Univerzita Hradec Králové

Přírodovědecká fakulta

BAKALÁŘSKÁ PRÁCE

2014 Tomáš Vlček

Univerzita Hradec Králové

Přírodovědecká fakulta

Katedra informatiky

Programování v MS Excel

Bakalářská práce

Autor: Tomáš Vlček

Studijní program: B1801 Informatika

Studijní obor: Informatika se zaměřením na vzdělávání

Společenské vědy se zaměřením na vzdělávání

Vedoucí práce: doc. RNDr. Štěpán Hubálovský, Ph.D.

Hradec Králové prosinec 2014

Univerzita Hradec Králové
Přírodovědecká fakulta

Zadání diplomové práce

Autor:	**Tomáš Vlček**
Studijní program:	B1801 Informatika
Studijní obor:	Informatika se zaměřením na vzdělávání Společenské vědy se zaměřením na vzdělávání
Název práce:	**Programování v MS Excel**
Název práce v AJ:	Programming in MS Excel spreadsheet

Cíl, metody, literatura, předpoklady
Cílem práce bude vytvořit sbírku řešených příkladů z programování v tabulkovém procesoru MS Excel. Příklady budou voleny tak, aby v co největší míře rozvíjely algoritmické myšlení žáků a studentů. Součástí práce bude i jednoduché srovnání výhod a nevýhod programování v MS Excelu vzhledem ke standardním programovacím jazykům.

Garantující pracoviště:	Katedra informatiky, Přírodovědecká fakulta
Vedoucí práce:	doc. RNDr. Štěpán Hubálovský, Ph.D.
Oponent:	PhDr. Michal Musílek, Ph.D.
Datum zadání závěrečné práce:	1. 3. 2013
Datum odevzdání závěrečné práce:	19. 12. 2014

Prohlášení:

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a že jsem v seznamu použité literatury uvedl všechny prameny, z kterých jsem vycházel.

V Hradci Králové dne 19. 12. 2014 Tomáš Vlček

Poděkování:

Tímto bych chtěl poděkovat panu doc. RNDr. Štěpánu Hubálovskému, Ph.D., za odborné vedení, poskytnuté informace a cenné rady při zpracování mé bakalářské práce.

Anotace

VLČEK, T. *Programování v MS Excel*. Hradec Králové, 2014. Bakalářská práce na Přírodovědecké fakultě Univerzity Hradec Králové. Vedoucí práce doc. RNDr. Štěpán Hubálovský, Ph.D. 41 str.

Bakalářská práce si klade za cíl být použitelným zdrojem pro výuku programování. V teoretické části se zabývá historií tabulkových procesorů. Rozebírá několik základních pojmů, které obecně spadají pod všechny programovací jazyky, srovnává a řadí jednotlivé programovací jazyky dle jejich struktury či použití. V praktické části se zabývá tabulkovými procesory a srovnává nejpoužívanější z nich, MS Excel, s dalšími dostupnými tabulkovými procesory. Zaměřuje se na tvorbu příkladů, které budou použitelné pro rozvoj algoritmického myšlení žáků a studentů. Příklady jsou vytvářeny v tabulkovém procesoru MS Excel, který je velmi populární a díky svému velkému rozšíření dostupný většině lidí. Příklady jsou seřazeny dle obtížnosti, od nejlehčího k nejtěžšímu, čímž jsou postupně vyvíjeny na případného studenta vyšší nároky a je kladen důraz na jeho odhodlání a znalosti.

Klíčová slova

MS Excel, programování, Visual Basic for Application, programovací jazyky

Annotation

VLČEK, T. *Programming in MS Excel spreadsheet*. Hradec Králové, 2014. Bachelor Thesis at Faculty of Science University of Hradec Králové. Thesis Supervisor doc. RNDr. Štěpán Hubálovský, Ph.D. 41 str.

Bachelor thesis aims to be a useful source for teaching programming. The theoretical part deals with the history of spreadsheets. It analyzes some basic notions that are generally in all programming languages, compares and ranks the different programming languages according to their structure or use. The practical part deals with spreadsheets and compares the most used of these, MS Excel, with other available spreadsheets. It focuses on the creation of examples that will be useful for the development of algorithmic thinking of students. The examples are created in MS Excel spreadsheet, which is very popular due to its large extension and it is available to the most of people. Examples are ranked according to difficulty, from the simplest to more difficult, there are gradually developing the higher demands for potential students and the emphasis is on their resolve and understanding.

Keywords

MS Excel, programming, Visual Basic for Application, programming languages

Obsah

1 Úvod

Svou bakalářskou práci jsem se rozhodl zpracovávat na téma programování v MS Excel, nejen proto, že je rozšířeným programem od společnosti Microsoft, tedy uživatelsky lehce dostupným, ale i srozumitelným a v programátorské praxi i na základní úrovni dobře využitelným. S programováním jsem se jako student poprvé setkal na střední škole pomocí programovacího jazyku Pascal a byl to pro mě celkem těžce uchopitelný předmět. Až později na UHK jsem se začal s programováním více sžívat a vzhledem k rozšířeným znalostem, tentokrát už v programovacím jazyku Visual Basic for Application (dále jen VBA) obsaženém v programu MS Excel, jsem byl i více motivován ke zlepšování se.

VBA mi od začátku připadl pro výuku i samotnou praxi jednodušší a srozumitelnější. Díky faktu, že je součástí kancelářského balíku společnosti Microsoft, přináší jeho programům mnohé výhody. Právě MS Excel je jedním z programů, ve kterých může VBA nejvíce vyniknout. Dostatek znalostí a využití předností MS Excelu jsou pro uživatele základem pro smysluplné programování. Proto bych chtěl poukázat na důležitost znalostí samotného MS Excelu, jeho uživatelského rozhraní, nastavení a možností, které nám tento tabulkový procesor nabízí. Z toho důvodu je jedním z cílů mé bakalářské práce právě poznání prostředí MS Excel a srovnání s dalšími tabulkovými procesory.

Každý, kdo chce programovat, by měl mít alespoň minimální přehled o tom, jak se programovací jazyky dělí a kam který patří. Pomůže mu to třeba i v dalším výběru a rozvoji při učení se novým jazykům. Když už chceme programovat, musíme také vědět, jak se programy zapisují. Nejen s touto činností se seznámíme v teoretické části, kde se budeme mimo jiné zabývat i makry.

Praktická část je rozdělena na dvě kapitoly. První z nich má za cíl porovnat nejpoužívanější tabulkové procesory a poukázat na některé rozdíly mezi nimi. Druhá přináší celkem sedm praktických příkladů, které by měly být použitelné i pro výuku a rozvoj žáků či studentů při vzdělávání se v programování. Příklady jsem se snažil vybírat a programovat tak, aby obsáhly škálu obtížnosti od jednoduchých po složitější, aby byly motivací pro uživatele ve využitelnosti, byly lehce představitelné v praxi a posloužily k rozšíření uživatelových znalostí i dovedností.

K tomu, aby se z někoho stal dobrý programátor, je třeba neustále zkoušet a programovat. Jedině tím, že člověk na sobě bude soustavně pracovat, má šanci naučit se opravdu dobře programovat. Právě příklady, které naleznete v poslední části mé bakalářské práce, slouží k rozvoji algoritmického myšlení, k procvičování programátorského ducha a případně i k možnosti rozšíření zadání a jejich postupů.

2 MS Excel

Ať už jste s tímto programem někdy pracovali nebo ne, zřejmě víte, že se jedná o tabulkový procesor. To může na první pohled znít možná složitě, ale ve zkratce jednoduše vysvětlím opak: Jedná se o program, s jehož pomocí vkládáme do tabulky různá data, s kterými dále pracujeme, upravujeme je, provádíme různé výpočty, doplňujeme je o grafy apod.

2.1 Historie

Kapitola byla zpracována s využitím [1]. Historie tabulkových procesorů sahá až do konce sedmdesátých let dvacátého století, kdy v roce 1978 vytvořili první tabulkový procesor s názvem VisiCalc pánové Dan Bricklin a Bob Frankston. Ty později napodobili další počítačoví nadšenci závidějící úspěch VisiCalcu vypracováním nové koncepce tabulkových procesorů a vytvořili tak program s názvem Lotus 1-2-3, který následně ovládl trh. Tento program, na rozdíl od ostatních, přišel již s tvorbou maker, čímž uživateli nabídl zjednodušení práce. Lotus se držel dlouhou dobu na výsluní, avšak v průběhu devadesátých let nezvládl přechod na nový operační systém Windows, a tak postupně začínala éra Excelu. Dalším významným hráčem byl ještě Quattro Pro, který se poprvé objevil v roce 1987. Nabízel podobnou strukturu jako Lotus a snažil se tím získat uživatele Lotusu na svou stranu. Když už se zdálo, že Quattro bude tím nejlepším, co tabulkové procesory nabízejí, objevila se pátá verze Excelu, která tyto naděje Quattra umlčela.

Microsoft začal pracovat na tabulkových procesorech už v počátku osmdesátých let. V té době (1982) totiž vyšel jejich program jménem MultiPlan. Nebyl však zdaleka tak dobrý jako Lotus a tudíž nikoho moc nenadchl. O tři roky později (1985) vydal Microsoft první verzi nám již dobře známého Excelu. Na rozdíl od MultiPlanu byl graficky orientovaným. Tato první verze byla pro operační systém Macintosh. Verze pro Windows spatřila světlo světa až po dalších dvou letech (1987). Druhá verze obsahovala jazyk maker, který později nahradil Visual Basic for Applications (dále jen VBA), kterému se budeme především věnovat. VBA se poprvé objevil v Excelu 5. Další verze Excelu vyšla s novým operačním systémem Windows 95, kde je z názvu patrný rok vzniku 1995. MS Excel byl společně s dalšími programy (např. MS Word) součástí balíku nazvaném Office, což se osvědčilo a zůstalo i do budoucna. Následovaly další verze - Excel 97, Excel 2000, Excel 2002 a Excel 2003, které přinášely různé změny, z nichž ale žádná nebyla tak výrazná jako ve verzi Excel 2007. Ten nabízí nově přepracované uživatelské rozhraní, větší velikost mřížky, zdokonalené tabulky, apod. Poté přišla další verze v roce 2010 a prozatím poslední v roce 2013, která sází zejména na intenzivnější propojení s internetem pomocí cloudovských řešení.

2.2 Přednosti

„Excel je velmi dobře „programovatelný“ produkt, proto je pro vývojáře tabulkových aplikací tou nejlepší volbou.“[2]

Některé z hlavních předností si tu dle [3] uvedeme:

- Souborová struktura – Orientace na práci s více listy, snadná organizace prvků a jejich ukládání do jednoho souboru.

- VBA – Jazyk maker, který umožňuje vytvářet strukturované programy.

- Snadný přístup k ovládacím prvkům – Snadné přidávání ovládacích prvků přímo na pracovní listy a jednoduchost při jejich použití.

- Vlastní funkce pracovních listů – Pomocí VBA, ve kterém můžeme vytvořit libovolnou funkci, kterou pak můžeme používat stejně jako ostatní funkce v MS Excel.

- Přizpůsobitelné uživatelské rozhraní – Můžeme si např. přizpůsobit pás karet dle našich potřeb.

