovanými pouze pravdivostními hodnotami **True** (pravda) a **False** (nepravda). True a False patří mezi klíčová slova jazyka Basic. V datovém typu je možno mimo těchto hodnot použít i celá čísla. V tomto případě 0 značí hodnotu False a všechny ostatní hodnoty True. Hodnota používaná obvykle pro True je 1.

3.3.1.1.7 Typ Variant

Jazyk Basic má mezi svými základními typy proměnných také jednu zvláštnost, značně ulehčující práci. Jedná se o typ proměnné **Variant**. Tento typ je totiž naprosto univerzální a lze do něho vložit jakoukoliv hodnotu od číselné, přes textovou až po logickou. Všeobecně platí, že co do této „schránky“ vložíme, to z ní také posléze dostaneme.

3.3.1.2 Deklarace Proměnných

Typ proměnné je v programu jasně definován již při prvním výskytu této proměnné – tzv. deklaraci proměnné. K tomu slouží naše klíčová slova. Deklarace proměnné je definování proměnné - „oznámení počítači“, že zde existuje nějaká proměnná s jasným názvem a že v budoucnu bude nabývat určitých hodnot (např. celočíselných).

Deklaraci označuje klíčové slovo **Dim** (*Dimension*). Za tímto klíčovým slovem následuje název proměnné, klíčové slovo **as** a klíčové slovo značící datový typ. Jazyk Basic také umožňuje užití tzv. znakových typů, které mohou při deklaraci nahradit klíčová slova označující dat. typ. Znakové typy jsou u jednotlivých dat. typů uvedeny také v Tab.3.1. Následující dva zápisy deklarace proměnné Délka jsou tedy rovnocenné:

```
Dim Delka as Double
Dim Delka#
```

Jazyk Basic umožňuje zápisy proměnných také bez jakékoliv deklarace. V tomto případě Basic využívá automaticky datového typu Variant. Není-li tedy proměnná v programu deklarována na nějaký datový typ, jazyk Basic ji automaticky přiřadí univerzální typ Variant. To práci programátora značně zjednodušuje a ve většině případů si s takovýmto postupem vystačíme. Nyní si ukážeme možnosti deklarace proměnných na příkladech 3.1, 3.2 a 3.3, které vypočítávají obvod kruhu o poloměru 2,5 cm a výsledek vypisují do dialogového okna.

Příklad 3.1: Ukázka deklarace proměnných

```
Sub ObvodKruhu
        Dim Obvod, Polomer, Pi as Double
        Polomer = 2.5
        Pi = 3.14
        Obvod = 2 * Pi * Polomer
        Print Obvod
End sub
```

Popis : V tomto příkladu vidíme za klíčovým slovem Dim klasickou deklaraci proměnných Obvod, Poloměr a Pi na datový typ Double. Všimněme si, že je možné za klíčovým slovem Dim deklarovat více proměnných stejného typu. Takovéto proměnné musejí být v zápise deklarace odděleny navzájem čárkou.

Příklad 3.2.: Ukázka určení typu proměnných pomocí typových znaků

```
Sub ObvodKruhu
        Polomer# = 2.5
        Pi# = 3.14
        Obvod# = 2 * Pi * Polomer
        Print Obvod
End sub
```

Popis : V tomto příkladu vidíme stejný program jako v příkladu 3.1a, avšak místo deklarace je zde použito určení typu pomocí typových znaků přímo u prvního výskytu proměnných. Jak jsme se již dozvěděli – jazyk Basic deklaraci přímo nevyžaduje.

Příklad 3.3: Ukázka příkladu bez deklarace proměnných

```
Sub ObvodKruhu
        Polomer = 2.5
        Pi = 3.14
        Obvod= 2 * Pi * Polomer
        Print Obvod
End sub
```

Popis : Tento příklad ukazuje, že lze příklad zapsat bez deklarace proměnných i bez jakéhokoliv určení typů proměnných. Jazyk Basic zde implicitně přiřadí automaticky proměnným univerzální typ Variant. Výsledek bude stejně jako u předchozích dvou příkladů naprosto stejný.

Na příkladu 3.4, který vypočítává a vypisuje výsledek obsahu čtverce o stranách 3,5 a 2,3 cm, si ještě ukážeme, jak je to s prací s proměnnými různého typu. U číselných typů je logické, že „širší“ typy automaticky obsahují „užší“. Např. je zřejmé, že typy Double a Long obsahují všechna celá čísla typu Integer. Práce s různými datovými typy je tedy v jazyce Basic možná, ale je třeba mít na paměti, že nelze překročit rozsah datového typu – viz příklad 3.4.

Příklad 3.4: Práce s různými datovými typy proměnných

```
Sub ObsahCtverce
        Dim ObsahCtverce as Integer
        Dim StranaA, StranaB as Double

        StranaA = 3.5
        StranaB = 2.3
        ObsahCtverce = StranaA * StranaB

        Print ObsahCtverce
End sub
```

Popis : Proměnná ObsahCtverce je nastavena na datový typ Integer a strany čtverce jsou nastaveny na typ Double – tedy reálná čísla. Vzorec vypočítávající obsah čtverce by měl vyjít 8,05, avšak v tomto případě je náš výsledek (proměnná ObsahCtverce) celočíselného typu Integer a výsledek je tedy zaokrouhlen na celé číslo a rovná se 8.

3.3.1.3 Platnost proměnných

Proměnné mohou mít v OpenOffice.org Basic různou platnost. Zatím jsme se v našich příkladech setkali s tzv. **lokálními proměnnými**. Např. v příkladu 3.4 jsou deklarovány proměnné ObsahCtverce, StranaA a StranaB a mají platnost pouze v této proceduře (mezi klíčovými slovy Sub a End sub). Jedná se tedy o **lokální proměnnou** – bude-li použita v jiné proceduře, vytvoří se nová proměnná, která s touto nebude mít nic společného.

Mimo lokálních (místních) proměnných existují ještě tzv. **globální proměnné**. Ty jsou deklarovány mimo procedury prostřednictvím příkazů **Private**, **Public** nebo **Global**. Příkaz **Private** deklaruje proměnné na úrovni modulu – platí tedy pro všechny procedury v daném modulu. Příkaz **Publi**c deklaruje proměnné pro všechny otevřené knihovny a moduly – má tedy nejširší platnost. Proměnná s příkazem **Global** uchovává hodnoty i po skončení makra – viz příklad 3.5. Pokud chceme uchovat výsledek lokální proměnné v proceduře, použijeme v deklaraci proměnné příkaz **Static**.

Příklad 3.5: Použití globálních proměnných

```
Global Vyplata as Integer
Private Mesic as String

Sub VyplataLeden
        Mesic = "Leden"
        Vyplata = 21000
        Print "Za měsíc " + Mesic + " činí výplata " + Vyplata + " Kč"
End sub

Sub VyplataUnor
        Print "Poslední výplata činila " + Vyplata + " Kč"
        Mesic = "Unor"
        Vyplata = 18598.5
        Print "Za měsíc " + Mesic + " činí výplata " + Vyplata + " Kč"
End sub
```

Popis: Nad procedurami jsou deklarovány globální proměnné Vyplata a Mesic. Proměnná Vyplata je deklarována jako datový typ Integer a proměnná Mesic jako typ String, který pracuje s textem. V první proceduře VyplataLeden je proměnné Mesic přiřazen text „Leden“, proměnné Vyplata číselná hodnota 21000. Příkaz Print způsobí vyskočení dialogového okna s textem „Za měsíc Leden činí výplata 21000 Kč“. V druhé proceduře VyplataUnor nejdříve příkaz Print vytiskne v dialogovém okně text „Poslední výplata činila 21000 Kč“.

Všimněme si, že proměnná Vyplata není v této proceduře inicializována (není ji přiřazena žádná hodnota) a přesto nám vypíše hodnotu. Je to dané tím, že je globálně deklarována s příkazem Global, který si pamatuje hodnotu i po skončení procedury – v našem případě si pamatuje výplatu za měsíc leden. Dále je v naší proceduře opět přiřazena textová hodnota proměnné Mesic a proměnné Vyplata nová číselná hodnota – touto se přepíše hodnota z minulé procedury. Druhý příkaz Print nám již do dialogového okna vypíše nový text: „Za měsíc Únor činí výplata 18599 Kč“. Všimněme si, že jsme proměnné Vyplata přiřadili hodnotu 18598,5 a výsledek je 18599. Je to způsobeno tím, že jsme proměnnou Vyplata deklarovali s celočíselným datovým typem Integer a reálné číslo je tedy zaokrouhleno na celá čísla. Povšimněme si také řetězení, které používáme za příkazem Print: zde text v uvozovkách je vypsán přímo a k němu jsou pomocí operátoru + připojeny jiné text. řetězce a volání proměnných. Po zavolání proměnné je vypsána vždy její hodnota.

Globální a lokální proměnné se mohou překrývat. Je-li deklarována globální proměnná a v proceduře lokální proměnná se stejným názvem, tak tato lokální proměnná v této proceduře „překrývá“ tu globální, jak je ukázáno na příkladu 3.6, který je mírně upravený příklad 3.5, avšak s trochu odlišným výsledkem.

Příklad 3.6: Použití globálních a lokálních proměnných

```
Global Vyplata as Integer
Private Mesic as String

Sub VyplataLeden
        Mesic = "Leden"
        Vyplata = 21000
        Print "Za měsíc " + Mesic + " činí výplata " + Vyplata + " Kč"
End sub

Sub VyplataUnor
        Dim Vyplata as Double
        Print "Poslední výplata činila " + Vyplata + “ Kč“
        Mesic = "Unor"
        Vyplata = 18598.5
        Print "Za měsíc " + Mesic + " činí výplata " + Vyplata + " Kč"
End sub
```

Popis: Nad procedurami jsou deklarovány opět globální proměnné Vyplata a Mesic. V první proceduře VyplataLeden je proměnné Mesic přiřazen text „Leden“, proměnné Vyplata číselná hodnota 21000. Příkaz Print způsobí vyskočení dialogového okna s textem „Za měsíc Leden činí výplata 21000 Kč“. To je stejné jako v příkladu 3.5. V druhé proceduře je deklarována lokální proměnná se stejným názvem jako globální proměnná Vyplata. První příkaz Print na rozdíl od příkladu 3.5 vypíše výplatu s hodnotou 0 – to je způsobeno překrytím globální proměnné lokální proměnnou, které ještě nebyla přiřazena žádná hodnota. Druhý příkaz Print vypíše text „Za měsíc Únor činí výplata 18598,5 Kč“. Zde si všimněme, že na rozdíl od příkladu 3.5 je zde vypsána výplata

nezaokrouhlená na celé číslo. To je dáno právě tím, že globální proměnná typu Integer byla překryta lokální proměnnou, která je datového typu Double.

3.3.1.4 Datová pole

Proměnná obsahuje pouze jednu hodnotu. Ta se může sice v průběhu času měnit, ale v jednom časovém okamžiku uchovává vždy pouze jeden údaj. Abychom v případě, kdy pracujeme s mnoha údaji stejných hodnot, nemuseli deklarovat nespočet proměnných, existují v jazyce Basic tzv. datová pole (pole proměnných) – viz příklad 3.7, který vypíše 5. prvek z pole měsíců – tedy pátý měsíc Květen.

Příklad 3.7: Použití pole proměnných

```
Sub PoleMesicu
        Dim Mesic(12) as String
        Mesic(0) = "Leden"
        Mesic(1) = "Únor"
        Mesic(2) = "Březen"
                .
                .
                .
        Mesic(11) = "Prosinec"
        Print Mesic(4)
End sub
```

Popis: Pole proměnných musí být na rozdíl od jednoduchých proměnných vždy deklarováno příkazem **Dim**. Následuje název pole proměnných a za ním je v závorce uveden počet prvků pole – v našem případě počet měsíců v roce. Přiřazení hodnot jednotlivým prvkům je stejné jako přiřazování jednoduchým proměnným, akorát je zde vždy v závorce uveden index prvku. POZOR!: indexování začíná od nuly, takže našich dvanáct měsíců má indexy od 0 do 11 => příkaz Print Mesic(4) vypíše pátý měsíc – Květen. Pokud bychom chtěli použít indexování od 1, tak toho lze dosáhnout tak, že při deklaraci uvedeme do závorky za názvem pole rozsah indexů pole, kde vypíšeme index prvního a posledního prvku oddělené slovem **To**. Např: Dim Mesic(1 To 12) as String.

Pole v příkladu 3.7 je jednorozměrné. Vytvářet však můžeme i vícerozměrná pole. Např. pomocí příkazu

Dim Matice(3,2) as Integer

získáme matici o rozměrech 4 x 3 (! Indexování od nuly !).

Někdy nemusíme dopředu znát počet prvků pole, potom můžeme pole deklarovat jako prázdné – do závorek nezapíšeme nic:

Dim PlemenaPsu() as String

Toto pole můžeme následně naplnit třeba výčtem prvků a to způsobem použitém v příkladu 3.8, který vypisuje ve výsledku počet prvků(plemen psů) zadaných v poli.

Příklad 3.8: Naplnění prázdného pole

```
Sub Plemena
        Dim PlemenaPsu() as String
        PlemenaPsu() = ARRAY("Jezevčíci", "Teriéři", "Ohaři", "Ovčáci" )
        PrvniIndex = LBOUND(PlemenaPsu())
        PosledniIndex = UBOUND(PlemenaPsu())
        PocetPlemen = PosledniIndex + 1
        Print “Počet plemen v seznamu(poli): “ + PocetPlemen
End sub
```

Popis: Prázdné pole PlemenaPsu naplníme pomocí funkce ARRAY(), v které jsou jednotlivé prvky vypsány. Dále v tomto příkladu využíváme funkcí LBOUND() a UBOUND(), které vrací první, respektive poslední index pole, které mají tyto funkce uvedené jako parametr v závorkách. Proměnné PocetPlemen je přiřazena proměnná PosledniIndex zvýšená o 1, opět kvůli indexování pole od 0. Výsledek tohoto příkladu je dialogové okno s počtem prvků pole.

