ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA STROJNÍ

Studijní program: B 2301 Strojní inženýrství
Studijní zaměření: Průmyslové inženýrství a management

BAKALÁŘSKÁ PRÁCE

Tvorba podkladů pro inovaci výuky Technické Informatiky v C# jazyce

Autor: **Michael Pompl**

Vedoucí práce: **Ing. Petr Hořejší, Ph.D.**

Akademický rok 2017/2018

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta strojní
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michael POMPL**
Osobní číslo:	**S15B0338P**
Studijní program:	**B2301 Strojní inženýrství**
Studijní obor:	**Průmyslové inženýrství a management**
Název tématu:	**Tvorba podkladů pro inovaci výuky Technické Informatiky v C# jazyce**
Zadávající katedra:	**Katedra průmyslového inženýrství a managementu**

Zásady pro vypracování:

1. Úvod
2. Analýza současného stavu
3. Tvorba podkladů
4. Závěr

Rozsah grafických prací: **0 výkresů**

Rozsah kvalifikační práce: **30 - 40 stran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. **MILES R. C#** *Programming Yellow Book.* **"Cheese" Edition, 2016. ISBN-13: 978-1509301157**
2. **KLAUSEN P.** *Introduction to programming and the C# language.* **Poul Klausen & bookboon.com, 2012. ISBN-13: 978-1509301157**
3. **SHANU S.** *Beginning C# Object Oriented Programming.* **C# Corner, 2016.**
4. *https://www.w3resource.com/csharp-exercises/,* **dostupné 29.10.2017**
5. *http://www.sanfoundry.com/csharp-programming-examples-on-mathematics/,* **dostupné 28.10.2017**

Vedoucí bakalářské práce: **Ing. Petr Hořejší, Ph.D.**
Katedra průmyslového inženýrství a managementu

Konzultant bakalářské práce: **Ing. Pavel Raška, Ph.D.**
Katedra průmyslového inženýrství a managementu

Datum zadání bakalářské práce: **20. září 2017**

Termín odevzdání bakalářské práce: **21. května 2018**

Doc. Ing. Milan Edl, Ph.D.
děkan

L.S.

Doc. Ing. Michal Šimon, Ph.D.
vedoucí katedry

V Plzni dne 20. září 2017

Poděkování

Při této příležitosti bych chtěl poděkovat Ing. Petru Hořejšímu, Ph.D., vedoucímu mé bakalářské práce, za trpělivost, cenné připomínky a rady, které mi v průběhu práce ochotně poskytl. Rovněž bych chtěl poděkovat všem, kteří mi radou a cennými zkušenostmi umožnili dokončit tuto práci.

Prohlášení o autorství

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě strojní Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů, uvedených v seznamu, který je součástí této bakalářské/diplomové práce.

V Plzni dne: …….. . . . . .

podpis autora

ANOTAČNÍ LIST BAKALÁŘSKÉ PRÁCE

AUTOR	**Příjmení** Pompl	**Jméno** Michael	
STUDIJNÍ OBOR	B2301 „Průmyslové inženýrství a management“		
VEDOUCÍ PRÁCE	**Příjmení (včetně titulů)** Ing. Hořejší Ph.D.	**Jméno** Petr	
PRACOVIŠTĚ	ZČU - FST - KPV		
DRUH PRÁCE	~~**DIPLOMOVÁ**~~	**BAKALÁŘSKÁ**	**Nehodící se škrtněte**
NÁZEV PRÁCE	Tvorba podkladů pro inovaci výuky Technické Informatiky v C# jazyce		

FAKULTA	strojní		**KATEDRA**	KPV		**ROK ODEVZD.**	2018

POČET STRAN (A4 a ekvivalentů A4)