- Široké možnosti zabezpečení – Aplikace lze zabezpečit a chránit je tak proti nežádoucím změnám.

- Výkonná analýza dat – Kontingenční tabulky dokážou snadno sumarizovat velké objemy dat.

- Podpora automatizace – VBA nabízí možnost ovládat různé aplikace, podporující automatizaci.

3 Programování

Primární činností, pro kterou byly vyvinuty počítače, je právě programování. Mnozí z nás si to možná vůbec neuvědomují, avšak s důsledkem programování, tedy s programy, se setkáváme v dnešním světě prakticky na každém kroku. Pro příklad, kdo z nás nemá v domácnosti pračku, myčku nebo třeba mikrovlnnou troubu? A právě pod každým tlačítkem takového zařízení se skrývá program. Program, který „říká“, co se stane, zmáčkneme-li dané tlačítko. Program je jeden z druhů zápisu algoritmu. A co je to algoritmus?

Na střední škole při našich programovacích začátcích nám říkali, že algoritmus je něco jako recept. Když chceme cokoliv uvařit, vezmeme si recept a přesně podle návodu postupujeme, až dosáhneme kýženého výsledku. Přesně tak fungují i programy. Programování je, řekněme, psaní receptů pro počítač, který tedy poté postupuje přesně podle našich kroků. Záleží jen na nás, jak přesně program napíšeme a co z toho nakonec vznikne.

Programování se skládá z několika částí:

- Návrh algoritmu
- Psaní kódu
- Testování
- Ladění programu

3.1 Programovací jazyky

Programy se dají vytvářet nebo psát v různých programovacích jazycích. Všechny programovací jazyky mají povětšinou stejný základ jednoduchých i složených příkazů. Zkusím to opět přirovnat k vaření a označím programovací jazyk, např. VBA, jako italskou kuchyni. Italská kuchyně zahrnuje obrovské množství různých běžných či méně běžných ingrediencí. Některé z nich, např. těstoviny, rajčata apod., nalezneme i v dalších kuchyních. Italové však používají i různé koření, sýry a další suroviny, které jsou typické pouze pro jejich kuchyni. To samé platí i pro programovací jazyky. „*Opravdu všechny programovací jazyky obsahují takové programové konstrukce, jako jsou přiřazovací příkaz, podmíněný příkaz, příkazy cyklu, příkaz vstupu a výstupu (získávání dat od uživatele např. z klávesnice a jejich tisk buď na tiskárnu, nebo na monitor atp.). Odlišnosti jsou většinou v syntaktickém zápisu těchto konstrukcí, ale také v oboru dat, se kterými lze pomocí těchto příkazů manipulovat.*“[4]

3.1.1 Základní rysy

- Typy dat

„Data jsou to nejdůležitější, co je v počítači uloženo. Počítač vlastně nic jiného než zpracovávat data neumí.“[5] Člověk pod pojmem data chápe většinou obrázky, dokumenty ve Wordu, v Excelu a podobně. Ovšem data, se kterými pracují programovací jazyky, jsou poněkud rozdílná. *„Data jsou oním materiálem, surovými informacemi, se kterými manipuluje váš program. Bez dat nemůže program provádět žádnou užitečnou činnost. Programy mohou s daty manipulovat různými způsoby. Často to závisí na typu dat. S každým datovým typem je spojena skupina operací — tj. činností, které lze s daty daného typu provést.“*[6]

- Proměnné a konstanty

„Proměnné ve Visual Basicu – stejně jako v jiných programovacích jazycích – slouží k dočasnému uchování hodnot během vykonávání programového kódu.“[7] Proměnnou je nejprve třeba deklarovat. Pokud bychom tak neučinili, VBA by ji sice deklaroval automaticky, ale jako datový typ Variant, což je nejnáročnější datový typ na paměť. Deklaraci provádíme vždy před tím, než ji prvně použijeme.

„Konstanta představuje symbolický název, kterému je přiřazen řetězec nebo číslo, které se nemění.“[8] Což je zřejmé už ze samého názvu. Velké množství konstant je již definováno samotným programem. Konstanty jsou všechny veřejné, takže je možné je použít v jakékoli části programu. Dále je možné si jakoukoli konstantu samostatně definovat za použití klíčového slova „Const“. Konstanty nám slouží k tomu, abychom mohli jednoduše používat v programu se několikrát opakující hodnotu.

- Procedury a funkce

Procedura je posloupnost příkazů, v našem případě VBA. Procedura pomocí těchto příkazů vykonává požadovanou úlohu. *„Většina kódu VBA se nachází právě uvnitř procedur. Proceduru můžete volat, neboli spouštět či vykonávat, celou řadou způsobů. Procedura je prováděna od začátku až do konce (může však také být ukončena předčasně).“*[9]

Funkce je procedura VBA, která vykonává jakýsi výpočet a vrací určitou hodnotu. MS Excel má v samotném základu mnoho funkcí dopředu vytvořených. Ty se dělí do skupin podle jejich konkrétního užití. Jedná se např. o skupiny Logické, Finanční, Statistické. Jako příklad Logických můžu uvést funkci KDYŽ, A nebo PRAVDA. Uživatelé si však mohou ve VBA vytvořit i své vlastní funkce, které lze použít v kódu VBA nebo ve vzorcích na pracovním listu.

3.1.2 Dělení programovacích jazyků

Programovacích jazyků je v dnešní době tolik, že o nich člověk pomalu ztrácí přehled. Proto si zde uvedeme několik základních rozdělení programovacích jazyků i s příklady, které do oněch kategorií spadají. Mezi nejobvyklejší způsoby, na které se programovací jazyky dělí, jsou tyto:

- Vyšší a nižší

Nižší programovací jazyky jsou označovány jako nižší proto, že *„poskytují pouze malou nebo žádnou abstrakci od toho, jak funguje procesor počítače.“*[10] Jinými slovy - strojové instrukce procesoru jsou jen minimálně odlišné od jakéhokoli nižšího programovacího jazyka. To je i důvod, proč jsou nižší programovací jazyky pro programátory obtížné, protože, aby v nich člověk mohl programovat, musí vědět, jak po technické stránce funguje hardware. Výhodou těchto programovacích jazyků je naopak přístup programátora k funkcím, které jsou ve vyšších nedosažitelné. V prvních nižších programovacích jazycích se psalo ve strojovém kódu, druhá generace těchto programovacích jazyků nabízí takzvaný assembler, čili jazyk symbolických instrukcí. Použití těchto jazyků je dnes spíše výjimečné, např. pro psaní jader operačních systémů.

Co se vyšších programovacích jazyků týče, jsou blíže lidskému myšlení – struktura zdrojového kódu je logická. Oproti nižším jsou obvykle kratší a více srozumitelné. To umožňuje programátorovi nejen jednodušší vývoj, ale i kratší čas, ve kterém vytvoří program a zároveň hrozí menší riziko chybovosti. Mezi další výhody patří např. přenositelnost, díky níž není problém použití programu na jiném počítači. Nevýhodou naopak je, že aby program napsaný vyšším programovacím jazykem fungoval, musí být přeložen neboli kompilován do nižšího jazyka. Mezi vyšší programovací jazyky patří *„v podstatě všechny programovací jazyky kromě Jazyka symbolických adres (často nesprávně označován jako Assembler) a strojového kódu.“*[11]

Vyšší programovací jazyky se pak následně dělí na imperativní (procedurální) a neimperativní (neprocedurální).

  - Imperativní, neimperativní

Už výše jsem se zmiňoval o receptech či návodu pro určitou činnost, to platí právě i pro imperativní programovací jazyky. V imperativním programování totiž píše programátor přesný postup (algoritmus), jak se má daná úloha vyřešit. Imperativnímu způsobu se též říká procedurální. A to z toho důvodu, že program bývá psán pomocí procedur. Tyto programovací jazyky jsou díky své jednoduchosti a logičnosti pochopitelnější pro lidi a tudíž i nejrozšířenější a nejpoužívanější. Mezi velmi populární imperativní programovací jazyky patří např. C, VBA, PHP, Java a další, které stejně jako ostatní pracují s proměnnými. Imperativní programování

používá tři základní typy, které slouží k zápisu algoritmů. Jedná se o přiřazení, které pracuje s proměnnými z paměti a následně je do paměti opět ukládá. Druhým příkazem jsou cykly, které umožňují opakování určitého kódu za sebou a posledním příkazem jsou příkazy pro větvení, které zhodnotí podmínky a následně podle nich vyberou příkaz, jenž má být proveden.

Neimperativní jazyky, někdy též deklarativní, jsou jazyky logické a funkcionální. Jedná se o programování, které je *„založeno na myšlence programování aplikací pomocí definic, co se má udělat a ne, jak se to má udělat.“*[12] U logických programovacích jazyků, jak už z názvu můžeme vyvodit, programátor pracuje s logickými výroky. Pomocí těchto výroků je napsán program, který si z nich poté vyvodí potřebný výsledek. Jako příklad logických programovacích jazyků mohu uvést např. Prolog.

Co se funkcionálních programovacích jazyků týče, jsou specifické absencí proměnných, kdy se místo nich pracuje se seznamy dat. Program, napsaný pomocí funkcionálních jazyků, je složité uskupení funkcí. Tento typ programovacích jazyků je velmi podobný matematice. Je důležité říci, že funkce v imperativním programování a funkce, se kterými pracuje funkcionální programování, jsou odlišné. Pro zjednodušení je daný rozdíl v tom, že stejné volání funkce může vést u imperativního programování k různým hodnotám, kdežto u funkcionálního programování, kde záleží pouze na argumentech funkce, vede vždy ke stejné hodnotě. Příkladem funkcionálních jazyků je např. Haskell nebo Scala.

- Interpretované a kompilované

Posledním způsobem dělení programovacích jazyků je dělení na interpretované a kompilované programovací jazyky. Jaký je mezi nimi rozdíl?

Interpretované programovací jazyky jsou jazyky překládány až za běhu programu. Pro jejich spuštění je nutné mít tzv. interpret, čili program, který dané instrukce za běhu programu také provádí. Výhodou interpretovaných jazyků je *„snadné provádění úprav v programu, snazší hledání a odstraňování chyb a obvykle dobrá přenositelnost programu na jinou platformu.“*[13] Naopak nevýhodami jsou v prvé řadě nižší rychlost programu a též nutnost mít onen interpret pro spuštění programu.

Naproti tomu kompilované programovací jazyky žádný interpret nepotřebují. Tyto jazyky, jak uvádí [14], musí být pro možnost spuštění nejprve celé přeloženy pomocí kompilátoru do strojového kódu. Největšími výhodami je jednak vysoká rychlost a zadruhé možnost spustit program na jakémkoli jiném počítači, protože programy jsou většinou překládány do souboru s koncovkou exe. Nevýhodou pak je, že při změně jakékoli části kódu musíme provést znovu překlad.

3.1.3 Srovnání programovacích jazyků

Obecně se s jistotou nedá říci, který programovací jazyk je lepší či horší. Dá se však říci, který programovací jazyk se na kterou činnost více hodí. Jsou jazyky podporované na všech platformách, jako např. jazyk C, ale jsou také specializované jazyky vytvořené pro konkrétní aplikaci. Také se dá říci, které jazyky se v dnešní době již nepoužívají a které jsou naopak nejrozšířenější. Nás nejvíce zajímá Visual Basic, resp. Visual Basic for Applications. Do jaké kategorie programovacích jazyků VBA patří?