3.3.1.5 Konstanty

Konstanty představují proměnné s konstantní hodnotou, tzn. konstantě přiřadíme hodnotu pouze jednou. Tato hodnota již nelze měnit a proměnná si ji uchovává po celou dobu své existence. Konstanty se musejí stejně jako datová pole vždy deklarovat. K deklaraci konstant slouží klíčové slovo **Const** a zároveň s deklarací se přiřazuje konstantě i hodnota. Pro odlišení od obyčejných proměnných je zvykem psát jméno konstant velkými písmeny (není to pravidlem a záleží to pouze na programátorovi).

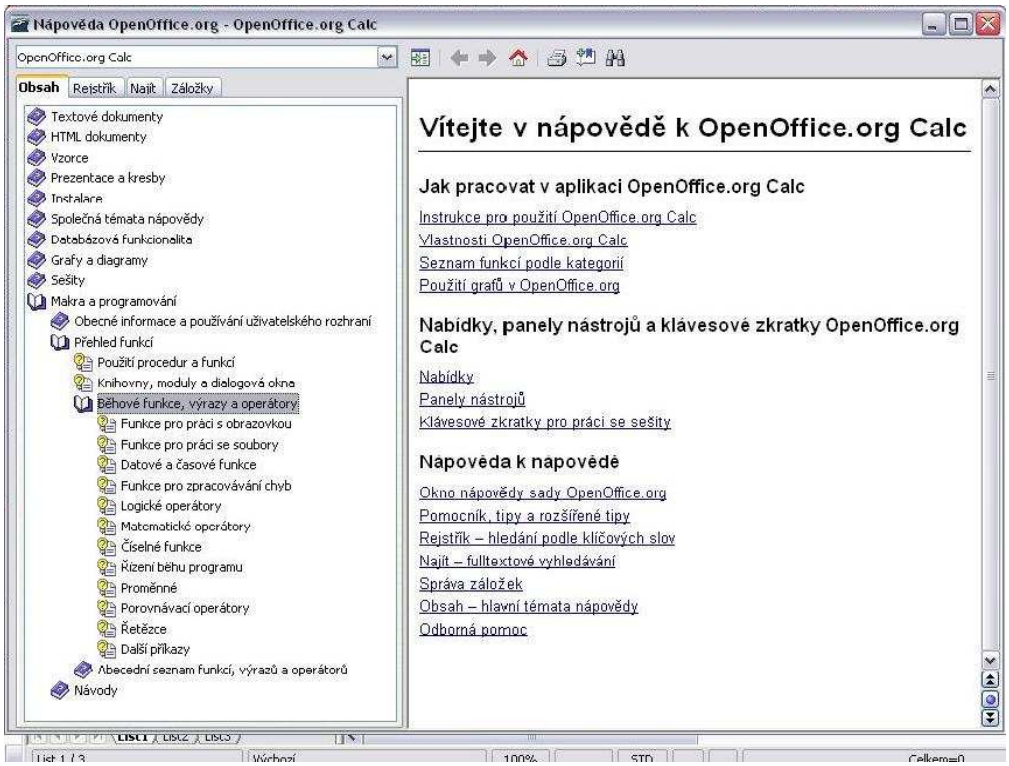
Const MAX = 100

Konstanty jsou využívány hlavně v případech, kdy potřebujeme neměnnou hodnotu proměnné a pokud chceme předejít chybám, které by způsobila nechtěná změna této hodnoty.

3.3.2 Funkce a příkazy

Dalšími stavebními kameny jazyka Basic jsou kromě proměnných, které slouží hlavně k uchovávání hodnot, funkce a příkazy. Příkazy slouží hlavně k práci – vykonávají nám určitou činnost. Funkce mimo to, že provedou nějakou činnost jako příkazy, ještě vrátí nějaký výsledek. Některé funkce se mohou použít i jako příkaz – nepotřebujeme-li znát návratovou hodnotu. Vstupní data předáváme příkazům a funkcím ve formě parametrů. Jak jsme se již dozvěděli v podkapitole 3.2.4, tak funkce se od příkazů odlišují tím, že mají parametry zadané v závorkách, příkazy toto nevyžadují.

Všechny informace potřebné pro použití funkcí a příkazů získáme z nápovědy, kterou aplikace OpenOffice.org obsahuje. Zmáčkneme-li v otevřeném dokumentu klávesu **F1**, tak se nám nápověda pro aplikaci OpenOffice.org otevře. V záložce **Obsah** otevřeme knihu **Makra a programování**, kde najdeme část **Přehled funkcí** a v této části otevřeme oddíl **Běhové funkce, výrazy a operátory**. Otevřou se nám hlavní oddíly pro práci s funkcemi a příkazy. Známe-li názvy funkcí či příkazů, můžeme podrobnosti o nich nalézt v nápovědě také přes sekci **Rejstřík**.



Obr.3.2: Okno nápovědy OpenOffice.org

3.3.2.1 Funkce pro práci s proměnnými

Pro práci s proměnnými je v jazyce Basic celá řada funkcí a příkazů. Některé hlavní příkazy jsme si již uvedli v kapitole o proměnných. Jednalo se o příkazy **Private, Public, Global, Static, Const** a **Dim**. Mimo příkazů bychom měli vědět také něco o funkcích pracujících s proměnnými.

3.3.2.1.1 Funkce určující typ proměnné

Funkce určující typ proměnné jsou dvě**: TypeName** a **VarType.** Proměnná, u které chceme zjistit její typ, se zadává jako parametr do závorek za název funkce. Návratová hodnota jednotlivých funkcí je uvedena v tabulce 3.2.

TypeName	VarType	Typ proměnné
Boolean	11	Logická proměnná
Date	7	Proměnná data
Double	5	Proměnná s dvojitou přesností a plovoucí desetinnou čárkou
Integer	2	Celočíselná proměnná
Long	3	Dlouhá celočíselná proměnná
Object	9	Objektová proměnná
Single	4	Proměnná s jednoduchou přesností a plovoucí desetinnou čárkou
String	8	Řetězec
Variant	12	Proměnná typu Variant (může obsahovat všechny typy a zadává se definicí)
Empty	0	Proměnná není inicializována
Null	1	Žádná platná data

Tab.3.2: Přehled návratových hodnot funkcí TypeName a VarType

Jak vidíme z tabulky, tak funkce TypeName má návratovou hodnotu ve formě textu, např. funkce TypeName s parametrem 66.6 (*TypeName(66.6)*) vrátí hodnotu Double. Funkce VarType s tímtéž parametrem by vrátila hodnotu **5**.

3.3.2.1.2 Funkce pro práci s datovým typem Variant

Pro určení typu hodnot uložených v proměnných typu Variant používáme funkcí **IsNumeric, IsDate, IsEmpty, IsNull, IsArray**. Tyto funkce nám zjišťují zda proměnná obsahuje hodnoty číselné, datové, hodnotu Empty (proměnná není

inicializována), hodnotu Null (proměnná neobsahuje žádná data), nebo jestli proměnná je pole.

3.3.2.1.3 Funkce pro převod typů proměnných

Jazyk Basic umí automaticky převádět jednotlivé datové typy mezi sebou. Pokud bychom to chtěli provést přesně námi definovaným způsobem, tak Basic má pro nás předdefinované funkce **CCur, CBool, CDate, CDbl, CInt, CLng, CSng, CStr a CVar.** Tyto funkce převádí hodnoty uvedené jako parametry funkcí na datové typy **Currency, Boolean, Date, Double, Integer, Long, Single, String a Variant**. Použití jedné z funkcí si ukážeme na příkladu 3.9, který vypisuje postupné kroky programu s užitím funkcí pro práci s proměnnými.

Příklad 3.9: Použití funkcí pro práci s proměnnými

```
Sub PrevodTypu
  TypDouble = 6.66
  Print “TypDouble = “ + TypDouble
  Print “Funkce TypeName zjistila typ proměnné “ + TypeName(TypDouble)
  Print “Funkce IsNumeric zjistí, zda se jedná o číslo: “ + IsNumeric(TypDouble)
  TypInteger = CInt(TypDouble)
  Print “Po přetypování pomocí funkce CInt je hodnota “ + TypInteger
  Print “Po přetypování funkcí CInt je typ proměnné “ + TypeName(TypInteger)
End sub
```

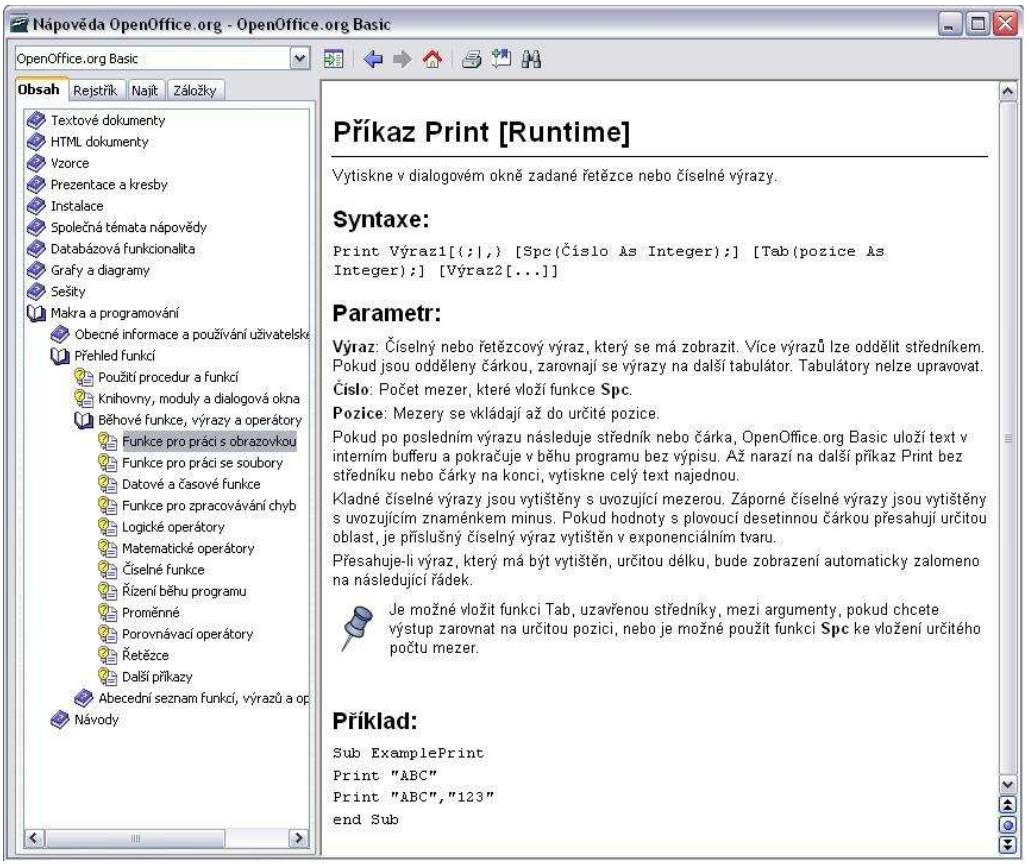
Popis: V tomto příkladu můžeme vidět použití všech typů funkcí pro práci s proměnnými. V příkladu využíváme toho, že jazyk Basic nevyžaduje deklaraci proměnných. Proměnné TypDouble je tedy automaticky přiřazen typ Variant a námi zadaná hodnota 6,66. Příkazem Print si hodnotu této proměnné vypíšeme. Za druhým příkazem Print je použitá funkce TypeName pro zjištění typu hodnoty proměnné. Přestože byl proměnné TypDouble automaticky přiřazen typ Variant, tak funkce TypeName zjistí z hodnoty této proměnné (6,66) typ Double a příkaz Print nám ho vypíše. Za třetím příkazem Print jsme použili funkci pro zjištění, zda typ Variant obsahuje číselnou hodnotu. Příkaz nám vypíše hodnotu True. Proměnné TypInteger přiřadíme funkcí CInt hodnotu převedenou z typu double na typ integer (z hodnoty 6,66 se stane hodnota 7 a tu si pomocí příkazu Print vypíšeme). Funkce TypeName nám posléze po provedení posledního příkazu Print vypíše typ Integer.

3.3.2.2 Funkce pro práci s obrazovkou (Input/Output)

Pro komunikaci s uživatelem slouží nám již dobře známý a využívaný příkaz **Print**, funkce **MsgBox** a **InputBox**, kterou jsme již také jednou v příkladech použili.

3.3.2.2.1 Příkaz Print

Pokud si tento příkaz vyhledáme v nápovědě, tak získáme informace zobrazené na obr. 3.3.



Obr.3.3: Výpis informací o příkazu Print z nápovědy OpenOffice.org

Popis jednotlivých příkazů a funkcí v nápovědě OpenOffice.org je podobný jako u tohoto příkazu, proto si tento výpis vysvětlíme podrobněji.

Výpis začíná informací o názvu příkazu či funkce a pod čarou je informace k čemu příkaz či funkce slouží. Slovo Runtime uvedené v hranatých závorkách značí, že je příkaz vykonáván při běhu procedury.

Dále je popis syntaxe příkazu. Za příkazem samotným jsou uvedeny parametry příkazu. Pokud jsou parametry uvedené v hranatých závorkách, jedná se o parametry nepovinné. Pokud existuje možnost volby, uvádí se jednotlivé volby ve složených závorkách. V syntaxi tohoto příkazu existuje možnost volby oddělení parametrů středníkem nebo čárkou. Pokud jsou parametry odděleny čárkou, jako u druhého

příkazu Print v Příkladu uvedeném v nápovědě, tak se vypíší oddělené tabulátorem (v tomto případě „ABC 123“). Pokud bychom použili místo čárky středník, bude výpis jednotlivých parametrů hned za sebou („ABC123“). Dále je uvedena v hranatých závorkách (tedy nepovinné) možnost použití doplňujících funkcí pro vložení tabulátorů nebo mezer, tak jak je vysvětleno v textu nápovědy.
Nakonec je uveden popis jednotlivých parametrů a úplně nakonec je výpis doplněn pro představu o jednoduchý příklad.

