

Programovací nástroje C# pro děti a mládež.

C# Programming Tools for Children and Youth.

Bc. Petr Martinák



OBSAH

ÚVOD	5
1 METODICKÝ LIST – ČÍSLO 1.....	7
1.1 ZÁKLADY ALGORITMIZACE:.....	7
1.1.1 Co je to algoritmus?.....	7
1.1.2 K čemu se používá?	7
1.1.3 Zápis algoritmu.....	7
1.1.4 Vlastnosti algoritmu	8
1.2 OTÁZKY Z PROBRANÉHO TÉMATU	9
1.3 PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	9
1.4 ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	9
1.5 SEZNAM POUŽITÉ LITERATURY	11
2 METODICKÝ LIST – ČÍSLO 2.....	12
2.1 VÝVOJOVÝ DIAGRAM	12
2.1.1 Symboly vývojových diagramů.....	12
2.2 OTÁZKY Z PROBRANÉHO TÉMATU	17
2.3 PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	18
2.4 ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	20
2.5 SEZNAM POUŽITÉ LITERATURY	22
3 METODICKÝ LIST – ČÍSLO 3.....	23
3.1 TYPY VÝVOJOVÝCH DIAGRAMŮ PRAKTICKY	23
3.1.1 Sekvence	23
3.1.2 Větvení.....	25
3.1.3 Cykly.....	27
3.1.4 Nejčastější chyby v algoritmu	28
3.2 OTÁZKY Z PROBRANÉHO TÉMATU	31
3.3 PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	31
3.4 ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	31
3.5 SEZNAM POUŽITÉ LITERATURY	33
4 METODICKÝ LIST – ČÍSLO 4.....	34
4.1 VÝVOJOVÝ PROSTŘEDÍ MICROSOFT VISUAL C# 2010 EXPRESS EDITION	34
4.1.1 Založení nového projektu v Microsoft Visual C# 2010 Express	35
4.1.2 Editor kódu.....	36
4.1.3 Úprava nastavení vývojového prostředí	37
4.1.4 Základní struktura programu	41
4.2 OTÁZKY Z PROBRANÉHO TÉMATU	41
4.3 PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	41
4.4 SEZNAM POUŽITÉ LITERATURY	42
5 METODICKÝ LIST – ČÍSLO 5.....	43
5.1 PRVNÍ PROGRAM VE VÝVOJOVÉM PROSTŘEDÍ.....	43
5.1.1 Ikony nápovědy IntelliSense	48
5.1.2 Komentáře zdrojového kódu	48
5.1.3 Jednotlivé bloky ve zdrojovém kódu.....	49

5.1.4	Únikové sekvence.....	49
5.2	OTÁZKY Z PROBRANÉHO TÉMATU	50
5.3	PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	50
5.4	ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	51
5.5	SEZNAM POUŽITÉ LITERATURY	51
6	METODICKÝ LIST – ČÍSLO 6.....	52
6.1	PRÁCE S PROMĚNNÝMI, DATOVÝMI TYPY, OPERÁTORY A PARSOVÁNÍM	52
6.1.1	Co je to proměnná?.....	52
6.1.2	Typový systém	54
6.1.3	Primitivní datové typy	55
6.1.4	Aritmetické operátory.....	57
6.1.5	Priorita vyhodnocování operátorů	58
6.1.6	Parsování.....	59
6.2	OTÁZKY Z PROBRANÉHO TÉMATU	59
6.3	PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	60
6.4	ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	61
6.5	SEZNAM POUŽITÉ LITERATURY	62
7	METODICKÝ LIST – ČÍSLO 7.....	63
7.1	ROZHODOVACÍ PŘÍKAZY IF, SWITCH A CYKLY FOR, WHILE A DO PŘI JEJICH POUŽITÍ.....	63
7.1.1	Logické proměnné	63
7.1.2	Logické operátory.....	63
7.1.3	Užitečné logické operátory	64
7.1.4	Operátory prefix a postfix	64
7.1.5	Rozhodování pomocí příkazu if	65
7.1.6	Rozhodování pomocí příkazu switch.....	66
7.1.7	Operátory složeného přiřazení	67
7.1.8	Cykly pomocí příkazu for	68
7.1.8.1	Obor platnosti příkazu for	69
7.1.9	Cykly pomocí příkazu while	70
7.1.10	Cykly pomocí příkaz do.....	71
7.1.10.1	Příkazy break a continue v cyklech:	72
7.2	OTÁZKY Z PROBRANÉHO TÉMATU	72
7.3	PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	73
7.4	ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	73
7.5	SEZNAM POUŽITÉ LITERATURY	75
8	METODICKÝ LIST – ČÍSLO 8.....	76
8.1	OŠETŘENÍ UŽIVATELSKÉHO VSTUPU, VÝJIMKY A POLE	76
8.1.1	Ošetření uživatelského vstupu.....	76
8.1.2	Výjimky	76
8.1.2.1	Příkazy try, catch a finally	77
8.1.2.2	Ošetření výjimky	78
8.1.3	Zachytávání výjimek několika typů obslužnou rutinou.....	80
8.1.4	Pole	81
8.1.4.1	Deklarace pole.....	81
8.1.4.2	Vytvoření instance pole	81

8.1.4.3	Naplněné pole.....	82
8.1.4.4	Výpis hodnot pole.....	82
8.1.5	Užitečné metody pole	83
8.1.5.1	Metoda Sort()	83
8.1.5.2	Metoda Average	83
8.1.6	Metoda Min a Max	84
8.2	OTÁZKY Z PROBRANÉHO TÉMATU	84
8.3	PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	84
8.4	ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	85
8.5	SEZNAM POUŽITÉ LITERATURY	86
9	METODICKÝ LIST – ČÍSLO 9.....	87
9.1	PRAKTICKÉ CVIČENÍ	87
9.2	ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	88
9.3	SEZNAM POUŽITÉ LITERATURY	93
10	METODICKÝ LIST – ČÍSLO 10.....	94
10.1	WINDOWS FORMS APPLICATIONS	94
10.1.1	Založení projektu.....	94
10.1.2	Vytvoření jednoduché grafické aplikace	95
10.1.3	Práce s okny se zprávou.....	101
10.2	OTÁZKY Z PROBRANÉHO TÉMATU	102
10.3	PRAKTICKÉ CVIČENÉ NA DANÉ TÉMA.....	102
10.4	ŘEŠENÍ PRAKTICKÉHO CVIČENÍ	103
10.5	SEZNAM POUŽITÉ LITERATURY	105
	ZÁVĚR	106
	SEZNAM POUŽITÉ LITERATURY	108
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	109
	SEZNAM OBRÁZKŮ.....	110
	SEZNAM TABULEK	112
	SEZNAM PŘÍLOH	113

ÚVOD

Problematika algoritmizace a programování v jazyce C# je tak moc obsáhlá informacemi, že by vydala na několik knih s pořádně širokou vazbou. Proto ani cílem těchto metodických listů není říci první, poslední o daných možnostech, funkcích, metod a podobně. Není to reálné a mohlo by to posloužit kontraproduktivně. Tyto metodické listy mají sloužit jako takový první odrazový můstek pro získání základních informací při práci s vytvářením jednoduchých vývojových algoritmů. Mají uživatele seznámit s vývojovým prostředím Microsoft Visual C# 2010 Express Edition a psaním zdrojového kódu, jeho ladění a spuštění výsledného programu.

Metodické listy obsahují celkově 10 listů. V každém metodickém listu čtenář nalezne hlavní téma, které se bude v daném metodickém listu probírat. Po prostudování tohoto tématu na čtenáře čekají jednoduché kontrolní otázky pro ověření jeho pozornosti. Po absolvování těchto kontrolních otázek má čtenář zadané praktické úkoly týkající se právě probraného tématu. Není ovšem pozapomněno i správně řešení k jednotlivým otázkám. Toto řešení je uvedené jako předposlední bod metodického listu. Posledním bodem je seznam literatury. V tomto bodu je vložen přehled literatury, odkud byly informace čerpány pro daný metodického list. Popřípadě mohou tyto informace sloužit čtenáři jako zdroj dalších informací o dané problematice.

První tři metodické listy jsou věnované hlavně pojmům algoritmizace vývojové diagramy, prvkům vývojového diagramu a typům vývojových diagramů. Metodické listy čtenáře seznámí se třemi základními typy vývojových diagramů a uvádí nejčastější chyby při jejich návrzích. Není zde pozapomenuto ani na praktické cvičení spolu s jeho řešením velmi jednoduchých základních příkladů.

Čtvrtý metodický list je věnován vývojovému prostředí Microsoft Visual C# 2010 Express Edition. Čtenář by se zde měl prakticky seznámit s tímto vývojovým prostředím a je zde uveden návod na jeho nastavení podle obrazu svému a založení prvotního projektu.

Pátý metodický list čtenáři nabízí první program v tomto vývojovém prostředí. Prakticky má za úkol seznámit čtenáře se základní strukturou programu. Praktickým nástrojem intellisense a podobně.

Metodický list číslo šest je již plně věnován programovacímu jazyku C# zejména práci s proměnnými, datovými typy, operátory a parsováním. Takto začíná cesta se

seznámením další nutných informací o programovacím jazyku C#. Proto i zbylé metodické listy na tuto problematiku nadále navazují a věnují se jí jak z teoretické stránky, tak ze strany praktického procvičování základních úkolů a problémů.

Sedmý metodický list je věnován problematice rozhodovacích příkazů `if`, `switch` a cyklům `for`, `while`, `do` a jejich použití. Na tuto problematiku navazuje metodický list číslo osm pro programátora velmi důležitým tématem a to ošetření uživatelského vstupu. Tedy vyvarování se problému, aby uživateli program přestal fungovat z jakéhokoliv důvodu. V devátém metodickém listu si čtenář své znalosti ze všech předešlých metodických listů může prakticky vyzkoušet. Desátý metodický list je již oddechový. Má za úkol čtenáře seznámit s vytvořením formulářové aplikace a základní práci s ní

1 METODICKÝ LIST – ČÍSLO 1

První a také i druhé číslo metodického listu je věnované ne příliš oblíbené teorii, kterou je zapotřebí si osvojit, aby následující probíraná látka dávala smysl. V dalších tématech se od teorie bude upouštět a budeme se věnovat z větší části praktickým činnostem.



CÍLEM TÉMATU – je objasnění problematiky týkající se algoritmu, jeho vlastností, zápisu a příklady použití



KLÍČOVÁ SLOVA – algoritmus, vývojový diagram, elementárnost, determinovanost, konečnost, rezultativnost, hromadnost



ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 60 minut.

1.1 Základy algoritmizace:

1.1.1 Co je to algoritmus?

V dnešní době se každý člověk setkává s algoritmem, aniž by si to uvědomoval. Často se hovoří o algoritmu ve spojení s počítačem, toto je ovšem jen jedna z řad možností jeho využití. Intuitivně algoritmem rozumíme postup, který nás dovede k řešení zadané úlohy. Proto i prostý návod na uvaření instantní polévky se může považovat za algoritmus.



Algoritmus je konečná uspořádaná množina úplně definovaných pravidel pro vyřešení libovolného problému.

1.1.2 K čemu se používá?

Algoritmus je využit, jak již bylo řečeno ke grafickému znázornění vyvíjeného softwaru. Slouží jako komunikační prostředek při týmové spolupráci analytiků a programátorů. V obecném měřítku lze algoritmus použít v jakémkoliv vědeckém odvětví. Algoritmem lze považovat i různé manuály elektronickým zařízením, kuchařské recepty nebo jen postupy, které uživatele dovedou ke správným výsledkům.

1.1.3 Zápis algoritmu

Možností zápisu algoritmu je několik od *slovního vyjádření*, přes *matematický zápis*, *rozhodovací tabulky* až po *vývojové diagramy* a *počítačový program*.

Slovní popis algoritmu je znám z běžného života, jelikož se do této skupiny řadí nejrůznější návody k používání různých výrobků či technologické postupy. Tato forma zápisu, je ale ze všech možných forem nejméně přehledná.

Matematický zápis je dobrý všude tam, kde je možné řešenou problematiku popsat pomocí matematických vztahů. Ale pro vývoj softwaru tento zápis také nebude nejvhodnější.

Rozhodovací tabulky jsou velice vhodné v případech, kdy se v úloze vyskytuje několik možností a vlastní řešení je pro každou možnost jednoduše popsitelné. Takovou to úlohou může být rozvrh hodin pro určitou konkrétní třídu.

Vývojové diagramy patří mezi nejdokonalejší formy zápisu algoritmu a i proto se s nimi budete setkávat v následujících metodických listech. Využívají se zejména při tvorbě softwaru. Vývojové diagramy se skládají z jednotlivých symbolů, které jsou mezi sebou spojeny orientovanými hranami – čárami. Tyto čáry jsou označeny šipkou pro vyjádření směru postupu algoritmu. Algoritmus ve formě vývojového diagramu by počítač nikdy nemohl zpracovat. Proto na řadu přicházejí programátoři, kteří algoritmus přepíší do některého programovacího jazyka, aby vytvořili funkční program.

1.1.4 Vlastnosti algoritmu



Při zápisu algoritmu musí jeho autor dodržet následující podmínky. Algoritmus musí mít začátek a konec, musí být věcně správný, jednoznačný, obecný, opakovatelný a srozumitelný.

Algoritmus udává přesný postup, který je zapotřebí dodržet, aby bylo dosaženo požadovaného výsledku. Aby byl algoritmus považován za správný, musí splňovat následující podmínky:

1. *Elementárnost* – pokud je algoritmus elementární znamená to, že se skládá z konečného počtu jednotlivých, snadno realizovatelných činností, které budeme označovat jako kroky [3].
2. *Determinovanost* – je myšlen takový stav algoritmu, kdy po každém kroku lze určit, zda popisovaný proces byl ukončen, a pokud ne, tak kterým směrem (krokem) má algoritmus následně pokračovat [2].
3. *Konečnost* – jedná se o velmi důležitou vlastnost algoritmu, kdy po konečném počtu kroků musí skončit [3].

4. *Rezultativnost* – je vlastnost, kdy algoritmus při zadání vstupních dat vždy vrátí nějaký výsledek, přičemž se může jednat jen o chybové hlášení [3].
5. *Hromadnost* – algoritmus musí vést k řešení celé třídy úloh, vzájemně se lišící pouze vstupními daty. Proto každý algoritmus musí být co nejobecnější, aby bylo možné pomoci něj řešit co nejširší množství úloh.

1.2 Otázky z probraného tématu

- ❖ Co je to algoritmus?
- ❖ Jakým způsobem lze zapsat algoritmus?
- ❖ Jaké vlastnosti algoritmu znáš?
- ❖ Jaký příklad použití znáš pro slovní vyjádření, pro matematický zápis či rozhodovací tabulky?
- ❖ K čemu se využívá vývojový diagram?
- ❖ Které možnosti využití algoritmu znáš?

1.3 Praktické cvičené na dané téma

Pokus se o realizaci algoritmu textově na papír nebo slovy následující algoritmy:

- (1) Přejítí cesty přes přechod.
- (2) Výměna náplně do propisky.
- (3) Nalistování stránky 25 v učebnici zeměpisu.
- (4) Popiš každodenní posloupnost úkonů, které je nutné vykonat, než odejdeš do školy.
- (5) Vymyslete si další příklady algoritmů.

1.4 Řešení praktického cvičení

(1) Přejítí cesty přes přechod.

- A. Start
- B. Stojím na přechodu.
- C. Jede auto?
 - a. Ano
 - b. Ne
 - i. Stůj!
 - j. Přejdi přes přechod.
- D. Konce

(2) Výměna náplně do propisky.

- A. Start
- B. Došla náplň v propisce.
- C. Mám novou náplň?
 - a. Ano
 - i. Otevři propisku.
 - ii. Vyjmi starou náplň.
 - iii. Vlož novou náplň.
 - iv. Uzavři propisku.
 - b. Ne
 - j. Jdu koupit novou náplň.
- D. Konec

(3) Nalistování stránky 25 v učebnici zeměpisu.

- A. Start
- B. Mám učebnici zeměpisu?
 - a. Ano
 - b. Ne
- C. Otevření učebnice na náhodné straně.
- D. Čti číslo aktuální stránky.
- E. Je číslo stránky rovno 25?
 - a. Ne
 - i. Je číslo stránky větší než 25?
 - Ano - listuj učebnicí nazpět
 - Ne - listuj učebnicí dopředu
 - b. Ano
- F. Konec

(4) Popiš každodenní posloupnost úkonů, které je nutné vykonat, než odejdeš do školy.

- A. Start
- B. Vypnutí budíku
- C. Stanutí z postele
- D. Svlečení z pyžama
- E. Obléknutí se
- F. Umývání se
- G. Nasnídání se
- H. Obutí se
- I. Opuštění domu
- J. Zamknutí dveří
- K. Konec

1.5 Seznam použité literatury

- [1] PŠENČÍKOVÁ, Jana. *Algoritmizace*. Vyd. 2. Kralice na Hané: Computer Media, 2009, 128 s. ISBN 978-80-7402-034-6.
- [2] *Jednotky v informačních technologiích* [online]. 2007 [cit. 2013-03-27]. Dostupné z: <http://home.zcu.cz/~wichs/index.html>
- [3] VIRIUS, Miroslav. *Základy algoritmizace*. Praha: ČVUT, 1995. ISBN 80-01-01346-4.

2 METODICKÝ LIST – ČÍSLO 2

Metodické číslo dvě je z větší části teorie, které se nelze vyvarovat. Tématem těchto listů jsou vývojové diagramy, které je zapotřebí si osvojit a seznámit se s nimi.



CÍLEM TÉMATU – je posluchače seznámit s problematikou kreslení vývojových diagramů. Po prostudování tohoto tématu by posluchač měl zvládnout nakreslit a správně použít symboly vývojového diagramu.



KLÍČOVÁ SLOVA – symboly vývojového diagramu, mezní značky, sekvenční blok, větvení, příprava, interní paměť, zobrazení, spojka, spojnice, podprogram



ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 90 minut.

2.1 Vývojový diagram

Vývojový diagram je grafické znázornění algoritmu. Znázorňuje „tok řízení“ v algoritmu. Jeho používání je vhodné zejména u začínajících programátorů, protože dovoluje názorným způsobem formulovat postup řešení daného úkolu s vyznačením všech jeho možných alternativ.

Pro kreslení vývojových diagramů platí od 1. ledna 1996 nová česká státní norma ČSN ISO 5807 [1]. Nahrazuje dřívější československou státní normu ČSN 36 9030 [2].

2.1.1 Symboly vývojových diagramů

Vývojové diagramy se skládají z jednotlivých symbolů, a ty jsou mezi sebou spojeny orientovanými hranami – čarami, které jsou označeny šipkou vyjadřující směr postupu algoritmu. Obecně vžitý postup psaní značek je *shora dolů* a *zleva doprava*. V některých případech tento postup nemůže být dodržen. Dochází k tomu například v cyklech, kdy se čára vrací o několik kroků zpět. V takovém případě je použití šipky velmi důležité [4].

Pro specifikaci symbolů se do symbolů vpisují slovní nebo symbolické operace, popřípadě i skupina operací. Tato vepsaná symbolika norma neurčuje je však doporučeno použití jednoduché věcné textu a řídit se výpočetními znaky s použitím matematických značek podle normy ČSN ISO 31-11 [3].

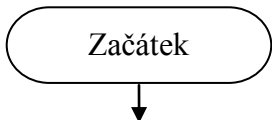
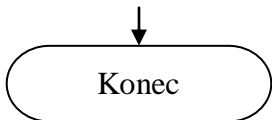
Díky tomuto upřesnění daného symbolu vývojového diagramu se zaručí přehlednost pro celý algoritmus a s takovýmto algoritmem nejenom lépe pracuje

programátor, ale i čtenář neznalý žádného programovacího jazyka popisovaný princip jednodušeji pochopí.

Pro nakreslení níže uvedených symbolů byly použity automatické tvary textového editoru Microsoft Word 2007. Mohou se tedy v některých detailech od normou definovaných značek tvarově lišit.

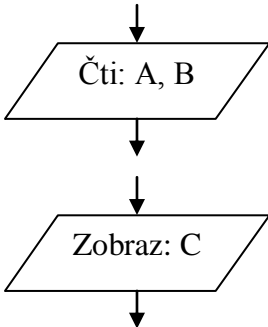
1. *Mezní značky* – představují vstup z vnějšího prostředí do programu nebo výstup z programu do vnějšího prostředí. Příkladem tomu může být začátek a konec programu či konec samostatně zpracovávané části programu. Má vždy jen jednu orientovanou hranu vstupní nebo výstupní [4].

Tab. 1. Mezní značky – začátku a konce algoritmu

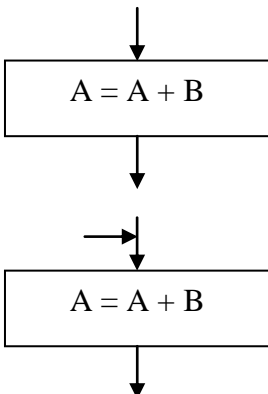
	<p>Začátek</p> <p>Pro symbol „Začátek“ platí ta pravidla, že do něj nesmí vstupovat žádná hrana a musí z něho vystupovat právě jedna hrana. Před tímto symbolem už nesmí být žádný další symbol [4].</p>
	<p>Konec</p> <p>Pravidla pro symbol „Konec“ jsou taková, že do symbolu musí vstupovat právě jedna hrana a nesmí z něho vystupovat žádná hrana. Za tímto symbolem už nesmí vystupovat žádný další symbol [4].</p>

2. *Sekvenční bloky* – vystupují uvnitř vývojového diagramu, tedy nesmějí být použity jako první nebo poslední symbol. Pracují na principu sekvenčního postupu algoritmem, kdy je možné se do dalšího prvku algoritmu dostat pouze z předchozího prvku. Po jeho vykonání lze postoupit pouze o jeden následující prvek. V jejich průběhu nesmí dojít k rozvětvení algoritmu [4].

Tab. 2. Sekvenční bloky – vstup a výstup dat

	<p>Vstup a výstup dat</p> <p>Symbol reprezentuje vstupně – výstupní operace s daty.</p> <p>Vstup zprostředkovává načtení dat potřebných pro činnost programu [4].</p> <p>Výstup slouží pro zobrazení výsledného stavu programu [4].</p> <p>Vstup i výstup jsou sekvenční bloky, proto musí mít na jedné hraně vstup a na hraně druhé výstup.</p>
---	---

Tab. 3. Sekvenční bloky – zpracování

	<p>Zpracování</p> <p>Symbol představuje jakýkoliv druh zpracování nebo provedení definované operace či skupiny operací, jejichž výsledkem je transformace dat [3]. Příkladem může být sečtení dvou čísel nebo jiná matematická či logická operace.</p> <p>Možnost vstupu do tohoto symbolu je z libovolné strany a těchto vstupů může být i několik. Přitom to mohou být vstupy samostatné, nebo se může jednat o jednu spojnicí výslednou, která vznikla spojením několika spojnic dílčích. Výstup je však v zásadě vždy jenom jeden [3].</p>
---	---

3. *Větvení* – K větvení nemůže docházet nahodile, ale vždy pouze na základě nějaké podmínky. Je-li podmínka splněna, pak program pokračuje jednou cestou, není-li splněna, pokračuje druhou cestou. Cestami jsou myšleny větve algoritmu.

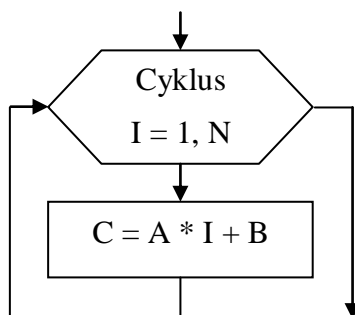


POZNÁMKA – při větvení se místo slovního vyjádření *ANO/NE* používají ekvivalentní značky $+/-$. Pro ANO je použita ekvivalentní značka $+$. Pro NE je použita ekvivalentní značka $-$.

Tab. 4. Symbol větvení

	<p>Rozhodovací symbol</p> <p>Symbol představuje rozhodovací nebo přepínací funkci. Je-li podmínka splněna, pak program pokračuje větví označenou ekvivalentní značkou „+“, není-li splněna, pak pokračuje větví označenou ekvivalentní značkou „-“.</p> <p>Nezáleží na umístění znamének, u které větve se vyskytuje. Záleží především na přehlednosti vývojového diagramu. Zpravidla má symbol obvyčejně 2 výstupy.</p>
--	---

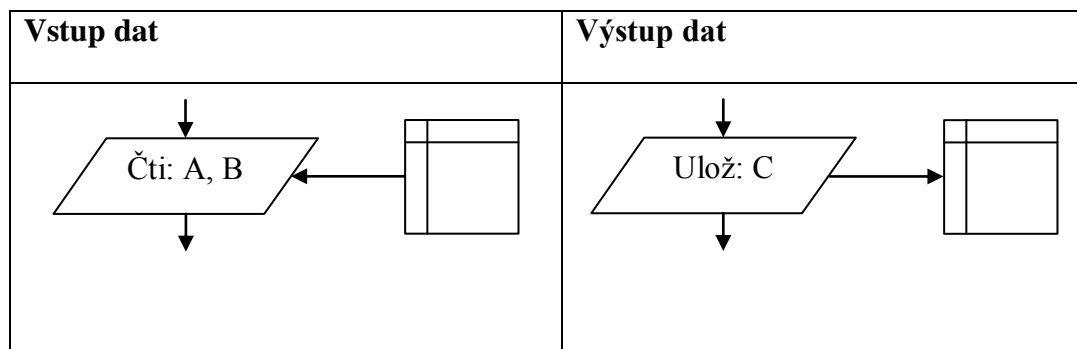
4. *Příprava* – tímto symbolem se označuje přípravnou fázi programu. Užívá se například pro zahájení cyklu o známém počtu opakování [4]. Symbol má dva vstupy. Jeden sekvenční a druhý pro návrat po provedení příslušného bloku operací. Dále má dva výstupy. Jeden vstupující do daného bloku operací a druhý sekvenční, který pokračuje do další části programu [3].



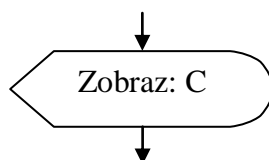
Obr. 1. Ukázka použití symbolu

5. *Interní paměť* – Symbol představuje nosič dat v případě, že nosičem je vnitřní paměť počítače. Data v tomto případě mohou do programu tohoto nosiče vstupovat, nebo se mohou z programu do této paměti ukládat. Symbol má pouze jeden vstup nebo jeden výstup [3].

Tab. 5. Interní paměť – vstup a výstup dat



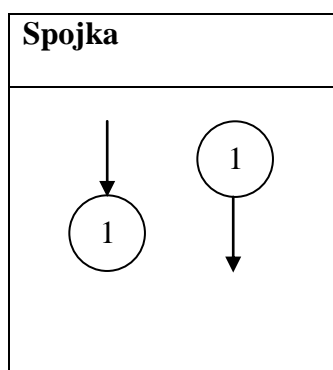
6. *Zobrazení* – tento symbol představuje zařízení pro vizuální zobrazení dat, jako je například monitor počítače či signální zařízení. Symbol má jenom jeden vstup [3].



Obr. 2. Symbol
zobrazení


7. *Spojka* – umožňuje spojit dvě části vývojového diagramu, které nebylo možné nakreslit souvisle. Spojky na konci přerušení a na začátku pokračování musí být označeny stejnými čísly [4].

Tab. 6. Symboly spojky

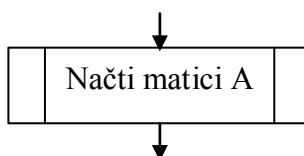


8. *Spojnice* – je symbol ve tvaru svislé nebo vodorovné čáry, která představuje tok dat nebo řízení. Slouží ke spojení jednotlivých symbolů ve vývojovém diagramu. Standardní směr toku informací je shora dolů a zleva doprava. To znamená, že spojnice by měly do symbolů vstupovat dole nebo vpravo [3].

Tab. 7. Symboly spojnic

Spojnice


9. *Podprogram* – jde o symbol, který znázorňuje část programu. Ten může obsahovat větší množství kroků [5]. Například pokud se vyskytuje v algoritmu nějaká stejná část na více místech, pak je vhodné tuto část zpracovávat samostatně. Část programu se označí jako jeden blok a ten se pak vloží do výsledného algoritmu na všechna potřebná místa. Ušetří se tím práce při vytváření vývojového diagramu a rovněž se tento diagram zpřehlední.

Obr. 3. – Symbol
podprogramu

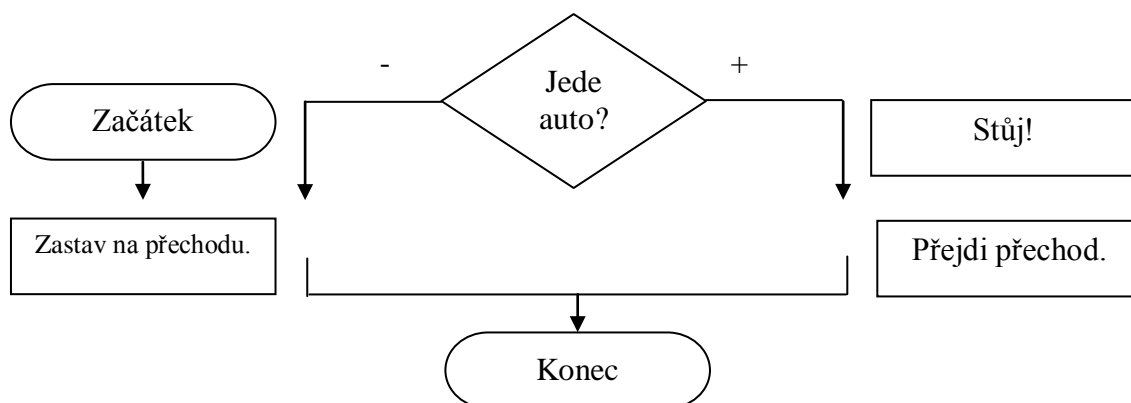
2.2 Otázky z probraného tématu

- ❖ Co víš o vývojovém diagramu?
- ❖ Proč se používají symboly ve vývojových diagramech?
- ❖ Jaké znáš mezní značky, k čemu jsou ve vývojovém diagramu použity?
- ❖ Jaké znáš sekvenční bloky, k čemu jsou ve vývojovém diagramu použity?
- ❖ K čemu jsou ve vývojovém diagramu použity podmínky?
- ❖ Nakresli symbol – mezní značky, sekvenční bloky, rozhodovací bloky, zobrazení, spojky.

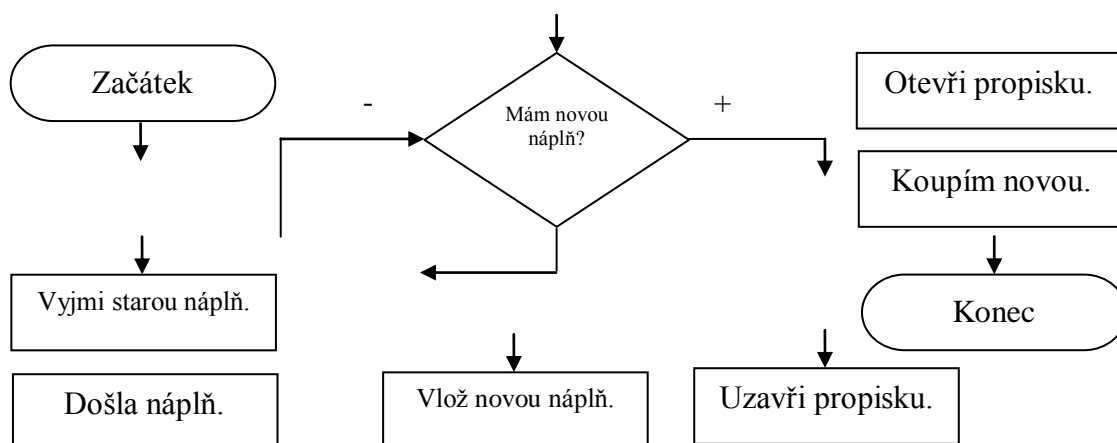
2.3 Praktické cvičené na dané téma

V programu MS Word nebo ručně na papír sestav algoritmy, pomocí dostupných symbolů pro následující úlohy:

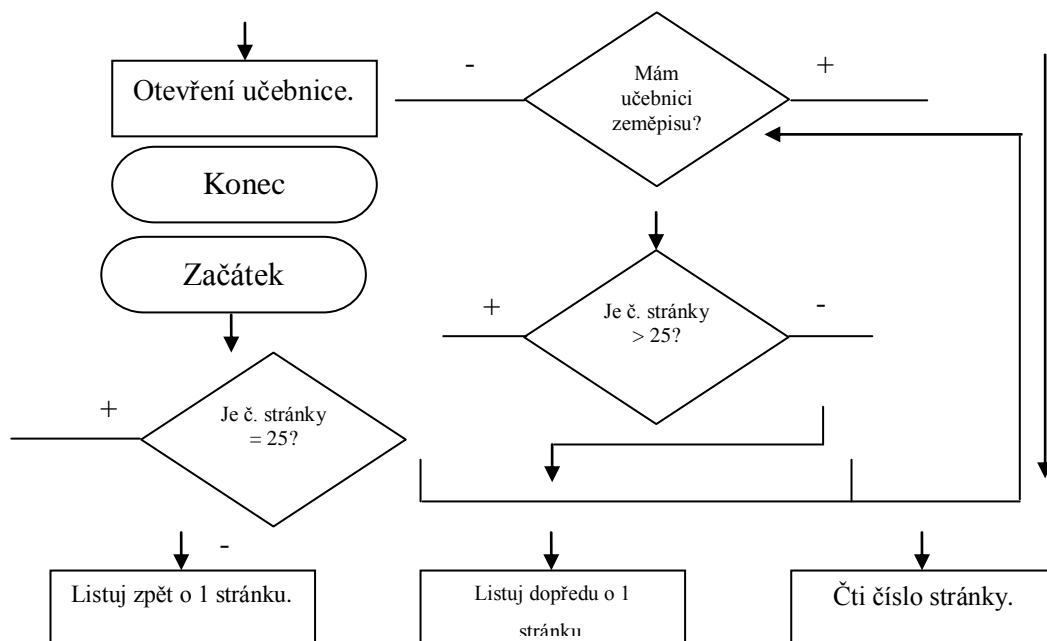
- (6) Vhodně slož algoritmus popisující přejítí cesty přes přechod.