3.1.4 Jazyk Visual Basic for Applications

„Visual Basic byl vyvinut firmou Microsoft proto, aby zjednodušil programování jednodušších i složitějších aplikací a zpřístupnil tak umění programovat široké veřejnosti.“[15] VBA se vyznačuje velkou schopností komunikace uživatele s programem. *„Výhodou tohoto jazyka je jeho jednoduchost, podpora a dostupnost. Najdeme jej například u aplikací: MS Office, MS MapPoint, MS Visio, AutoCAD, Invertor, 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. 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 nechuť jeho nasazení na konkurenčních platformách, přesto se tak částečně děje.“*[16]

Když bychom si našli jazyk VBA v předchozím dělení programovacích jazyků, zjistíme, že patří do kategorie vyšších programovacích jazyků, dále do imperativních programovacích jazyků a také do interpretovaných programovacích jazyků. Vezmeme-li kategorii imperativních jazyků a půjdeme hlouběji, dostaneme dvě další možné podkategorie dělení těchto jazyků. Nás bude zajímat právě ta, ve které se nachází VBA. Jmenuje se Objektivně orientované programování.

V MS Excel pak platí, že prvky objektového programování neboli objekty jsou např.: Samotná aplikace Excelu, Sešit v Excelu, List v sešitu, Oblast na listu, List grafu, apod.

3.2 Algoritmus

Již jsem se výše zmiňoval o tom, co algoritmus je, ale dovolím si sem přidat i definici:

Algoritmus:

- *„Je konečná posloupnost/uspořádání postupů aplikovaných na konečný počet dat, jež dovoluje řešit přibližně stejné třídy problémů.“*[17]

- *„Souhrn postupů charakteristických pro jisté výpočty nebo informatické funkce.“*[17]

Základní znaky algoritmu jsou:

- **Konečnost** – Algoritmus po konečném počtu kroků končí. Počet kroků může být libovolně velký, velikost určují vstupní údaje.

- **Obecnost** – Algoritmus neřeší jeden konkrétní problém, je použitelný pro obecná řešení.

- **Determinovanost** – Pro každý krok musí být jasně dané, co se má vykonat teď i v dalším kroku.

- **Výstup** – Algoritmus musí mít alespoň jeden výsledek, který odpovídá na námi zadaný problém.

- **Elementárnost** – Kroky, z kterých je tvořen algoritmus, musí být jednoduché, resp. základní.

3.2.1 Historie algoritmů

Původ samotného slova algoritmus je stále ještě ne zcela jasný. Nejpravděpodobnější verze je, že toto slovo pochází z příjmení spisovatele a matematika Abu Jafara Muhammada ibn Musa al-Khwarizmiho pocházejícího z Persie. Ten *„je tvůrcem jasných regulí, které krok za krokem objasňují postup aritmetických operací na desetinných číslech.“*[17] Možná si řeknete, z které části jeho jména je to tedy odvozené. Když bychom si jeho jméno ale přeložili do latiny, pak bychom slovo Algoritmus dostali. Algoritmus se však objevil už dříve před zmíněným perským matematikem. Postupy, které již můžeme nazvat algoritmem, se objevily už v období starého Řecka. Jako důkaz nám slouží takzvaný Euklidův algoritmus, tedy výpočet největšího společného dělitele.

Rozvoj algoritmizace se však odehrával především za posledních dvě stě let, v období technické revoluce. Jeden z prvních, kdo algoritmy dokázali využít, byl Joseph Maria Jacquard, vynálezce tkalcovského stavu, který dokázal tkát dle vzoru pomocí děrných štítků. Na stroji pracoval velmi dlouhou dobu, než se mu podařilo dosáhnout kýženého výsledku. Jednalo se o stroj vyrábějící rybářské sítě někdy na počátku devatenáctého století. Práce J. M. Jacquarda zaujala některé z dalších matematiků či vědců a jedním z nich byl pozván do Paříže. Komu se toto zlepšení nelíbilo, byli tkalci, jelikož tento stroj nahrazoval pět pracovních míst. Stojí za to podotknout, že *„děrný štítek opatřený kódem dvoustavové logiky … byl předchůdcem současných pamětí, kde se binární čísla používají pro zapsání řídícího kódu algoritmu!“*[18] Velký stroj využívající děrné štítky byl poprvé použit na konci

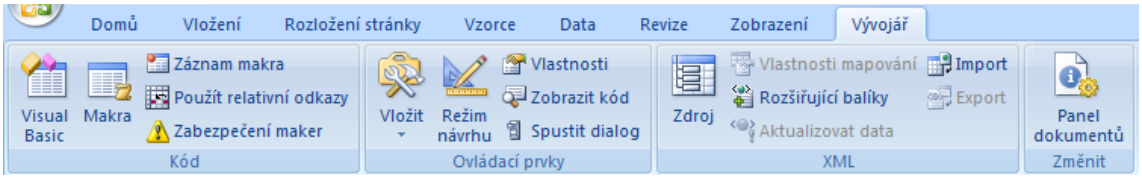
19. století. Jeho autorem byl Američan Herman Hollerith. Z Hollerithova podniku se v roce 1911 stal nám již známý IBM.

Ve dvacátém století se toho stalo mnohem více, především díky vzniku prvních počítačů v největší míře pracujících s algoritmy. Z počátku vznikaly pro vojenské účely, dále pro vědecké a poté i pro účely studijní. S tímto rozvojem poté vznikaly nové a nové algoritmy.

3.3 Makra

Makro je nejjednodušším programem ve VBA. Pokud pracujeme v MS Excel a neustále opakujeme určitou činnost, makro nám může velmi zjednodušit naši práci. V Excelu si totiž můžeme naši činnost pomocí makra zaznamenat a poté, dle potřeby, makro vyvolat, čímž dostaneme posloupnost akcí předtím vytvořených.

Pro práci s makry slouží panel Vývojář. Pokud se u vás nenachází mezi ostatními panely, je třeba jej v nastavení zapnout. Najdete jej zde: **Soubor – Možnosti – Přizpůsobit pás karet – Vývojář**.



Obr. 1 Panel – Vývojář

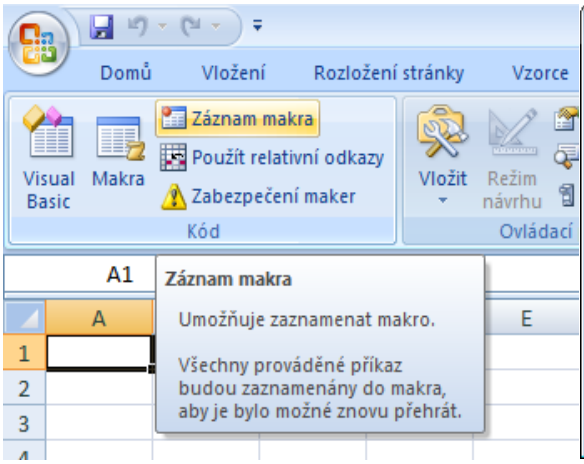
3.3.1 Tvorba Maker

Tato část je vypracovaná pomocí [20].

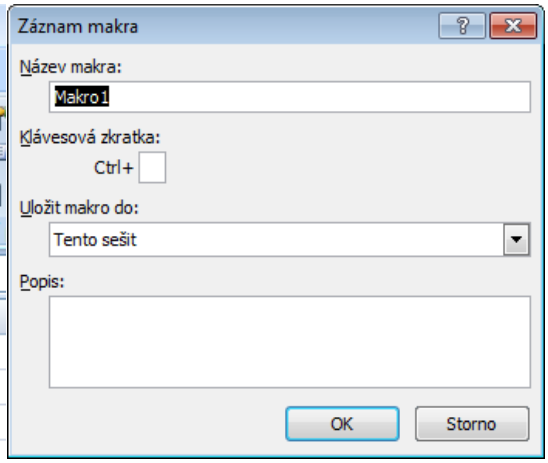
Pokud se zaměříme na prostředí MS Excel, můžeme makra vytvářet třemi základními metodami:

1. záznamem činností

Jedná se o nejjednodušší činnost. Je to podobné, jako kdybychom nahrávali video. V prvním kroku zapneme nahrávání zaznamenávající všechnu naši práci. Tímto způsobem může makro obsahovat jen akce, které si uživatel může udělat i sám pomocí nástrojů.



Obr. 2 Záznam makra 1

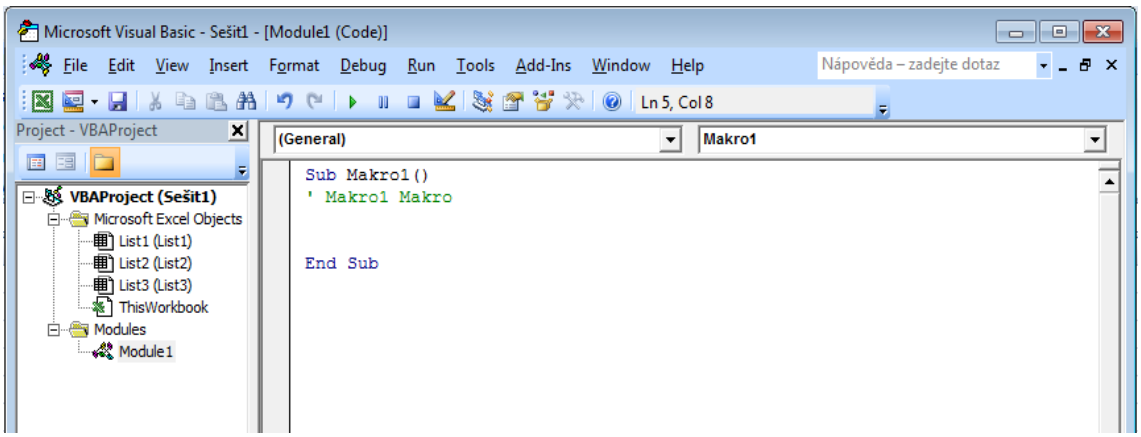


Obr. 3 Záznam makra 2

2. programovým zápisem

Když nahráváte makro, tvoříte tím i kód, který se zapisuje ve VBA. Když si otevřete editor jazyka VBA, kód si můžete zapsat sami. To s sebou nese několik výhod.

- *„Zpracovávat údaje na listu způsobem, který by při uživatelské práci byl velmi pracný nebo nemožný;*
- *vytvářet nové vlastní funkce;*
- *spouštět automatické akce při otevření a zavření sešitu;*
- *vytvářet vlastní dialogová okna pro zadávání vstupních údajů.“*[19]



Obr. 4 editor jazyka VBA

3. kombinací záznamu a programového zápisu

Samozřejmostí je také, že uživatel může kombinovat dvě výše uvedené možnosti dohromady.