CELKEM	104		**TEXTOVÁ ČÁST**	101		**GRAFICKÁ ČÁST**	3

STRUČNÝ POPIS (MAX 10 ŘÁDEK) **ZAMĚŘENÍ, TÉMA, CÍL POZNATKY A PŘÍNOSY**	Zaměření BP na tvorbu podkladů k výuce technické informatiky vztažené k přednáškám. Bylo zkoumáno několik zdrojů. Pomocí rozhodovací analýzy provedené Saatyho maticí byl stanoven nejvhodnější základ k tvorbě podkladů. Na základě výsledku z analýzy byla postupně tvořena osnova. K jednotlivým tématům z osnovy byl tvořen obsah na základě zkoumaných zdrojů. Záměrem bylo zakomponovat do obsahu podkladů tématiku strojírenství. Posledním výstupem práce je krátká vzorová prezentace vztahující se k jednomu z témat v osnově dokumentu.
KLÍČOVÁ SLOVA **ZPRAVIDLA JEDNOSLOVNÉ POJMY, KTERÉ VYSTIHUJÍ PODSTATU PRÁCE**	Programování, jazyk, C#, rozhodování, analýza, informatika, podklady, inovace

SUMMARY OF BACHELOR SHEET

AUTHOR	**Surname** Pompl	**Name** Michael	
FIELD OF STUDY	B2301 "Industrial Engineering and Management"		
SUPERVISOR	**Surname (Inclusive of Degrees)** Ing. Hořejší Ph.D.	**Name** Petr	
INSTITUTION	ZČU - FST - KPV		
TYPE OF WORK	~~**DIPLOMA**~~	**BACHELOR**	**Delete when not applicable**
TITLE OF THE WORK	Creation of materials for innovation in Technical IT classes using C# language		

FACULTY	Mechanical Engineering		**DEPARTMENT**	Industrial engineering and management		**SUBMITTED IN**	2018

NUMBER OF PAGES (A4 and eq. A4)

TOTALLY	104		**TEXT PART**	101		**GRAPHICAL PART**	3

BRIEF DESCRIPTION **TOPIC, GOAL, RESULTS AND CONTRIBUTIONS**	The main goal of bachelor thesis is the creation of a support documentation for technical informatics. The documentation is based on TI lectures. There was a research of several resources. With the useage of decision analysis (Saaty´s matrix) the most suitable option for creating documentation was decided. The sylablus of documentation was created based on the result of analysis. A contet was created for each of the topic included in the sylablus. Excercises with engineering background were inserted into the documentation. The last outcome of the thesis is a short presentation which has a contet about one topic from created document.
KEY WORDS	Programming, language, C#, decision, analysis, informatics, documentation, inovation

Obsah

Seznam použitých zkratek 10
1 Úvod do problematiky 11
2 Cíl BP 12
3 Rešerše dostupných pramenů 13
3.1 Pramen č.1 − C# Programming Yellow Book, Rob Miles 13
3.2 Pramen č.2 − C# 1 – Introduction to programming and the C# language 15
3.3 Pramen č.3 − Beginning C# Object Oriented Programming, Syed Shanu 17
3.4 Pramen č.4 – 1. webový odkaz − w3resource.com 18
3.5 Pramen č.5 − 2. webová stránka − sanfoundry.com 18
3.6 Pramen č.6 − Kurz edX − Introduction to C# 19
3.7 Dodatečně využitý pramen 20
4 Rozhodovací analýza 20
4.1 Saatyho metoda 22
4.2 Metoda "strom kritérií" 23
4.3 Rozhodovací postup autora 23
5 Analýza výsledků 29
5.1 Porovnání metod 30
5.2 Porovnání přístupů studenta a kantorů 31
6 Sestavení osnovy 31
6.1 Tvorba jednotlivých témat osnovy 36
7 Hrubé podklady k výuce 38
7.1 Cykly 38
7.1.1 DO – WHILE cyklus 38
7.1.2 WHILE cyklus 39
7.1.3 FOR cyklus 40
7.1.4 Vnořené cykly 41
7.1.5 Únik z cyklů 42
7.1.6 Vracení se na začátek cyklu 43
7.2 Styl autorovy tvorby podkladů 43
8 Tvorba předběžné formy prezentace dokumentu 44
8.1 Změny provedené v prezentaci 44
9 Závěr 47
10 Bibliografie 48

Seznam příloh

Příloha 1 – Hrubý podpůrný dokument k výuce technické informatiky v programovacím jazyce C#

Příloha 2 – Krátká vzorová prezentace vztažená k části obsahu vytvořeného dokumentu

