



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Vysoká škola báňská – Technická univerzita Ostrava



PROGRAMOVACÍ TECHNIKY

učební text

David Fojtík

Ostrava 2012

Recenze: Ing. Miroslav Mahdal Ph.D.

Mgr. Jan Veřmiřovský

Název: Programovací techniky

Autor: David Fojtík

Vydání: první, 2012

Počet stran: 134

Náklad: 20

Studijní materiály pro studijní obor Aplikovaná informatika a řízení Fakulty strojní

Jazyková korektura: nebyla provedena.

Určeno pro projekt:

Operační program Vzdělávání pro konkurenceschopnost

Název: Personalizace výuky prostřednictvím e-learningu

Číslo: CZ.1.07/2.2.00/07.0339

Realizace: VŠB – Technická univerzita Ostrava

Projekt je spolufinancován z prostředků ESF a státního rozpočtu ČR

© David Fojtík

© VŠB – Technická univerzita Ostrava

ISBN 978-80-248-2608-0

OBSAH

1	ÚVOD	7
1.1	Počátky moderní výpočetní techniky a software	7
1.2	Programovací jazyk	9
2	AUTOMATIZACE PRÁCE V MS OFFICE ZÁZNAMEM MAKER .	11
2.1	Makra.....	11
2.2	Záznam maker v prostředí MS Excel 2010	12
2.3	Používání maker v prostředí MS Excel 2010	15
2.4	Omezení záznamu maker.....	23
3	EDITACE MAKER PROGRAMOVÝM ZÁPÍSEM.....	26
3.1	Editor jazyka Visual Basic for Application	27
3.2	Programové založení makra	32
3.3	Zápis programového kódu	34
3.4	Test spuštění makra z prostředí editoru jazyka Visual Basic	36
4	ÚVOD DO PROGRAMOVÁNÍ V JAZYCE VISUAL BASIC FOR APPLICATION	39
4.1	Úvod do jazyka Visual Basic.....	40
4.2	Příkazy dialogů (funkce) MsgBox() a InputBox()	44
4.3	Vývojové diagramy	47
5	PROMĚNNÉ A DATOVÉ TYPY	50
5.1	Automatické proměnné.....	50
5.2	Deklarace proměnných	53
5.3	Datové typy.....	55
5.4	Obor platnosti proměnných	57
5.5	Základní operátory.....	57
5.6	Výjimky při práci s proměnnými.....	58
6	LADĚNÍ PROGRAMOVÉHO KÓDU	63
6.1	Metody zahájení ladění.....	64
6.2	Sledování hodnot proměnných – analýza vzniku výjimky.....	65
6.3	Krokování	69
7	VĚTVENÍ A LOGICKÉ VÝRAZY	75
7.1	Logické výrazy a proměnná Boolean	75
7.2	Větvení pomocí funkce If.....	79
7.3	Větvení pomocí konstrukce If-Else	80
7.4	Větvení pomocí konstrukce Select-Case	83
8	CYKLY - OPAKOVÁNÍ ČINNOSTÍ	88
8.1	Inkrementační cyklus For – Next	89
8.2	Logický cyklus Do-Loop s podmínkou na začátku	92
8.3	Logický cyklus Do-Loop s podmínkou na konci	94
8.4	Logický cyklus Do-Loop s podmínkou uprostřed	95
8.5	Cyklus procházení kolekcí.....	97

9	PROGRAMOVÉ PŘÍSTUPY K BUŇKÁM	102
9.1	Programový výběr buněk a oblastí	102
9.2	Programový přístup k hodnotám buněk a jejich editace.....	104
9.3	Procházení oblastí.....	105
10	FUNKCE, SUBROUTINY A ARGUMENTY	110
10.1	Definice Funkcí	111
10.2	Volání funkcí v kódu:	112
10.3	Volání funkcí v tabulkách MS Excelu.....	113
10.4	Definice subrutiny	114
10.5	Subrutiny s argumenty.....	115
10.6	Předávání argumentů odkazem a hodnotou.....	116
10.7	Globální proměnné	117
10.8	Ladění subrutin a funkcí	118
10.9	Buňka a oblast jako argument funkce.....	120
10.10	Systémové funkce	121
	Rejstřík	130

POKYNY KE STUDIU

Programovací techniky

Pro předmět **Programovací techniky**. Zimního semestru jste obdrželi studijní balík obsahující

- integrované skriptum pro distanční studium obsahující i pokyny ke studiu
- CD-ROM s doplňkovými animacemi vybraných částí kapitol

Prerekvizity

Pro studium tohoto předmětu se předpokládá absolvování předmětu Výpočetní technika.

Cílem předmětu

je seznámení se základními pojmy z oblasti algoritmizace. Po prostudování modulu by měl student být schopen zvládat tvořit základní algoritmy ve Visual Basicu for Application.

Pro koho je předmět určen

Modul je zařazen do bakalářského studia oboru Aplikovaná informatika a řízení studijního programu Strojírenství, ale může jej studovat i zájemce z kteréhokoliv jiného oboru, pokud splňuje požadované prerekvizity.

Skriptum se dělí na části, kapitoly, které odpovídají logickému dělení studované látky, ale nejsou stejně obsáhlé. Předpokládaná doba ke studiu kapitoly se může výrazně lišit, proto jsou velké kapitoly děleny dále na číslované podkapitoly a těm odpovídá níže popsaná struktura.

Při studiu každé kapitoly doporučujeme následující postup:



Čas ke studiu: xx hodin

Na úvod kapitoly je uveden **čas** potřebný k prostudování látky. Čas je orientační a může vám sloužit jako hrubé vodítko pro rozvržení studia celého předmětu či kapitoly. Někomu se čas může zdát příliš dlouhý, někomu naopak. Jsou studenti, kteří se s touto problematikou ještě nikdy nesetkali a naopak takoví, kteří již v tomto oboru mají bohaté zkušenosti.



Cíl: Po prostudování této kapitoly budete umět

- popsat ...
- definovat ...
- vyřešit ...

Ihned potom jsou uvedeny cíle, kterých máte dosáhnout po prostudování této kapitoly – konkrétní dovednosti, znalosti.



Výklad

Následuje vlastní výklad studované látky, zavedení nových pojmů, jejich vysvětlení, vše doprovázeno obrázky, tabulkami, řešenými příklady, odkazy na animace.



Shrnutí pojmů

Na závěr kapitoly jsou zopakovány hlavní pojmy, které si v ní máte osvojit. Pokud některému z nich ještě nerozumíte, vraťte se k nim ještě jednou.



Otázky

Pro ověření, že jste dobře a úplně látku kapitoly zvládli, máte k dispozici několik teoretických otázek.



Úlohy k řešení

Protože většina teoretických pojmů tohoto předmětu má bezprostřední význam a využití v databázové praxi, jsou Vám nakonec předkládány i praktické úlohy k řešení. V nich je hlavní význam předmětu a schopnost aplikovat čerstvě nabyté znalosti při řešení reálných situací hlavním cílem předmětu.



Klíč k řešení

Výsledky zadaných příkladů i teoretických otázek výše jsou uvedeny v závěru učebnice v Klíči k řešení. Používejte je až po vlastním vyřešení úloh, jen tak si samokontrolou ověříte, že jste obsah kapitoly skutečně úplně zvládli.

Úspěšné a příjemné studium s touto učebnicí Vám přeje autor výukového materiálu

David Fojtík

1 ÚVOD

Je až s podivem, jak rychle a silně nás obklopila výpočetní technika. Během několika málo desítek let se z velmi drahých a obrovských sálových počítačů, jež obsluhovalo několik desítek specialistů, stal poměrně malý a levný prostředek, který v různých podobách dennodenně používá téměř každý z nás.

1.1 Počátky moderní výpočetní techniky a software

Tento rozmach odstartovala v roce 1981 společnost IBM, kdy na trh uvedla otevřený systém osobního počítače IBM PC (personal computer). V té době byl tento počín mnohými odborníky brán přinejmenším jako velmi odvážný. Nyní je vše jinak, počítače jsou všude kolem nás a jsou zapojeny do všech moderních činností člověka. Mnohdy, především u specializovaných zařízení, dokonce existenci počítače ani nevnímáme, a dáváme jim jen jiná jména, např. mobil (smartmobil), řídicí jednotka, navigace, inteligentní spotřebič atd.

Na celé historii je ještě jedna a troufám si říci zajímavější skutečnost... Ačkoliv průkopníkem otevřených osobních počítačů byla společnost IBM, nejvíce na jejich odvaze vydělal společnost Microsoft. Dalo by se říct, že zásadní chybu IBM udělala v okamžiku, kdy z časových důvodů přestala vyvíjet vlastní operační systém a místo toho v roce 1980 podepsala smlouvu s Microsoftem, na dodávku nového operačního systému pro IBM PC. Smlouva zavazovala IBM zaplatit 60 dolarů z každého prodaného kusu, přičemž Microsoftu ponechala plná vlastnická práva.



Obr. 1. IBM PC

Vytvořený operační systém MS-DOS (MicroSoft Disk Operating System) přitom nebyl čistým dítkem Microsoftu. Ve skutečnosti Microsoft koupil a upravil QDOS (Quick and Dirty Operating System), který ale byl založen na systému CP/M (Control Program for Microcomputers). Zlí jazykové tvrdí, že Microsoft takto systém MS-DOS vybudoval na krádeži duševního vlastnictví Garyho Kildalla, který systém CP/M vytvořil.

Ač jsou počátky společnosti Microsoft jakékoliv, nelze zakladatelům Billu Gatesovi a Paulu Allenovi ubrat schopnost včas rozpoznat vysokou důležitost software. V té době totiž většina výrobců počítačů software podceňovala. Vlastně se není ani čemu divit, vždyť do té doby používané sálové počítače obsluhovali především programátoři, jejichž pracovní náplní bylo tvořit programy k řešení úkolů. Software se tak říkajíc dělal za pochodu podle potřeb řešených úkolů.

Osobní počítače však tuto situaci zcela změnily. Téměř přenoc se objevila nová komunita uživatelů (neprogramátorů), která chtěla počítače efektivně využívat. Software se stal důležitější než samotný hardware. Dnes každý výrobce hardware ví, že pokud má být prodej jeho výrobků úspěšný, musí pro něj existovat dostatečné množství kvalitního software.

Samotný software prodělal také bouřlivou evoluci. Velice zajímavý je vývoj ve vztahu k uživatelům. V počátcích byl software tvořen výhradně specialisty programátory, kteří řešili

téměř vše od realizace základních algoritmů, přes funkční logiku programu až k uživatelskému rozhraní. Vývoj software byl zdoluhavý, výsledný produkt byl často velice specializovaný a uživatelsky málo přívětivý. Uživatelé se museli každý software učit používat, tzv. uživatelská zkušenost v podstatě nebyla.

Později se komunita vývojářů rozdělila na programátory, jež vytvářeli nástroje a prostředky pro specialisty a tvůrce zakázkového software. Tím se vývoje nejenom zrychlil a zlevnil, ale především vznikly standardy v ovládání. Software se začal více přizpůsobovat uživatelům a nově se objevila možnost rozšiřovat funkcionalitu programů zkušenými uživateli.

Dnešní trend vývoje je v první řadě zaměřen na uživatele. Cílem je aby uživatel byl schopen co nejrychleji program v maximální míře využívat. Ergonomie, uživatelská zkušenost, design jsou stejně důležité mnohdy důležitější nežli samotná funkcionalita software. Také modularita programů je dnes neodmyslitelná. Především u profesionálního software uživatelé vyžadují možnost si prostředí přizpůsobovat, činnosti zautomatizovat nebo i samotnou funkcionalitu rozšiřovat. A tak se kruh opět uzavírá, pokud si chce uživatel funkcionalitu rozšířit či práci zautomatizovat musí mít alespoň základní dovednosti z programování.

1.2 Programovací jazyk

Existuje celá řada profesionálního software, jehož hlavní devizou je schopnost přizpůsobení se konkrétním požadavkům. Typickým představitelem je tabulkový kalkulátor, u kterého je širší možných uplatnění až ohromující. Nezřídka je takovýto software vybaven nástroji, jež umožňují uživateli specifikovat a následně vyvolávat různé operace. Ačkoliv se tvůrci intenzivně snaží najít intuitivní a snadné řešení jak dané uživatelské operace zaznamenat, schopnosti programového jazyka se jen těžko nahrazují. Pokud se jiné plnohodnotné řešení nalezne, je obvykle stejně náročné jako samotné programovací jazyky. Navíc programovací jazyk je svou povahou univerzální a jeho znalost se dá ve větší šíři uplatnit.

Z těchto důvodů je znalost programovacího jazyka velmi přínosná. Otázkou je, s jakým programovacím jazykem by měl začátečník začít? Odpověď se jeví být složitá, neboť programovacích jazyků je nepřeberné množství od univerzálních podporovaných prakticky na všech platformách jako je jazyk C, ke specializovaným vytvořeným pro konkrétní aplikaci. Zaměříme-li se na nejrozšířenější platformu MS Windows, zjistíme, že zde jako rozšiřující jazyk aplikací jednoznačně kraluje Visual Basic for Application (VBA). Výhodou tohoto

jazyka je jeho jednoduchost, podpora a dostupnost. Najdeme jej například u aplikací: MS Office, MS MapPoint, MS Visio, AutoCAD, Inventor, WordPerfect, ArcGis, CorelDRAW atd. VBA je součástí jazyků rodiny MS Visual Basic, ze kterého vychází jazyky Visual Basic for scripting (VBS) a Visual basic .NET. Jinými slovy, znalosti VBA je možné uplatnit v širokém záběru možností. Další výhodou a zároveň největší nevýhodou je, že tento jazyk pochází z dílny Microsoftu, takže je zde vysoká podpora ze strany silného výrobce a naopak nechutí jeho nasazení na konkurenčních platformách, přesto se tak částečně děje.

2 AUTOMATIZACE PRÁCE V MS OFFICE ZÁZNAMEM MAKER



Čas ke studiu: 2 hodiny



Cíl Po prostudování této kapitoly budete umět

- definovat pojem makro
- popsat způsob spouštění maker v prostředí MS Excelu
- zaznamenávat makra



Výklad

2.1 Makra

Makra jsou uživatelsky programovatelné nástroje, umožňují automatizovat uživatelskou činnost především v prostředí aplikací MS Office. Nejčastěji jsou uchovávána v dokumentech aplikací (například v souborech doc, docm, xls, xlsx), mohou být také uloženy ve speciálních souborech.

Makra jsou v podstatě subrutiny jazyka Visual Basic for Application (VBA) rozšířeného o funkcionalitu (objektový model) zpřístupňující typické vlastnosti dané aplikace. Pro každou aplikaci je jiná rozšiřující specializovaná knihovna.

Pokud se zaměříme na prostředí MS Excel, můžeme makra vytvářet třemi základními metodami:

1. záznamem činností,
2. programovým zápisem,
3. kombinací záznamu a programového zápisu.

2.2 Záznam maker v prostředí MS Excel 2010

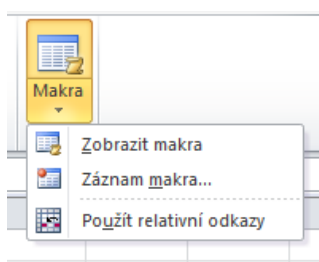
Záznam makra je postavený na principu automatického překladu uživatelem prováděných operací do programového kódu jazyka VBA. Tento převod funguje naprosto automaticky:

1. uživatel zahájí záznam,
2. následně běžným způsobem postupně provede všechny operace,
3. a ukončí záznam.

Zahájení záznamu

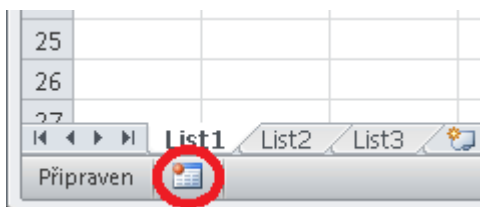
Záznam je možné zahájit třemi způsoby:

1. Záložka „Zobrazení“, skupina „Makro“, rozbalením tlačítka „Makra“, nabídka „Záznam makra...“.



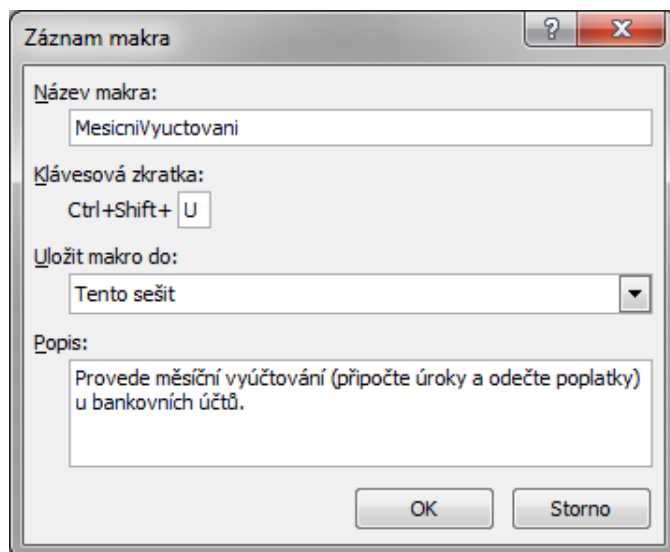
Obr. 2. Volba zahájení záznamu makra na záložce zobrazení

2. Záložka „Vývojář“, skupina „Kód“, tlačítko „Záznam makra...“ (více v kapitole 2.3).
3. Kliknutím na tlačítko „Záznam makra“, které se nachází na stavovém řádku aplikace.



Obr. 3. Zahájení záznamu tlačítkem na stavovém řádku

Ve všech uvedených případech se automaticky zobrazí dialog „Záznam makra“, ve kterém se vyplní potřebné údaje.



Obr. 4. Dialog „Záznam makra“

V první řadě se povinně uvede „Název makra“, pro které platí stejná pravidla jako pro identifikátor jazyka Visual Basic:

- Název makra (identifikátoru) nesmí obsahovat mezery,
- nesmí začínat číslicí,
- a nesmí být shodné s klíčovými slovy jazyka VBA.

Navíc se nedoporučuje česká diakritika, která zpravidla způsobuje problémy při přenosu makra na systém, který nemá podporu českého jazyka (častý problém mezinárodních společností).

Dále se může uvést klávesová zkratka, která v prostředí MS Excelu je vždy v kombinaci s klávesou Control. Stačí doplnit písmeno nebo stisknou kombinaci Shift+Písmeno.

Další povinným údajem je „Uložit makro do“, kde je na výběr jedna ze tří možností:

1. Tento sešit – makro bude uloženo v aktivním sešitu.
2. Osobní sešit maker – makro bude uloženo do skrytého sešitu personal.xlsm (xls).
3. Nový sešit – založí se nový sešit, kde bude makro uloženo.

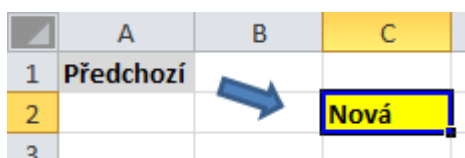
Posledním údajem je položka „Popis“, kde se může uvést libovolná poznámka.

Klepnutím na tlačítko OK se zahájí záznam.

Zaznamenávání uživatelských operací

Od této chvíle se veškeré uživatelské operace, které modifikují dokument, nebo provádějí změnu výběru objektů (buňka, soubor, list, graf atd.), nebo provádějí změny stavu dokumentu (uložení, tisk apod.) se zaznamenáním v jazyce VBA. Na druhou stranu se zaznamenávají způsoby, jak se k dané operaci došlo. Je tedy nepodstatné, zdali se daná operace provedla klepnutím myši do nabídky, nebo klávesovou zkratkou či jinak. Dále se nezaznamenává pohyb myši, změna umístění či velikosti aplikace, procházení nabídky a veškeré operace mimo aplikaci.

V MS Excelu je během záznamu navíc možné kdykoliv provést přepnutí volby „Použít relativní adresy“, která má vliv na způsobu záznamu výběru buňky či oblasti buněk. Je-li volba vypnuta, makro si zapamatuje přesnou adresu vybrané buňky (oblasti). Je-li však zapnuta, makro si zapamatuje výběr buňky relativně vůči předchozí. Například na obrázku 5 je zobrazen výběr buňky C4 z předchozího výběru A1. Bude-li volba vypnuta, makro si zapamatuje výběr buňky C4 (konkrétně si zaznamená `Range("C4").Select`). Při zapnuté volbě si makro zapamatuje výběr buňky jako přesun o jeden řádek níže a dva sloupce doprava (konkrétně si zaznamená `Selection.Offset(1,2).Select`).

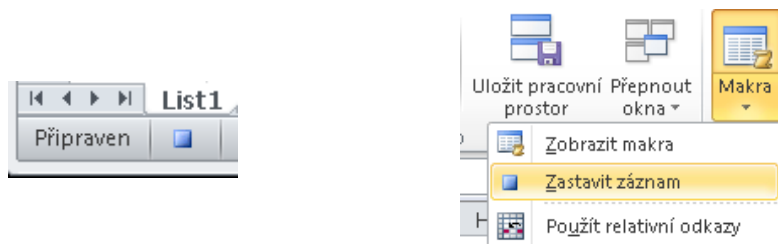


Obr. 5. Výběr buňky během záznamu

Makro, jež má být aplikovatelné na uživatelem vybranou buňku, nesmí obsahovat operaci výběru buňky. Jinými slovy buňka se musí vybrat před záznamem.

Ukončení záznamu makra

Ukončení záznamu stejně jako jeho zahájení se dá provést třemi způsoby: tlačítkem na stavovém řádku, položkou „Zastavit záznam“ v nabídce Makra nebo na kartě Vývojář.

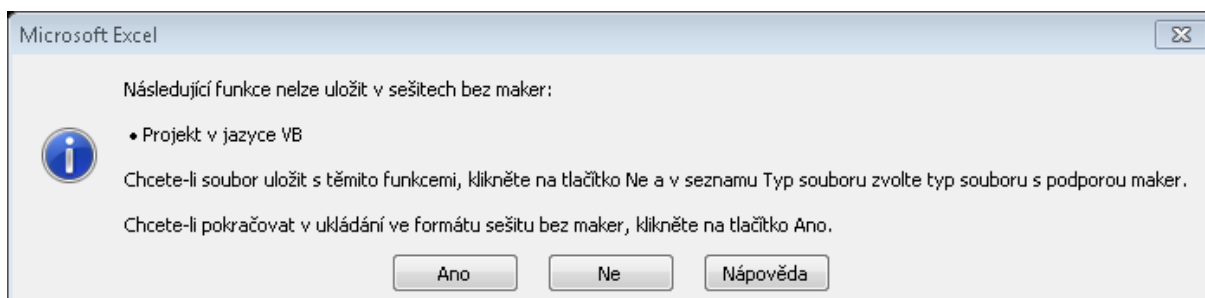


Obr. 6. Ukončení záznamu

2.3 Používání maker v prostředí MS Excel 2010

Makro může být provedeno, pouze když je otevřené jeho úložiště (soubor, doplněk, databáze, šablona apod.). V MS Excelu může být makro uloženo v souboru, šabloně či doplňku. Soubory se liší příponou v závislosti na verzi aplikace. Zatímco soubory do verze office 2003 měly příponu `xls`, od verze 2007 má soubor bez podpory maker příponu `xlsx` a s podporou maker `xlsm`. Důvodem je lepší rozpoznatelnost souboru s makry již na úrovni souborového manažeru.

Je velmi důležité si uvědomit závislost přípon na podpoře uchování maker. Při špatné volbě přípony se všechny makra v souboru nenávratně ztratí. Navíc se implicitně nové soubory ukládají s příponou `xlsx` bez podpory maker. Aplikace sice na tento rozpor upozorní, ale i tak implicitně volí soubor bez podpory maker. Uživatel, který na upozornění (viz obr. 7) zareaguje pouhým potvrzením dialogu, o makra přijde.

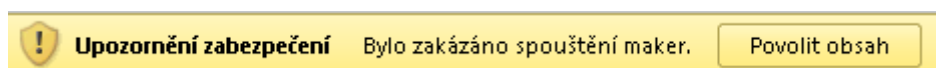


Obr. 7. Upozornění na ukládání maker do souboru nepodporující makra

Obdobné je to s příponami ostatních souborů. Od verze MS Office 2007 mají všechny soubory s makry poslední písmeno přípony „m“ kdežto implicitní zakončení je „x“. Například šablony v MS Excelu ve verzi 97-2003 měly příponu `xlt`, od verze 2007-2010 mohou mít příponu `xltx` bez podpory maker a `xltxm` s podporou maker.

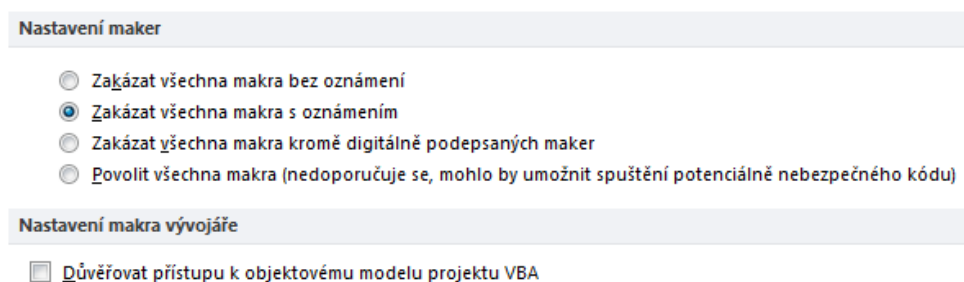
Zabezpečení maker

Makra mohou vykonávat potenciálně nebezpečné činnosti. Proti tomu jsou aplikace vybaveny celou řadou kontrolních vlastností, které chrání před možnou aktivací nebezpečného kódu. Úroveň ochrany je závislá na verzi aplikace (novější verze mají vyšší formu zabezpečení) a na aktuálním nastavení (viz obr. 9). Uživatel může ochranu zcela vypnout, nebo naopak může zakázat všechna makra. Standardně aplikace MS Office 2010 reaguje tak, že se při prvním otevření dokumentu makra zablokují s upozorněním, které umožňuje uživateli makra povolit (viz obr. 8). Povolí-li uživatel makra, pak aplikace soubor označí jako bezpečný a při dalším otevření se již automaticky makra povolí.



Obr. 8. Upozornění MS Excelu 2010 o zablokování maker při otevření souboru

Změna chování je možná na kartě vývojář, položkou „Zabezpečení maker“.

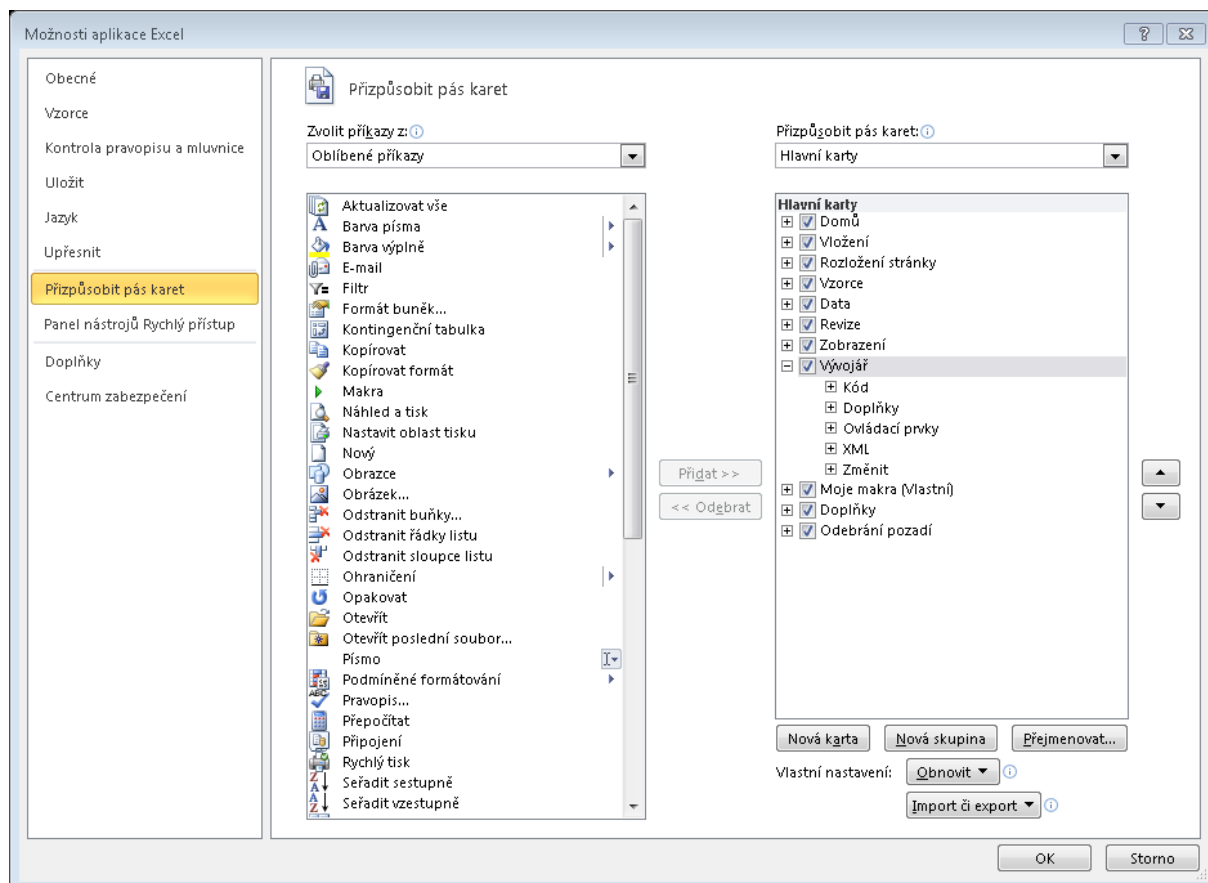


Obr. 9. Základní možnosti úrovně zabezpečení maker

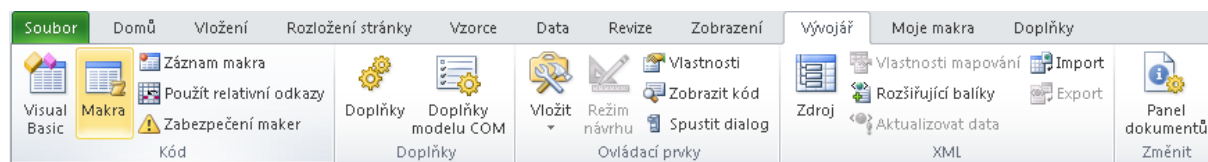
Více informací o zabezpečení maker je k dispozici v nápovědě aplikace.

Zobrazení karty vývojář

Již několikrát byla zmíněna záložka vývojář. Tato záložka je standardně skrytá. V MS Excelu 2010 se aktivuje přes záložku „Soubor“ volbou „Možnosti“. V dialog „Možnosti aplikace Excel“ se vybere položka „přizpůsobit pás karet“ kde se v pravé části zatrhne položka „Vývojář“ (viz obr. 10).