3.3.2 Spuštění makra

Abychom spustili nějaké makro, můžeme využít hned několika možností:

- Tlačítkem na panelu Rychlý přístup
- Klávesovou zkratkou
- Tlačítkem na některé z karet
- Tlačítkem na listu

4 Praktická část

4.1 Srovnání MS Excel s dalšími tabulkovými procesory

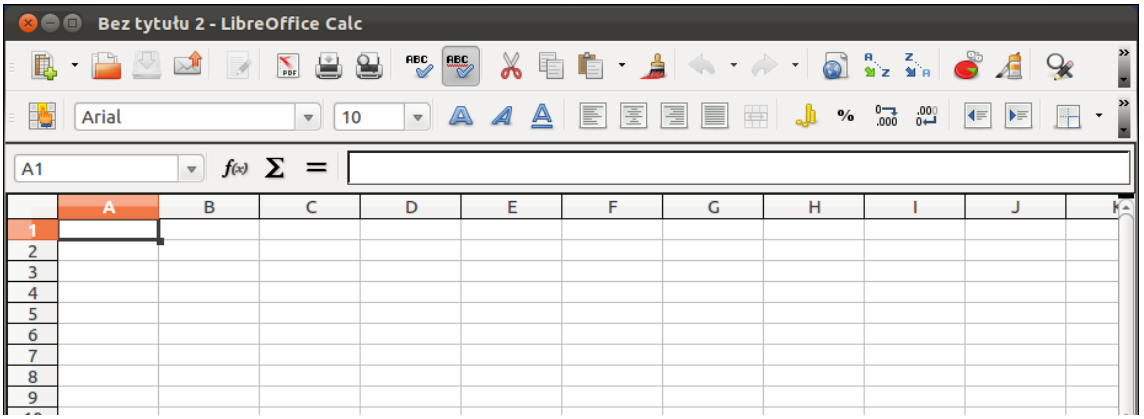
V následující kapitole se budeme věnovat srovnání MS Excel, případně celého kancelářského balíku MS Office, s dalšími programy nabízejícími alternativu za tento software. Těmito dalšími programy budou OpenOffice a LibreOffice. OpenOffice již funguje poměrně dlouhou dobu, zatímco LibreOffice vznikl teprve nedávno, odštěpením právě od OpenOffice. Díky tomu jsou si tyto dva programy částečně podobné. Vzhledem k neustálému vývoji všech těchto programů jen upřesním, že mé srovnání se týká verzí MS Excel 2010, OpenOffice 3.3 a LibreOffice 3.4.

4.1.1 Náročnost programů

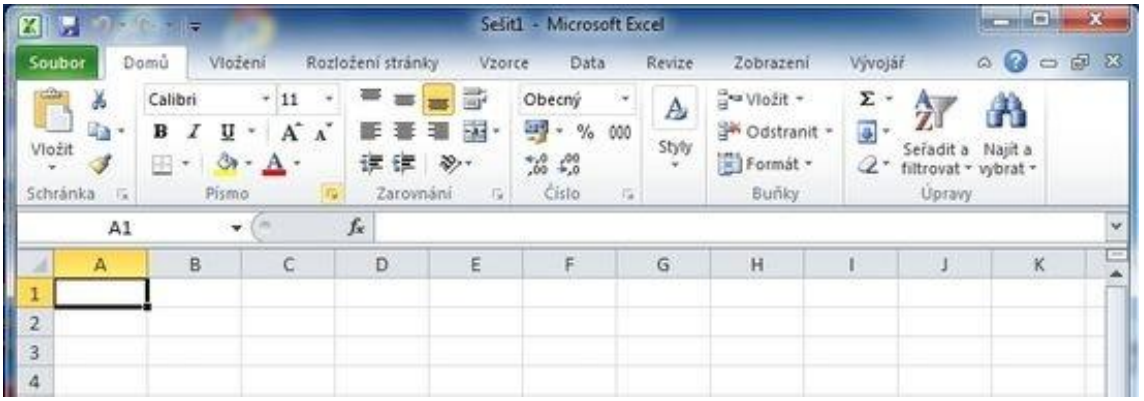
Kapitola byla vypracovaná pomocí [21], [22],[23].
Už z názvů těchto dvou naposledy zmíněných balíků můžeme vyčíst, že se jedná o svobodný software šířený pod licencí LGPL. Můžeme jej tedy volně stahovat, používat, upravovat či distribuovat. Oproti tomu MS Excel je proprietární software, což znamená, že jeho licencí autor definuje jeho používání. Takovýto software bývá povětšinou komerční, což je právě i případ MS Excelu. Jeho cena, resp. cena celého balíku MS Office (protože MS Excel se sám o sobě koupit nedá), začíná zhruba na 2000 Kč. Co se týče velikostní náročnosti programů, tak nejmenší velikost instalátoru nabízí OpenOffice s velikostí 129 MB, druhý je se 189 MB LibreOffice a nejvíce má MS Office s 900 MB. Mezi podporované operační systémy patří u všech těchto tří programů MS Windows a Mac OS X. U obou svobodných je to dále ještě Linux, popř. neoficiálně i další operační systémy.

4.1.2 Ovládání

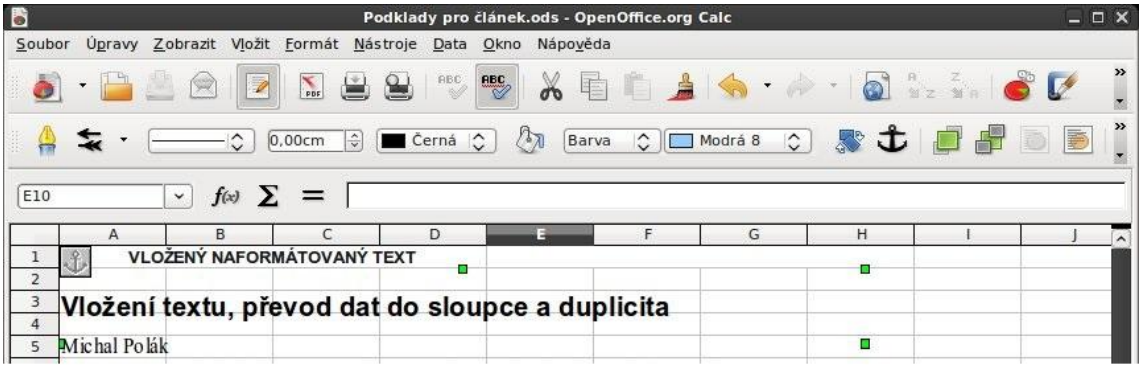
Co se týče ovládání, tak tady je rozdíl jasně vidět. MS Excel se ovládá pomocí pásu karet, zatímco LibreOffice a OpenOffice mají oba panel nástrojů. Pás karet se objevil poprvé ve verzi 2007, předtím se MS Excel ovládal taktéž pomocí panelu nástrojů. Já osobně jsem se tehdy delší dobu bránil přechodu na novou verzi, protože se mi upřímně moc nechtělo zkoumat nové rozložení ovládání. Dnes na něj ovšem nedám dopustit. I proto musím říci, že, ať už je to tak nebo jinak, vždy jde o zvyk každého z uživatelů. Toto základní ovládání můžeme shlédnout na následujících obrázcích těchto programů.



Obr 1. LibreOffice



Obr. 2 Excel 2010



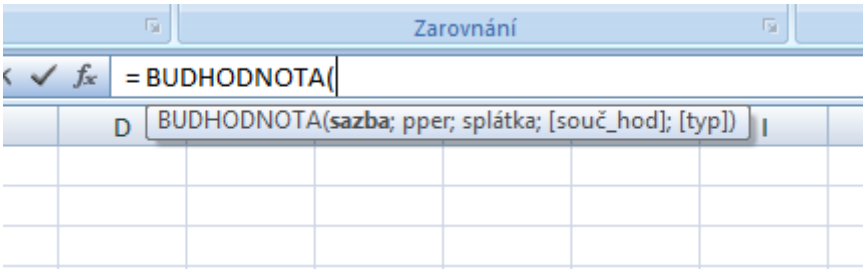
Obr. 3 OpenOffice

4.1.3 Funkce

Česká verze MS Excelu nabízí část funkcí přeložených též do českého jazyka. Bohužel ne všechny funkce jsou takto přeloženy, a proto u některých musíte zapojit své znalosti angličtiny a pustit se do překladu sami. To program Calc, jak se jmenuje tabulkový procesor v OpenOffice i v LibreOffice, nabízí pouze anglickou verzi funkcí, respektive funkce pouze v anglickém jazyce. Při srovnávání funkcí v MS Excelu a Calcu zjistíme, že pokud vezmeme anglický název nějaké funkce z MS Excelu, najdeme ji pod stejným názvem i v Calcu. Některé funkce, jako

např. SUMIFS nebo COUNTIFS bychom však v Calcu hledali marně. Jednodušší funkce tedy Calc zvládá naprosto srovnatelně, s těmi obtížnějšími může mít ale problémy, ovšem i ty se dají vyřešit nahrazením jinou metodou.

Pokud budete v MS Excelu psát funkci, jistě si všimnete, že vyskočí políčko s nápovědou, jakou část kódu dále psát. Tato poměrně pohodlná vlastnost však Calcu úplně chybí.



Obr 5. Nápověda funkce

4.1.4 Shrnutí

S určitostí mohu říci, že ať bychom se snažili sebevíce, trvalo by nám velmi dlouhou dobu, než bychom zjistili všechny odlišnosti MS Excelu a svobodných programů. I z tohoto mála je snad jasné, že pokud jste s programem, který používáte, spokojeni, není třeba přecházet na jiný. Běžný uživatel si bez problémů vystačí s OpenOffice nebo LibreOffice, které se z velké části s MS Excelem vyrovnají a v něčem ho i předčí. MS Excel je však propracovanější, uživatelsky přívětivější a nabízí přeci jen o něco více než jeho výše zmínění kolegové.

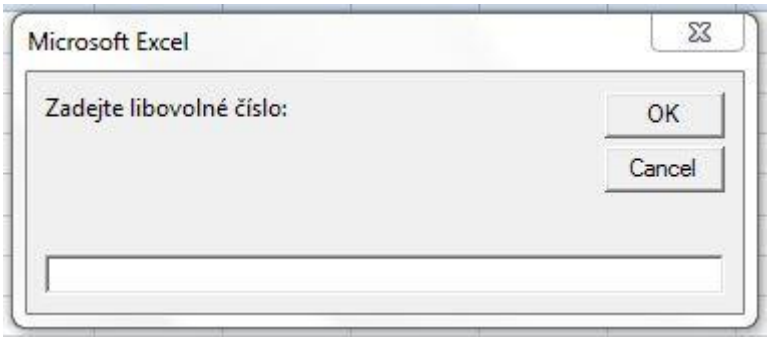
4.2 Programy

Kapitola s názvem Programy obsahuje několik řešených příkladů, které nám demonstrují použití VBA v MS Excel. Setkáme se zde s příklady různých stupňů obtížnosti, od jednodušších až po složité.

4.2.1 Prvočíslo

V prvním řešeném příkladě se zabýváme hledáním prvočísla.

Zadání
Vytvořte program, na jehož vstupu zadáte libovolné celé číslo, které program následně vyhodnotí a podá zprávu, jestli je číslo prvočíslem nebo nikoliv.