Příloha 3 – CD – Excel soubor s tabulkami

Seznam obrázků

Obr. 3-1 − ukázka stylu výkladu z prvního pramene [7] 13
Obr. 3-2 − ukázka stylu výkladu druhého pramene [4] 15
Obr. 3-3 − ukázka stylu výkladu třetího pramene [9] 17
Obr. 3-4 − ukázka výkladu látky čtvrtého pramene [11] 18
Obr. 3-5 − ukázka výkladu pátého pramene [8] 19
Obr. 3-6 − ukázka výkladu šestého pramene [2] 19
Obr. 4-1 − vysvětlení škály hodnocení kritérií [1] 22
Obr. 5-1 − grafické znázornění výsledných vah kritérií 30
Obr. 6-1 − dosavadní okruhy výuky TI [5] 32
Obr. 6-2 − hrubá předběžná osnova dokumentu 33
Obr. 6-3 − předběžná osnova dokumentu 34
Obr. 6-4 − finální osnova dokumentu 35
Obr. 7-1 − příklad na cyklus FOR [11] 40
Obr. 7-2 − příklad vnořeného cyklu FOR [2] 42
Obr. 8-1 − vizualizace stylu předběžné prezentace 44
Obr. 8-2 − přepsání cyklu "repeat-until" 45
Obr. 8-3 − změna zápisu cyklu "for" 45
Obr. 8-4 − změny v diagramu cyklu 46
Obr. 8-5 − rozdíl mezi kódy C# a Delphi vztažených k diagramu 46

Seznam tabulek

Tab. č. 4-1 − ukázka jednoduchého příkladu Saatyho matice [1] 23
Tab. č. 4-2 − ukázka metody "strom kritérií" [1] 23
Tab. č. 4-3 − vizualizace podoby první tabulky dle autora 24
Tab. č. 4-4 − vizualizace podoby druhé tabulky dle autora 24
Tab. č. 4-5 − vizualizace podoby třetí tabulky dle autora 25
Tab. č. 4-7 − sestupné seřazení některých vah kritérií z tabulky č.4 26
Tab. č. 4-6 − vizualizace podoby čtvrté tabulky dle autora 26
Tab. č. 4-8 − vizualizace podoby páté tabulky dle autora 27
Tab. č. 4-9 − naznačení postupu metodou "stromu kritérií" pro skupinu S1 28
Tab. č. 4-10 − sestupné seřazení některých vah kritérií z tabulky č.6 28
Tab. č. 4-11 − vizualizace podoby sedmé tabulky dle autora 29
Tab. č. 5-1 − tabulka získaných dat od kantorů s výslednými vahami kritérií 31

Seznam použitých zkratek

Zkratka	*Název*
ZČU	Západočeská univerzita v Plzni
FST	Fakulta strojní
KPV	Katedra průmyslového inženýrství a managementu
BP	Bakalářská práce
TI	Technická informatika
OOP	Objektově orientované programování
např.	například
Tab.	tabulka
Obr.	obrázek
Vs.	versus

1 Úvod do problematiky

V dnešní době se na FST ZČU vyučuje předmět TI – Technická informatika. Náplní předmětu je naučit studenty základy programování. Specificky se jedná o důležité pojmy, algoritmy, syntaxe, přehled programovacích jazyků, postup při tvorbě programů, základ procedurálního programovacího jazyka, datové typy ukazatele, cykly, rekurze a další.

Technologické odvětví jak strojírenské, tak i programátorské se neustále v dnešní době vyvíjí. Z tohoto důvodu se také objevují nové a nové způsoby, jak programovat. Je zde pozorovatelný vývoj nových programovacích jazyků a spolu s nimi nové cesty, jak si práci (programování) usnadnit.

Z tohoto důvodu je třeba i přizpůsobit po určitém čase výuku, a to hlavně na univerzitách. Inovacemi výuky se univerzita stává perspektivnější a více lidí se zájmem o aktuální technologické záležitosti si může z tohoto důvodu aktuálnosti vybrat právě Vaši katedru, či z pohledu většího měřítka fakultu.