Obr. 10. Zobrazení karty vývojář



Obr. 11. Karta vývojář

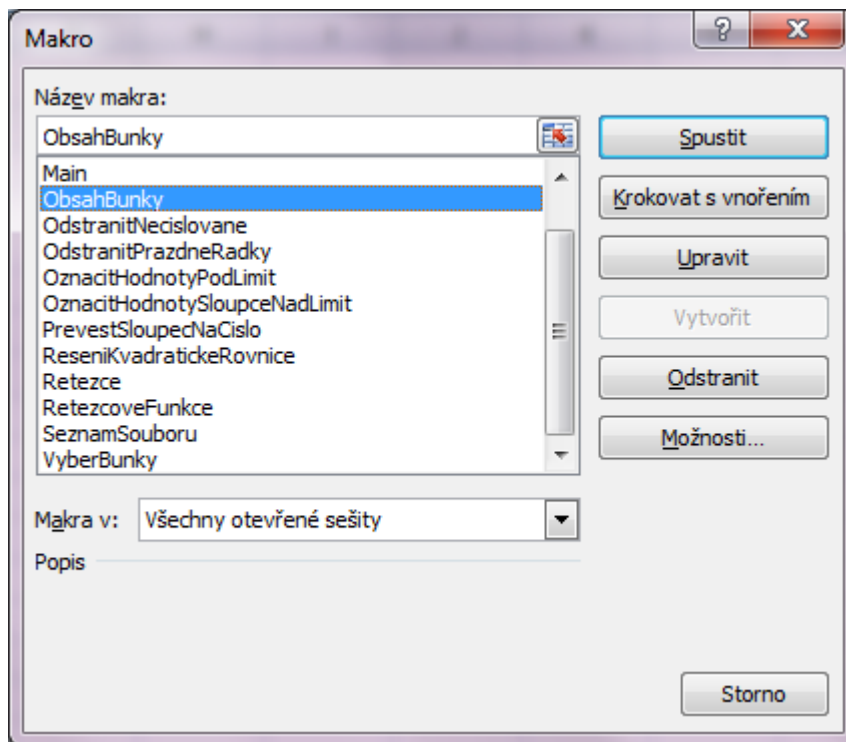
Spouštění maker

Makra se mohou spouštět několika způsoby:

1. prostřednictvím dialogu „Makra“,
2. klávesovou zkratkou (je-li k makru přiřazeno),
3. přes nabídku na panelu Rychlého přístupu,
4. přes nabídku na pásu karet,

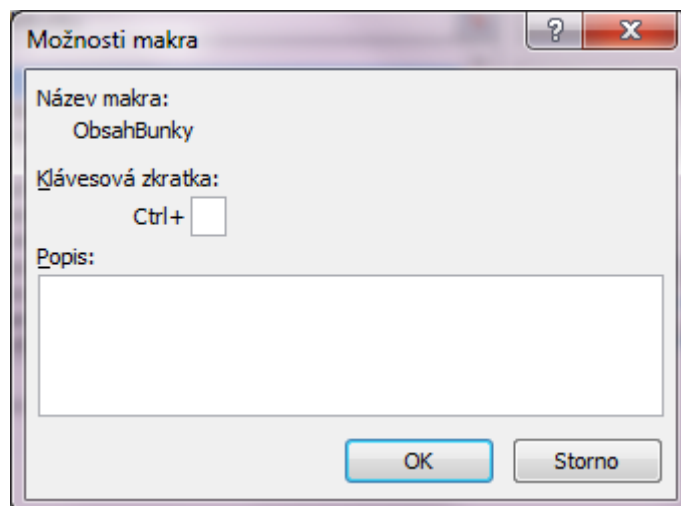
5. spouštěcím tlačítkem v dokumentu.

Nejjednodušší způsob, který je navíc vždy k dispozici, je přes dialog Makra. Tento dialog se zobrazí po stisku tlačítka „Makra“ na záložce zobrazení nebo klávesou **Alt+F8**.



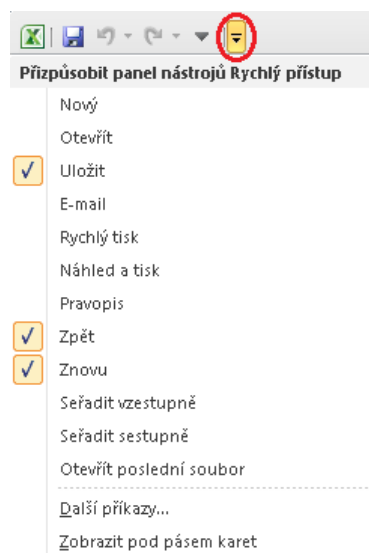
Obr. 12. Dialog makra

V dialogu se vybere požadované makro a tlačítkem „Spustit“ se vykoná. Přes tento dialog je také možné dodatečně přiřadit klávesovou zkratu. Opět se nejprve vybere makro a pak tlačítkem „Možnosti“ se zobrazí dialog „Možnosti makra“, kde se klávesová zkratka uvede.



Obr. 13. Dialog Možnosti makra

Makro, které má přiřazenou klávesovou zkratku, je možné ihned spouštět. Makra, která se často využívají v různých dokumentech, je dobré spouštět přes panel Rychlého přístupu, nebo přes zástupce v pásu karet. Zástupce do panelu rychlého přístupu se nejrychleji přidá rozbalovacím tlačítkem (viz obr. 14). Zobrazí se nabídka, ve které se zvolí položka „Další příkazy“.



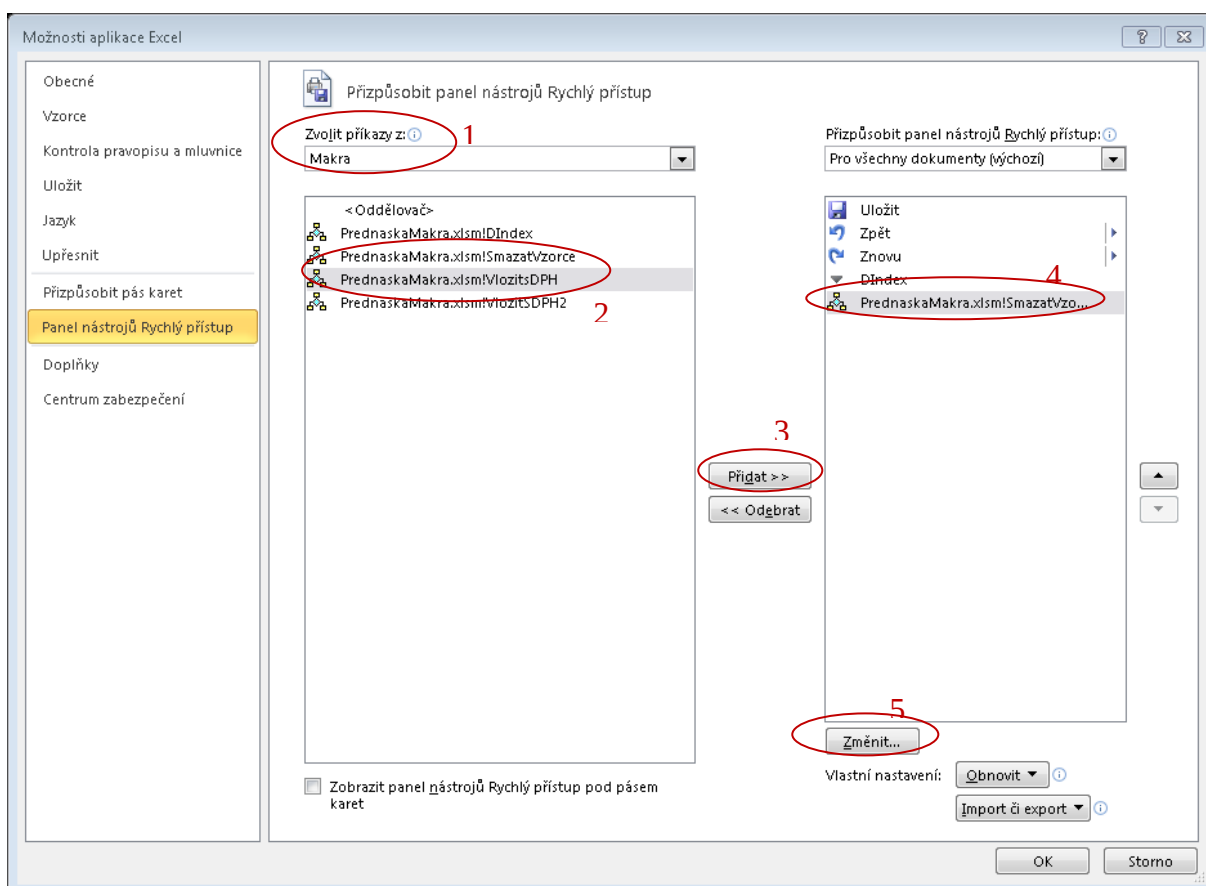
Obr. 14. Rychlá volba přizpůsobení panelu Rychlý přístup

Zobrazí se dialog „Možnosti aplikace Excel“ s aktivní položkou „Panel nástrojů Rychlý přístup“. Přidání zástupce makra se provede následujícím postupem:

1. v seznamu „Zvolit příkazy z“ se vybere položka Makra,

2. označí se požadované makro,
3. tlačítkem přidat se vloží zástupce do panelu,
4. pro úpravu se zástupce označí,
5. tlačítkem „Změnit“ se provede úprava názvu a změna ikony zástupce.

Po zavření dialogu se zobrazí ikona v panelu rychlého přístupu, přes kterou se makro bude spouštět.



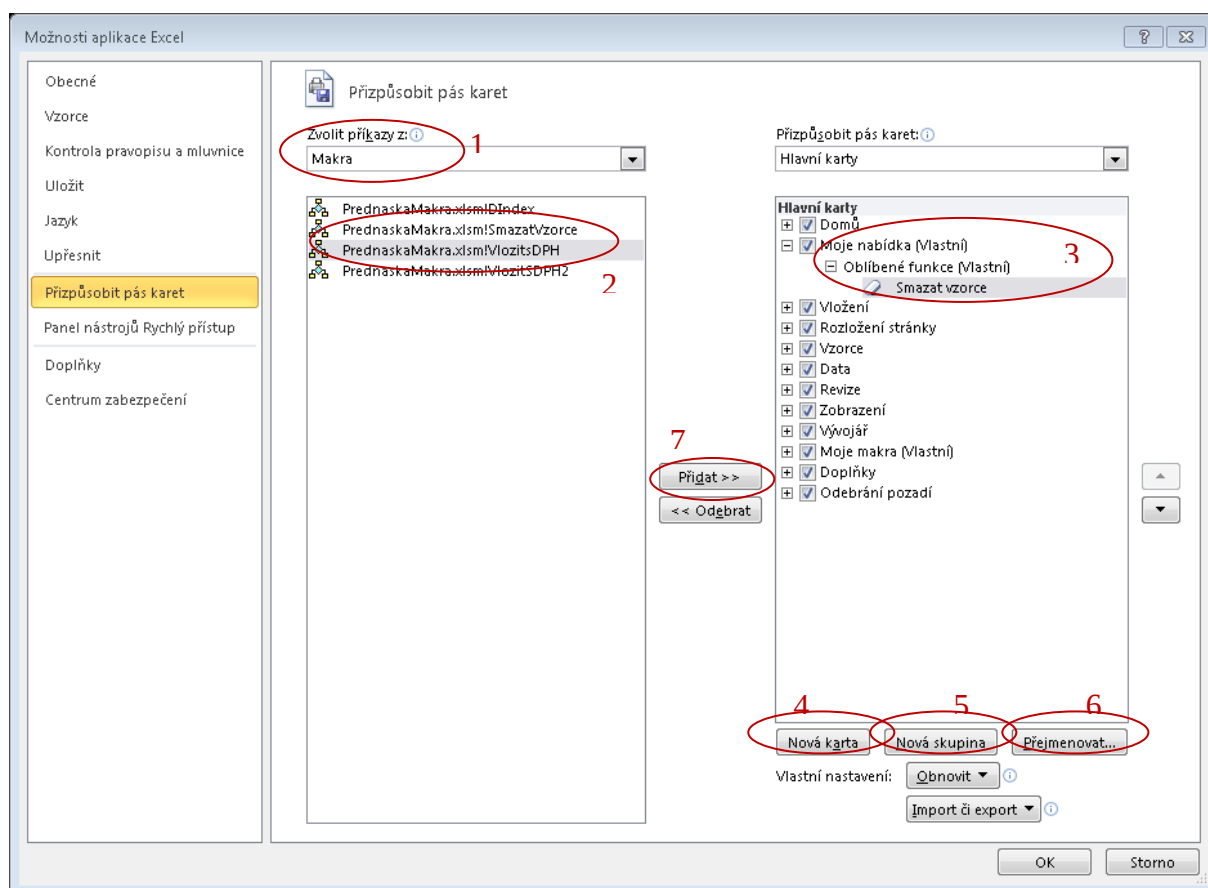
Obr. 15. Přidání zástupce do panelu Rychlý přístup

Další možností (od verze MS Office 2010) je přidat zástupce do pásu karet. Za tímto účelem se nejprve musí aktivovat dialog „Možnosti aplikace Excel“ nejlépe přes záložku „Soubor“ volbou „Možnosti“. V dialogu se vybere položka „přizpůsobit pás karet“.

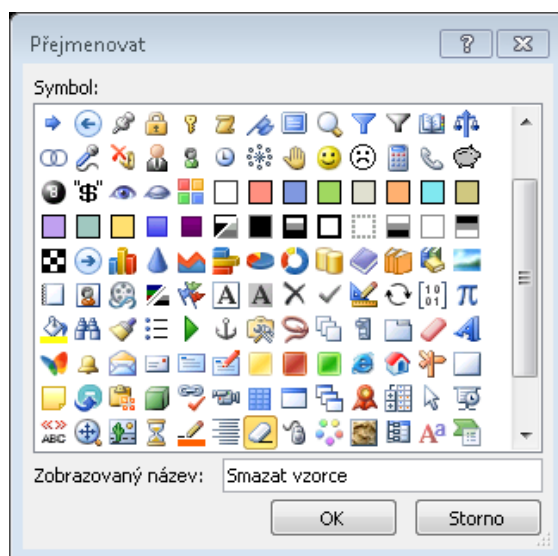
Zde se následujícím postupem může přidat nová záložka a do ní nová skupina se zástupcem makra:

1. v seznamu „Zvolit příkazy z“ se vybere položka Makra,
2. označí se požadované makro,
3. vybere se místo pro novou kartu,
4. tlačítkem „+“ se přidá nová karta,
5. tlačítkem „Nová skupina“ se do karty přidá nová skupina,
6. po vybrání nové karty a skupiny se tlačítkem „Přejmenovat“ záložka a skupina pojmenuje podle požadavků,
7. tlačítkem „Přidat“ se pak přidá zástupce makra.

Dále se již pokračuje obdobně jako u přidání zástupce do panelu Rychlého přístupu.



Obr. 16. Přidání vlastní karty a zástupce makra do pásu karet



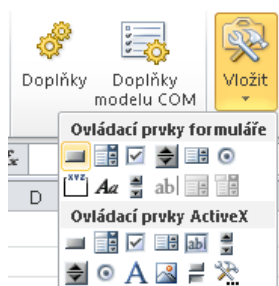
Obr. 17. Přejmenování zástupce

Poslední možností jak spouštět makro je přes tlačítko vložené přímo do dokumentu. Tato volba se používá v případě maker, která mají úzkou vazbu s daným dokumentem, to znamená, že se mimo daný dokument nepoužívá.

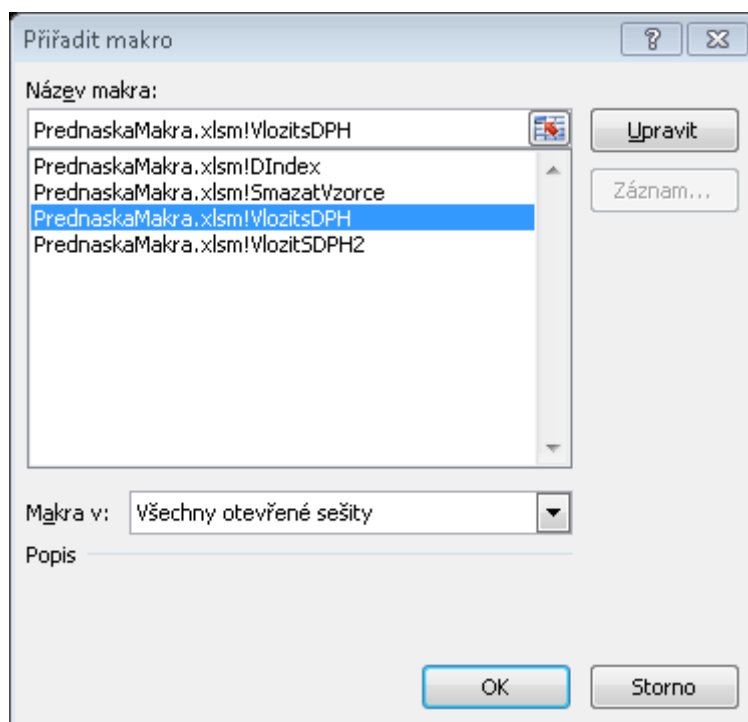
Tlačítko se do dokumentu vloží prostřednictvím nabídky „Vložit“ na kartě Vývojář. Po rozbalení se nabídnou dvě skupiny ovládacích prvků (obr. 18):

1. Ovládací prvky formulářů,
2. Ovládací prvky ActiveX.

Jednodušeji se používají prvky z první skupiny. Stačí zvolit zástupce tlačítka a v dokumentu na požadovaném místě nakreslit obdélník. Ihned na to se zobrazí dialog „Přiřadit makro“, kde se vybere požadované makro (obr. 18).



Obr. 18. Vložení ovládacích prvků



Obr. 19. Přiřazení makra k tlačítku

Tlačítko se aktivuje automaticky po výběru libovolného objektu. Je-li potřeba tlačítko přemístit nebo jinak upravit, stačí na tlačítko klepnout pravým tlačítkem myši.

2.4 Omezení záznamu maker

Vytváření makra záznamem má však značná omezení. Prakticky je možné čistým záznam vytvořit makra pouze v 10% případech. Ve zbylých případech je záznam aplikovatelný pouze částečně. Důvodem jsou následující omezení:

- Záznam makra nelze zahájit v režimu editace buňky (totéž platí pro spuštění makra).
- Je-li zaznamenána editace buňky, vždy se uchová kompletní stav buňky. Nástroj nerozpozná změny, ale jen výsledný stav včetně obsahu buňky.
- Záznamem nelze uskutečnit logické rozhodování.
- Také nelze zaznamenat opakování činností. Záznam opakované operace je možné jedině opakováním záznamu.
- Nelze zaznamenat práci na více tabulkách. Vždy se zaznamená konkrétně vybraný list, sešit. Vzniká tak problém s vykonáním makra nad jinými dokumenty.

Každá aplikace, umožňující záznam má podobná omezení.



Shrnutí pojmů

Název makra je jednoznačný identifikátor makra, pro který platí, že nesmí začínat číslicí, nesmí obsahovat mezery a nesmí se shodovat s klíčovými slovy jazyka Visual Basic.

Použití relativní adresy je volba při záznamu makra, která zajišťuje, že výběr buňky během záznamu si makro zapamatuje relativně vůči předchozímu výběru.

Zabezpečení maker je ochranná vlastnost aplikací proti spuštění nežádoucího makra s nebezpečným kódem. Úroveň ochrany je závislá na verzi aplikace a na aktuálním nastavení.



Otázky

1. Jaká pravidla platí pro tvorbu názvu makra?
2. Jaký význam má volba „Použití relativní odkazy“ při záznamu maker?
3. Je možné zaznamenat editaci části buňky?
4. Kde se uchovávají makra?
5. Jaký je rozdíl mezi soubory s příponou XSLX a XSLM?
6. K čemu slouží zabezpečení maker a jak funguje?
7. Jaké jsou způsoby spuštění maker?
8. Jak zobrazit kartu „Vývojář“?



Úlohy k řešení

Pro prostředí MS Excel si stáhněte dokument VB01.xlsx a metodou záznamu v něm vytvořte dvě makra:

9. Makro SmazatVzorce, jež nad vybranou tabulkou smaže všechny vzorce a zachová pouze hodnoty. Makro se bude vykonávat po stisknutí kombinace kláves CTRL+SHIFT+S. Makro si ověřte nad tabulkou zboží listu „Mazání vzorců“.
10. Makro MesicniBankovniOperace, jež bude nad tabulkou listu „Bankovní účty“ provádět bankovní operaci odpočtu měsíčních poplatků a současného navýšení o bankovní úrok. Hodnota ročního úroku a měsíčního poplatků jsou uvedeny v buňce B1 a B2. Pro výpočet násobícího koeficientu měsíčního úroku z roční sazby použijte vzorec $\sqrt[12]{(1 + \text{RočníÚrok})}$), který se v MS Excelu realizuje zápisem =(1+B1)^(1/12). Makro se bude vykonávat pomocí tlačítka (ovládacího prvku formuláře) „Bankovní operace“ který vložíte do téhož listu na místě překrytí buněk C1:C2.

3 EDITACE MAKER PROGRAMOVÝM ZÁPISEM



Čas ke studiu: 2 hodiny



Cíl Po prostudování této kapitoly budete umět

- Otevírat zdroj maker
- Editovat a vytvářet jednoduchá makra programovým zápisem
- Orientovat se v prostředí Visual Basic



Výklad

Metoda záznamu makra má tedy své limity. Přes tyto limity je možné se dostat odborným zásahem do textového zápisu makra. V tomto případě je možné prostřednictvím editoru, který je součástí aplikací makra upravovat. Je však nutná znalost jazyka ve Visual Basicu for Application a objektové knihovny aplikace. Obdobnou metodou je možné makro od začátku vytvořit, čemuž zkušení programátoři dávají přednost.

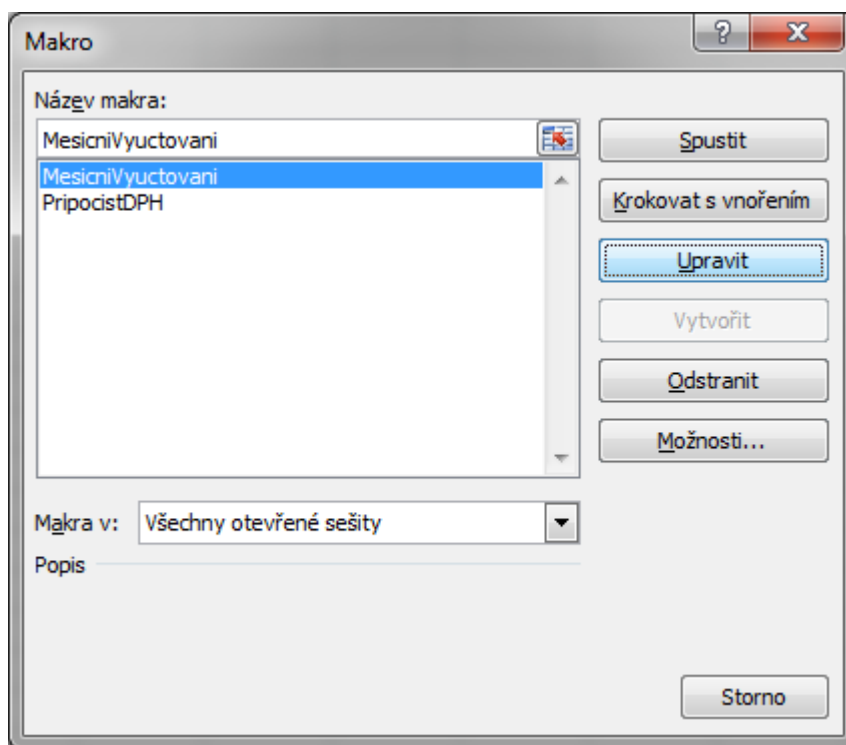
Zajímavou možností je tvořit programové makro, které ovládá jinou aplikaci. Například makro v MS Excelu může spustit MS Word, vytvořit dokument a odeslat jej e-mailem.

Další možností je vytvářet automatizované činnosti v některém profesionálním nástroji mimo prostředí aplikace. Vznikají tak profesionální doplňky, které je instalují stejně jako jiné aplikace avšak místo svého zástupce v nabídce Start si vytvoří své rozhraní přímo v aplikaci. Tento způsob je však určen pro profesionály a nebudeme se jím zabývat.

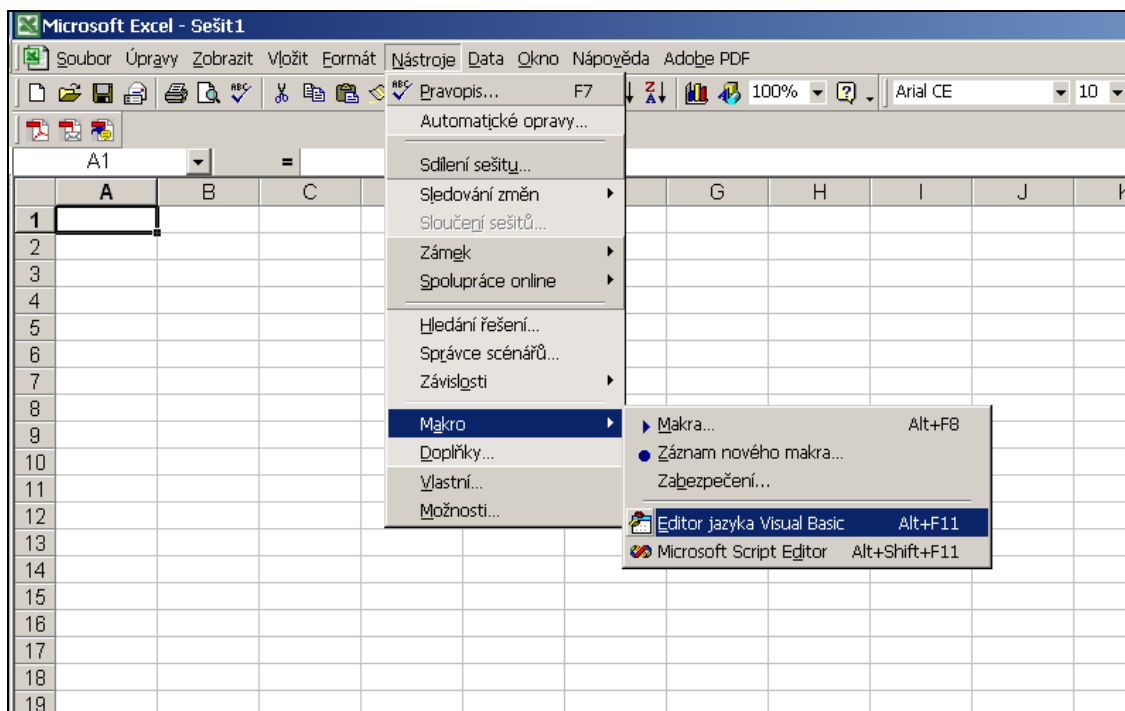
3.1 Editor jazyka Visual Basic for Application

Pro editaci programového kódu makra v prostředí MS Office, se kterým tyto studijní opory výhradně pracují, potřebujeme spustit příslušný editor. Editor je možné spustit přes dialog makra (Alt+F8) tlačítkem „Upravit“ (viz obr. 19). Také je možné použít klávesovou zkratku Alt+F11.

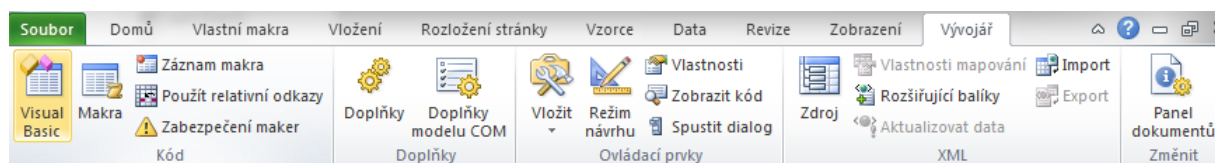
Další možnosti se již liší v závislosti na verzi MS Office. Například pro verze MS Excelu 7 – 2003 je možné editor spustit přes nabídku „Nástroje“ položku „Makra“ a „Editor jazyka Visual Basic“. V MS Office 2007 - 2010 se editor spustí příkazem „Visual Basic“ na kartě „Vývojář“ (viz obr. 20 - 21).



Obr. 20. Zahájení editace makra



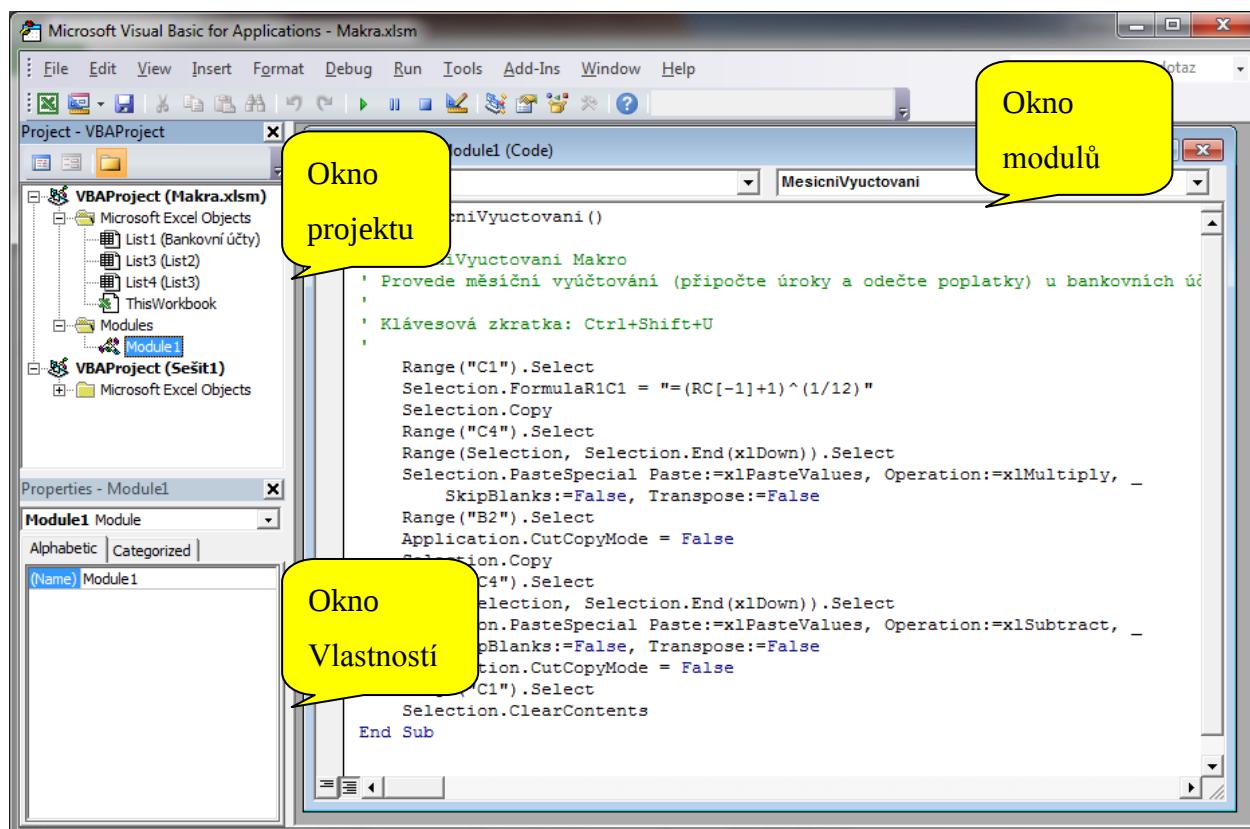
Obr. 21. Otevření editoru v prostředí MS Excel 2003



Obr. 22. Otevření editoru v prostředí MS Excel 2010

Prostředí Editoru jazyka Visual Basic

Editor jazyka Visual Basic je standardně rozdělen na několik částí. Vlevo se dokují okna Projektu a Vlastností. Největší část pak zabírají plovoucí okna s programovým kódem.