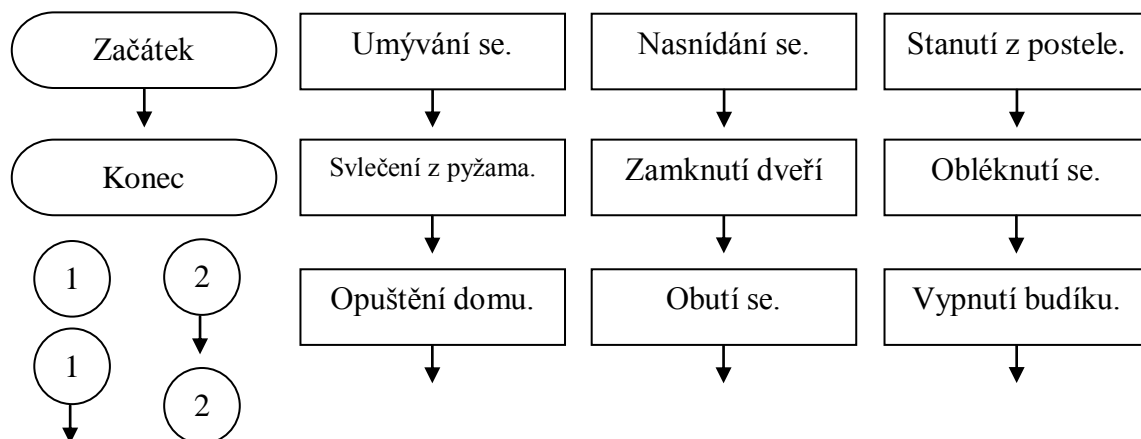
- (7) Vhodně slož algoritmus popisující výměnu náplně do propisky.



(8) Vhodně slož algoritmus popisující nalistování stránky 25 v učebnici zeměpisu.

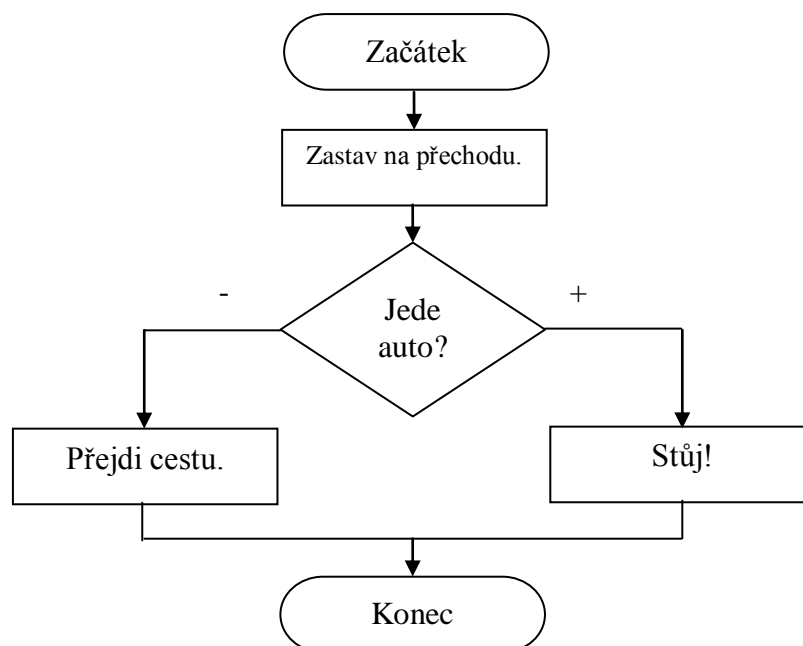


(9) Poskládej sekvenční blok algoritmu znázorňující každodenní posloupnost úkonů, které je nutné vykonat, než odejdeš do školy za využití symbolů spojek.

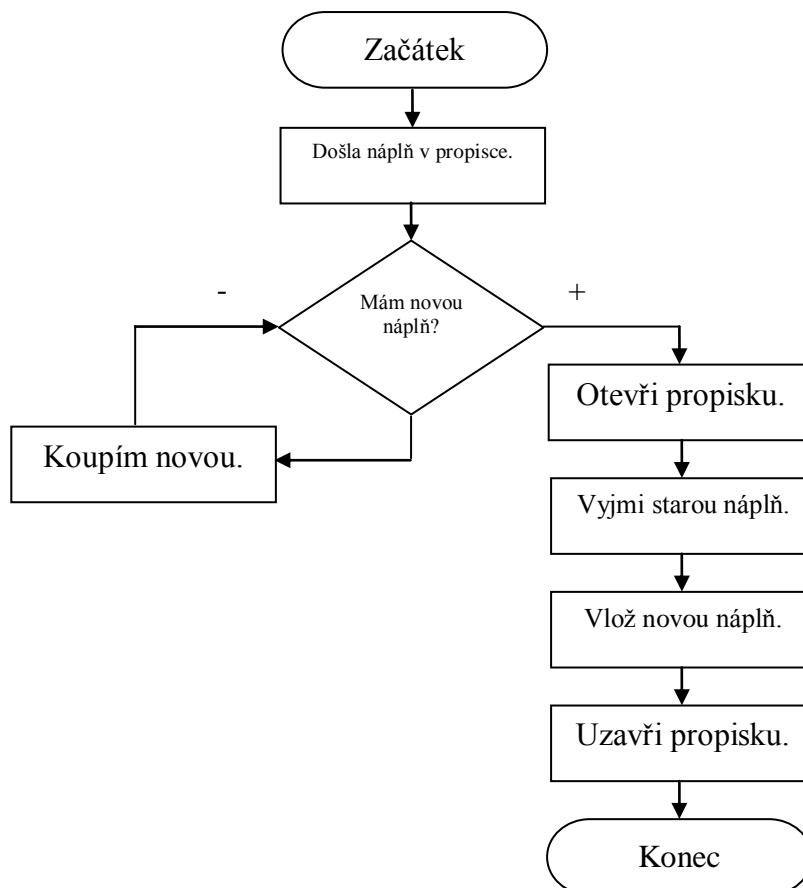


2.4 Řešení praktického cvičení

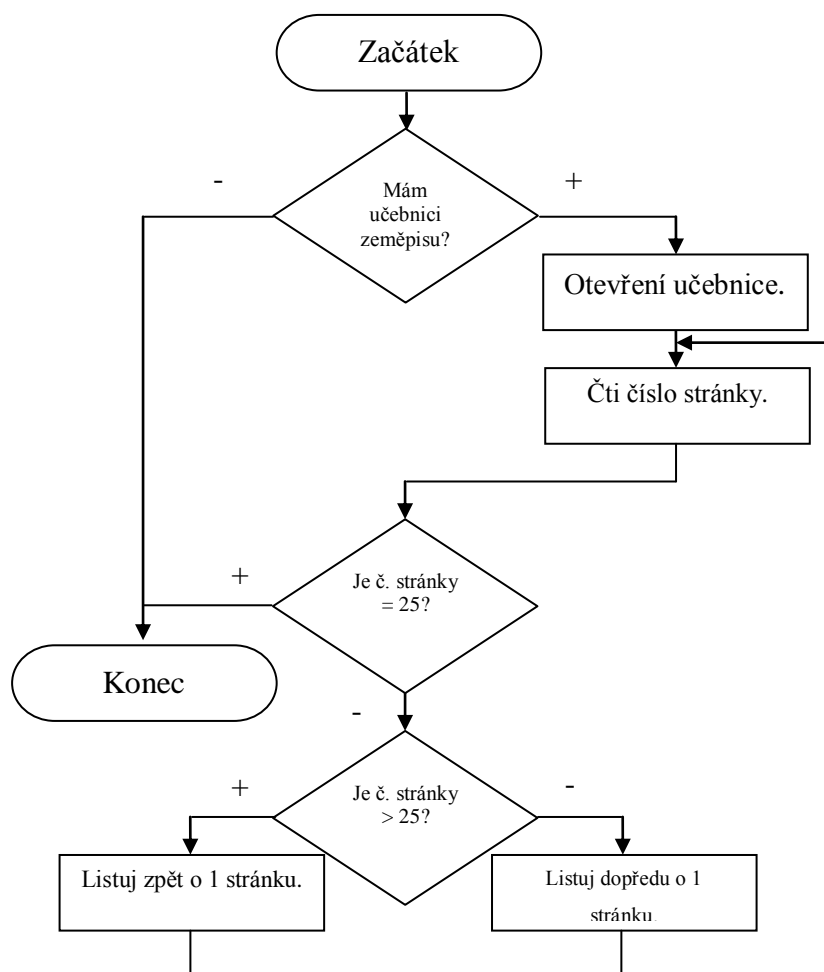
(6) *Přejítí cesty přes přechod.*



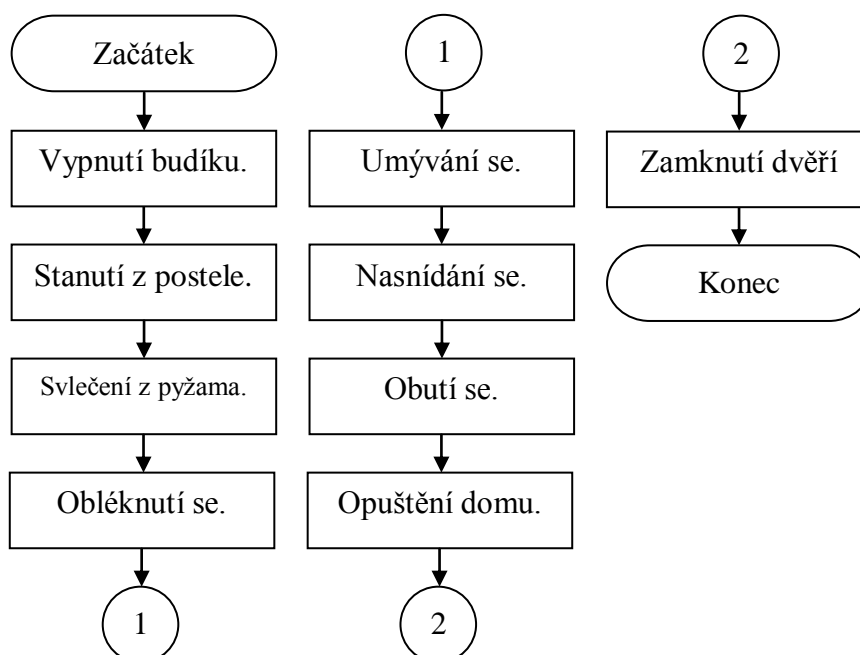
(7) *Výměna náplně do propisky.*



(8) *Nalistování stránky 25 v učebnici zeměpisu.*



(9) *Popis každodenní posloupnost úkonů, než odejdeš do školy za využití symbolů spojek.*



2.5 Seznam použité literatury

- [1] ČSN ISO 5807. *Zpracování informací: Dokumentační symboly a konvence pro vývojové diagramy toku dat, programu a systému, síťové diagramy programu a diagramy zdrojů systému*. Praha: TNK 20 Informační technika, 1996.
- [2] ČSN 36 9030. *Počítače a systémy zpracování dat: Symboly vývojových diagramů pro systémy zpracování dat*. Praha: VUAP Engineering A. S., 1974.
- [3] ČSN ISO 31-11. *Veličiny a jednotky: Část 11: Matematické znaky a značky používané ve fyzikálních vědách a v technice*. Praha: ADVIS, 1999.
- [4] PŠENČÍKOVÁ, Jana. *Algoritmizace*. Vyd. 2. Kralice na Hané: Computer Media, 2009, 128 s. ISBN 978-80-7402-034-6.
- [5] KRESLÍKOVÁ, Jitka. *Základy programování*. Brno: Fakulta informačních technologií VUT v Brně, 2002.

3 METODICKÝ LIST – ČÍSLO 3



CÍLEM TÉMATU – je čtenáře seznámit s třemi základními typy algoritmů a jejich použití. Dále jsou uvedeny nejčastější chyby při tvorbě algoritmu.



KLÍČOVÁ SLOVA – sekvence, větvení, cykly, přiřazovací příkaz



ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 90 minut.

3.1 Typy vývojových diagramů prakticky

Při vytváření algoritmu za pomoci vývojových diagramů rozeznáváme tři základní typy algoritmů. Jedná se o *sekvenci*, *větvení* a *cykly*.

3.1.1 Sekvence

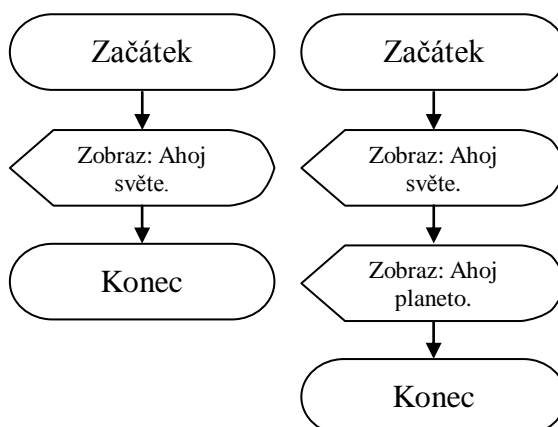
Sekvence patří k nejjednodušším typům algoritmu, který se skládá krom mezních značek pouze ze sekvenčních bloků [1].



TIP – Během sekvence nesmí docházet k větvení algoritmu ani k návratu zpět k cyklu [1].

Autor algoritmu se neobejde při jeho vytváření bez sekvence. Sekvence je základním stavebním prvkem každého algoritmu od toho nejjednoduššího až po ten nejsložitější. Tudíž by se dalo říci, že sekvence je nejpoužívanějším typem algoritmu.

Začneme velice jednoduše a ukážeme si algoritmus programu s jednou sekvencí, jehož úkolem bude vypsát na obrazovku větu „Ahoj světe.“. Druhý algoritmus bude vycházet z algoritmu prvního. Bude jen o něco složitější, jelikož se bude jednat o algoritmus s dvěma sekvenčními bloky. Jeho úkolem bude navíc zobrazení věty „Ahoj planeto.“. Příklad sekvence s jedním a s dvěma sekvenčními bloky:



Přiřazovací příkaz – značí se symbolem dvojtečky s rovnítkem ($:=$). Tento operátor má charakter podobně jako rovnítko, ale ještě o něco navíc. Příkladem může být součet dvou čísel $C:=A+B$, kde lze do proměnné C přímo přiřadit výsledek matematické operace součtu čísel A a B.

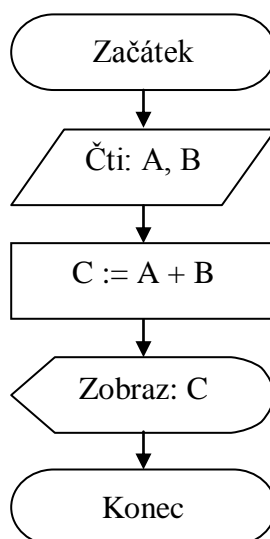


POZNÁMKA – takto lze použít i další matematické operace odečítání, násobení, dělení a další možné složitější funkce. Nová hodnota buňky je dána nějakou funkcí hodnoty původní buňky.



TIP – proměnné slouží jednoduše řečeno jako vymezený úsek paměťových buněk, kam si lze ukládat libovolná data.

Příklad: Algoritmus, který vykonává součet dvou čísel za použití 3 proměnných A, B a C.



V následující tabulce jsou uvedeny povolené a zakázané operace v sekvenčních algoritmech.

Operace	Rozdíl	Součin	Mocnina	Logický součet	Logický součin	Negace
Značka	-	*	SQR	OR	AND	NOT



Podíl „/“ nebo *Celočíselné dělení „DIV“* u těchto operací je nutné brát v potaz, že při dělení musí být jmenovatel různý od nuly. Výsledek dělení nulou by byla neurčitá hodnota, tento fakt je nutné ošetřit větvením. A tudíž se již nejedná o sekvenci.



Odmocnina u této matematické operace je nutné brát v potaz, že pokud se neprovádí výpočty s komplexními čísly, nesmí být výraz pod odmocninou záporný. V algoritmu se tato skutečnost ošetří větvením. A tudíž se již nejedná o sekvenci.



Cyklus je jedním z nejsilnějších nástrojů algoritmu, ale u těchto typů algoritmu dochází k návratu zpět a tudíž se nejedná o sekvenci.

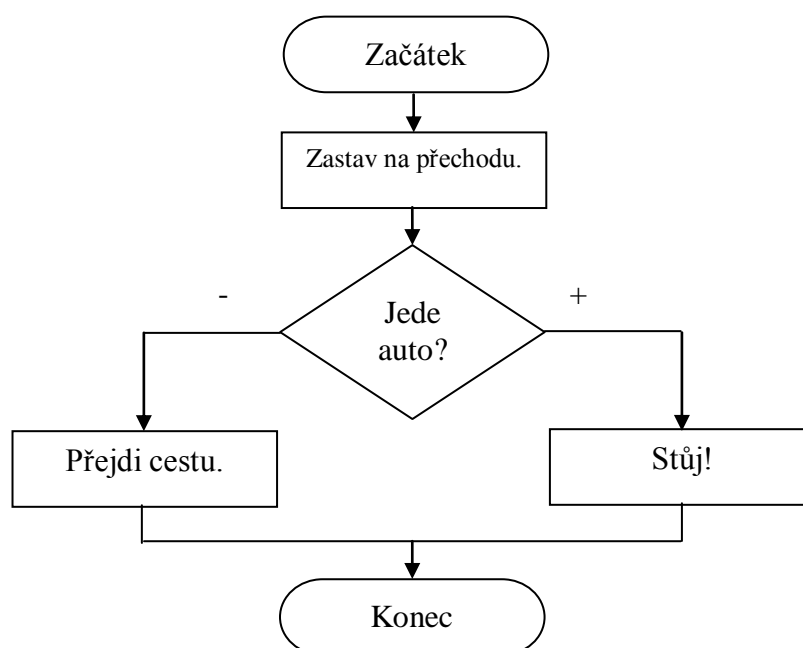
3.1.2 Větvení

Větvení se využívá v algoritmu při ošetření žádoucích nebo nežádoucích důsledků, které při průchodu algoritmu mohou nastat. Zejména ošetření nežádoucích důsledků je autory algoritmu a následně i programátory využíván nejčastěji. Správný algoritmus musí totiž vždy skončit. A proto musí mít pomocí větvení předpřipravenou cestu bez jakýchkoliv problémů.



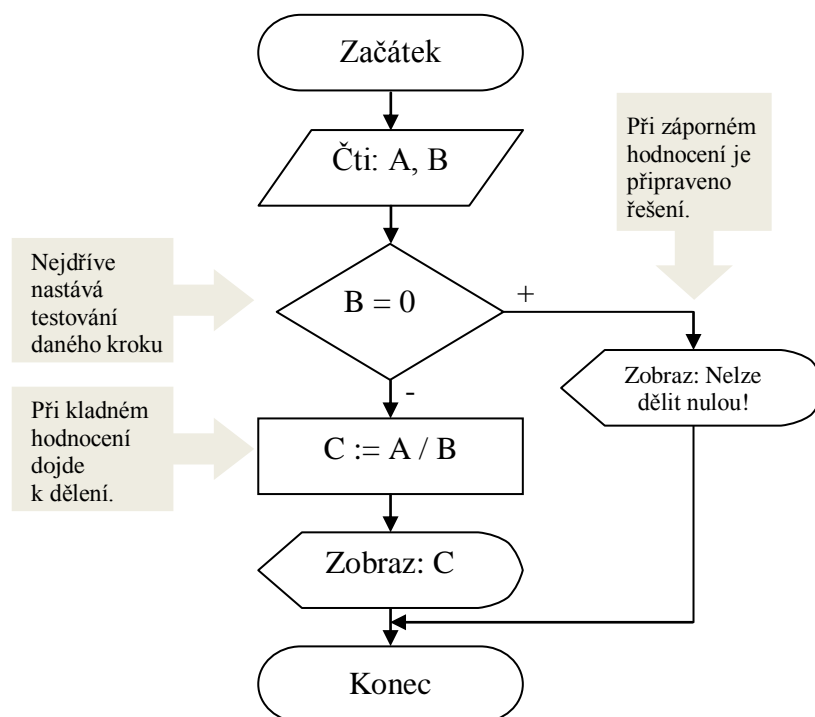
POZNÁMKA – Než začne autor tvořit algoritmus, měl by dobře analyzovat veškeré možnosti, které mohou nastat. K jakým problémům by mohlo dojít, a tyto problémy by měl autor za pomoci větvení ošetřit. Před samotným větvením by mělo nastat testování, jestli daný krok je správný, pokud bude nastane větvení.

Vhodným příkladem ze života je přejití cesty přes přechod. Algoritmus takového chování by vypadal následovně:

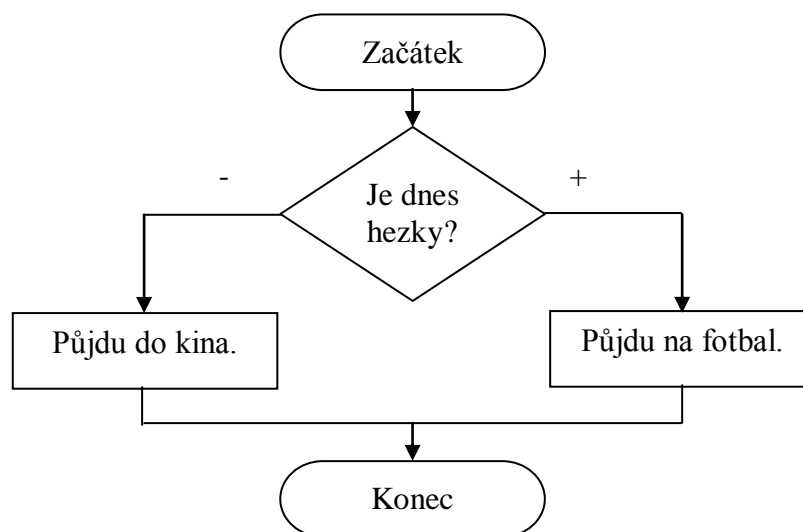


V matematických operacích se vyskytuje mnoho skrytých hrozeb, které je nutné ošetřit větvením, aby nedošlo k havárii algoritmu. Nejčastější se ošetřují důsledky při dělení, výpočtu výrazu pod odmocninou a obdobných funkcí.

Příklad: Algoritmus s větvením k ošetření nežádoucího důsledku při matematické operaci dělení nulou.



Větvení může nastat i při ošetření žádoucího důvodu. Názorným příkladem ze života může být stav, kdy se člověk rozmýšlí, co bude dělat, když je hezký den.

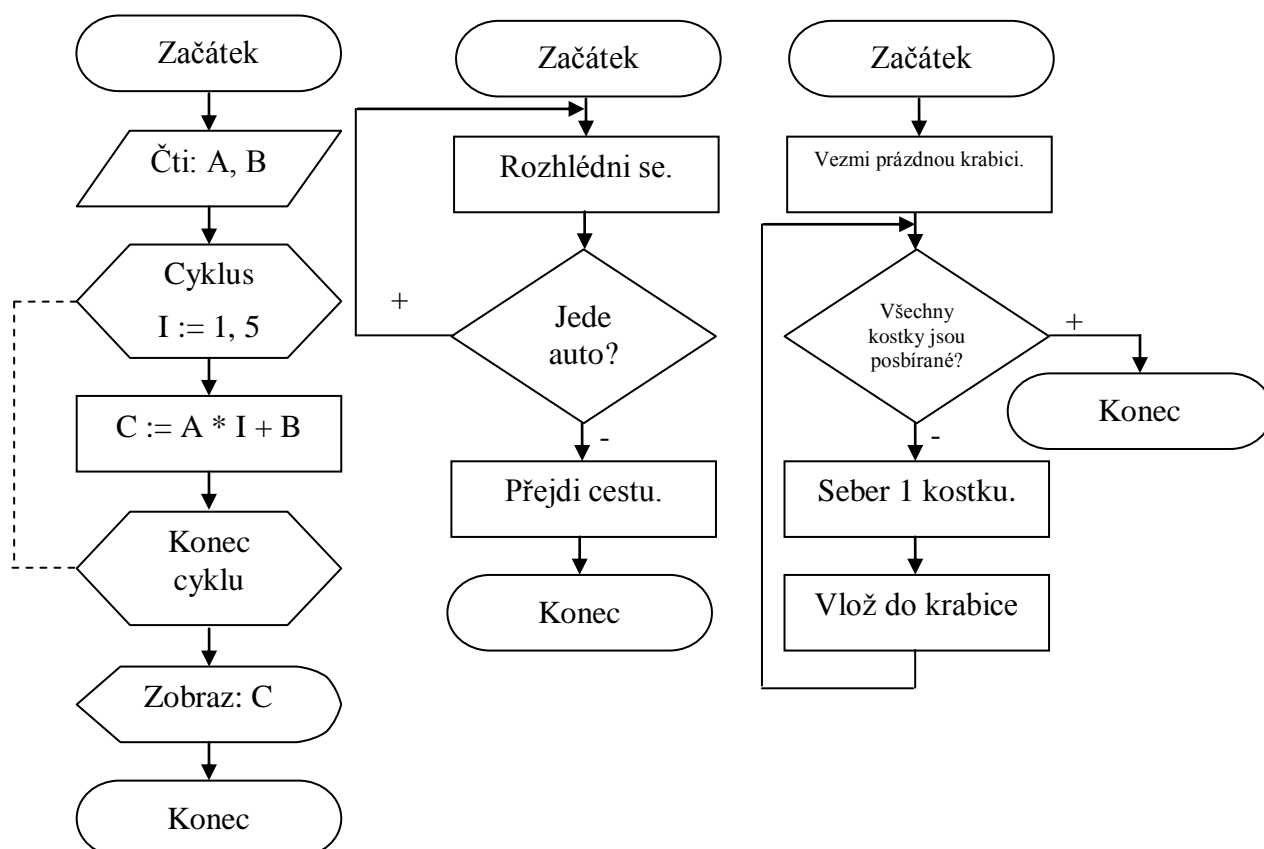


3.1.3 Cykly

Třetím typem algoritmů jsou cykly. Jedná se o nejsilnější nástroj algoritmů. Hlavním úkolem cyklu je opakování určité části programu, buď to se stejnými daty, nebo pokaždé s odlišnými. Je velice důležité dodržet konečnost algoritmu, protože nesmí žádný cyklus běžet nekonečně dlouho. Proto před začátkem každého cyklu by měly být přesně stanoveny jeho podmínky a počet opakování, aby byl cyklus včas ukončen.

Podle způsobu řízení opakování se rozeznávají tři typy cyklů:

- Cykly s pevným počtem opakování – v tomto typu cyklu je pevně stanoven počet opakování.
- Řízené cykly s podmínkou na začátku cyklu – v tomto typu cyklu nelze dopředu určit, kolikrát se bude cyklus opakovat. Do cyklu se vstoupí, jestliže je podmínka splněna. Jestliže podmínka na začátku cyklu přestane platit, cyklus se opustí a pokračuje se dalším krokem algoritmu. Není-li podmínka splněna na začátku cyklu, pak se do cyklu vůbec nevstoupí.
- Řízené cykly s podmínkou na konci cyklu – v tomto typu cyklu nelze dopředu určit, kolikrát se bude cyklus opakovat. Do cyklu se vstoupí vždy a proběhne nejméně jednou. Cyklus se bude opakovat do té doby, než bude podmínka na konci cyklu splněna. Algoritmus pokračuje dalším krokem až poté, kdy je cyklus opuštěn.

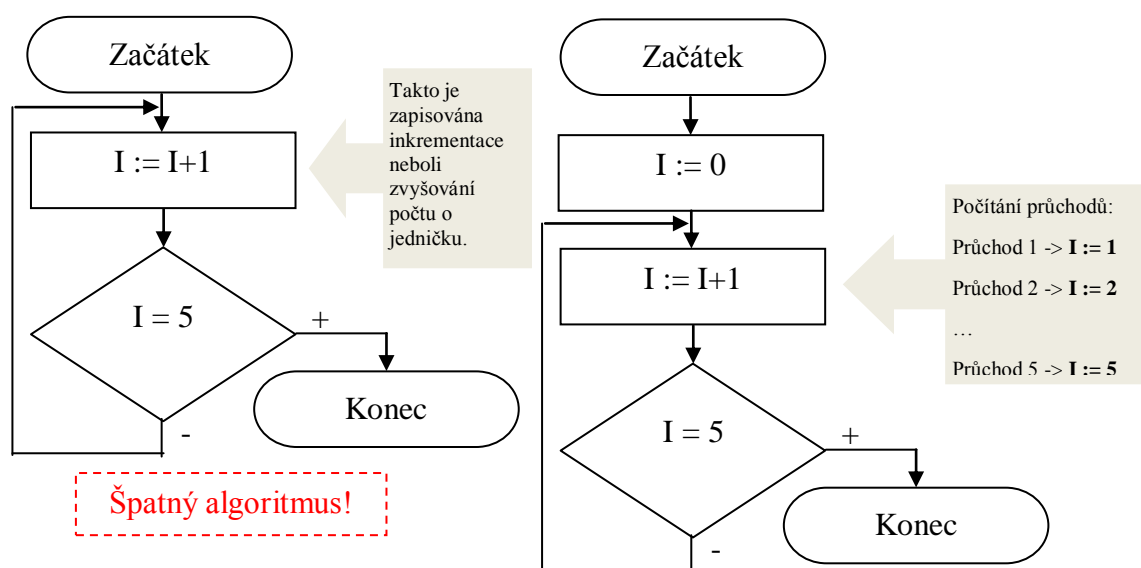


3.1.4 Nejčastější chyby v algoritmu

Chyba s konečností - jednou z častých chyb je problém s konečností algoritmu. Vlevo je ukázka špatně vytvořeného algoritmu, kde není dodržena vlastnost konečnosti algoritmu. Tato chyba patří na první pohled mezi ty méně patrné. Na počátku algoritmu není zřejmé, jestli proměnná I již není větší než hodnota 5. Pokud by byla, algoritmus by poté neustále přičítal číslo jedna do proměnné I , ale k hodnotě 5 by již nikdy algoritmus nedospěl. Tím by nastala nekonečná smyčka. Vpravo je již ošetřený algoritmus před touto chybou, kdy je před započítáním cyklu hodnota proměnné I vynulována.



TIP – Pokud programátor realizuje „nakóduje“ program podle algoritmu, který není konečný. Ve výsledné aplikaci se toto chování projeví tak, že program bude opakovat neustále stejné kroky, přestane odpovídat „zatuhne“ nebo systém havaruje z důvodu přetečení paměti.

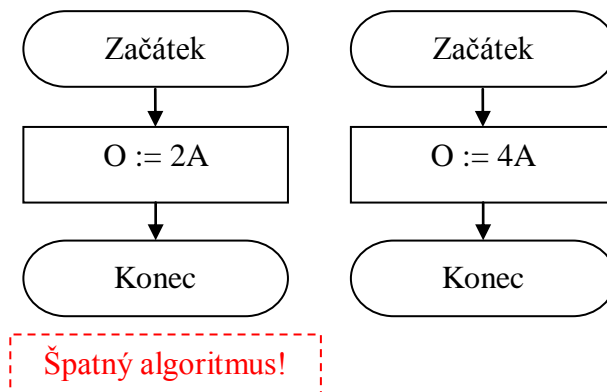


Chyba špatného přepisu vzorce – podmínka věcné správnosti je velmi důležitá vlastnost algoritmu. Pokud autor algoritmu špatně použije vzorec pro numerický výpočet veškeré informace a poučky, které zná o algoritmech jsou rázem k ničemu.