V této práci bude popisován především programovací jazyk C#. Tento jazyk spadá do jednodušší náročnosti pochopení programovacího jazyka. Jazyk je moderní, objektově orientovaný, robustní a programově produktivní. V dnešní době se jazyk stal mezinárodně využívaný hlavně díky společnosti Microsoft. Z těchto důvodů byl pravděpodobně shledán kantory jako vhodná možná náhrada doposud využívaného programovacího jazyka Delphi.

Využitím několika dokumentů a výukových osnov ze zahraničních univerzit bude v popředí této práce snaha pomocí rozhodovací analýzy určit nejlepší možnost, jak osnovu pro tento předmět sestavit. Využito bude několik kritérií pro ohodnocení dokumentů. Autor si nadále určuje váhu jednotlivých kritérií, z čehož vyplývá subjektivnost tohoto rozhodování. Z výsledků rozhodovací analýzy se nadále bude odvíjet sestava osnovy.

2 Cíl BP

Cílem této práce je nastínění možné nové osnovy předmětu TI. Cílem také je, aby se předmět stal pro studenty zajímavým, ale zároveň aby obtížnost výkladu nebyla příliš náročná pro daný ročník, ve kterém se předmět vyučuje.

Provedením rešerše současného stavu výuky předmětu TI bude posouzena dosavadní osnova předmětu. Využitím a prostudováním volně dostupných zdrojů bude pomocí rozhodovací analýzy stanoven základ pro vytvoření nové osnovy předmětu v programovacím jazyce C#. V analýze budou k ohodnocení jednotlivých zdrojů zdroje porovnávány dle určitých kritérií. Bude stanovena důležitost těchto kritérií z pohledu studenta i z pohledu některých kantorů z katedry KPV.

V návaznosti na výsledky obdržené z rozhodovací analýzy bude sestavena hrubá osnova a následně hrubý obsah podkladů k výuce předmětu TI. Inspirace ke stavbě struktury podkladů bude brána převážně z porovnávaných dokumentů. Struktura bude obsahovat důležitá témata, která jsou již součástí struktury nynější, avšak celkově bude přizpůsobena novému programovacímu jazyku využitého v BP. Nadále se zde objeví i části nové, které ještě v současném stavu obsaženy nejsou, a to především z důvodu omezení využívaného programovacího jazyka. Dalším "zpestřením" nové struktury budou příklady se zaměřením na strojírenství nebo témata s ním související. Budou vybrána především témata nezbytně nutná k pochopení podstaty předmětu.

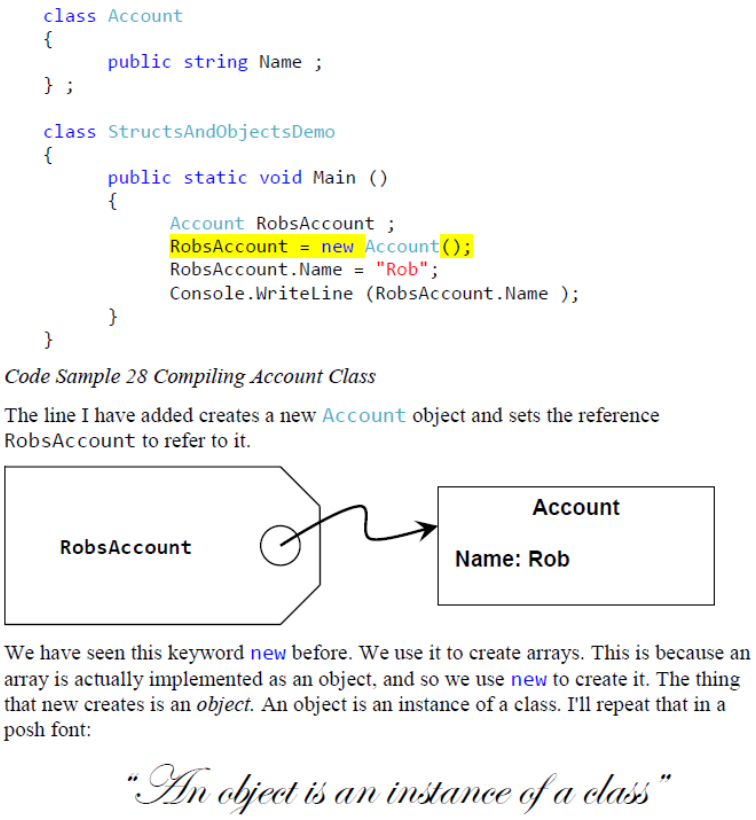
Práce bude zakončena prezentací, která nastíní, jakým způsobem by podklady mohly být prezentovány žákům ve výuce předmětu TI. Prezentace autorem vytvořená bude vycházet ze současného konceptu přednášených prezentací na hodinách TI. V autorem vytvořené prezentaci bude obsaženo pouze jedno z témat vytvořeného nového dokumentu. Je tak učiněno na základě domluvy s vedoucím bakalářské práce. V prezentaci by měly být využity prostředky k zachycení pozornosti. Cílem prezentace by mělo být nastínit studentům, že programování sice není lehké k pochopení, ale zároveň může vzbudit určitý zájem o toto odvětví. Může tak být docíleno například využitím moderního trendu vtipných cílených obrázků. Záměrem použití tohoto prostředku je přiblížení vykládané látky i studentům, kteří jeví velmi malý zájem se jakkoli v tomto odvětví nadále vzdělávat.