Obr. 23. Editor jazyka Visual Basic

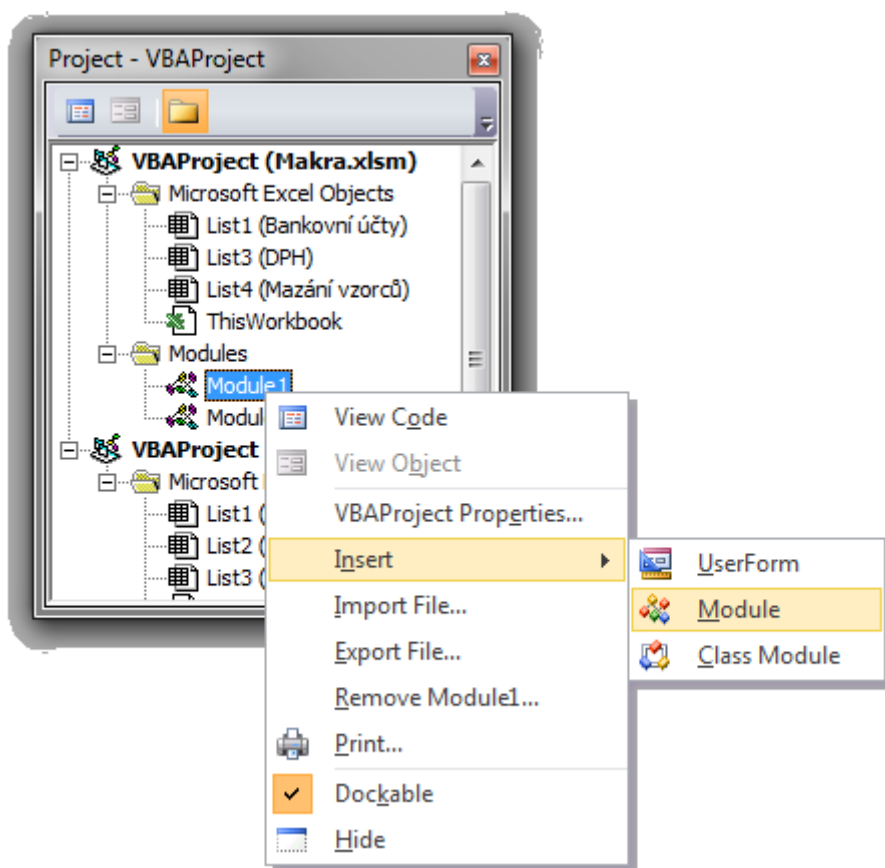
Okno projekt (project)

Okno obsahuje objektový pohled na otevřené soubory (v MS Excelu Sešity). Každý soubor je chápán jako projekt jazyka Visual Basic (VBAProject). Název souboru je uveden v závorkách. Pod každým projektem jsou pak rozbalovací složky:

- Skupina „Microsoft Excel Objects“, která obsahuje seznam listů a zástupce samotného dokumentu „ThisWorkbook“.
- Skupina „Modules“, která obsahuje seznam Modulů, což jsou dílčí úložiště maker. Tato skupina se zobrazí pouze, je-li v souboru alespoň jeden modul.
- Skupina Forms – jenž obsahuje všechny uživatelské dialogy (soubor na obrázku nemá žádný dialog, takže tato skupina chybí).

- Skupina Class Modules – jenž obsahuje všechny uživatelské třídy objektů (také na obrázku není k dispozici).

Kromě skupiny „Microsoft Excel Objects“, která je závislá na prostředí MS Excelu můžeme do všech ostatních skupin přidávat dílčí položky přes kontextové menu vyvolané pravým tlačítkem myši volbou „Insert“. Zobrazí se nabídka všech dostupných objektů, které je možné přidat.

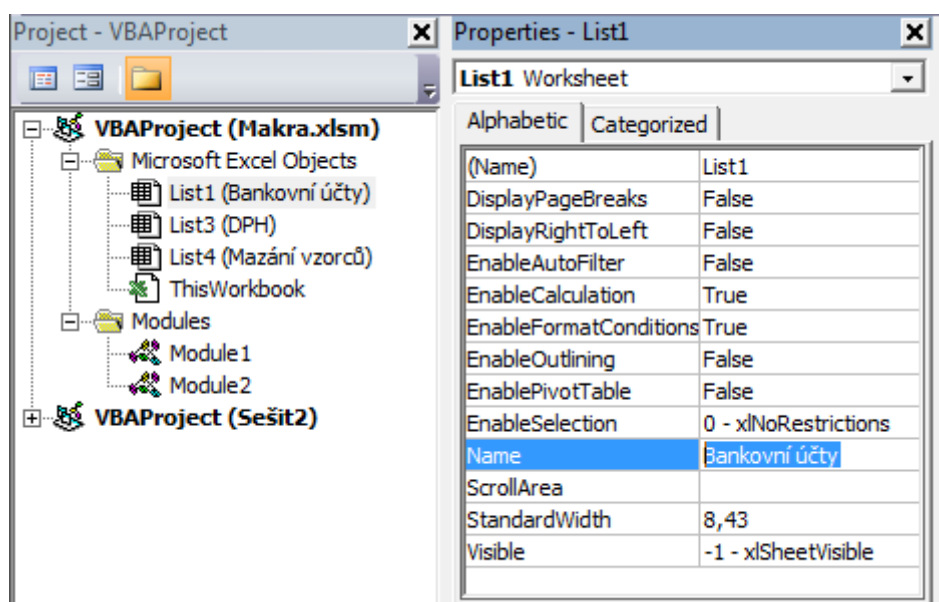


Obr. 24. Okno projektu a vkládání nového objektu do projektu

Obdobně se objekty ruší. Konkrétně se nad vybraným objektem pravým tlačítkem myši vyvolá kontextové menu, kde se zvolí položka Remove s názvem daného objektu.

Okno vlastností (Properties)

Okno vlastností mění svůj obsah v závislosti na výběru objektu v Okně projektu (viz Obr. 25). Okno zobrazuje vlastnosti vybraného objektu a je-li to možné, také nabízí jejich editaci. Seznam vlastností může být seřazen abecedně (záložka Alphabetic) nebo podle kategorií (záložka Categorized). Nejčastěji se zde upravuje Vlastnost „(Name)“, která představuje interní název daného objektu, který se používá výhradně v programovém kódu. U listů je navíc ještě jedna vlastnost „Name“ (pozor není v závorkách), která představuje jméno záložky listu, tak jak ji vidí uživatel. Ostatní vlastnosti jsou závislé na typu vybraného objektu.



Obr. 25. Provázanost okna Vlastností s oknem Projektů

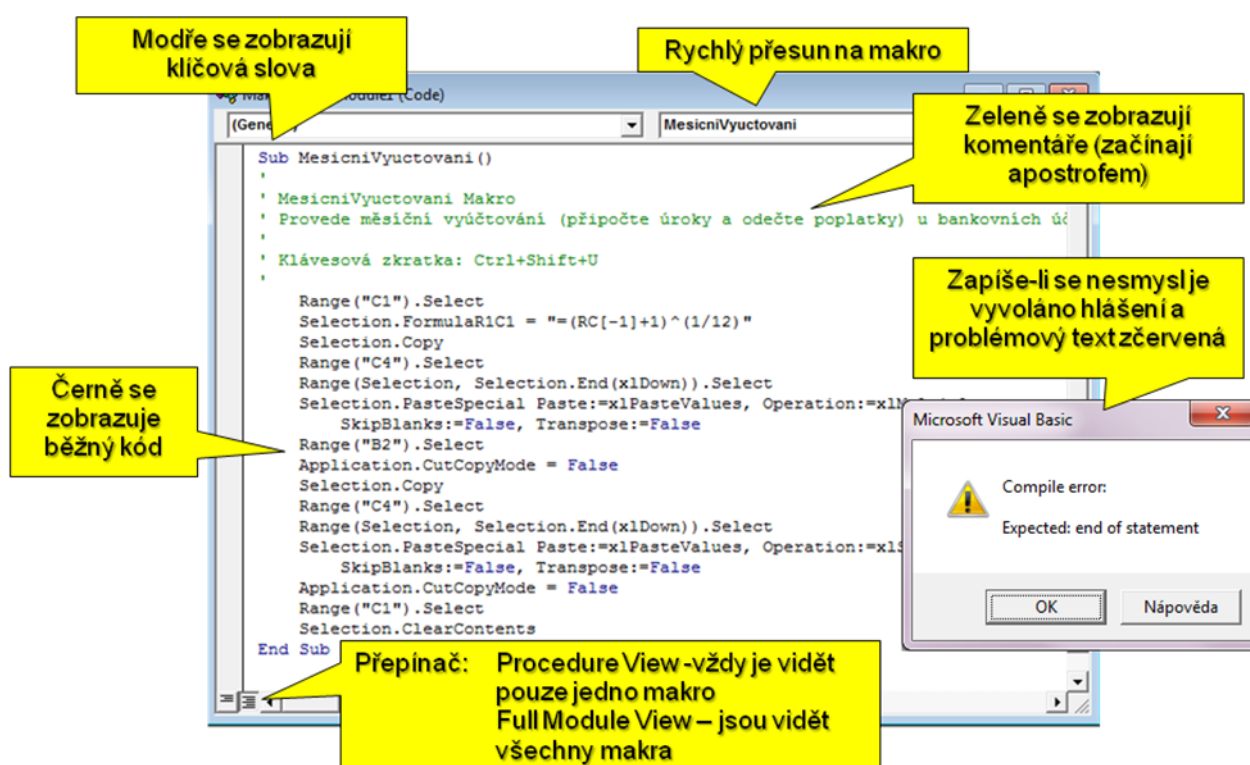
Okno modulu programového kódu (Code)

Poklepnutím nad objektem v okně Projekt se zobrazí Okno kódu obsahující textový zápis algoritmu (makra, funkce). V editoru můžeme svobodně psát či editovat programový kód, přičemž průběžně editor na pozadí analyzuje zapisovaný text a provádí drobné zásahy do textu.

V první řadě kontroluje syntaxi. Pokud vyhodnotí chybu, oznámí ji uživateli formou varovného dialogu (viz Obr. 26) přičemž daný text změní svou barvu na červenou.

Změnu barvy také provádí u bezchybného kódu v závislosti na tom, zda rozpozná klíčové slovo nebo komentář. Komentáře automaticky převádí do zelené barvy a klíčová slova do modré.

Další zajímavou úpravu textu editor provádí u klíčových slov a identifikátorů. Jakmile rozpozná známý identifikátor nebo klíčové slovo upraví jim velikost písmen podle deklarace. Například uvedeme-li klíčové slovo `sub` s malým `s` na začátku, editor automaticky text opraví a písmeno `s` převede na velký znak.



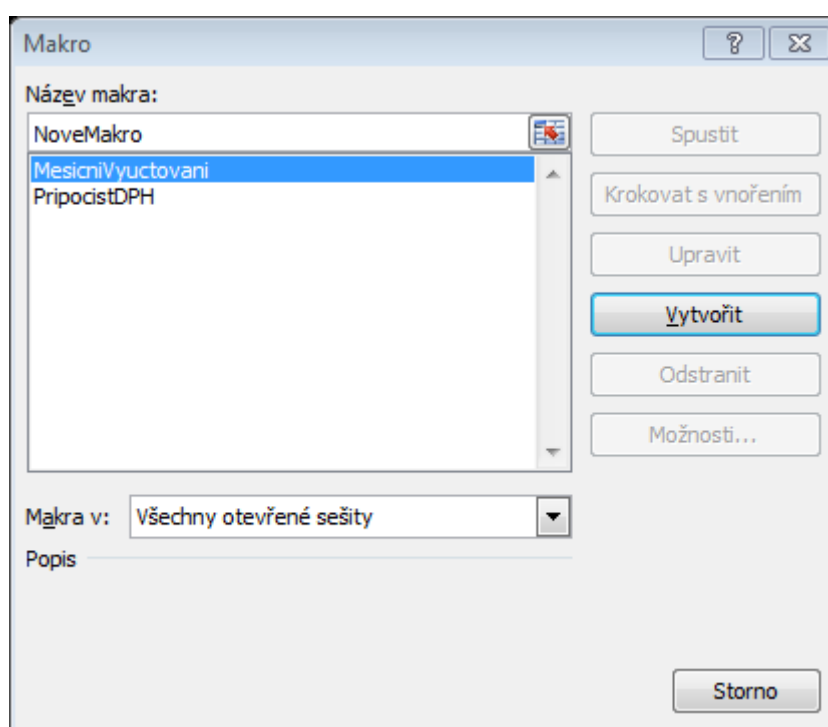
Obr. 26. Okno modulu programového kódu

3.2 Programové založení makra

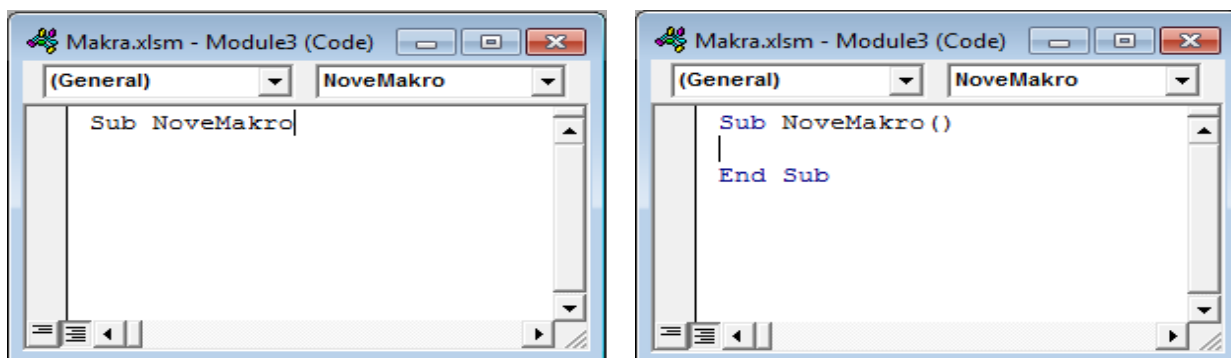
Makro můžeme založit několika způsoby. Přímou z prostředí aplikace dialogem Makro (Alt+F8), kdy se do položky „Název makra“ uvede identifikátor makra a stiskem tlačítka

„Vytvořit“ se makro založí (viz Obr. 27). Zkušenější programátoři však raději volí přímý zápis nového makra.

K tomu nejdříve potřebujeme otevřít Editor jazyka Visual Basic a vybrat nebo založit modul. Pak se do modulu mimo tělo jiného makra uvede „`sub NazevMakra`“, kde `NazevMakra` je libovolný dosud nepoužitý identifikátor nového makra. Po stisku klávesy Enter se makro založí. Editor automaticky za názvem doplní závorky a ob jeden řádek dopíše značku konce makra „`End Sub`“.



Obr. 27. Založení nového makra přes dialog Makro



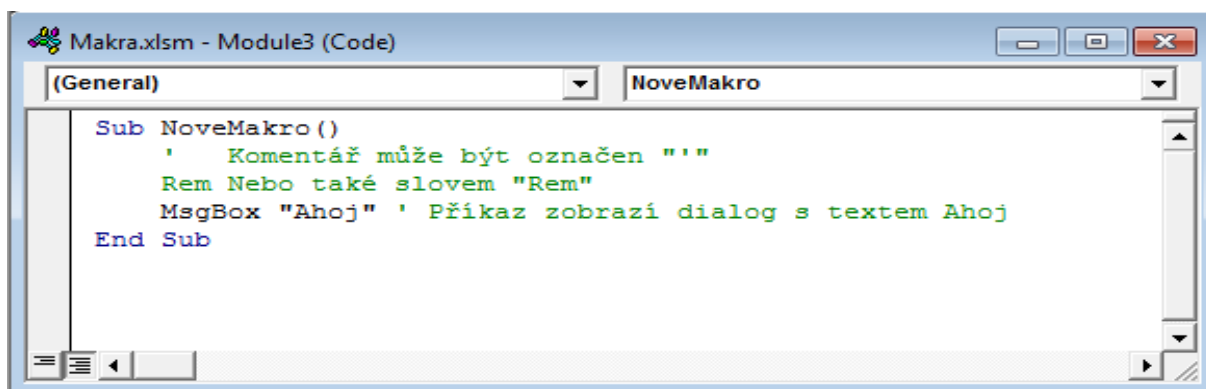
Obr. 28. Programové založení makra

3.3 Zápis programového kódu

Programový kód se zapisuje stejně jako běžný text. Každý příkaz se zapisuje na samostatný řádek, přičemž se striktně musí dodržovat názvy příkazů, mezery, symbolu operátorů atd. Programovací jazyk je velice striktní, jakýkoliv překlep či opomenutí způsobí nefunkčnost kódu.

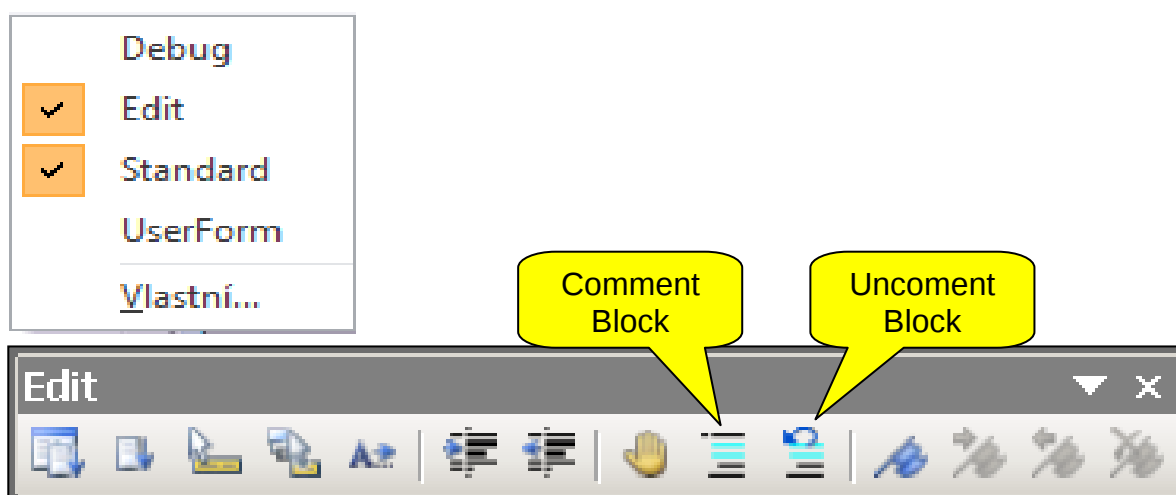
V následujícím příkladu na obrázku 27 je patrné, že se také dodržují pravidla formátování kódu ve smyslu odsazení řádku podřízenými s příkazy. Tento způsob zarovnání sice nemá na funkci kódu žádný vliv, ale je takto mnohem přehlednější.

Další zajímavou částí uvedeného příkladu jsou komentáře. Komentáře mají automaticky zelenou barvu a pro funkci programu nemají žádný vliv. Využívají se jako poznámky autora kódu nebo se také jimi některé příkazy dočasně vypínají. Komentář se vyrobí tak, že se na začátku vloží znak apostrofu (popřípadě se uvede slovo *Rem*). Vše od apostrofu do konce řádku je pak chápáno jako komentář.



Obr. 29. Programový kód makra s komentářem

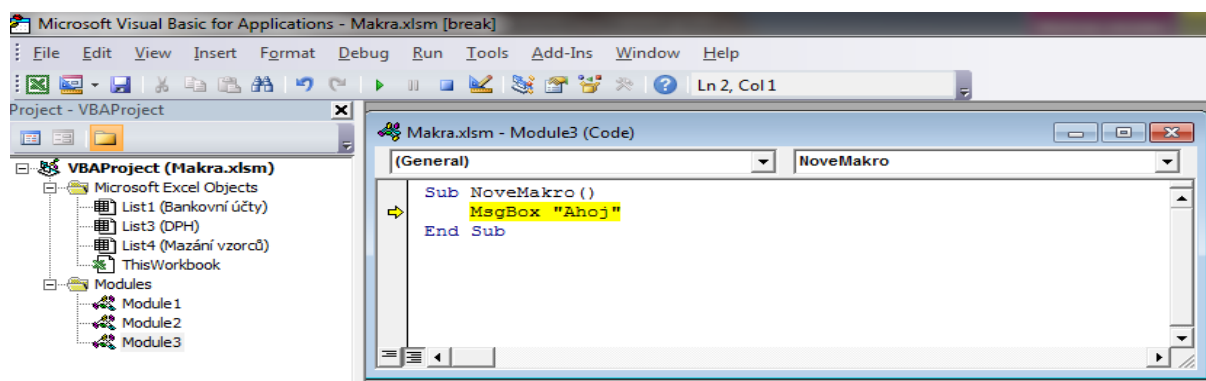
Další možností jak rychle vyrobit komentář je použít ovládací panel „Edit“, který se v editoru zobrazí přes nabídku kontextového menu vyvolaného klepnutím pravého tlačítka myši nad libovolným panelem nebo položkou Menu editoru. Na tomto panelu jsou dvě ikony „“ a „“. První z nich z vybraných řádku udělá komentář tím, že vloží znak apostrofu a druhé jej odebere (viz obr. 30).



Obr. 30. Zobrazení panelu Edit a použití příkazů pro vytvoření a odebrání komentářů

3.4 Test spuštění makra z prostředí editoru jazyka Visual Basic

Makro je možné ihned v prostředí editoru vyzkoušet. Nejrychleji se makro spustí tak, že se přemístí kurzor klávesnice kdekoliv dovnitř makra a stiskem tlačítka F5 nebo ikonou  se makro spustí. Zablokuje-li se makro (například z důvodu chyby) je možné makro rychle resetovat ikonou reset . Zablokované makro poznáme podle toho, že je libovolný řádek v makru označen žlutě a v titulku okna editoru se objeví text [break].



Obr. 31. Pozastavené makro



Shrnutí pojmů

Projekt obsahuje výčet stavebních prvků aplikace. Je nejdůležitější oknem a můžeme jej volně přemísťovat uchopením myši za titulkový pruh. Ve VBA je to vlastně každý otevřený dokument (například ve Wordu dokument, v Excelu sešit atd.). Přidávání a odebrání objektů projektu se provádí prostřednictvím kontextové nabídky (po stisku pravého tlačítka myši) nebo hlavní nabídky. Mezi objekty neboli stavební prvky patří například formulář a modul.

Modul je úložiště programového kódu, které obsahuje procedury, deklarace a funkce. Umožňuje definovat globální proměnné, uživatelské datové typy, procedury a funkce. Moduly můžeme přidávat nebo odstraňovat (viz obr. 24) přes okno Projektu.

Identifikátor je název programového prvku (algoritmu, proměnné či objektu), který jednoznačně prvek identifikuje v místě jeho úložiště. Ve stejném úložišti nemohou být dva identifikátory stejného jména. Pro identifikátor platí následující pravidla: nesmí obsahovat mezery, nesmí začínat číslicí a nesmí být shodný s klíčovými slovy jazyka Visual Basic. Navíc se nedoporučuje použití české diakritiky.



11. Co je identifikátor a jaká jsou pravidla jeho tvorby?
12. Kde se uchovávají makra?
13. Jakým způsobem můžeme makro otevřít pro editaci?
14. Jaký význam mají komentáře kódu?
15. Co jsou klíčová slova a jaká tvoří ohraničení makra?



1. Nad listem „Změna znaménka“ metodou záznamu vytvořte makro **ZmenaZnamenka**, měnící znaménko číselné hodnoty vybrané buňky. Vyzkoušejte funkčnost nad dvěma čísly ve sloupci.

Postup: před měněnou hodnotu vložíme prázdnou buňku a do ní zapíšeme hodnotu -1, poté nad buňkou zahájíme kopírování (CTRL+C) a nazpět vybereme požadovanou hodnotu. Vyvoláme dialog Vložit jinak, kde zvolíme hodnotu a násobit. Nakonec odstraníme pomocnou buňku s hodnotou -1.

2. Upravte makro **ZmenaZnamenka** tak, aby bylo možné změnit hodnoty celého označeného sloupce a přiřadte mu klávesovou zkratku **Ctrl+Shift+Z**. K ověření makra upravte znaménka hodnot tabulky tak, aby ve sloupci A byly pouze záporné hodnoty a ve sloupcích B a C pouze kladné hodnoty.

Postup: v tomto případě se musí makro upravit ručně:

- Otevřete makro (záložka "Zobrazení" tlačítko "Makra", v dialogu označit příslušné makro a stisknout tlačítko "Upravit").*
- Vyhledejte v makru oba výskyty příkazu `ActiveCell.Offset(?,?).Range("A1").Select` (na místě otazníku jsou číselné hodnoty označující relativní výběr buňky).*
- Upravte je do této podoby `Selection.Offset(?,?).Select`*

Vyzkoušejte si upravené makro nad řádkem hodnot a analyzujte jeho chování.

3. Programovou metodou (ve stejném modulu pod makro **ZmenaZnamenka**) vytvořte nové makro **ZmenaZnamenkaCisla**, jež bude měnit znaménko pouze číselných hodnot libovolně vybrané oblasti. Přiřadte novému makru klávesovou zkratku **Ctrl+Z** a modul pojmenujte „A_ZmenaZnamenka“. Funkčnost makra si ověřte nad maticí v listu „Změna znaménka“ tak, že všechny hodnoty převedete na záporná čísla. Vlastní kód makra je následující (kód zkopírujte):

```
Dim B As Range 'Deklarace promenne bunky
For Each B In Selection 'Cyklus, který projde všechny vybrane bunky
    'Je-li v bunce cislo, vynásobí se -1
    If IsNumeric(B.Formula) Then B.Value = B.Value * (-1)
Next 'Konec cyklu
```

4 ÚVOD DO PROGRAMOVÁNÍ V JAZYCE VISUAL BASIC FOR APPLICATION



Čas ke studiu: 2 hodiny



Cíl Po prostudování této kapitoly budete umět

- Popsat způsob vytváření kódu v jazyce VB
- Definovat základní pojmy programování
- Používat funkce MsgBox a InputBox



Výklad

Úspěšnost programové tvorby maker je závislá především na znalostech jazyka Visual Basic for Application. Tento jazyk byl poprvé uveden v roce 1991. Jazyk vychází z Visual Basicu a prakticky se od roku 1998 nezměnil. V daném roce byl totiž uveden Visual Basic 6.0, ze kterého Visual Basic for Application dodnes vychází. Například VBA v MS Office 2007 nese číselné označení 6.3. Teprve v Office 2010 je typové označení 7 avšak se od předchozí verze prakticky neliší.

V roce 2002 byl uveden nový programovací jazyk Visual Basic .NET jehož poslední verze nese označení 2010. Nicméně VBA rozhodně z tohoto jazyka nevychází a je dokonce výrazně odlišný. Pokud bychom oba jazyky mezi sebou začali srovnávat, zjistili bychom propastné rozdíly, přestože oba vycházejí ze stejného základu.

4.1 Úvod do jazyka Visual Basic

Makra jsou ve Visual Basicu prezentována jako subrutiny bez argumentů. Subrutina je v obecné definici algoritmem. Algoritmus je přesný popis práce řešení problému konečným počtem kroků. Dalším typem algoritmu jsou funkce, které se od subrutin liší tím, že mají návratovou hodnotu.

Algoritmy jsou tvořeny příkazy (dílčími povely algoritmu), kterým zpracovatel (překladač) jednoznačně rozumí. Ve VBA je každý příkaz standardně ohraničen řádkem. Příkaz může být tvořen:

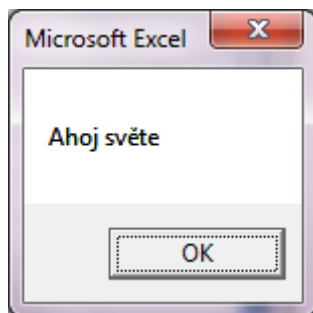
- voláním subrutiny nebo funkce,
- vyhodnocením výrazu,
- klíčovým slovem,
- kombinací.

Příkazy volání subrutin a funkcí

Příkaz volání subrutiny se realizuje zápisem názvu (identifikátoru) požadované subrutiny a uvedením seznamu hodnot jejich argumentů. Příkladem je volání subrutiny `MsgBox` (viz obr. 32), která má jeden povinný argument „prompt“ jímž se předává text, který má subrutina zobrazit v jednoduchém dialogu (obrázek 33).

```
Sub Main()  
    MsgBox "Ahoj světe"  
End Sub
```

Obr. 32. Příklad volání subrutiny



Obr. 33. Výsledek příkazu MsgBox "Ahoj světe"

Subrutin má však více argumentů, jimiž můžeme vzhled dialogu upravit. Ke správnému a úplnému zadání všech argumentů pomáhá editor zobrazením rychlé nápovědy, která se zobrazí ve žlutém poli (obr. 34). Tato nápověda se po rozpoznání subrutiny zobrazí zcela automaticky, ihned poté co se za názvem uvede mezera nebo čárka případně se může vyvolat explicitně ikonou . Nápověda informuje programátora o všech argumentech subrutiny, povinností tyto argumenty uvádět a aktuální pozici zadávání argumentu. Argumenty se oddělují čárkami, přičemž tučně se označí aktuálně zadávaný argument.

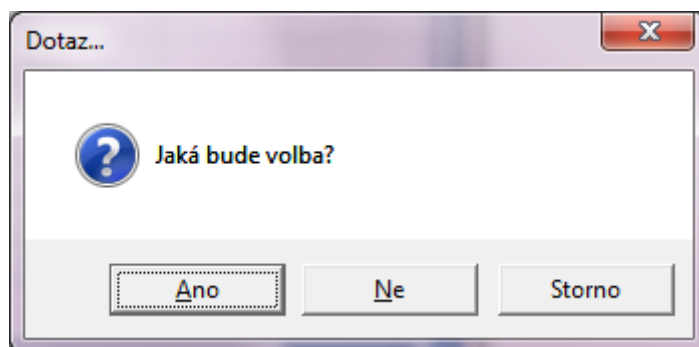
MsgBox |
MsgBox(**Prompt**, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult

Obr. 34. Zobrazení nápovědy po zadání názvu subrutiny

Povinné argumenty jsou uvedeny pouhým názvem, kdežto nepovinné jsou ohraničeny [hranatými] závorkami. Z obr. 34 je patrné, že subrutina dialogu `MsgBox` má jeden povinný argument `Prompt` a další čtyři nepovinné argumenty. Nepovinné argumenty se mohou pochopitelně vynechat nebo libovolně přeskočit. Subrutina pak na jejich místě použije implicitní hodnoty. Například pro příkaz

`MsgBox "Jaká bude volba?", vbYesNoCancel Or vbQuestion, "Dotaz..."`

bude výsledek dialogu odpovídat obr. 35.



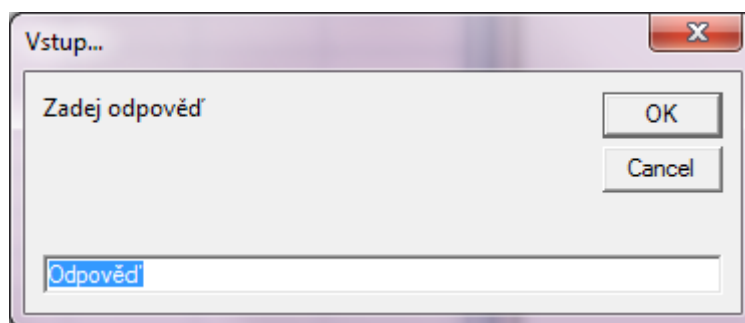
Obr. 35 – Ukázka provedení příkazu volání funkce MsgBox "

Příkazy volání funkcí jsou velmi podobné až na to, že funkce vracejí návratovou hodnotu, kterou je potřeba zachytit. Funkce je tak často součástí výrazů nebo je před funkcí uvedena proměnná, do které se navracená hodnota zapisuje.