3.3.2.2.2 **Funkce a příkaz MsgBox**

MsgBox existuje jak ve formě příkazu, tak i ve formě funkce. Rozdíl mezi nimi je pouze v tom, že funkce oproti příkazu obsahuje ještě návratovou hodnotu. Proč existuje MsgBox v obou formách si ukážeme v této podkapitole na příkladu 3.10.
Příkaz se používá pro zobrazení informace na obrazovku v různých formách dialogových oken. Z nápovědy si zjistíme syntaxi tohoto příkladu a zjistíme, že příkaz má zpravidla tři parametry a zadává se ve formě

MsgBox Zprava, [Vzhled], [Nadpis]

Parametr Zprava je typu String, Vzhled je v celočíselné formě(Integer) , Nadpis opět String, přičemž poslední dva parametry jsou nepovinné. Parametr Zprava obsahuje informaci, kterou chceme zobrazit, parametr Vzhled formu dialogového okna, ve kterém chceme informaci zobrazit (viz tabulka 3.3) a parametr Nadpis obsahuje nadpis dialogového okna.

0	Zobrazí se pouze tlačítko OK.
1	Zobrazí se tlačítka OK a Zrušit.
2	Zobrazí se tlačítka Zrušit, Opakovat a Ignorovat.
3	Zobrazí se tlačítka Ano, Ne a Zrušit.
4	Zobrazí se tlačítka Ano a Ne.
5	Zobrazí se tlačítka Opakovat a Zrušit.
16	Do dialogového okna je přidána ikona Zastavit.
32	Do dialogového okna je přidána ikona Otazník.
48	Do dialogového okna je přidána ikona Vykřičník.
64	Do dialogového okna je přidána ikona Informace.
128	První tlačítko v dialogu je použito jako výchozí tlačítko.
256	Druhé tlačítko v dialogovém okně je použito jako výchozí tlačítko.
512	Třetí tlačítko v dialogovém okně je použito jako výchozí tlačítko.

Tab.3.3: Hodnoty a jejich funkce druhého parametru fce MsgBox

Tak např. příkaz *MsgBox "Už to chápeš?", 36, "Otázka"* zobrazí dialogové okno zobrazené na obrázku 3.4. Je zde uveden vzhled jako číslo 36, ale v naší tab.3.3 takovýto vzhled popsán není. Jak je to možné? - Je to tím, že lze jednotlivé vzhledy „sčítat". Vzhled 36 tedy znamená vzhled 32 + vzhled 4. (32 + 4 = 36 ;-)).

Obr.3.4: Dialogové okno se vzhledem 36

Po zmáčknutí jakéhokoliv tlačítka se toto okno zavře bez jakéhokoliv výsledku. A právě proto existuje také funkce téhož názvu. Ta je nám navíc po stisknutí tlačítka v dialogovém okně schopna vrátit nějakou číselnou hodnotu, značící určitou akci (viz tabulka 3.4). Syntaxi funkce opět vidíme v nápovědě OpenOffice.org. Je stejná jako u příkazu, akorát parametry jsou (jak je u funkce zvykem) uzavřeny v kulatých závorkách. Využití funkce i příkazu MsgBox vidíme na příkladu 3.10, který ve výsledku vypisuje do dialogového okna informaci o použitém tlačítku v předchozím dialogovém okně.

Návratová hodnota	Použité tlačítko
1	OK
2	Zrušit
3	Zrušit
4	Opakovat
5	Ignorovat
6	Ano
7	Ne

Tab.3.4: Návratové hodnoty fce MsgBox

Příklad 3.10: Použití fce/příkazu MsgBox

```
Sub PouziteTlacitko
        Dim Zprava1, Zprava2, Nadpis as String
        Dim Vzhled1, Vzhled2 as Integer
        Zprava1 = "Vyberte tlacitko:"
        Zprava2 = "Vybrali jste tlacitko: "
        Nadpis = "Tlacitka"
        Vzhled1 = 36
        Vzhled2 = 64
        Tlacitko = MsgBox(Zprava1, Vzhled1, Nadpis)
        MsgBox Zprava2 + Tlacitko, Vzhled2, Nadpis
End sub
```

Popis: Pro přehlednost jsme zde vytvořili jednotlivé proměnné pro zprávu, nadpis a vzhled dialogového okna. Mohli jsme vše také napsat do jediného řádku příkazu MsgBox, ale musíme uznat, že takovýto zápis by byl velmi dlouhý a značně nepřehledný. Proměnné Tlacitko jsme přiřadili funkci MsgBox, která zobrazí dialogové okno s tlačítky ano/ne a textem z proměnné Zprava1 a zároveň přiřadí proměnné Tlacitko svou návratovou hodnotu. Všimněme si, že jsme proměnnou tlačítko na začátku deklarovali jako Integer – to je dáno tím, že návratové hodnoty fce MsgBox jsou číselné (viz tab.3.4). Na posledním řádku těla procedury vidíme tentokrát použitý příkaz MsgBox, který nás následně informuje o použitém tlačítku. Informace je ve formě čísla (buď 6-Ano, nebo 7-Ne).

3.3.2.2.3 **Funkce InputBox**

Funkci InputBox jsme již použili v příkladu 3.1. Tato funkce slouží pro čtení textu zadaného uživatelem z klávesnice. Zobrazí dialogové okno s výzvou umožňující uživateli zadat text do textového pole a tento vstup je po zmáčknutí tlačítka OK přiřazen proměnné. Pokud zavřeme dialogové okno pomocí tlačítka Zrušit, **InputBox** vrátí řetězec nulové délky (""). Podrobnější popis syntaxe s krátkým příkladem najdeme opět v nápovědě.

3.3.2.3 Příkazy pro řízení běhu programu

Dostali jsme se v programování k velmi důležitým příkazům. Zatím jsme se zabývali programy, které se vykonávaly postupně - řádek po řádku. Velice často je ale v programování potřeba některé kroky opakovat, „přeskočit“ určitý blok programu, nebo větvit program tak, aby reagoval různě na různé vstupní údaje. K tomu používáme příkazy, které najdeme v nápovědě pod pojmem **Řízení chodu programu**. Tyto příkazy můžeme rozdělit na

- **podmíněné příkazy**, které využívají k chodu podmínek
- **cykly** - tyto příkazy spouštějí smyčky
- **příkazy skoku**, které provádějí skoky v programu

3.3.2.3.1 **Příkaz If -Then - Else**

Jedná se o podmíněný příkaz, který se provede pouze tehdy, je-li splněna podmínka.

Syntaxe:

If *podmínka=True* ***Then*** *Blok příkazů1 [****ElseIf*** *podmínka=true* ***Then****] Blok příkazů2 [****Else****] Blok příkazů3* ***End If***

Popis syntaxe:

Je-li(If) podmínka splněna(=True), **potom proveď(Then)** Blok příkazů1. **Pokud není splněna první podmínka, zjisti zda je splněna další a je-li(ElseIf)** splněna **potom proveď(Then)** Blok příkazů2, nejsou-li splněny žádné podmínky **proveď(Else)** Blok příkazů3. Ukonči **podmíněný příkaz(EndIf).**

Všimněme si v syntaxi, že klíčová slova ElseIf a Else nejsou povinné. Příkaz tedy může také vypadat takto:

If** podmínka=true **Then** Blok příkazů1 **Else** Blok příkazů2 **End If

Pokud je splněna podmínka, splň Blok příkazů1, jinak(není-li splněna) splň Blok příkazů2. Konec

Více si vysvětlíme v popisu příkladu 3.11, který dle zadaného čísla(1 nebo 2) vypíše, zda se jedná o liché či sudé číslo.

Příklad 3.11: Praktické použití podmíněného příkazu If-Else

```
Sub SudaLicha
        Vstup = InputBox("Zadejte číslo 1 nebo 2: ", "Sudé - liché")
        If Vstup = 1 Then
                Print "Zadali jste liché číslo " + Vstup
        ElseIf Vstup = 2 Then
                Print "Zadali jste sudé číslo " + Vstup
        Else
                Print "Zadali jste jiné číslo než 1 nebo 2 "
        EndIf
End sub
```

Popis: Proměnné Vstup jsme přiřadili výslednou hodnotu fce InputBox (vstup zadaný uživatelem). Následuje podmínka: Je-li splněna podmínka, že uživatel zadal na klávesnici číslo 1, potom příkaz Print vypíše do dialogu, že bylo zadáno liché číslo. Byla-li tato podmínka splněna, příkaz ostatní podmínky přeskočí a skončí příkazem **EndIf**. Není-li splněna první podmínka, zjišťuje se, zda je splněna druhá. Je-li splněna, potom příkaz Print vypíše, že bylo zadáno sudé číslo a podmíněný příkaz skončí. Není-li splněna žádná z podmínek (může jich být nekonečně mnoho), potom se splní příkaz za klíčovým slovem **Else**.

Tento příkaz je velice pěkně popsán v nápovědě, kde je také uveden další příklad…

3.3.2.3.2 Příkaz Select - Case

Podobný příkaz pro vyhodnocování podmínek jako If-Else je i příkaz Select-Case, který na základě hodnoty výrazu definuje jeden nebo více bloků příkazů. Tento příkaz je vhodný zejména tehdy, pokud je v podmínce uvedeno více možností výsledku.

Syntaxe:

Select Case *podmínka* ***Case*** *výraz Blok příkazů [****Case*** *výraz2 Blok příkazů][****Case Else****] Blok příkazů* ***End Select***

Popis syntaxe:

Podmínka je jakýkoliv výraz, podle kterého se určí spuštění jednotlivých bloků. Pokud výraz za **Case** odpovídá podmínce, spustí se následující Blok příkazů. Není-li splněna ani jedna z podmínek provede se Blok příkazů za **Case Else.** Příkaz končí opět splněním jedné z podmínek, nebo doběhne až do konce k **End Select**. Použití je vidět na příkladu 3.12, který dle zadaného čísla na vstupu (tentokrát 1-20) zjistí, zda je zadáno sudé či liché číslo. Více v popisu programu.

Příklad 3.12: Použití příkazu Select-Case

```
Sub SudaLicha2
   Dim Vstup as Integer
   Vstup = InputBox("Zadejte číslo 1 až 20: ", "Sudé - liché")
   Select Case Vstup
         Case 1, 3, 5, 7, 9
               Zprava = "Zadali jste liché číslo "
         Case 2, 4, 6, 8, 10
               Zprava = "Zadali jste sudé číslo "
         Case 11 to 20
               Zprava = "Zadali jste číslo v rozmezí 11 - 20"
         Case Else
               Zprava = "V rozmezí 1-20 se nevyskytuje vámi zadané číslo "
   End Select
   Print Zprava + Vstup
End sub
```

Popis: Rozšířili jsme příklad 3.11 na vyhodnocení čísel 1-20. Po spuštění makra je vyžádáno zadání hodnoty a tato hodnota je přiřazena proměnné Vstup. Vyhodnocení začíná parametrem Select Case, za nímž je uveden hodnocený výraz Vstup. Jednotlivé podmínky začínají slovem Case. Podmínka Case nám umožňuje zadat více hodnot pro vyhodnocení a to výčtem prvků oddělených čárkou a nebo rozmezí čísel s klíčovým slovem **to.** Po splnění jedné

z podmínek se vyhodnocování ukončí a makro pokračuje za příkazem End Select dál. V tomto případě přejde na příkaz Print, který nám vypíše proměnnou Zprava, které byla přiřazena textová hodnota dle jedné z podmínek a za touto hodnotou vypíše ještě číslo zadané na vstupu. Např. po zadání čísla 6 je vyhodnocena jako správná druhá podmínka, která ve výčtu prvků číslo 6 obsahuje. Tato podmínka přiřadí proměnné Zprava text "Zadali jste sude číslo" a podmíněný příkaz ukončí. Poté program pokračuje dál příkazem Print, který nám text z proměnné Zprava vypíše spolu s hodnotou proměnné Vstup (naše zadané číslo 6) a makro skončí příkazem End Sub.

3.3.2.3.3 **Příkaz Do-Loop**

Od podmíněných příkazů se dostáváme k dalšímu druhu příkazů – k cyklům. Jedním z příkazů pro opakování je příkaz Do - Loop. Příkazy uvedené mezi parametry Do a Loop se budou opakovat, je-li podmínka splněna, nebo do splnění podmínky. Nejlépe si to asi předvedeme na příkladu, kde je uživatel nucen zadávat hodnoty na vstupu tak dlouho, dokud nezadá číslo.

Příklad 3.13: Užití cyklu Do-Loop

```
Sub ZadejCislo
        Do
                Vstup = InputBox("Zadejte číslo")
                JeCislo = IsNumeric(Vstup)
                If JeCislo = False Then
                        Print "Chyba, zadejte cislo!"
                End If
        Loop Until JeCislo = True
        Print "Zadali jste cislo " + Vstup
End Sub
```

Popis: Klíčovým slovem Do začíná cyklus. V dialogovém okně je uživatel vyzván k zadání čísla. Zadá nějakou hodnotu, ta je pomocí fce IsNumeric zkontrolována a pokud se jedná o číslo, pak je proměnné JeCislo přiřazena logická hodnota True. Nejedná-li se o číslo, potom je přiřazena hodnota False. Dále vidíme podmínku If-Then, která pokud nebylo zadáno číslo, vypíše pomocí příkazu Print dialog s chybou a žádostí o opětovné zadání čísla. Řádek s klíčovými slovy Loop Until a následnou podmínkou, ve které se zjišťuje zda je proměnná JeCislo rovna True, nám určuje další běh programu. Je-li podmínka splněna, cyklus se ukončí. Není-li splněna, opakuje se blok příkazů mezi Do … Loop tak dlouho, dokud podmínka není splněna – v našem případě se zobrazuje dialogové okno s výzvou opakovaně tak dlouho, dokud uživatel nezadá na vstupu číselnou hodnotu.