3 Rešerše dostupných pramenů

K provedení porovnávací analýzy bylo nutností se seznámit s několika dokumenty. Dokumenty a látka, kterou obsahují, byly vybírány tak, aby odpovídaly náročnosti pro daný předmět v daném semestru. Autor vybral celkem šest těchto zdrojů pro dostatečnou rozmanitost. Byly vybrány dvě učebnice využívané k výuce na zahraničních univerzitách, učebnice od c# vývojáře, dva webové odkazy se sbírkou příkladů a kurz programování v c#. Každý z těchto zdrojů má své klady a zápory. Z této diverzity druhů zdrojů lze určit jakým způsobem ideálně nakombinovat jejich silné stránky tak, aby se mohlo dosáhnout co nejlepší možnosti k sestavení obsahu předmětu TI.

3.1 Pramen č.1 − C# Programming Yellow Book, Rob Miles

Autor této práce si našel seznam učebnic pro začátečníky v programování v jazyce C#, které jsou volně ke stažení. Tato učebnice se vyskytovala na 1. místě v oblíbenosti. Učebnice je napsána univerzitním kantorem z University of Hull v Anglii. Látka této učebnice je velmi zajímavým a kvalitním způsobem přednášena čtenáři.



```
class Account
{
      public string Name ;
} ;

class StructsAndObjectsDemo
{
      public static void Main ()
      {
            Account RobsAccount ;
            RobsAccount = new Account();
            RobsAccount.Name = "Rob";
            Console.WriteLine (RobsAccount.Name );
      }
}
```

Code Sample 28 Compiling Account Class

The line I have added creates a new `Account` object and sets the reference `RobsAccount` to refer to it.

We have seen this keyword `new` before. We use it to create arrays. This is because an array is actually implemented as an object, and so we use `new` to create it. The thing that new creates is an *object*. An object is an instance of a class. I'll repeat that in a posh font:

"An object is an instance of a class"

Obr. 3-1 − ukázka stylu výkladu z prvního pramene [7]

Osnova učebnice je pro pochopení základů vhodně poskládána. Autor nejprve uvádí obecnou definici počítače a jakou funkci vzhledem k programování má. Dále pojednává o obecné teorii programů a programování. Autor uvádí základní příklad, na kterém vysvětlí použité programovací náležitosti, jako jsou datové typy, důvod použití "void" nebo co program

především dělá funkční v tom směru, ve kterém očekáváme, že fungovat bude. Každá z náležitostí je následně popsána ve výkladu pod názorným kódem.

Navazuje uchování/skladování textu, kde popisuje datové typy "char, string, bool" a přiřazovaní hodnot k proměnným V textu se vyskytují krátké odstavce nazvané "Programmer´s Point". Zde autor uvádí praktické připomínky k právě probírané problematice. Po této části následuje obecný popis, jak psát program. Logicky dalším krokem jsou možnosti, jak program ovládat v průběhu vývoje. Zde jsou poprvé zmíněny "loop" funkce (podmínky) a operátory k nim vztažené. Autor nadále uvádí různé variace těchto funkcí například: "do-while", "while", "for" a jak s těmito funkcemi pracovat.