Například na obr. 36 je volána funkce vstupního dialogu `InputBox`, jež vrací hodnotu, kterou do dialogu zadá uživatel. V příkladu je navracená hodnota přiřazena do proměnné `Odpoved`. Vzhled dialogu demonstruje obr. 37.

```
Sub Main()  
    Dim Odpoved  
    Odpoved = InputBox("Zadej odpověď", "Vstup...", "Odpověď" )  
End Sub
```

Obr. 36. Příkaz volání funkce InputBox



Obr. 37 – Demonstrace příkazu InputBox("Zadej odpověď", "Vstup...", "Odpověď")

Podstatný rozdíl je také v zadávání argumentů, kdy se oproti subrutinám uzavírají do závorek.

Konstanty

Často je potřeba přímo do kódu zadat určitou hodnotu (číslo, text, datum atd.). Tyto hodnoty jsou zapsány přímo v přirozeném formátu a během chodu algoritmu se nemění – jsou konstantní. Takovýmto hodnotám říkáme konstanty. Ve VBA máme následující konstanty:

- celočíselné konstanty v desítkové soustavě, například 10,
- celočíselné konstanty v šestnáctkové soustavě (hexa tvar), například &H10,
- konstanty reálných čísel (vždy se uvádí desetinná tečka) například 3.14,
- řetězcové konstanty (text je vždy ohraničen uvozovkami), například "Ahoj, jak se máš",
- logické konstanty - mají pouze dvě hodnoty True (interně -1) False (interně 0) ,
- časové konstanty (datum a čas), vždy se uvádí v anglickém formátu ohraničený symbolem #, například #1/31/2000 2:00:00 PM#.

Výrazy

Výrazem rozumíme operaci, jejímž výsledkem je jedna hodnota. Může být formátu:

- početní operace např. $3+2*10$,
- logické operace např. $a \geq b$,
- řetězcové (textové) operace např. "Petr" & " " & "Pavel",
- Kombinace např. "Máslo – cena: " & $100 * \text{DPH}$ & ", - Kč"

```
Sub Main()  
    D = 3^2-4*2*0.2  
End Sub
```

Výraz (jednoduchá početní operace)

Obr. 38 Přiřazení výsledku výrazu do proměnné

Výrazy jsou často součástí argumentu funkcí.

```

Sub Main()
    MsgBox 10 + 10 'Sečte a oznámí výsledek
End Sub

Sub Main()
    MsgBox "10 + 10 = " & 10 + 10
End Sub

```

Obr. 39 Výrazy jako argumenty funkcí

4.2 Příkazy dialogů (funkce) MsgBox() a InputBox()

MsgBox – funkce, která zobrazí dialogové okno. Používá se v případě, že potřebujeme uživatele na něco upozornit nebo mu něco oznámit. Podle stisknutého tlačítka vrací hodnotu, která určuje, které tlačítko uživatel stiskl (viz tabulka 1). Do samostatného příkazového řádku vypíšeme příkaz **MsgBox** po stisknutí mezerovníku se nám objeví nabídka parametrů-možností. Parametry-argumenty se dělí na povinné a nepovinné umístěné v hranatých závorkách.

Tabulka 1 – konstanty definice tlačítek

Identifikátor	Hodnota	Popis
vbOK	1	OK
vbCancel	2	Storno
vbAbort	3	Zpět
vbRetry	4	Znovu
vbIgnore	5	Ignorovat
vbYes	6	Ano
vbNo	7	Ne

Kromě textu zprávy určujeme, které tlačítko se objeví v nabídce dialogového okna.

Syntaxe:

MsgBox (prompt [, buttons] [, title] [, helpfile, context])

Popis argumentů:

- prompt** Řetězcový výraz zobrazený jako dialogová zpráva.
- buttons** Číselný výraz určující vzhled dialogu, tj. jaká bude mít tlačítka a ikonu. Hodnota vzniká součtem identifikačních čísel tlačítek, které mají být zobrazeny, ikon a předvoleného tlačítka viz tabulka a tabulka 1.
- title** Řetězcový výraz zobrazený v titulkovém pruhu dialogu. Pokud title vynecháme, je do titulkového pruhu vloženo jméno aplikace.
- helpfile** Řetězcový výraz určující název souboru, který obsahuje text nápovědy k dialogu.
- context** Číselný výraz, který je přiřazen k tématu nápovědy.

Hodnota argumentu **buttons** vzniká jako součet hodnot vybraných po jedné z následujících částí. V prvním sloupci je určen **identifikátor** (konstanta), který lze použít místo číselné hodnoty, dále hodnota a popis:

Tabulka 2 - Určení zobrazení tlačítek v dialogu

Identifikátor	Hodnota	Popis
vbOKOnly	0	Zobrazí pouze tlačítko "OK".
vbOKCancel	1	Zobrazí tlačítka "OK" a "Storno".
vbAbortRetryIgnore	2	Zobrazí tlačítka "Zpět", "Znovu" a "Ignorovat".
vbYesNoCancel	3	Zobrazí tlačítka "Ano", "Ne" a "Storno".
vbYesNo	4	Zobrazí tlačítka "Ano" a "Ne".
vbRetryCancel	5	Zobrazí tlačítka "Znovu" a "Storno".

Tabulka 1 - Určení stylu ikony

Identifikátor	Hodnota	Zobrazení
vbCritical	16	 vbCritical
vbQuestion	32	 vbQuestion
vbExclamation	48	 vbExclamation
vbInformation	64	 vbInformation

InputBox – funkce, dialogové okno umožňující získat od uživatele textové i číselné hodnoty.

Syntaxe:

`InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])`

Funkce otvírá dialogové okno, ve kterém zobrazí hodnotu `prompt`. Výraz `title` určuje název v titulkovém pruhu okna. Při vynechání tohoto parametru bude místo titulku jméno aplikace. Parametr `default` určuje hodnotu, která se nám objeví ve vstupním poli okna. Tuto hodnotu můžeme nahradit nebo potvrdit. Parametry `xpos` a `ypos` jsou souřadnice určující vodorovné a svislé umístění levého horního rohu okna. Mohou to být numerické výrazy udávající hodnotu ve *twip*ech (Jeden centimetr je 567 *twip*ů.). Pokud chceme okno vystředit vodorovně, vynecháme parametr `xpos`. U parametru `ypos` je to trochu jiné, zde při vynechání bude okno umístěno svisle zhruba v jedné třetině obrazovky. Pro zpřístupnění uživatelské nápovědy slouží parametry `helpfile` a `context` v daném vstupním okně.

Popis argumentů:

prompt Řetězcový výraz zobrazený jako dialogová zpráva. Maximální délka promptu je závislá na šířce znaků písma, které je použito (cca 1024 znaků).

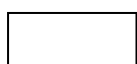
title Řetězcový výraz zobrazený v titulkovém pruhu dialogu. Pokud `title` vynecháme, je do titulkového pruhu vloženo jméno aplikace.

default	Parametr default určuje hodnotu, která se objeví ve vstupním poli okna. Tuto hodnotu pak může uživatel buď změnit, nebo potvrdit.
xpos	Určuje vodorovnou souřadnici dialogového okna, pokud parametr vynecháme, bude okno vodorovně vystředěné.
ypos	Určuje svislou souřadnici dialogového okna, pokud parametr vynecháme, bude okno svisle vystředěné.
helpfile	Řetězcový výraz určující název souboru, který obsahuje text nápovědy k dialogu. Je-li argument helpfile určen, musí být také určen argument context .
context	Číselný výraz, který je přiřazen k tématu nápovědy.

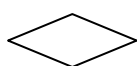
4.3 Vývojové diagramy

Pro popis algoritmu se často používají vývojové diagramy. Je to velice přehledný způsob popisu algoritmu. V následujících kapitolách budou některé postupy vyjádřeny také pomocí vývojového diagramu.

Základní grafické prvky vývojového diagramu:



Obdélník – vyjadřuje jeden příkaz nebo sekvenci navazujících příkazů.

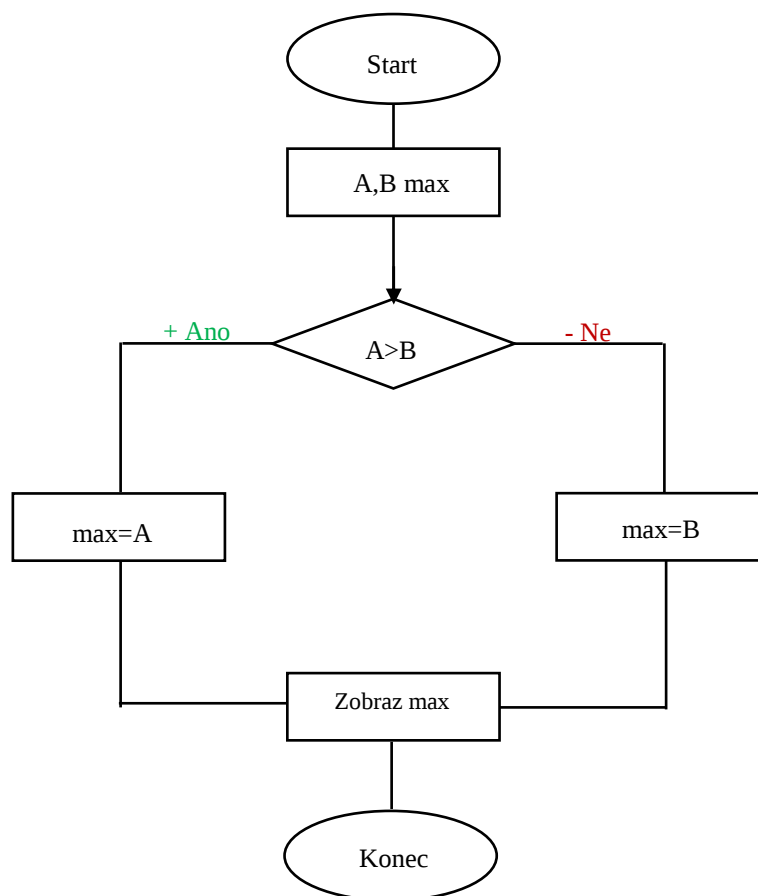


Kosočtverec – rozhodovací blok ve většině případů s jedním vstupem a dvěma výstupy.



Elipsa – začátek nebo konec algoritmu.

Například algoritmus pro nalezení větší ze dvou hodnot můžeme zobrazit pomocí vývojového diagramu následujícím způsobem.



Obr. 40 Příklad vývojového diagramu



Shrnutí pojmů

Algoritmus – je přesný popis práce pro řešení daného zadání konečným počtem kroků. Algoritmus lze opakovaně použít, ale za stejných podmínek musí dávat vždy stejné řešení. Ve VBA algoritmy tvoří subrutiny a funkce. Každý algoritmus má svůj jedinečný název (identifikátor).

Příkaz - je jeden samostatný povel algoritmu.

Program – je komplexní algoritmus řešící konkrétní problém. Tento algoritmus je v tomto případě zapsán podle syntaxe programovacího jazyka VBA.

Subrutina – je ohraničený algoritmus, který může být využíván opakovaně v různých místech programu a jenž nemá návratovou hodnotu. Makro je vždy tvořeno subrutinou bez argumentů.

Funkce – je uzavřený algoritmus, který může být využíván opakovaně v různých místech programu a má návratovou hodnotu.

Návratová hodnota – je hodnota, kterou vrací funkce jako výsledek řešené operace.

Argument – je vstupní parametr algoritmu, přes nějž se do algoritmu předávají hodnoty.

Konstata – je hodnota přímo uvedená v programovém kódu.



Otázky

16. Co jsou povinné a volitelné argumenty a jak je poznáme?
17. Jaký je rozdíl při volání subrutiny a funkce?
18. Jak poznáme nepovinné argumenty?
19. Jaký je rozdíl mezi makrem a subrutinou?
20. Co je návratová hodnota funkce?
21. Co je to příkaz?

5 PROMĚNNÉ A DATOVÉ TYPY



Čas ke studiu: 1,5 hodiny



Cíl Po prostudování této kapitoly budete umět

- Popsat základní datové typy jazyka Visual Basic
- Deklarovat proměnné



Výklad

V předchozí kapitole bylo uvedeno, jak do programového kódu zapsat hodnoty formou konstant. Bohužel, v reálných algoritmech převážně dopředu hodnoty neznáme. Ty získáme až za běhu algoritmu, například dotazem uživatele ze souboru, ze síťové komunikace, z připojených periférií apod. V těchto případech místo konkrétních hodnot použijeme symbolické náhrady tzv. proměnné.

5.1 Automatické proměnné

Visual Basic je velice uživatelsky přívětivý a snaží se programátorům ušetřit co nejvíce práce. Bohužel ne vždy je tato vstřícnost přínosem. Příkladem jsou proměnné, které standardně může programátor používat, aniž by je předem deklaroval. Například na obr. 41 je zobrazen funkční programový kód, který používá dvě číselné proměnné `Cislo1` a `Cislo2`. Program zobrazí výsledek součtu těchto proměnných ve dvou dialogích (viz obr. 42).

```
' Použití proměnných
Sub Main()
    Cisko1 = 10 'Automatické proměnné bez deklarace
    Cisko2 = 10
    MsgBox Cisko1 & " + " & Cisko2 & " = " & Cisko1 + Cisko2, , "Správně" 'Správně
    MsgBox Cisko1 & " + " & Cisko2 & " = " & Cisko1 + Cisko2, , "S překlepem" 'Překlep
End Sub
```

Obr. 41 Příklad problému použití automatických proměnných

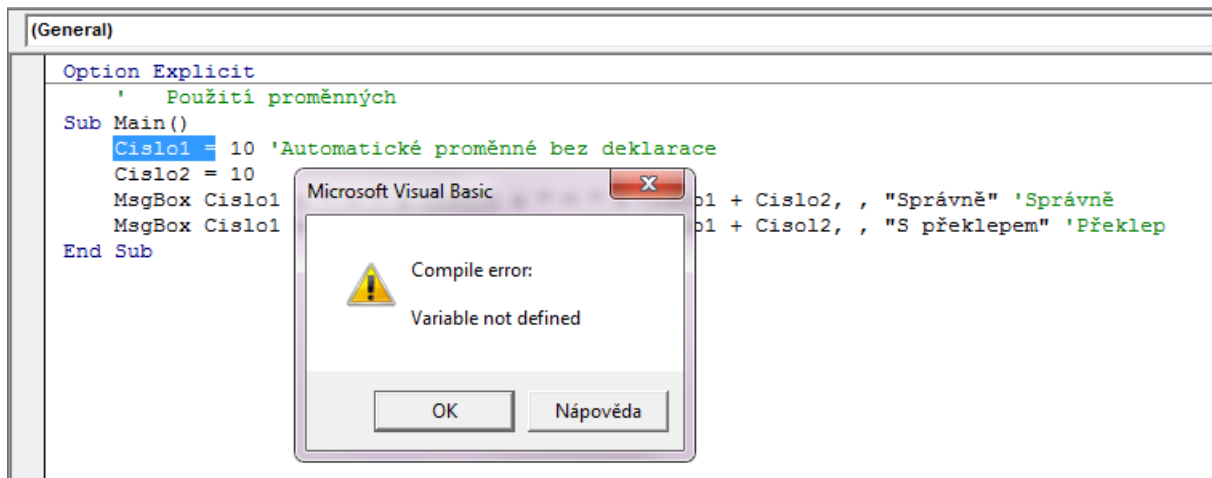


Obr. 42 Zobrazení výsledků příkladu problému použití automatických proměnných

Příklad demonstruje, jak snadně se automatické proměnné používají, ale jak jsou zároveň citlivé na chyby. Problémem je, že pokud si výsledek programátor nezkontroluje, nebude o chybě vůbec vědět. Program se totiž chová, jako by bylo vše v pořádku.

Visual Basic for Application se standardně chová tak, že jakmile narazí na identifikátor, který nezná, automaticky jej považuje za novou proměnnou, kterou založí. Současně ve Visual Basicu platí, že každé nově založené proměnné přiřadí hodnotu nula (pro Variant Empty a pro text prázdný řetězec). V příkladu u druhého dialogu se ve skutečnosti objeví hodnoty $10 + 0$, takže z matematického hlediska je výsledek naprosto pořádku.

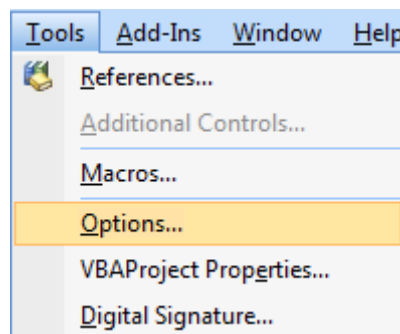
Těmto nepříjemným chybám lze snadno zabránit tím, že požádáme Visual Basic, aby prováděl kontrolu deklarace proměnných. Jinými slovy, aby jakýkoliv neznámý identifikátor jasně označil. Tato žádost se provádí direktivou `Option Explicit`, která se musí uvést na začátku modulu (viz obr. 43).



Obr. 43 Příklad problému použití automatických proměnných

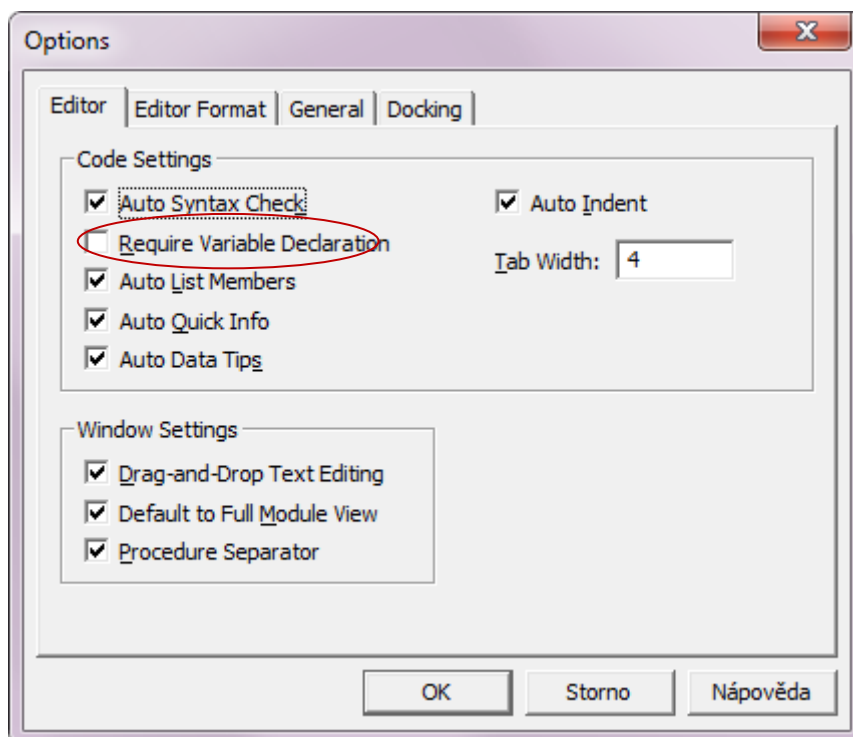
Bohužel po dopsání této direktivy se subrutinu vůbec nepodaří spustit, neboť hned první příkaz používá neznámou proměnnou `Cislo1`. V programu tak musíme nejdříve provést deklaraci proměnných.

Aby se nemuselo pokaždé direktivu `Option Explicit` zapisovat, nabízí editor jazyka Visual Basic volbu automatického vyplnění u každého nově založeného modulu. Tuto funkci zapneme v prostředí editoru přes nabídku „Tools“ volbou položky „Options“.



Obr. 44 Vyvolání dialogu Options

Zobrazí se dialog „Options“, kde na kartě editor se zatrhne volba „Require Variable Declaration“.



Obr. 45 Zapnutí automatického doplňování direktivy Option Explicit

5.2 Deklarace proměnných

Deklarací proměnných sdělujeme překladači, že uvedený identifikátor bude odteď proměnná. Současně můžeme (doporučuje se vždy provést) uvést datový typ této proměnné.

Zjednodušená syntaxe:

Dim jméno_proměnné1, jméno_proměnné2

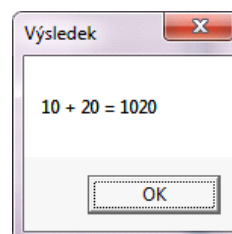
Pokud doplníme deklaraci do již zmíněného příkladu, ihned odhalíme překlep. Po opravě již lze subrutinu vykonat.

```
Option Explicit
Sub Main()
    Dim Cisl1, Cisl2
    Cisl1 = 10
    Cisl2 = 10
    MsgBox Cisl1 & " + " & Cisl2 & " = " & Cisl1 + Cisl2, , "Správně" 'Správně
    MsgBox Cisl1 & " + " & Cisl2 & " = " & Cisl1 + Cisl2, , "S překlepem" 'Překlep
End Sub
```

Obr. 46 Odhalení překlepu při použití proměnných

I když se nyní jeví, že je vše již v pořádku, není tomu tak. Stačí, když se doplní příklad o příkazy zadání hodnot uživatelem (přidáme dialogy `InputBox`). Teď dostaneme ještě záhadnější výsledek (obr. 47).

```
Option Explicit
Sub Main()
    Dim Cisl1, Cisl2
    Cisl1 = InputBox("Zadej číslo 1", , 10)
    Cisl2 = InputBox("Zadej číslo 2", , 20)
    MsgBox Cisl1 & " + " & Cisl2 & " = " & Cisl1 + Cisl2, , "Výsledek"
End Sub
```



Obr. 47 Chyba výpočtu u proměnných typu Variant

Tentokrát je na vině kombinace dvou vlastností jazyka VBA. První část problému způsobuje datový typ `Variant`, který má každá proměnná pokud u její deklarace nebyl její typ specifikován. V příkladu byla použita zkrácená forma deklarace, takže obě proměnné jsou datového typu `Variant`. Druhou částí problému je fakt, že ve Visual Basicu operátor `+` provádí dvě různé operace. Buď provádí součet dvou čísel, nebo spojování dvou textů. Vše závisí na operandech, jsou-li oba textové, provede spojení dvou textů. Je-li alespoň jeden číselný, pokusí se provést operaci součtu.

Jelikož datový typ `variant` si vždy pamatuje hodnotu ve formátu, v jaké ji obdržel a zároveň funkce `InputBox` vždy vrací vstup v textovém formátu, dojde ve výrazu ke spojení textů "10" a "10".

Uvedeným problémů se dá snadno vyhnout, použije-li se úplná syntaxe deklarace. Ta se liší tím, že se za každou proměnnou uvede její datový typ.

Plná syntaxe:

```
Dim jméno_proměnné1 As typ_proměnné, jméno_proměnné2 As typ_proměnné
```

Příklad:

```
Option Explicit
Sub Main()
    Dim Cisl1 As Long, Cisl2 As Long
    Cisl1 = InputBox("Zadej číslo 1")
    Cisl2 = InputBox("Zadej číslo 2")
    MsgBox Cisl1 & " + " & Cisl2 & " = " & Cisl1 + Cisl2, , "Výsledek"
End Sub
```

Tentokráte bude výpočet již zcela korektní.

5.3 Datové typy

Datový typ jasně specifikuje druh proměnné a její formát uložení v paměti, s čímž souvisí také jejich přesnost. Datové typy můžeme dělit podle druhu uchovávaných hodnot na:

- Číselné datové typy, které se dále dělí na:
 - celočíselné datové typy:
 - Byte,
 - Integer,
 - Long
 - reálné datové typy:
 - Single,
 - Double,
 - Currency,
- Speciální typ `Decimal`
- Řetězcový datový typ `String`.
- Datový typ `Date` pro uchování Datumu a času.
- Datový typ `Boolean` pro uchování logické hodnoty `True/False`.
- Speciální datové typy:

- uživatelský datový typ **Type**,
- výčtový typ **Enum**.
- Objektové datové typy.

Číselné datové typy:

Byte	(celočíselný) Ukládá čísla v intervalu 0 – 255.
Integer	(celočíselný) Nejčastější použití je pro proměnné sloužící jako parametry cyklu, čítače a proměnné obsahující bitové informace (indikátory, příznaky atd.). Typová přípona - %
Long	(celočíselný) Oproti předchozímu typu, má dvojnásobnou délku uložení. Z tohoto důvodu se používá pro celočíselné velké hodnoty. Má přesnost pouze na 9 čísel. Typová přípona - &
Single	(reálný) Jedná se o číselné hodnoty v pohyblivé řadové čárce s jednoduchou přesností. Má přesnost pouze na 7 čísel. Typová přípona - !
Double	Jedná se o číselné hodnoty v pohyblivé řadové čárce s dvojnásobnou přesností. Má přesnost pouze na 15 čísel. Typová přípona - #
Currency	Je reálný typ s přesností 15 cifer celého čísla a čtyřmi desetinnými místy. Je zaručen pro uchování finančních hodnot.

Tabulka 4 – Číselné datové typy

Data type	Popis	Velikost [B]	Rozsah
Byte	Celé kladné číslo	1	0 až 255
Integer	Celé číslo	2	-32 768 až 32 767

Long	Celé číslo	4	-2 147 483 648 až 2 147 483 647
Single	Reálné číslo v pohyblivé řádové čárce s jednoduchou přesností na 7 cifer	4	±3.402823E38 do ±1.401298E-45
Double	Reálné číslo v pohyblivé řádové čárce s dvojitou přesností na 15 cifer	8	±1.79769313486231E308 do ±4.94065645841247E-324

5.4 Obor platnosti proměnných

Ke způsobu deklarace se vztahuje doba platnosti proměnných. Proměnné, které se deklarují uvnitř algoritmů (funkcí a subrutin) jsou platné pouze v těchto algoritmech. Vznikají a zanikají spolu s algoritmem. Výjimku tvoří statická proměnná, která s rutinou nezaniká a pamatuje si svou hodnotu pro příští spuštění algoritmu. U statické proměnné se místo klíčového slova `Dim` uvádí `Static`.

Proměnné mohou být také deklarovány na úrovni modulu ihned za direktivou `Option Explicit`. Pak hovoříme o globálních proměnných. Tyto proměnné vznikají při spuštění libovolné subrutiny modulu a zanikají až s uzavřením dokumentu. U globálních proměnných se místo klíčového slova `Dim` uvádí slovo `Private` nebo `Public`.

5.5 Základní operátory

= přiřazení, přiřadí hodnotu určenou pravým výrazem do proměnné vlevo od operátoru

`Cislo = 10` ‘promena Cislo ma hodnotu 10

+ součet dvou čísel nebo spojení dvou textů

`Cislo = 10 + 20` ‘30

- Rozdíl hodnot nebo změna znaménka

`Cislo = -10 - 20` ‘-30

*	Násobení
	Cislo = 3 * 3 '9
/	Reálné dělení
	Cislo = 10 / 3 '3.33333333
\	Celočíselné dělení
	Cislo = 10 \ 3 '3
Mod	Zbytek po celočíselném dělení
	Cislo = 10 Mod 3 '1
^	Umocnění/Odmocnění
	Cislo = 10 ^ 2 '100
	Cislo = 27 ^ (1/3) '3
&	Spojování textů
	CeleJmeno = "James" & " Bond" '"James Bond"

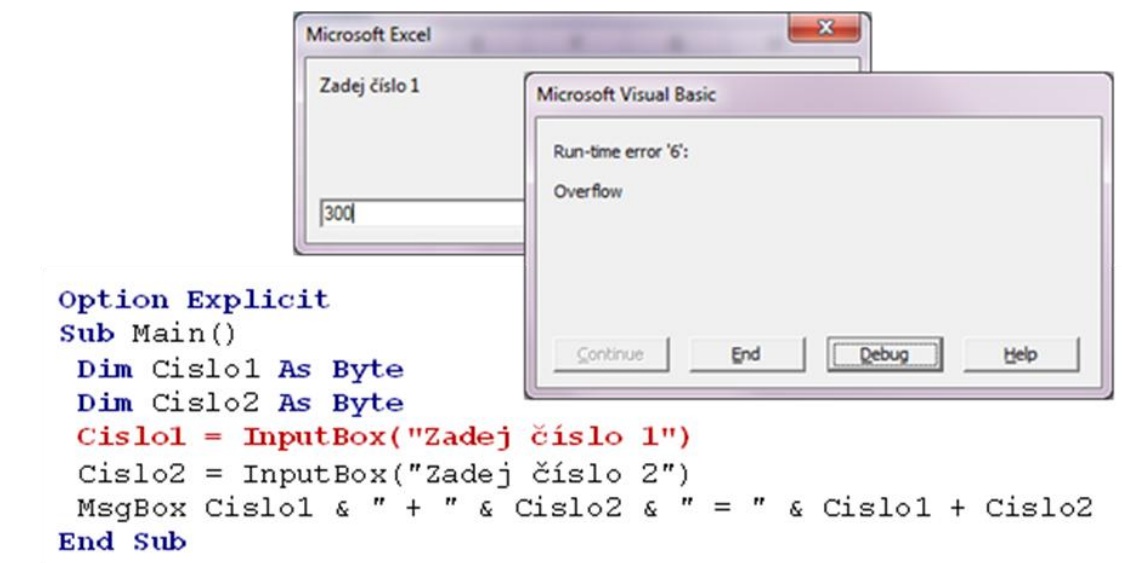


Obr. 48 Priority základních operátorů

5.6 Výjimky při práci s proměnnými

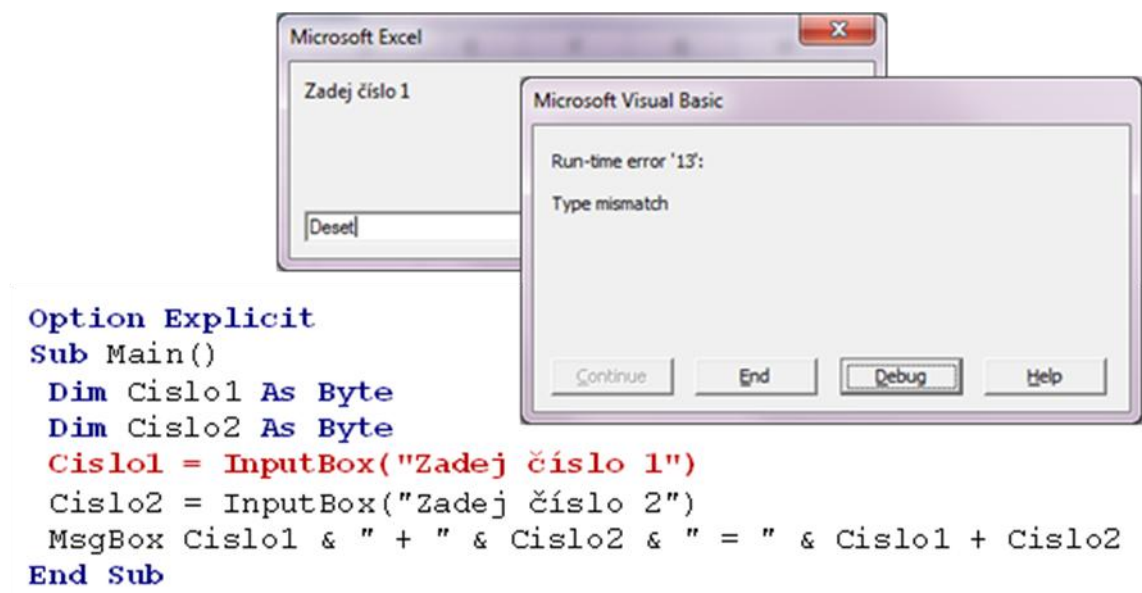
Během chodu programu, mohou v souvislosti s proměnnými vzniknout dvě chyby (výjimky):

- **Přetečení Overflow.** Tato výjimka vzniká, když se program pokusí uložit do proměnné hodnotu mimo rozsah jejího datového typu. K výjimce může také dojít, když výsledek výrazu přesahuje velikost největší proměnné ve výrazu.