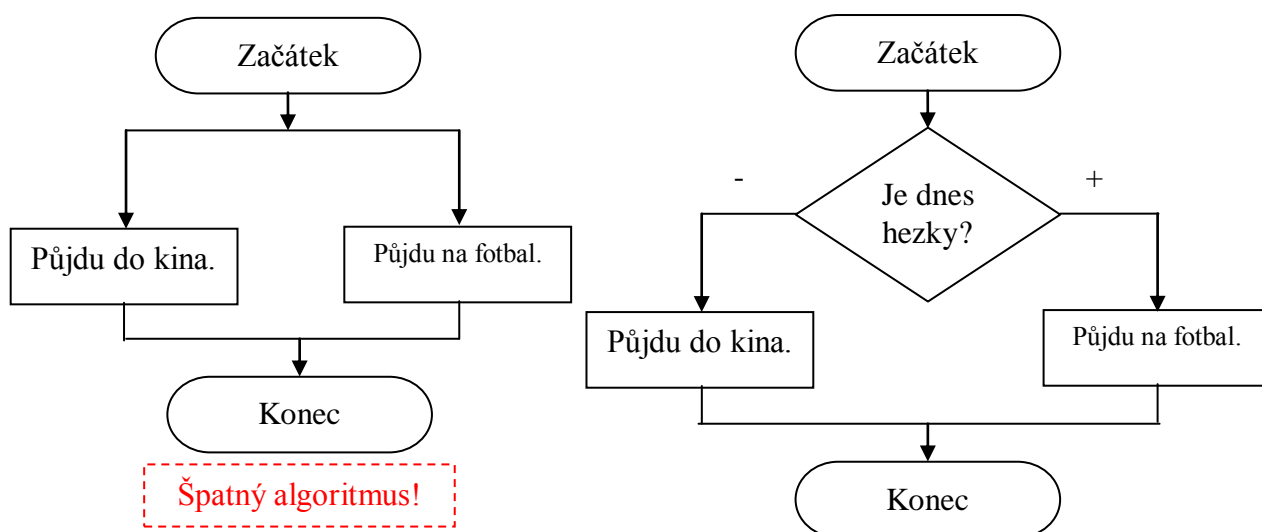
TIP – Chyba věcné správnosti je nejčastější chybou začínajících autorů vývojových diagramů. Je zapotřebí dát si pozor na chybu ve vzorečku. Použití zlomkové čáry, kdy se namísto ní používá symbol lomítka „/“ nebo znak odmocniny $\sqrt{\quad}$ je nahrazen výrazem SQRT.

Příklad pro výpočet obvodu čtverce. Vlevo je algoritmus se špatným vzorcem, vpravo je již algoritmus se správným vzorcem pro výpočet obvodu čtverce. Algoritmus by fungoval, ale uživatel by dostával nesprávný výsledek.



Determinovanost – nedodržení vlastnosti determinovanosti algoritmu, někdy také označován jako jednoznačnost algoritmu, bývá velmi častým problémem. Autor při tvorbě vývojového diagramu musí brát v potaz veškeré možnosti, které mohou během zpracování algoritmu nastat. Po ukončení aktuálního kroku v algoritmu musí následovat jednoznačný další krok nebo algoritmus musí být ukončen.

Příklad jednoznačnosti algoritmu. Vlevo je uveden algoritmus, u kterého je zřejmé porušení vlastnosti jednoznačnosti. Algoritmus neví podle čeho se rozhodnout a jakou cestu má zvolit. Vpravo je již algoritmus doplněn o větvení, kdy po vyhodnocení podmínky algoritmus přesně ví, jakým krokem bude pokračovat.



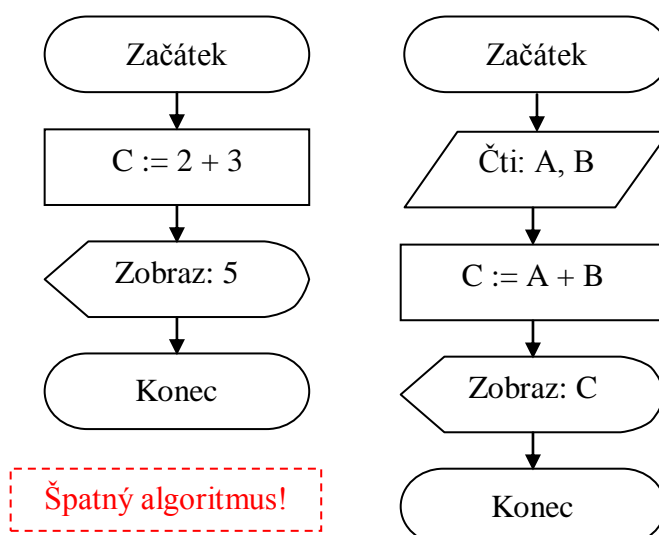
Chyba dělení – jedná se o celou řadu matematických a logických úloh, ve kterých se musí autor zabývat podmínkami řešitelnosti algoritmu:

- Výrazy se zlomky – je třeba dávat pozor, aby jmenovatel se nerovnal nule, jelikož nulou dělit nejde. Jedná se o jednu z nejčastějších a nejzávažnějších chyb v programech.
- Odmocniny – řešení v oboru reálných čísel nesmí být záporné.

Ošetření této chyby se provádí za pomoci větvení, kdy po splnění podmínky lze pokračovat dále v algoritmu nebo je zobrazeno varování a algoritmus je ukončen.

Chyba hromadnosti – autor musí vždy realizovat algoritmus tak, aby byl co nejobecnější a mohl plnit velkou řadu podobných úloh.

Příklad součtu dvou čísel. Vlevo je uveden příklad algoritmu, který sečte dvě známá čísla. Jedná se o velice jednoduchý algoritmus, ale pokud by uživatel potřeboval sečíst jiná čísla, než jsou v algoritmu použity, musel by vytvořit algoritmus nový. Vpravo je proto uveden algoritmus, který získá od uživatele libovolně zadaná čísla a ty poté bude sčítat. Takto zobecněný algoritmus lze nadále rozvíjet o další matematické operace sčítání, násobení, dělení a vytvořit tak jednoduchou kalkulačku.



3.2 Otázky z probraného tématu

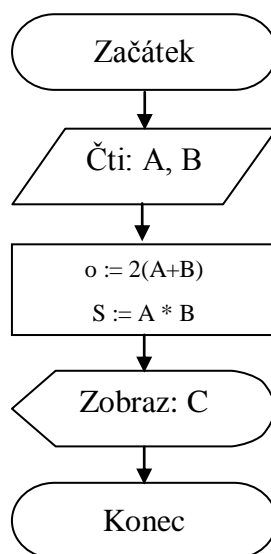
- ❖ Jaké jsou tři základní typy algoritmů?
- ❖ Co si pamatuješ o sekvencích?
- ❖ Kdy se nesmí sekvence používat?
- ❖ Co si pamatuješ o větvení?
- ❖ K čemu se používá větvení v algoritmech?
- ❖ Co si pamatuješ o cyklech?
- ❖ Jaké tři typy cyklů dělené podle způsobu řízení znáš?

3.3 Praktické cvičené na dané téma

- (10) Sekvence - realizuj algoritmus pro výpočet obvodu a obsahu obdélníku. Dodrž vlastnost konečnosti a hromadnosti. Použij vzorce $o = 2(A+B)$; $S = A * B$
- (11) Větvení – využij algoritmus, který ošetřil chybu dělení nulou a vytvoř podle něj algoritmus pro výpočet výrazu $C = A / (B + D)$
- (12) Cykly – vytvoř algoritmus pomocí cyklů, který bude mít za úkol vypsát na displej text „Ahoj“. Uživatel bude moci zadat počet opakování, kolikrát se text na displeji vypíše.

3.4 Řešení praktického cvičení

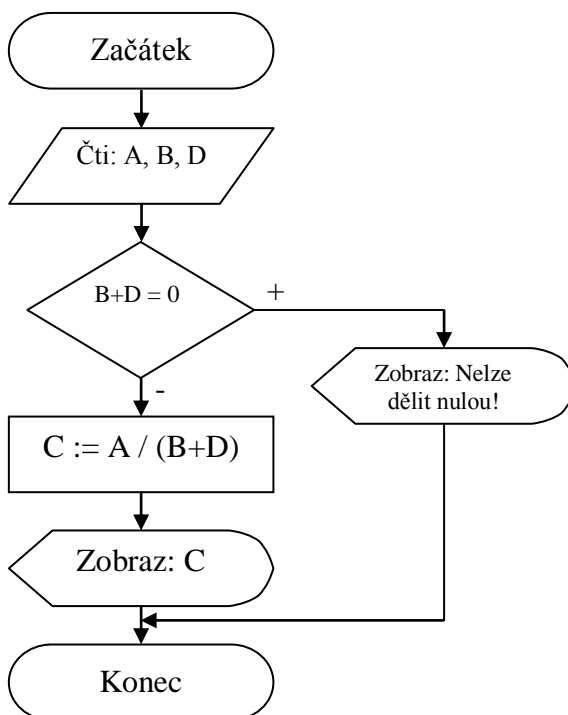
- (10) *Výpočet obvodu a obsahu obdélníku*



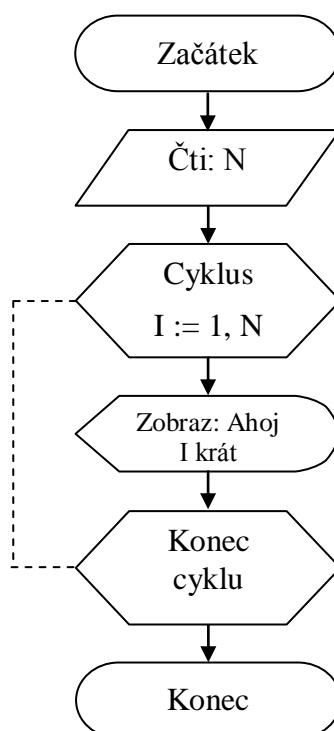
(11) *Algoritmus pro výpočet výrazu $C = A / (B + D)$*



POZNÁMKA – U výrazů se zlomky je zapotřebí použít závorky ve výrazu kvůli přednosti jejich vyhodnocení. Jinak má přednost násobení a dělení před sečítáním a odečítáním.



(12) *Algoritmus pro výpis textu „Ahoj“ s možností zadání počtu opakování*



3.5 Seznam použité literatury

- [1] PŠENČÍKOVÁ, Jana. *Algoritmizace*. Vyd. 2. Kralice na Hané: Computer Media, 2009, 128 s. ISBN 978-80-7402-034-6.

4 METODICKÝ LIST – ČÍSLO 4



CÍLEM TÉMATU – je čtenáře seznámit s vývojovým prostředím Microsoft Visual C# 2010 Express Edition



KLÍČOVÁ SLOVA – vývojové prostředí, intellisense, direktiva



ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 90 minut.

4.1 Vývojové prostředí Microsoft Visual C# 2010 Express Edition


Vývojové prostředí je software, který umožňuje programátorovi vytvořit program podle navrženého algoritmu. Obvykle je vývojové prostředí vázané k jednomu specifickému programovacímu jazyku.

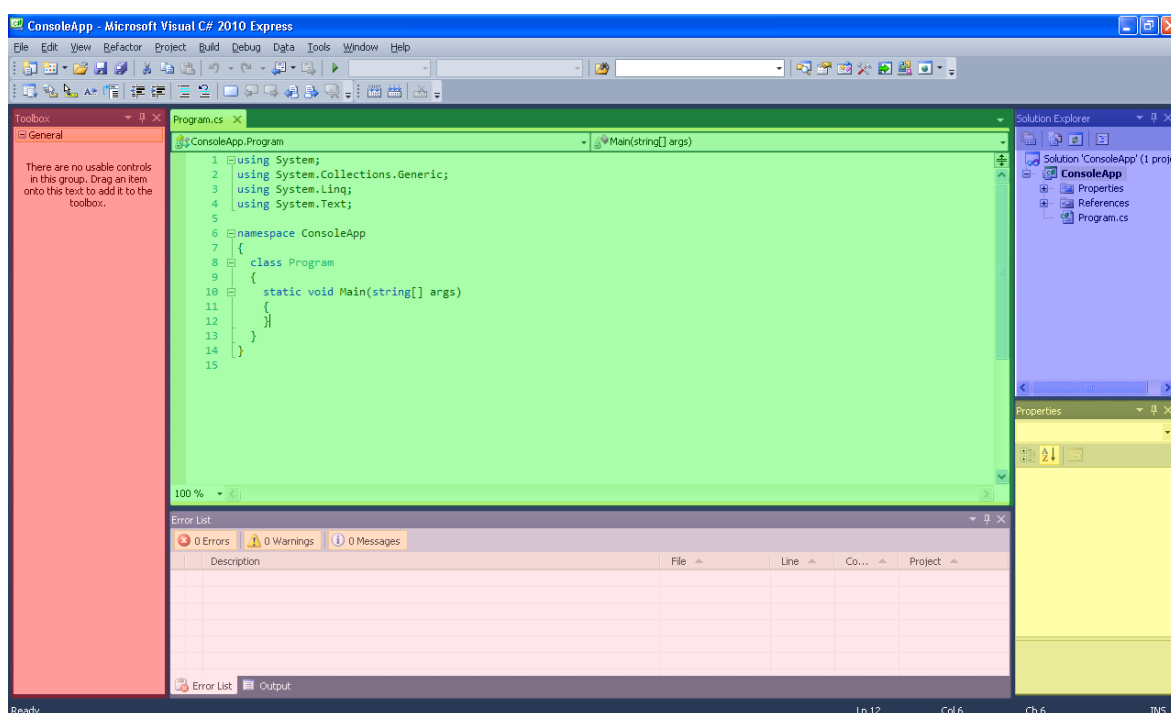
Veškeré výukové programy jsou vytvořené ve vývojovém prostředí Microsoft Visual C# 2010 Express Edition. Toto vývojové prostředí je nabízené od společnosti Microsoft zdarma, jediné co uživatel musí po instalaci udělat je registrace v prvních třiceti dnech používání. Svými možnostmi dokáže toto prostředí mnohé vývojáře oslnit. Obsahuje bohatou výbavu a podporou nejnovějších technologií. Program má nový moderní vzhled rozhraní, vhodný zejména pro začínající a mírně pokročilé programátory.

Většina vývojových prostředí je rozdělena do několika částí mezi ty základní patří textový editor (zapisuje se do něj zdrojový kód), debugger (ladící program na vyhledávání chyb ve zdrojovém kódu) a překladač, který překládá zdrojový kód vytvořený programátorem do kódu strojového. Procesor totiž pracuje jen s kódem strojovým a veškeré zdrojové kódy se do něj musí přeložit.

IDE (Integrated Desktop Environment) tedy integrované vývojové prostředí Microsoft Visual C# 2010 Express Edition je rozděleno do několika částí s panely:

- Toolbox panel – obsahuje různé prvky, které se umísťují na pracovní plochu designéru (červeně na obr. 4)
- Main Window – neboli editor kódu, tedy místo kde se zdrojový kód píše. V tomto okně je umožněno barevné rozlišení kódu, podporu automatického doplnění kódu a automatickou opravu chyb (zeleně na obr. 4)
- Properties panel – editor vlastností pro úpravu vlastností objektu (žlutě na obr. 4)

- Solution Explorer panel – adresářová struktura pro správu souborů a objektů daného projektu (modře na obr. 4)
- Debugger – ladící program na odhalení chyb, spouští se ikonou zelené šipky  nebo pomocí klávesy F5.
- Kompilátor – překladač zdrojového kódu na kód strojový
- Online podporu – kontextová nápověda s vazbou na webovou knihovnu MSDN
- Panel Error list – uvádí řádek s pozicí chyby a popisuje nalezenou chybu

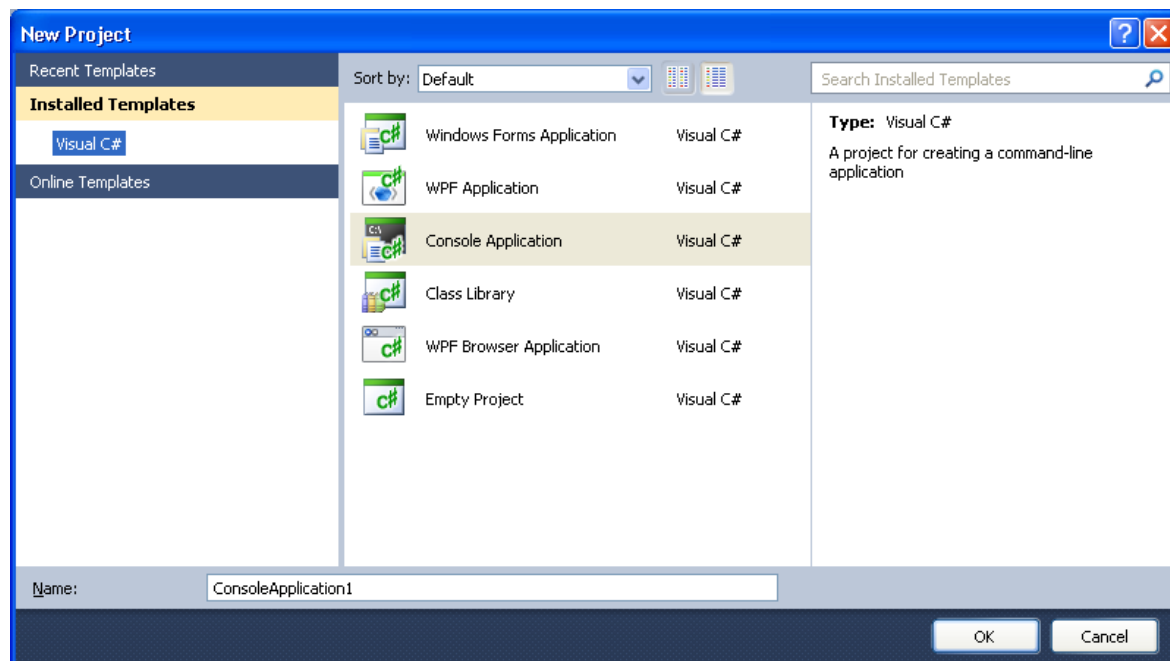


Obr. 4. Integrované prostředí vývojového prostředí Microsoft Visual C# 2010 Express

4.1.1 Založení nového projektu v Microsoft Visual C# 2010 Express

Ve vývojovém prostředí jde založit několik rozdílných projektů, my se seznámíme se založením konzolové aplikace a aplikace s formulářovým oknem.

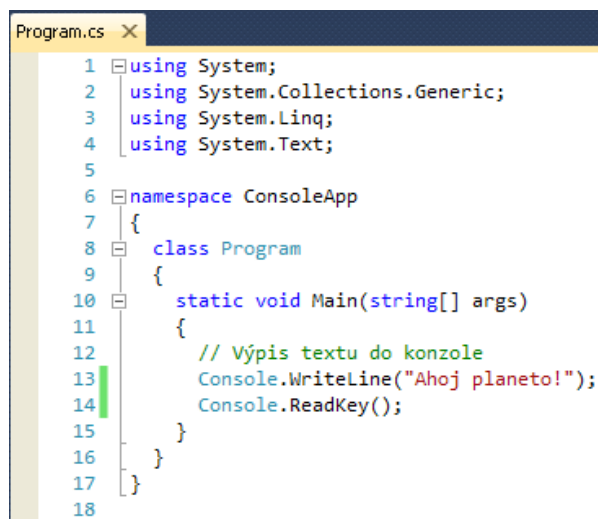
Založení konzolové nebo formulářové aplikace se provádí několika možnými způsoby. Jedním z nich je založení projektu přes menu File -> New Project... (Ctrl+Shift+N) a výběrem *Console Application*. Nebo přímo po startu programu lze na záložce *Start Page* vybrat pole *New project...* a zde lze vybrat projekt *Console Application*. Stejným způsobem se založí formulářový projekt, jen se vybírá projekt *Windows Forms Application*.



Obr. 5. Výběr typu projektu Console Application

4.1.2 Editor kódu

Jedná se o editor pro zápis zdrojový kód. Velkou grafickou pomocí je barevné zvýraznění syntaxe. Některé prvky syntaxe kódu jsou různě barevně odlišeny. Například klíčové slovo **using** jsou jednou barvou, ale typy **console** jsou barvou jinou. Ostatní prvky syntaxe jako řetězce nebo komentáře jsou také barevně odlišeny. Vývojář ocení dynamickou podporu neboli *intellisense*. Pokud je za klíčovým slovem použita tečka zobrazí se k tomuto objektu sada vlastností, metod, atributů či událostí, kterým klíčové slovo disponuje.



```

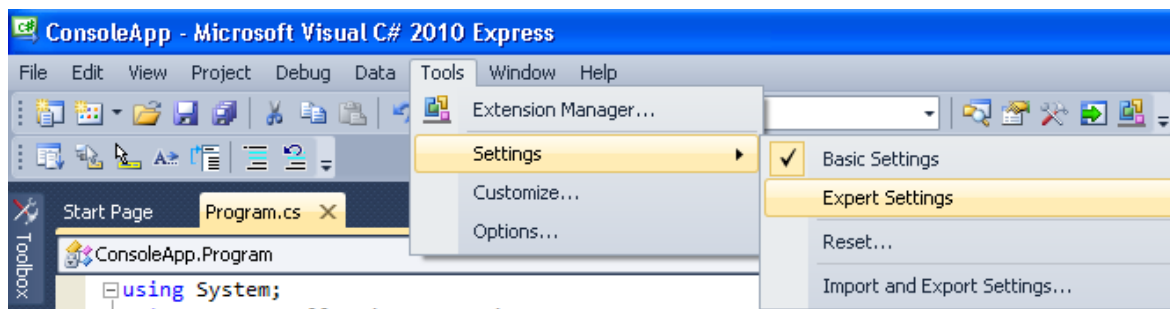
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ConsoleApp
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12             // Výpis textu do konzole
13             Console.WriteLine("Ahoj planeto!");
14             Console.ReadKey();
15         }
16     }
17 }
18

```


Obr. 6. Editor kódu s barevným zvýrazněním
syntaxe

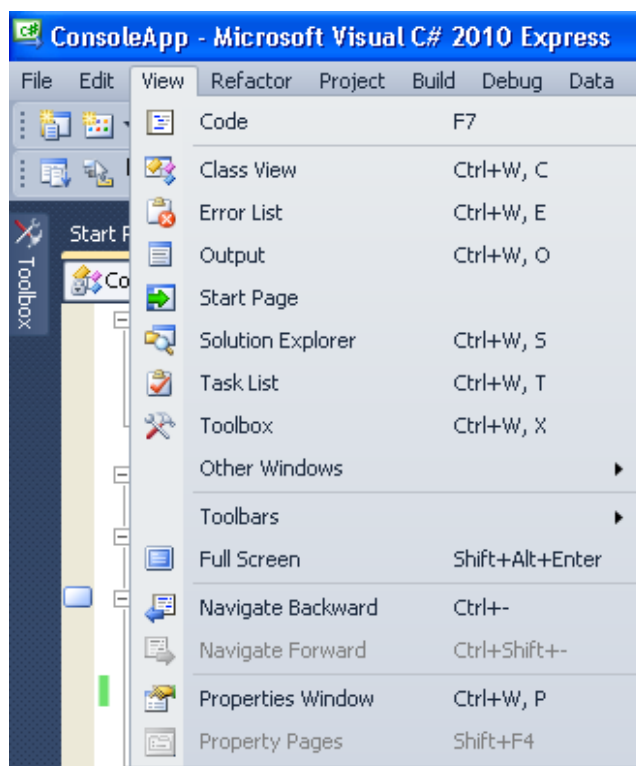
4.1.3 Úprava nastavení vývojového prostředí

Pro účelnější práci s vývojovým prostředím Microsoft Visual C# 2010 Express i přesto, že se jedná o začátečníka, je dobré si toto prostředí trochu poupravit podle obrazu svému. Začneme změnou nastavení z basic na expert. Tato změna se provádí v menu *Tools* -> *Settings* -> *Expert Settings*. Díky tomu se nástrojová lišta rozšíří o několik dalších polí.



Obr. 7. Změna nastavení vývojového prostředí

Druhým krokem by mělo být přidání panelů s vlastnostmi objektů, panel s chybovým hlášením a ukotvení panelu nástrojů. Přidání jednotlivých panelů se provádí v menu *View* a výběrem daného okna. Pro panel s vlastnosti je nutný výběr položky menu *Properties Window*, pro panel s chybovým hlášením je to *Error List*. Ukotvení panelu nástrojů se provádí ikonou v toolboxu 

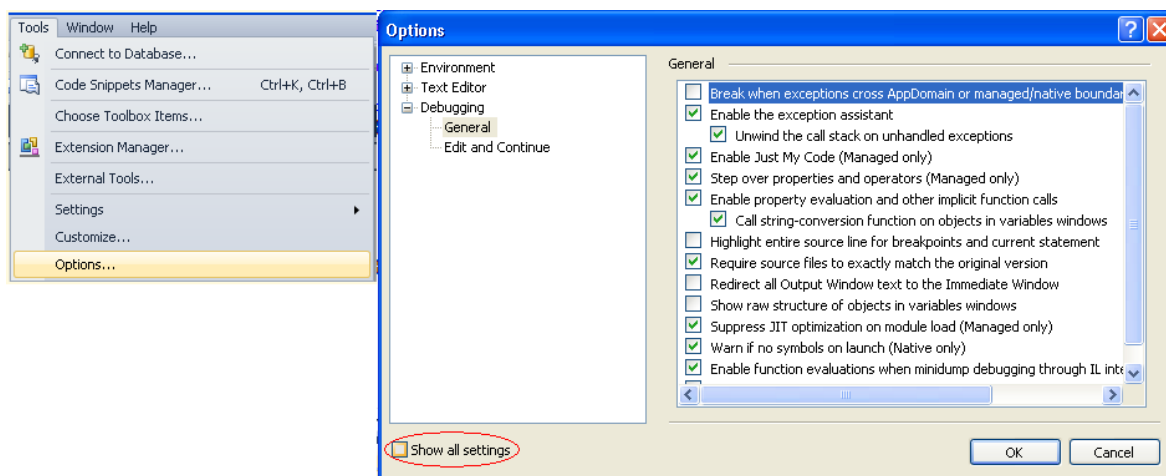


Obr. 8. Výběr panelů z menu View



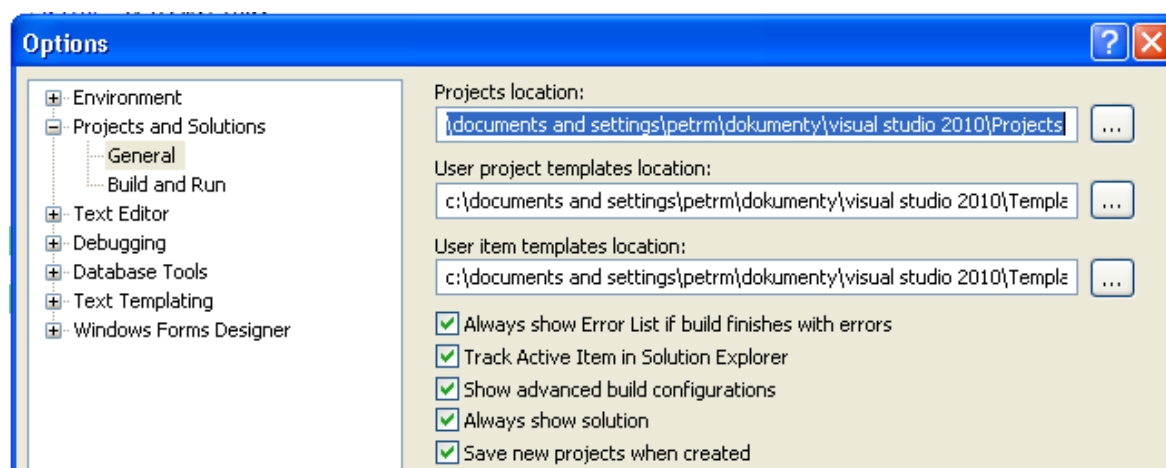
POZNÁMKA – V okně *Error List* můžete poklepat na libovolnou položku, čímž dojde k přemístění kurzoru na řádek, který způsobil danou chybu. Všimněte si také, že při psaní Vám vývojové prostředí podtrhává červenou vlnovkou jakékoliv místo v kódu, které způsobí chybu při překladu.

Posledním krokem je úprava editoru kódu. Je dobré si nastavit odsazení tabulátoru na 2 pozice, poté pro přehlednější hledání a velikosti kódu je vhodné si zapnout číslování řádků. A jako poslední je povolení možnosti uměny módu Debug a Release. V menu *Tools* zvolíme možnost *Options...* a zaškrtneme zaškrťovací pole Show all settings pro zobrazení veškerého nastavení.



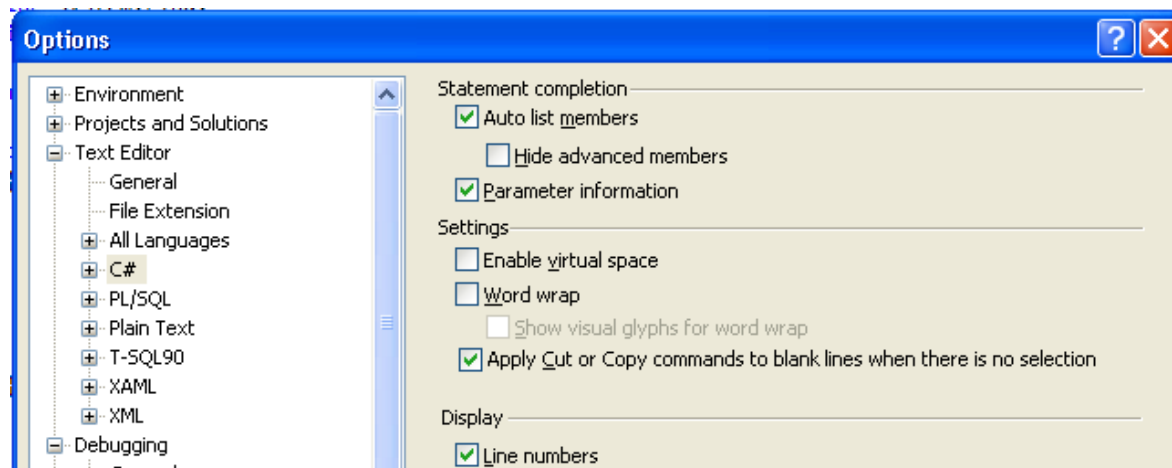
Obr. 9. Zobrazení dialogového okna možností

Tím je přidána větev nastavení *Projects and Solutions*, kde zaškrtně pole *Show advanced build configurations* a *Save new projects when created*. Tímto budou povolené módy Debug, Release a umožněno uložení projektů při jeho založení.