Obr. 6 Prvočíslo vstup

Zdrojový kód

```
Sub Prvocislo()
Dim B As Integer, C As Integer, CisloVstup As Integer
CisloVstup = InputBox("Zadejte libovolné číslo:")
B = CisloVstup - 1

Do Until B = 1
        C = CisloVstup Mod B
        B = B - 1
        If C = 0 Then
                MsgBox ("Číslo " & CisloVstup & " není prvočíslo")
                B = 1
        Else
                 If B = 1 Then
                        MsgBox ("Číslo " & CisloVstup & " je prvočíslo")
                Else
                End If
        End If
Loop
End Sub
```

Popis programu

Po spuštění programu je uživatel pomocí *InputBoxu* vyzván k zadání čísla, o kterém potřebuje zjistit, jestli je prvočíslem nebo nikoliv. Číslo se následně načte do proměnné s názvem CisloVstup.

V dalším kroku se do proměnné B načte hodnota proměnné CisloVstup, zmenšená o hodnotu jedna.

Následuje cyklus *Do Until*, který nám říká, co udělat, dokud se B nebude rovnat jedné. V tomto cyklu se nachází několik příkazů, mezi nimi i dvě *podmínky If*. Nejprve však dochází k přiřazení hodnoty do proměnné C., do které se poté načítá výsledná hodnota po celočíselném dělení proměnné CisloVstup proměnnou B. Nakonec dochází ke zmenšení proměnné B opět o hodnotu jedna.

A nyní se již dostáváme k první *podmínce If* udávající, co se stane, bude-li proměnná C rovna nule. Když by tomu tak bylo, pomocí *MsgBoxu* nám bude sděleno, že naše zadané číslo (CisloVstup) není prvočíslo a do proměnné B se načte číslo 1, čímž se následně cyklus ukončí. V případě opačném, kdy se proměnná C nebude rovnat nule, nastává chvíle vysvětlení významu Else. V našem případě je to další *podmínka If*, která říká, co dělat, když se proměnná B rovná jedné a co dělat v případech ostatních. Pokud je tedy B rovno jedné, pak se pomocí *MsgBoxu* dozvíme, že naše číslo (CisloVstup) je prvočíslo.

Program je nastaven tak, že zadané číslo začneme postupně dělit. Dělíme ho nejprve číslem o jedna menším než zadané číslo. Následně zmenšíme dělitele opět o jedna a postup opakujeme až do hodnoty 2. Nás však nezajímá výsledek tohoto celočíselného dělení, jako především jeho zbytek. Ten je v každém kroku porovnáván, a pokud by jeho hodnota nabyla hodnoty 0, můžeme říci, že dané číslo není prvočíslem. Pokud však v průběhu tohoto cyklu zbytek nenabyl hodnoty 0, můžeme naopak s určitostí říci, že naše vložené číslo prvočíslem je.

Obr. 7 Prvočíslo výstup

4.2.2 Číselné kombinace

Zadání

Vytvořte program tak, aby nám, když na jeho vstupu zadáme libovolná tři čísla, vypsal všechny možné kombinace těchto čísel.

Zdrojový kód

```
Sub Ciselne_kombinace ()
Dim A As Integer, B As Integer, C As Integer
Dim X As Integer, Y As Integer

A = Cells (1, 1)
B = Cells (1, 2)
```

```
C = Cells (1, 3)
X = 2
Y = 1

Do While X < 8
        Cells (X, 1) = Cells (1, Y)
        Cells (X + 1, 1) = Cells (1, Y)

        If Y = 1 Then
                Cells (X, 2) = B
                Cells (X, 3) = C
                Cells (X + 1, 2) = C
                Cells (X + 1, 3) = B
        ElseIf Y = 2 Then
                Cells (X, 2) = A
                Cells (X, 3) = C
                Cells (X + 1, 2) = C
                Cells (X + 1, 3) = A
        Else
                Cells (X, 2) = A
                Cells (X, 3) = B
                Cells (X + 1, 2) = B
                Cells (X + 1, 3) = A
        End If

        X = X + 2
        Y = Y + 1

Loop
End Sub
```

	A	B	C	D	E
1	1	3	5	Místo pro zadání 3 čísel	
2	1	3	5		
3	1	5	3		
4	3	1	5	Všechny možné kombinace výše zmíněných čísel	
5	3	5	1		
6	5	1	3		
7	5	3	1		

Obr. 9 Číselné kombinace

Popis programu

Aby program mohl pracovat, musíme nejprve zadat do buněk A1, B1 a C1 tři čísla, se kterými budeme dále pracovat.

Program nejprve načte hodnoty z buněk A1, B1 a C1 do proměnných A, B a C, takže pod každou proměnnou se nám nyní skrývají naše zadaná čísla.

Dále se nám tu nacházejí proměnné X a Y, které slouží jako pomocné proměnné pro pohyb mezi jednotlivými buňkami.

Abychom prošli všechny možnosti, máme k dispozici *cyklus Do While* obsahující několik podmínek. Cyklus je závislý na proměnné X, která se při každém průchodu cyklem navýší o hodnotu dvě, a to až do té doby, kdy dosáhne hodnoty osm. V průběhu cyklu se nám také mění proměnná Y po průchodu měnící svou velikost o hodnotu jedna.

Podmínky If, které jsou obsažené v cyklu, slouží k vložení správné číselné kombinace do správného řádku. Program začíná vždy s prvním sloupcem, do kterého vkládá nejprve do prvních dvou řádků stejné číslo a následně podle této první hodnoty doplní dvě kombinace zbylých dvou čísel pro daný řádek. Poté program vyplní v prvním sloupci další dva řádky, tentokrát však druhým číslem a následně doplní dvě možné kombinace ze zbylých dvou čísel atd.

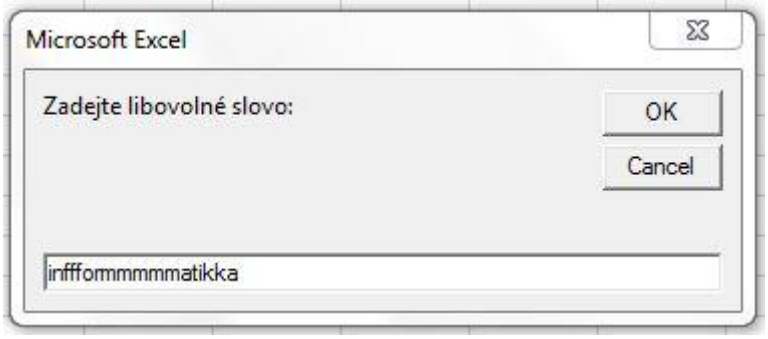
4.2.3 Komprese textu

Zadání

Vytvořte program, kde při vstupu zadá uživatel libovolný text a program následně vypíše daný text zpátky, ale rozdělený na jednotlivá písmena. Pokud by v textu bylo více stejných písmen za sebou, program vypíše pouze jedno toto písmeno a před něj vloží číslo značící počet písmen vypsaných vícekrát za sebou.

Př. Vstup = infffformmmmmatikka

Výstup = in3for5mati2ka



Obr. 10 Komprese vstup

Zdrojový kód

```
Sub Komprese ()
Dim VlozenyText As String, Text As String
Dim X As Integer, Delka As Integer, C As Integer, I As Integer

VlozenyText = InputBox("Zadejte libovolné slovo:")
Delka = Len(VlozenyText)
X = 1

For I = 1 To Delka
        Text = Mid(VlozenyText, I, 1)
        If text = Mid(VlozenyText, I + 1, 1) Then
                C = 1
                Do While Mid(VlozenyText, I, 1) = Mid(VlozenyText, I + 1, 1)
                        I = I + 1
                        C = C + 1
                        Cells(X, 1) = C & Text
                Loop
        Else
                Cells(X, 1) = Text
        End If
X = X + 1
Next I
End Sub
```

Popis programu

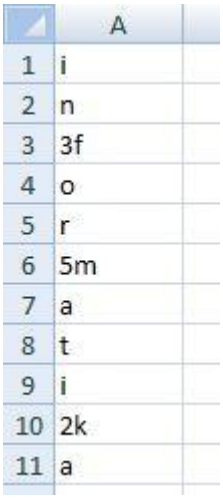
Náš program začíná opět zadáním určitých dat od uživatele prostřednictvím *InputBoxu*. Co uživatel napíše do *InputBoxu*, to se nám automaticky vloží do proměnné s názvem VlozenyText. Dále pomocí funkce LEN zjistíme, jak dlouhé je ono zadané slovo (VlozenyText). Tato hodnota se uloží do proměnné Delka a poté se přidělí proměnné X hodnota 1.

Následuje *For cyklus*, který začíná na I = 1 a bude probíhat do té doby, dokud I nedosáhne hodnoty délky slova (Delka). V těle cyklu se přiřadí proměnné Text první písmeno z našeho slova (VlozenyText). Přiřazení probíhá pomocí *funkce Mid.*

Cyklus dále obsahuje *podmínku If,* která porovnává proměnnou Text (tzn. první písmeno textu) s písmenem následujícím (tzn. druhé písmeno textu). Pokud dojde ke shodě, pak následuje přiřazení proměnné C hodnoty 1 a dále *cyklus Do While.* Ten porovnává další písmena, která jsou v pořadí a v případě shody jednak zvyšuje proměnnou I o hodnotu 1, za druhé zvyšuje proměnnou C o hodnotu 1 a za třetí do

buňky (X, 1) zapisuje hodnotu C a písmeno, se kterým právě pracuje (Text). V případě, že *podmínka If* nebude splněna, do buňky (X, 1) se zapíše hodnota uložená v proměnné Text.

V těle *For cyklu* se nachází také navýšení proměnné X o hodnotu o jednu větší (X = X + 1). Poté dojde k navýšení proměnné I taktéž o hodnotu jedna, a jak jsem již psal výše, opakuje se celý cyklus do doby, dokud I nedosáhne stejné hodnoty jako je délka slova Delka.



	A
1	i
2	n
3	3f
4	o
5	r
6	5m
7	a
8	t
9	i
10	2k
11	a

Obr. 11 Komprese výstup

4.2.4 Práce s polem

Zadání

Vytvořte program, který se vás při vstupu zeptá, kolik čísel chcete vložit do pole. Následně, po vložení čísla do pole, vám program vypíše největší číslo uložené v poli a počet, kolikrát se zjištěné číslo v Excelu vyskytuje.

Zdrojový kód

```
Sub Nejvetsi_cislo_v_poli()
Dim Pole() As Integer, PocetCisel As Integer, I As Integer, C As Integer
Dim Nejvetsi As Integer, X As Integer

PocetCisel = InputBox("Zadejte počet čísel, které chcete vložit do pole:")

Do While I < PocetCisel
        X = InputBox("Zadejte číslo do pole:")
        ReDim Preserve Pole(I)
        Pole(I) = X
        If I = 0 Or X > Nejvetsi Then
                Nejvetsi = X
        End If
        I = I + 1
Loop
```

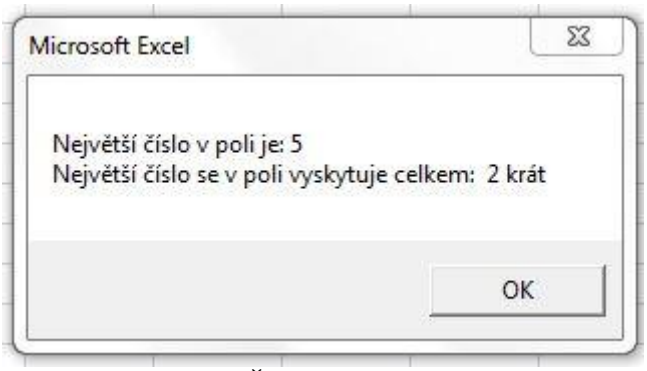
```
For I = 0 To PocetCisel - 1
        If Pole(I) = Nejvetsi Then
        C = C + 1
        End If
Next I

MsgBox ("Největší číslo v poli je: " & Nejvetsi & vbCrLf & "Největší číslo se v poli
vyskytuje celkem: " & Str(C) & " krát")

End Sub
```

Popis programu

Tentokrát nás na začátek nečeká pouze jeden *InputBox*, ale hned několik. Při prvním vstupu budeme do *InputBoxu* zadávat, kolik čísel budeme následně chtít vložit do Pole. Podle druhu zadané hodnoty (proměnná PocetCisel) se nás následně *InputBox* „zeptá“ na další číslo, které chceme do Pole vložit.