Obr. 49 Vznik výjimky přetečení

- Nesoulad typů `type mismatch`. V Tato výjimka vzniká, vždy když se program pokusí uložit do proměnné hodnotu, která nemůže být konvertována na datový typ proměnné. Např. slovo „Deset“ nelze uložit do číselné proměnné typu `Integer`.



Obr. 50 Vznik výjimky nesouladu typů



Řešený příklad

Vytvořte algoritmus výpočtu reálných kořenů kvadratické rovnice

```
Sub Main()
    Dim D As Double
    Dim a As Double, b As Double, c As Double
    Dim x1 As Double, x2 As Double
    'Diskriminant
    'Koeficienty kvadraticke rovnice
    'Hledane reseni
    'Ziskani koeficientu rovnice
    a = InputBox("Zadej A:", "Ax^2 + Bx + C")
    b = InputBox("Zadej B:", "Ax^2 + Bx + C")
    c = InputBox("Zadej C:", "Ax^2 + Bx + C")
    'Vypocet diskriminantu
    D = b ^ 2 - 4 * a * c
    'Vypocet korenu rovnice
    x1 = (-b + Sqr(D)) / (2 * a)
    x2 = (-b - D^(1/2)) / (2 * a)
    'Vypis vysledku
    MsgBox "x1 = " & x1 & vbCrLf & "x2 = " & x2, , "
    "Řešení rovnice: " & a & "x^2 + " & b & "x + " & c
End Sub
```



Shrnutí pojmů

Proměnná je symbolická reprezentace hodnoty, kterou nabude až během činnosti programu.

Deklarace proměnné je příkaz programovacího jazyka, kterým se oznamuje založení proměnné včetně datového typu.

Datový typ jasně specifikuje druh proměnné a její formát v paměti, s čímž souvisí také jejich přesnost.

Výjimka je chyba, která vzniká za chodu programu.



Otázky

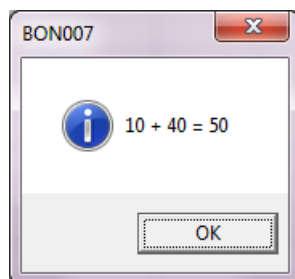
22. K čemu slouží direktiva Option Explicit?
23. Jaké jsou celočíselné datové typy ve VB a v čem se liší?
24. Ve kterém datovém typu bude hodnota 123456789 uchována přesně Long, Single?
25. Co znamená výjimka přetečení (overflow) a kdy vzniká?
26. Co znamená výjimka nesoulad typů (type mismatch)?
27. Jakým způsobem se realizuje 3. odmocnina čísla?
28. Jaký bude výsledek výrazu $3 + 10 / 2 * (3 + 2) \text{ Mod } 5$?



Úlohy k řešení

Pro prostředí MS Excel si stáhněte dokument VB03.xlsx:

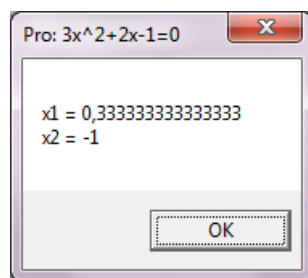
1. V modulu „B_SoucetCisel“ opravte a doplňte programový kód subrutiny **SoucetCisel** tak, aby prováděla součet dvou celých čísel každé o maximální hodnotě +/- 32000, přičemž vstupní dialogy "InputBox" budou mít předdefinovanou hodnotu 0. Zobrazující dialog bude mít v titulku vaše osobní číslo a v textu bude zobrazen plný zápis součtu a výsledku. Například: pro součet čísel 10 a 40 bude dialog zobrazovat: $10 + 40 = 50$.



Postup:

- a. Otevřete modul s makrem.

- b. Zápisem `Option Explicit` zapněte kontrolu deklarací proměnných a pokusem o spuštění odhalte chybějící deklaraci a překlep.
 - c. Doplňte deklaraci proměnné `Cislo 2`.
 - d. Upravte výraz argumentu funkce `MsgBox` (opravte překlep a doplňte proměnnou).
 - e. Spuštěním odhalte možné problémy.
 - f. Zvolte vhodný datový typ (optimálně `Long`) a upravte deklarace.
2. V modulu „C_KorenyKvadratickeRovnice“ opravte a doplňte programový kód subrutiny **ReseniKvadratickeRovnice** tak, aby na základě uživatelem zadaných koeficientů kvadratické rovnice vrátil její reálné kořeny (pouze má-li rovnice řešení). Pro testování použijte kladné hodnoty pro koeficienty A a B, a zápornou hodnotu pro koeficient C (úloha má vždy řešení).



Postup:

- a. Otevřete modul s makrem.
- b. Realizujte programový zápis výrazů:

$$D = b^2 - 4ac$$

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$$

6 LADĚNÍ PROGRAMOVÉHO KÓDU



Čas ke studiu: 1,5 hodiny



Cíl Po prostudování této kapitoly budete umět

- Pracovat s ladicími nástroji editoru jazyka Visual Basic
- Analyzovat důvody vzniku výjimek



Výklad

Předchozí kapitola byla zakončena popisem dvou výjimek, které mohou nastat při přiřazování hodnot do proměnných. Výčet možných výjimek je ale daleko větší. Přičemž málokdy je možné z hlášení vzniku výjimky rozpoznat důvod jejího vzniku. V tomto okamžiku je potřeba program pozastavit a příkaz, který výjimku vyvolal analyzovat.

Jindy zase program běží bez jakékoliv výjimky, ale neprovádí přesně to co má. V tomto případě je potřeba program sledovat při každém příkazu a tím můžeme odhalit chybu v logice programu nebo jiné příčiny neúspěchu.

Oba případy analýzy kódu umožňuje ladění programů...

- Laděná programů je účinná metoda odhalování chyb algoritmů.
- Během ladění je program vykonáván příkaz po příkazu (krok za krokem) tzv. krokování.
- Po každém příkazu je možné sledovat obsah proměnných.
- Každý příkaz se vykonává na podnět programátora.

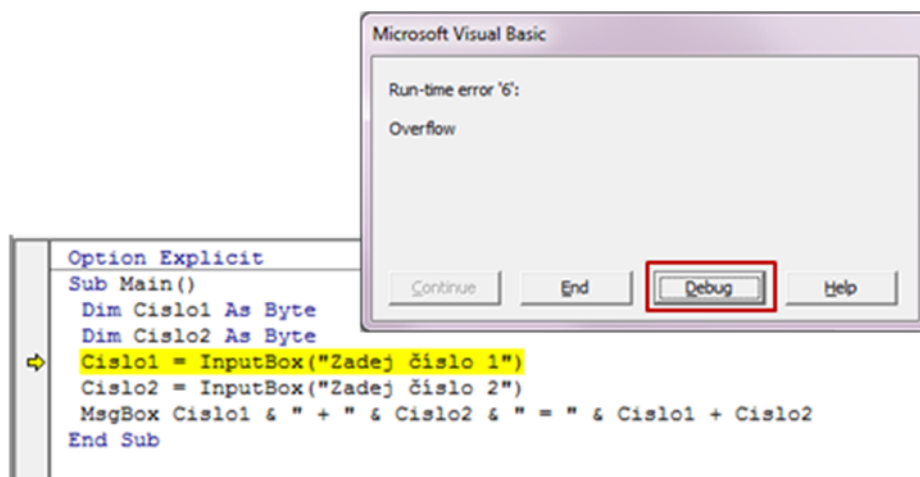
6.1 Metody zahájení ladění

Ladění je možné zahájit třemi základními způsoby:

1. tlačítkem „Debug“ v dialogu oznámení výjimky,
2. klávesou F8 nebo ikonou **Step Into**,
3. běžným spuštěním a automatickým přerušením na zarážce.

Zahájení ladění po výjimce

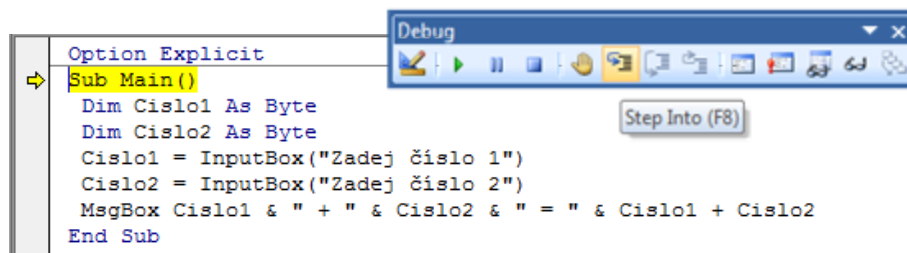
V tomto případě se program pozastaví na příkazu (řádku), který výjimku způsobil.



Obr. 51 Zahájení ladění tlačítkem Debug

Zahájení krokování

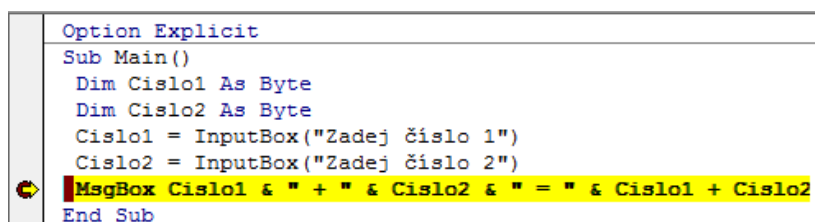
V tomto případě je nezbytné nejprve otevřít algoritmus v editoru jazyka Visual Basic a přenést do něj kurzor klávesnice. Ladění se zahájí funkční klávesou F8 nebo ikonou **Step Into** na panelu nástrojů **Debug**. Program se přeruší na hlavičce algoritmu.



Obr. 52 Zahájení ladění ikonou Step Into

Přerušním na zarážce

Tento způsob využívá zarážek (breakpoints), které automaticky pozastaví chod programu. Nejprve se zarážka umístí na požadovaném řádku a následně se algoritmus spustí.



Obr. 53 Přerušení programu na zarážce

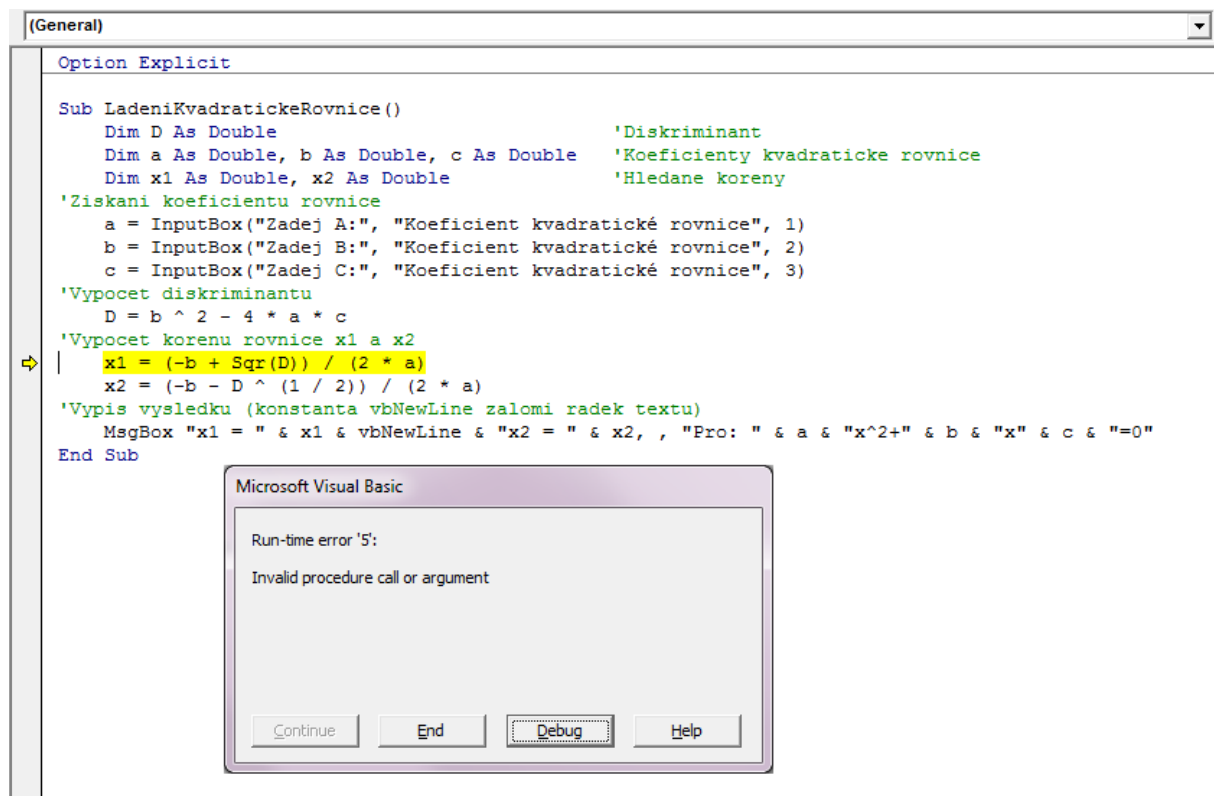
Ukončení ladění



Ladění je možné kdykoliv přerušit tlačítkem Reset.

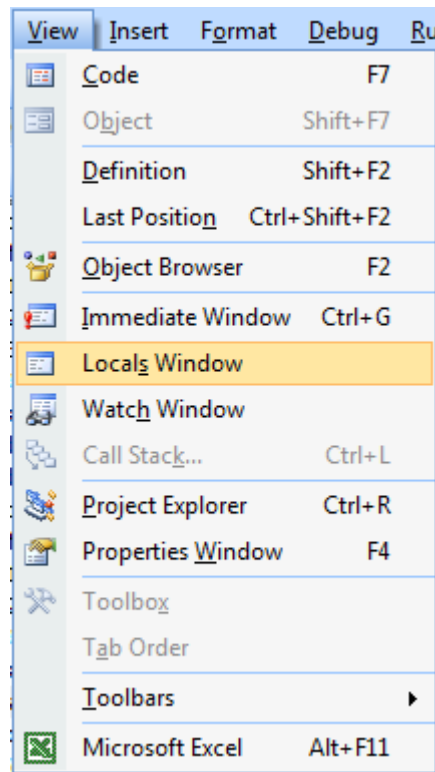
6.2 Sledování hodnot proměnných – analýza vzniku výjimky

Nejčastěji se ladění zahajuje kvůli analýze vzniku výjimky. Například máme vytvořený algoritmus výpočtu reálných kořenů kvadratické rovnice (viz obr. 54). Pokud zadáme koeficienty a, b, c hodnotami 1, 2, 3 vznikne výjimka č. 5 – chybné volání nebo chyba v argumentu funkce.



Obr. 54 Vznik výjimky u algoritmu řešení reálných kořenů kvadratické rovnice

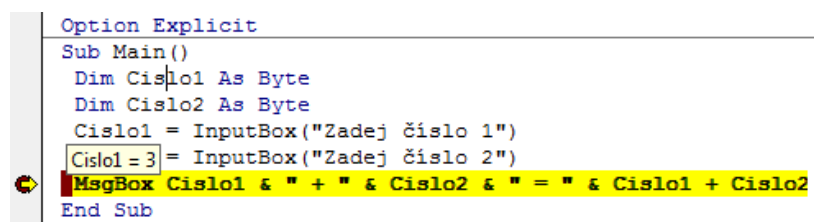
Zahájíme-li ladění tlačítkem „Debug“ program se pozastaví na řádku, který výjimku vyvolal. Pro analýzu příčiny je nutné zjistit stav jednotlivých proměnných výrazu.



Obr. 55 Zobrazení sledovacích oken

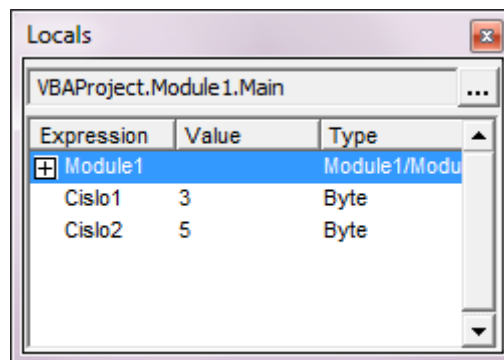
Obsah proměnné je možné zjistit několika způsoby:

1. Okamžitým náhledem - zobrazí se po ponechání kurzoru nad proměnou v kódu



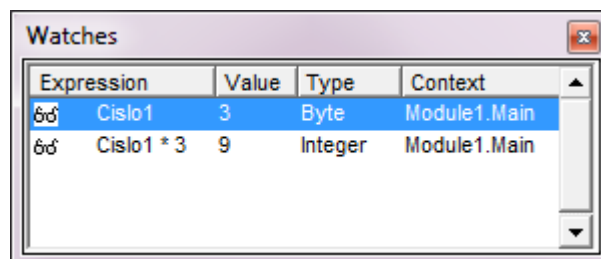
Obr. 56 Okamžitý náhled na hodnotu proměnné

2. Pomocí okna **Locals** – okno se zobrazí nabídkou „Locals Window“ v menu „View“. V okně se zobrazí všechny proměnné laděné subrutiny nebo funkce. Okno navíc umožňuje okamžitou editaci hodnot proměnných – (editace se otevře poklepáním nad požadovanou hodnotou).



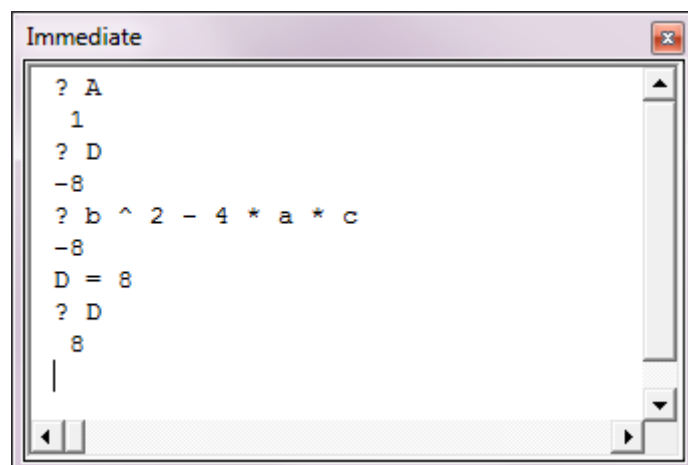
Obr. 50 Okno Locals

3. Pomocí okna **Watches** – okno se zobrazí nabídkou „Watch Window“ v menu „View“. Toto okno je implicitně prázdné, proměnou je možné přidat přetažením myši, vepsáním nebo také přes nabídku „Add Watch...“ kontextové menu, které se zobrazí po stisku pravého tlačítka myši nad proměnnou v kódu. Také přes toto okno je možné editovat hodnoty proměnné. Oproti oknu Locals navíc umožňuje zobrazovat výsledky výrazů.



Obr. 57 Okno Watches

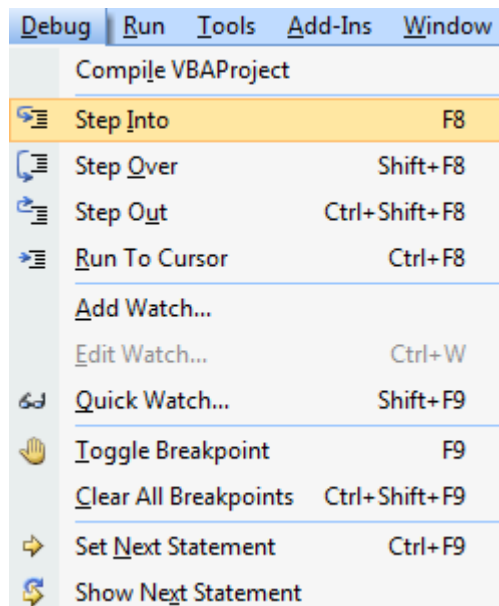
4. Prostřednictvím okna **Immediate** – okno se zobrazí klávesovou zkratkou Ctrl+G, nebo nabídkou „Immediate Window“ v menu „View“. Okno se chová jako běžný textový editor. Pro zobrazení hodnoty proměnné se do okna zapíše otazník, pak název proměnné a stiskne se klávesa ENTER. Hodnota se zobrazí na novém řádku. Stejným postupem je možné získat výsledek výrazu. V okně je také možné měnit hodnoty proměnných příkazem přiřazení nebo zkoušet libovolné příkazy.



Obr. 58 Okno Immediate

6.3 Krokování

Krokování je řízené provádění algoritmu příkaz po příkazu. Po každém kroku je program pozastaven a čeká na povel programátora k provedení dalšího kroku. Jednotlivé funkce provádění kroků jsou dostupné v menu „Debug“ nebo také v nástrojovém panelu „Debug“.



Obr. 59 Nabídka Debug

Mezi základní metody krokování jsou:

Step Into (krok dovnitř) je nejčastější metoda. Provede se vždy jeden příkaz, přičemž je-li tento příkaz volání uživatelské subrutiny nebo funkce přejde do krokování této rutiny. Nejrychleji se tato operace provádí funkční klávesou F8.

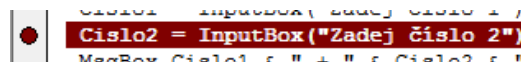


Step Over (krok přes) provede také jeden příkaz, přičemž je-li tento příkaz volání uživatelské subrutiny nebo funkce, vykoná se celá v jednom kroku. Nejrychleji se tato operace provádí kombinací kláves Shift + F8.

Step Out (krok ven) se používá v případě, když se během krokováním programátor nechtěně vnoří do uživatelské subrutiny či funkce, přičemž ji nepotřebuje ladit. Tento povel dokončí algoritmus, vyskočí z něj ven a pozastaví se ihned na příkazu po volání dané subrutiny či funkce. Nejrychleji se tato operace provádí kombinací kláves Ctrl + Shift + F8.

Run to Cursor (Běh ke kurzoru) je metoda rychlého posunu vpřed při ladění algoritmů. Využívá se v okamžiku, kdy programátor během ladění chce rychle pokročit vpřed. Pak přesune na požadované místo kurzor klávesnice a vyvolá operaci. Nejrychleji se tato operace provádí kombinací kláves Ctrl + F8.

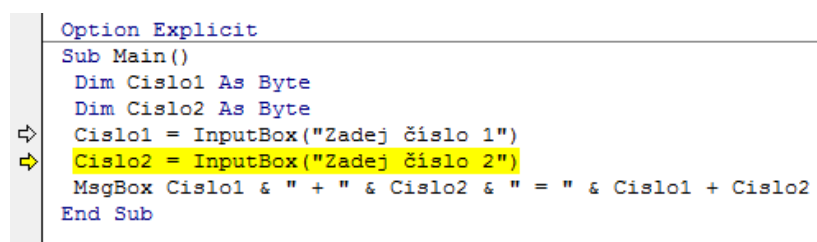
Podobného zrychlení je možné dosáhnout pomocí zárážek (breakpoint). Zárážka se vloží funkční klávesou F9 nebo klepnutím na šedou lištu před řádek, kde zárážka má být vložena. Přesun na zárážku se pak provádí příkazem **Continue** (funkční klávesa F5). Výhodou je, že zárážka na řádku zůstává, dokud se stejnou cestou nezruší (klepnutím na červený puntík zárážky nebo klávesou F9). Navíc je možné zárážky přidávat neomezeně kdykoliv, před i během ladění.



Obr. 60 Zárážka

Všechny dosud uvedené metody krokování se prováděly pouze směrem dopředu a vždy se vykonávaly všechny příkazy. Editor jazyka Visual Basic má ještě jednu zajímavou funkci, která umožňuje přesun v kódu bez vykonání příkazů mezi původní a novou pozicí. Metoda dokáže také přesun v algoritmu nazpět.

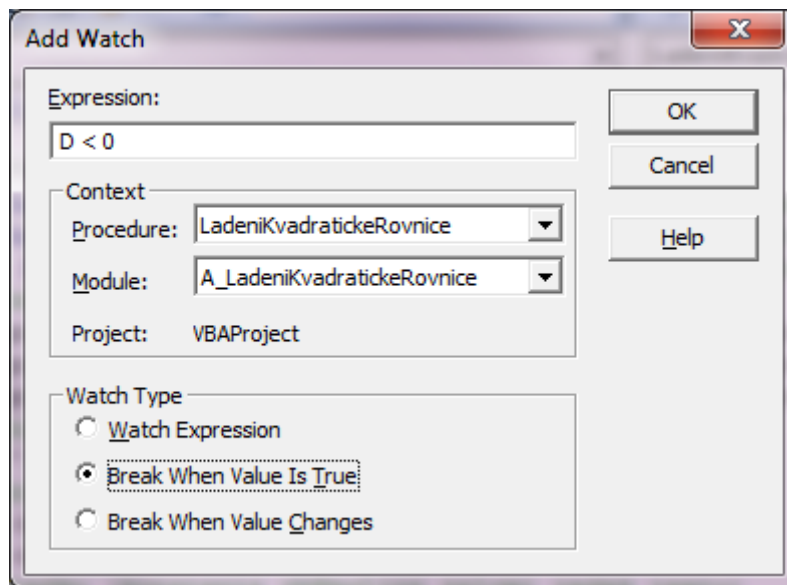
Princip je postavený na přesunu aktivního řádku tažením myši. Pomocí kurzoru myši se uchopí žlutá šipka aktivního řádku, která se nachází na levé liště modulu a tažením se přesune na požadované místo.



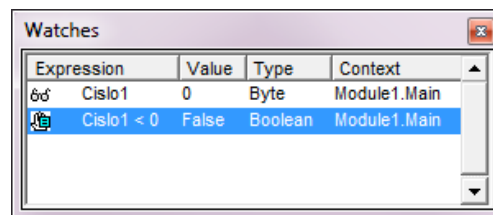
Obr. 61 Přetažením kurzoru o jeden příkaz zpět

Kromě pevných zářezek editor podporuje logické zářezky. Ty fungují tak, že přeruší program, jakmile se splní logická podmínka. Například u algoritmu výpočtu kořenů kvadratické rovnice je požadováno algoritmus automaticky pozastavit v případě, že hodnota D dosáhne záporných hodnot. Tohle efektivně zajistí logická zářezka.

Logické zářezky se realizují přes okno „Watches“. Pravým tlačítkem se vyvolá nabídka „Add Watch“ kde se do boxu „Expresion“ uvede podmínka přerušení algoritmu a současně se zvolí „Break When Value Is True“.



Obr. 62 Dialog Add Watch při vkládání logické zarážky



Obr. 63 Okno Watches s logickou zarážkou



Shrnutí pojmů

Laděná je účinná metoda odhalování chyb algoritmů.

Krokování je řízené provádění algoritmu příkaz po příkazu. Po každém kroku je program pozastaven a čeká na povel programátora k provedení dalšího kroku.

Zarážka (breakpoint) je značka v programovém kódu, která zajistí přerušení programu při dosažení příkazu, na kterém je zarážka aplikovaná.



Otázky

29. Jakými způsoby můžeme zahájit ladění programu?
30. Jakými způsoby zjistíme hodnoty proměnných?
31. Co je zarážka v programu a jak se vkládá?
32. Co se během ladění vykoná při stisku klávesy F8?
33. K čemu slouží okno Immediate?



Úlohy k řešení

Pro prostředí MS Excel si stáhněte dokument VB03.xlsx a proveďte následující úkon:

1. Laděním subrutiny **LadeniKvadratickeRovnice** analyzujte problém vzniku výjimky *"Invalid procedure call or argument"*, jenž vzniká při zadání kladných koeficientů A, B, C (např. 1, 2, 3). Vyzkoušejte si ladění při zadání koeficientů, jež odpovídají posledním třem cifrám vašeho osobního čísla, každé zvýšené o jedničku. Například pro BON007 bude A = 1, B = 1, C = 8. Proveďte kopii obrazovky (Alt+PrintScreen) přerušeno programu s aktivní zarážkou nad příkazem výpočtu diskriminantu a se zobrazeným aktivním oknem Watches, ve kterém bude proměnná D, výraz výpočtu hodnoty x1 a koeficienty A B C. *Postup:*
 - a. Otevřete makro (např. záložka "Zobrazení" tlačítko "Makra", v dialogu označit příslušné makro a stisknout tlačítko "Upravit").
 - b. Vyzkoušejte si makro při zadání tří kladných koeficientů a při vzniku výjimky zvolte tlačítko „Debug“.
 - c. Pomocí okna Locals a Watches analyzujte problémový řádek a zjistěte, která z proměnných výjimku vyvolá.
 - d. Ověřte si zjištěný problém - přes okno Locals změňte hodnotu problémové proměnné tak, aby výpočet hodnoty X1 negeneroval výjimku a pomocí krokování (klávesa F8) dokončete program.
 - e. Kliknutím na šedivou lištu (nebo klávesou F9) vložte zarážku (breakpoint) na řádku „ $D = b^2 - 4 * a * c$ “.

- f. Znovu spustě makro a vložte koeficienty A, B, C tak aby měli hodnoty odpovídající třem koncovým cifrám vašeho osobního čísla zvýšené o jedničku.*
- g. Po automatickém přerušení programu zobrazte si okno Watches a do něj vložte proměnou D, výraz $(-b + \text{Sqr}(D)) / (2 * a)$ a všechny tři koeficienty A, B, C.*

7 VĚTVENÍ A LOGICKÉ VÝRAZY



Čas ke studiu: 1,5 hodiny



Cíl Po prostudování této kapitoly budete umět

- Tvořit logické výrazy
- Používat logickou proměnnou
- Používat direktivu If-Else pro větvení algoritmů
- Používat direktivu Select-Case pro větvení algoritmů



Výklad

Život je plný rozhodování, bez slovíček když, pak, kde, kdy těžko popíšeme pracovní postup jakékoliv práce. Totéž pochopitelně platí pro algoritmy, kdy dělení činností podle rozhodování nazýváme Větvením programu. Ve visual Basicu máme k dispozici dvě konstrukce větvení If-Else a Select-Case. Obě konstrukce jsou však primárně závislé na logickém výrazu, kterým se tvoří základní otázka celého rozhodování.

7.1 Logické výrazy a proměnná Boolean

Výsledkem logického výrazu je jedna ze dvou možností

True (výraz je pravdivý – interně hodnota -1),

False (výraz je nepravdivý – interně hodnota 0).

Tuto hodnotu můžeme uchovávat v proměnné datového typu **Boolean**. Jelikož hodnota **False** je interně prezentována jako **0** a **True** jako **-1**, má každá nově založená proměnná **Boolean** hodnotu **False**.

Syntaxe:

Dim jméno_logicképroměnné **As Boolean**

jméno_logicképroměnné = **True**

Logický výraz můžeme tvořit pomocí relačních (porovnávacích) a logických operátorů.

Relační operátory

Relační nebo též porovnávací operátory jsou známé z matematiky. Rozdíl je pouze v symbolice zápisu a také ve faktu, že pomocí nich neděláme nerovnice, ale tvoříme výrazy, které se vyhodnotí jako pravda **True** nebo nepravda **False**.

Tabulka 5 – Relační operátory

Operátor	Matematická operace	Priorita
=	Rovnost	1
<>	Nerovnost	2
<	Menší než	3
>	Větší než	4
<=	Menší nebo rovno	5
>=	Větší nebo rovno	6
Is	Rovnost objektů	7

V tabulce jsou také uvedeny priority vyhodnocování operátorů. Pokud priorita nevyhovuje, můžeme ji změnit pomocí závorek.