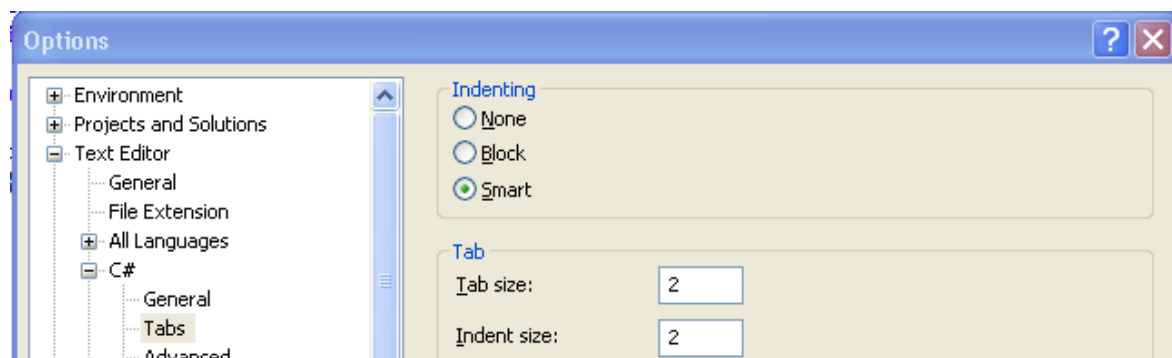


Obr. 10. Nastavení - Show advanced build configurations

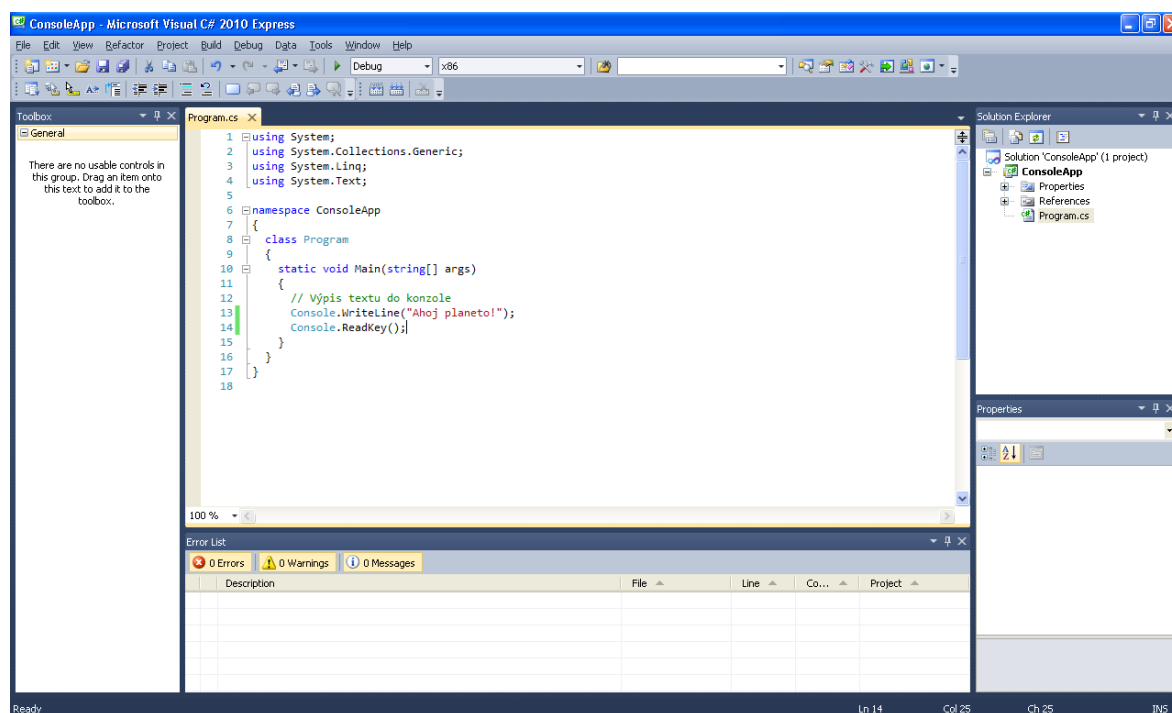
Zapnutí číslování řádků, se provádí zaškrtnutím položky *Line numbers* v *Text Editor* – *C#*. Pro nastavení odsazení tabulátoru je nutné větev *C#* otevřít a v ní je sekce *Tabs*. Zde hodnoty polí *Tab size* a *Indent size* změnit na 2. Tímto jsou veškeré změny v nastavení u konce a výsledek by se měl rovnat obrázku třináct uvedený níže.



Obr. 11. Nastavení číslování řádků



Obr. 12. Nastavení odsazení na hodnotu dvě



Obr. 13. Nově nastavené vývojového prostředí

4.1.4 Základní struktura programu

Prvním několik řádků kódů začíná direktivou `System`. Jedná se o základní jmenný prostor rámce .NET a říká jaké knihovny bude využívat.

```
using System;
```

Klíčové slovo `namespace` definuje jmenný prostor. Ten sdružuje související typy tříd, struktury, rozhraní, delegáty.

```
namespace NazevJmennehoProstoru
```

```
{  
    class NazevTridy  
    {
```

Main je vstupní bod programu, jedná se o funkci, která se provádí jako první. Jedním z hlavních úkolů překladačů je tuto funkci najít.

```
static void Main(string[] args)
```

```
{  
    Zápis programu začíná v těchto složených závorkách.  
}  
}
```

TIP – Ve jmenném prostoru `System` existuje třída `Console`. Úplný název třídy je `System.Console`. Aby programátor nemusel neustále psát kód `System.Console`, napíše na začátek programu `using System;` zakončené středníkem a všechny třídy ze jmenného prostoru `System` může používat se zkráceným názvem.



4.2 Otázky z probraného tématu

- ❖ Popiš jednotlivé panely vývojového prostředí Microsoft Visual C# 2010 Express.
- ❖ Co je to intellisance?
- ❖ Jak je označen vstupní bod v programu a kdo jej používá?
- ❖ K čemu je vhodná direktiva `using`?

4.3 Praktické cvičené na dané téma

- (13) Zapni vývojové prostředí a založ nový projekt pro konzoli a druhý projekt pro formulářové okno. Prozkoumej jaký v nich je rozdíl.
- (14) Vytvořený projekt zkus přeložit pomocí debuggeru.

(15) Ulož projekt z bodu (14) na disk.

4.4 Seznam použité literatury

- [1] ELLER, Frank. *C# - začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2002, 240 s. ISBN 80-247-0324-6.
- [2] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2008, 592 s. ISBN 978-80-251-2027-9.
- [3] VYSTAVĚL, Radek. *Moderní programování: pro začátečníky*. Ondřejov: moderníProgramování, 2007, 2 sv. ISBN 978-80-903951-1-4.

5 METODICKÝ LIST – ČÍSLO 5



CÍLEM TÉMATU – je seznámení se zápisem kódu programovacího jazyka C# a vyřešení prvního programu ve vývojovém prostředí.



KLÍČOVÁ SLOVA – WriteLine, ReadKey, solution, debugger, středník, komentář

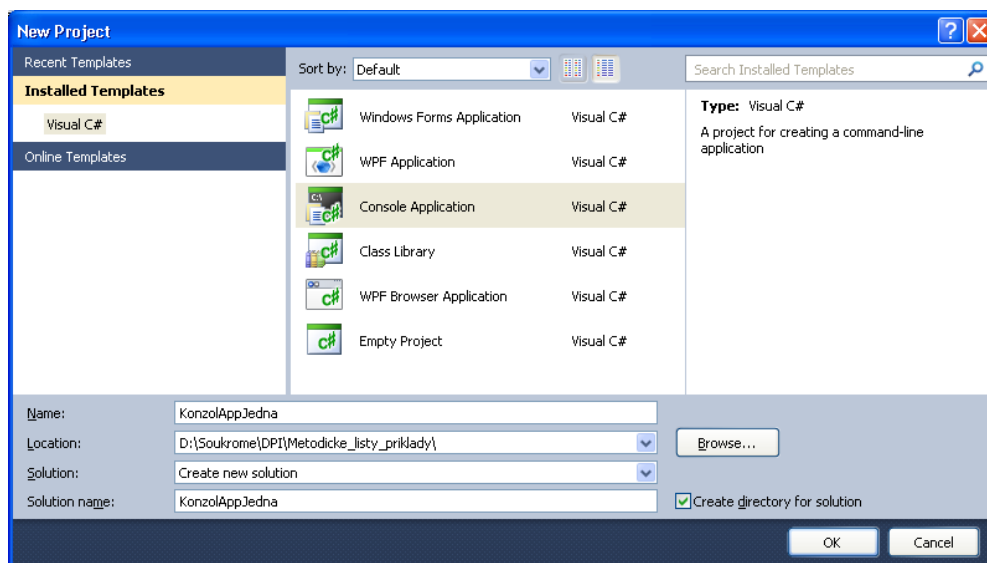


ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 90 minut.

Každé začátky jsou problematické, a proto není vhodné studenta zavalit od první chvíle hromadou teorie s novými pojmy. V těchto pojmech se neorientuje, až ztrácí. Je vhodnější si ukázat praktické věci ihned na počátku a na ně navazovat komplikovanější látku.

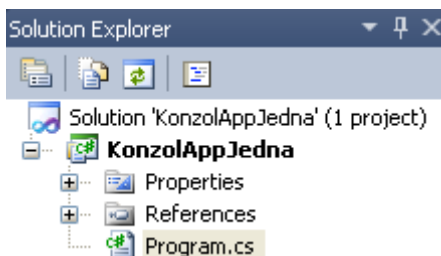
5.1 První program ve vývojovém prostředí

Pomocí menu *File* založíme nový projekt *Console Application*, který uložíme do předem připraveného adresáře na disku počítače pomocí tlačítka *Browse*. Do tohoto adresáře se budou ukládat veškeré vytvořené programy. Jako jméno aplikace zvolíme KonzolAppJedna. Větší část cvičných programů bude věnována právě konzolovým aplikacím, jelikož k jejich obsluze je zapotřebí minimální znalost objektového programování a tudíž jsou ideální k naučení, pochopení a získání základních dovedností v oblasti jazyka C#. Všimněte si, že je zatrženo pole *Create directory for solution* což znamená, že vytvářený projekt se založí do nové adresářové složky. Výsledné okno bude vypadat takto:



Obr. 14. Založení nového projektu

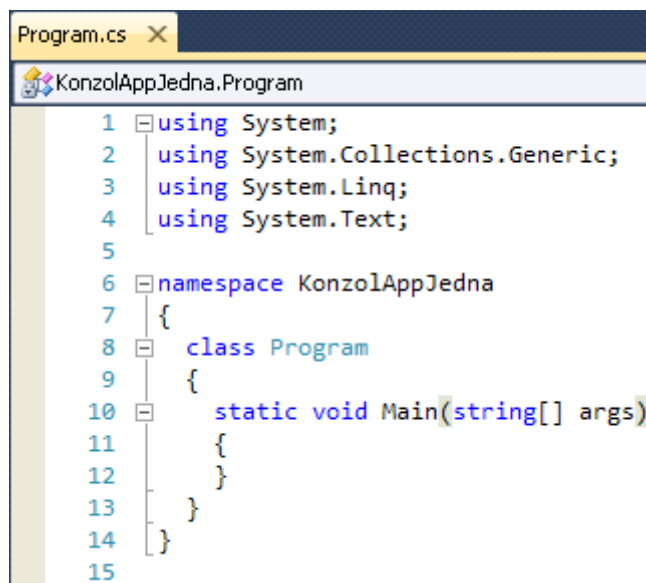
Programy, které budou vytvořeny se nazývají projekty. Každý projekt je složen s jednoho nebo více souborů, ve kterých je obsažen zdrojový kód programu a výstupní spustitelný soubor s koncovkou *.exe. Řešení anglicky solution je skupina jednoho nebo více projektů, které řeší daný úkol. Program napsán ve zdrojovém kódu C# se ukládá s příponou *.cs.



Obr. 15. Řešení prvního projektu

Prvních několik řádků vygenerované kostry kódu patří seznamu direktiv označené klíčovým slovem `using`. Díky nim nemusí programátor opakovaně vypisovat názvy využívané třídy. Například bez direktiv by programátor pro každý výpis textu na displej musel používat příkaz `System.Console.WriteLine("");`. Při použití direktivy `using System;` programátor používá zkrácený příkaz `Console.WriteLine("");`

Zbytek programu je rozdělen do bloků, které se do sebe vnořují složeným závorkami `{}`. Prvním blokem je jmenný prostor, který sdružuje jednotlivé třídy a další typy. Třídy obsahují bloky funkcí vlastně procedur v C# a vstupní bod programu. Jedná se o hlavní funkci označenou slovem `Main`. V kulatých závorkách za názvem hlavní funkce jsou uvedeny vstupní parametry. Funkce tyto parametry vyžaduje pro provedení svého kódu.



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 namespace KonzolAppJedna
7 {
8     class Program
9     {
10         static void Main(string[] args)
11         {
12         }
13     }
14 }
15
```

Obr. 16. Vygenerovaná kostra zdrojového kódu C#

Příkaz `using` zařazuje daný obor názvů do aktuálního oboru platnosti, takže v následném kódu v tomto souboru již nemusíte explicitně uvádět v názvech objektů také příslušný obor názvů. Čtyři výše uvedené obory názvů obsahují třídy, které jsou používány tak často, že je Microsoft Visual C# 2010 Express automaticky přidává do každého nového projektu.



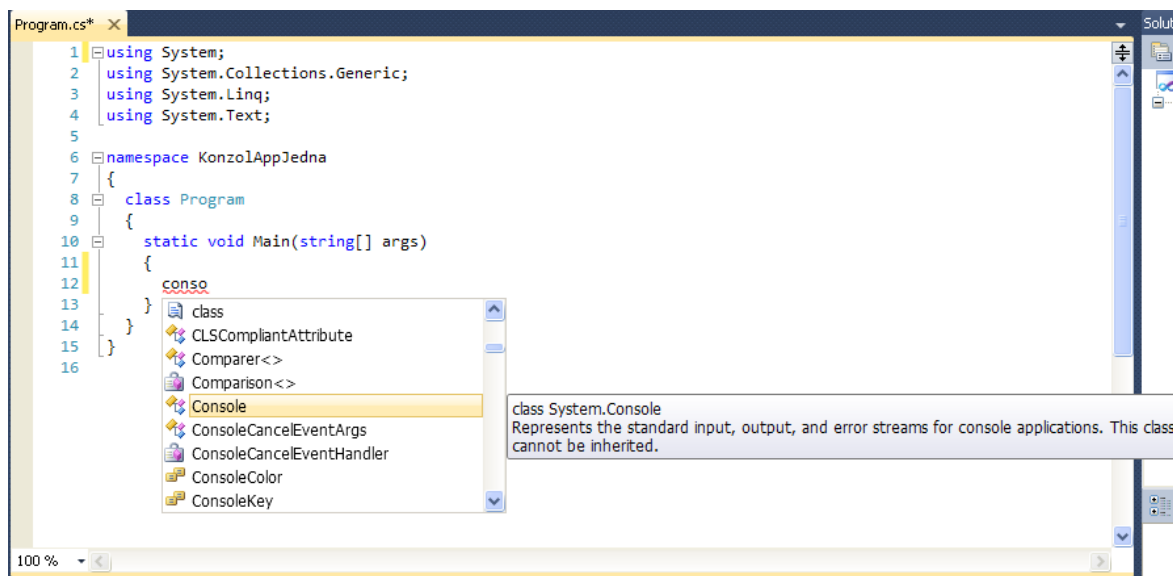
TIP – C# patří do skupiny programů označovaný termínem **case-sensitive**.

Znamená to totiž, že tento programovací jazyk rozlišuje velká a malá písmena. Dávej si pozor na zápis proměnných, příkazů, metod atd. Identifikátory **maleCislo** a **malecislo** označují zcela jiné proměnné.



POZNÁMKA – Vstupní bod programu, tedy metoda **Main** musí být v programu použita pouze jednou. Pokud i přesto je použita hlavní metoda v programu vícekrát, musí se kompilátoru sdělit, která metoda bude používána jako hlavní.

Od teorie nyní upustíme a vrátíme se do hlavní třídy, kde vložíme příkaz `Console.WriteLine("Můj první program");` zakončený středníkem. Jistě si všimáte, že po prvních několika úhozech se Vám otevře seznam nápovědy *IntelliSense*. Tento seznam obsahuje všechna klíčová slova a datové typy jazyka C#, jež jsou v daném kontextu platné. V tomto okamžiku můžete buď pokračovat v psaní, nebo posouvat obsah seznamu směrem dolů a myší poklepat na danou položku. My vybereme třídu `Console`. Nyní si postačíme s tím, že třída pro nás bude chápána jako soubor nějakých metod. Pro vyvolání seznamu metod nám poslouží operátor tečky za klíčovým slovem `Console`.



Obr. 17. Zápis příkazu Console.WriteLine().



ZDROJOVÝ KÓD – **console** je vestavěná třída, která obsahuje metody pro zobrazování zpráv na obrazovce a pro načítání vstupních dat z klávesnice.

`Console.WriteLine()`

▲ 1 of 19 ▼ void Console.WriteLine()
Writes the current line terminator to the standard output stream.

Obr. 18. Ukázka intellisense u metody WriteLine()



ZDROJOVÝ KÓD – **WriteLine()** obsahuje parametry. Tato metoda je ve skutečnosti přetížená metoda, což znamená, že třída **Console** obsahuje více než jednu metodu s názvem **WriteLine()**. Nabízí celkem 19 verzí této metody. Každá verze metody **WriteLine()** slouží k výpisu jiného typu dat.

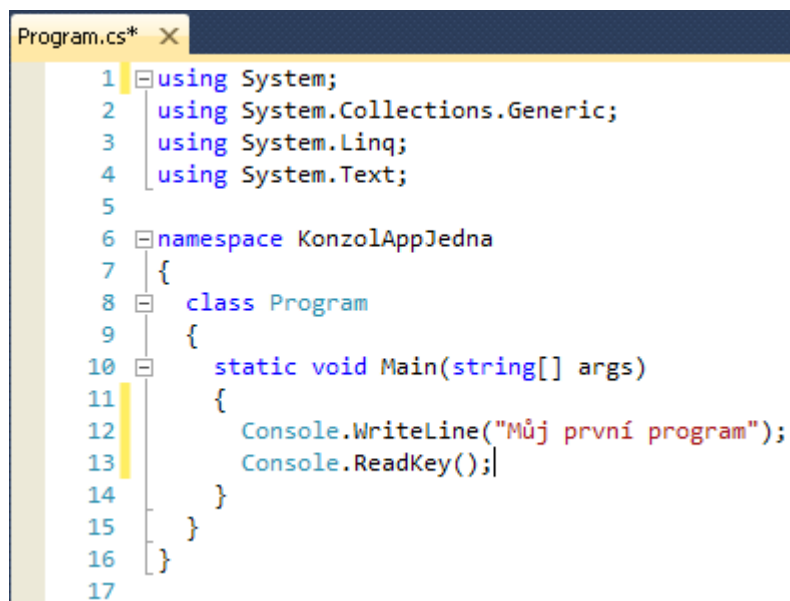


UPOZORNĚNÍ – **Středník** za metodou např. **WriteLine()** je velmi důležitý při psaní zdrojového kódu v C#. Středník totiž označuje konec předchozího příkazu, tedy odděluje jednotlivé příkazy/metody. Každá metoda může obsahovat nějaké vstupní parametry. Tyto parametry se zadávají do **kulatých závorek** a jsou oddělené čárkou. Toto je v zdrojovém kódu povinností.




TIP – Mezi jednotlivými přetíženými verzemi metody **WriteLine()** můžete procházet pomocí směrových šipek nahoru a dolů. Zvykněte si vždy nejdříve zapsat sobě odpovídající páry znaků, například „(“ a „)“ nebo „{“ a „}“; a teprve poté doplnit to, co by mělo být mezi nimi. Budete-li postupovat opačně, snadno na uzavírací znak zapomenete.

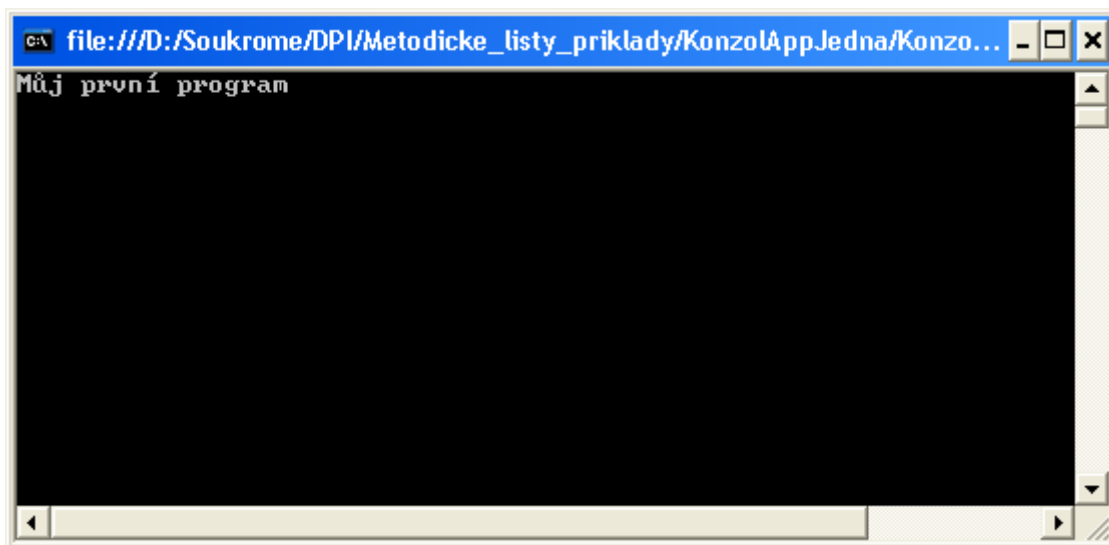
Parametrem metody `WriteLine()` je text k vypsání. Text je označován v programování jako textový řetězec anglicky string. Tento textový řetězec se bude uvádět do uvozovek, aby C# rozuměl tomuto zápisu a nezaměňoval jej s jinými příkazy. Dalším příkazem tentokrát bez parametru bude `Console.ReadKey()`; proto, aby se nám program po vypsání hlášky ihned neukončil, ale čekal na úhoz libovolné klávesy. Od této chvíle se stáváte programátory. Výsledný zdrojový kód by měl vypadat následovně:



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 namespace KonzolAppJedna
7 {
8     class Program
9     {
10         static void Main(string[] args)
11         {
12             Console.WriteLine("Můj první program");
13             Console.ReadKey();
14         }
15     }
16 }
17
```

Obr. 19. Zdrojový kód k prvnímu programu

Vaše práce s tímto programem neskončila, je nutné jej ještě překompilovat a zkontrolovat jestli neobsahuje chyby a hlavně spustit a ověřit jestli opravdu funguje. To provedete buďto pomocí debbugeru  v nástrojové liště nebo stiskem klávesy F5. Třetí možností je přes menu *Debug -> Start Debugging F5*. Výsledkem bude konzolová aplikace, která zobrazí text 'Můj první program'.











Obr. 20. Konzolová aplikace s prvním programem

5.1.1 Ikony nápovědy IntelliSense

Když po názvu třídy napíšete tečku, zobrazí nápověda *IntelliSense* seznam názvů všech členů dané třídy. Vlevo od těchto názvů je ikona, která určuje typ příslušného člena. Následující tabulka uvádí přehled obvyklých ikon a jim odpovídajících typů:

Tab. 8. Tabulka ikon intellisense

Ikona	Význam
	Metoda (method)
	Vlastnost (property)
	Třída (class)
	Struktura (struct)
	Výčet (enum)
	Rozhraní (interface)
	Delegát (delegate)
	Rozšiřující metoda (extension method)

5.1.2 Komentáře zdrojového kódu

Velmi často narazíte na řádky, které obsahují dvě lomítka, za kterými je „normální“ text. Jedná se o jednořádkový komentář zdrojového kódu. Překladač kódu tyto komentáře ignoruje a přeskočí je, pro vývojáře jsou však velmi užitečné a důležité, protože pomáhají objasnit, co vlastně program na daném místě kódu vykonává. Příkladem komentáře je:



```
// Toto je jednořádkový komentář
```

Existuje i víceřádkový komentář, který začíná lomítkem s hvězdičkou „/*“. Překladač přeskočí vše za těmito znaky, dokud nenarazí na ukončovací komentář s hvězdičkou a lomítkem „*/“. Příkladem komentáře je:



```
/*  
 * Toto je víceřádkový komentář  
 */
```

5.1.3 Jednotlivé bloky ve zdrojovém kódu

Ve vzorovém kódu jste si museli povšimnout rozdělení kódu do jednotlivých bloků za pomoci složených závorek {}. Na příkazy uvnitř jedné složené závorky můžeme pohlížet jako na jeden příkaz. Složené závorky tedy v C# označují programové bloky. Pomoci nich jsou odděleny bloky kódu, jako jmenné prostory, třídy, cykly, podmínky či metody.



```
namespace ConsoleApp  
{ // Třídy, struktury, rozhraní, delegáty  
  class Program  
  {  
    // Funkce nebo procedury  
    static void Main(string[] args)  
    { // Inicializace proměnných  
      {  
        // Příkazy  
      }  
    }  
  }  
}
```

5.1.4 Únikové sekvence

Prvním výstupem, který byl proveden bylo vypsání jednoduchého textového řetězce „Ahoj planeto“. Existuje mnoho dalších možností pro výstup i pro jeho formátování. Nastává, ale problém pokud uživatel potřebuje vypsát textový řetězec který obsahuje apostrofy, zpětné lomítka apod. Programovací jazyk neví, kdy uživatel chce vypsát znak a kdy ne. Z tohoto důvodu se využívají zvláštní znaky tzv. únikové sekvence (escape sequence).

Tab. 9. Espace sequence

Znak	Význam	Unicode
\"	Uvozovky	\u0022
\'	Apostrof	\u002C
\\	Zpětné lomítko	\u005C
\0	Prázdná hodnota znaku (null)	\u0000
\a	Zvukový signál	\u0007
\b	Klávesa zpět	\u0008
\f	Posuv formuláře (nová stránka)	\u000C
\n	Znak nového řádku	\u000A
\r	Návrat na začátek řádku	\u000D
\t	Tabulátor	\u0009
\v	Svislý tabulátor	\u000B

Příklad výpisu textu s uvozovkami:



```
Console.WriteLine("Zadaná věta s uvozovkami: \"Můj první  
program\"");
```

5.2 Otázky z probraného tématu

- ❖ K čemu se ve zdrojovém kódu využívá u programovacího jazyku C# středník?
- ❖ Jaké druhy komentáře znáš? Jak se zapisují?
- ❖ K čemu se používají ve zdrojovém kódu kulaté a k čemu složené závorky?
- ❖ K čemu se používá intellisense?
- ❖ Jaký příkaz se používá k výpisu textu na displej a jaký příkaz se používá na úhoz klávesy od uživatele?

5.3 Praktické cvičené na dané téma

- (16) Vytvořený první úkol vhodně doplň o komentář.
- (17) Smaž středník za příkazem WriteLine(). Dávej pozor, co ti vývojové prostředí vypíše za chybu do Error listu. Pomocí dvojkliku si ověř, jestli se na pozici chyby dostaneš. Poté chybu oprav.

- (18) Vytvoř konzolovou aplikaci, která bude mít za úkol výpis textu „Vítám tě v binárním světě.“ včetně uvozovek. Program po vykonání příkazu nebude ukončen, ale bude čekat na úhoz libovolné klávesy. Program vhodně okomentuj jednořádkovým komentářem.

5.4 Řešení praktického cvičení

- (18) *Úsek programu pro výpis textu „Vítám tě v binárním světě.“ včetně uvozovek. Program po vykonání příkazu nebude ukončen, ale bude čekat na úhoz libovolné klávesy.*



```
// Příkaz pro výpis zadaného textu
Console.WriteLine("\"Vítám tě v binárním světě.\"");
Console.ReadKey();
```

5.5 Seznam použité literatury

- [1] ELLER, Frank. *C# - začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2002, 240 s. ISBN 80-247-0324-6.
- [2] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2008, 592 s. ISBN 978-80-251-2027-9.

6 METODICKÝ LIST – ČÍSLO 6



CÍLEM TÉMATU – je objasnění pojmů proměnná, práce s proměnnými a datovými typy na ně navázány, operátory a jejich operandy, parsování datových typů



KLÍČOVÁ SLOVA – proměnná, operátor, operand, parsování, camelCase station, datové typy



ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 90 minut.

6.1 Práce s proměnnými, datovými typy, operátory a parsováním

6.1.1 Co je to proměnná?

Proměnná je určité místo přechovávající nějakou hodnotu. Můžete si ji představit jako schránku v paměti počítače uchovávající dočasnou informaci. Každé proměnné v programu musíte přidělit jednoznačný název, který jedinečným způsobem identifikuje tuto proměnnou v kontextu, v němž se používá. Prostřednictvím názvu proměnné se potom odkazujete na hodnotu v ní uloženou. Jednoduše řečeno si lze proměnnou představit stejně jako proměnnou v matematice, do které se nejčastěji ukládá libovolné číslo. V informatice je to naprosto stejné, jen se do proměnné ukládají data. Daty chápeme nějaké **datové typy**, může to být číslo, znak, text a podobně, záleží na tom, k čemu je chceme používat.

Doporučená pravidla při vytváření podmínek:


- Název by měl začínat malým písmenem.
- U identifikátorů složených z více slov začínějte druhé a každé další slovo velkým písmenem. Této zásadě se říká "velbloudí" zápis (*camelCase notation*).
- Nevytvářejte identifikátory, které se liší jen velikostí písmen. Vyhněte se například proměnným s názvy `MojePromenna` a `mojePromenna` v jednom místě zdrojového kódu, protože se dají velmi snadno zaměnit.
- Nepoužívejte v identifikátorech podtržítka.

Poslední dvě doporučení berte jako povinná, protože vycházejí ze specifikaci CLS (Common Language Specification). Deklarace proměnné se provádí tak, že definujete *název proměnné* a *typ dat*, které do ní lze uložit. Deklarace proměnné musí být zakončena

středníkem. Každá proměnná musí být před použitím inicializována tedy je potřeba, aby měla přiřazenou hodnotu. Obecně lze znázornit deklaraci a inicializaci proměnné takto:


datovýTyp názevProměnné = hodnotaproměnné;

Deklarace a inicializace proměnné se zapisuje v kódu:



```
//neinicializovaná proměnná
int celocislenaPromenna;
//inicializace tj. první přiřazení hodnoty do proměnné
celocislenaPromenna = 32;
```

Je možné použít i zkrácený zápis, kdy do proměnné lze přiřazovat již při její deklaraci:




```
int celocislenaPromenna = 32;
```

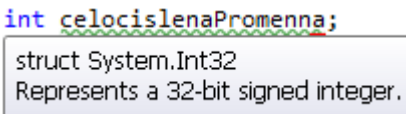


POZNÁMKA – Programovací jazyk C# nepovoluje implicitní deklarace proměnných. Každou proměnnou je nutné před jejím prvním použitím v kódu řádně deklarovat.

Příkladem může proměnná *celocislenaPromenna*, která obsahuje *celočíslné hodnoty*:



```
int celocislenaPromenna;
```



Obr. 21. Příklad s ScreenTipem



POZNÁMKA – Jestliže v okně editoru kódu Microsoft Visual C# 2010 Express najedete ukazatelem myši nad název proměnné, objeví se okno ScreenTip s informací o typu proměnné.

Cvičný program „KonzolAppVypisPromenne“: Založte si konzolovou aplikaci, která bude mít za úkol výpis dosazené hodnoty 32 v proměnné *celocislenaPromenna* do konzole.

- 1 Krok jedna založení proměnné.

```
int celocislenaPromenna;
```

- 2 Krok dva dosazení hodnoty 32 do proměnné, se provádí pomocí přiřazovacího znaku = „rovná se“. Všimněte si, že jakmile se uvedly první dva znaky proměnné, intellisense nám nabídne tuto proměnnou se ScreenTipem, jelikož tento název je jedinečný v tomto programu.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace KonzolAppVypisPromenne
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12             int celocislenaPromenna;
13             ce
14         }
15     }
16 }
17

```

celocislenaPromenna (local variable) int celocislenaPromenna

Obr. 22. Ukázka proměnné nabídnuté intellisense

```
celocislenaPromenna = 32;
```

- 3 Třetím krokem je použití příkazu pro výpis do konzole. Do tohoto příkazu použijeme místo textu název proměnné.

```
Console.WriteLine(celocislenaPromenna);
```

- 4 V kroku čtyři je použit příkaz čekající na stisk klávesy.

```
Console.ReadKey();
```

- 5 Posledním krokem je výsledný kód kompilován a spuštěn.