Cyklus Do – Loop může probíhat mnoha způsoby. Nejjednodušší je zápis

Do

Blok příkazů

Loop

Takovýto zápis by opakoval příkazy umístěné v těle cyklu do nekonečna, nebo do chvíle, dokud by jeden z příkazů v těle cyklus neukončil. Častější je tedy použití cyklu s podmínkami. Pro zapsání podmínky jsou určeny klíčová slova **Until** a **While**. Slovo **Until** určuje opakování do té chvíle, dokud se podmínka splní (viz. Příklad 3.13). Při splnění podmínky se cyklus ukončí. Slovo **While** určuje opakování po dobu, kdy podmínka platí. Ve chvíli, kdy není podmínka splněna, cyklus končí. Kontrolu podmínek Until a While můžeme zapsat buďto na začátek cyklu za slovo Do, nebo na konec za slovo Loop. Máme tedy mnoho možností zápisu. Podmínku na konci cyklu jsme si již ukázali na příkladu 3.13 – v tomto případě cyklus vždy proběhne alespoň jednou. Pokud bychom podmínku zapsali na začátek cyklu, tak by cyklus nemusel v případě splnění podmínky proběhnout ani jednou.

U použití cyklů je vždy nebezpečí, že se nám může program zacyklit. Musíme na toto tedy pamatovat již při psaní kódu.

Cyklus je možné také ukončit příkazem **Exit Do** umístěným uvnitř těla cyklu. Použití tohoto příkazu ukazuje příklad 3.14, který má naprosto stejný výsledek jako př.3.13.

Příklad 3.14: Užití příkazu Exit Do pro opuštění cyklu

```
Sub ZadejCislo2
        Do
                Vstup = InputBox("Zadejte číslo")
                JeCislo = IsNumeric(Vstup)
                If JeCislo = False Then
                        Print "Chyba, zadejte cislo!"
                Else
                        Exit Do
                End If
        Loop
        Print "Zadali jste cislo " + Vstup
End Sub
```

Popis: Výsledek programu je pro uživatele naprosto stejný jako u předchozího příkladu 3.13. V kódu programu se liší tím, že cyklus neobsahuje žádnou podmínku. Ukončí se příkazem **Exit Do**, který cyklus ukončí v případě, že je zadána požadovaná hodnota – číslo. Toho je docíleno díky podmíněnému příkazu **If-Else** a pomocí příkazu **Exit Do** v těle tohoto příkazu.

3.3.2.3.4 **Příkaz For-Next**

Předchozí příkaz Do-Loop se používá v situacích, kdy nevíme kolikrát cyklus proběhne. Pro situace, kdy víme počet opakování cyklu, je výhodnější příkaz **For-Next**. Použití vidíme na příkladu 3.15, který nám vypíše postupně jednotlivé prvky pole jmen.

Příklad 3.15: Užití příkazu For-Loop

```
Sub VypisPole
        Dim PoleJmen(3) as String
        PoleJmen(1) = "Pete"
        PoleJmen(2) = "Bob"
        PoleJmen(3) = "Pam"
        For n=1 To 3 Step 1
                Print PoleJmen(n)
        Next n
End Sub
```

Popis: Na začátku těla cyklu si deklarujeme pole proměnných o velikosti 3. Na následujících třech řádkách toto pole postupně naplníme jmény. Dále zde vidíme cyklus For-Next, který má tři kroky a postupně nám vypíše pomocí příkazu Print (který má uvnitř těla) všechny tři jména z pole.

Na příkladu 3.15 vidíme použití cyklu **For-Next**. Za klíčovým slovem **For** přiřadíme proměnné n číslo 1, odtud začne cyklus počítat. Za slovem **To** je číslo 3, které udává konečnou hodnotu počítadla a velikost kroku udává číslice za slovem **Step**. Řádek tedy můžeme přeložit jako: Počítej od jedné do tří v kroku jedna (přičítej 1). Po spuštění makra a přiřazení jmen do pole proměnných se dostaneme k cyklu. Proměnné n se přiřadí hodnota 1, přejde se na příkaz Print PoleJmen(n), kde se za n dosadí hodnota proměnné n (číslo 1) a vypíše se první jméno z pole proměnných. Klíčové slovo Next s uvedením proměnné n na konci cyklu znamená zvýšení hodnoty proměnné n o krok (Step), v našem případě o 1 a navrácení na začátek cyklu. A začínáme opět od začátku, avšak nyní již s hodnotou proměnné n = 2. Postupně se nám vypíše druhý a po zvýšení proměnné n na 3 i třetí prvek. Při zvýšení proměnné n na číslo 4 se sice pokusí cyklus provést znovu, ale zastaví se ihned na svém prvním řádku za slovem For, protože číslo 4 již nesplňuje podmínku n = 1 to 3. Cyklus je tedy ukončen a makro pokračuje dál a skončí.

Stejně jako u cyklu **Do-Loop** existoval příkaz pro ukončení cyklu přímo z těla cyklu **Exit Do**, tak i u cyklu **For-Next** najdeme stejný příkaz se jménem **Exit For**.

3.3.2.3.5 **Příkaz GoTo**

Příkaz GoTo je příkazem skoku. Tento příkaz funguje tak, že pokračuje (skočí) v provádění programu na řádek, který je označen tzv. návěstím. Běh programu s takovýmto příkazem si vysvětlíme na příkladu 3.16. Funkce programu je popsána v jeho popisu.

Příklad 3.16: Použití příkazu skoku GoTo

```
Sub ZadejCislo3
        Zacatek:
                Vstup = InputBox("Zadejte číslo")
                JeCislo = IsNumeric(Vstup)
                If JeCislo = False Then GoTo Chyba
                GoTo Konec
        Chyba:
                Print "Chyba, zadejte cislo!"
                GoTo Zacatek
        Konec:
                Print "Zadali jste cislo " + Vstup
End Sub
```

Popis: Opět zde máme mírně modifikovaný příklad ZadejCíslo, který má opět pro uživatele naprosto stejný výsledek jako u příkladu 3.13 a 3.14.
V těle procedury si všimněme třech míst označených jako Začátek: , Chyba: a Konec: . Jedná se o tzv. návěstí. Samo o sobě běh programu nijak nezmění. Dále si všimněme použití příkazu GoTo. První výskyt máme v podmínce If. Pokud je tato podmínka splněna - tzn. zadaný znak není číslo, tak tento příkaz odkazuje na návěstí Chyba:. Program tedy skočí v kódu na tento řádek a pokračuje dál od tohoto místa. Splní příkaz Print a narazí na další příkaz skoku, tentokrát s odkazem na návěstí Zacatek:. Běh programu se tedy přesune na řádek s tímto návěstím a pokračuje dál od tohoto místa – v našem případě opět od začátku programu. Toto se opakuje tak dlouho, dokud není uživatelem zadáno číslo. Potom se podmínka If nesplní, příkaz GoTo Chyba se přeskočí a program pokračuje dál příkazem GoTo Konec. Program skočí na návěstí Konec:, pokračuje příkazem Print a skončí.

3.3.2.3.6 **Příkaz Exit**

Jako příkaz ovlivňující chod makra můžeme ještě zařadit příkaz Exit. Již jsme se s ním setkali ve spojení se slovy Do a For pro ukončení cyklů. Pro ukončení běhu procedury se používá příkaz **Exit Sub**. Používá se hlavně při ukončení jednotlivých větví procedury. Více opět na příkladu 3.17, který je opět modifikací předchozího př.3.16.

Příklad 3.17: Použití příkazu Exit pro ukončení procedury

```
Sub ZadejCislo4
        Zacatek:
                Vstup = InputBox("Zadejte číslo")
                JeCislo = IsNumeric(Vstup)
                If JeCislo = False Then GoTo Chyba
                Print "Zadali jste cislo " + Vstup
                Exit Sub
        Chyba:
                Print "Chyba, zadejte cislo!"
                GoTo Zacatek
                Print "Zadali jste cislo " + Vstup
End Sub
```

Popis: Opět zde máme modifikaci příkladu ZadejCislo. Na rozdíl od příkladu 3.16 jsme místo použití příkazu skoku GoTo s návěstím Konec: použili pro ukončení programu rovnou příkaz Exit Sub. K tomuto příkazu se běh programu dostane pouze tehdy, je-li nesplněna podmínka v podmíněném příkazu If-Else. V našem případě, pokud uživatel zadá jakoukoliv číselnou hodnotu, která je po něm opět požadována.

3.3.2.4 Číselné funkce

Pro práci s čísly jsme si již popsali v kapitole 3.2.5 matematické a logické operátory. Ty vrací výsledek spojením dvou výrazů. Číselné funkce se od operátorů liší tím, že funkci je předán parametr a funkce vrátí výsledek. Např. fce

Sqr (Cislo)

vrátí druhou odmocninu čísla, které je zadáno jako parametr. Práce s funkcemi je velmi jednoduchá a proto si zde ty hlavní pouze s krátkým popisem vyjmenujeme. Více se o nich můžeme dozvědět opět v nápovědě v oddíle Číselné funkce.

3.3.2.4.1 **Trigonometrické funkce**

Funkce **Atn(Cislo)** Trigonometrická funkce, která vrátí arkustangens číselného výrazu. Vrácená hodnota je v rozsahu -Pi/2 až +Pi/2.

Funkce **Cos(Cislo)** Vypočítá kosinus úhlu. Úhel se zadává v radiánech. Výsledek je v rozsahu -1 až 1.

Funkce **Sin(Cislo)** Vrátí sinus úhlu. Úhel se zadává v radiánech. Výsledek je v rozsahu -1 až 1.

Funkce **Tan (Cislo)** Určí tangens úhlu. Úhel se zadává v radiánech.

3.3.2.4.2 Exponenciální a logaritmické funkce

Funkce **Exp(Cislo)** Vrátí základ přirozeného logaritmu (e = 2,718282) umocněný na zadané číslo.

Funkce **Log(Cislo)** Vrátí přirozený logaritmus čísla.

3.3.2.4.3 Celá čísla

Funkce **Fix(Vyraz)** Vrátí celočíselnou hodnotu číselného výrazu, z něhož odstraní zlomkovou část.

Funkce **Int(Cislo)** Vrátí celočíselnou část čísla.

3.3.2.4.4 Funkce pro převod čísel

Funkce **Hex(Cislo)** Vrátí řetězec, který představuje šestnáctkovou hodnotu čísla.

Funkce **Oct(Cislo)** Vrátí osmičkovou hodnotu čísla.

3.3.2.4.5 Ostatní funkce a příkazy pro práci s čísly

Funkce **Rnd[(Vyraz)]**

Slouží ke generování čísel. Vrátí náhodné číslo mezi 0 a 1 v podobě reálného čísla.

Příkaz **Randomize[Cislo]**

Inicializuje generátor náhodných čísel. Nepovinný parametr Cislo je číselná hodnota, která inicializuje generátor.

Funkce **Sqr(Cislo)** Vypočítá druhou odmocninu číselného výrazu.

Funkce **Sgn(Cislo)** Vrací celé číslo mezi -1 a 1, které indikuje, jestli zadané číslo bylo kladné, záporné nebo nula.

Funkce **Hex(Cislo)** Vrátí řetězec, který představuje šestnáctkovou hodnotu čísla.

Funkce **Oct(Cislo)** Vrátí řetězec, který představuje osmičkovou hodnotu čísla.

3.3.2.5 Funkce pro práci s časem

Pomocí těchto příkazů a funkcí je možné pracovat s datovými a časovými údaji. Jak jsme se již dozvěděli, tak čas v počítači je zachycen jako reálné číslo od jakéhosi bodu nula. Bodem nula byla určena půlnoc z 29. na 30. prosince roku 1899. Číslo 10 tedy značí časový údaj 0hod, 0min, 0s, 9.ledna 1900 a číslo 1,5 je poledne 31.prosince 1899. Celá čísla tedy určují počet dnů od bodu nula a desetinná čísla část dne. OpenOffice.org Basic nám umožňuje počítat rozdíly v datech a časech pomocí převodu datových a časových hodnot na číselná vyjádření. Po vypočtení rozdílu se použije speciální funkce pro převod získaných hodnot na standardní časové nebo datové formáty. Jazyk OpenOffice.org Basic také podporuje typ proměnné **Date**, která obsahuje určení času skládající se z data i času. Podrobný popis funkcí pro práci s datem a časem opět nalezneme v nápovědě, ty hlavní si zde popíšeme a jejich použití si ukážeme na příkladech.

3.3.2.5.1 Systémové datum a čas

Příkaz **Date**	Vrací aktuální systémové datum jako řetězec nebo nastaví datum. Formát data závisí na místním nastavení systému.
Funkce **Now**	Vrátí aktuální systémové datum a čas jako hodnotu typu **Date**.
Příkaz **Time**	Tato funkce vrátí aktuální systémový čas jako řetězec ve formátu "HH:MM:SS".
Funkce **Timer**	Vrátí hodnotu, která určuje, kolik sekund uběhlo od půlnoci.

Příklad 3.18 nám simuluje činnost stopek:

Příklad 3.18: Použití funkcí pro zjištění systémového času, „Stopky“

```
Sub Stopky
        MsgBox "Stisk tlačítka = Start",,"Stopky"
        Start = Timer
        MsgBox "Stisk tlačítka = Stop",,"Stopky"
        Konec = Timer
        Vysledek = Konec - Start
        MsgBox "Změřený čas: " + Vysledek,,"Výsledek měření v sekundách"
End Sub
```

Popis: V prvním dialogovém okně se po stisknutí tlačítka zahájí měření času, v druhém dialogu se po stisknutí tlačítka měření času ukončí a ve třetím

dialogu se vypíše výsledek. Takto vidí výsledek uživatel. Běh programu je poněkud jiný. Po stisknutí tlačítka v prvním dialogu se dostaneme na řádku s proměnnou Start, které přiřadíme hodnotu funkce Timer. Máme tedy uložen počáteční stav (kolik sekund uběhlo od půlnoci). Po stisku tlačítka v druhém dialogu se nám do proměnné Konec uloží konečný stav, opět díky fci Timer. Proměnná Vysledek spočítá rozdíl časů a ve třetím dialogu tento rozdíl vypíše jako výsledek (v sekundách).

3.3.2.5.2 **Funkce pro převod hodnot data a času**

Funkce **DateSerial(Rok,Mesic,Den)**

Vrátí hodnotu typu **Date** pro určený rok, měsíc, nebo den.