Pozor Operátor = má dva významy:

přiřazení: **A = 10**

a porovnání: **if A = 10 then**

Příklady:

Dim Odpoved As Boolean

Dim Cislo As Integer

Cislo = 10

Odpoved = Cislo > 0 ' True

Odpoved = Cislo = 10 ' True

Odpoved = Cislo <> 0 ' True

Odpoved = Cislo = 0 ' False

Logické operátory

Logické operátory již pracují s logickými hodnotami, které se obvykle získají relačními operátory. V praxi se využívají tři logické operátory:

And – vrátí pravdu, pouze když jsou oba operandy pravdivé,

Or – vrátí pravdu, když je alespoň jeden operátor pravdivý,

Not – převrátí hodnotu z True na False a opačně.

Všechny možné kombinace výsledků tří základních logických operátorů vyjadřuje tabulka 6.

Tabulka 6 – Tabulka výsledků logických operací

	True	False	True	True	False	False
Not	False	True				
And	False		True		False	
Or	True		True		False	

Logických operátorů je ve Visual Basicu více, jejich použití ale není tak časté, přičemž se dají pomocí kombinace tří základních operátorů nahradit. Tabulka 7 uvádí seznam vše logických operátorů včetně jejich priorit. Za povšimnutí stojí to, že operátor **Not** má nejvyšší prioritu, pak následuje **And** a až pak **Or**.

Tabulka 7 – Logické operátory

Operátor	Matematická operace	Provede logickou	Priorita
Not	Negace (zápor)	Negaci výrazu	1
And	Konjunkce (sjednocení)	Průnik dvou výrazů	2
Or	Disjunkce (rozdíl)	Sjednocení dvou výrazů	3
Xor	Exkluzivní disjunkce	Vylučovací sjednocení dvou výrazů	4
Eqv	Logická rovnost	Porovnání shodnosti dvou výrazů	5
Imp	Logická implikace	Implikaci dvou výrazů	6

```
Option Explicit
Sub Main()
    Dim Hodnota As Integer
    Dim Odpoved As Boolean
    Hodnota = InputBox("Zadej hodnotu")

    Odpoved = Hodnota > 0
    MsgBox "Hodnota je kladna?" & Odpoved

    Odpoved = Hodnota <> 0
    MsgBox "Hodnota není nula?" & Odpoved

    Odpoved = Hodnota >= 1 And Hodnota <= 12
    MsgBox "Hodnota představuje číslo měsíce?" & Odpoved

    Odpoved = Hodnota = 7 Or Hodnota = 8
    MsgBox "Hodnota představuje číslo prázdninového měsíce? " & Odpoved

    Odpoved = Not (Hodnota = 7 Or Hodnota = 8 )
    MsgBox "Hodnota nepředstavuje číslo prázdninového měsíce?" & Odpoved
End Sub
```

Obr. 64 Příklad logických výrazů

7.2 Větvení pomocí funkce IIf

Pokud potřebujeme vypočítat hodnotu na základě jednoho ze dvou výrazů, mezi kterými se rozhodujeme logickým výrazem, můžeme použít funkci IIf.

Syntaxe:

IIf(logický výraz, VýrazProPravdu , VýrazProNepravdu)

Option Explicit

```

Sub LogickeVrazy()
MsgBox "Ukázka funkce makra ""LogickeVrazy""", vbExclamation, "Ukázka..."
Dim Hodnota As Integer
Dim Odpoved As Boolean
Hodnota = InputBox("Zadej číselnou hodnotu")

'Hodnota je kladná?
Odpoved = Hodnota > 0
MsgBox "Hodnota " & Hodnota & IIf(Odpoved, " je ", " není ") & "kladná."

'Hodnota je nula?
Odpoved = Hodnota = 0
MsgBox "Hodnota " & Hodnota & IIf(Odpoved, " je ", " není ") & "nula."

'Hodnota je lichá? (zbytek po celočíselném dělení není nula)
Odpoved = Hodnota Mod 2 <> 0
MsgBox "Hodnota " & Hodnota & IIf(Odpoved, " je ", " není ") & "lichá."

'Hodnota může představovat číslo kalendářního měsíce? (číslo je v intervalu <1;12>)
Odpoved = Hodnota >= 1 And Hodnota <= 12
MsgBox "Hodnota " & Hodnota & IIf(Odpoved, " může ", " nemůže ") & "představovat číslo kalendářního měsíce."

'Hodnota může představovat číslo kalendářního měsíce zimy 12,<1;3> ? Zima je od 21.12 do 21.3
Odpoved = Hodnota = 12 Or Hodnota >= 1 And Hodnota <= 3
MsgBox "Hodnota " & Hodnota & IIf(Odpoved, " může ", " nemůže ") & "představovat číslo kalendářního měsíce zimy."

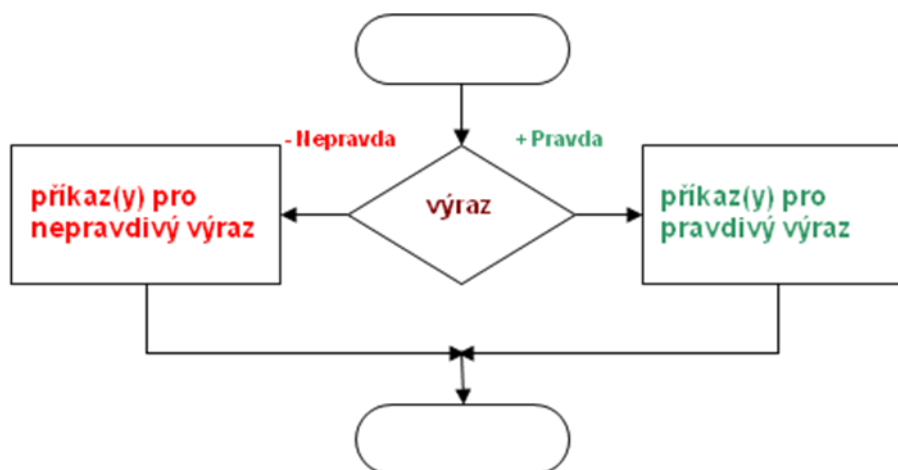
'Hodnota není shodná s numerickou koncovkou vašeho osobního čísla? Použijte operátor Not
Odpoved = Not Hodnota = 7
MsgBox "Hodnota " & Hodnota & IIf(Odpoved, " není ", " je ") & "shodná s numerickou koncovkou mého osobního čísla."
End Sub

```

Obr. 65 Příklad logických výrazů a využití funkce IIF

7.3 Větvení pomocí konstrukce If-Else

Rozhodovací příkaz určený k možnosti větvení programu do dvou větví. Také je možno použít pro ukončování nebo pozastavování programu a obsluhování chybových stavů.



Obr. 66 Algoritmus větvení If-Else ve vývojovém diagramu

Jednořádková syntaxe:**If podmínka Then příkaz1 [Else příkaz2]**

K podmíněnému provedení příkazu nebo bloků příkazů slouží příkaz *If*. Pokud je podmínka splněna provede se příkaz1 za klíčovým slovem *Then*. Pokud není splněna, provede se příkaz2 za klíčovým slovem *Else*. Když není klíčové slovo *Else* napsáno, neprovede se nic.

Víceřádková syntaxe:**If podmínka Then****[příkaz1]****[ElseIf podmínka Then****[příkaz2]****[Else****[příkaz_n]]****End If**

Druhá forma použití je složitější, ale dovoluje větvit program do více bloků. Pokud je podmínka splněna, provede se příkaz1 a přejde se na řádek za příkazem *End If*. Pokud není splněna, testují se všechny podmínky za klíčovým slovem *ElseIf* do doby splnění podmínky. Pokud nebyla splněna podmínka v žádné větvi, program přechází na klíčové slovo *Else* a provede *příkaz_n*.

Následující algoritmus na obr. 67 pomocí větvení určuje cenu vstupenky na přestavení na základě čísla řady sedačky.

```
Sub Main()  
    Dim rada As Integer ' Cena vstupenky podle požadované řady sedadel  
    Dim cena As Currency ' Určená cena  
    rada = InputBox("Do které řady chcete vstupenku ?", "Cena vstupenky")  
    ' pomocí And a Or lze vytvořit podmínku, která vrací True pro více intervalů  
    If (rada >= 1 And rada <= 5) Or (rada > 20 And rada <= 25) Then  
        'Pro interval 1..5 a 20..25 je cena 150 Kč  
        cena = 150  
    ElseIf rada >= 6 And rada <= 15 Then  
        'Pro interval 6..15 je cena 180 Kč  
        cena = 180  
    ElseIf rada >= 16 And rada <= 20 Then  
        'Pro interval 16..20 je cena 200 Kč  
        cena = 200  
    Else  
        ' ve skutečnosti jde o zbytečný příkaz, jelikož každá  
        ' proměnná ve VB je implicitně nulovaná  
        cena = 0  
    End If  
    'Je-li cena nulová, uživatel zadal neplatnou řadu  
    If (cena > 0) Then  
        MsgBox "Cena lístků je: " & Format(cena, "Currency"), , "Odpověď..."  
    Else  
        MsgBox " Řada " & rada & " neexistuje !!!", vbExclamation, "CHYBA !!!"  
    End If  
End Sub
```

Obr. 67 Příklad větvení If-Else ve Visual Basicu

Stejného výsledku také dosáhneme postupným vnořování dílčích konstrukcí if-Else (viz příklad na obr. 68). Příkaz `Exit Sub` okamžitě ukončí subrutinu.

```
Sub Main()  
    Dim rada As Integer ' Cena vstupenky podle požadované řady sedadel  
    Dim cena As Currency ' Určená cena  
    rada = InputBox("Do které řady chcete vstupenku ?", "Cena vstupenky")  
    ' pomocí And a Or lze vytvořit podmínku, která vrací True pro více intervalů  
    If (rada >= 1 And rada <= 5) Or (rada > 20 And rada <= 25) Then  
        'Pro interval 1..5 a 20..25 je cena 150 Kč  
        cena = 150  
    Else  
        If rada >= 6 And rada <= 15 Then  
            'Pro interval 6..15 je cena 180 Kč  
            cena = 180  
        Else  
            If rada >= 16 And rada <= 20 Then  
                'Pro interval 16..20 je cena 200 Kč  
                cena = 200  
            Else  
                MsgBox " Řada " & rada & " neexistuje !!!", vbExclamation, "CHYBA"  
                Exit Sub  
            End If  
        End If  
    End If  
End If  
    MsgBox "Cena lístků je: " & Format(cena, "Currency"), , "Odpověď..."  
End Sub
```

Obr. 68 Alternativa příkladu větvení If-Else ve Visual Basicu

7.4 Větvení pomocí konstrukce Select-Case

Podobný rozhodovací příkaz jako předešlý, použití nebo výběr z těchto dvou záleží pouze na zkušenosti člověka, který vytváří program. Konstrukce začíná slovem **Select** pokračuje slovem **Case** a za ním se uvede hodnota nebo interval pro který bude platit postup následující za hledanou hodnotou nebo intervalem. Větvení programu probíhá v závislosti na hodnotě numerického nebo řetězcového výrazu.

Syntaxe:

Select Case výraz

[Case seznam_výrazů1

 [příkazy1]]

[Case seznam_výrazů2

 [příkazy2]]

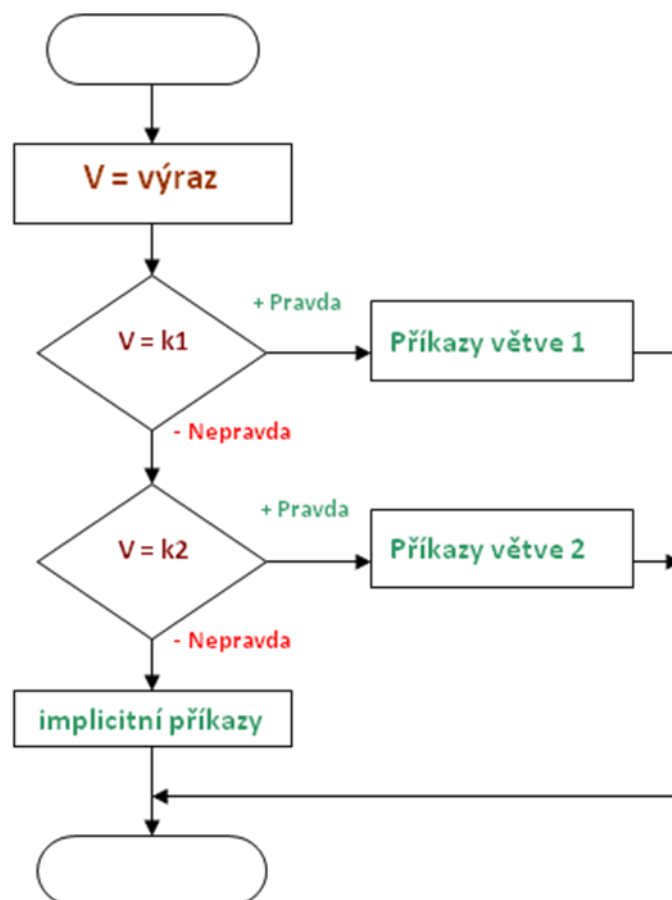
...

[Case Else

 [příkaz_n]]

End Select

Za klíčovým slovem **Case** se testuje, zda v seznamu_výrazů je hledaný výsledek následující za příkazem **Select Case**. Pokud ano, provede se blok příkazy následující za slovem **Case**. Při nenalezení shody program přechází na klíčové slovo **Case Else** a provede příkaz_n.



Obr. 69 Algoritmus větvení Select-Case ve vývojovém diagramu

```
Sub Main()  
    Dim rada As Integer ' Cena vstupenky podle požadované řady sedadel  
    Dim cena As Currency ' Určená cena  
    rada = InputBox("Do které řady chcete vstupenku ?", "Cena vstupenky")  
    ' pomocí And a Or lze vytvořit podmínku, která vrací True pro více intervalů  
    Select Case rada  
    Case 1 To 5, 20 To 25  
        'Pro interval 1..5 a 20..25 je cena 150 Kč  
        cena = 150  
    Case 6 To 15  
        'Pro interval 6..15 je cena 180 Kč  
        cena = 180  
    Case 16 To 20  
        'Pro interval 16..20 je cena 200 Kč  
        cena = 200  
    Case Else  
        ' ve skutečnosti jde o zbytečný příkaz, jelikož každá  
        ' proměnná ve VB je implicitně nulovaná  
        cena = 0  
    End Select  
    'Je-li cena nulová, uživatel zadal neplatnou řadu  
    If (cena > 0) Then  
        MsgBox "Cena lístků je: " & Format(cena, "Currency"), , "Odpověď..."  
    Else  
        MsgBox " Řada " & rada & " neexistuje !!!", vbExclamation, "CHYBA !!!"  
    End If  
End Sub
```

Obr. 70 Příklad větvení Select-Case ve Visual Basicu



Shrnutí pojmů

Logický výraz je výraz, jehož výsledkem je logická hodnota True/False.

Relační operátory jsou porovnávací operátory, pomocí nichž se tvoří logické výrazy.

Logické operátory jsou operátory, kterými tvoříme složité logické výrazy. V praxi vystačíme s operátory Not, And, Or. Výsledek je opět log

Větvení programů je programová konstrukce, která na základě rozhodnutí logického výrazu vybere činnost.



Otázky

34. Jaké známe relační operátory ve Visual Basicu

35. Pro $C=10$, jaký bude výsledek výrazu $C \geq 1 \text{ AND } C < 10$?

36. Kde je vhodnější použít větvení If-Else než Select Case

37. K čemu slouží příkaz Exit Sub?



Úlohy k řešení

Pro prostředí MS Excel si stáhněte dokument VB03.xlsx a proveďte následující úkony:

1. Upravte subrutinu (makro) **LogickeVrazy** tak, aby všechna tvrzení odpovídala logice.

Postup:

- a. Otevřete makro
 - b. Doplněte jednotlivé logické výrazy např. $\text{Odpoved} = \text{Hodnota} > 0$.
2. Doplněte všechny úrovně větvení subrutiny (makra) **CenaBarevnehoTiskulf** tak, aby výsledná cena tisku byla vypočtena na základě následujících pravidel:

- do 10 stran ... 10,00 Kč / stranu,
- od 10 stran ... 9,00 Kč / stranu,
- od 100 stran ... 7,00 Kč / stranu,
- od 500 stran ... 6,00 Kč / stranu,
- nad 10.000 stran ... 0,60 Kč / stranu.

Postup:

- a. Otevřete subrutinu.
 - b. Přidejte vnořené ElseIf podmínky pro další případy počtu tištěných stran.
 - c. Pro každý interval doplňte výpočet ceny.
 - d. Ověřte funkci subrutiny.
3. Doplněte subrutinu **CenaBarevnehoTiskuCase** o větvení direktivou Select-case tak, aby byl výsledek shodný se subrutinou CenaBarevnehoTiskulf.

Postup:

- a. Otevřete subrutinu.
- b. Přidejte direktivu Select-Case.

```

Select Case PocetStran
Case 1 To 9
    Cena = PocetStran * 10
    ...
End Select

```

- c. Definujte všechny případy větvení.
- d. Ověřte funkci subrutiny.



Řešený příklad

Doplňte algoritmus řešení kvadratické rovnice i o určení imaginárních kořenů.

V tomto případě se do programového kódu doplní větvení If-Else které na základě hodnoty diskriminantu rozhodne, zda se mají počítat reálné nebo imaginární kořeny

```

'Doplnění programu o řešení v imaginární oblasti kořenů
Sub Main()
    Dim D As Double                                     'Diskriminant
    Dim a As Double, b As Double, c As Double          'Koefficienty kvadratické rovnice
    'Získání koeficientu rovnice
    a = InputBox("Zadej A:", "Ax^2 + Bx + C")
    b = InputBox("Zadej B:", "Ax^2 + Bx + C")
    c = InputBox("Zadej C:", "Ax^2 + Bx + C")
    'Výpočet diskriminantu
    D = b ^ 2 - 4 * a * c
    'Výpočet kořenu rovnice
    If D < 0 Then
        'Kořeny jsou imaginární
        Dim Re As Double, Im As Double                 'Hledané řešení
        Re = -b / (2 * a)
        Im = (Sqr(Abs(D))) / (2 * a)
        'Výpis výsledku
        MsgBox Re & "±" & Im & "i", , _
            "Řešení rovnice: " & a & "x^2 + " & b & "x + " & c
    Else
        'Kořeny jsou reálné
        Dim x1 As Double, x2 As Double                 'Hledané řešení
        x1 = (-b + Sqr(D)) / (2 * a)
        x2 = (-b - D ^ (1 / 2)) / (2 * a)
        'Výpis výsledku
        MsgBox "x1 = " & x1 & vbCrLf & "x2 = " & x2, , _
            "Řešení rovnice: " & a & "x^2 + " & b & "x + " & c
    End If
End Sub

```

8 CYKLY - OPAKOVÁNÍ ČINNOSTÍ



Čas ke studiu: 2 hodiny



Cíl Po prostudování této kapitoly budete umět

- Tvořit inkrementační cyklus For
- Vytvářet všechny tři varianty logického cyklu
- Určit vhodný typ varianty cyklu
- Procházet kolekcí objektů cyklem For Each



Výklad

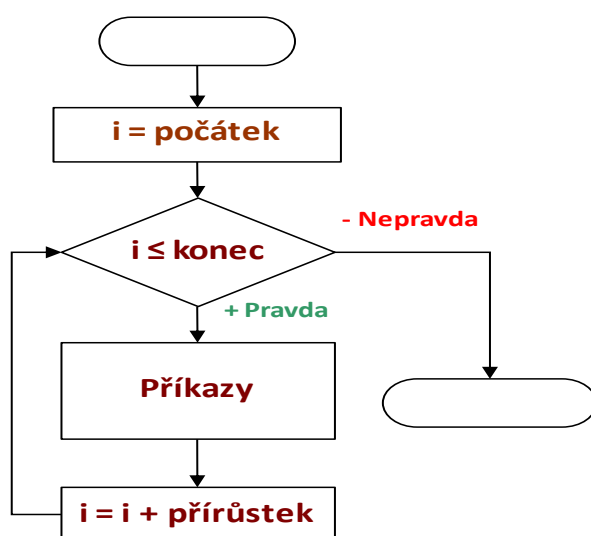
Pod pojmem cyklus v souvislosti s programováním rozumíme řídicí strukturu (direktivu) programovacího jazyka, která zajišťuje opakování zvolené sekvence příkazů.

Rozlišujeme hned několik typů cyklů:

- Inkrementační (přírůstkový) cyklus (**For - Next**).
- Logické cykly (**Do - Loop**):
 - s podmínkou na začátku,
 - s podmínkou na konci
 - a s podmínkou uprostřed.
- Cyklus procházení kolekcí (**For Each - Next**).

8.1 Inkrementační cyklus For – Next

Cyklus, který je předem známý počtem opakování. Zvaný taky přírůstkový cyklus. Ukončuje se slovem Next. Nastavovat můžeme krok, s jakým se nám bude měnit velikost přírůstku. Pokud je možné, že obdržíme výsledek (například při hledání textu) dříve než proběhne předem nastavený počet opakování, můžeme okamžitě ukončit cyklus příkazem **Exit For**.



Obr. 71 Algoritmus přírůstkového cyklu For-Next ve vývojovém diagramu

Syntaxe:

```

For počítadlo = počátek To konec [Step přírůstek]
    [příkazy]
    [If podmínka Then Exit For]    'tělo cyklu
    [příkazy]
Next [počítadlo]
  
```

- Cyklus opakuje příkazy tolikrát, kolikrát se vejde přírůstek do intervalu <počátek;konec>.
- Přírůstek může být kladný i záporný, implicitně je roven jedné. Není vhodné používat reálný typ.
- Počítadlo se často v těle cyklu využívá pozor však na modifikaci počítadla.
- Počátek, konec, přírůstek může být proměnná.
- Exit For ihned ukončí cyklus.



Obr. 72 Cyklu For-Next v praxi

Jako příklad využití cyklu For Next se následující algoritmem (viz obr. 73) provede vyčíslení faktoriálu čísla 5! = (((5*4)*3)*2)*1.

```
Sub Main()  
    Dim Cislo As Integer      'Číselná hodnota pro výpočet faktoriálu  
    Dim i As Integer         'Iterační pomocná proměnná  
    Dim Faktorial As Double  'Výsledek výpočtu faktoriálu  
    'Nactení hodnoty pro výpočet faktoriálu  
    Cislo = InputBox("Zadej číslo:", "Výpočet Faktoriálu")  
    'Vlastní výpočet  
    Faktorial = 1 '0! = 1  
    For i = 2 To Cislo  
        Faktorial = i * Faktorial  
    Next  
    'Výpis výsledku      MsgBox Cislo & "! = " & Faktorial, , "Výpočet  
Faktoriálu"  
End Sub
```

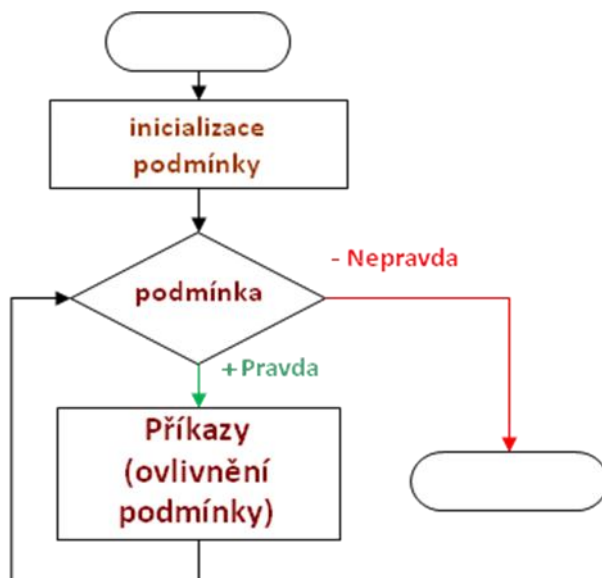
Obr. 73 Využití cyklu For-Next k výpočtu faktoriálu čísla

Jelikož na pořadí násobení nezáleží, je možné algoritmus přepsat do následující podoby (obr. 74).

```
Sub Main()  
    Dim Cislo As Integer      'Číselná hodnota pro výpočet faktoriálu  
    Dim i As Integer         'Iterační pomocná proměnná  
    Dim Faktorial As Double  'Výsledek výpočtu faktoriálu  
    'Nactení hodnoty pro výpočet faktoriálu  
    Cislo = InputBox("Zadej číslo:", "Výpočet Faktoriálu")  
    'Vlastní výpočet  
    Faktorial = 1 '0! = 1  
    For i = 2 To Cislo  
        Faktorial = i * Faktorial  
    Next  
    'Výpis výsledku      MsgBox Cislo & "! = " & Faktorial, , "Výpočet Faktoriálu"  
End Sub
```

Obr. 74 Alternativa cyklu For-Next k výpočtu faktoriálu čísla

8.2 Logický cyklus Do-Loop s podmínkou na začátku



Obr. 75 Algoritmus cyklu Do-Loop s podmínkou na začátku ve vývojovém diagramu

Syntaxe:

[příkazy inicializující podmínku]

Do While **Podmínka**

[příkazy]

[If StopPodmínka Then Exit Do] 'tělo cyklu

[příkazy]

Loop

- Cyklus opakuje příkazy tak dlouho, dokud je podmínka pravdivá.
- Podmínka je tvořena logickým výrazem.
- Je-li podmínka na počátku nepravdivá cyklus se nevykoná ani jednou.

- Před cyklem se musí inicializovat podmínka a v těle cyklu se musí podmínka ovlivňovat, v opačném případě hrozí nekonečný cyklus.
- **Exit** Do ihned ukončí cyklus



Obr. 76 Logický cyklus Do-Loop s podmínkou na začátku v praxi

Jako příklad si uveďme algoritmus postupného sčítání neznámého počtu hodnot. Program se bude neustále dotazovat na hodnoty, dokud uživatel nezadá nulu, která algoritmus ukončí s oznámením celkového součtu hodnot.

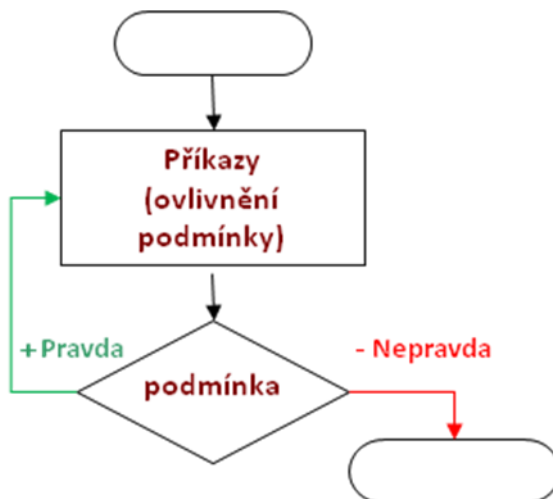
```
Sub Main()
Dim Cislo As Long 'Aktuálně zadané číslo
Dim Soucet As Long 'Celkový součet
Cislo = InputBox("Zadej číslo:", "Sčítání...", 0)
Do While Cislo <> 0 'Podmínka na začátku
    Soucet = Soucet + Cislo
    Cislo = InputBox("Zadej číslo:", "Sčítání...", 0)
Loop
MsgBox "Součet všech zadaných čísel je: " & Soucet, , "Sčítání..."
End Sub
```

Obr. 77 Příklad logického cyklu s podmínkou na začátku

Z příkladu vyplývá jedna nevýhoda cyklu s podmínkou na začátku. To je nutnost provádět příkazy ovlivňující podmínku před i uvnitř cyklu. Konkrétně volání funkce InputBox se provádí ve dvou místech algoritmu.

Problém opakovaného příkazu se za určitých podmínek daří eliminovat cyklům s podmínkou na konci.

8.3 Logický cyklus Do-Loop s podmínkou na konci



Obr. 78 Algoritmus logického cyklu Do-Loop s podmínkou na konci ve vývojovém diagramu

Syntaxe:

Do

[příkazy]

[If StopPodmínka Then Exit Do] 'tělo cyklu

[příkazy]

Loop While Podmínka

- Cyklus opakuje příkazy tak dlouho, dokud je Podmínka pravdivá.
- Oproti cyklu s podmínkou na začátku je obvykle kratší – ovlivnění podmínky je pouze v těle cyklu.
- Cyklus se vždy vykoná aspoň jednou. Tzn. že se tělo jednou vykoná, aniž by byla podmínka splněna (omezené použití).
- **Exit Do** ihned ukončí cyklus.

Jako příklad si opět uveďme algoritmus postupného sčítání neznámého počtu hodnot, tentokrát za použití cyklu s podmínkou na konci.

```
Sub Main()  
Dim Cislo As Long    'Aktuálně zadané číslo  
Dim Soucet As Long   'Celkový součet  
  
Do  
    Cislo = InputBox("Zadej číslo:", "Sčítání...", 0)  
    Soucet = Soucet + Cislo  
Loop While Cislo <> 0 'Podmínka na konci  
MsgBox "Součet všech zadaných čísel je: " & Soucet, , "Sčítání..."  
End Sub
```

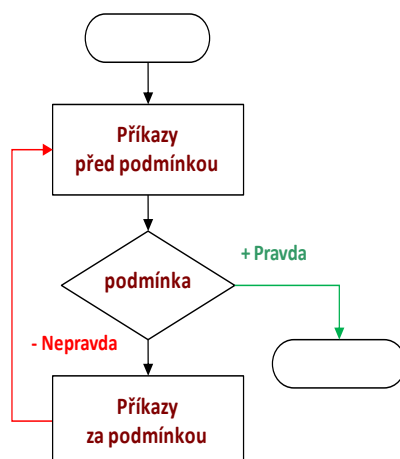
Obr. 79 Příklad logického cyklu s podmínkou na začátku

Z příkladu je patrné zkrácení algoritmu vůči variantě s podmínkou na začátku. Na druhou stranu je nevýhoda tohoto cyklu v tom, že se operace přičtení hodnoty do součtu provede jednou zbytečně (přičte se nula). V uvedené realizaci to sice nevádí, ale pokud bychom změnili podmínku pro ukončení tak, že se ukončení provede po zadání i záporné hodnoty, algoritmus by nepracoval správně. Tato vlastnost (provedení operace s neplatnými daty) je pro cyklus s podmínkou typická, díky čemuž není možné tento typ cyklu libovolně aplikovat.

Uvedené problémy cyklů s podmínkou na začátku a konci se dají elegantně eliminovat cyklem s podmínkou uprostřed.

8.4 Logický cyklus Do-Loop s podmínkou uprostřed

Ve své podstatě jde o nekonečný cyklus s příkazem ukončení, který se provede na základě větvení programu obvykle konstrukcí If-Else.



Obr. 80 Algoritmus cyklu Do-Loop s podmínkou uprostřed ve vývojovém diagramu

Syntaxe:

Do

[příkazy před podmínkou]

If Podmínka Then Exit Do

[příkazy za podmínkou]

Loop

- Cyklus opakuje příkazy tak dlouho, dokud je Podmínka nepravdivá.
- Oproti cyklu s podmínkou na začátku je zde blok příkazů, který se provede vždy alespoň jednou. V této části musí být operace, které Podmínku ovlivňují.
- Příkaz **Exit Do** ihned ukončí cyklus.