```

namespace KonzolAppVypisPromenne
{
    class Program
    {
        static void Main(string[] args)
        {
            int celocislenaPromenna;
            celocislenaPromenna = 32;
            Console.WriteLine(celocislenaPromenna);
            Console.ReadKey();
        }
    }
}

```

- 6 Program po spuštění vypíše do konzole číslo **32**.

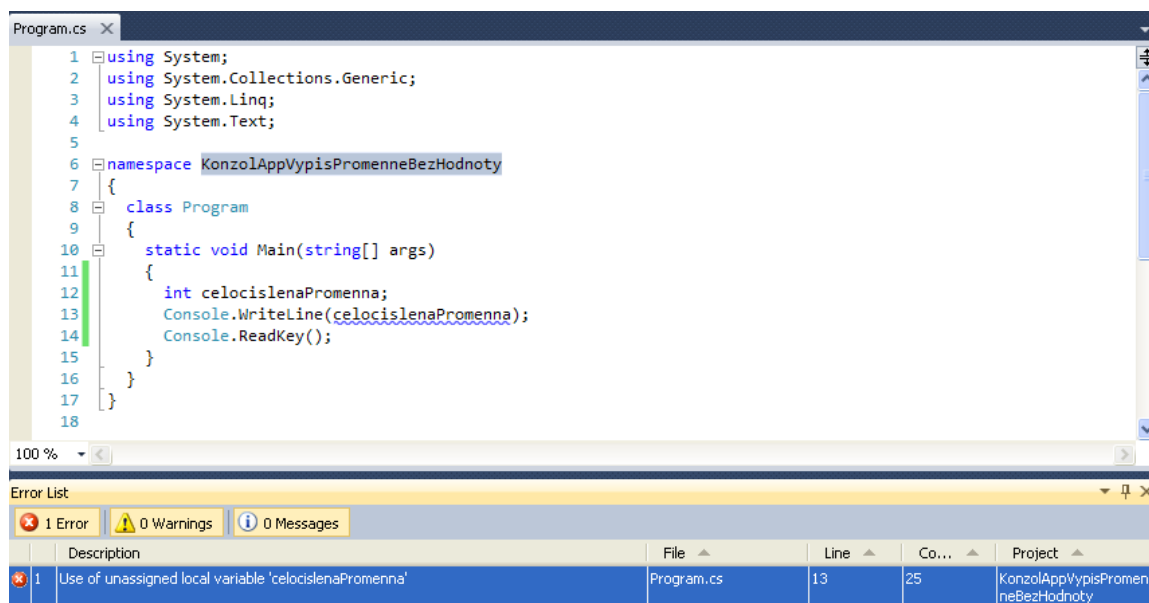
6.1.2 Typový systém

Používají se dva základní typové systémy *statický* a *dynamický*. Dynamické datové typy programátory oprostí od zadávání datového typu k proměnné. Tuto povinnost nevyžadují. Dynamické datové typy jsou mnohdy tak „chytré“, že když proměnná nebyla nikdy deklarována tak při jejím prvním použití ve zdrojovém kódu jej programovací jazyk sám založí. Přitom do takové to proměnné může programátor ukládat textový řetězec,

desetinné číslo a poté objekt. Proměnná se automaticky může měnit. Oproti tomu statické datové typy striktně vyžadují definovat datový typ a tento typ je dále neměnný. Jakmile se proměnná jednou deklarovala, není možné její datový typ měnit.

Programovací jazyk C# je statickým jazykem. Nepovoluje používat proměnnou, které nebyla přiřazena hodnota. Před použitím proměnné je nutné přiřadit jí hodnotu, jinak program nepůjde zkompileovat. V dokumentaci toto pravidlo najdete pod názvem *Definite Assignment Rule* (pravidlo jednoznačného přiřazení). Nevýhodou je, že díky deklaracím je zdrojový kód objemnější a vývoj pomalejší. Obrovskou výhodou však je to, že kompilér před spuštěním programu zkontroluje, zda všechny datové typy sedí.

Cvičný program „**KonzolAppVypisPromenneBezHodnoty**“: Uprav předešlý program tak, že odstraníš z něj přiřazení hodnoty 32. Všimni si hlášení chyby, že proměnná nemá přiřazenou hodnotu.



Obr. 23. Zvýraznění proměnné bez přiřazené hodnoty ve vývojovém prostředí

6.1.3 Primitivní datové typy

Programovací jazyk C# obsahuje několik vestavěných datových typů. Tyto typy se nazývají *primitivní datové typy*. Následující tabulka uvádí seznam nejpoužívanějších primitivních datových typů. Daly by se rozdělit na dva typy celočíselné a reálné typy.

Celočíselní datové typy – jak již název napovídá jsou takové datové typy, které používají celá čísla

Tab. 10. Tabulka celočíselných primitivních datových typů

Název	Typ	Velikost v bitech	Rozsah	Příklad použití
sbyte	System.SByte	Znaménkové 8 bitové číslo	-128 až 127	<code>sbyte sb = -15;</code> <code>Console.WriteLine(sb);</code>
short	System.Int16	Znaménkové 16 bitové číslo	-32 768 až 32 767	<code>short sh = -1000;</code> <code>Console.WriteLine(sh);</code>
int	System.Int32	Znaménkové 32 bitové číslo	-2 147 483 648 až 2 147 483 647	<code>int i = -200000;</code> <code>Console.WriteLine(i);</code>
long	System.Int64	Znaménkové 64 bitové číslo	-9,223,372,036,854,775,808 až 9,223,372,036,854,775,807	<code>long l = -900000000000L;</code> <code>Console.WriteLine(l);</code>
byte	System.Byte	Neznaménkové 8 bitové číslo	0 až 255	<code>sbyte b = 255;</code> <code>Console.WriteLine(b);</code>
ushort	System.UInt16	Neznaménkové 16 bitové číslo	0 až 65,535	<code>ushort ush = 40000;</code> <code>Console.WriteLine(ush);</code>
uint	System.UInt32	Neznaménkové 32 bitové číslo	0 až 4,294,967,295	<code>int ui = 200000;</code> <code>Console.WriteLine(ui);</code>
ulong	System.UInt64	Neznaménkové 64 bitové číslo	0 až 18 446 744 073 709 551 615	<code>ulong ul = 180000000000;</code> <code>Console.WriteLine(ul);</code>
string	System.String	16 bitů na každý znak	Nelze vyjádřit	<code>string str = "Text";</code> <code>Console.WriteLine(str);</code>
bool	System.Boolean	Logická hodnota 8 bitová	true nebo false	<code>bool bo = true;</code> <code>Console.WriteLine(bo);</code>
char	System.Char	Unicode 16-bitový znak	0 až $2^{16}-1$	<code>char ch = 'a';</code> <code>Console.WriteLine(ch);</code>


Reálné datový typy a typ decimal – datové typy *float* a *double* se používají pro počítání s desetinou čárkou. Typ *decimal* také slouží pro počítání s desetinou čárkou, ale jeho použití je hlavně při počítání s peněžními hodnotami.

Tab. 11. Tabulka reálných datových typů a typu decimal

Název	Typ	Přesnost	Přibližný rozsah	Příklad použití
float	System.Single	7 desetinných míst	$\pm 1.5 \times 10^{-45}$ až $\pm 3.4 \times 10^{38}$	<code>float f = 11.12F;</code> <code>Console.WriteLine(f);</code>
double	System.Double	15-16 desetinných míst	$\pm 5.0 \times 10^{-324}$ až $\pm 1.8 \times 10^{308}$	<code>double dbl = 11.1234;</code> <code>Console.WriteLine(dbl);</code>
decimal	System.Decimal	28-29 podstatných desetinných míst	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$	<code>decimal dec = 11.1234567M;</code> <code>Console.WriteLine(dec);</code>

6.1.4 Aritmetické operátory

Jazyk C# podporuje běžné aritmetické operátory, které jste se naučili již na základní škole: znak plus „+“ pro sčítání, minus „-“ pro odčítání, hvězdičku „*“ pro násobení a lomítko „/“ pro dělení. Tyto symboly jsou operátory, protože "operují" s hodnotami, na jejichž základě vytvářejí hodnoty nové.



```
namespace KonAppOperandy
{
    class Program
    {
        static void Main(string[] args)
        {
            short sh;
            sh = 10 * 20;
            Console.WriteLine(sh);
            Console.ReadKey();
        }
    }
}
```



UPOZORNĚNÍ – Hodnotám, se kterými operátor pracuje, se říká operandy. Ve výrazu `sh = 10 * 20;` je `*` operátor, čísla 10 a 20 jsou operandy a `sh` samozřejmě proměnná.

- ✓ Operátor „+“ provede součet číselných hodnot operandů vlevo a vpravo a vrátí na dané místo vypočtenou hodnotu
- ✓ Operátor „-“ provede rozdíl číselných hodnot operandů vlevo a vpravo a vrátí na dané místo vypočtenou hodnotu
- ✓ Operátor „*“ provede součin číselných hodnot operandů vlevo a vpravo a vrátí na dané místo vypočtenou hodnotu
- ✓ Operátor „/“ provede podíl číselných hodnot operandů vlevo a vpravo a vrátí na dané místo vypočtenou hodnotu
- ✓ Operátor „%“ (modulo) provede zbytek po dělení číselných hodnot operandů vlevo a vpravo a vrátí na dané místo vypočtenou hodnotu

Ne všechny operátory lze aplikovat na všechny datové typy, takže možnost použití operátoru na danou hodnotu je obecně závislá na typu této hodnoty. Na hodnoty typu `char`, `int`, `long`, `float`, `double` nebo `decimal` lze použít všechny aritmetické operátory. *Výjimka při použití operátorů* - operátor „+“ u datového typu `string` a `bool` nelze aplikovat. Jelikož nám datový typ nebude sčítat. U datový typ `string` operátor „+“ slouží pro spojení dvou a více textových řetězců.



```
Console.WriteLine("Spojení"+"Textového"+"Řetězce");
```



UPOZORNĚNÍ – Při psaní kódu je důležité dávat pozor na to, že typ výsledné hodnoty aritmetické operace je závislý na použitých operandech. Například výsledkem výrazu $7.0 / 2.0$ je hodnota 3.5, protože typ obou operandů je `double`. V jazyku C# jsou čísla s desetinnými místy vždy typu `double`, nikoli `float`! Je to kvůli zachování maximální možné přesnosti výpočtů, proto i výsledná hodnota je typu `double`. Ale pozor výsledkem výrazu $7 / 2$ je číslo 3. V tomto případě jsou totiž oba operandy typu `int`, proto i výsledek je typu `int`. Jazyk C# vždy v těchto situacích zaokrouhluje směrem dolů. Pokud se typy operandů navzájem liší například výraz $7 / 2.0$ se skládá z operandů typu `int` a `double`. Kompilátor jazyka C# tento nesoulad rozpozná a vygeneruje kód, který ještě před provedením dané operace převede hodnotu `int` na `double`. Výsledkem je hodnota 2.5 typu `double`. **Použití dvou odlišných typů ve výrazu se, ale důsledně nedoporučuje!**



```
namespace KonAppVyjimkyOperatoru
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Spojení"+"Textového"+"Řetězce");
            Console.WriteLine(7.0 / 2.0);
            Console.WriteLine(7 / 2);
            Console.WriteLine(7 / 2.0);
            Console.ReadKey();
        }
    }
}
```

6.1.5 Priorita vyhodnocování operátorů


V jazyce C# mají multiplikativní operátory „*“, „/“, „%“ přednost před aditivními operátory „+“ a „-“. Ve výrazu $1 + 2 * 3$ je nejdříve vyhodnoceno násobení a až poté sčítání. Výsledkem výrazu je tedy hodnota 7. Změna pořadí vyhodnocení operátoru se provádí za pomoci složených závorek. Tudíž předešlý výraz obohacený o závorky například $(1 + 2) * 3$ by měl výslednou hodnotu 9. Program by nejdříve musel vyhodnotit závorky a až poté by následoval operátor násobení.



```
Console.WriteLine(1 + 2 * 3);
Console.WriteLine((1 + 2) * 3);
Console.ReadKey();
```


6.1.6 Parsování

Parsováním je myšlen převod textové podoby na nějakou specifickou hodnotu. Tedy pokud je zapotřebí sečíst čísla uložená v textových řetězcích nebo s nimi provádět jiné aritmetické operace. Programovací jazyk C# používá k parsování metodu `Parse()`, která bere jako parametr textový řetězec a vrací číslo. Metodu `Parse()` používá většina datových typů C#. Příkladem je program, který má v proměnné s datovým typem `string` uloženou hodnotu 25. Aby mohl s touto hodnotou program provést aritmetickou operaci násobení, musí se nejdříve textový řetězec převést na číselnou hodnotu a až poté násobit. Bez použití metody `Parse()`, by kompilátor hlásil chybu, že nejde násobit datový typ `string` a `int`.



```
namespace KonAppParsovani
{
    class Program
    {
        static void Main(string[] args)
        {
            string str = "25";
            // Převod textové podoby čísla na číselný datový typ short
            short s = short.Parse(str);
            Console.WriteLine(s*2);
            Console.ReadKey();
        }
    }
}
```

Formátování je velmi užitečné, umožňuje nám vkládat do samotného textového řetězce zástupné značky. Ty jsou reprezentovány jako číslo ve složených závorkách, prvním číslem je vždy 0. Pro každý datový typ (kromě `bool`) můžeme určit minimální a maximální hodnotu, kterou může tento datový typ obsahovat. Na následném příkladu je použito jak formátování, tak vlastnost pro určení minimální a maximální hodnoty datového typu `int`.



```
Console.WriteLine("Int - min: {0}, max: {1}", int.MinValue, int.MaxValue);
```

6.2 Otázky z probraného tématu

- ❖ Co je to proměnná? Jak proměnnou zapisuješ?
- ❖ Jaké 2 druhy typového systému znáš?
- ❖ Co víš o statickém a co o dynamickém typovém systému?

- ❖ Jaký typový systém používá programovací jazyk C#?
- ❖ Jaký je rozdíl mezi celočíselnými a reálnými datovými typy?
- ❖ Jaký je rozdíl mezi operátorem a operandem?
- ❖ Co víš o prioritě vyhodnocování operátorů? Jak je možné prioritu změnit?
- ❖ Jakou metodu použije v programovacím jazyku C# pro převod textového čísla na číselný datový typ? Co o parsování víš?

6.3 Praktické cvičené na dané téma

- (19) Program ozvěna. Vytvoř konzolovou aplikaci, která bude mít za úkol výpis textu vloženého od uživatele. Pokud uživatel zadá jakýkoliv text, program jej třikrát po sobě zopakuje. Program po zopakování zadaného textu nebude ukončen, ale bude čekat na úhoz libovolné klávesy. Program vhodně okomentuj jednořádkovým komentářem.
Pomůcka: Pro načítání z konzole slouží metoda `ReadLine()`, která vrací textový řetězec z konzole.
- (20) Program primitivní kalkulačka s operátorem sečítání. Vytvoř konzolovou aplikaci, která bude mít za úkol součet dvou libovolných zadaných uživatelem do konzole. Zadané proměnné budou typu `double` a výsledná hodnota bude také typu `double`. Pro převod textu na číslo použij metodu `Parse()`. Pro ošetření desetinné čárky použij metodu `Replace()`.
- (21) Program pro výpis minimální a maximálních hodnot datových typů krom datového typu `bool`. V datovém typu `bool` použij vlastnost `TrueString` a `FalseString`. Výsledný program bude vytvořen jako konzolová aplikace. Použij formátování pomocí zástupných značek, jako tomu bylo na vzorovém příkladu.

6.4 Řešení praktického cvičení

(19) Program ozvěna.



```

Console.WriteLine("Vítejte v programu ozvěna.");
// metoda Write() vypíše text do konzole, ale nevytvoří nový řádek
Console.Write("Napiš text na zopakování: ");
// deklarace proměnné zadanyText pro vložení textového řetězce
string zadanyText;
// Vložení textového řetězce do proměnné
zadanyText = Console.ReadLine();
// deklarace proměnné opakovanyText pro výstup textového řetězce
string opakovanyText;
// zřetězení několika textových řetězců pomocí operátoru +
opakovanyText = zadanyText + ", " + zadanyText + ", " + zadanyText;
Console.WriteLine("Výstupní text zopakován třikrát: " + opakovanyText);
Console.ReadKey();

```

(20) Program primitivní kalkulačka s operátorem sečítání.



```

Console.WriteLine("Vítejte v programu primitivní kalkulačka s operátorem sčítání.");
Console.Write("Vlož první číslo: ");
// deklarace proměnné zadaneCislo1 a přiřazení zadané hodnoty do proměnné cislo 1
// Pomocí metody Replace('.', ',') se nahradí znak tečky (pokud je vložen) '.' znakem čárky ','
string zadaneCislo1 = Console.ReadLine().Replace('.', ',');

// převod textové řetězce na číslo pomocí metody Parse()
double prevodCisla1 = double.Parse(zadaneCislo1);

Console.Write("Vlož druhé číslo: ");
string zadaneCislo2 = Console.ReadLine().Replace('.', ',');
double prevodCisla2 = double.Parse(zadaneCislo2);

// deklarace proměnné soucet pro výstup textového řetězce
double soucetCisel = prevodCisla1 + prevodCisla2;
Console.WriteLine("Výsledkem součtu je: " + soucetCisel);
Console.ReadKey();

```

(21) Program pro výpis minimální a maximálních hodnot datových typů



```

Console.WriteLine("Vítejte v programu pro výpis hodnot datových typů.");
Console.WriteLine("sbyte - min: {0}, max: {1}", sbyte.MinValue, sbyte.MaxValue);
Console.WriteLine("byte - min: {0}, max: {1}", byte.MinValue, byte.MaxValue);
Console.WriteLine("char - min: {0}, max: {1}", char.MinValue, char.MaxValue);
Console.WriteLine("short - min: {0}, max: {1}", short.MinValue, short.MaxValue);
Console.WriteLine("ushort - min: {0}, max: {1}", ushort.MinValue, ushort.MaxValue);
Console.WriteLine("int - min: {0}, max: {1}", int.MinValue, int.MaxValue);
Console.WriteLine("long - min: {0}, max: {1}", long.MinValue, long.MaxValue);
Console.WriteLine("ulong - min: {0}, max: {1}", ulong.MinValue, ulong.MaxValue);
Console.WriteLine("float - min: {0}, max: {1}", float.MinValue, float.MaxValue);
Console.WriteLine("double - min: {0}, max: {1}", double.MinValue, double.MaxValue);
Console.WriteLine("decimal - min: {0}, max: {1}", decimal.MinValue, decimal.MaxValue);
Console.WriteLine("bool - {0}, {1}", bool.TrueString, bool.FalseString);
Console.ReadKey();

```

6.5 Seznam použité literatury

- [1] ELLER, Frank. *C# - začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2002, 240 s. ISBN 80-247-0324-6.
- [2] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2008, 592 s. ISBN 978-80-251-2027-9.
- [3] Programování se zaměřením na .NET a jazyk C#. *Aritmetické operátory* [online]. 2005 [cit. 2013-05-14]. Dostupné z: <http://projektysipvz.gytool.cz/ProjektySIPVZ/Default.aspx?uid=83>

7 METODICKÝ LIST – ČÍSLO 7



CÍLEM TÉMATU – je čtenáře seznámit s použitím rozhodovacích příkazů a cyklů v programovacím jazyku C#, práce s dalšími typy operátorů a jejich sdružování



KLÍČOVÁ SLOVA – příkaz if, switch, for, while, bool proměnná



ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 90 minut.

7.1 Rozhodovací příkazy if, switch a cykly for, while a do při jejich použití

7.1.1 Logické proměnné

Deklaraci logických proměnných – v programovacím světě je na rozdíl od toho reálného vše bílé nebo černé, správné nebo špatné, pravdivé nebo nepravdivé. K tomuto se využívají *logické výrazy*. Tento výraz nabývá hodnot buď *true* nebo *false*. Programovací jazyk C# k tomuto používá datový typ `bool`. Příklad deklarace, inicializace a vyhodnocení datového typu `bool`:



```
bool logPromenna = true;
Console.WriteLine(logPromenna);
```

7.1.2 Logické operátory

K nejčastějším využívaným operátorům patří operátor rovnosti a nerovnosti. Pomocí těchto binárních operátorů lze zjistit, zda se hodnoty téhož typu rovnají. Příkladem je následující tabulka, kde se vyhodnocuje inicializovaná proměnná `vek` s hodnotou 26.

```
int vek = 26;
```

Tab. 12. Použití logických operátorů

Operátor	Význam	Výraz	Kód	Výsledek
<code>==</code>	Rovnost	<code>vek == 50</code>	<code>bool rovnost = (vek == 30);</code> <code>Console.WriteLine(rovnost);</code>	false
<code>!=</code>	Nerovnost	<code>vek != 50</code>	<code>bool nerovnost = (vek != 15);</code> <code>Console.WriteLine(nerovnost);</code>	true
<code>></code>	Větší než	<code>vek > 50</code>	<code>bool vetsiNez = (vek > 50);</code> <code>Console.WriteLine(vetsiNez);</code>	false
<code><</code>	Menší než	<code>vek < 50</code>	<code>bool mensiNez = (vek < 50);</code> <code>Console.WriteLine(mensiNez);</code>	true

>=	Větší nebo rovno	vek >= 50	<code>bool vetsiNeboRovno = (vek >= 50); Console.WriteLine(vetsiNeboRovno);</code>	false
<=	Menší nebo rovno	vek <= 50	<code>bool mensiNeboRovno = (vek <= 50); Console.WriteLine(mensiNeboRovno);</code>	true

7.1.3 Užitečné logické operátory

Jazyk C# poskytuje dva další logické operátory. Operátor logického součinu/konjunkce (a zároveň) vyjádřený symbolem `&&`, a operátor logického součtu/disjunkce (nebo) vyjádřený symbolem `||`. Oba patří mezi podmíněné logické operátory. Jejich účelem je zkombinovat dva logické výrazy či hodnoty do jediného logického výsledku. Oba binární operátory se podobají operátorům rovnosti a relačním operátorům v tom, že hodnota výrazů, v nichž se objevují, je buď `true` nebo `false`. Liší se však v tom, že jejich operandy musí také samy o sobě nabývat hodnoty `true` nebo `false`.

Tab. 13. Užitečné logické operátory

Operátor	Význam
!	Operátor negace (z pravdy učiní nepravdu a naopak)
&&	Logická konjunkce (obě podmínky musí být splněny)
	Logická disjunkce (alespoň jedna z podmínek musí být splněna)

Příkladem může být vyhodnocení věku, kdy věk musí být větší než 18 a zároveň nesmí být menší než 30. Výsledkem výrazu bude hodnota `true`.



```
int vek = 26;
bool vyhodnoceni = (vek > 18) && (vek < 30);
Console.WriteLine(vyhodnoceni);
Console.ReadKey();
```

7.1.4 Operátory prefix a postfix

Operátory `++` (inkrement) a `--` (dekrement) jsou neobvyklé v tom, že je lze umístit před nebo za proměnnou. Je-li operátor umístěn před proměnnou, jedná se o prefixový zápis, pokud je za proměnnou, jde o postfixový zápis. Tyto operátory zvyšují inkrementují svůj obsah o jedničku nebo snižují dekrementují svůj obsah o jedničku. Umístění operátoru nemá vliv zda se hodnota zvyšuje nebo snižuje. Způsoby zápisu operátoru:



```
postInkrement++; // postfixový inkrement
++prefInkrement; // prefixový inkrement
postDekrement--; // postfixový dekrement
--prefDekrement; // prefixový dekrement
```

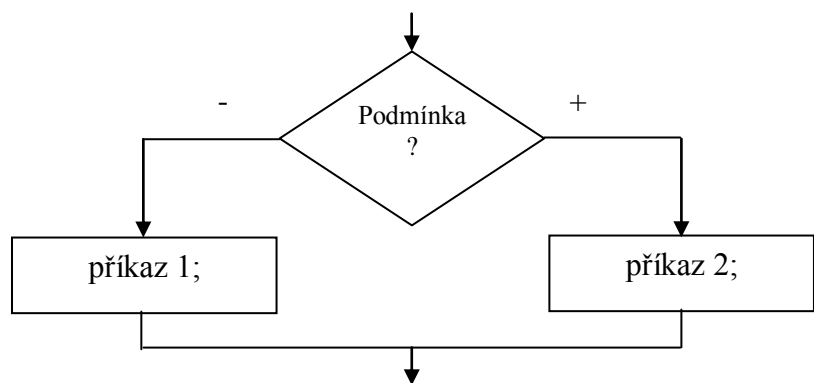

Rozdíl v umístění operátoru je v tom, kdy operátor tuto výslednou hodnotu vrátí. Napíšete-li operátor postfixově `postInkrement++`; pak je nejdříve vrácena výsledná hodnota operátoru, a teprve poté se daná proměnná inkrementuje. V opačném případě, tedy při prefixovém zápisu `++prefInkrement`, dojde nejprve k inkrementaci proměnné, a teprve poté je vrácena hodnota. Rozdíl si nejlépe ukáže následující příklad:

```
int postInkrement = 10; // postfixový inkrement
int prefInkrement = 10; // prefixový inkrement
int postDekrement = 10; // postfixový dekrement
int prefDekrement = 10; // prefixový dekrement
// promenna postInkrement má hodnotu nyní 11, program vypíše hodnotu 10
Console.WriteLine("Postfixový inkrement (počáteční hodnota 10) vrátí:
{0}", postInkrement++);
// promenna prefInkrement má hodnotu nyní 11, program vypíše hodnotu 11
Console.WriteLine("Prefixový inkrement (počáteční hodnota 10) vrátí:
{0}", ++prefInkrement);
// promenna postDekrement má hodnotu nyní 9, program vypíše hodnotu 10
Console.WriteLine("Postfixový dekrement (počáteční hodnota 10) vrátí:
{0}", postDekrement--);
// promenna prefDekrement má hodnotu nyní 9, program vypíše hodnotu 9
Console.WriteLine("Prefixový dekrement (počáteční hodnota 10) vrátí:
{0}", --prefDekrement);
Console.ReadKey();
```

7.1.5 Rozhodování pomocí příkazu if

Použití příkazu `if` se provádí v případech, kdy je zapotřebí provést jeden ze dvou různých bloků kódu, přičemž výběr bloku je dán nějakou logickou podmínkou. Syntaxe příkazu `if` v programovacím jazyku C# vypadá následovně:

```
If (logickyVyras)
příkaz 1;
else
příkaz 2;
```



Je-li `logickyVyras` vyhodnocen jako `true`, bude spuštěn příkaz 1. V opačném případě (kdy je `logickyVyras` vyhodnocen jako `false`) bude spuštěn příkaz 2. Klíčové slovo `else` a za ním následující příkaz 2 jsou nepovinné. Následující příklad rozhoduje, zda uživatel má podle věku nárok na bezplatné studium nebo již za studium bude platit.



```
Console.WriteLine("Program na rozeznání studenta a pracovníka");
Console.Write("Zadej svůj věk: ");
int vek = int.Parse(Console.ReadLine());
```

```
// podmínka větvení, jestliže je věk menší nebo rovno 26 je
// podmínka splněna
if (vek <= 26)
    Console.WriteLine("Studujes bezplatně.");
else
    Console.WriteLine("Za studium musíš platit.");
Console.ReadKey();
```

Kaskádové použití rozhodovacích příkazů se provádí postupným zanořováním do jiných příkazů `if`. Takto je možné zřetěžit posloupnost logických výrazů, které budou postupně vyhodnocovány, dokud jeden z nich nebude mít hodnotu `true`. Příkladem může být zjištění počtu knih v domácnosti s následným vyhodnocením typu čtenáře.

```
Console.WriteLine("Program knihomol");
Console.Write("Zadej počet knih v domácnosti: ");
// Parsování textové hodnoty pro proměnné pocetKnih
int pocetKnih = int.Parse(Console.ReadLine());
/*pokud není splněna první podmínka pokračuje se podmínkou druhou, pokud
ani tato podmínka není splněna pokračuje s následující dokud jedna z
možných podmínek je splněna */
if (pocetKnih <= 5)
    Console.WriteLine("Četba knih neuškodí.");
else if (pocetKnih <= 15)
    Console.WriteLine("Čtenář začátečník.");
else if (pocetKnih <= 25)
    Console.WriteLine("Čtenář pokročilý.");
else if (pocetKnih <= 50)
    Console.WriteLine("Čtenář profesionál.");
else if (pocetKnih <= 100)
    Console.WriteLine("Čtenář expert.");
else if (pocetKnih <= 500)
    Console.WriteLine("Čtenář knihomol.");
else if (pocetKnih <= 10000)
    Console.WriteLine("Knihovník.");
else if (pocetKnih > 10000)
    Console.WriteLine("Nemám slov.");
Console.ReadKey();
```

7.1.6 Rozhodování pomocí příkazu `switch`

V některých kaskádových příkazech `if` jsou si všechny příkazy podobné, protože všechny vyhodnocují stejný výraz. Jediným rozdílem je, že každý z příkazů `if` porovnává výsledek výrazu s jinou hodnotou. V těchto situacích je často možné vnořené příkazy `if` nahradit příkazem `switch`, který zlepší čitelnost kódu a zvýší jeho efektivitu. Syntaxe příkazu `switch` vypadá následovně:


```
switch (ridiciVýraz)
{
    case konstattníVýraz:
        příkazy;
        break;
    case konstattníVýraz:
```

```

    příkazy;
    break;
default:
    příkazy;
    break;}

```

Program na vyhodnocení zadaného aritmetického operátoru pomocí příkazu switch:




```

Console.WriteLine("Program aritmetický oeprátor");
Console.Write("Vlož operátor: ");
char znak = char.Parse(Console.ReadLine());
/* vyhodnocování výrazu podle příkazu switch - Hodnota každého
výrazu pocetKnih musí být jedinečná, aby tak řídící výraz mohl
odpovídat nejvýše jedné znich. Jestliže se hodnota pocetKnih
nerovná žádné z hodnot výrazů pocetKnih pokračuje se návěštím
default.*/
switch (znak)
{
    case '+':
        Console.WriteLine("Operátor plus.");
        break;
    case '-':
        Console.WriteLine("Operátor minus.");
        break;
    case '*':
        Console.WriteLine("Operátor násobení.");
        break;
    case '/':
        Console.WriteLine("Operátor dělení.");
        break;
    case '%':
        Console.WriteLine("Operátor modulo.");
        break;
    default:
        Console.WriteLine("Neznámý operátor.");
        break;
}
Console.ReadKey();

```

7.1.7 Operátory složeného přiřazení

Pokud chcete zvětšit hodnotu proměnné o 15, dá se zkombinovat operátor přiřazení a operátor sčítání.



```

int hodnota = 10;
hodnota = hodnota + 30;
Console.WriteLine(hodnota);
Console.ReadKey();

```

Tyto příkazy sice bezproblémově fungují, ale v praxi těžko narazíte na zkušeného programátora, který by je používal. Přičtení určité hodnoty k proměnné je totiž tak běžná operace, že jazyk C# má pro tuto operaci připraven operátor +=.