Funkce **DateValue [(Datum)]**

Vrací datovou hodnotu zadaného datového řetězce. Datový řetězec je kompletní datum. Toto číslo je také možné použít pro zjištění rozdílu mezi dvěma daty.

Funkce **TimeSerial(Hodina,Minuta,Sekunda)**

Vypočte číselnou časovou hodnotu ze zadaných hodin, minut a sekund v číselných parametrech. Tuto hodnotu je možné použít pro výpočet rozdílu mezi časy.

Funkce **TimeValue("HH:MM:SS")**

Ze zadané hodnoty hodin, minut a sekund (tyto parametry se zadávají jako řetězce) vypočítá číselnou hodnotu, která představuje časový údaj. Tuto hodnotu lze použít k výpočtu rozdílu mezi dvěma časy.

Funkce **Day(Cislo)** Vrací hodnotu určující den v měsíci odpovídající datu v číselné podobě (vytvořené **DateSerial** nebo **DateValue**).

Funkce **Month(Cislo)** Vrací hodnotu určující měsíc v roce odpovídající datu v číselné podobě (vytvořené **DateSerial** nebo **DateValue**).

Funkce **WeekDay(Cislo)**

Vrací hodnotu určující den v týdnu odpovídající datu v číselné podobě (vytvořené **DateSerial** nebo **DateValue**).

Funkce **Year(Cislo)** Vrací hodnotu určující rok odpovídající datu v číselné podobě (vytvořené **DateSerial** nebo **DateValue**).

Funkce **Hour(Cislo)** Vrátí hodinu z číselné časové hodnoty vygenerované funkcí TimeSerial nebo TimeValue.

Funkce **Minute(Cislo)**

Vrátí minuty z číselné časové hodnoty vygenerované funkcí TimeSerial nebo TimeValue.

Funkce **Sekond(Cislo)**

Vrátí sekundy z číselné časové hodnoty vygenerované funkcí TimeSerial nebo TimeValue.

3.3.2.6 Funkce pro práci s řetězci

Funkcí pro práci s řetězci je celá řada – od funkcí zajišťujících převádění řetězců mezi kódy ASCII a ANSI, přes funkce na úpravu řetězce až po funkce porovnávající řetězce mezi sebou. Ty nejpoužívanější si zde opět ve stručnosti popíšeme.

Mezi funkce pracující s kódy ASCII a ANSI patří funkce **Asc(Znak)**, která vypíše zadaný znak na jeho číselnou hodnotu kódu ASCII. Pokud je jako parametr zadán řetězec, převede se první znak z řetězce. Fce **Chr(Integer)** má opačnou fci – číslo z kódu ASCII převede na znak. Např. Chr(13) nám vrátí znak Enter. Funkce **Str(Integer)** převádí číselnou hodnotu na řetězec a naopak z řetězce na číslo nám hodnotu převede funkce **Val(Text)**. Fce Val(Text) převádí na číslo číselnou hodnotu zadanou na začátku řetězce, tzn. že např. Val("20 hodin, 30 minut") převede řetězec na číslo 20.

Funkcí na úpravu řetězce je hodně, mezi ty nejpoužívanější patří fce **LCase(Text)**, která převádí všechna velká písmena z řetězce na malá, opačný převod zajistí funkce **UCase(Text)**. Fce **Right(Text, Číslo)** a **Left(Text, Číslo)** vrátí počet znaků dle zadaného čísla, které jsou nejvíce vpravo, respektive vlevo. Funkce **Mid(Text,Začátek,Délka)** vrací část řetězce. Fce **Trim(Text)** smaže všechny mezery na začátku a na konci řetězce, **RTrim(Text)** a **LTrim(Text)** maže mezery pouze na konci, respektive začátku. Důležitou funkcí je také **Format(Číslo,[Formát])**. Ta převádí číslo na řetězec s možností určení jeho formátu (viz nápověda).

Mezi porovnávací fce patří **Len(Text)**, která vrací délku řetězce nebo funkce **StrComp(Text1, Text2)** porovnávající řetězce.

Užití některých funkcí je vidět na příkladu 3.19, který nám vypíše některé informace o zadaném řetězci.

Příklad 3.19: Práce s řetězci

```
Sub Retezce
        Vstup = CStr(InputBox("Zadejte jakýkoliv text"))
        Uprava = Trim(Vstup)
        DelkaTextu = Len(Uprava)
        PrvniZnak = Left(Uprava,1)
        PosledniZnak = Right(Uprava,1)
        KodPrvni = Asc(PrvniZnak)
        KodPosledni = Asc(PosledniZnak)
        MsgBox("Zadali jste text: " + Uprava + " o délce " + DelkaTextu + _
                Chr(13) + "ASCII kódy prvního a posledního znaku jsou " + _
                KodPrvni + " a " + KodPosledni)
End Sub
```

Popis: Proměnné vstup se přiřadí text zadaný uživatelem. Pro případ, že by uživatel zadal číslo, tak je tato hodnota pomocí fce CStr převedena na řetězec. Pomocí fce Trim je tento řetězec „očesán“ o mezery na začátku a konci řetězce. Fce Left a Right přiřadí proměnným první a poslední znak z řetězce a další fce Asc ASCII kód těchto znaků. Nakonec se všechny informace pomocí fce MsgBox vypíší do dialogového okna.

3.3.2.7 Příkazy a funkce nespadající do žádné katogerie

Tyto funkce najdeme v nápovědě pod oddílem **Další příkazy**. Mezi ty nejzajímavější patří např. příkaz **Wait milisekundy**, který přerušuje spuštění programu na dobu uvedenou v milisekundách nebo příkaz **Beep**, který zahraje tón na speakeru. Dále je v nápovědě v tomto oddílu uvedeno mnoho dalších funkcí, převážně pro práci s objekty.

3.3.2.8 Funkce a příkazy pro ošetření chyb

Jako poslední podkapitolu kapitoly o funkcích bychom si měli uvést ošetřování chyb v běhu programu. Možná jste si již všimli, že v našich příkazech jsou neošetřeny určité situace, které nám spustí chybu programu. Např. při stisknutí tlačítka **Zrušit** při zadávání hodnot v dialogovém okně vyvolaném fcí **InputBox.**

Chyby mohou vzniknout i při samotném psaní procedury. Programátor si nemusí všimnout překlepu, chybějící uvozovky nebo třeba záměny znaménka.

Takovéto chyby se dají ošetřit a můžeme předem určit, jak se program v takovéto situaci zachová. Můžeme si také nechat vypsat místo v kódu, kde počítač chybu objevil.

Příkaz **Option Explicit**

Tento příkaz je velmi užitečný. Uvádí se mimo naše makra(procedury), obvykle na první řádek v modulu. Má funkci automatické kontroly zápisu procedur – konkrétně kontroluje, zda jsou použité proměnné v procedurách také deklarovány. Díky takovéto možnosti kontroly se snadno odhalí překlepy v názvech proměnných.

Funkce **Erl**

Funkce **Erl** vrací číslo řádku, na kterém došlo za běhu programu k chybě. Pozor!: vrací číslo řádku od začátku modulu, nikoliv procedury.

Funkce **Err**

Tato funkce vrací celé číslo, které značí kód identifikující konkrétní chybu. Seznam chyb a jim příslušných kódů najdeme po zadání textu „chybové kódy v Basic“ v rejstříku nápovědy a nebo v popisu funkce **Error** (počet popsaných chyb je asi 64).

Funkce **Error**

Funkce Error doplňuje funkci Err. Tato fce totiž vrací chybovou zprávu(ve formě řetězce) k chybě, kterou má tato fce zadanou jako parametr ve formě chybového kódu. Praktické použití je vidět na příkladu.

Příkaz **On Error GoTo**

Jak je již z názvu tohoto příkazu patrné, bude mít něco společného s příkazem skoku **GoTo**. V podstatě má naprosto stejnou funkci jako příkaz GoTo, s jediným rozdílem – provede se pouze v případě výskytu chyby. Poté program odskočí na definované návěstí.

Příklad 3.20: Ošetření chyb v běhu programu pro výpočet odmocniny

```
Option Explicit

Sub VypocetOdmocniny
        Dim Vstup, Odmocnina As Double
        On Error GoTo Chyba
        Vstup = CDbl(InputBox("Zadej číslo: ", "Výpočet odmocniny"))
        Odmocnina = sqr(Vstup)
        MsgBox("Odmocnina z  čísla " + Vstup + " činí " + Odmocnina)
Chyba:
        MsgBox("Na řádku " + Erl + "se vyskytla chyba číslo " + Err + _
                Chr(13) + "Popis chyby:" + Error, 16, "Chybové hlášení" )
End Sub
```

Popis: V proceduře si všimněme chybového příkazu skoku **On Error GoTo Chyba,** který běh programu v případě chyby přesune na návěstí **Chyba:**, za kterým je fce MsgBox. Ta nám popíše v dialogovém okně díky funkcím **Erl, Err a Error** číslo řádku, na kterým se vyskytla chyba, číslo chyby a její popis. Tato situace může nastat pokud při vyzvání k zadání čísla zadáme cokoliv mimo čísla nebo záporné číslo (ze záporného čísla nelze odmocninu vypočítat).

3.4 Vzájemné propojení procedur

Dostali jsme se k velmi důležité kapitole. Pro řešení složitějších problémů, které bývají i velmi rozsáhlé, je výhodné tento problém rozdělit na problémy dílčí (jednotlivá makra). Je to výhodné nejen z hlediska přehlednosti, ale i z hlediska ošetření chyb. V malé proceduře chybám předejdeme snáze, než v několikastránkovém kódu programu jediného makra.

3.4.1 Makro jako podprogram jiného makra

Pro řešení složitých problémů můžeme využít toho, že jazyk Basic umožňuje vzájemnou spolupráci maker. Makro lze volat z jiného makra, podobně jako voláme funkci nebo příkaz, které lze také označit jako jakési podprogramy. Pokud zavoláme makro, které je ohraničeno klíčovými slovy **Sub** a **End Sub**, tak se toto volané makro chová jako příkaz. Pokud chceme, aby nám toto makro vrátilo i nějakou hodnotu (chovalo se jako funkce), tak makro ohraničíme místo klíčových slov **Sub** a **End Sub** klíčovými slovy **Function** a **End Function** a za název makra zapíšeme do závorek parametr, který chceme, aby byl volajícímu makru vrácen – viz příklad 3.21.

Příklad 3.21: Makro jako příkaz nebo funkce

```
Sub Makro
        MakroPrikaz
        MakroFunkce("Právě bylo zavoláno makro MakroFunkce _
                                s parametrem obsahující tento text")
End Sub

Sub MakroPrikaz
        Print "Právě bylo zavoláno makro MakroPrikaz"
End Sub

Function MakroFunkce(Parametr)
        Print Parametr
End Function
```

Popis: Spustíme makro s názvem Makro. Na prvním řádku těla tohoto makra je název jiného makra. Program toto makro bere jako příkaz a toto makro spustí. Toto makro vypíše text z parametru příkazu Print, který má v těle a skončí. Tím byl „příkaz“ Makropříkaz vykonán a program přejde na další řádek kódu. Zde vidíme volání dalšího makra MakroFunkce, avšak s parametrem v závorkách. Parametrem je v tomto případě řetězec uzavřený do závorek. Podíváme-li se na kód volaného makra MakroFunkce, vidíme, že je ohraničeno slovy Function. To nám říká, že se toto makro bude chovat jako funkce a po zadání parametru nám vrátí nějaký výsledek – v tomto případě pomocí příkazu Print vypíše hodnotu parametru a skončí. Vracíme se do našeho spuštěného makra, kde pokračujeme. Narazili jsme však na klíčová slova End Sub a procedura tedy skončí.

Na příkladu 3.21 je vidět použití obou druhů volání makra. Po zavolání makra jinou procedurou se vlastně z našeho volaného makra(Makropříkaz nebo MakroFunkce) stává podprogram volajícího makra(Makro).

4.0 Objektové programování

V předchozích kapitolách jsme se postupně dozvěděli, co jsou to makra, jejich použití a naučili jsme se konfigurovat OpenOffice.org. Naučili jsme se základním dovednostem programování v jazyku Basic a postupně jsme si popsali pravidla pro psaní maker. Také jsme vyjmenovali všechny základní prvky jazyka Basic, jako byly vestavěné funkce a příkazy a ukázali jsme si jejich použití na jednoduchých příkladech a v neposlední řadě jsme si ukázali využití maker jako podprogramů jiných maker.

Toto všechno nyní zhodnotíme v poslední kapitole této příručky, která nám odkryje možnosti programování maker přímo ovlivňující tvorbu, formátování, ukládání a mazání dokumentů a jejich jednotlivých prvků. K tomu bychom si se základními funkcemi a příkazy, které jsme si dosud popsali, nevystačili. K tomu nám pomůže práce s tzv. objekty. Objektové programování je téma velmi rozsáhlé, proto se omezíme pouze na základní popis nejpoužívanějších objektů a jejich vlastností. Vše si opět ukážeme na konkrétních příkladech, které by nám měly problematiku trochu osvětlit.

4.1 Objekt, jeho vlastnosti a metody

Pod pojmem **objekt** si můžeme představit např. buňku tabulkového procesoru, nebo list, ve kterém se tato buňka nachází. Tyto objekty mají určité **vlastnosti** a **možnosti vykonávat určité činnosti**. Pod vlastností objektu si můžeme představit např. barvu buňky nebo její formát, pod činností objektu např. výběr textu v buňce. Objekt může obsahovat jiný objekt, např. list tabulkového dokumentu obsahuje buňky.

Jak takový objekt značíme v jazyce Basic? Je to jednoduché – objekty jsou typem proměnné. Tzn. že krom základních typů, které jsme si již vyjmenovali existuje i další typ s názvem **Object**. Tyto objekty můžeme tedy deklarovat stejně jako jiné typy, např. *Dim Bunka1 As Object* . Většinou ale využíváme možnosti jazyka Basic používat proměnné bez deklarace. Objekty jsou tedy potom uloženy pod typem **Variant**.