Znovu si cyklus předvedeme na příkladu algoritmu postupného sčítání neznámého počtu hodnot.

```
Sub Main()  
Dim Cislo As Long 'Aktuálně zadané číslo  
Dim Soucet As Long 'Celkový součet  
Do  
    Cislo = InputBox("Zadej číslo:", "Sčítání...", 0)  
    If Cislo = 0 Then Exit Do 'Podmínka uprostřed  
    Soucet = Soucet + Cislo  
Loop  
MsgBox "Součet všech zadaných čísel je: " & Soucet, , "Sčítání..."  
End Sub
```

Obr. 81 Příklad logického cyklu s podmínkou uprostřed

8.5 Cyklus procházení kolekcí

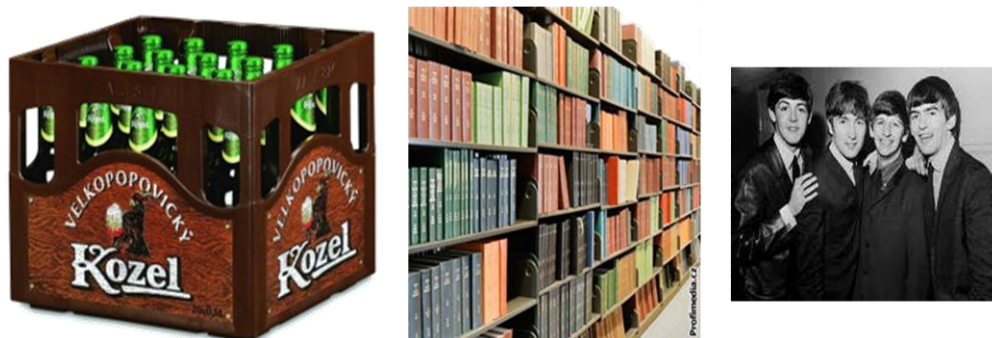
Pod pojmem kolekce v programovacím jazyce rozumíme skupinu shodných nebo příbuzných objektů tvořící celek. Prvek (element, člen) kolekce je právě jeden dílčí objekt kolekce.

Příklady reálných kolekcí:

- Přepravka (elementy: Láhve).
- Vlák (elementy: Vagóny).
- Čtvrť (elementy: Domy).
- Knihovna (elementy: Knihy).
- Studijní skupina (elementy: Studenti).

Kolekce v aplikaci MS Excelu:

- Workbooks (Workbook - Otevřené soubory).
- WorkSheets (WorkSheet – Listy souboru).
- Cells (Range – buňky listu).
- Selection (Range – vybrané buňky).



Obr. 82 Příklad reálných kolekcí objektů

Protože kolekce jsou velmi často používána, je pro procházení jejich prvků speciální konstrukce cyklu For-Each-Next.

Syntaxe:

```
Dim Element As TypElementuKolekce
```

```
For Each Element in Kolekce
```

```
    [příkazy]
```

```
    [Zpracování elementu]
```

```
    [If podmínka Then Exit For]
```

```
Next
```

- Cyklus postupně projde všechny prvky kolekce a dočasně je přiřadí do proměnné Element.
- Element musí být předem deklarován datovým typem shodným s prvky elementu

Jak na to?

```
Dim Láhev As LáhvoVéPivo
For Each Láhev In Přepravka
    Pít(Láhev)
Next
```



Obr. 83 Cyklus For-Each-Next v praxi

Jako příklad si uveďme algoritmus, který změní u všech uživatele vybraných buněk barvu pozadí na červenou. Připomínám, že objekt `Selection` představuje kolekci všech vybraných buněk.

```
Sub Main()
    Dim Bunka As Range 'Deklarace pomocné proměnné
    For Each Bunka In Selection
        Bunka.Interior.Color = vbRed
    Next
End Sub
```

Obr. 84 Procházení vybrané oblasti buněk



Řešený příklad

Vytvořte algoritmus, který určí, zda zadané číslo je nebo není prvočíslem.

```

Sub Main()
    Dim Cislo As Long           'Zkoumané Číslo
    Dim Odpoved As Boolean      'Zjištění
    Cislo = InputBox("Zadej číslo:", "Prvočíslo?")
    Odpoved = True 'Předpokládám že číslo je prvočíslem
    Dim i As Integer
    For i = 2 To Sqr(Cislo)
        If (Cislo Mod i) = 0 Then 'Je-li dělitelné beze zbytku pak
            Odpoved = False      'NENÍ prvočíslo
            Exit For 'Nemá smysl dál pokračovat
        End If
    Next
    If Odpoved Then
        MsgBox "Číslo " & Cislo & " JE prvočíslo.", , "Řešení:"
    Else
        MsgBox "Číslo " & Cislo & " NENÍ prvočíslo.", , "Řešení:"
    End If
End Sub

```



Shrnutí pojmů

Cyklus je řídicí strukturu (konstrukce) programovacího jazyka, která zajišťuje opakování zvolené sekvence příkazů.

Iterační cyklus je typ cyklu, u kterého se počet opakování stanovuje výpočtem. Během každé iterace se inkrementuje proměnná o pevně stanovený přírůstek.

Kolekce je skupinu shodných nebo příbuzných objektů tvořící celek.

Prvek kolekce (element, člen) je právě jeden dílčí objekt kolekce.

Procházení kolekci je cyklická operace, která projde všechny objekty kolekce.



Otázky

38. Kolikrát se provede cyklus `For i = 1 To 10 Step -1` ?

39. K čemu slouží příkaz `Exit For`?

- 40. Jakou hodnotu musí mít výraz za While pro ukončení?
- 41. Kdy nelze aplikovat cyklus s podmínkou na konci?
- 42. Jaká je nevýhoda cyklu s podmínkou na začátku?
- 43. Kdy nelze aplikovat cyklus s podmínkou uprostřed?
- 44. Kdy se používá cyklus For Each ?

9 PROGRAMOVÉ PŘÍSTUPY K BUŇKÁM



Čas ke studiu: 1 hodiny



Cíl Po prostudování této kapitoly budete umět

- Programově vybírat buňky a oblasti buněk dokumentu MS Excelu
- Programově přistupovat a editovat obsahy buněk
- Procházet oblasti



Výklad

V předchozí kapitole probrané cykly jsou často využívány v prostředí MS Excelu k procházení oblastí. Jedna z možností již byla uvedena jako příklad použití cyklu For-Each-Next, kdy byla procházena oblast uživatelem vybraných buněk. Ne vždy je ale takovýto postu možný. Dříve, než se předvedou jiné možnosti procházení, je nezbytné se seznámit s metodou výběru buněk a jejich editací.

9.1 Programový výběr buněk a oblastí

Pro práci s buňkami je potřeba znát způsob jejich výběru.

ActiveCell

- aktuální buňka, nebo referenční buňka oblasti.

Selection

- uživatelem vybraná oblast buněk (je-li vybraná pouze jedna buňka je shodný s **ActiveCell**).

Range("C4")

- výběr buňky podle adresy. Výběr se uskuteční na aktuálním listu.

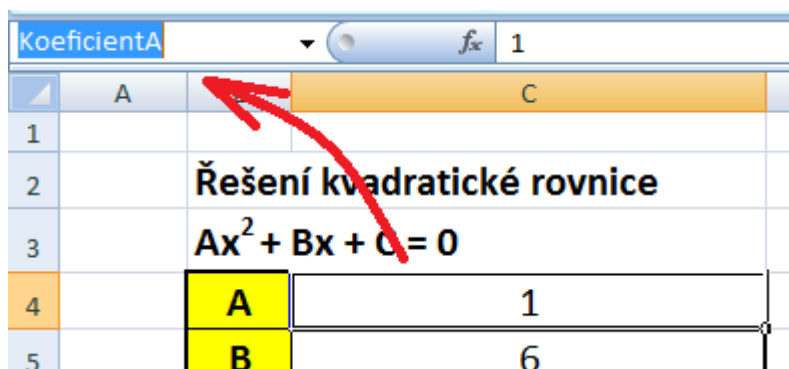
ActiveCell.Offset(r,s) -výběr buňky relativně k aktuální. Hodnoty **r**, **s** představují posuny v řádku a sloupci vůči aktuální pozici. Pro kladné hodnoty **r** je posun v řádcích dolů a pro záporné nahoru. Pro posun ve sloupcích kladné **s** posouvá doprava a záporné doleva. Nulové hodnoty znamenají bez posunu.

Selection.Offset(r,s) - výběr oblasti relativně k aktuální. Tvar a rozsah původní oblasti bude zachována.

Range("Nazev") - výběr pojmenované buňky nebo oblasti. Pojmenování se provádí v prostředí MS Excelu buď v poli názvu, nebo položkou „definovat název“ na kartě „vzorce“ (viz obr. 85). Pojmenování je jedinečné pro celý dokument.

Range("C4:E7") - výběr souvislé oblasti podle adresy - výběr se provede na aktuálním listu.

Range("A1,C4:E7,F2:G5") - výběr nesouvislé oblasti podle adresy - výběr se provede na aktuálním listu.



Obr. 85 Pojmenování oblastí buněk

```
Sub VyberBunky()  
    'Podle adresy  
    Range("C4").Select  
    MsgBox "Podle adresy", , "Range(""C4"") "  
    'Podle názvu (buňka musí být pojmenována)  
    Range("Bunka").Select  
    MsgBox "Podle názvu", , "Range(""Bunka"") "  
    'Relativně vůči aktuální (sousední vpravo)  
    ActiveCell.Offset(0, 1).Select  
    MsgBox "Sousední vpravo", , "ActiveCell.Offset(0, 1) "  
    'Relativně vůči aktuální (sousední vlevo)  
    ActiveCell.Offset(0, -1).Select  
    MsgBox "Sousední vlevo", , "ActiveCell.Offset(0, -1) "  
    'Relativně vůči aktuální (sousední o dvě nahoru)  
    ActiveCell.Offset(-2, 0).Select  
    MsgBox "Sousední o dvě nahoru", , "ActiveCell.Offset(-2, 0) "  
    'Relativně vůči aktuální (sousední o tři dolů)  
    ActiveCell.Offset(3, 0).Select  
    'Podle adresy oblasti  
    Range("C4:E7").Select  
    MsgBox "Podle adresy oblasti", , "Range(""C4:E7"") "  
    'Podle názvu oblasti (oblast musí být pojmenována)  
    Range("Tabulka").Select  
    MsgBox "Podle názvu oblasti", , "Range(""Tabulka"") "  
End Sub
```

Obr. 86 Metody výběru buněk

9.2 Programový přístup k hodnotám buněk a jejich editace

Hodnotu buňky je možné získat či zapsat v různých formátech.

ActiveCell.Value - čtení výsledku vzorce nebo zápis hodnoty.

ActiveCell.Formula - čtení a zápis vzorce v přirozených adresách buněk.

ActiveCell.FormulaR1C1 - čtení a zápis vzorce ve formátu RC. Odkazy na buňky jsou zadány relativně formátem **RrCs**. Hodnoty **r**, **s** představují relativní posuny v řádku a sloupci vůči pozici vzorce. Pro kladné hodnoty **r** je posun v řádcích dolů a pro záporné nahoru. Pro posun ve sloupcích kladné **s** posouvá doprava a záporné doleva. Jeli adresovaná buňka ve stejném sloupci nebo řádku pak se hodnoty **r**, **s** vynechávají.

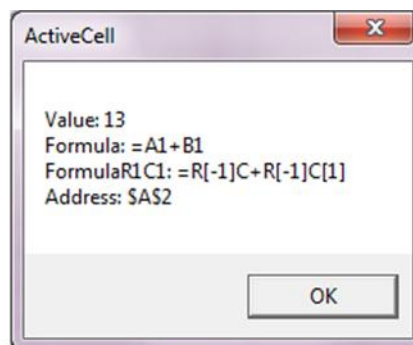
ActiveCell.Address - čtení adresy buňky.

Selection.Address - čtení adresy oblasti.

ActiveCell.Name - čtení názvu buňky/oblasti, zápis (přiřazení) nového názvu.

Selection.Name

	A2				
	A	B	C	D	
1	10	3			
2	13				



```
Sub Main()
    MsgBox "Value: " & ActiveCell.Value & vbCrLf & _
        "Formula: " & ActiveCell.Formula & vbCrLf & _
        "FormulaR1C1: " & ActiveCell.FormulaR1C1 & vbCrLf & _
        "Address: " & ActiveCell.Address, , "ActiveCell"
End Sub
```

Obr. 87 Metody přístupu k hodnotám buňky

9.3 Procházení oblastí

K obecnému procházení buněk se nejčastěji používá logický cyklus s podmínkou na začátku. Podmínka bývá nastavena tak, aby cyklus prošel souvislou neprázdnou oblast buněk. Jelikož hodnota buňky je datového typu **Variant**, nejvhodnější je podmínka **Value <> Empty**.

V každé iteraci cyklu se mění pracovní buňka buď přímým výběrem **ActiveCell.Offset(1,0).Select**, nebo se mění proměnná poukazující na pracovní buňku **Set Bunka = Bunka.Offset(0,1)**. Druhá varianta je rychlejší a umožňuje procházet vícero oblastí najednou (například při porovnávání hodnot dvou tabulek).

Syntaxe procházení neprázdných buněk ve sloupci přímým výběrem

```
Do While ActiveCell.Value <> Empty
    [příkazy]
    ActiveCell.Offset(1,0).Select
Loop
```

Syntaxe procházení neprázdných buněk v řádku bez změny aktivní buňky

```
Dim Bunka As Range 'Deklarace pomocné proměnné
Set Bunka = ActiveCell 'Přiřazení aktivní buňky
Do While Bunka.Value <> Empty
    [příkazy]
    Set Bunka = Bunka.Offset(0,1)
Loop
```

Jiný případem je situace, když uživatel předem označí buňky oblasti. Pak je nejvhodnější metoda procházení pomocí cyklu For Each – Next.

Syntaxe procházení vybrané oblasti buněk

```
Dim Bunka As Range 'Deklarace pomocné proměnné
For Each Bunka In Selection
    [příkazy]
Next
```

```

Sub OznacitHodnotySloupceNadLimit()
Dim Limit As Double
Limit = InputBox("Označit buňky s hodnotou >= než...", , 1000)

Do While ActiveCell.Value <> Empty 'Dokud není buňka prázdná
    If ActiveCell.Value >= Limit Then
        ActiveCell.Interior.Color = vbRed
    End If
    ActiveCell.Offset(1, 0).Select 'Přesun na další buňku
Loop
End Sub

```

Obr. 88 Procházení oblasti buněk přímým výběrem

**Řešený příklad**

Vytvořte algoritmus, který ve sloupci importovaných hodnot převede načtená data na čísla. Data představují numerické hodnoty reálných čísel, které však mají oddělovač desetinnou tečku. Excel hodnoty mylně identifikuje jako texty. Cílem je nahradit tečku za desetinnou čárku. Počet hodnot dopředu není znám – uživatel vybere pouze počáteční buňku sloupce.

```

Sub PrevestSloupecNaCislo()
Dim Format As String
Format = InputBox("Formát čísel", _
    "Převod textů na čísla se záměnou "." za ",", "", "0.00")
Do While ActiveCell.Value <> Empty
    ' Funkce Replace provede záměnu tečky za čárku,
    ' IsNumeric vrací pravdu, když text reprezentuje číslo.
    If IsNumeric(Replace(ActiveCell.Value, ".", ",")) Then
        ' Funkce CDBl převádí text na číslo typu Double
        ActiveCell.Value = CDBl(Replace(ActiveCell.Text, ".", ","))
        ActiveCell.NumberFormat = Format ' Nastavení formátu čísla
    End If
    ActiveCell.Offset(1, 0).Select 'Přesun na další buňku
Loop
End Sub

```

**Otázky**

45. Jakou buňku vybere příkaz Range("C4").Offset(-1,1) a ActiveCell.Offset(0,0)

46. Je-li v aktivní buňce vzorec $=10^2$, jakou hodnotu vrátí `ActiveCell.Value` a `ActiveCell.FormulaR1C1`



Úlohy k řešení

Pro prostředí MS Excel si stáhněte dokument VB04.xlsx a proveďte následující úkony:

- Laděním subrutiny **PrincipCykluFor** prostudujte princip inkrementačního cyklu For. V následující subrutině **VypocetFaktorialu** doplňte cyklus, jenž zajistí výpočet faktoriálu čísla.

Postup:

- Otevřete subrutinu (např. záložka "Vývojář" tlačítko "Visual Basic", v aplikaci „Microsoft Visual Basic“ zvolit modul „A_InkrementacniCyklus“.
 - Krokováním analyzujte subrutinu **PrincipCykluFor**.
 - Prostudujte subrutinu **VypocetFaktorialu** a dopište cyklus For, jehož náplní bude opakovaně realizovat operaci `Faktorial = Faktorial * i`, kde `i` bude nabývat hodnot od `Cislo` po 2 (např. pro `Cislo=5` bude proměnná nabývat hodnot 5,4,3,2). Provedením cyklu se tak postupně uskuteční výpočet $((1*5)*4)*3)*2$.
 - Ověřte funkci faktoriálu.
- Prostudujte všechny tři varianty logického cyklu: **PrincipLogickehoCykluPodminkaNaZacatku**, **PrincipLogickehoCykluPodminkaNaKonci**, **PrincipLogickehoCykluPodminkaUprostred**. V následující subrutině **ScitaniCisel** realizujte sčítací algoritmu, který bude opakovaně vyzývat uživatele k zadání čísla do té doby, dokud nezadá nulu. Po zadání nuly se zobrazí součet všech zadaných čísel.

Postup:

- Otevřete modul „B_LogickeCykly“.
 - Krokováním analyzujte všechny tři varianty logického cyklu.
 - Prostudujte subrutinu **ScitaniCisel** a dopište libovolnou variantu logického cyklu, jehož náplní bude opakovaně realizovat operaci součtu `Soucet = Soucet + Cislo` a výzvy k zadání čísla `Cislo = InputBox(...)` do okamžiku zadání nuly.
 - Ověřte vytvořenou subrutinu.
- Vyzkoušejte subrutinu **VyberBunky** v modulu „C_PristupBunky“ a prostudujte základní varianty programového výběru buňky. Taktéž prostudujte subrutinu

ObsahBunky demonstrující způsoby přístupu k obsahu buňky. Poté upravte subrutinu **ReseniKvadratickeRovnice** modulu „D_KvadratickaRovnice“ tak, aby koeficienty a, b, c byly načteny z listu „Kvadratická rovnice“ a naopak zápis vypočtených kořenů byl proveden do příslušných buněk téhož listu.

Postup:

- a. Aktivujte list „Kvadratická rovnice“ a v poli názvu pojmenujte příslušné buňky jako **KoeficientA**, **KoeficientB** a **KoeficientC**.
 - b. Otevřete modul „D_KvadratickaRovnice“.
 - c. Upravte příkazy přiřazení hodnot koeficientům, např. `a = Range("KoeficientA").Value`.
 - d. Obdobným postupem realizujte zápis vypočtených kořenů do příslušných buněk.
 - e. Aktivujte list „Kvadratická rovnice“ a ověřte si správnou funkci makra.
4. Vyzkoušejte a analyzujte subrutinu **OznacitHodnotySloupceNadLimit**. Zaměřte se na metodu procházení sloupce hodnot. Následně doplňte v makru **PrevestSloupecNaCislo** cyklus procházení buněk od aktuální do konce sloupce, za účelem převodu textové formy reálných hodnot s oddělovačem tečka na reálné hodnoty s oddělovačem desetinné čárky.

Postup:

- a. Na listu „Procházení oblasti“ aktivujte Buňku C8 a pomocí tlačítka „Označit nadlimitní hodnoty sloupce“ spusťte makro **OznacitHodnotySloupceNadLimit**.
 - b. Otevřete modul „E_ProchazeniOblastiBunek“ a krokováním prostudujte makro **OznacitHodnotySloupceNadLimit**.
 - c. Doplňte subrutinu **PrevestSloupecNaCislo** o cyklus procházení sloupce dat.
 - d. Nad listen „Procházení oblasti“ ověřte funkci subrutiny.
5. Vyzkoušejte a analyzujte subrutinu **OznacitHodnotyNadLimit**. Zaměřte se na metodu procházení libovolného výběru buněk. Následně v subrutině **PrevestNaCislo** doplňte cyklus procházení uživatelem vybrané skupiny buněk, za účelem převodu textové formy reálné hodnoty s oddělovačem tečka na reálnou hodnotu s desetinným oddělovačem čárka.

Postup:

- a. Na listu „Procházení oblasti“ označte skupinu buněk a tlačítkem „Označit nadlimitní hodnoty“ spusťte makro **OznacitHodnotyNadLimit**.
- b. Otevřete modul „E_ProchazeniOblastiBunek“ a krokováním prostudujte makro **OznacitHodnotySloupceNadLimit**.
- c. Doplňte subrutinu **PrevestNaCislo** o cyklus procházení výběru skupiny buněk.
- d. Nad listen „Procházení oblasti“ ověřte funkci subrutiny.

10 FUNKCE, SUBROUTINY A ARGUMENTY



Čas ke studiu: 2,5 hodiny



Cíl Po prostudování této kapitoly budete umět

- Volat subrutiny a funkce
- Tvořit vlastní funkce
- Definovat argumenty funkcí a subrutin
- Používat vybrané systémové funkce



Výklad

Subrutiny a funkce (obecně rutiny) jsou ohraničené a pojmenované sekvence příkazů tvořící logické celky algoritmů, které můžeme opakovaně volat z různých míst kódu. Tyto rutiny jsou základními elementy programů. V podstatě neexistuje program, který by neměl alespoň jednu subrutinu nebo funkci. Názvy rutin jsou identifikátory, tudíž v rámci jejich platnosti jsou vždy jedinečné. Funkce i subrutiny mohou mít argumenty, kterými se dovnitř předávají proměnné nebo hodnoty, na jejichž obsahu pak závisí činnost nebo výsledek rutin.

Funkce se od subrutiny liší tím, že má návratovou hodnotu, reprezentující její výsledek. Tato odlišnost se projevuje nejen ve vlastní deklaraci těchto algoritmů, ale především ve způsobu volání nebo použití. Například v prostředí MS Office jsou makra vždy tvořena subrutinami bez argumentů. Na druhou stranu, uživatelské funkce se v MS Excelu mohou vkládat do buněk.

10.1 Definice Funkcí

Funkce je pojmenovaná rutina reprezentující skupinu příkazů ohraničených klíčovými slovy `Function` a `End Function`, které na základě argumentů nebo stavů jiných proměnných určí výsledek. Tento výsledek se předává přes návratovou hodnotu funkce, kterou tvoří její identifikátor. Funkce se v programech obvykle nacházejí na místech, kde lze použít výraz, konstantu nebo proměnnou.

Syntaxe:

```
Function Název([Argument As Typ][, ...]) As TypFunkce
    [Kód funkce]
    Název = [návratová hodnota]
End Function    [příkazy před podmínkou]
```

- Název funkce je současně proměnná, přes kterou funkce vrací výsledek (návratovou hodnotu funkce).
- Typ návratové hodnoty se uvádí v hlavičce funkce za závorkami. Není-li explicitně typ uveden, je automaticky chápán jako variant.
- Výsledek funkce se předává přiřazením hodnoty do proměnné s názvem funkce.
- Argumenty funkce se uvádějí do závorek za identifikátorem a oddělují se čárkami.
- U každého argumentu se uvádí jeho typ (neuvede-li se je automaticky Variant).
- Počet argumentů není omezen. Funkce může také existovat bez argumentů.

```
Option Explicit
Function Pozdrav() As String
    If Time < #9:00:00 AM# Then
        Pozdrav = "Dobré ráno"
    ElseIf Time >= #6:00:00 PM# Then
        Pozdrav = "Dobrý večer"
    Else
        Pozdrav = "Dobrý den"
    End If
End Function
Function ObsahObdelniku(StranaA As Double, StranaB As Double) As Double
    ObsahObdelniku = StranaA * StranaB
End Function
```

Obr. 89 Příklady deklarace funkcí

10.2 Volání funkcí v kódu:

Funkce mohou být volány dvěma způsoby v závislosti na metodě předávání argumentů:

1. pozičně předané argumenty,
2. předání argumentů podle názvu.

Syntaxe volání funkcí pozičně předaných argumentů:

Dim Hodnota As Typ, Arg1 As Typ, Arg2 As Typ

Hodnota = Funkce([Arg1][, Arg2][, 3])

- Argumenty se uvádějí v pořadí podle umístění v hlavičce funkce. Pořadí je dáno definicí funkce.
- Oddělují se čárkami, přičemž nepovinné se mohou přeskočit zápisem dvou čárek za sebou.
- Je-li návratová hodnota funkce čtena, musí být argumenty uzavřeny v závorkách. Jinými slovy, pokud nás návratová hodnota nezajímá, můžeme funkci volat jako subrutinu.

```
Sub Test()  
    Dim Obsah As Double, A As Double: A = 5  
    Obsah = ObsahObdelniku(A, 5)  
End Sub
```

Obr. 90 Volání funkce a poziční předávání argumentů

Syntaxe volání funkcí s předáváním argumentů podle názvu:

Dim Hodnota As Typ, Arg1 As Typ, Arg2 As Typ

Hodnota = Funkce([ArgA:=Arg1][, ArgC:=3][, ArgB:=Arg2])

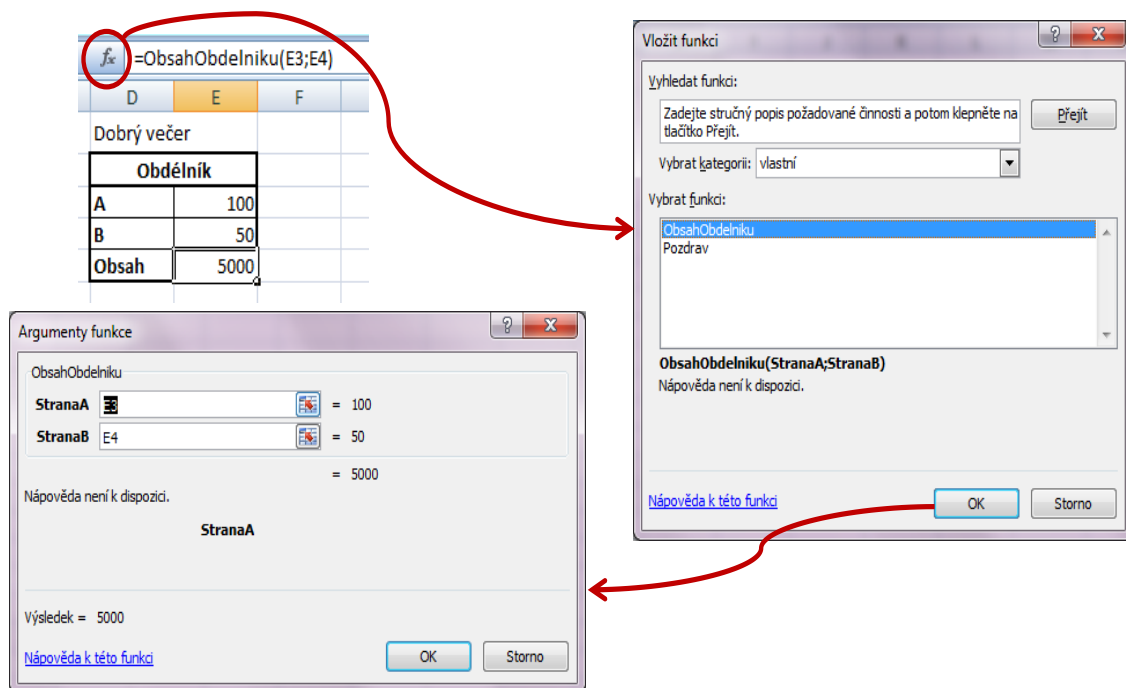
- U každého argumentu se nejprve uvede jeho název, pak se zapíše sekvence dvojtečky a rovnítka (:=) a za nimi se uvede hodnota argumentu nebo proměnná.
- Na pořadí argumentů nezáleží.
- Nepovinné argumenty se jednoduše neuvádějí.

```
Sub Test()  
    Dim Obsah As Double, A As Double: A = 5  
    Obsah = ObsahObdelniku(StranaA:=A, StranaB:=5)  
End Sub
```

Obr. 91 Volání funkce a předávání argumentů podle názvu

10.3 Volání funkcí v tabulkách MS Excelu

Vytvoříme-li funkce v modulech dokumentu MS Excelu, můžeme tyto funkce volat stejně jako funkce ze standardní knihovny. Pro uživatelské funkce MS Excel vytvoří kategorii „Vlastní“. V buňkách se argumenty funkcí oddělují středníkem.



Obr. 92 Demonstrace vkládání uživatelských funkcí do buněk MS Excelu

10.4 Definice subrutiny

Subrutina je pojmenovaná sekvence příkazů ohraničených klíčovými slovy **Sub** a **End Sub**. V prostředí jazyka Visual Basic for Application se pomocí subrutin bez parametrů vytvářejí makra. Makro je subrutina, která nemá argumenty, je veřejná a je umístěna v běžném (makro) modulu (obvykle v dokumentech aplikace).

Syntaxe definice subrutiny (makra) bez argumentů:

Sub Název()

[Kód subrutiny]

End Sub

- Subrutiny nemají návratovou hodnotu, tudíž název nemá druhotný význam, takže se ani neuvádí jeho datový typ.

Syntaxe volání subrutiny z kódu:**Název**

- Subrutina se volá uvedením názvu (identifikátoru) na samostatný řádek (jeden příkaz).

10.5 Subrutiny s argumenty

Stejně jako funkce i subrutiny mohou mít argumenty. Ty se tvoří naprosto stejným způsobem jako u funkcí s tím rozdílem, že zde není návratová hodnota.

Syntaxe definice subrutiny s argumenty:

```
Sub Název([Argument1 As Typ][,Argument2 As Typ] [, ...])
```

```
    [Kód subrutiny]
```

```
End Sub
```

- Argumenty subrutin se uvádějí do závorek za identifikátorem a oddělují se čárkami.
- U každého argumentu se uvádí jeho typ (neuvede-li se je automaticky Variant).
- Počet argumentů není omezen.

Syntaxe volání subrutin s argumenty:

```
Dim Arg1 As Typ, Arg2 As Typ
```

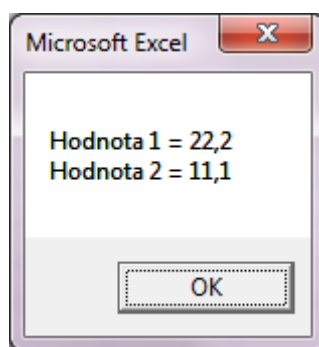
```
Název [Argument1:=Arg1][, Argument2:=Arg2][, Argument3:=3]
```

- Volání subrutiny je vždy na samostatném řádku.
- Subrutina se volá uvedením názvu (identifikátoru), za kterým se uvádějí hodnoty argumentů odděleny čárkami, přičemž nepovinné se mohou přeskočit zápisem dvou čárek za sebou.
- Obdobně jako u funkce se argumenty mohou předávat pozičně, nebo podle názvu.
- Jelikož subrutiny nemají návratový typ, je možné výsledky přes argumenty také navracet, nebo je možné využít globální proměnné.

```
Sub ZamenitHodnoty(A As Double, B As Double)
    Dim Pomocna As Double
    Pomocna = A
    A = B
    B = Pomocna
End Sub

Sub Main()
    Dim Hodnota1 As Double, Hodnota2 As Double
    Hodnota1 = 11.1
    Hodnota2 = 22.2
    ZamenitHodnoty Hodnota1, Hodnota2
    MsgBox "Hodnota 1 = " & Hodnota1 & vbNewLine & _
        "Hodnota 2 = " & Hodnota2
End Sub
```

Obr. 93 Příklad subrutiny s argumenty a její volání



Obr. 78 Výsledek příkladu

V příkladu je patrné, že argumenty do subrutiny nejen vstupují, ale jsou také zpětně čteny. V tomto případě hovoříme o předávání argumentů odkazem.