Student začátečník by měl věnovat pozornost dalšímu tématu, což je popis různých prvků programování, jako například inkrementace nebo označení místa v textu ("placeholder"), kam bude hodnota přiřazena.

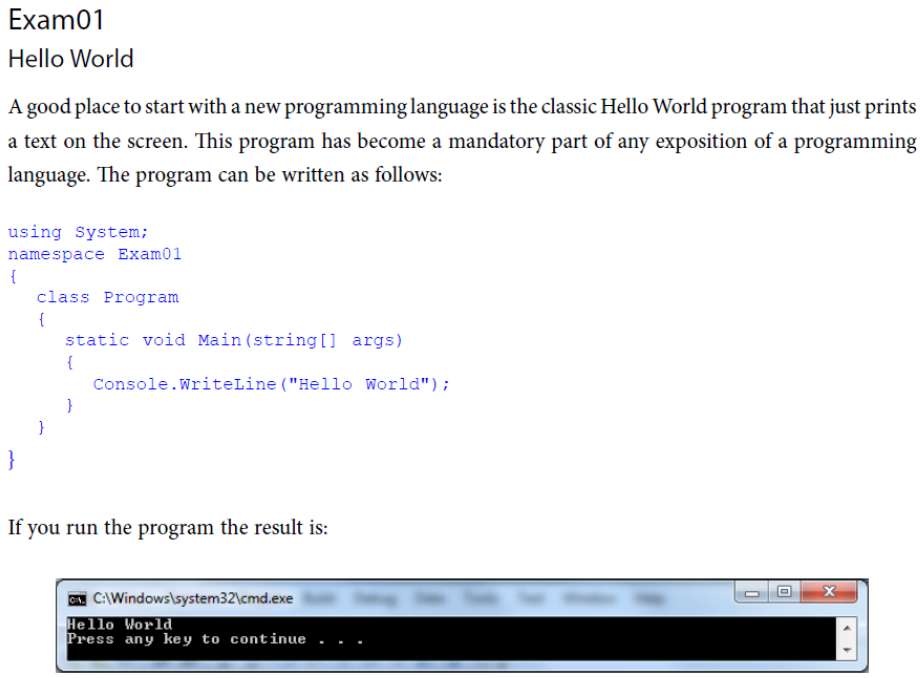
Dalším velkým tématem, které následuje, je pojednávání o metodách. Autor nejdříve stručně téma charakterizuje, přičemž k popisu využívá hezké slovní spojení "metody a lenost", čímž chce autor říci, že metodami si lze ušetřit spoustu práce, a navíc udělat kód přehledným. Nadále se poukazuje na práci s metodami, jejich ovládání skrze parametry/argumenty a vracení hodnot.

Po metodách autor začíná popisovat "Arrays" neboli pole. Udává důvody využití polí, druhy polí, což může být jednodimenzionální či multidimenzionální, a popisuje na jednoduchých případech, jakým způsobem je využít. Po tomto tématu autor vysvětluje výjimky a chyby ("exceptions and errors"), jak chytit výjimku ("catch"), jak přidat a vlastnost "finally" a házení podmínek ("throw"). Velice užitečnou funkcí při výběru z vícero možností je "switch", ke které se v učebnici uvádí, jaké jsou způsoby využití a jak by mohla vypadat konstrukce takovéto funkce. S funkcí "switch" se velmi často používá enumerace, která je popisována spolu se strukturami jako další větší okruh. Struktury jsou v učebnici uváděny spolu s objekty a referencemi. Autor vysvětluje definice jednotlivých pojmů pomocí textu, a také obrázků pro lepší vizuální představivost. Tato témata jsou využita v komplexnějším příkladu "bankovní účet". Na ten navazují i témata jako: statické třídy a metody, vysvětlení pojmu konstruktér ("constructor"), rozhraní ("interface"), dědičnost ("inheritance"), virtuální metoda a její přepsání ("virtual" a "override"), abstraktní třídy a metody.

Již mimo příklad je uvedeno další téma, a to je práce s textem, a to zahrnuje manipulaci, editaci textu nebo příkazy, délka nebo zastřihnutí (".Length" nebo ".Trim()"). Dále dokument obsahuje pokročilejší programování, které je pro účel této práce neužitečné. Na závěr autor přiložil přehled důležitých pojmů.