Jednotlivé vlastnosti objektu si můžeme nechat vypsat. Můžeme je také nastavovat. Činnosti se v objektovém programování uvádějí pod pojmem metody.

Metody tedy provádějí určitou činnost. Jsou podobné příkazům, ty nám také provedli v běhu programu specifickou činnost. Vlastnosti se chovají podobně jako funkce – vracejí nám nějakou hodnotu. Pro spojení objektu s vlastností či metodou se používá tzv. tečková konvence. Vlastnost jméno listu tedy získáme např. předáním této informace proměnné JmenoListu:

JmenoListu = ObjektListu.Name

Nebo můžeme vlastnosti přímo nastavovat. Chceme-li list pojmenovat, toto pojmenování přiřadíme vlastnosti Name:

ObjectListu.Name = "Seznam studentů"

Pro pojmenování listu můžeme také využít metody setName. Zápis by tedy vypadal takto:

ObjektListu.setName("Seznam studentů")

Na zápisu si všimněme, že se zápis provádí podobně jako u funkcí do závorek, avšak jo povolen i zápis bez nich. Pro přehlednost se ovšem raději používají.

4.2 Informace o objektech

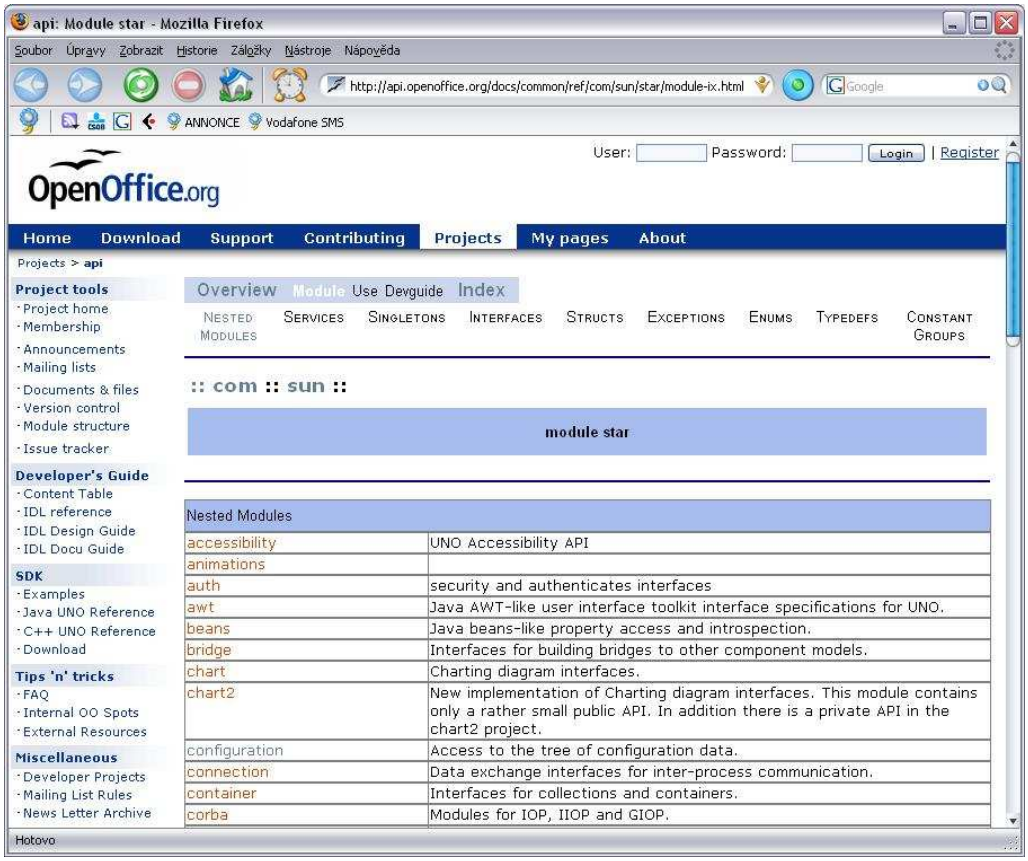
Protože metod a vlastností objektu jsou tisíce a problematika jejich popisu a funkcí je v mnoha případech velmi složitá, tak si v této kapitole ukážeme a popíšeme, kde lze popis objektů nalézt.

Aplikace OpenOffice.org využívá **univerzální síťové objekty(UNO)**. Tyto UNO využívají tzv. manažerů služeb, jejichž úkolem je vytvářet služby pro objekty. Univerzalita této technologie spočívá v tom, že s objekty vytvořenými touto technologií je možno pracovat v různých prostředích a využívat různé programovací jazyky. Technologie UNO se využívá také v přístupu k aplikaci prostřednictvím **aplikačního programovacího rozhraní (API)**. API lze přirovnat k dálkovému ovladači televize, díky němuž televizi můžeme ovládat na dálku. Stejně jako televize bude reagovat na tlačítka ovladače jen tehdy, když jim rozumí, tak i naše makra musí být zformulována tak, aby jim počítač rozuměl. Budeme-li tedy chtít ovládat objekty aplikace OpenOffice.org, jejichž vlastnosti a metody mají své jméno a syntaxi, musí být přístup k nim(objektům) zformulován tak, aby jim počítač rozuměl. K tomu slouží

popis API. Tento popis nalezneme v souboru module-ix.html na internetové adrese ***http://api.openoffice.org/docs/common/ref/com/sun/star/module-ix.html***, nebo po nainstalování balíčku pro návrháře (SDK) ve složce **OpenOffice.org_SDK/docs/common/ref/com/sun/star/module-ix.html**. Tento soubor je velmi podrobný a obsáhlý materiál pro návrháře a my se v něm omezíme pouze na informace potřebné pro práci s jazykem Basic.

4.2.1 Popis API

V popisu API jsou vlastnosti a metody objektů v souhrnu označovány jako **služby**. Každá služba popisuje objekt nebo jeho část pomocí **rozhraní** a **vlastností**. Rozhraním je chápana sada metod, které mají něco společného. Pro rozlišení rozhraní od metod a vlastností je v jejich názvu vždy na prvním místě uvedeno písmeno X.



Obr.4.1: Hlavní modul com.sun.star v popisu API

V popisu API jsou **služby** (Services), které mají něco společného, seskupeny do modulů. Tyto **moduly** (modules) jsou dále seskupeny do **skupin modulů** a tyto skupiny jsou seskupeny do **hlavního modulu**. Hlavním modulem služeb je modul **com.sun.star**. Pokud si otevřeme soubor module-ix.html, tak by se nám měl tento hlavní modul zobrazit (Obr.4.1). Najdeme ho také pod odkazem Overview v horní nabídce.

Struktura tohoto modulu se nám objeví na zobrazené stránce. Jsou zde vypsány jednotlivé odkazy na skupiny modulů s jejich krátkým popisem. Klikneme-li na jeden z těchto odkazů, objeví se seznam jednotlivých modulů a po otevření některého z odkazů na modul se nám zobrazí jeho popis.

Prvky použité v popisu API:

- Moduly (Modules)

Moduly slouží pro sjednocení služeb podobného zaměření do větších celků.

- Služby (Services)

Označení služba se používá pro objekty API. Do úplného názvu služby se zahrnují pomocí tečkové notace všechny rodičovské moduly např. **com.sun.star.sheet.Spreadsheet**.

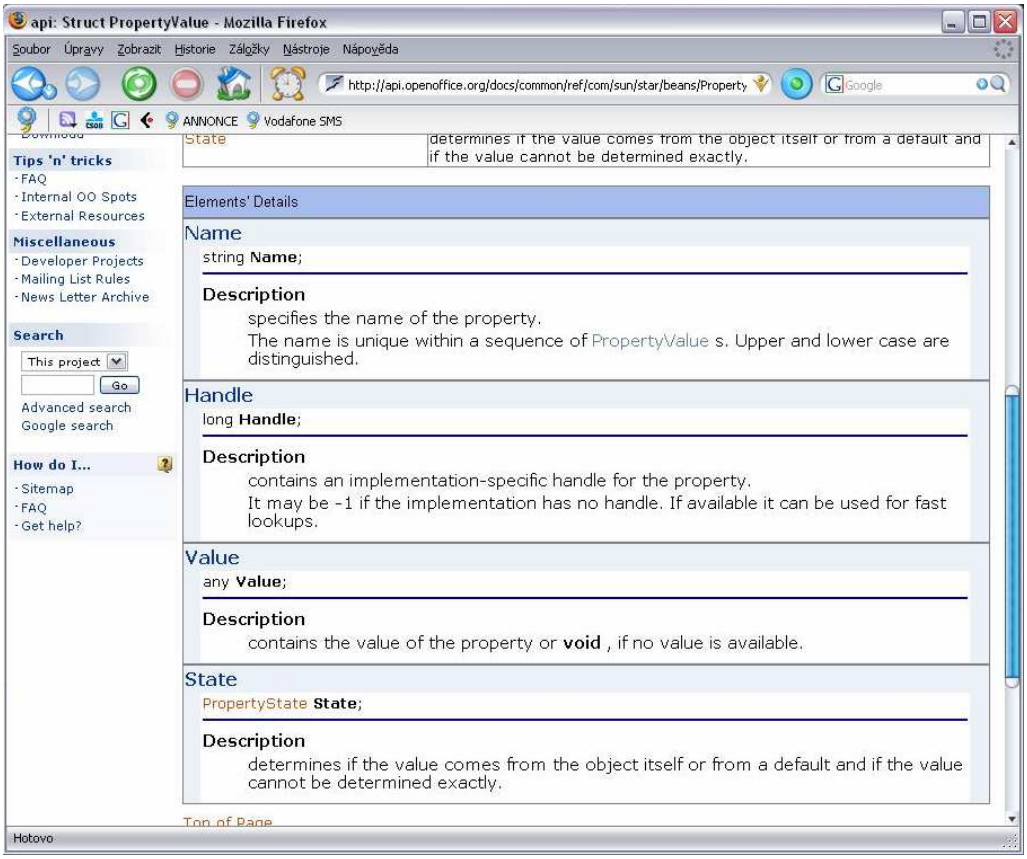
- Rozhraní (Intreface)

V jednotlivých rozhraních jsou sjednoceny služby, které spolu nějakým způsobem souvisejí. Název rozhraní vždy pro odlišení od ostatních prvků začíná písmenem X.

- Struktury (Structs)

Struktury umožňují zápis několika hodnot různých typů do jediné proměnné.

Např. pokud si v popisu API nalezneme strukturu **com.sun.star.beans.PropertyValue** obsahuje prvky zobrazené na obr.4.2.



Obr.4.2: Popis struktury com.sun.star.beans.PropertyValue

- Sekvence (Sequence)

Sekvence jsou podobné jako pole. Jsou to kolekce prvků, v tomto případě objektů. Stejně jako datová pole se musejí vždy před prvním použitím deklarovat.

Jak je z popisu API vidět, tak informace v něm jsou tak obsáhlé, že vysvětlení jednotlivých prvků by vydalo na minimálně 2. další takovéto příručky. My si zde ukážeme pouze postupy objektového programování. Dále si ukážeme jak a kde nalézt správné informace a na jednoduchých příkladech si ukážeme nejpoužívanější metody pro práci s tabulkovým dokumentem, kde je použití maker asi nejpoužívanější. Pokud by si programátor chtěl udělat větší přehled, je nutné podrobné prostudování popisu API.

4.3 Práce s tabulkovým dokumentem

Jaké metody můžeme použít pro práci se samotným dokumentem? Tak určitě nás napadne samotné vytvoření nebo otevření dokumentu (metoda **LoadComponentFromURL**), uložení dokumentu (metoda **store**), uložení do přesného umístění (metoda **storeToURL nebo storeAsURL**) nebo zavření dokumentu (pomocí metody **dispose**). Použití si vysvětlíme opět na příkladech. Makro z příkladu 4.1 nám vytvoří a otevře nový tabulkový dokument.

Příklad 4.1: Vytvoření tabulkového dokumentu

```
Sub NovyDokument
        Dim VlastnostiDok()
        StarDesktop.loadComponentFromURL("private:factory/scalc",_
                "_blank",0, VlastnostiDok())
End Sub
```

Popis: Nejprve jsme si nadeklarovali prázdnou sekvenci VlastnostiDok(), kterou budeme potřebovat jako parametr metody pro vytvoření dokumentu. Se sekvencemi se setkáme i v dalších příkladech, kde si je podrobněji vysvětlíme. Dále používáme metodu LoadComponentFromURL , která má čtyři parametry. První parametr je cesta k dokumentu, který chceme otevřít. Lze také použít text, který vidíme v příkladu, potom se dokument vytvoří nový. Další parametr nám zajistí vytvoření rámečku, poslední parametr udává ve formě sekvence vlastnosti dokumentu.

Na příkladu 4.1 vidíme použití metody **LoadComponentFromURL** se čtyřmi parametry. Informace o této metodě získáme z popisu API buďto v rejstříku, nebo pokud známe umístění metody ve službě či rozhraní, tak v tomto umístění. Konkrétně tuto metodu nalezneme ve službě com.sun.star.frame.Desktop v rozhraní XcomponentLoader. Z popisu zjistíme informace o parametrech metody. Pokud bychom chtěli **otevřít** již existující dokument, tak v prvním parametru zadáme jeho cestu ve formátu "**file:///C:/Dokumenty/Příklad1.ods**" (Přípona .ods je příponou aplikace Calc).

Příklady 4.2 a 4.3 nám ukáží možnosti jak dokument uložit, popřípadě mu přiřadit i nějaké vlastnosti.

Příklad 4.2: Uložení dokumentu

```
Sub UlozeniDokumentuDoURL
```

```
    Dim VlastnostiDok()
    AktualniDokument = ThisComponent
    Nazev = InputBox("Zadejte název dokumentu")
    AktualniDokument.storeToURL("file:///C:/" + Nazev, VlastnostiDok() )
    Print "Dokument byl uložen do C:/" + Nazev
End Sub
```

Popis: Proměnné AktualniDokument přiřadíme hodnotu metody ThisComponent. Tato metoda vrací hodnotu typu Object a představuje aktuálně otevřený dokument. Do okna napíšeme název dokumentu (název píšeme včetně přípony .ods pro aplikaci Calc: např. Příklad1.ods). Pod tímto názvem se pomocí metody **storeToURL** náš dokument uloží v našem případě přímo na disk C:. Metoda má dva parametry, první je kompletní cesta, druhý opět sekvence vlastností. Metoda je zaměnitelná s metodou **storeAsURL**, která má funkci jako příkaz v menu nabídek Soubor → Uložit jako.