Vkládání je vytvořené pomocí cyklu *Do While* a platí, že cyklus bude probíhat, pokud I bude větší než PocetCisel. V těle cyklu se pak nachází *InputBox*, pomocí kterého se do proměnné X přiřazují čísla vložená uživatelem. Při každém průchodu cyklem se tedy za proměnnou X dosadí nové číslo, které uživatel napíše do *InputBoxu.* Aby se však pouze neměnila čísla v proměnné X, ale byla zároveň někam ukládána, nachází se v těle cyklu vkládání do Pole. Před ukládáním je však třeba Pole definovat pomocí *ReDim Preserve Pole* (I), kde I udává velikost pole. V těle cyklu se dále nachází *podmínka If* udávající, co se má provést, je-li I rovno nule nebo pokud aktuální hodnota Pole (X) je větší než proměnná Nejvetsi. Pokud je tedy podmínka splněna, pak proměnná Nejvetsi je nahrazena hodnotou Pole (X). Tím zjišťujeme a porovnáváme největší číslo v poli. Poté se nám zvětší proměnná I o jedna a cyklus může pokračovat dál, samozřejmě jen v případě stále platné podmínky.

Následuje další cyklus, tentokrát *For cyklus* umožňující zjištění, kolikrát se největší číslo vyskytuje v Poli. Pomocí podmínky si nadefinujeme projití celého Pole. Je tedy nutné nastavit I na nulu, protože pole začínají právě nulou. Následně budeme procházet pole až do hodnoty PocetCisel zmenšené o jedna. Zmenšené právě proto, že začínáme nulou. V cyklu je pak *podmínka If* porovnávající aktuální číslo v poli s proměnnou Nejvetsi, ve které je v tu chvíli uložené největší číslo z Pole. Pokud by některé z čísel z Pole splnilo podmínku, pak se nám proměnná C zvýší o hodnotu jedna. Po skončení cyklu *MsgBox* vypíše největší číslo z Pole a také počet výskytů v Poli.



Obr. 12 Číselné kombinace

4.2.5 Formulář

Zadání

Vytvořte jednoduchý formulář, ve kterém budou dvě tlačítka (*Command Button*) a jeden štítek (*Label*). V jednom tlačítku bude napsáno „Rychle klikni.“, ve druhém „Zkus to tady.“ a ve štítku bude napsáno „Pozdě, zkus to znovu!“. Po spuštění formuláře uvidíme pouze první tlačítko s nápisem „Rychle klikni.“. Pokud najedeme kurzorem myši na toto tlačítko, pak toto tlačítko zmizí, ale objeví se nám druhé tlačítko s nápisem “Zkus to tady.“. Stejně tak tomu bude i obráceně, když najedeme myší na druhé tlačítko, které následně zmizí a objeví se první. Najedeme-li podruhé kurzorem myši na první tlačítko, objeví se i štítek s názvem „Pozdě, zkus to znovu!“. Když však počtvrté přemístíme kurzor myši nad druhé tlačítko, ve štítku se změní nápis na „Konec? Pro ukončení klikni zde.“. Když tedy klikneme na štítek, formulář se ukončí. Jinak můžeme pohybovat myší z tlačítka na tlačítko, jak dlouho chceme.

Zdrojový kód

a) Deklarace proměnných

```
Dim A, B As Integer
```

b) První tlačítko

```
Private Sub CmdBtn1_MouseMove(ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
CmdBtn1.Visible = False
CmdBtn2.Visible = True
A = A + 1
If A > 1 Then
        Lbl1.Visible = True
End If

End Sub
```

c) Druhé tlačítko

```
Private Sub CmdBtn2_MouseMove(ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
CmdBtn2.Visible = False
CmdBtn1.Visible = True
B = B + 1
If B > 3 Then
        Lbl1.Caption = ("Konec? Pro ukončení klikni zde.")
End If

End Sub
```

d) Štítek

```
Private Sub Lbl1_Click()
Unload UserForm1
End Sub
```

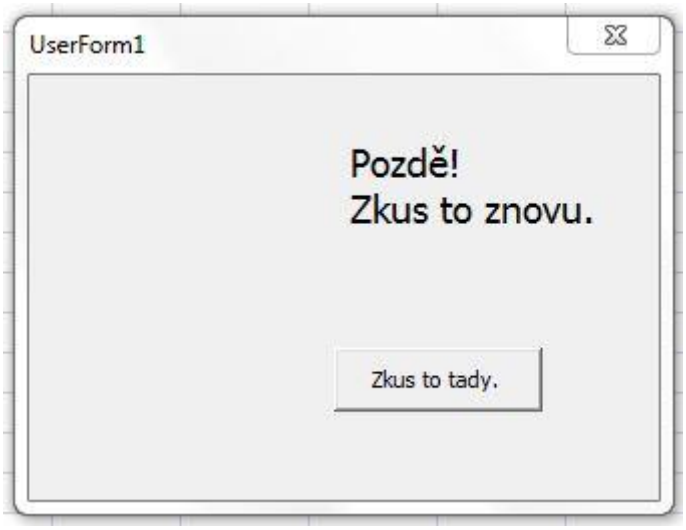
Popis programu

Nejprve je nutné deklarovat proměnné, v našem případě proměnné A a B. Tento první krok musíme udělat ještě před jakoukoli další částí kódu, protože tyto proměnné potřebujeme používat u obou tlačítek. Kdybychom si tyto proměnné deklarovali pouze u jednotlivých tlačítek zvlášť, nezobrazil by se nám výsledek správně.

Co se nám stane, když najedeme kurzorem myši na první tlačítko, můžeme vidět v proceduře s názvem *CmdBtn1_MouseMove*, kde *CmdBtn1* je název tlačítka a *MouseMove* je událost, při které se provede vše obsažené v této proceduře. Měníme tam i viditelnost jednotlivých tlačítek. Tlačítku *CmdBtn1* nastavujeme viditelnost (Visible) na False, což znamená, že tlačítko ukryjeme. Naopak tlačítku *CmdBtn2* měníme viditelnost na True, takže se nám zobrazí na formuláři. Dále zvětšíme proměnnou A o hodnotu 1, která spolu s následující *podmínkou If* určuje zobrazení štítku na formuláři (Lbl1). Formulář se zobrazí ve chvíli, kdy po druhé zajedeme na *CmdBtn1*, proto je v podmínce, že v případě, že A je větší než jedna, se štítek zobrazí.

Druhé tlačítko je v podstatě stejné jako první s jediným rozdílem, že skrývání tlačítek je prohozené - místo proměnné A se zvětšuje o hodnotu 1 proměnná B a jiná je i *podmínka If*. Ta nám říká, co se stane, pokud bude hodnota B větší než 3. Výsledek zjistíme při čtvrtém najetí kurzoru myši na tlačítko *CmdBtn2*. Změní se nám popisek v našem štítku a víme, že tuto „nekonečnou“ hru můžeme ukončit kliknutím na štítek.

Ukončení je popsané v události *Lbl1_Click* říkající, co se stane právě po kliknutí tlačítka na tento štítek. K tomu nám dopomáhá funkce UNLOAD.



Obr. 13 Formulář

4.2.6 Zadávání kódu PIN

I když ne každý má tuto službu aktivovanou, všichni jistě znáte, co se stane při zapínání mobilního telefonu. Mobilní telefon po vás bude chtít zadání čtyřmístného PIN kódu, bez kterého vás do telefonu nepustí. V této aplikaci si něco takového zkusíme naprogramovat.

Zadání
a) Abychom se mohli pomocí PIN kódu někam dostat či přihlásit, je třeba si nejprve PIN nastavit a jeho vytvoření bude tedy náš první úkol. Aby byl příklad užitečnější, po nastavení PIN se uzamkne zároveň List1 v Excelu.

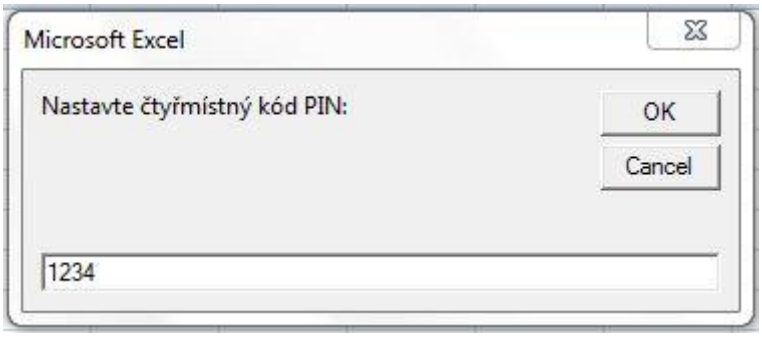
b) Nyní, když máme nastavený PIN, můžeme se zkoušet přihlašovat. Proto tedy vytvoříme funkční program na principu zadávání PIN dotazující se nás na toto čtyřmístné číslo. Pokud jej zadáte špatně, program vypíše nesprávnost zadání PIN a sníží vám možnosti zadávání o jeden pokus. Pokusy budou dohromady tři. Po třech nesprávných zadáních program vypíše, že již nemáte žádné pokusy a ukončí se. Pokud v těchto třech pokusech zadáte PIN správně, program naopak vypíše správnost PIN kódu a zároveň nám odemkne uzamčený List1.