```
int hodnota = 10;
hodnota += 30;
Console.WriteLine(hodnota);
Console.ReadKey();
```

V tomto zkráceném zápisu jednoho operátoru místo dvou lze kombinovat libovolný aritmetický operátor s přiřazovacím operátorem, jak je vidět v následující tabulce. Tyto operátory jsou souhrnně označovány jako *operátory složeného přiřazení* (compound assignment operators). Tyto operátory mají stejnou prioritu a asociativitu (zprava doleva!) jako jednoduché operátory přiřazení.

Tab. 14. Operátory složeného přiřazení

Nevhodný zápis kombinace operátorů	Zkrácený zápis operátoru (doporučeno)
promenna = promenna + cislo;	promenna += cislo;
promenna = promenna - cislo;	promenna -= cislo;
promenna = promenna * cislo;	promenna *= cislo;
promenna = promenna / cislo;	promenna /= cislo;
promenna = promenna % cislo;	promenna %= cislo;

Operátor složeného přiřazení lze také použít při zřetězování textových řetězců:



```
string jmeno = "Pavel ";
string prijmeni = "Novak";
jmeno += prijmeni;
Console.WriteLine(jmeno);
```

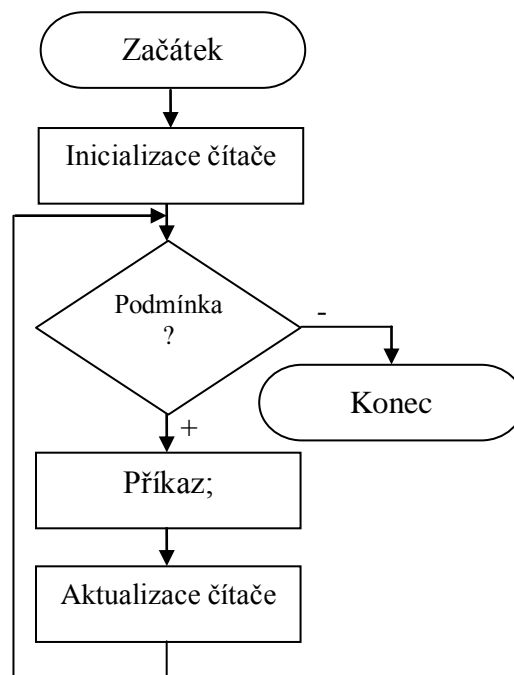
7.1.8 Cykly pomocí příkazu for

V příkazu `for` je inicializace, logický výraz a aktualizace řídicí proměnné zkombinován do jedné závorky. V tomto příkazu nelze zapomenout na žádnou z těchto tří částí. Jednotlivé části musí být odděleny od sebe středníkem. Pokud bude v tomto příkazu použito několik dalších příkazů je zapotřebí použití složených závorek. Syntaxe příkazu `for` je následující:

```
for (inicializace; logickyVyras; aktualizace řídící proměnné)
```



```
Console.WriteLine("Program pro výpis přirozených čísel");
Console.Write("Zadej počet čísel pro výpis od 1: ");
// Parsování textové hodnoty pro proměnné koncoveCislo
int koncoveCislo = int.Parse(Console.ReadLine());
// jestliže proměnná i je menší než koncoveCislo proved' cyklus
for (int i = 1; i <= koncoveCislo; i++)
{
    Console.WriteLine(i);
}
Console.ReadKey();
```



Inicializace proběhne jen jednou, na začátku cyklu. Poté, pokud má logický výraz hodnotu `true`, bude daný příkaz spuštěn. Pak se aktualizuje řídicí proměnná a následně se znovu vyhodnotí logický výraz. Je-li stále platný (tj. = `true`), znovu se spustí, načež se aktualizuje řídicí proměnná a znovu se vyhodnotí logický výraz, a tak stále dokola.



TIP – Inicializace řídicí proměnné, logický výraz a aktualizace řídicí proměnné musí být navzájem vždy odděleny středníky, ať už jsou vynechány či nikoliv.

7.1.8.1 Obor platnosti příkazu `for`


Jistě jste si všimli, že v inicializační části příkazu `for` je možné deklarovat proměnné. Obor platnosti těchto proměnných je omezen jen na tělo příkazu `for`, po jehož ukončení přestanou platit (existovat). Toto pravidlo má dva závažné důsledky. Za prvé, řídicí proměnnou nelze používat po ukončení cyklu jinde v programu, protože již není v oboru platnosti, jak ukazuje příklad:




```

Console.WriteLine("Zadej číslo: ");
int cislo = int.Parse(Console.ReadLine());
for (int i = 1; i <= cislo; i++)
{
    // Příkazy
}
Console.WriteLine(i); // Chyba při kompilaci
Console.ReadKey();
  
```

Za druhé, můžete napsat několik příkazů `for`, které mají stejně pojmenovanou řídicí proměnnou, protože každá z nich leží v jiném oboru platnosti:



```
Console.WriteLine("Zadej číslo: ");
int cislo = int.Parse(Console.ReadLine());
for (int i = 1; i <= cislo; i++ )
{
    Console.WriteLine(i);
}
Console.ReadKey();
```

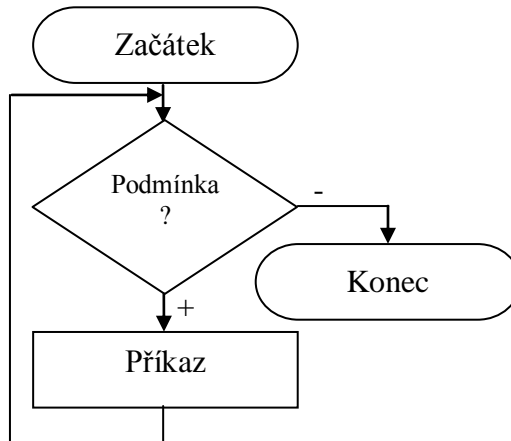


```
Console.WriteLine("Zadej číslo: ");
int cisloDve = int.Parse(Console.ReadLine());
for (int i = 1; i <= cisloDve; i++ ) // Bez problému funguje
{
    Console.WriteLine(i);
}
Console.ReadKey();
```

7.1.9 Cykly pomocí příkazu `while`

Příkaz `while` je používán pro opakované spouštění příkazu nebo příkazů, dokud platí určitá podmínka. Jedná se tedy o zástupce cyklu. Syntaxe příkazu `while` vypadá takto:

```
// inicializace
while (logickyVyras)
{
    příkaz;
    aktualizace řídicí proměnné;
}
```



Všeobecné podmínky použití příkazu `while`:

- Výraz v závorkách za příkazem `while` musí být logického typu a musí nabývat hodnoty `true` nebo `false`.
- Logický výraz je nutno zapsat do závorek.
- Pokud je hodnota logického výrazu hned napoprvé `false`, pak se příkaz neprovede ani jednou.
- Chcete-li v příkazu `while` provádět více než jeden příkaz, musíte všechny příkazy uzavřít složenými závorkami „{ }“ do bloku.

Princip práce příkazu while je: Logický výraz příkazu `while` je vyhodnocen, pokud tento výraz platí (tedy jeho hodnota je `true`), je spuštěn příkaz a poté je logický výraz vyhodnocen znovu. Pokud stále platí, je příkaz znovu spuštěn a následně je opět vyhodnocen logický výraz. Tento proces se opakuje, dokud není logický výraz vyhodnocen na hodnotu `false`. Pak příkaz `while` skončí a provádění programu dále pokračuje prvním příkazem za příkazem `while`.

Příklad: Program požádá uživatele pro zadání koncového přirozeného čísla. Toto číslo se uloží do proměnné `koncoveCislo`. Následně program vypíše číselnou řadu od 1 do `koncoveCislo`.



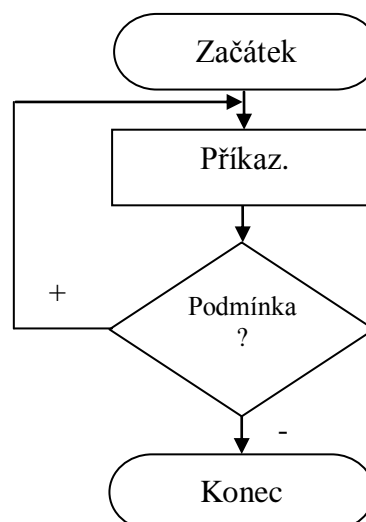
```
Console.WriteLine("Program pro výpis přirozených čísel");
Console.Write("Zadej počet čísel pro výpis od 1: ");
// Parsování textové hodnoty pro proměnné koncoveCislo
int koncoveCislo = int.Parse(Console.ReadLine());
int i = 0;
// jestliže proměnná i je menší než koncoveCislo proved' cyklus
while (i++ < koncoveCislo)
{
    Console.WriteLine(i);
}
Console.ReadKey();
```

7.1.10 Cykly pomocí příkaz do

Příkaz `do` je ve vyhodnocování výrazů odlišný od příkazu `for` a `while`, protože logický výraz je u příkazu `do` vyhodnocován až na konci každé iterace, takže tělo příkazu proběhne vždy alespoň jednou. U syntaxe je zapotřebí dát pozor na závěrečný středník a pokud se jedná o více příkazů tak na blokové rozlišení pomocí složených závorek.

Syntaxe příkazu `do` vypadá takto:

```
Do
{
    Příkaz 1;
    Příkaz 2;
}
While (logickyVyras);
```



Příklad: Upravená program výpisu čísel pomocí příkazu do:



```
Console.WriteLine("Program pro výpis přirozených čísel");
Console.Write("Zadej počet čísel pro výpis od 1: ");
// Parsování textové hodnoty pro proměnné koncoveCislo
int koncoveCislo = int.Parse(Console.ReadLine());
int i = 0;
// Cyklus se provede minimálně jednou
do
{
    Console.WriteLine(i);
}
// jestliže proměnná i je menší než koncoveCislo proved' cyklus
while (i++ < koncoveCislo);
Console.ReadKey();
```

7.1.10.1 Příkazy break a continue v cyklech:

Příkazem break lze vyskočit z těla iteračních příkazů (cyklů). Při opuštění cyklu je cyklus okamžitě ukončen a program pokračuje prvním příkazem za ním. Nedojde ani k aktualizaci řídicí proměnné, ani k vyhodnocení podmínky (logického výrazu).

Příkaz continue oproti tomu způsobí okamžitý přesun na začátek další iterace cyklu (po opětovném vyhodnocení logického výrazu).

Příklad: Upravená program výpisu čísel pomocí příkazu break a constinue.



```
Console.WriteLine("Program pro výpis přirozených čísel");
Console.Write("Zadej počet čísel pro výpis od 1: ");
// Parsování textové hodnoty pro proměnné koncoveCislo
int koncoveCislo = int.Parse(Console.ReadLine());
int i = 0;
// Cyklus se provede minimálně jednou
while (true)
{
    Console.WriteLine(++i);
    // jestliže proměnná i je menší než koncoveCislo proved' cyklus
    jinak práci ukonči
    if (i < koncoveCislo)
        continue;
    else
        break;
}
Console.ReadKey();
```

7.2 Otázky z probraného tématu

- ❖ Jakou logickou proměnou znáš?
- ❖ K čemu se používají logické operátory, vzpomeneš si na 3 příklady?
- ❖ Jak vypadá operátor „a zároveň“ a operátor „nebo“?
- ❖ Jaký je rozdíl mezi postfixem a prefixem?

- ❖ Jaký rozhodovací příkaz v jazyku C# znáš?
- ❖ Jaký příkaz v jazyku C# se využívají na realizaci cyklu?
- ❖ Napiš syntaxi příkazu for, while, switch, do.

7.3 Praktické cvičené na dané téma

- (22) Vytvoř konzolovou aplikaci, která bude mít za úkol výpis textu „Ahoj kamaráde.“
Počet výpisu textu zadá uživatel sám. Číslování vypsaného textu bude také obsahem výpisu například takto: „1.krát vypsáno „Ahoj kamaráde.““
- (23) Vytvoř konzolovou aplikaci, která bude mít za úkol výpis čísel intervalu od 0 do 50 s krokem 5. Jednotlivé čísla odděluj čárkou.
- (24) Vytvoř konzolovou aplikaci, která bude mít za úkol součet čísel intervalu. Interval si uživatel bude moci zvolit sám. Výsledek zobraz do konzole.
- (25) Vytvoř konzolovou aplikaci, do které přepíše následující zdrojový kód pomocí příkazu for a while:



```
Console.WriteLine("Program úpravy zdrojového kódu");  
Console.Write("Pejsku třikrát zaštěkej: ");  
Console.Write("Haf ");  
Console.Write("Haf ");  
Console.Write("Haf ");  
Console.ReadKey();
```

7.4 Řešení praktického cvičení

- (22) *Program, který má za úkol vypsat text a jeho číslování podle vzoru: „1.krát vypsáno „Ahoj kamaráde.““*



```
Console.WriteLine("Program pro výpis zadaného textu x-krát");  
Console.Write("Zadej počet výpisů: ");  
// Parsování textové hodnoty pro proměnné koncoveCislo  
int pocetVypisu = int.Parse(Console.ReadLine());  
// jestliže proměnná i je menší než pocetVypisu proved cyklus  
for (int i = 1; i <= pocetVypisu; i++)  
{  
    Console.WriteLine("{0}.krát vypsáno \"Ahoj kamaráde.\"\"", i);  
}  
Console.ReadKey();
```

(23) *Program, výpisu intervalu 0 – 100 s krokem 5*



```
Console.WriteLine("Program pro výpis čísel intervalu 0 - 50 s krokem 5");  
// jestliže proměnná i je menší než pocetVypisu proved' cyklus  
for (int i = 0; i <= 50; i+=5)  
{  
    Console.Write("{0}, ", i);  
}  
Console.ReadKey();
```

(24) *Program, součtu jednotlivých čísel zadaného intervalu*



```
Console.WriteLine("Program pro součet kladných čísel zadaného intervalu");  
Console.Write("Zadej počátek intervalu: ");  
int dolniMezIntervalu = int.Parse(Console.ReadLine());  
Console.Write("Zadej konec intervalu: ");  
int horniMezIntervalu = int.Parse(Console.ReadLine());  
// jestliže proměnná i je menší než horniMezIntervalu proved' cyklus  
int vysledek = 0;  
for (int i = dolniMezIntervalu; i <= horniMezIntervalu; i++)  
{  
    vysledek += i;  
}  
Console.Write("Výsledkem součtu čísel intervalu je: {0}", vysledek);  
Console.ReadKey();
```

(25) *Program, úpravy zdrojového kódu pomocí cyklu while a for*



```
Console.WriteLine("Program úpravy zdrojového kódu");  
Console.Write("Pejsku třikrát zašťkej: ");  
Console.Write("Haf ");  
Console.Write("Haf ");  
Console.Write("Haf ");  
  
string strFor = "";  
for (int i = 1; i <= 3; i++)  
{  
    strFor += "Haf ";  
}  
Console.WriteLine();  
Console.WriteLine("Cyklus for:");  
Console.WriteLine("Pejsku třikrát zašťkej: " + strFor);  
  
string strWhile = "";  
int j = 1;  
while (j++ <= 3)  
{  
    strWhile += "Haf ";  
}  
Console.WriteLine("Cyklus while:");  
Console.WriteLine("Pejsku třikrát zašťkej: " + strWhile);  
Console.ReadKey();
```

7.5 Seznam použité literatury

- [1] ELLER, Frank. *C# - začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2002, 240 s. ISBN 80-247-0324-6.
- [2] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2008, 592 s. ISBN 978-80-251-2027-9.

8 METODICKÝ LIST – ČÍSLO 8



CÍLEM TÉMATU – ošetření uživatelského vstupu, využití výjimek `c#` pro zachytávání problémů a práce s polem



KLÍČOVÁ SLOVA – výjimka, `try`, `catch`, `TryParse`, pole



ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 90 minut.

8.1 Ošetření uživatelského vstupu, výjimky a pole

8.1.1 Ošetření uživatelského vstupu

Ošetření uživatelského vstupu patří mezi jednu z nejdůležitějších fází vývoje softwaru. Uživatele musí programátor chápat za velmi málo gramotného v oblasti informačních technologií. Při vývoji se řiďte pravidlem, čím hloupějšího uživatele budete očekávat při vývoji aplikace, tím větší úspěch budou vaše aplikace mít.

K ověření správnosti zadaného uživatelského vstupu při jeho parsování lze místo metody `Parse()` použít metodu `TryParse()`. Tato metoda vrací logické hodnoty `true/false` podle toho, jestli se parsování podařilo či nikoli. Tato metoda používá modifikátor *out*. Klíčové slovo `out` je zkratka slova "output" (výstup). Když předáte metodě parametr `out`, musí mu metoda přiřadit hodnotu.



```
Console.WriteLine("Program ošetření uživatelského vstupu");
Console.Write("Zadej číslo: ");
int cislo;
while (!int.TryParse(Console.ReadLine(), out cislo))
    Console.Write("Neplatné zadání, zadej číslo znovu: ");
Console.Write("Tebou zadané číslo je: {0}", cislo);
Console.ReadKey();
```

Program vyzve uživatele k zadání čísla. Poté deklaruje proměnnou `cislo`. Do podmínky `while` cyklu je vložena metoda `TryParse`, pomocí operátoru negace „!“ podmínku znegujeme. Pokud `while` vrací logickou hodnotu `false`, bude se cyklus stále opakovat a vyzývat uživatele k novému zadání. Pokud se zadaný text z konzole parsuje, vkládá do proměnné `cislo`, je vrácena hodnota `true`. Cyklus tímto končí.

8.1.2 Výjimky

Špatné věci se prostě stávají, to je život. Někdy prostě k chybě programu dojde buď to při jeho startu nebo za běhu či při ukončení. Nemusí jít vždy jen o vinu programátora,


velkou měrou se na tom podílí i uživatel. Ten buď neznalostí systému, nebo negramotností zapříčiní pád programu. To se prostě stane. Proto velké softwarové firmy využívají pomoc testerů. Je proto zapotřebí co největší počet problémů odstranit v zárodku nebo při jeho vývoji. Jazyk C# a většina dalších moderních objektově orientovaných jazyků řeší ošetření chyb specifickým způsobem a to za použití výjimek. Chcete-li psát v jazyku C# robustní programy, musíte výjimky dobře ovládat.

8.1.2.1 Příkazy *try*, *catch* a *finally*

V jazyku C# se dá pomocí výjimek a obslužných rutin výjimek velmi snadno oddělit kód, který představuje hlavní tok programu, od kódu pro ošetření chyb. Při psaní kódu je nutné dodržet toto:

Svůj kód píšete vždy do bloků `try` (`try` je klíčové slovo jazyka C#). Jakmile dojde k jeho spuštění, pokusí se provést všechny příkazy v bloku `try`, a pokud žádný z nich nevyvolá výjimku, proběhnou všechny postupně jeden po druhém až do konce. Při nalezení problému se skočí do bloku `catch`.

Bloků `catch` (bezprostředně umístěny za blok `try`) ošetří všechny možné chybové stavy. Slouží k zachycení a ošetření specifického typu výjimky. Po bloku `try` můžete následovat několik bloků `catch`, každý navržený pro zpracování specifické výjimky. Může tedy mít různé obslužné rutiny pro různé typy chyb, k nimž může dojít v bloku `try`.



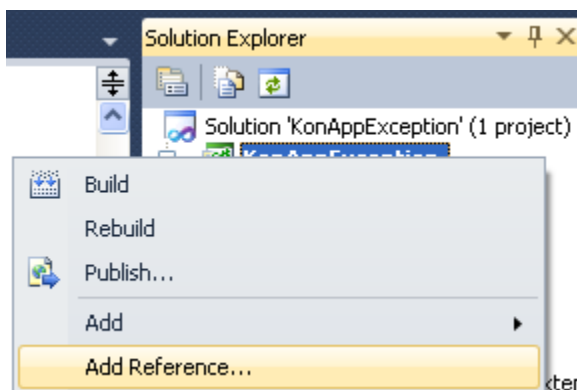
```
try
{
    Console.WriteLine("Program ošetření výjimek při dělení");
    Console.Write("Zadej čitatele: ");
    int citatel = int.Parse(Console.ReadLine());
    Console.Write("Zadej jmenovatele: ");
    int jmenovatel = int.Parse(Console.ReadLine());
    int vysledek = citatel/jmenovatel;
    Console.WriteLine("Výsledkem dělení čísel: {0}", vysledek);
    Console.ReadKey();
}
catch (DivideByZeroException)
{
    Console.WriteLine("Nelze dělit nulou!");
    Console.ReadKey();
}
catch (OverflowException)
{
    Console.WriteLine("Zadal jsi moc velké číslo. Interval je {0} a {1}", int.MinValue, int.MaxValue);
    Console.ReadKey();
}
catch (FormatException)
{
}
```

```
Console.WriteLine("Vložil jsi desetinné číslo, lze jen  
celočíslné.");  
Console.ReadKey();  
}
```

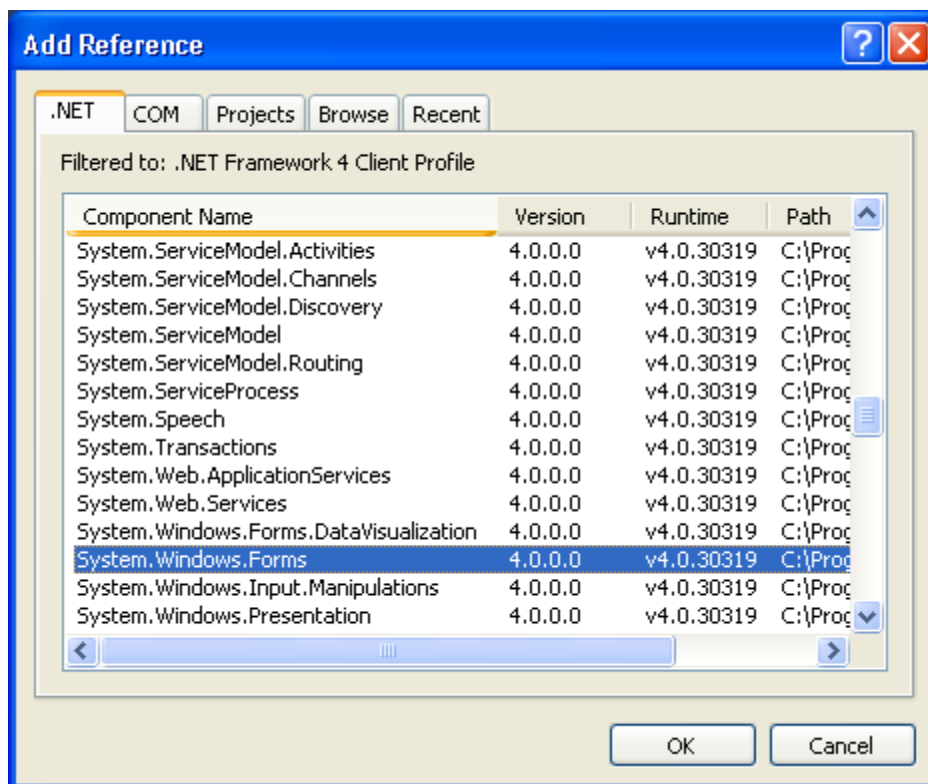
Blok `Finally` zajistí, aby byl nějaký příkaz vždy spuštěn, ať už někde před ním byla či nebyla vyvolána výjimka. Tento blok se umísťuje bezprostředně za blok `try` nebo za poslední blok `catch` uvedený za blokem `try`. Pokud program začne provádět blok `try` s přidruženým blokem `finally`, bude obsah bloku `finally` proveden vždy, i když dojde k výjimce.

8.1.2.2 Ošetření výjimky

Syntaxe obslužné rutiny `catch` se podobá metodě. Výjimka, která má být zachycena, je vyjádřena jako parametr. V předchozím příkladu bude při vyvolání výjimky typu `DivideByZeroException` proměnná `exDivideByZero` naplněna objektem obsahujícím podrobné údaje o výjimce. Typ `DivideByZeroException` obsahuje množství položek, které lze prozkoumat a určit z nich přesnou příčinu výjimky. Řada vlastností se vyskytuje u všech typů výjimek. Pro použití tohoto ošetření u konzolové aplikace nutné nejdříve přidat do konzolového projektu referenci `System.Windows.Form` přes kontextové menu:



Obr. 24. Položka menu pro přidání reference



Obr. 25. Výběr reference Systém.Windows.Forms

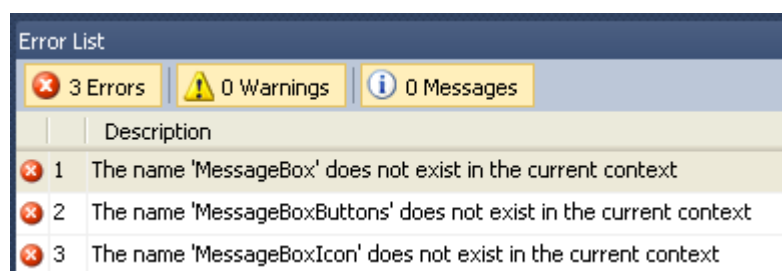
Poté je nutné vložit do projektu direktivu s třídou `using System.Windows.Forms;` proto, aby kompilátor při překladu nám nehlásil chyby a intellisense obsahoval dialogové okno třídy `MessageBox()`.

```

catch (DivideByZeroException exDivideByZero)
{
    string exDiv = exDivideByZero.Message;
    MessageBox.Show(exDiv, "Error", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
    Console.WriteLine("Nelze dělit nulou!");
    Console.ReadKey();
}

```

Bez použití direktivy `using System.Windows.Forms;` by Vám kompilátor hlásil tyto chyby:



Obr. 26. Konzolová aplikace bez direktivy

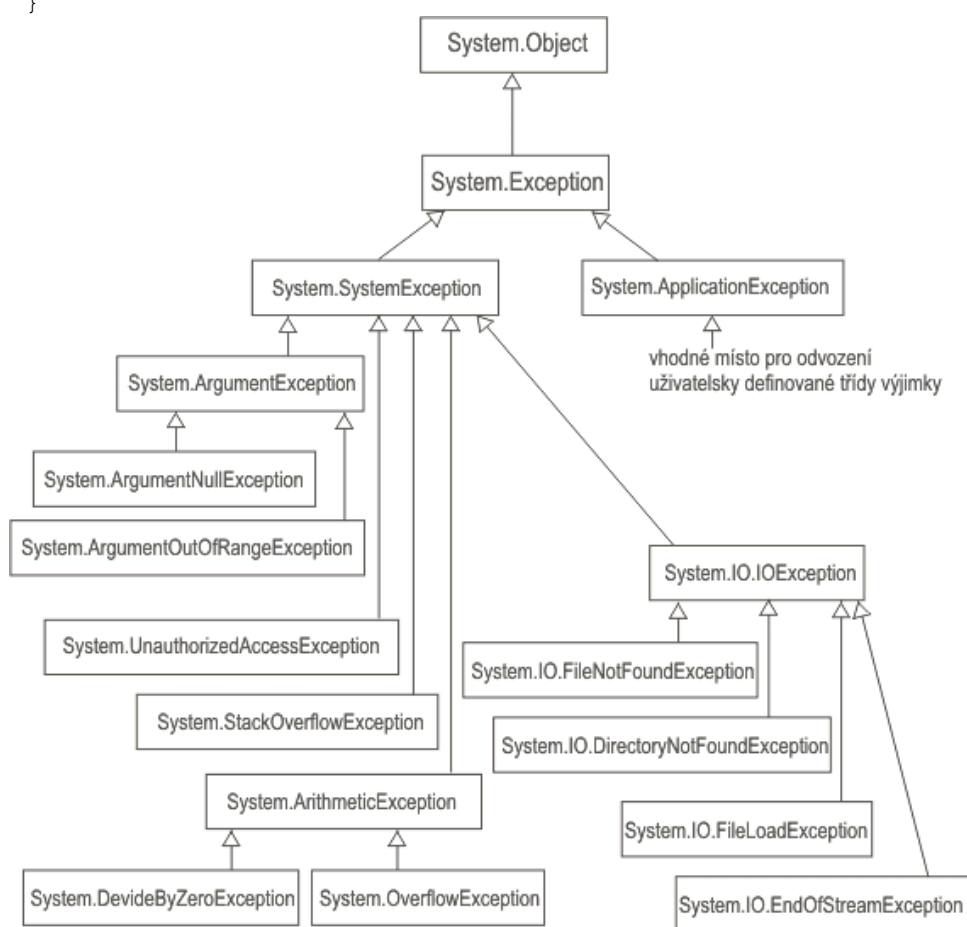
`System.Windows.Forms`

8.1.3 Zachytávání výjimek několika typů obslužnou rutinou

Namísto individuálního zachytávání jednotlivých typů výjimek je vhodnější vytvořit obslužnou rutinu, která zachytí přímo výjimky typu `System.Exception`. Výjimka typu `System.Exception` je členem rodiny s prostým názvem `Exception`, což je prapůvodce všech ostatních typů výjimek. Pokud budete v obslužné rutině zachytávat výjimky typu `Exception`, bude tato rutina spuštěna pro všechny vůbec možné typy výjimek.



```
try
{
    Console.WriteLine("Program ošetření výjimek při dělení");
    Console.Write("Zadej čitatele: ");
    int citatel = int.Parse(Console.ReadLine());
    Console.Write("Zadej jmenovatele: ");
    int jmenovatel = int.Parse(Console.ReadLine());
    int vysledek = citatel / jmenovatel;
    Console.WriteLine("Výsledkem dělení čísel: {0}", vysledek);
    Console.ReadKey();
}
catch (Exception ex)
{
    string except = ex.Message;
    MessageBox.Show(except, "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
```



Obr. 27. Hierarchie Exception třídy

8.1.4 Pole

Pole je neuspořádaná posloupnost prvků. Všechny prvky v poli jsou stejného typu (na rozdíl od datových složek ve strukturách nebo třídách, kde jsou povoleny rozdílné typy). Prvky pole jsou uloženy v souvislém bloku paměti a pro přístup k nim slouží celočíselný index. Prvním indexem je vždy 0.



UPOZORNĚNÍ – Indexy pole začínají vždy nulou. První prvek pole má tedy index 0, nikoli 1! Index 1 patří druhému prvku. Při každém přístupu k prvkům pole se kontroluje, zda hodnota indexu leží v mezích daného pole. Jestliže uvedete celočíselný index menší než nula, nebo větší či rovný velikosti pole, vyvolá program výjimku typu `IndexOutOfRangeException`.

8.1.4.1 Deklarace pole

Deklarace proměnné typu pole se provádí názvem datového typu pro všechny prvky pole, za kterým následuje dvojice hranatých závorek, mezera, jméno proměnné a středník. Hranaté závorky signalizují, že proměnná představuje pole. `datovyTyp[]`
`nazevPromene;`

Příklad pole s datovým typem `int`: `int[] pole`

8.1.4.2 Vytvoření instance pole

Instance pole se vytváří pomocí klíčového slova `new`, za kterým následuje název typu uchovávaných prvků a v hranatých závorkách velikost pole. Při vytvoření pole jsou také inicializovány jeho prvky (nám již známými) výchozími hodnotami, odpovídajícími zvolenému typu (0, `null` nebo `false`, v závislosti na tom, jde-li o typ číselný, referenční nebo logický).

Příklad pole s datovým typem `int` a jeho instancí: `int[] pole = new int[5];`

Velikost instance pole samozřejmě nemusí být stanovena pomocí konstanty, může být zadána uživatelem:

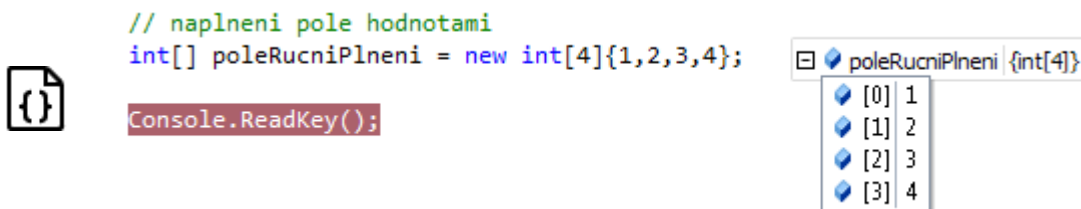


```
Console.WriteLine("Zadej velikost pole: ");  
int velikostPole = int.Parse(Console.ReadLine());  
int[] pole = new int[velikostPole];
```

V programovacím jazyku C# je povoleno vytvořit pole s nulovou velikostí. Vypadá to sice jako dosti bizarní nápad, ale hodí se to v situacích, kdy je velikost pole určována dynamicky a může nabýt i nulové hodnoty. Pole o velikosti 0 není prázdné pole, ale obsahuje `null`.

8.1.4.3 Naplněné pole

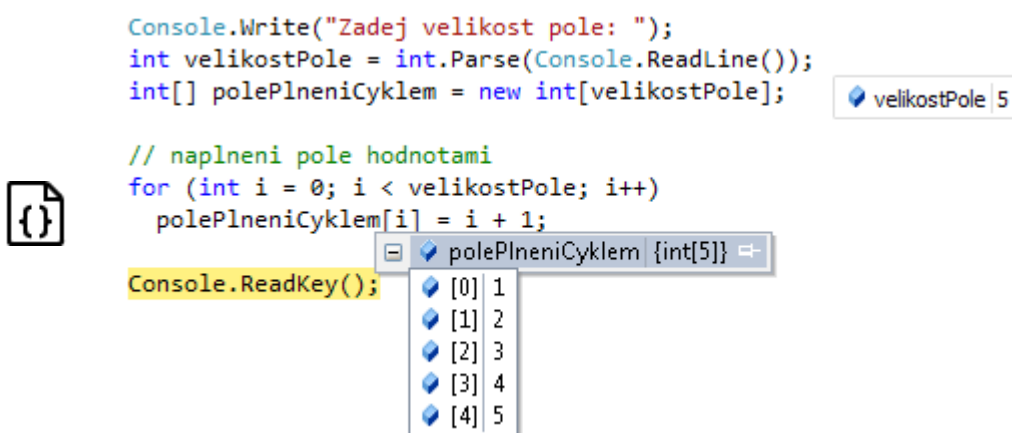
Naplnění pole lze ručně nebo pomocí cyklu například `for`. Pokud pole budete plnit ručně, provádí se pomocí složených závorek umístěné za velikosti pole. Příkladem může být tento:



Tab. 15. Ukázka prvků pole a jejich indexů

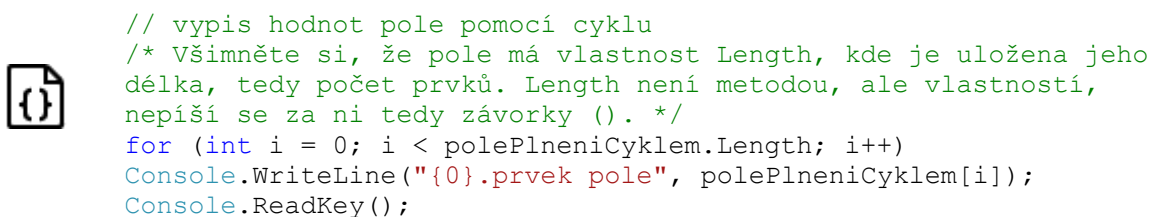
Hodnota prvku pole	1	2	3	4
Index pole	[0]	[1]	[2]	[3]

Naplnění pole pomocí cyklu `for`:



8.1.4.4 Výpis hodnot pole

Provádí se opět pomocí cyklu `for`. Názorný příklad uvádí jak by mohlo být pole vypsáno:



8.1.5 Užitečné metody pole

Programovací jazyk C# obsahuje celou řadu metod při práci s polem. Uvedl jsem jen několik názorných příkladů, jelikož by popis všech metod vydalo na knihu. Proto doporučuji při větším zájmu o problematiku s polem v jazyku C# samostudium.

8.1.5.1 Metoda Sort()

Je vhodná pro setřídění pole. Jediným parametrem této metody je pole, které je zapotřebí setřídít. Tato metoda je tak chytrá, že prvky pole třídí podle datového typu. Tedy textové řetězce podle abecedy, čísla podle velikosti.