3.2 Pramen č.2 – C# 1 – Introduction to programming and the C# language

Učebnice se nacházela mezi oblíbeným výběrem volně dostupných pramenů pro studium programování v jazyce C#. Učebnice byla sepsána kantorem Aarhus University v Dánsku. Látka je vykládána srozumitelně a příklady jsou jednoduché k pochopení i pro začátečníky.



Exam01

Hello World

A good place to start with a new programming language is the classic Hello World program that just prints a text on the screen. This program has become a mandatory part of any exposition of a programming language. The program can be written as follows:

```
using System;
namespace Exam01
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Hello World");
      }
   }
}
```

If you run the program the result is:

```
C:\Windows\system32\cmd.exe
Hello World
Press any key to continue . . .
```

The program runs in a command window (prompt), where it prints the text *Hello World* on the screen.

Obr. 3-2 – ukázka stylu výkladu druhého pramene [4]

Kniha začíná uvedením do světa programování v jazyce C#, kde je hned přiložen jednoduchý příklad typu "Hello World" a k názorné ukázce je i uveden postup založení nového projektu v aplikaci. Následuje popis architektury běžného programu, kde autor udává jednoduchý příklad již obsahující metody, které zde také stručně vysvětluje. Přirozeně dalším tématem v osnově důležitým k chápání podstaty programování jsou proměnné Autor udává, jak se deklarují jednotlivé typy a jejich správné využití. V rámci tohoto tématu jsou představeny i operátory s ukázkou na různých příkladech.

Nyní autor začíná pojednávat o konzolových programech pomocí zajímavého příkladu pro vypočtení obvodu a plochy kruhu. Zde vysvětluje způsob výpočtu a proč bylo nutné použití třídy "Math" a přidává užití a vysvětlení pojmu "placeholder".

Po základním uvedení přichází na řadu téma kontrola programů. Zde se začínají popisovat "loop" funkce, neboli "while", "for", "if", "switch" nebo "do". Ke každé z funkcí je uvedena teoretická část, kde se popisuje také výstup z daného cyklu. Každý cyklus je doplněn příkladem pro lepší pochopení a obvykle pod příkladem následuje důvod využití různých operátorů podstatných pro správnou funkčnost kódu. Názornou ukázkou na cyklickou funkci "if-elseif-

else" je program na výpočet diskriminantu, nebo pro funkci "switch" program na výběr dne z týdne.

Dalším velkým tématem je práce s textem. Autor popisuje, jak deklarovat text v podobě objektu, formování textu a následně je uveden příklad s vysvětlením.

Dále se definují pole ("Arrays"). Problematiku jednotlivých typů polí zde autor popisuje stylem příklad + vysvětlení dané podoby pole, což může být text ("string"), "int" a další datové typy.

Po tomto tématu následuje detailnější popis principu objektově orientovaného programování. Prvním popisovaným důležitým pojmem je třída ("Class"). V tomto dokumentu se autor snaží vysvětlit třídu pomocí příkladu "hod mincí". V příkladu je poukázáno na vícero důležitých věcí, jako jsou metody, funkce "rand" pro vybrání náhodné hodnoty nebo užití operátorů. Vše je následně smysluplně vysvětleno v textu pod příkladem.