Zdrojový kód
K tomu, aby celý tento program fungoval, je nutné zvlášť definovat proměnnou SpravnyPIN před jakýmkoli dalším kódem na samém začátku.

```
Sub SpravnyPIN As Integer
```

a) Zamknutí a nastavení PIN

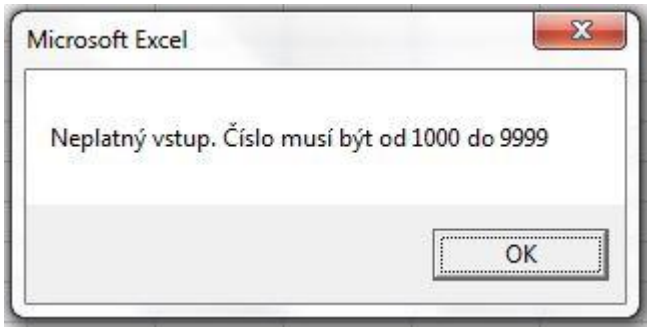
```
Sub Zamknuti_PIN()
SpravnyPIN = InputBox ("Nastavte čtyřmístný kód PIN:")
If SpravnyPIN > 1000 And SpravnyPIN < 9999 Then
        MsgBox ("PIN byl úspěšně změněn.")
        Sheets ("List1").Protect Password:=SpravnyPIN
Else
        MsgBox ("Neplatný vstup. PIN musí být od 1000 do 9999")
End If

End Sub
```



Obr. 14 Nastavení PIN kódu

b) Vstup pomocí PIN kódu a odemknutí Listu1

```
Sub Zadani_PIN()
Dim A As Integer, I as Integer

For I = 2 To 0 Step -1
        A = InputBox ("Zadejte váš PIN:")
        If A > 1000 And A < 9999 Then
                If A = SpravnyPIN Then
                        MsgBox ("Zadaný PIN je správný")
                        I = 0
                        Sheets ("List1").Unprotect Password:=SpravnyPIN
                Else
                        MsgBox ("Zadaný PIN je chybný, zbývá " & I & " pokusů")
                End If
        Else
                MsgBox ("Neplatný vstup. Číslo musí být od 1000 do 9999")
                I = 2
        End If
Next I
End Sub
```

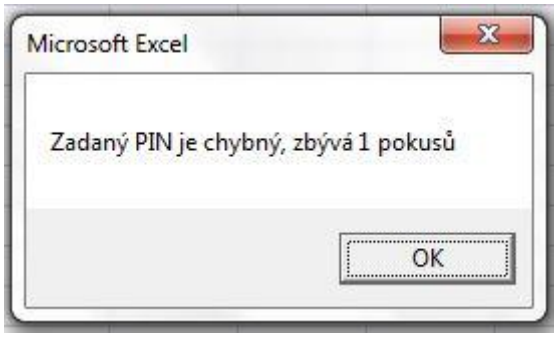


Obr. 15 Neplatný vstup

Popis programu

V první části budeme nastavovat PIN, což znamená zadávání jakéhosi čísla, které budeme muset někam uložit. Zadání provedeme pomocí *InputBoxu.* Po spuštění programu budeme dotázáni na čtyřmístné číslo, které se po zadání uloží do proměnné SpravnyPIN. Abychom však jako PIN nemohli zadat libovolné číslo, hned za *InputBoxem* následuje *podmínka If* mající za účel vyfiltrování správného zadání PIN. Pokud je zadané číslo větší než 1000 a menší než 9999, pak se nám v Excelu zamkne List1 pod zadaným heslem (SpravnyPIN). Při nesplnění podmínky nám bude pomocí *MsgBoxu* sděleno, že je zadaný PIN v nesprávném formátu či rozsahu a nic dalšího se nestane.

V této části začínáme *For cyklem*, který začíná na hodnotě 2 a bude probíhat až do nuly tak, že se po každém kroku I zmenší o jedna. Cyklus tedy proběhne maximálně třikrát, stejně jako v mobilu, kde máme na zadání PIN kódu také jen 3 pokusy. V těle cyklu nás čeká nejprve *InputBox*, do kterého zadáme PIN, přesvědčeni o jeho správnosti. Následuje *podmínka If* mající stejně jako v předchozí části za úkol vyfiltrování špatného zadání kódu PIN. Při zadání PIN ve špatném rozsahu nám tato chyba bude oznámena pomocí *MsgBoxu* a proměnná I bude nastavena znovu, tedy na hodnotu 2, takže nepřijdeme o pokus zadání. Při správném rozsahu zadání PIN kódu pokračujeme k další *podmínce If*. V této podmínce se nám porovnává, jestli se PIN nastavený při zamykání Listu1 (SpravnyPIN) shoduje s tím, který jsme zadali teď, při odemykání (proměnná A). V případě shody se skrze *MsgBox* dozvíme o správném zadání PIN, odemkne se nám List1 a proměnná I se nastaví na 0, čímž dojde k ukončení cyklu. V případě neshody nám *MsgBox* vypíše chybné zadání PIN kódu a počet zbývajících pokusů. To je řešené pomocí proměnné I.



Obr. 16 Chybné zadání kódu PIN

4.2.7 Budík

Myslím, že nikdo z nás už si nedovede představit život bez hodin a bez budíku. Pro příklad: Kdo z nás by si dokázal večer říci, že chce vstát třeba v sedm ráno a přesně na čas se uměl bez jakékoli pomoci vzbudit? Budík je v dnešní době takřka povinná výbava jakéhokoli zařízení obsahujícího i hodiny.

Zadání
Vytvořte program, který bude měřit čas. Hodiny je možné zastavit a opět spustit. Další funkcí, obsaženou v nastavených hodinách, je budík. To znamená, že uživatel zadá čas upozornění a v ten daný okamžik ho budík pomocí zprávy upozorní na nastalou časovou shodu. Když zprávu uživatel potvrdí, budík se posune o 30 vteřin. Pro vypnutí budíku je nutné stisknutí tlačítka pro zastavení.

Obr. 17 Spuštění formuláře

Zdrojový kód
K tomu, aby celý program dobře fungoval, je nutné definovat proměnnou NextTick a NextCheck, úplně nahoře před předchozím kódem.

```
Dim NextTick, NextCheck
```

a) Spuštění hodin

```
Private Sub CmdBtn_ZH_Click()
UpdateClock
End Sub
```

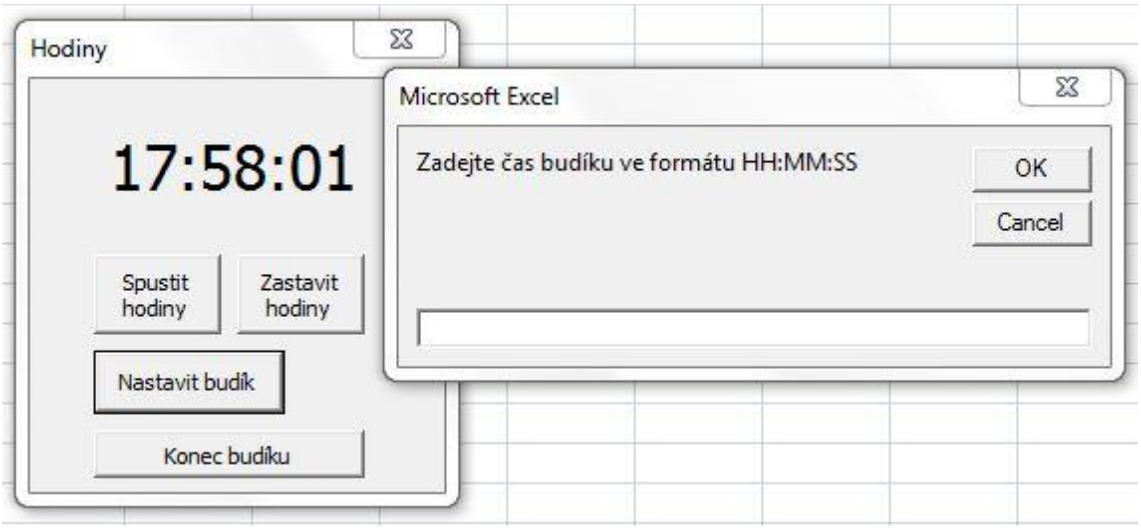
```
Sub UpdateClock()
UserForm1.Lbl_Hod = Time
NextTick = Now + TimeValue("00:00:01")
Application.OnTime NextTick, "UpdateClock"
```

```
End Sub
```

b) Zastavení hodin

```
Private Sub CmdBtn_ZH_Click()
StopClock
End Sub
```

```
Sub StopClock()
On Error Resume Next
Application.OnTime NextTick, "UpdateClock", , False
End Sub
```



Obr. 18 Nastavení času budíku

c) Nastavení budíku

```
Private Sub CmdBtn_NB_Click()
Dim X As String
UserForm1.Lbl_Bud = InputBox("Zadejte čas budíku ve formátu HH:MM:SS:")
Image1.Visible = True
Budik
End Sub
```

```
Sub Budik()
If UserForm1.Lbl_Bud = UserForm1.Lbl_Hod Then
        MsgBox ("Budík zvoní. Po stisknutí tlačítka OK bude posunut o 30 vteřin.")
        UserForm1.Lbl_Bud.Caption = Time + TimeValue("00:00:30")
        Application.OnTime NextCheck, "Budik"
Else
        NextCheck = Now + TimeValue("00:00:01")
        Application.OnTime NextCheck, "Budik"
```

```
End If
End Sub
```

d) Konec budíku

```
Private Sub CmdBtn_KB_Click()
StopBudik
End Sub
```

```
Sub StopBudik()
On Error Resume Next
Application.OnTime NextCheck, "Budik", , False
End Sub
```

Popis programu

Program tvoří v první řadě Formulář, který se skládá ze tří tlačítek, jednoho obrázku a dvou štítků. Jedno tlačítko spouští hodiny, druhé hodiny zastavuje a třetí tlačítko nám umožňuje nastavit budík. Jeden štítek je pro zobrazení hodin a druhý je pomocný pro nastavení budíku. Obrázek nám zobrazuje aktivní nastavený budík.

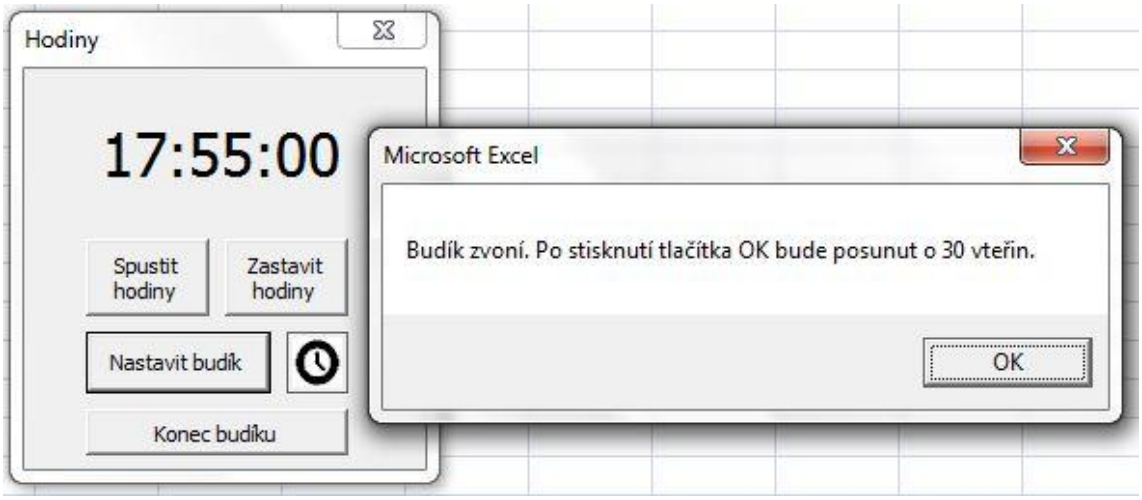
Po kliknutí na tlačítko pro spuštění hodin (*CmdBtn_SH*) se nám spustí Makro s názvem *UpdateClock* aktualizující štítek (*Lbl_Hod*) momentálním přesným časem. Zároveň nastaví pro proměnnou NextTick čas o vteřinu pozdější než je aktuální čas. Jako poslední nastalý krok po stisku tohoto tlačítka je nastavení spuštění makra s názvem *UpdateClock* po čase NextTick, což je v našem případě vteřina. Tím pádem se nám toto makro spouští každou vteřinu. Makro (*UpdateClock*) pokračuje na pozadí do té doby, než co jej opět nezastavíme pomocí druhého tlačítka.