```
// rucni naplneni pole hodnotami
int[] pole = new int[5] { 5, 2, 1, 4, 3 };
// vypis hodnot pole pomocí cyklu
Console.WriteLine("Nesetridene pole: ");
for (int i = 0; i < pole.Length; i++)
    Console.Write("{0} ", pole[i]);
Console.WriteLine("\nSetridene pole: ");
//Metoda na setřídění pole.
Array.Sort(pole);
foreach (int i in pole)
    Console.Write("{0} ", i);
Console.ReadKey();
```

8.1.5.2 Metoda Average

Tato metoda slouží na výpočet průměru položek čísel v poli. Příkladem můžou být nejruznější výpočty průměru známek, průměru váhy a podobných dat



```
Console.Write("Zadej velikost pole: ");
int velikostPole = int.Parse(Console.ReadLine());
int[] poleCisel = new int[velikostPole];
// naplneni pole hodnotami cyklem
for (int i = 0; i < velikostPole; i++)
{
    Console.Write("Zadej {0}.prvek pole = ", i + 1);
    poleCisel[i] = int.Parse(Console.ReadLine());
}
// vypis hodnot pole pomocí cyklu
Console.WriteLine("Průměr zadaných čísel je: {0}",
    poleCisel.Average());
Console.ReadKey();
```

8.1.6 Metoda Min a Max

Jak název metody napovídá tak touto metodou se vyhledávají nejmenší prvky a největší prvky daného datového typu v procházeném poli.



```
Console.Write("Zadej velikost pole: ");
int velikostPole = int.Parse(Console.ReadLine());
int[] poleCisel = new int[velikostPole];
// naplneni pole hodnotami cyklem
for (int i = 0; i < velikostPole; i++)
{
    Console.Write("Zadej {0}.prvek pole = ", i + 1);
    poleCisel[i] = int.Parse(Console.ReadLine());
}
// vypis hodnot pole pomocí cyklu
Console.WriteLine("Minimální hodnota prvku je: {0}",
    poleCisel.Min());
Console.WriteLine("Maximální hodnota prvku je: {0}",
    poleCisel.Max());
Console.ReadKey();
```

8.2 Otázky z probraného tématu

- ❖ Co víš o metodě TryParse()?
- ❖ Jaké jsou rozdíly mezi bloky try, catch a finally?
- ❖ K čemu jsou používány v programovacím jazyku C# výjimky?
- ❖ Co je to pole?
- ❖ Jakým číslem začíná index u pole?

8.3 Praktické cvičení na dané téma

- (26) Vytvoř konzolovou aplikaci, která prohledá pole poleCisel={-1, -4, -20, -34, 0, 1, 4, 20, 34}, nalezne nejmenší číslo a vypíše toto číslo s pozicí/indexu pole.
- (27) Vytvoř konzolovou aplikaci, která prohledá pole poleHesel = { "QWERTZ", "123", "Pass", "Heslo", "ZXC", "MNB", "PoLkMn" }; pokud uživatel zadá správné heslo vypíše se do konzole „Vítej v programu.“, nenalezne-li program heslo vypíše „Špatně zadané heslo.“
- (28) Vytvoř konzolovou aplikaci, která bude simulovat výpočet malé násobilky. Uživatel bude moci zadat jaké hodnoty má program vynásobit. Malá násobilka se bude týkat interval 1 – 10. Program vhodně ošetří před uživatelským zadáním. Každý řádek malé násobilky bude vypsán jinou barvou, pomocí kódu `Console.ForegroundColor = (ConsoleColor)(velikostPole - 1++);`

8.4 Řešení praktického cvičení

- (26) Vytvoř konzolovou aplikaci, která prohledá pole `poleCisel={-1, -4, -20, -34, 0, 1, 4, 20, 34}`, nalezne nejmenší číslo a vypíše toto číslo s pozicí/indexu pole.

```
int[] poleCisel = new int[9] { -1, -4, -20, -34, 0, 1, 4, 20, 34 };
int min = poleCisel[0];
int pozicePole = 0;
int indexPole = 0;
string str = "";
for (int i = 0; i < poleCisel.Length; i++)
{
    if (poleCisel[i] < min)
    {
        min = poleCisel[i];
        pozicePole = i + 1;
        indexPole = i;
    }
    // Využití cyklu for pro naplnění řetězce hodnotami cyklu
    // Třída Convert obsahuje přetíženou metodu ToString, která převede datový typ int na string
    str += Convert.ToString(poleCisel[i]) + " ";
}
Console.WriteLine("Program na nalezení nejmenšího prvku pole na dané pozici");
Console.WriteLine("Pole je naplněné čísly: {0}", str);
// vypis nejmenší nalezé hodnoty pole pomocí cyklu
Console.WriteLine("Hodnota nejmenšího prvku je {0} na {1}/{2} (pozici/indexu) pole", min,
    pozicePole, indexPole);
Console.ReadKey();
```

- (27) Vytvoř konzolovou aplikaci, která prohledá pole `Hesel = { "QWERTZ", "123", "Pass", "Heslo", "ZXC", "MNB", "PoLkMn" }`; a vyhledává zadané heslo v poli.

```
Console.WriteLine("Program na vyhledávání hesla z pole hesel!");
string[] poleHesel = { "QWERTZ", "123", "Pass", "Heslo", "ZXC", "MNB", "PoLkMn" };
Console.Write("Zadej heslo: ");
string heslo = Console.ReadLine().ToLower();
int poziceHesla = Array.IndexOf(poleHesel, heslo);
if (poziceHesla >= 0)
    Console.WriteLine("Vítej v programu.");
else
    Console.WriteLine("Špatně zadané heslo!");
Console.ReadKey();
```

(28) Program malá násobilka pomocí pole s ošetřením uživatelských vstupů

```
Console.WriteLine("Program malá násobilka");
Console.Write("Zadej číslo pro malou násobilku: ");
int velikostPole;
int l = 0;
while ((!int.TryParse(Console.ReadLine(), out velikostPole)))
    Console.Write("Neplatné zadání, zadej číslo znovu: ");
if ((velikostPole >= 1) && (velikostPole <= 10))
{
    int[] poleNasobilka = new int[velikostPole];
    // naplneni pole hodnotami cyklem
    for (int i = 0; i < velikostPole; i++)
        poleNasobilka[i] = i + 1;
    for (int j = poleNasobilka[0]; j <= poleNasobilka.Length; j++)
    {
        // Barvu písma lze nastavit pomocí vlastnosti ForegroundColor
        Console.ForegroundColor = (ConsoleColor)(velikostPole - l++);
        for (int k = poleNasobilka[0]; k <= poleNasobilka.Length; k++)
        {
            Console.Write("{0} ", k * j);
        }
        Console.WriteLine();
    }
}
else
    Console.WriteLine("Nezadal jsi číslo z intervalu 1 - 10, program bude ukončen.");
Console.ReadKey();
```

8.5 Seznam použité literatury

- [1] ELLER, Frank. *C# - začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2002, 240 s. ISBN 80-247-0324-6.
- [2] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2008, 592 s. ISBN 978-80-251-2027-9.
- [3] Programování se zaměřením na .NET a jazyk C#. *Výjimky* [online]. 2006 [cit. 2013-05-15]. Dostupné z: <http://projektysipvz.gytool.cz/ProjektySIPVZ/Default.aspx?uid=262>

9 METODICKÝ LIST – ČÍSLO 9



CÍLEM TÉMATU – je z nasbíraných vědomostí realizovat větší praktické příklady



ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 180 minut.

9.1 Praktické cvičení

(29) Vytvoř konzolovou aplikaci, která bude mít za úkol porovnání dvou hodnot zadaných od uživatele. Výsledkem budou seřazená čísla od nejmenšího. Ošetři uživatelský vstup. Nejdříve vytvoř k tomuto zadání algoritmus bez ošetření uživatelského vstupu.

(30) Vhodně doplň kód hádání čísla z intervalu 0 – 100. Počet pokusů na uhodnutí čísla bude mít uživatel 10.



```
Console.WriteLine("Program hádání čísel");
Random nahodne = new Random();
int nahodneCislo = nahodne.Next(1, 100);
int odhad = 0;
int pocetOdhadu = 0;
do
{
    // Doplň kód
} while (odhad != nahodneCislo);
Console.ReadKey();
```

(31) Vytvoř konzolovou aplikaci jednoduché kalkulačky s ošetřením uživatelských vstupů. Pro tuto aplikaci se pokuste vytvořit algoritmus.

(32) Vytvoř konzolovou aplikaci, která bude mít za úkol porovnání dvou hodnot zadaných od uživatele pomocí třech proměnných. Výsledkem budou seřazená čísla od nejmenšího. Ošetři uživatelský vstup. Nejdříve vytvoř k tomuto zadání algoritmus bez ošetření uživatelského vstupu.

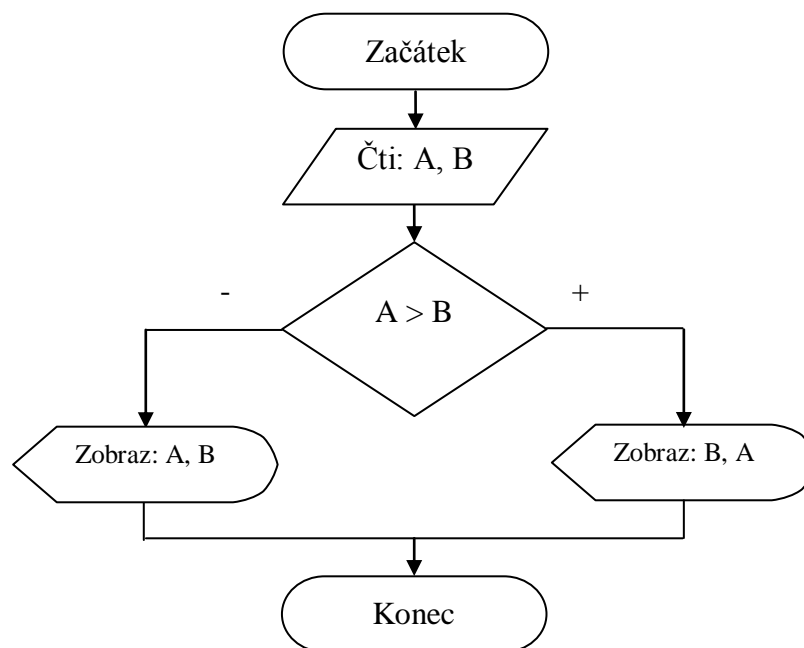
(33) Vytvoř konzolovou aplikaci, která bude mít za úkol naučit papouška mluvit. Dále se uživatele bude ptát, zda chce ukončit práci s programem nebo nadále papouška učit.

9.2 Řešení praktického cvičení

(29) Program porovnání a řazení čísel



```
Console.WriteLine("Program porovnání a řazení čísel od nejmenšího");
Console.Write("Zadej číslo A: ");
int cisloA;
while ((!int.TryParse(Console.ReadLine(), out cisloA)))
    Console.Write("Neplatné zadání, zadej číslo znovu: ");
Console.Write("Zadej číslo B: ");
int cisloB;
while ((!int.TryParse(Console.ReadLine(), out cisloB)))
    Console.Write("Neplatné zadání, zadej číslo znovu: ");
if (cisloA > cisloB)
    Console.WriteLine("Číslo B: {0}, číslo A: {1}", cisloB, cisloA);
else
    Console.WriteLine("Číslo A: {0}, číslo B: {1}", cisloA, cisloB);
Console.ReadKey();
```



(30) Program hádání čísel

```

Console.WriteLine("Program hádání čísel - max. povoleno 10
tipů");
Random nahodne = new Random();
int nahodneCislo = nahodne.Next(1, 100);
int odhad = 0;
int pocetOdhadu = 0;
do
{
    pocetOdhadu++;
    if (pocetOdhadu > 10)
    {
        Console.WriteLine("Nepodařilo se ti uhodnout číslo na
{0}.pokus. ", pocetOdhadu-1);
        break;
    }
    Console.Write("Zadej svůj {0}.odhad: ", pocetOdhadu);
    while (!int.TryParse(Console.ReadLine(), out odhad))
        Console.Write("Neplatné zadání, zadej číslo znovu: ");
    if (odhad < nahodneCislo)
        Console.WriteLine("Hadej vetsi cislo.");
    if (odhad > nahodneCislo)
        Console.WriteLine("Hadej mensi cislo.");
    if (odhad == nahodneCislo)
        Console.WriteLine("Uhodl jsi hledané číslo {0} na
{1}.pokus.", nahodneCislo, pocetOdhadu);
} while (odhad != nahodneCislo);
Console.ReadKey();

```

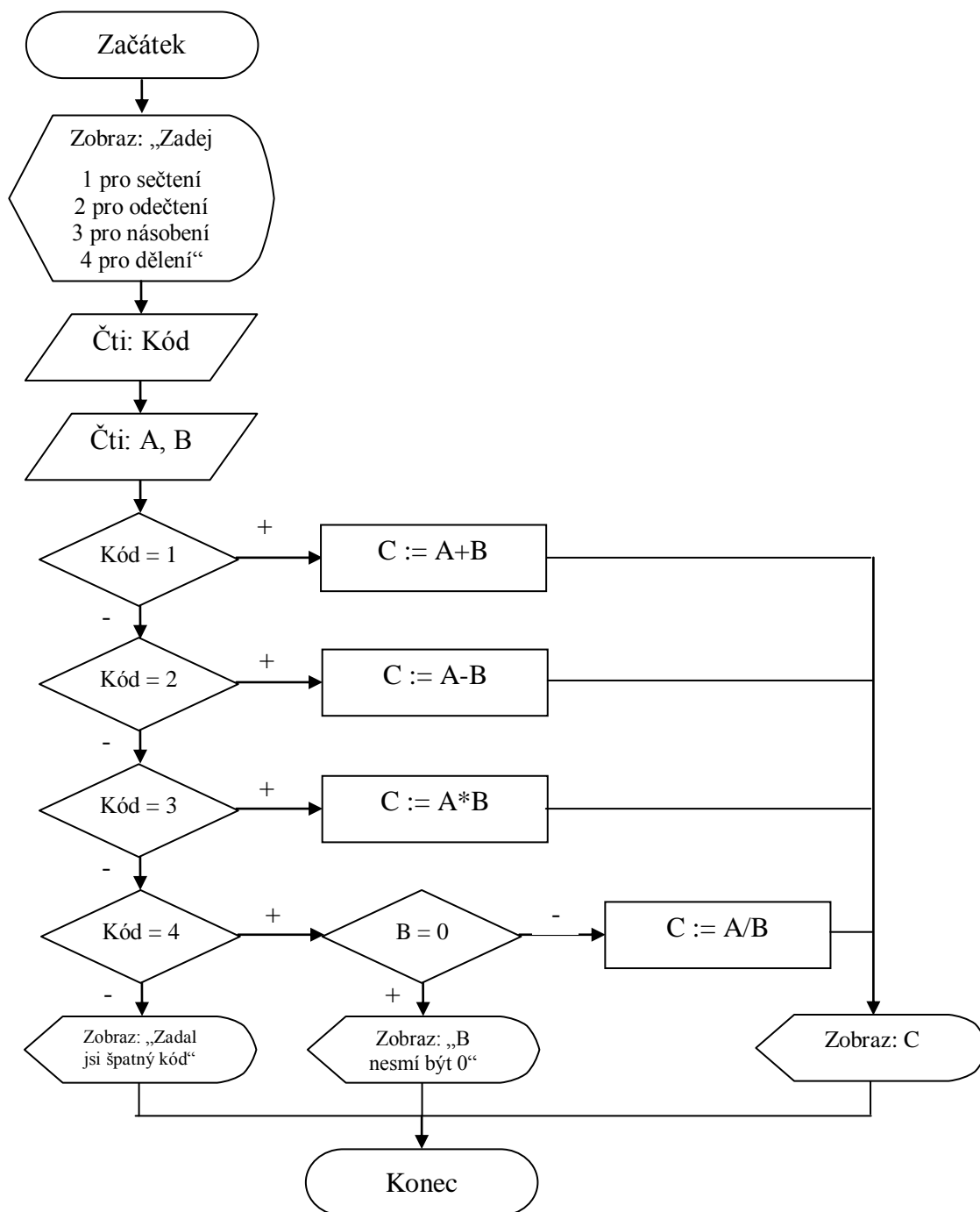
(31) Vytvoř konzolovou aplikaci jednoduché kalkulačky s ošetřením uživatelských vstupů. Pro tuto aplikaci se pokuste vytvořit algoritmus.


```

Console.WriteLine("Vítej v programu jednoduchá kalkulačka");
Console.Write("Zadej první číslo: ");
float cisloA;
while (!float.TryParse(Console.ReadLine(), out cisloA))
    Console.WriteLine("Neplatná hodnota, zadejte prosím znovu:");
Console.Write("Zadej druhé číslo: ");
float cisloB;
while (!float.TryParse(Console.ReadLine(), out cisloB))
    Console.WriteLine("Neplatná hodnota, zadejte prosím znovu:");
Console.WriteLine("Pro danou matematickou funkci zvol danou
ciselnou volbu:");
Console.WriteLine("Volba - 1: Scitani");
Console.WriteLine("Volba - 2: Odecitani");
Console.WriteLine("Volba - 3: Nasobeni");
Console.WriteLine("Volba - 4: Deleni");
int volbaSwitch = int.Parse(Console.ReadLine());
float vysledek = 0;
bool delenoNulou = false;
switch (volbaSwitch)
{
    case 1:
        vysledek = cisloA + cisloB;
        break;
    case 2:
        vysledek = cisloA - cisloB;

```

```
        break;
    case 3:
        vysledek = cisloA * cisloB;
        break;
    case 4:
        if (cisloB == 0)
        {
            Console.WriteLine("Dělení nulou nelze!");
            delenoNulou = true;
        }
        else
            vysledek = cisloA / cisloB;
        break;
    }
    if (!delenoNulou)
    {
        if ((volbaSwitch > 0) && (volbaSwitch < 5))
            Console.WriteLine("Vysledek cisel {0} a {1} je {2}:", cisloA,
                                cisloB, vysledek);
        else
            Console.WriteLine("Zvolil jsi spatnou volbu nez cisla mezi 1 -
                                4");
    }
    else
        Console.WriteLine("Pokracuj stiskem libovolné klávesy...");
    Console.ReadLine();
```



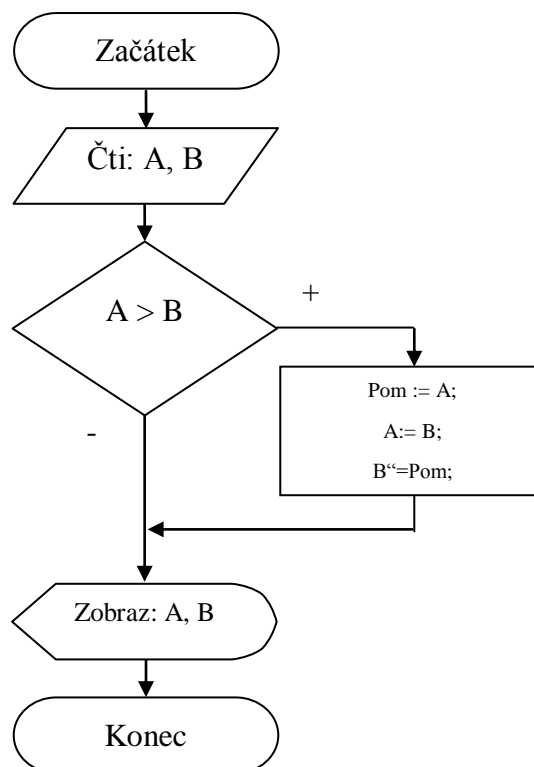
(32) Program seřazení čísel s pomocnou proměnnou



```

Console.WriteLine("Program porovnání a seřazení čísel od nejmenšího");
Console.Write("Zadej číslo A: ");
int cisloA;
while ((!int.TryParse(Console.ReadLine(), out cisloA)))
    Console.Write("Neplatné zadání, zadej číslo znovu: ");
Console.Write("Zadej číslo B: ");
int cisloB;
while ((!int.TryParse(Console.ReadLine(), out cisloB)))
    Console.Write("Neplatné zadání, zadej číslo znovu: ");
if (cisloA > cisloB)
{
    int pom = 0;
    pom = cisloA;
    cisloA = cisloB;
    cisloB = pom;
}
Console.WriteLine("Číslo A: {0}, číslo B: {1}", cisloA, cisloB);
Console.ReadKey();

```



(33) *Vytvoř konzolovou aplikaci papoušek.*



```
bool dalsilekce = true;
string str = "";
while (dalsilekce)
{
    Console.Clear();
    Console.WriteLine("Program papoušek s ukončení programu!");
    Console.Write("Zadej text pro papouška: ");
    str = str + Console.ReadLine() + " ";
    Console.WriteLine("Papoušek říká: " + str);
    Console.WriteLine("Přejete si papouška dále učit a/n!");

    bool konecProgramu = false;
    while (!konecProgramu)
    {
        switch (Console.ReadKey().KeyChar.ToString().ToLower())
        {
            case "a":
                konecProgramu = true;
                dalsilekce = true;
                break;
            case "n":
                konecProgramu = true;
                dalsilekce = false;
                break;
            default:
                Console.WriteLine("Neplatná volba, zadejte prosím [a/n].");
                break;
        }
    }
    Console.ReadKey();
}
```

9.3 Seznam použité literatury

- [1] ELLER, Frank. *C# - začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2002, 240 s. ISBN 80-247-0324-6.
- [2] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2008, 592 s. ISBN 978-80-251-2027-9.

10 METODICKÝ LIST – ČÍSLO 10



CÍLEM TÉMATU – je čtenáře uvést do problematiky programování grafických aplikací za pomoci objektů a realizovat základní program



KLÍČOVÁ SLOVA – toolbox, vlastnost, Windows Forms Application



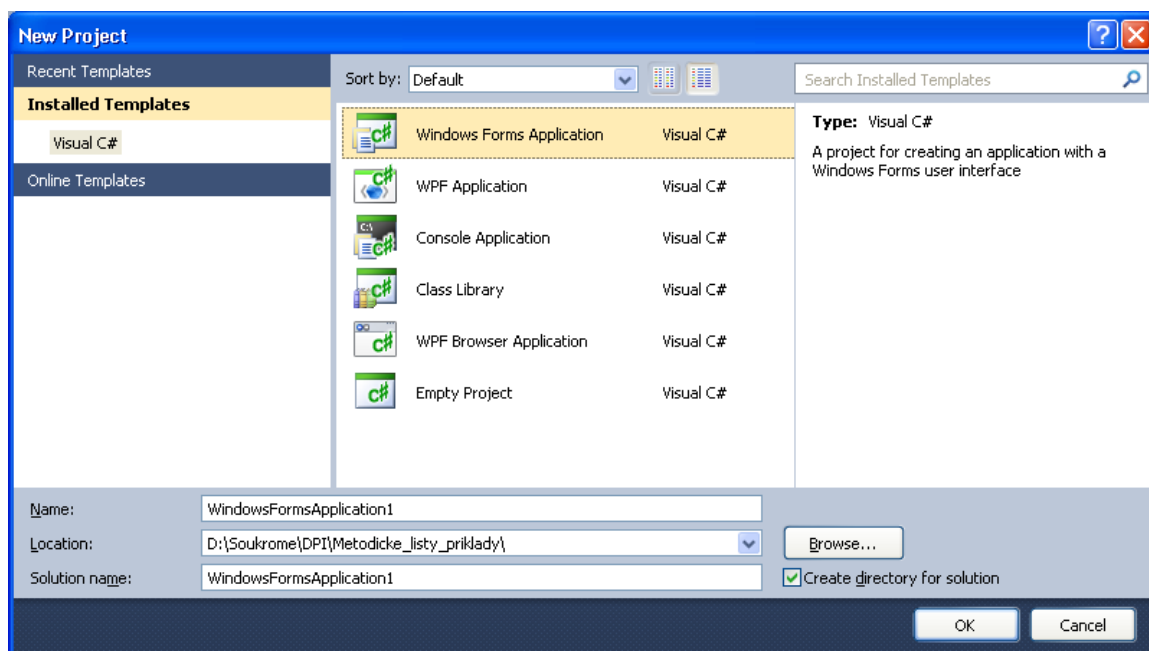
ČASOVÁ NÁROČNOST – tohoto tématu je stanovena na 90 minut.

10.1 Windows Forms Applications

Až doposud čtenář/programátor-beginner ve vývojovém prostředí Microsoft Visual C# 2010 Express vytvářel a spouštěl jen jednoduchou konzolovou aplikaci. Vývojové prostředí, ale obsahuje také vše potřebné k tomu, aby v něm mohl vytvářet grafické aplikace pro Windows. Formulářové uživatelské rozhraní aplikace pro Windows lze navrhnout interaktivně. Microsoft Visual C# 2010 Express poté vygeneruje programové příkazy, které implementují navržené uživatelské rozhraní.

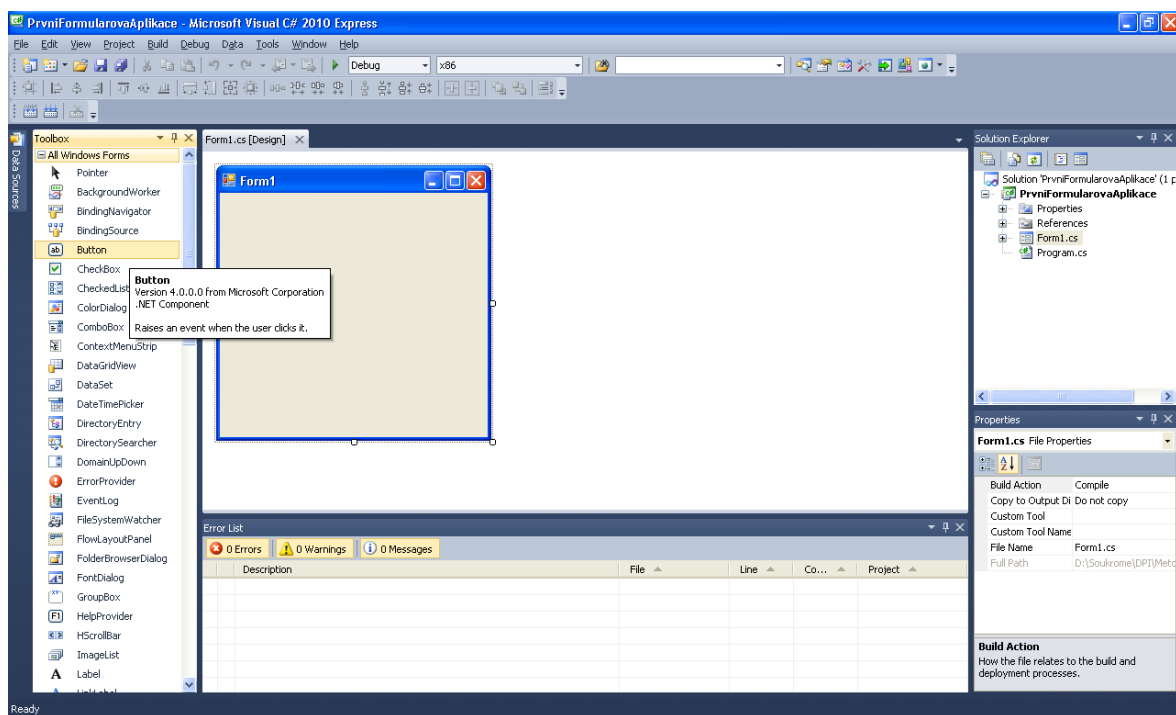
10.1.1 Založení projektu

Založení formulářové aplikace se provádí podobným postupem, jako založení konzolové aplikace. Jen při výběru nyní musí čtenář zvolit první nabízenou položku a to Windows Forms Application.



Obr. 28. Zakládání aplikace Windows Forms Application

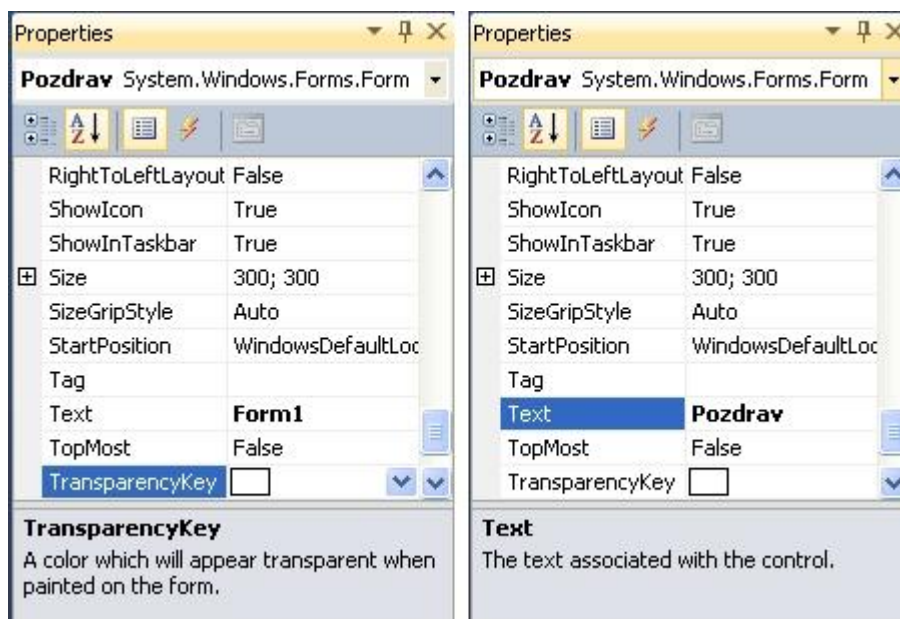
Microsoft Visual C# 2010 Express nabízí dva možné pohledy na grafické aplikace: *návrhové zobrazení* a *zobrazení kódu*. Je možné použít okno editoru k modifikaci a údržbě zdrojového kódu a logiky pro grafickou aplikaci. Pomocí okna designéru (návrháře) je možné vytvářet uživatelské rozhraní. Mezi těmito dvěma pohledy můžete libovolně přepínat. Vývojové prostředí vygeneruje šablonu uvedenou níže pro vytváření grafické aplikace. Povšimněte si, že levý panel Toolbox nyní obsahuje možné objekty pro vytváření uživatelského rozhraní.




10.1.2 Vytvoření jednoduché grafické aplikace

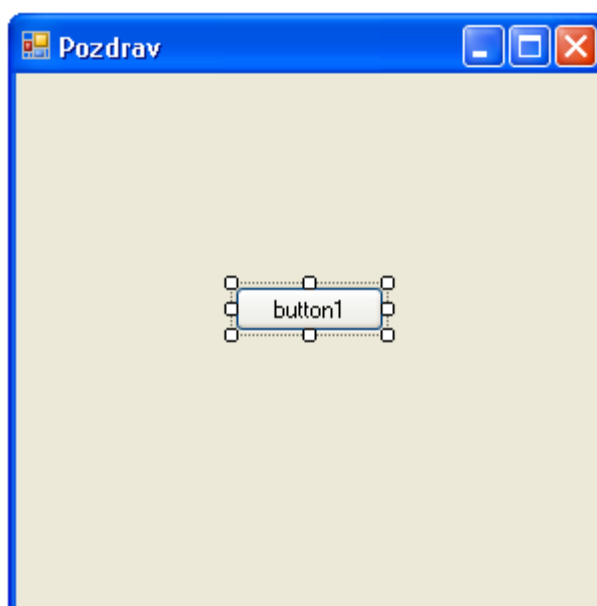
Nyní si vytvoříme v několika jednoduchých krocích naši oblíbenou zdravící aplikaci světa. Budeme k tomuto potřebovat objekt nazvaný *Button*.

1. První co je nutné udělat je kliknout na šablonu uživatelského rozhraní *Form1*, které mu změním název okna na „Pozdrav“.
 - a. Po kliknutí se Vám okno vlastnostmi změní a zobrazí veškeré vlastnosti formuláře. Najděte pole „Text“ a změňte jeho text *Form1* na *Pozdrav*.



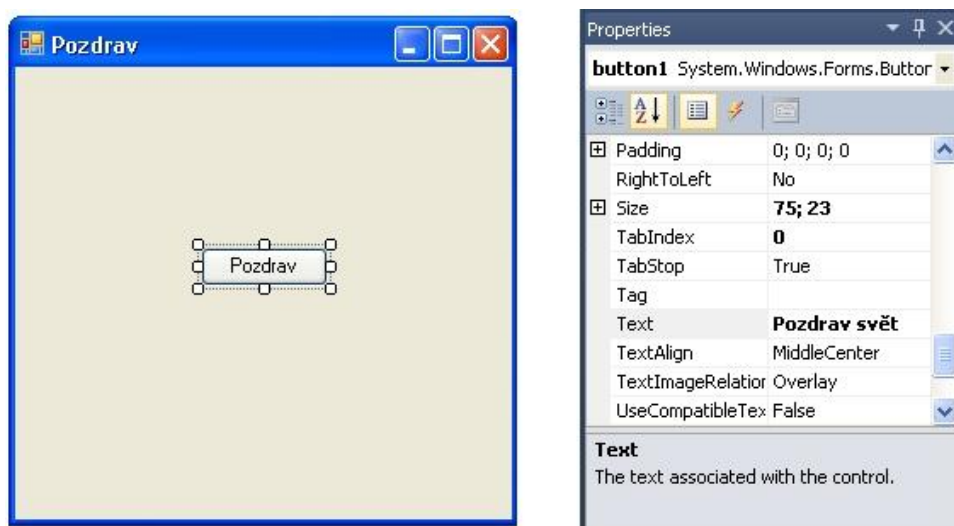
Obr. 29. Pole Text vlevo před změnou, vpravo po změně

- b. Druhou vlastností formuláře, kterou změníme je v poli „Start Position“ změna vlastnosti z *WindowsDefaultLocation* na *CenterScreen*, proto aby se nám uživatelské okno formuláře vyvolalo přesně ve středu obrazovky.
2. Nyní do formuláře přidáme prvek tlačítka. V toolbaxu je označeno tlačítko ikonou  Button. Tento objekt pomocí myši uchopíte a přenesete někam do formuláře. Anglicky se toto chování označuje D&D (drag and drop) česky „táhni-a-pust“.



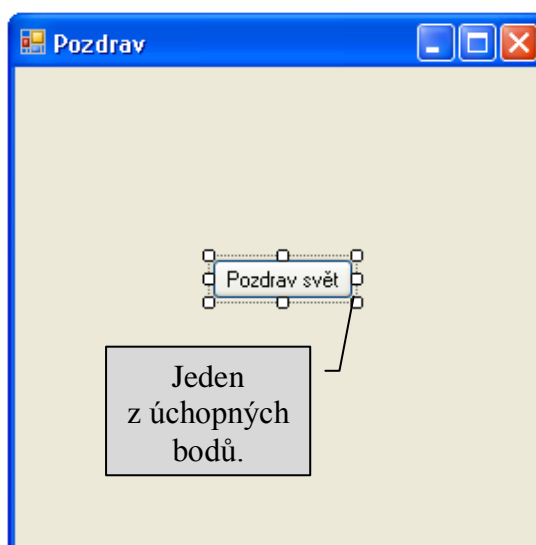
Obr. 30. Formulář po přidání prvku tlačítka

3. Změna popisku tlačítka z button1 na Pozdrav svět. Toto nastavení se opět provádí v sekci vlastností. Je podmínkou, že pokud chcete měnit vlastnost objektu, je nutné mít tento objekt označen/vybrán.
- a. Tuto změnu provedete v poli „Text“ a změníte původní hodnotu textu *button1* na *Pozdrav svět*.




Obr. 31. Nastavení vlastnosti Text na hodnotu Pozdrav svět

- b. Jak jde vidět tlačítko na tak velký textový řetězec je male proto je nutné je zvětšit. Roztažení okna je možné pomocí jednoho z úchopných bodů jak znázorňuje následující obrázek. Úchopný bod je malý bod, většinou na okrajích grafického objektu.



Obr. 32. Roztažení tlačítka pomocí
úchopného bodu

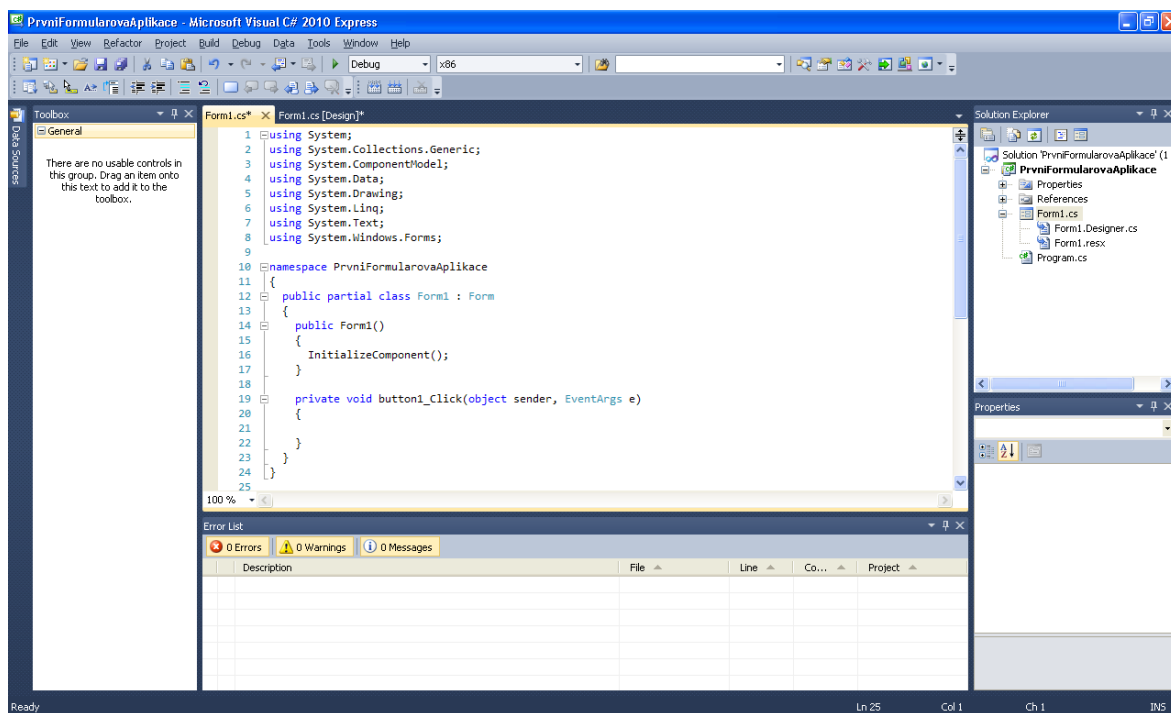
4. Nyní ve formuláři dvojklikem poklepejte na tlačítko. Dostanete se do události tlačítka. Microsoft Visual C# 2010 Express Vám vygeneruje následný zdrojový kód.