Příklad 4.3: Uložení dokumentu s přiřazením vlastnosti – hesla.

```
Sub UlozeniDokumentuDoURL2
    Dim VlastnostiDok(0) As New com.sun.star.beans.PropertyValue
    AktualniDokument = ThisComponent
    Nazev = InputBox("Zadejte název dokumentu")
    Heslo = InputBox("Zadej heslo")
    VlastnostiDok(0).Name = "Password"
    VlastnostiDok(0).Value = Heslo
    AktualniDokument.storeToURL("file:///C:/" + Nazev, VlastnostiDok())
    Print "Dokument byl uložen do C:/" + Nazev
    AktualniDokument.dispose
End Sub
```

Popis: Příklad 4.2 jsme rozšířili o přiřazení vlastnosti zaheslování dokumentu před uložením. Zde právě využíváme sekvence VlastnostiDok(0), kde nám číslo značí počet prvků sekvence. Indexováno je od nuly, proto tato sekvence obsahuje 1.prvek typu Object i když má uvedenou 0 jako parametr (číslo 2 by značilo 3 parametry). V makru si všimněme, že prvku s indexem 0 nastavujeme vlastnosti "Password" hodnotu proměnné Heslo, do které zadáváme text z dialogu funkce InputBox. Metoda **dispose** nám dokument zavře.

V příkladu 4.3 si všimněme deklarace sekvence VlastnostiDok(0), kde sekvenci deklarujeme jako sekvenci struktury com.sun.star.beans.PropertyValue. Slovo New označuje vytvoření nového objektu této struktury. V předchozích příkladech stačilo prázdné sekvence nadeklarovat jako typ Variant, protože s žádnými vlastnostmi nepracovali. Dále si všimněme způsobu přiřazování hodnot vlastnostem. Seznam a popis všech vlastností pro práci s dokumentem najdeme v popisu API ve službě **com.sun.star.document.Mediadescriptor**. Na konci příkladu si všimněme metody **dispose,** jež zavírá aktuální dokument, pro který je volána prostřednictvím tečkové konvence. Pokud bychom si nyní zkusili otevřít dokument, který jsme uložili pomocí

našeho makra do umístění C:/(název souboru, který jsme zadali…), tak tento dokument po nás bude požadovat před otevřením námi zadané heslo…

4.3.1 Metody a vlastnosti pro práci s listy tabulkového dokumentu

Listy tabulkového dokumentu jsou také objekty. Mají mnoho vlastností, jako je název, typ a barva písma, záhlaví a zápatí, nebo např. uzamknutí listu. S listy dokumentu můžeme také manipulovat – vytvářet, přesunovat či odebírat. Pro všechny tyto atributy existuje opět velká řádka metod a vlastností.

Listů může mít dokument nespočet. Pro práci s listy byl vytvořen kontejner, který tyto objekty (listy) sdružuje. Pomocí metod můžeme zjistit počet listů v kontejneru (= počtu listů v dokumentu) a vypsat jejich názvy. Listy zde také můžeme vytvářet, kopírovat, přesouvat a odstraňovat. Existují také metody, které nám ověří existenci listů určitého jména, nebo zjistí, který list je právě aktivní.

Metoda **getSheets** Touto metodou získáme objekt kontejneru z dokumentu.

Metoda **getCount** Zjišťuje počet listů v dokumentu(kontajneru).

Metoda **getElementNames**

Vrátí jména všech listů ve formě pole o velikosti počtu listů v dokumentu. Jednotlivá jména získáme jako prvky tohoto pole – vypsat je můžeme např. pomocí příkazu For.

Metoda **getByName(Jmeno)**

Slouží pro získání objektu listu prostřednictvím jména. Např:

List = Listy.getByName(“Seznam“)

Proměnné List přiřadíme objekt listu s názvem Seznam z kontejneru Listy.

Metoda **getName** Získá jméno konkrétního objektu listu. *Print List.getName*

Metoda **getByIndex(Index)**

Slouží pro získání objektu listu prostřednictvím indexu.

Metoda **insertNewByName(Jmeno,Index)**

Vloží do dokumentu nový list se jménem Jmeno a pod indexem Index.

Metoda **hasByName(Jmeno)**

Zjistí, zde se v dokumentu nachází list se jménem Jmeno. Návratová hodnota je typu boolean.

Metoda **copyByName(Jmeno1,Jmeno2)**

Kopíruje list jména Jmeno1 na list se jménem Jmeno2.

Metoda **moveByName(Jmeno, Index)**

Přesune list jména Jmeno na indexové místo Index.

Metoda **setName(Jmeno)**

Přejmenuje list na list se jménem Jmeno.

List1.setName(List2)

Metoda **removeByName(Jmeno)**

Odstraní list se jménem Jmeno.

Metoda **getCurrentController**

Vytvoří z dokumentu objekt obsahující informace o jeho stavu. S tímto objektem lze dále pomocí dalších metod pracovat.

Metoda **getActiveSheet**

Z objektu vytvořeného metodou getCurrentController vrací jméno aktivního listu dokumentu. Viz. Příklad 4.4.

Příklad 4.4: Ukázka použití metod pro práci s listy dokumentu

```
Sub AktivniList
    KontajnerListu = ThisComponent.getSheets
    PocetListu = KontajnerListu.getCount
    Print "Počet listů zjištěný metodou getCount: " + PocetListu
    KontajnerListu.insertNewByName("Nový list", PocetListu + 1)
    Print "pomocí metody insertNewByName vložen další list..."
    Jmeno = InputBox("Zadejte jméno nového listu: ")
    NovyList = KontajnerListu.getByName("Nový list")
    NovyList.setName(Jmeno)
    Print "pomocí metody setName byl Nový list přejmenován na " + Jmeno
    ObjektDok = ThisComponent.getCurrentController
    Print "Aktivní list je " + ObjektDok.getActiveSheet.getName
    Print "Nyní pomocí metody select zaktivujeme list " + Jmeno
    ObjektDok.select(NovyList)
End Sub
```

Popis: Proměnné KontajnerListu přiřadíme pomocí metody **getSheets** objekt kontejneru listů. Dále je ukázka použití metod **getCount**, **insertNewByName**, **getByName** a **setName**.
Pro určení aktivního listu je třeba nejdříve pomocí metody **getCurrentController** vytvořit z dokumentu objekt. Tento objekt obsahuje všechny informace o dokumentu a také je může ovlivňovat. Objekt aktivního listu potom vrací metoda **getActiveSheet** a jméno tohoto objektu získáme metodou **getName**

Listy mají také mnoho vlastností. Tyto vlastnosti si můžeme pomocí metod nechat vypsat nebo je také nastavovat a měnit. Vlastnosti lze dle způsobu práce s nimi rozdělit:

Na vlastnosti, které se zjišťují a nastavují pomocí metod **getPropertyValues** a **setPropertyValues**. Vlastnosti lze také připojit přímo k objektu např. *TypPisma = AktivniList.CharFontName* připojuje vlastnost **CharFontName** aktivnímu listu. Lze ji potom vypsat např pomocí příkazu Print: *Print TypPisma*

Na vlastnosti, které se zjišťují a nastavují pomocí speciálních metod určených pouze těmto vlastnostem. Metody mají potom název jako vlastnost, pouze je před nimi uvedeno slovíčko **set** (nastav) nebo **get** (jdi). Např. **setPrintTitleColums** nastaví opakovaný tisk záhlaví sloupců. Tyto vlastnosti lze stejně jako u první skupiny taktéž přiřadit přímo objektu.

Na vlastnosti, které nelze objektu přiřadit, pouze nám zjišťují stav vlastnosti např. metoda **IsProtected.** Tyto vlastnosti je pak nutné nastavovat pomocí dalších metod, v tomto případě list zamkneme či odemkneme pomocí metod **protect** a **unprotect**.

Příklady metod pro práci s vlastnostmi:

Metoda **getProperties** Vrací pole vlastností listu

Metoda **isProtected** určí zda je list uzamčen

Metoda **protect** a **unprotect** Zamkne či odemkne list

Metoda **getPropertyValue(JmenoVlastnosti)**

Zjistí nastavení vlastnosti JmenoVlastnosti

Metoda **setPropertyValue(JmenoVlastnosti, Hodnota)**

Nastaví vlastnost JmenoVlastnosti na Hodnota

4.3.2 Vlastnosti a metody pracující s buňkami

Postupně jsme se propracovali k objektům, které jsou v tabulkovém dokumentu nejpoužívanější – k buňkám. Buňkám lze stejně jako listům nastavovat vlastnosti a pracovat s nimi pomocí metod. Často se stává, že buňky v nějaké oblasti – např. v tabulce – mají vlastnosti stejné. Abychom nemuseli pro každou buňku v takovéto oblasti nastavovat vlastnosti zvlášť, existují i objekty, které buňky sdružují. Jedná se o objekty výběru buněk – řádky, sloupce, skupiny řádků a sloupců nebo také pravoúhlé výběry buněk, které může sdružovat buňky např. v tabulce uprostřed listu. Příklad 4.5, který zapíše uživatelem zadaný text do uživatelem určené buňky, nám použití některých metod a vlastností ukáže.

Příklad 4.5: Zápis zadaného textu do určené buňky

```
Sub ZapisDoBunky
        Pozice = InputBox("Zadej pozici buňky (např. A2): ")
        ZadanyText = InputBox("Zadej text: ")
        AktivniList = ThisComponent.getCurrentController.getActiveSheet
        Bunka = AktivniList.getCellRangeByName(Pozice)
        Bunka.String = ZadanyText
End Sub
```

Popis: Tento příklad zapisuje zadaný text do určené buňky. Pomocí metod **ThisComponent**, **getCurrentController** vytvoříme z aktivního dokumentu objekt pro práci a metodou **getActiveSheet** dostaneme z tohoto dokumentu aktivní list. Dále proměnné **Bunka** přiřadíme pomocí metody **getCellRangeByName** pozici zadanou uživatelem v dialogu a nakonec proměnné **Bunka** nastavíme vlastnost **String** na text zadaný uživatelem.

Na příkladu 4.5 si všimněme způsobu zjištění aktivního listu, dále metody **getCellRangeByName**, které se zadává parametr ve formě názvu pozice buňky (A2, C13, D1, …). Tato funkce lze zaměnit funkcí **getCellByPosition**, která má dva parametry, které určují pozici buňky. První parametr určuje pořadí sloupce, druhý pořadí řádku. Indexováno je opět od nuly (=> první řádek má index 0, druhý 1, …). Tuto metodu lze použít i pro výběr oblasti buněk – to má metoda čtyři parametry označující pořadí počátečního sloupce, počátečního řádku, koncového sloupce a koncového řádku. Nakonec si všimněme přiřazení vlastnosti **String** naší určené buňce. Tato vlastnost načítá nebo ukládá textové hodnoty v buňce. Kdybychom chtěli ukládat číselné hodnoty, tak bychom použili vlastnost **Value**, pro vzorce vlastnost **Formula**.

Některé další nejužívanější metody a vlastnosti pro práci s buňkami:

Metoda **getCurrentSelection**.

Tato metoda vrací objekt právě aktivní buňky.

Metoda **getCellAdress** Vrací nám informaci o adrese buňky. Tato informace je ve formě struktury a obsahuje index listu, číslo sloupce a číslo řádku.

Metoda **getStart** Tato metoda nastaví kurzor na začátek buňky.

Metoda **getEnd** Tato metoda nastaví kurzor na konec buňky.

Metoda **getText** Vrací text zapsaný v buňce.

Metoda **insertString(PoziceKurzoru, Text, TrueOrFalse)**

Tato metoda vkládá text druhého parametru Text do buňky. První parametr určí pozici kurzoru a poslední parametr ve formě True nebo False určí, zda se již existující text přepíše či zachová a nový text se k němu pouze připojí…

Vlastnost **CellStyle** Touto vlastností zjišťujeme a nastavujeme styl buňky. Styly buňky zjistíme zobrazením této vlastnosti v popisu API.

Vlastnost **CellBackColor**

Tato vlastnost nastavuje barvu pozadí buněk.

Vlastnost **VertJustify** Touto vlastností můžeme určit zarovnání textu v buňce. Možnosti jsou STANDARD, TOP, CENTER a BOTTOM, které se předávají službě com.sun.star.table.CellVertJustify.

Bunka.VertJustify = com.sun.star.table.CellVertJustify.TOP

tedy zarovná text v buňce nahoru. Další možností je použít

Bunka.VertJustify = 1 ,

kdy číslo 1 určuje ve výčtovém typu hodnotu TOP.

Seznam všech vlastností buňky můžeme najít v popisu API ve službě **com.sun.star.table.CellProperties**, kde najdeme i jejich popis. Tyto vlastnosti se nastavují pomocí struktur jak jsme již uvedli v příkladu 4.3 u nastavení vlastnosti Password. Většina těchto vlastností je použitelná i pro oblasti buněk, které si stručně popíšeme nyní:

Příklad 4.6: Součet hodnot čísel buněk v zadané oblasti

```
Sub SoucetCisel
    AktivniList = ThisComponent.getCurrentController.getActiveSheet
    Vstup = InputBox("Zadejte buňky ohraničující výběr oblasti pro součet _
            čísel(např. "B5:E10")")
    Oblast = AktivniList.getCellRangeByName(Vstup)
    Soucet = Oblast.computeFunction(com.sun.star.sheet.GeneralFunction.SUM)
    ThisComponent.getCurrentController.select(Oblast)
    Print "Součet čísel ve vybrané oblasti buněk je " + Soucet
End Sub
```

Popis: Opět si na začátku příkladu vytvoříme objekt AktivniList, se kterým budeme dále pracovat. Pomocí parametrů metody **getCellRangeByName** vybereme oblast buněk (zadanou uživatelem ve formě vstupu) a přiřadíme proměnné Oblast. Pro tuto proměnnou dále zavoláme metodu **computeFunction** s parametrem SUM služby **com.sun.star.sheet.GeneralFunction**, který slouží pro součet hodnot a tento součet si na konci makra vypíšeme. Všimněme si ještě předposledního řádku těla programu – zde pro ilustraci pomocí metody **select** vybereme oblast buněk v dokumentu a ten se v dokumentu zvýrazní jako výběr.