Druhé tlačítko nás odkazuje opět na modul, kde se skrývá naše makro, tentokrát s názvem StopClock. Funkce tohoto makra je zastavení události Application.OnTime pomocí příkazu False na konci této události. Abychom mohli přistoupit k zastavení události, je třeba před ni vložit On Error Resume Next. Bez vložení by výsledkem tohoto makra byla pouze chyba a k ničemu by nedošlo. A to z toho důvodu, že událost Application.OnTime se každou vteřinu znovu spouští a tím pádem k ní nemůžeme „za běhu“ přistupovat.

Tlačítko pro nastavení budíku (*CmdBtn_NB*) funguje následovně: Uživatel je *InputBoxem* dotázán na čas, kdy chce být vzbuzen. Čas zadaný od uživatele se nastaví do štítku (*Lbl_Bud*), jehož hodnota viditelnosti je False. Slouží nám pouze k následnému porovnávání s aktuálním časem. Poté se nastaví obrázek s budíkem (*Image1*) a hodnotou viditelnosti na True, což zajistí zobrazení obrázku na

formuláři, a tedy označení zapnutí budíku. Posledním článkem je spuštění makra s názvem Budik.
V makru je vložena jedna *podmínka If*, která při splnění skrze *MsgBox* vypíše, že budík zvoní, a že v případě potvrzení této zprávy posuneme budík o 30 vteřin. Proto je dále nastaven do *Lbl_Bud.Caption* aktuální čas, ke kterému je přičteno právě oněch 30 vteřin. Poslední částí při splnění podmínky je nastavení opakování tohoto makra po jedné vteřině, stejně jako tomu bylo u tlačítka spouštějícího hodinu.
Pokud by podmínka splněna nebyla, dojde pouze k nastavení opakování tohoto makra po jedné vteřině.

Posledním zbývajícím tlačítkem je tlačítko ukončení budíku. Abychom tzv. vyřadili budík z provozu, je nutné přerušit ono neustálé opakování makra Budik. Když budu konkrétnější, musí se přerušit ono opakování spouštění makra, které se odehrává každou vteřinu. Tady je to prakticky takřka totožné se zastavením hodin, jen je třeba změnit NextTick na NextCheck a Application.OnTime nastavit na False.



Obr. 13 Zvonění budíku

5 Závěr

Závěrem bych rád shrnul všechny potřebné informace, které považuji ve své bakalářské práci za důležité.
Teoretická část nám umožnila nahlédnout do historie tabulkových procesorů, kde jsme mohli zjistit, že Excel nebyl od začátku dominantním programem svého druhu. Nejlépe však zvládl vývoj počítačových technologií a přinesl v pravou chvíli to nejlepší.
Dále jsme se v obecné rovině věnovali programování a programovacím jazykům. Naprostá většina lidí už jistě název některého z programovacích jazyků slyšela. Rozebíral jsem ale i méně známé skutečnosti, a to zejména, jak se tyto programovací jazyky řadí, které byly první nebo jak se vyvíjely, případně kam který zařadit. Možná to nejsou informace, které by měly být základem všeobecného povědomí společnosti, ale vzhledem k faktu, že jsem se setkal s případy programátorů, a sám jsem se mezi ně počítal, neznalých těchto, pro programátory základních znalostí, považoval jsem za vhodné těmito vědomostmi svou bakalářskou práci doplnit. Tento pohled na strukturu programovacích jazyků totiž může člověku leccos ujasnit.

V praktické části se při porovnávání tabulkových procesorů vyskytl menší problém s hledáním srovnání nebo zhodnocení tabulkových procesorů, z kterého bych se mohl inspirovat nebo z něho čerpat. Až po důkladnějším pátrání jsem konečně našel malé množství článků, které mi ve své použitelnosti byly alespoň malou satisfakcí. Vůbec jsem při svém hledání kladně nepochodil u zdrojů knižních, až teprve s pomocí internetu jsem začal mít pozitivní výsledky. Co se porovnání týče, tak v první řadě záleží na vašem záměru využití tabulkového procesoru. Pro běžnou práci se bohatě spokojíte s jakýmkoli základním tabulkovým procesorem. Pokud byste však chtěli složitější úpravy, pak je MS Excel, dle mého zjištění, vhodnějším nástrojem, neboť nabízí přece jen něco navíc. Zhodnocení není sice nikterak obsáhlé, leč předpokládám, že pro naši potřebu dostačující.

V příkladech programování, uvedených v poslední části bakalářské práce, jsem vytvořil dohromady 7 příkladů zaměřených na určitou problematiku či část programování a hlavně na procvičení v praxi. Případný zájemce má tak možnost vyzkoušet si nejen práci s formulářem, funkci pole nebo procvičení použití cyklů. Příklady jsem se snažil vytvářet neotřelé, pro uživatele zajímavé, v praxi použitelné a tématikou výjimečné. Internet je plný stejných příkladů typu „Hello, world.“, které se používají při výuce většiny programovacích jazyků. Výhodou mnou uvedených a řešených příkladů je jejich rozšiřitelnost. Většinu z nich je, dle mého názoru, možné vylepšit či rozšířit, čímž vznikají další a další příklady pro rozvoj a osvojení si programování.

Důležitým faktorem pro tvorbu mé bakalářské práce bylo poskytnout uživatelům nové informace, seznámit je s příklady programování, ukázat případným studentům a žákům užitečnost a přitom v některých příkladech i nepříliš náročný postup ke kýženému výsledku. Věřím, že po těchto zjištěních se mezi studenty najdou tací, kteří se programování budou věnovat detailněji i ve své následné praxi.

6 Seznam použité literatury

[1] WALKENBACH, John. Microsoft *Office Excel 2007: programování ve VBA*. Vyd. 1. Brno: Computer Press, 2008. ISBN 978-80-251-2011-8, s. 38-46

[2] WALKENBACH, John. *Microsoft Office Excel 2007: programování ve VBA*. Vyd. 1. Brno: Computer Press, 2008, ISBN 978-80-251-2011-8, s. 46

[3] WALKENBACH, John. *Microsoft Office Excel 2007: programování ve VBA*. Vyd. 1. Brno: Computer Press, 2008, ISBN 978-80-251-2011-8, s. 47

[4] MORKES, David. *Základy programování*. Vyd. 1. Praha: Computer Press, 1998, ISBN 80-7226-062-6, s. 17

[5] MORKES, David. *Základy programování*. Vyd. 1. Praha: Computer Press, 1998, ISBN 80-7226-062-6, s. 18

[6] GAULD, Alan. *Data, datové typy a proměnné* [online]. 3. 9. 2005 [cit. 3. 11. 2014] Dostupné z: http://jaksenaucitprogramovat.py.cz/cztutdata.html

[7] MORKES, David. *Visual Basic 4.0 pro střední školy: učebnice pro střední školy*. Vyd. 1. Praha: Computer Press, 1997, ii, ISBN 80-85896-92-3, s. 64

[8] MORKES, David. *Visual Basic 4.0 pro střední školy: učebnice pro střední školy*. Vyd. 1. Praha: Computer Press, 1997, ii, ISBN 80-85896-92-3, s. 74

[9] WALKENBACH, John. *Microsoft Office Excel 2007: programování ve VBA*. Vyd. 1. Brno: Computer Press, 2008, ISBN 978-80-251-2011-8, s. 243

[10] Wikipedie: Otevřená encyklopedie: *Nižší programovací jazyk* [online]. 2013 [cit. 5. 11. 2014]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Ni%C5%BE%C5%A1%C3%AD_programovac%C3%AD_jazyk

[11] Wikipedie: Otevřená encyklopedie: *Vyšší programovací jazyk* [online]. 2014 [cit. 11. 11. 2014]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Vy%C5%A1%C5%A1%C3%AD_programovac%C3%AD_jazyk

[12] Wikipedie: Otevřená encyklopedie: *Deklarativní programování* [online]. 2013 [cit. 11. 11. 2014]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Deklarativn%C3%AD_programov%C3%A1n%C3%AD

[13] Wikipedie: Otevřená encyklopedie: *Interpretovaný jazyk* [online]. 2014 [cit. 20. 11. 2014]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Interpretovan%C3%BD_jazyk

[14] *Programovací jazyky* [online]. 13. 2. 2003 [cit. 21. 11. 2014] Dostupné z: http://k-prog.wz.cz/progjaz/index.php

[15] MORKES, David. *Visual Basic 4.0 pro střední školy: učebnice pro střední školy*. Vyd. 1. Praha: Computer Press, 1997, ii, ISBN 80-85896-92-3, s. 1

[16] WRÓBLEWSKI, Piotr. *Algoritmy: datové struktury a programovací techniky*. Vyd. 1. Brno: Computer Press, 2004, ISBN 80-251-0343-9, s. 10

[17] WRÓBLEWSKI, Piotr. *Algoritmy: datové struktury a programovací techniky*. Vyd. 1. Brno: Computer Press, 2004, ISBN 80-251-0343-9, s. 17

[18] WRÓBLEWSKI, Piotr. *Algoritmy: datové struktury a programovací techniky*. Vyd. 1. Brno: Computer Press, 2004, ISBN 80-251-0343-9, s. 19

[19] LAURENČÍK, Marek a Michal BUREŠ. *Programování v Excelu 2010 & 2013: záznam, úprava a programování maker*. 1. vyd. Praha: Grada, 2013, ISBN 978-80-247-5033-0, s. 11

[20] WRÓBLEWSKI, Piotr. *Algoritmy: datové struktury a programovací techniky*. Vyd. 1. Brno: Computer Press, 2004, ISBN 80-251-0343-9, s. 11

[21] JELÍNEK, Lukáš. *Srovnání LibreOffice, OpenOffice.org a Microsoft Office* [online]. 23. 12. 2011 [cit. 6. 11. 2014] Dostupné z: http://www.linuxexpres.cz/kancelar/srovnani-libreoffice-openoffice-org-a-microsoft-office#t11

[22] LAURENČÍK, Marek. *OpenOffice 3.4.1 vs. Microsoft Office 2010* [online]. 18. 2. 2013 [cit. 6. 11. 2014] Dostupné z: http://www.napocitaci.cz/33/openoffice-3-4-1-vs-microsoft-office-2010-uniqueidgOkE4NvrWuNY54vrLeM670fvKRLx6va11UWBS0tlgLw/?reltype=2&uri_view_type=12

[23] LAURENČÍK, Marek. *OpenOffice Calc vs. Microsoft Excel* [online]. 27. 2. 2013 [cit. 6. 11. 2014] Dostupné z: http://www.napocitaci.cz/33/openoffice-calc-vs-microsoft-excel-uniqueidgOkE4NvrWuNY54vrLeM676GITgRZMUUe1UWBS0tlgLw/?sekce=33&esupid=17p8uj49lvQPyEscYUPluSS5Qg%3D%3D&uniqueid=gOkE4NvrWuNY54vrLeM676GITgRZMUUe1UWBS0tlgLw&justlogged=1)