```
namespace PrvniFormularovaAplikace
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
        }
    }
}
```

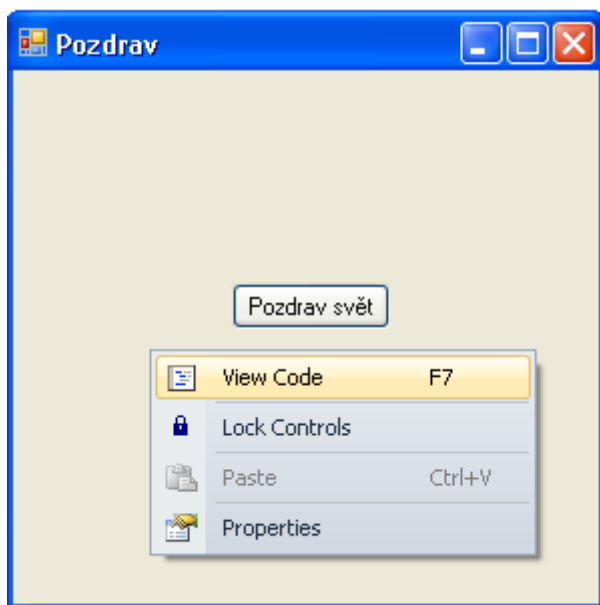
- a. A přejde z návrhového zobrazení do zobrazení kódu, jak ukazuje následný obrázek



Obr. 33. Přechod z návrhového zobrazení do zobrazení kódu




TIP – Zdrojový kód C# daného formuláře můžete zobrazit také tak, že klepnete pravým tlačítkem myši kamkoli v okně návrháře a poté zvolíte položku menu View Code.



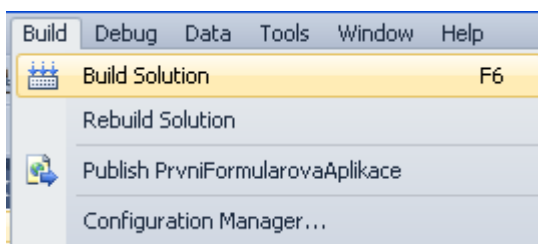
Obr. 34. Vyvolání zdrojového kódu C#

5. Do procedury `button1_Click` vložíte následující kód:



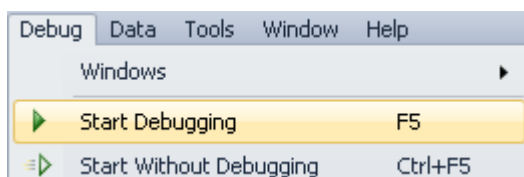
```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Ahoj
světe", "Pozdravuji...", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

6. V nabídce menu *Build* klepněte na příkaz *Build Solution* a ověřte si, že se projekt úspěšně sestaví.



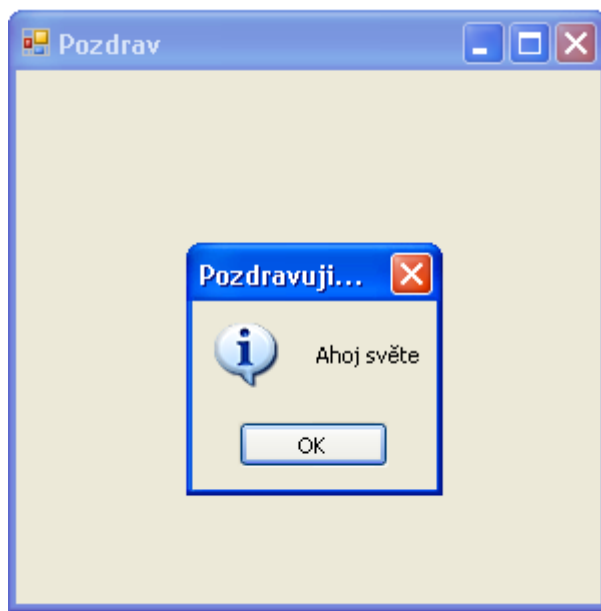
Obr. 35. Překlad programu

7. V nabídce *Debug* klepněte na příkaz *Start Debugging*. A ověřte zda se program spustil. Grafická aplikace by se měla spustit a zobrazit vytvořený formulář s tlačítkem.



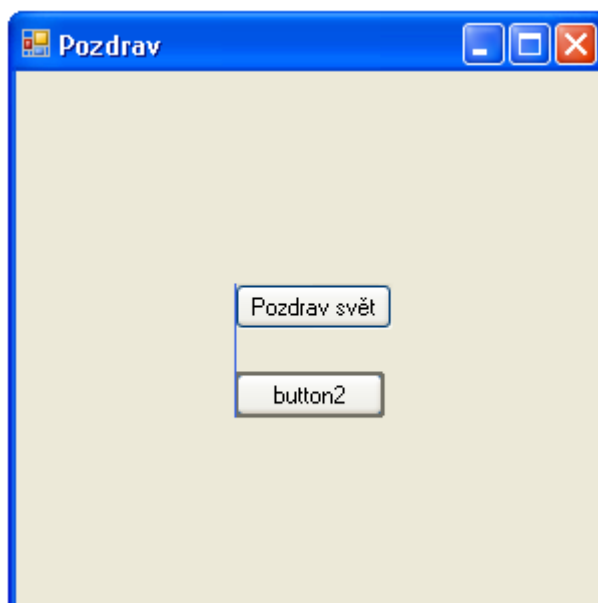
Obr. 36. Spuštění programu

8. Klikněte na tlačítko a ověřte událost tlačítka, zda program vyvolá dialogové okno s pozdravem a ikonou.



Obr. 37. Ověření funkcionality tlačítka


9. Doplníme tento formulář ještě o jedno tlačítko s metodou na ukončení formuláře `Close()`.
 - a. Přidáme tlačítko a nastavíme mu vlastnost `Text = Konec`.



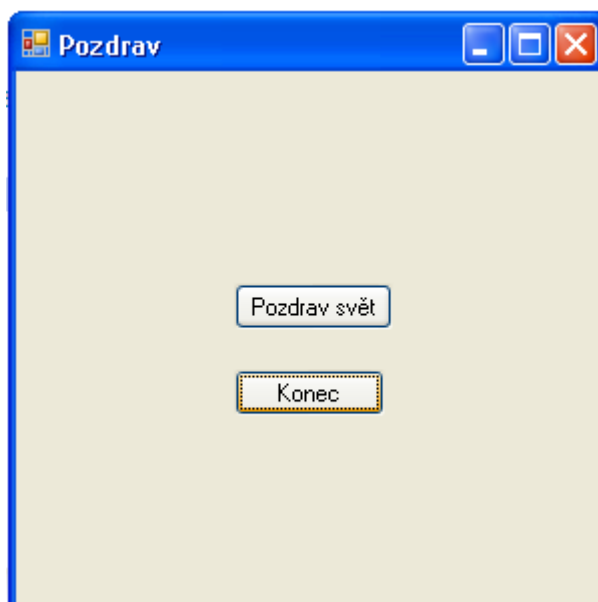
Obr. 38. Vodící linka

Všimněte si, že designér má jednu vhodnou vlastnost a to tu, že při přidání dalšího prvku automaticky ukazuje vodící linky (modrá tenká čára) a my přesně umístíme jednotlivé prvky.

- b. Dvojklikem na tlačítko založíme událost tomuto tlačítku a přidáme do něj metodu `Close()`.

```
 private void button2_Click(object sender, EventArgs e)
{
    Close();
}
```

10. Program nyní sestavíme a spustíme. Po kliknutí na tlačítko „Konec“ by se měl formulář uzavřít.



Obr. 39. Formulář s tlačítkem Konec

10.1.3 Práce s okny se zprávou

Okna se zprávou slouží pro výpis informací a vyhodnocení odpovědi uživatele. Jedná se o malá okna, která obsahují téměř vždy titulek okna, text zprávy, tlačítko(a) případně ikonu. Lze je použít jako informační okno před varováním na špatný úkon uživatele nebo pro hlášení chyby, která nastala. Pro výpis okna se zprávou je použita metoda `Show()` třídy `MessageBox`.

Parametry metody `Show()`:

- Titulek okna zprávy

- Text v okně zprávy
- Tlačítka v okně – `MessageBoxButtons.OK`
- Ikony v okně - `MessageBoxIcon.Information`
- Předvolení tlačítko okna




```
MessageBox.Show("Text okna", "Titulek okna", MessageBoxButtons.OK,  
MessageBoxIcon.Information);
```

10.2 Otázky z probraného tématu

- ❖ Jak založíš grafickou aplikaci?
- ❖ Co je to drag and drop funkcionality?
- ❖ V kterém panelu vývojového prostředí Microsoft Visual C# 2010 Express lze měnit vlastnosti prvku?
- ❖ Jak založíš událost tlačítka? Jak se dostaneš do kódového zobrazení?

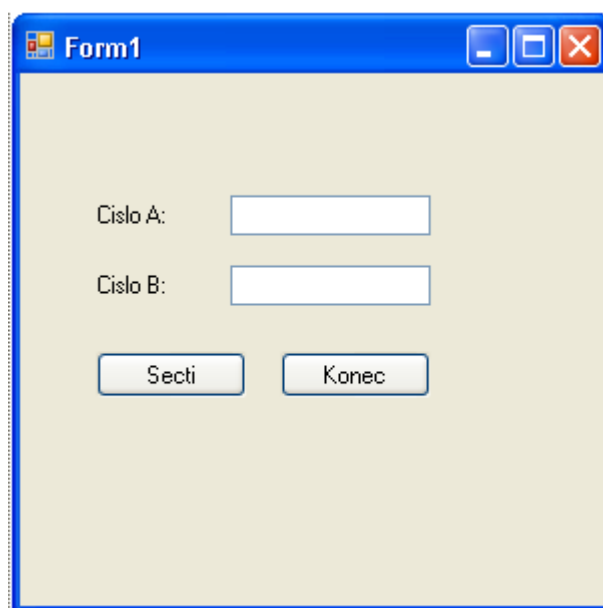
10.3 Praktické cvičení na dané téma

- (34) Ověř si jaký je rozdíl mezi formulářem v designéru a formulářem spuštěné aplikace.
- (35) Založ novou grafickou aplikaci a projdi si jednotlivé prvky v panelu Toolbox. Ověř si, že vkládání prvku z panelu Toolbox lze jen v grafickém zobrazení nikoliv v kódovém zobrazení.
- (36) Založ si grafickou aplikaci. Vlož do formuláře dva prvky tlačítka a jeden textového pole  `TextBox`. Pokus se vytvořit aplikaci, která ti bude vypisovat do dialogového okna `MessageBox` text vložený v textovém poli. Využij k tomuto vlastnost textového pole `Text()`. Program ukonči tlačítkem Konec.
- (37) Založ si grafickou aplikaci. Aplikace bude mít za úkol vypsatí všech zpráv `MessageBoxu` spolu s jejich ikonami.
- (38) Založ si grafickou aplikaci. Aplikace bude vyhodnocovat výsledek okna se zprávou `MessageBoxButtons.YesNo`. Pokud bude vybrána volba Ano, program bude ukončen. Pro vyhodnocení využij metody výčtové hodnoty `DialogResult`.



```
DialogResult konecProgramu;
konecProgramu = MessageBox.Show("// Doplně kód");
if (konecProgramu == DialogResult.Yes)
{
    // Doplně kód
}
```

- (39) Založ si grafickou aplikaci. Využij zdrojový kód z bodu (38) a vytvoř grafickou aplikaci na součet dvou čísel. Nezapomeň na ošetření uživatelského vstupu pomocí Try a Catch(). Možný návrh aplikace:



Obr. 40. Návrh aplikace součtu

- (40) Konzolovou aplikaci z bodu (27) pro kontrolu správnosti zadaného hesla na aplikaci grafickou. Využij vlastnost PasswordChar u textboxu pro výpis znaků v textboxu tedy „*“ na místo písmen. Grafickou aplikaci vycentruj na střed obrazovky.

10.4 Řešení praktického cvičení

- (36) *Grafická aplikace výpisu vloženého textu v textboxu.*



```
private void button1_Click(object sender, EventArgs e)
{
    string strTextOkna = textBox1.Text;
    string strTextTitulkuOkna = "Zadaný text";
    MessageBox.Show(strTextOkna, strTextTitulkuOkna, MessageBoxButtons.OK, MessageBoxIcon.Information);
}


private void button2_Click(object sender, EventArgs e)
{
    Close();
}
```

(37) Vápis všech zpráv MessageBoxu spolu s jejich ikonami.

```

MessageBox.Show("Tlačítko Ok, ikona Asterisk", "Ok",
MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
MessageBox.Show("Tlačítko Přerušit Opakovat Přeskočit, ikona Error",
"Přerušit Opakovat Přeskočit", MessageBoxButtons.AbortRetryIgnore,
MessageBoxIcon.Error);
MessageBox.Show("Tlačítko OK Storno, ikona Exclamation", "OK Storno",
MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
MessageBox.Show("Tlačítko Opakovat Storno, ikona Hand", "Opakovat
Storno", MessageBoxButtons.RetryCancel, MessageBoxIcon.Hand);
MessageBox.Show("Tlačítko Ano Ne, ikona Information", "Information",
MessageBoxButtons.YesNo, MessageBoxIcon.Information);
MessageBox.Show("Tlačítko Ano Ne Storno, ikona None", "Ano Ne Storno",
MessageBoxButtons.YesNoCancel, MessageBoxIcon.None);
MessageBox.Show("Tlačítko Ano Ne Storno, ikona Question", "Ano Ne
Storno", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);
MessageBox.Show("Tlačítko OK, ikona Stop", "OK", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
MessageBox.Show("Tlačítko OK, ikona Stop", "OK", MessageBoxButtons.OK,
MessageBoxIcon.Warning);


```

(38) Grafickou aplikaci na ukončení programu.


```

private void button1_Click(object sender, EventArgs e)
{
    DialogResult konecProgramu;
    konecProgramu = MessageBox.Show("Chcete ukončit program", "Konec",
    MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (konecProgramu == DialogResult.Yes)
    {
        Close();
    }
}

```

(39) Grafickou aplikaci součet dvou čísel.


```

private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int cisloA = int.Parse(textBox1.Text);
        int cisloB = int.Parse(textBox2.Text);
        int vysledek = cisloA + cisloB;
        string str = Convert.ToString(vysledek);
        MessageBox.Show("Výsledkem součtu je: " + str, "Vysledek",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

private void button2_Click(object sender, EventArgs e)
{

```



```

    DialogResult konecProgramu;
    konecProgramu = MessageBox.Show("Chcete ukončit program", "Konec",
    MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (konecProgramu == DialogResult.Yes)
    {
        Close();
    }

```

(40) Grafická aplikace pro kontrolu správnosti zadaného hesla z bodu (27)

```

private void button1_Click(object sender, EventArgs e)
{
    string[] poleHesel = { "QWERTZ", "123", "Pass", "Heslo", "ZXC",
    "MNB", "PoLkMn" };
    string heslo = textBox1.Text.ToLower();
    int poziceHesla = Array.IndexOf(poleHesel, heslo);
    if (poziceHesla >= 0)
        MessageBox.Show("Vítej v programu.", "Vstup", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
    else
        MessageBox.Show("Špatně zadané heslo!", "Chyba",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```



```

private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    /* pokud je potřeba převést ihned hodnotu na odpovídající datový typ
    využívá se k tomuto kulatých závorek
    zde je hodnota 13 předena na datový typ char
    Dále program kontroluje zda byla stisknuta klávesa Enter, která má
    hodnotu 13,
    pokud enter byl zmáčknut vyvolá se událost kliknutí na tlačítko */
    if (e.KeyChar == (char)13)
    {
        // Vyvolání události na obsluhu tlačítka
        button1_Click((object) sender, (EventArgs) e);
    }
}

```

10.5 Seznam použité literatury

- [1] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2008, 592 s. ISBN 978-80-251-2027-9.

ZÁVĚR

Čtenář pro prostudování všech deseti metodických listů by měl zvládnout navrhnout vývojové diagramy podle předem zadaného úkolu. Měl by mít dostatek základních znalostí pro ovládání vývojového prostředí Microsoft Visual C# 2010 Express, kde bude schopen zakládat projekty buď to konzolové aplikace, nebo aplikace grafické. Poté by měl čtenář zvládnout naprogramovat základní jednoduché příklady ve vývojovém prostředí Microsoft Visual C# 2010 Express.

Při vytváření metodických listů nebylo mým záměrem vytvoření materiálů, která by komplexně obsáhly celou problematiku, jelikož by taková publikace vydala na několik pěkně tlustých dílů knih. Spíše jsem se snažil vytvořit takový základní rychlokurz, kde si čtenář osvojí znalosti programování jak teoretického, tak praktického charakteru. Mou snahou bylo předat základní znalosti a dovednosti, které čtenáři pomohou zvládnout realizovat zadanou úlohu. Správně analyzovat její problematiku a podle této analýzy vhodně navrhnout a zrealizovat vývojový diagram. Poté jak by uživatel měl ověřeno, že daný vývojový diagram je natolik dostačující, aby vyřešil daný problém, měl by čtenář zvládnout tento vývojový diagram převést na zdrojový kód programovacího jazyka C# a řešení dokázat realizovat.

Tyto metodické listy by tedy měly sloužit jako jakýsi odrazový můstek a ve čtenáři vyvolat zvědavost a touhu ohledně problematiky algoritmizace a programování a odstartovat tak jeho další růst gramotnosti v oblasti celého odvětví informatiky.

Zvolenou metodou probírané látky u jednotlivých metodických listů očekávám, že čtenář získá potřebné informace o probrané látce. Na konci látky má vybrané kontrolní otázky na které by měl znát správné odpovědi. Pokud tyto kontrolní otázky čtenář nedokáže zodpovědět, doporučoval bych čtenáři znovu a pečlivěji prostudovat daný metodický list minimálně do té chvíle než bude znát správnou odpověď. Když čtenář bude znát správné odpovědi na dané kontrolní otázky, měl by postoupit do další fáze a to praktickým úkonům a řešením zadaných příkladů. Na konci každého metodického listu je uvedeno správné řešení zvoleného problému, kde si čtenář může ověřit, jestli postupoval správně nebo kde případně provedl nějakou chybu či jak se vlastně dala tato problematika řešit. Nikdo není neomylný a chybami se člověk učí.

V každém metodickém listu je uveden zdroj literatury odkud jsem informace ohledně dané problematiky čerpal. Pro hlubší informovanost doporučuji si tyto materiály

obstarat a vyhledávat v něm další užitečné informace, které se do probírané látky nedostaly.

Jak jsem již jednou řekl, nikdo není neomylný a proto ani já si netroufnu na sto procent tvrdit, že veškeré informace nebo příklady jsou správné a jinou správnou variantu řešení nemá. To bych byl velmi naivní. Při programování je možné jednotlivé programy napsat programátorsky jednodušeji, ale také je pravda že je možné je napsat programátorsky složitěji. Proto nechte uvolnit svou představivost a logické myšlení a hurá do správného řešení zadaných problémů.

SEZNAM POUŽITÉ LITERATURY

- [1] PŠENČÍKOVÁ, Jana. *Algoritmizace*. Vyd. 2. Kralice na Hané: Computer Media, 2009, 128 s. ISBN 978-80-7402-034-6.
- [2] *Jednotky v informačních technologiích* [online]. 2007 [cit. 2013-03-27]. Dostupné z: <http://home.zcu.cz/~wichs/index.html>
- [3] VIRIUS, Miroslav. *Základy algoritmizace*. Praha: ČVUT, 1995. ISBN 80-01-01346-4.
- [4] ČSN ISO 5807. *Zpracování informací: Dokumentační symboly a konvence pro vývojové diagramy toku dat, programu a systému, síťové diagramy programu a diagramy zdrojů systému*. Praha: TNK 20 Informační technika, 1996.
- [5] ČSN 36 9030. *Počítače a systémy zpracování dat: Symboly vývojových diagramů pro systémy zpracování dat*. Praha: VUAP Engineering A. S., 1974.
- [6] ČSN ISO 31-11. *Veličiny a jednotky: Část 11: Matematické znaky a značky používané ve fyzikálních vědách a v technice*. Praha: ADVIS, 1999.
- [7] KRESLÍKOVÁ, Jitka. *Základy programování*. Brno: Fakulta informačních technologií VUT v Brně, 2002.
- [8] ELLER, Frank. *C# - začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2002, 240 s. ISBN 80-247-0324-6.
- [9] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2008, 592 s. ISBN 978-80-251-2027-9.
- [10] VYSTAVĚL, Radek. *Moderní programování: pro začátečníky*. Ondřejov: moderníProgramování, 2007, 2 sv. ISBN 978-80-903951-1-4.
- [11] Programování se zaměřením na .NET a jazyk C#. *Aritmetické operátory* [online]. 2005 [cit. 2013-05-14]. Dostupné z: <http://projektysipvz.gytool.cz/ProjektySIPVZ/Default.aspx?uid=83>
- [12] Programování se zaměřením na .NET a jazyk C#. *Výjimky* [online]. 2006 [cit. 2013-05-15]. Dostupné z: <http://projektysipvz.gytool.cz/ProjektySIPVZ/Default.aspx?uid=262>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK



CÍL PROBÍRANÉ LÁTKY – každá sekce bude směřována s určitým záměrem daného tématu, které je zapotřebí dosáhnout pro další postup.



KLÍČOVÁ SLOVA – každé téma bude obsahovat na svém začátku, klíčové slova probíraná v tomto tématu



ČASOVÁ NÁROČNOST – každá probíraná látka bude označena symbolem určující průměrnou dobu, strávenou nad probíraným tématem.



POZNÁMKA – tento symbol je použit tam, kde je potřeba uvedený text více rozvinout a upřesnit. Není potřeba uváděný text nezbytně nutně znát.



TIP – metodická sekce takto označena znázorňuje informace, které je vhodné si zapamatovat a osvojit.



UPOZORNĚNÍ – touto ikonou jsou označovány sekce, které si zaslouží Vaši zvýšenou pozornost. Varují například před možnými překážkami, chybami a problémy v programu.



ZDROJOVÝ KÓD – sekce označená tímto symbolem znázorňuje zdrojový kód uvedený v probíraném tématu. **Neproporcionálním písmem** jsou označeny třídy a metody, které jsou poprvé uváděna v daném tématu.

CLS Common Language Specification

D&D Drag and Drop

SEZNAM OBRÁZKŮ

Obr. 1. Ukázka použití symbolu	15
Obr. 2. Symbol zobrazení	16
Obr. 3. – Symbol podprogramu	17
Obr. 4. Integrované prostředí vývojového prostředí Microsoft Visual C# 2010 Express	35
Obr. 5. Výběr typu projektu Console Application	36
Obr. 6. Editor kódu s barevným zvýrazněním syntaxe	37
Obr. 7. Změna nastavení vývojového prostředí	37
Obr. 8. Výběr panelů z menu View	38
Obr. 9. Zobrazení dialogového okna možností	39
Obr. 10. Nastavení - Show advanced build configurations	39
Obr. 11. Nastavení číslování řádků	40
Obr. 12. Nastavení odsazení na hodnotu dvě	40
Obr. 13. Nově nastavené vývojového prostředí	40
Obr. 14. Založení nového projektu	44
Obr. 15. Řešení prvního projektu	44
Obr. 16. Vygenerovaná kostra zdrojového kódu C#	45
Obr. 17. Zápis příkazu Console.WriteLine()	46
Obr. 18. Ukázka intellisense u metody WriteLine()	46
Obr. 19. Zdrojový kód k prvnímu programu	47
Obr. 20. Konzolová aplikace s prvním programem	48
Obr. 21. Příklad s ScreenTipem	53
Obr. 22. Ukázka proměnné nabídnuté intellisense	54
Obr. 23. Zvýraznění proměnné bez přiřazené hodnoty ve vývojovém prostředí	55
Obr. 24. Položka menu pro přidání reference	78
Obr. 25. Výběr reference System.Windows.Form	79
Obr. 26. Konzolová aplikace bez direktivy System.Windows.Forms	79
Obr. 27. Hierarchie Exception třídy	80
Obr. 28. Zakládání aplikace Windows Forms Application	94
Obr. 29. Pole Text vlevo před změnou, vpravo po změně	96
Obr. 30. Formulář po přidání prvku tlačítka	96
Obr. 31. Nastavení vlastnosti Text na hodnotu Pozdrav svět	97

Obr. 32. Roztažení tlačítka pomocí úchopného bodu	97
Obr. 33. Přejít z návrhového zobrazení do zobrazení kódu	98
Obr. 34. Vyvolání zdrojového kódu C#	99
Obr. 35. Překlad programu	99
Obr. 36. Spuštění programu	99
Obr. 37. Ověření funkcionality tlačítka	100
Obr. 38. Vodič linka	100
Obr. 39. Formulář s tlačítkem Konec	101
Obr. 40. Návrh aplikace součtu.....	103

SEZNAM TABULEK

<i>Tab. 1. Mezní značky – začátku a konce algoritmu.....</i>	<i>13</i>
<i>Tab. 2. Sekvenční bloky – vstup a výstup dat.....</i>	<i>14</i>
<i>Tab. 3. Sekvenční bloky – zpracování</i>	<i>14</i>
<i>Tab. 4. Symbol větvení.....</i>	<i>15</i>
<i>Tab. 5. Interní paměť – vstup a výstup dat</i>	<i>16</i>
<i>Tab. 6. Symboly spojky</i>	<i>16</i>
<i>Tab. 7. Symboly spojnic.....</i>	<i>17</i>
<i>Tab. 8. Tabulka ikon intellisense.....</i>	<i>48</i>
<i>Tab. 9. Espace sequence</i>	<i>50</i>
<i>Tab. 10. Tabulka celočíselných primitivních datových typů.....</i>	<i>56</i>
<i>Tab. 11. Tabulka reálných datových typů a typu decimal</i>	<i>56</i>
<i>Tab. 12. Použití logických operátorů</i>	<i>63</i>
<i>Tab. 13. Užitečné logické operátory</i>	<i>64</i>
<i>Tab. 14. Operátory složeného přiřazení</i>	<i>68</i>
<i>Tab. 15. Ukázka prvků pole a jejich indexů.....</i>	<i>82</i>

SEZNAM PŘÍLOH

P I: METODICKE_LISTY_PRIKLADY