V příkladu 4.6 si všimněme metody pro výběr pravoúhlé oblasti **getCellRangeByName**, která má parametr ve tvaru "PrvniBunkaVyberuVlevoNahore**:**PosledniBunkaVyberuVpravoDole". Dále si všimněme, že další nová metoda **computeFunction** volá jako parametr výčtový typ **GeneralFunction** ze služby **com.sun.star.sheet.GeneralFunction** a z tohoto výčtového typu volá funkci SUM. Tento výčtový typ obsahuje i jiné tabulkové funkce, ty najdeme opět v popisu API právě ve službě **com.sun.star.sheet.GeneralFunction**. Stejně jako u výčtového typu CellVertJustify (viz kap.4.3.2) je i zde možnost zadávat parametry ve formě čísla. Funkci TOP odpovídá pak číslo 2 a zapisujeme: computeFunction(2).

Na konci této kapitoly, která je opravdu jen úvodem do velmi rozsáhlého tématu – objektového programování v makrech v aplikacích OpenOffice.org, si ještě uvedeme a popíšeme jeden příklad. Příklad 4.7 nám ukáže, jak prakticky využít metod pro filtrování a exportování dat. Tento příklad bude z dat v tabulkovém dokumentu vypisovat požadované údaje dle klíčových slov. Více na obrázku 4.3 a v jeho popisu.

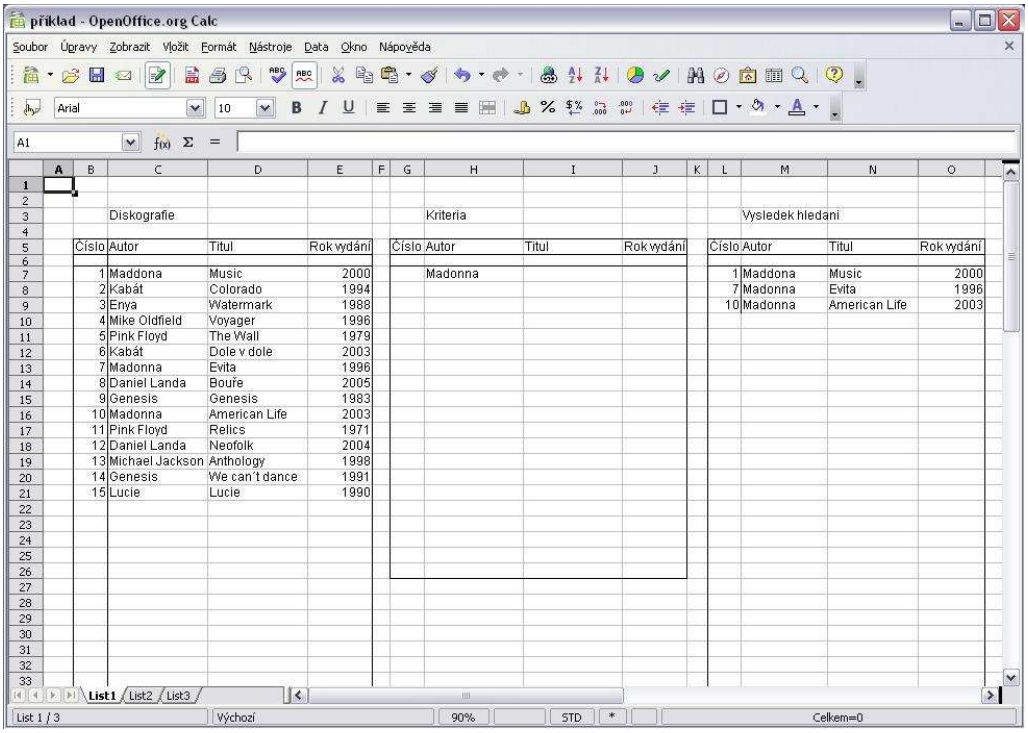
Příklad 4.7: Vyhledávání a exportování dat

```
Sub Vyhledavani

    Dim AdresaCile As New com.sun.star.table.CellAddress

    AktivniList = ThisComponent.getCurrentController.getActiveSheet
    OblastVyhledavani = AktivniList.getCellRangeByName("B5:E71")
    OblastKriterii = AktivniList.getCellRangeByName("G5:J26")

    AdresaCile.Sheet = 0
    AdresaCile.Column = 11
    AdresaCile.Row = 4

    Popis = OblastKriterii.createFilterDescriptorByObject(OblastVyhledavani)
    Popis.ContainsHeader = True
    Popis.CopyOutputData = True
    Popis.OutputPosition = AdresaCile

    OblastVyhledavani.filter(Popis)

End Sub
```

Popis: Na počátku příkladu si všimněme deklarace proměnné AdresaCile na objekt ze služby **com.sun.star.table.CellAddress**. Tato služba představuje strukturu **CellAddress** (proto povinná deklarace), která určuje přesnou adresu buňky. Opět si přiřadíme aktivní list a dvě oblasti buněk: OblastVyhledavani nám představuje oblast s vloženými daty a OblastKriterii tabulku pro zadávání kriterií pro vyhledávání. Dále přiřadíme vlastnostem proměnné AdresaCile hodnoty. Vlastnost **Sheet** představuje index listu, **Column** index sloupce a **Row** index řádky buňky, kterou chceme přesně adresovat. V příkladu to představuje levou horní buňku oblasti, do které budeme v konečném výsledku exportovat data z oblasti vyhledávání. Způsob filtrování dat určuje objekt Popis, který vytvoříme z objektu oblasti kriterií (OblastKriterii) metodou **createFilterDescriptorByObject(OblastVyhledavani)**. Parametr této metody určuje z jaké oblasti bude filtrování prováděno. Vlastnostem objektu (Popis) přiřadíme hodnoty:

- True u vlastnosti **ContainsHeader** určuje, že první řádek tabulky bude představovat záhlaví.
- True u vlastnosti **CopyOutputData** určuje, že data budou filtrována mimo filtrovanou tabulku.
- AdresaCile u vlastnosti **OutputPosition** určuje cíl, kam se uloží vyfiltrovaná data.

Na úplný konec si data pomocí metody **filter(Popis)** vyfiltrujeme z tabulky dat (OblastVyhledavani) dle zadaného parametru(objekt popisující způsob filtrace) této metody. Výsledek příkladu vyhledávání je vidět na obrázku 4.3.



Obr.4.3: Příklad použití makra z příkladu 4.7 na tabulkovém dokumentu

Popis obrázku: List tabulkového dokumentu na obrázku 4.3 obsahuje 3 tabulky. Do levé tabulky uživatel zadává své údaje – v tomto případě si zapisuje diskografii vlastněných zvukových alb a přiřazuje jim pořadová čísla. Prostřední tabulka představuje tabulku pro zadávání kriterií pro filtraci dat a do pravé tabulky se tyto data vypisují. Na obrázku je konkrétní výběr dat dle zadaného kritéria (jméno autora). Tento výpis dat do pravé tabulky se uskutečnil po zadání kritéria do prostřední tabulky a po spuštění makra z příkladu 4.7 (popis a průběh programu je popsán v popisu příkladu)

Tímto bych tuto poslední kapitolu „úvodu“ do objektového programování maker uzavřel. Zájemce o další studium této problematiky odkazuji na zdroje uvedené na konci této práce. Hlavně na seriál o tvorbě maker na slovenském serveru ***http://www.inet.sk/tema/makra*** a příklady maker zveřejněné na internetové adrese ***http://ooomacros.org/user.php***. Dále je nutné řádné prostudování popisu API a nakonec odkazuji i na materiály uložené v příloze (na CD), která je přiložena u této práce.

Shrnutí a závěr

Cílem této práce bylo vytvořit uživatelskou příručku, která by podrobně popsala od úplného začátku, co jsou to makra, co je to OpenOffice.org, k čemu makra slouží a uvedla by postupy tvorby maker v OpenOffice.org krok za krokem. Při tvorbě této příručky byl předpoklad, že čtenář nemá předchozí zkušenosti s programováním maker, avšak s kancelářskou aplikací OpenOffice.org se již setkal. Vzhledem k šíři tématu a omezené délce této práce jsem se omezil hlavně na podrobný popis programovacího jazyka BASIC, kterého aplikace OpenOffice.org pro programování maker používá. Všechny typy vysvětlovaných prvků jsem se snažil vždy ukázat na praktickém příkladu (programu) a v jeho popisu tento příklad podrobně vysvětit. Takovýchto příkladů je v práci bezmála třicet.

V úvodních dvou kapitolách jsem se snažil čtenáři přehledně a srozumitelně popsat, co jsou to makra a proč je v kancelářských aplikacích výhodné je používat. Je zde také ukázáno, jak lze takováto makra jednoduše v aplikaci nahrát. Dále jsem popsal postupy, jak makro přiřadit jednotlivým prvkům aplikace OpenOffice.org a způsoby, které aplikace používá pro ukládání maker – kam, jak a proč se makro ukládá.

Ve třetí kapitole jsem rozebral problematiku samotného programování maker. Čtenář se zde dozví, že makro je program, jehož jazyk (BASIC) má svá pravidla. Tyto pravidla jsou v této kapitole podrobněji popsána. Dále jsou zde popsány jednotlivé prvky programovacího jazyka Basic s podrobným popisem jednotlivých typů příkazů a funkcí. Tato kapitola je doplněna průběžně o množství praktických programů (příkladů), na kterých je v jejich popisu podrobně popsána problematika a způsob použití jednotlivých prvků jazyka Basic. Na konci kapitoly se zmiňuji o důležité možnosti vzájemného propojení maker a jak tohoto v programování využít.

V poslední kapitole se zmiňuji o objektovém programování. Toto téma je velmi široké a samo o sobě by mohlo být tématem velmi obsáhlé diplomové práce. Proto jsem se snažil hlavně čtenáři osvětlit, co je to objektové programování a jak se v programování maker využívá. Rozebírám zde hlavně, kde o metodách a vlastnostech objektů nalézt informace a na závěr uvádím příklady a popis použití objektového

programování v tabulkovém dokumentu. Na názorných příkladech by mělo být patrné, že jako objekt lze chápat nejen celý dokument , ale i jeho části – listy, buňky, výběry buněk, …

Použitím maker v kancelářských aplikacích lze rozšířit jejich základní funkce o prakticky neomezené možnosti a zefektivnit práci v každodenně používaných dokumentech. Je pouze na schopnostech a míře znalostí uživatele, jak nedostatky aplikace, které pro svoji práci v dokumentech nalezl, bude schopen pomocí naprogramování svých maker odstranit. K uvedení do této problematiky by mu měla posloužit tato práce – uživatelská příručka programování maker v OpenOffice.org.

Seznam použitých zkratek

tzn..... To znamená

tzv ..... Tak zvané

např ..... Například

fce..... Funkce

Seznam použitých pojmů

API ..... [Application Programming Interface] – rozhraní pro programování aplikací.

Basic ..... [Beginner's All-purpose Symbolic Instruction Code] – rodina programovacích jazyků vysoké úrovně, původně určené pro výuku programovacích jazyků, nyní využívaných hlavně jako jazyk pro tvorbu maker.

C++ ..... Objektově orientovaný programovací jazyk.

DMATHS.… ..... Francouzský makromodul, který vkládá do textového dokumentu grafy funkcí.

XML ..... Jazyk, určený pro výměnu dat mezi aplikacemi a pro publikování dokumentů.

OpenDokument..... Standardizované formáty dokumentů s otevřenou specifikací a volným použitím v jakékoliv aplikaci, která je podporuje.

OpenOffice.org..... Kancelářský balík podobný balíku MS Office.

Open Source ..... Počítačový program s otevřeným zdrojovým kódem a legální dostupností.

Subroutine ..... Podprocedura.

UNO ..... [Universal Network Objects] - univerzální síťové objekty.

Seznam použitých zdrojů

Internet:

[1] http://cs.wikipedia.org/wiki/Openoffice

[2] http://api.openoffice.org/docs/common/ref/index-files/index-1.html

[3] http://www.osu.cz/katedry/kip/aktuality/sbornik99/fussek.html

[4] http://www.david-zbiral.cz/PC-tipy.htm#makra

[5] http://www.inet.sk/clanok/2728/makra-v-openofficeorg-i-zaciname

[6] http://www.openoffice.org/

[7] http://www.openoffice.cz/

[8] http://cs.wikipedia.org/wiki/Hlavn%C3%AD_strana

[9] http://scripting.openoffice.org/

[10] http://www.opensource.org/

[11] http://ooomacros.org/user.php

[12] http://www.go-cybernet.estranky.cz/clanky/navody_tipy-a-rady/jaky-mam-pouzit-software-misto-wordu_

Literatura:

[13] **OpenOffice.org: Podrobná uživatelská příručka**, Lapáček Jiří, Computer Press, 2004, ISBN: 80-251-0360-9, CD-ROM

[14] **OpenOffice.org: Uživatelská příručka**, Pavel Sobek, Computer Press, 2003, ISBN: 80-7226-867-8, CD-ROM

[15] **OpenOffice.org: tipy a triky pro záznam a úpravu maker**, Pavel Sobek, Grada Publishing, Praha 2006, ISBN: 80-247-1374-8

[16] **OpenOffice.org pro zelenáče**, Pavel Satrapa, Neocortex spol. s. r. o., 2004, ISBN: 80-86330-15-X

Přílohy a jejich obsah:

1. CD-ROM:
   - bakalářská práce ve formátu .PDF
   - instalace OpenOffice.org 2.2
   - příklady, tabulky a obrázky použité v práci
   - další studijní materiály pro problematiku programování maker v OpenOffice.org