10.6 Předávání argumentů odkazem a hodnotou

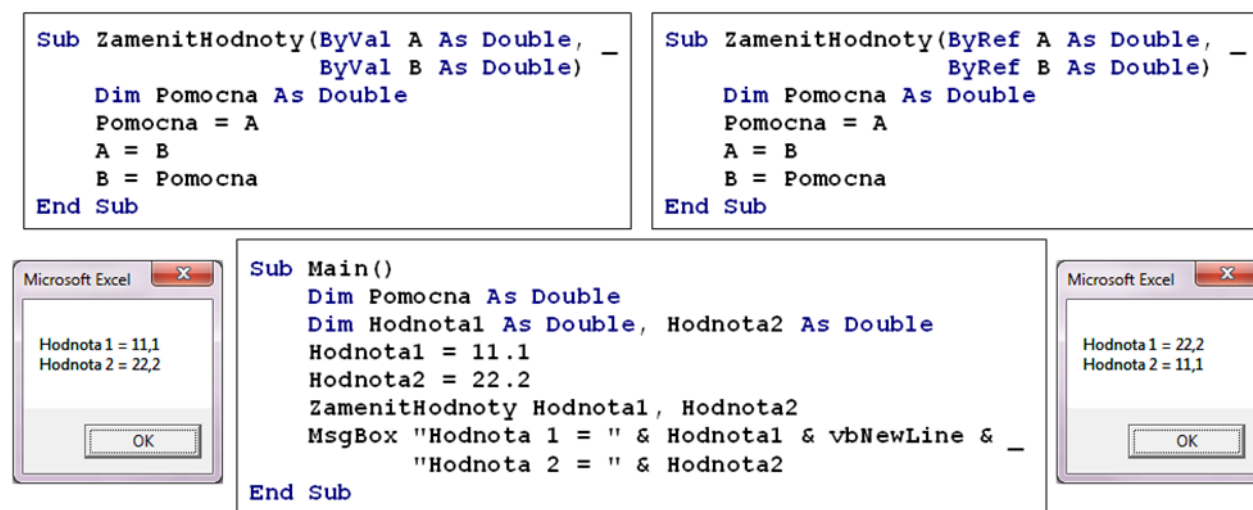
Argumenty funkcí a subrutin se mohou předávat dvěma způsoby:

- hodnotou (ByVal) nebo
- odkazem (ByRef).

U prvního způsobu má funkce/subrutina vlastní zcela izolované proměnné (argumenty), které jsou při volání rutiny naplněny kopiemi předávaných hodnot. Změna hodnoty takového argumentu v těle rutiny neovlivní původní proměnné nadřazené rutiny. V tomto případě

mohou být proměnné nadřazené rutiny jiného datového typu nežli argumenty rutiny, nicméně musí být zachována možnost konverze. Předávání hodnotou se zapíná klíčovým slovem **ByVal** uvedeným před argumentem.

U druhého způsobu jsou argumenty rutin ve své podstatě zástupci předávaných proměnných. V tomto případě musí být datové typy shodné s datovými typy argumentů, jinak algoritmus nebude spuštěn. Výjimku tvoří případy, kdy se místo proměnných předávají konstanty. Předávání odkazem je ve Visual Basicu implicitní. Pro zdůraznění se může zapsat klíčové slovo **ByRef** před argument.



Obr. 94 Vliv předávání hodnot argumentů odkazem a hodnotou

Jinou možností sdílení hodnot mezi algoritmy nabízí globální proměnné.

10.7 Globální proměnné

Syntaxe

Private|Public Název As Typ

- Jsou deklarovány mimo těla funkcí a subrutin.
- **Private** proměnné jsou sdíleny všemi subrutinami a funkcemi téhož modulu.
- **Public** proměnné jsou sdíleny všemi subrutinami a funkcemi celého projektu.

```
Option Explicit
Private Hodnota1 As Double, Hodnota2 As Double
Sub ZamenitHodnoty()
    Dim Pomocna As Double
    Pomocna = Hodnota1
    Hodnota1 = Hodnota2
    Hodnota2 = Pomocna
End Sub

Sub Main()
    Hodnota1 = 11.1
    Hodnota2 = 22.2
    ZamenitHodnoty
    MsgBox "Hodnota 1 = " & Hodnota1 & vbCrLf & _
        "Hodnota 2 = " & Hodnota2
End Sub
```

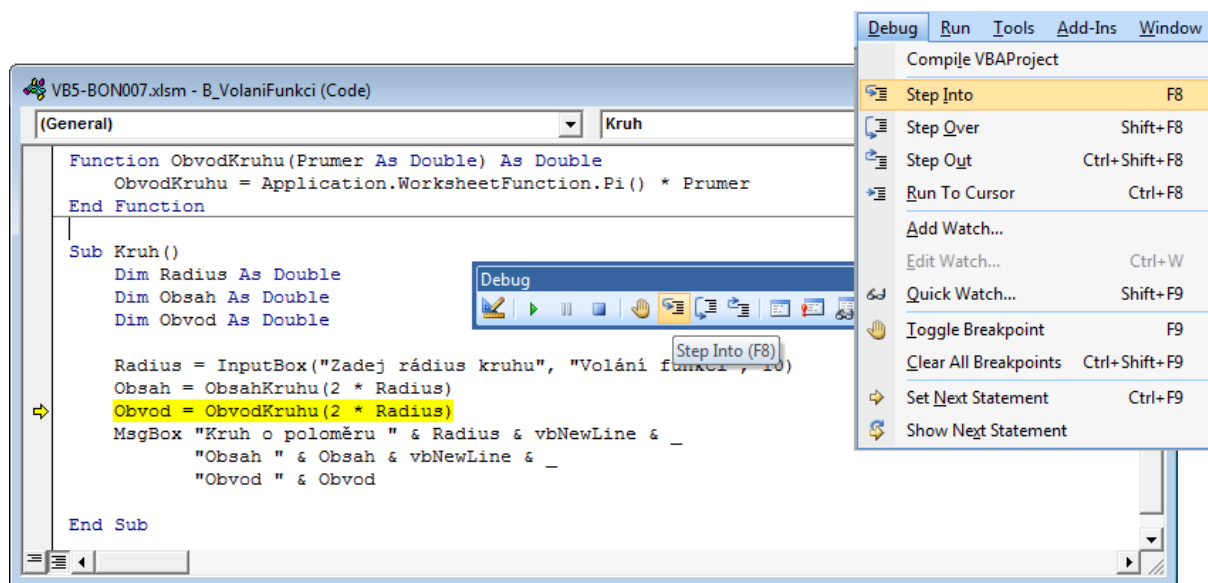
Obr. 95 Příklad globálních proměnných

Výhodou a zároveň nevýhodou globálních proměnných je jejich sdílení mezi všemi subrutinami v daném modulu nebo celém projektu. Je-li potřeba proměnnou sdílet mezi několika rutinami současně, pak globální proměnné nabízejí snadné řešení. Na druhou stranu se kód stává méně přehledným a navíc hrozí, že proměnnou bude modifikována jinými rutinami. Rozhodně není vhodné, přes globální proměnné sdílet všechny potřebné proměnné. Jejich použití bychom si měli vždy pečlivě rozvážit.

10.8 Ladění subrutin a funkcí

Při ladění algoritmů, které volají subrutiny či funkce se využívají tři metody kroků:

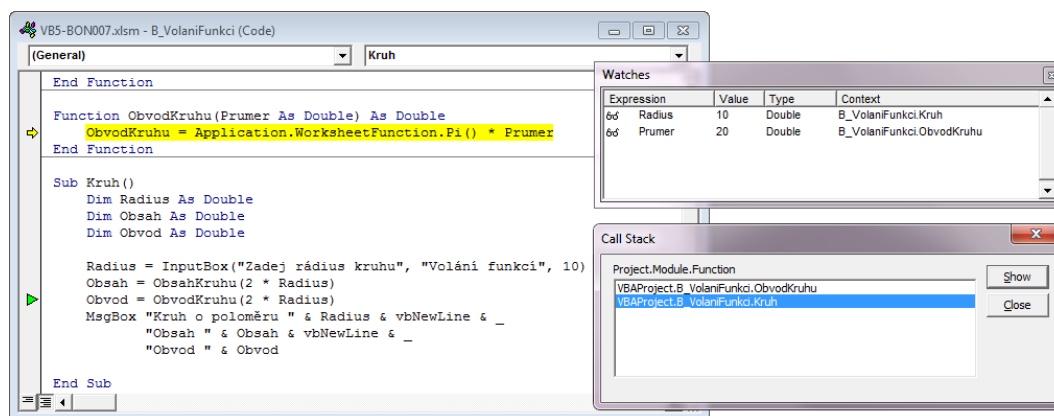
- Step Into (F8) krok s vnořením do rutiny. V tomto případě krokování pokračuje ve volané rutině.
- Step Over (Shift + F8) vykonání celé volané rutiny v jednom kroku. Při této volbě se vykoná celá rutina najednou, přičemž krokování pokračuje na dalším příkazu za jejím voláním. Do rutiny se vnoří pouze, je-li v rutině zarážka nebo při vzniku výjimky.
- Step Out (Ctrl + Shift + F8) v jednom kroku rutinu dokončí a pozastaví se až na následujícím příkazu za jejím voláním. Obvykle se tento krok používá, když dojde k nechtěnému vnoření do rutiny (příkazem Step Into).



Obr. 96 Nabídka tří druhů kroků při ladění rutin

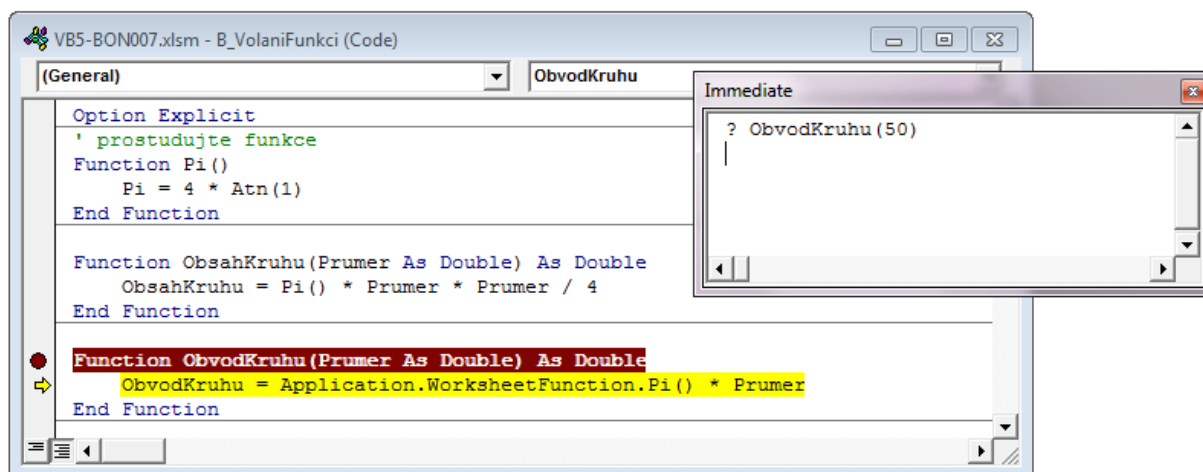
Pro sledování proměnných je nevhodnější okno **Watches**, které umožňuje sledovat proměnné volající i volané rutiny současně.

Dalším zajímavým oknem je **Call Stack** (Ctrl+L), které zobrazuje kaskádu vzájemného volání rutin a po poklepnutí umožňuje zobrazit místo v kódu, odkud byla rutina volána (viz zelená šipka v levé liště okna kódu na následujícím obrázku).



Obr. 97 Sledování proměnných volané a volající rutiny a okno Call Stack

Funkce a subrutiny s argumenty můžeme také ladit samostatně tak, že do dané rutiny vložíme zarážku a zavoláme ji přes okno **Immediate**.



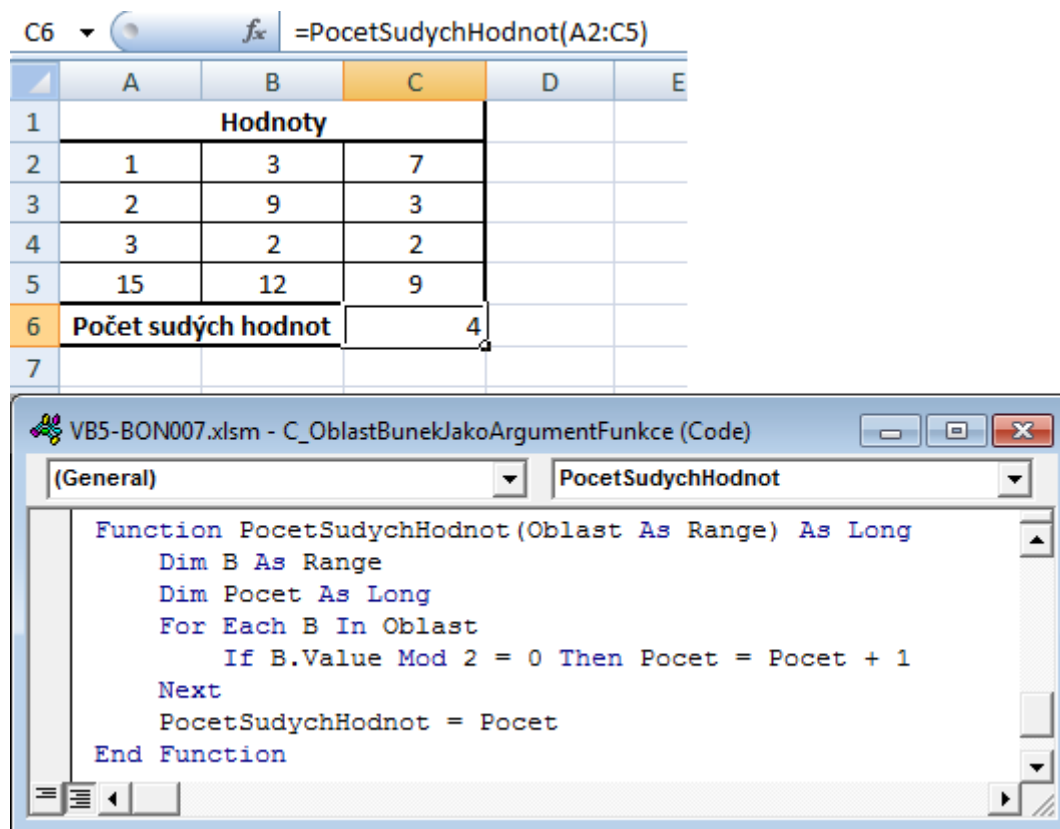
Obr. 98 Příklad zahájení ladění funkce ObvodKruhu přes okno Immediate

Ladění funkcí v buňkách MS Excelu

Funkce vložené do buněk MS Excelu je také možné ladit. V tomto případě se do funkce nejprve vloží zarážka (například na hlavičku funkce) a pak se vyvolá její přepočítání. Přepočítání je možné vynutit ukončením editace buňky (F2 a pak ENTER), nebo změnou libovolného argumentu funkce.

10.9 Buňka a oblast jako argument funkce

Funkce v listech MS Excelu často pracují s oblastmi buněk. Například funkce SUMA, POČET, MAX atd. Také uživatelské funkce mohou mít takovéto argumenty. Vše zajistí datový typ **Range**. Argumenty tohoto typu Excel automaticky rozpozná a na jeho místě předá buňku nebo celou oblast buněk podle výběru uživatele. Funkce pak obdrží kolekci buněk, která se obvykle prochází cyklem **For Each – Next**.



Obr. 99 Příklad funkce s argumentem typu kolekce buněk

10.10 Systémové funkce

Visual Basic for Application poskytuje sadu systémových funkcí, které efektivně řeší mnoho standardních úloh. Podle typu těchto úloh můžeme funkce dělit do několika kategorií:

- Matematické funkce,
- Řetězcové funkce,
- Funkce data a času,
- Konverzní funkce,
- Finanční funkce,
- Funkce pro práci se soubory,
- Funkce pro práci s registry Windows

a mnoho dalších. Seznamy všech funkcí s podrobnostmi jejich používání jsou uvedeny v nápovědě jazyka Visual Basic for Application. S některými funkcemi je dobré se seznámit co nejdříve.

Řetězcové funkce

Řetězcové funkce se používají ke zpracování textů. Většina z nich vrací nebo očekává hodnotu datového typu **String**.

- Abecední porovnání řetězců s podporou národních abeced. Funkce vrací kladné číslo, je-li první řetězec v abecedě později nežli druhý řetězec, zápornou hodnotu je-li tomu naopak a nulu jsou-li texty shodné.

StrComp(string1, string2[, compare])

- Transformace řetězců na jiné kódování.

StrConv(string, conversion, LCID)

- Změna velikosti písma řetězce na malé (LCASE) nebo velké (UCASE).

LCASE(string); UCASE(string)

- Vytvoření řetězce opakováním mezer, nebo libovolného znaku.

Space(number); String(number, character)

- Zjištění délky řetězce – vrací počet znaků textu nebo nulu pro prázdný text.

Len(string)

- Formátovaný převod hodnot na řetězce. Tato funkce umožňuje převádět hodnoty různých typů na String podle specifického formátu.

Format(expression[, format])

- Vyhledání řetězce v řetězci. Vrací celé číslo představující pozici hledaného řetězce zleva, nebo nulu, není-li text nalezen. Funkce **InStr** vyhledává od pozice start řetězec zleva, kdežto **InStrRev** zprava.

InStr([start,] string1, string2 [,compare])

InstrRev(strcheck, strmatch[, start[, compare]])

- Vystřížení části řetězce o délce lenght zleva (Left), zprava (Right) nebo z libovolné pozice start (Mid).

Left(string, length)

Right(string, length)

Mid(string, start[, length])

- Ořezání řetězce od počátečních nebo koncových mezer zleva, zprava a z obou stran.

LTrim(string);RTrim(string);Trim(string)

- Záměna řetězce v řetězci nebo odstranění všech výskytů řetězce.

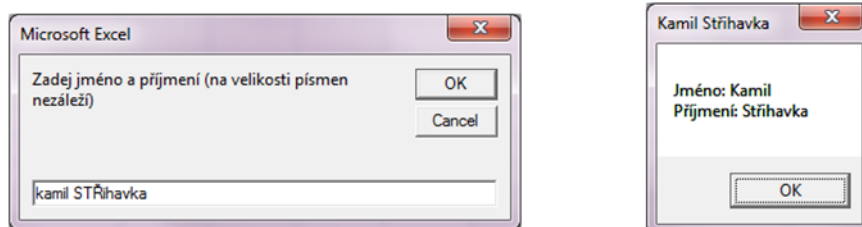
Replace(expression, find, replace)

Příklad na obr. 100 demonstruje používání řetězcových funkcí na rozkladu a naformátování textu zadaného jména osoby. Algoritmus rozdělí text na dvě slova Jméno a Příjmení a tyto slova upraví tak, že začínají velkými písmeny a pokračují malými.

```

Sub Main()
    Dim CeleJmeno As String, Jmeno As String, Prijmeni As String
    CeleJmeno = InputBox("Zadej jméno a příjmení (na velikosti písmen nezáleží)")
    CeleJmeno = Trim(CeleJmeno) 'Odstranění mezer na začátku a konci
    Dim i As Integer
    i = InStrRev(CeleJmeno, " ") 'Vyhledání mezery mezi slovy
    If i > 0 Then
        Jmeno = Left(CeleJmeno, i - 1) 'První slovo zleva
        'První písmeno na velké, zbytek na malé
        Jmeno = UCase(Left(Trim(Jmeno), 1)) & LCase(Mid(Trim(Jmeno), 2))
        Prijmeni = Mid(CeleJmeno, i + 1)
        Prijmeni = UCase(Left(Trim(Prijmeni), 1)) & LCase(Mid(Trim(Prijmeni), 2))
    End If
    CeleJmeno = Jmeno & " " & Prijmeni
    MsgBox "Jméno: " & Jmeno & vbNewLine & _
        "Příjmení: " & Prijmeni, , CeleJmeno
End Sub

```



Obr. 100 Příklad použití řetězcových funkcí

Funkce data a času

Datum a čas se uchovává v proměnných typu **Date**.

- Získání aktuálního (systémového) data, času a okamžiku.

Date(); Time(); Now()

- Datum po přičtení nebo odečtení zadaného počtu časových intervalů.

DateAdd(interval, number, date)

- Zjištění počtu časových intervalů mezi dvěma daty.

DateDiff(interval, date1, date2)

- Číselné vyjádření části data (dne, týdne, pořadového dne týdne, měsíce, roku apod.).

DatePart(interval, date)

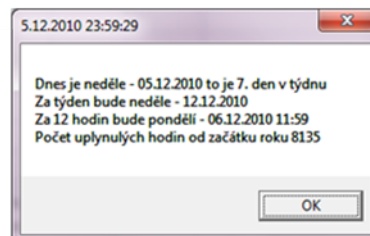
- Datum na základě čísla roku, měsíce a dne.

DateSerial(year, month, day)

- Číselně vyjádření kalendářního dne, roku, měsíce.

Day(date), Year(date), Month(date)

```
Sub DatumCas()
    Dim Dnes As Date, Nyni As Date
    Dim ZaTyden As Date, Zal2hodin As Date
    Dim PoradovyDenTydnu As Integer
    Dim PocetHodinOdZacatkuRoku As Integer
    Dnes = Date: Nyni = Now
    ZaTyden = DateAdd("ww", 1, Date)
    Zal2hodin = DateAdd("h", 12, Now)
    PoradovyDenTydnu = DatePart("w", Date, vbMonday)
    PocetHodinOdZacatkuRoku = DateDiff("h", DateSerial(Year(Date), 1, 1), Now)
    MsgBox "Dnes je " & Format(Dnes, "dddd - dd.mm.yyyy") & _
        " to je " & PoradovyDenTydnu & ". den v týdnu" & vbNewLine & _
        "Za týden bude " & Format(ZaTyden, "dddd - dd.mm.yyyy") & vbNewLine & _
        "Za 12 hodin bude " & Format(Zal2hodin, "dddd - dd.mm.yyyy HH:NN") & vbNewLine & _
        "Počet uplynulých hodin od začátku roku " & PocetHodinOdZacatkuRoku, , Nyni
End Sub
```



Obr. 101 Příklad použití řetězcových funkcí

**Shrnutí pojmů**

Rutiny jsou ohraničené a pojmenované sekvence příkazů tvořící logické celky algoritmů, které můžeme opakovaně volat z různých míst kódu. Ve Visual Basicu se dělí na Subrutiny a Funkce.

Subrutina je skupinou příkazů vložených mezi příkazy Sub a End Sub, vykonávajících požadovanou úlohu.

Funkce je rutina reprezentující skupinu příkazů ohraničených klíčovými slovy Function a End Function, které na základě argumentů, nebo stavů jiných proměnných určí výsledek.

Argument funkce/subrutiny je parametrická proměnná přes kterou do algoritmu vstupují hodnoty, nebo algoritmus přes něj vrací výsledky.

Návratová hodnota funkce/subrutiny je výsledná hodnota funkce.



Otázky

47. Může být volání subrutiny součástí výrazu?
48. Je při volání subrutiny možné, aby na místě argumentu byla volaná jiná funkce?
49. Kdy se argumenty uzavírají do závorek a kdy ne?
50. Jaký je oddělovač argumentů funkcí v buňkách MS Excelu?
51. Jaký je rozdíl mezi pozičním a jmenným předáváním argumentů?
52. Jaký je rozdíl mezi globální proměnnou **Public** a **Private**?
53. Jak zjistíme délku textu uloženého v proměnné **Název**?
54. V proměnné **Jméno** je uvedeno celé jméno osoby, jak z něj získat pouze příjmení velkými písmeny?
55. Jak snadno odstranit všechny mezery v textu?
56. Jakým způsobem zjistíme datum, které bude za 14 dní?



Úlohy k řešení

Pro prostředí MS Excel si stáhněte dokument VB05.xlsx a proveďte následující úkony:

1. Aktivujte list **Funkce** a prohlédněte si obsah buňky B1, ve které je vložena funkce **Pozdrav**. Následně do funkce **Pozdrav** v modulu **A_Funkce** vložte zářádku a krokováním prostudujte tělo funkce. Funkci upravte tak, aby při každém pozdravu navíc vypisovala vaše jméno. Například „Dobrý den Bonde“.

Postup:

- a. Otevřete modul **A_Funkce** a nad hlavičkou funkce **Pozdrav** klepnutím na levou lištu (nebo klávesou F9) vložte zářádku.
- b. Aktivujte buňku B1 nad listem **Funkce** a potvrzením editace buňky (F2 následně Enter) vyvolejte přepočítání funkce.
- c. Krokováním prostudujte algoritmus funkce.
- d. Do řetězců jednotlivých pozdravů dopište vaše jméno v pátém pádu.

e. *Opět přepočítejte buňku.*

2. Krokováním prostudujte funkci **ObsahObdelniku**. Zaměřte se na její argumenty **StranaA** a **StranaB** a způsob vložení do buňky B6. Změnou hodnot buněk B4 a B5 upravte rozměry obdélníka tak, aby délka strany A byla shodná s číselnou částí vašeho osobního čísla a strana B měla hodnotu 100.

Postup:

- Otevřete modul Funkce a nad hlavičkou funkce **ObsahObdelniku** klepnutím na levou lištu (nebo klávesou F9) vložte zarážku.*
 - Aktivujte buňku B6 a prostudujte zápis vzorce.*
 - Změnou hodnot argumentů (B4 a B5) vyvolejte přepočet funkce, který vyvolá její ladění.*
 - Zrušte zarážku a nastavte velikost strany A (buňka B4) podle hodnoty numerické části vašeho osobního čísla (např. pro Bon007 vložíte 7). Stranu B (buňka B5) nastavte na hodnotu 100.*
 - Krokováním prostudujte algoritmus funkce.*
 - Opět přepočítejte buňku.*
3. V modulu **A_Funkce** vytvořte funkci **ObvodObdelniku**, která vypočte obvod obdélníku na základě argumentů **StranaA** a **StranaB** a vložte ji do buňky B7 tak, aby funkce přebírala argumenty stran obdélníku z buněk B4 a B5.

Postup:

- V modulu **A_Funkce** zkopírujte funkci **ObsahObdelniku** a přejmenujte její kopii na **ObvodObdelniku**.*
 - Upravte výpočet funkce*
$$\text{ObvodObdelniku} = 2 * (\text{StranaA} + \text{StranaB})$$
 - Aktivujte buňku B7 a vyvolejte dialog Vložit funkci (například kombinací Shift + F3).*
 - V rozbalovacím boxu „Vybrat kategorii“ vyberte položku „Vlastní“, následně v seznamu vyberte funkci **ObvodObdelniku** a stiskněte tlačítko OK.*
 - V dialogu „Argumenty funkce“ vepište adresy buněk s velikostmi stran obdélníku a uzavřete dialog tlačítkem OK.*
4. V modulu **B_VolaniFunkci** prostudujte funkce **ObsahKruhu** a **ObvodKruhu**, povšimněte si metody určení hodnoty π . Následně funkce vložte do buněk B11 a B12 tak, aby rádius byl převzat z buňky B11 (pozor na rádius a průměr). Poté si prostudujte subrutinu **Kruh** a upravte ji tak, aby výpočet obsahu a obvodu se prováděl pomocí funkcí **ObsahKruhu** a **ObvodKruhu**. Nakonec si krokováním subrutinu vyzkoušejte, přičemž u volání funkcí si vyzkoušejte rozdíl mezi krokem dovnitř (F8) a krokem přes (Shift + F8).

Postup:

- a. Do buňky B11 zapište vzorec `=ObsahKruhu(B10*2)` a do buňky B12 `vzorec=ObvodKruhu(B10*2)`.
 - b. Otevřete modul **B_VolaniFunkci**.
 - c. Krokováním analyzujte subrutinu **Kruh**.
 - d. Doplněte příkazy volání funkcí pro výpočet obvodu a obsahu:
`Obsah = ObsahKruhu(2 * Radius)`
`Obvod = ObvodKruhu(2 * Radius)`
 - e. Krokováním příkazem **Step Into** (F8) se vnořte do těla funkce **ObsahKruhu** a volbou **Step Out** (Ctrl + Shift + F8) z něj vyskočte.
 - f. Krokováním příkazem **Step Over** (Shift + F8) proveďte příkaz volání funkce **ObvodKruhu**.
5. V modulu **C_OblastBunekJakoArgumentFunkce** prostudujte funkci **PocetHodnot**. Zaměřte se na princip předávání oblasti buněk do funkce a nepovinného argumentu **Hodnota**. Do funkce **PocetSudychHodnot** přidejte argument **Oblast** a vložte tuto funkci do buňky B22 tak, aby vracela počet sudých hodnot tabulky A15:C18.

Postup:

- a. Prostudujte zápis vložené funkce v buňkách B20 a B21, povšimněte si rozdílný počet argumentů.
 - b. Otevřete modul „OblastBunekJakoArgumentFunkce“ a prostudujte zápis argumentu **Hodnota** u funkce **PocetHodnot**.
 - c. Vložte zarážku do funkce **PocetHodnot** a změnou hodnot jedné z buněk tabulky **Hodnoty** vyvolejte ladění.
 - d. Zrušte zarážku.
 - e. Do funkce **PocetSudychHodnot** vložte argument
`Oblast As Range`
 - f. Vložte funkci do buňky B22 tak, aby funkce spočetla počet sudých hodnot tabulky.
6. Prostudujte subrutinu **Zamena**, která volá subrutinu **ZamenitHodnoty** za účelem prohození hodnot mezi proměnnými **Hodnota1** a **Hodnota2**. Pak opravte zápis argumentů u subrutiny **ZamenitHodnoty** tak, aby se operace zdařila.

Postup:

- a. Příkazem **Step Into** (F8) zahajte krokování subrutiny.
- b. Stejným příkazem se vnořte do funkce **ZamenitHodnoty** a zobrazte si obsah proměnných **Hodnota1** a **Hodnota2** a argumentů **A** a **B**.
- c. Krokováním sledujte, jak se tyto proměnné navzájem ovlivňují.

d. U argumentu **A** změňte způsob předávání, tak aby místo obdržení kopie hodnoty (**ByVal**) sdílel hodnotu s proměnnou **Hodnota2** (**ByRef**).

e. Změňte datový typ argumentu **A** tak, aby odpovídal proměnné **Hodnota2**.

ZamenitHodnoty(ByRef A As Double, ByRef B As Double)

7. Prostudujte subrutiny v modulu **E_UzitiSystemovychFunkci**.

Rejstřík

Algoritmus, 52

Argument, 53, 129

Cyklus, 104

Datový typ, 64

Deklarace, 64

False, 79

Formulář, 41

Funkce, 53, 129

Identifikátor, 41

Iterační cyklus, 104

Klíčové slovo, 41

Kolekce, 104

Konstata, 53

Krokování, 76

Laděná, 76

Logické operátory, 89

Logický výraz, 89

Modul, 40

Návratová hodnota, 53, 129

Název makra, 28

Použít relativní adresy, 28

Program, 52

Procházení kolekcí, 104

Projekt, 40

Proměnná, 64

Prvek, 104

Příkaz, 52

Relační operátory, 89

Run to Cursor, 74

Rutiny, 129

Step Into, 74

Step Out, 74

Step Over, 74

Subrutina, 53, 129

True, 79

Větvení programů, 89

Výjimka, 64

Zabezpečení maker, 28

Zarážka, 76