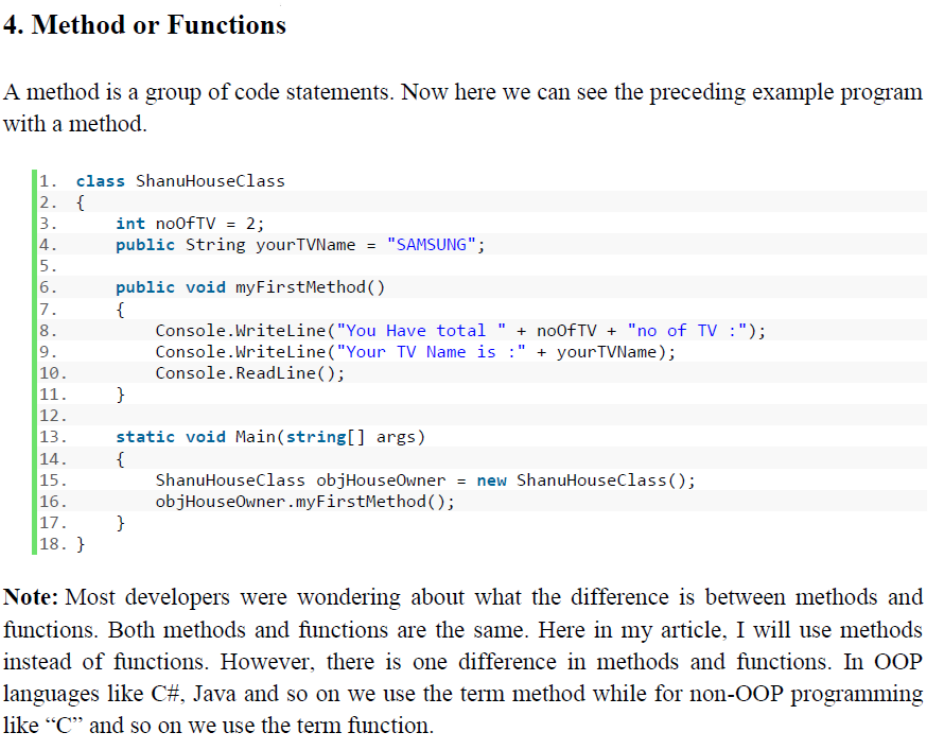
Dalším důležitým tématem jsou metody. Zde je již autor uvádí detailněji. Vysvětluje zde definici pojmenování metod, jak metody vrací hodnoty a užití parametrů v metodách.

Nadále se autor zabývá dědičností, kterou uvádí na příkladu "bod". Vytvořením rodiče a dále děděním na potomka poukazuje na způsob, jak dědičnost vlastně funguje. Později v dokumentu se toto cvičení využije k vysvětlení rozhraní ("interface"). Ještě lépe viditelnější použití dědičnosti je následující příklad "zaměstnanci", kde jsou ukázány i další funkce jako jsou "virtual + override", "base", "protected" nebo dále třída "abstract".

V prostřední části učebnice se autor věnuje statickým členům ("static members"), doplňuje informace ohledně multidimenzionálních polí, detailněji charakterizuje typy, vysvětluje typ "enum", strukturu ("struct") a dále již popisuje generické typy, výjimky a další látku obtížnější.

3.3 Pramen č.3 − Beginning C# Object Oriented Programming, Syed Shanu

Dokument byl sepsán programátorem z Microsoft Technologies v Jižní Koreji. Autor je původem z Indie. Tento dokument je krátkého obsahu a je spíše informační než výukový. V této učebnici je snaha programování ukázat na příkladech z běžného života normálního člověka.



4. Method or Functions

A method is a group of code statements. Now here we can see the preceding example program with a method.

```
1.  class ShanuHouseClass
2.  {
3.      int noOfTV = 2;
4.      public String yourTVName = "SAMSUNG";
5.
6.      public void myFirstMethod()
7.      {
8.          Console.WriteLine("You Have total " + noOfTV + "no of TV :");
9.          Console.WriteLine("Your TV Name is :" + yourTVName);
10.         Console.ReadLine();
11.     }
12.
13.     static void Main(string[] args)
14.     {
15.         ShanuHouseClass objHouseOwner = new ShanuHouseClass();
16.         objHouseOwner.myFirstMethod();
17.     }
18. }
```

Note: Most developers were wondering about what the difference is between methods and functions. Both methods and functions are the same. Here in my article, I will use methods instead of functions. However, there is one difference in methods and functions. In OOP languages like C#, Java and so on we use the term method while for non-OOP programming like "C" and so on we use the term function.

Obr. 3-3 − ukázka stylu výkladu třetího pramene [9]

Celý dokument je vysvětlován na příkladu "rodinný dům". Je tak učiněno pro lepší pochopení vykládané látky.

Na začátku dokumentu jsou vysvětlovány třídy ("class"), co je to objekt, vysvětlení druhů variabilních na příkladu "obývací pokoj" a nadále na tento příklad navazují metody a funkce.

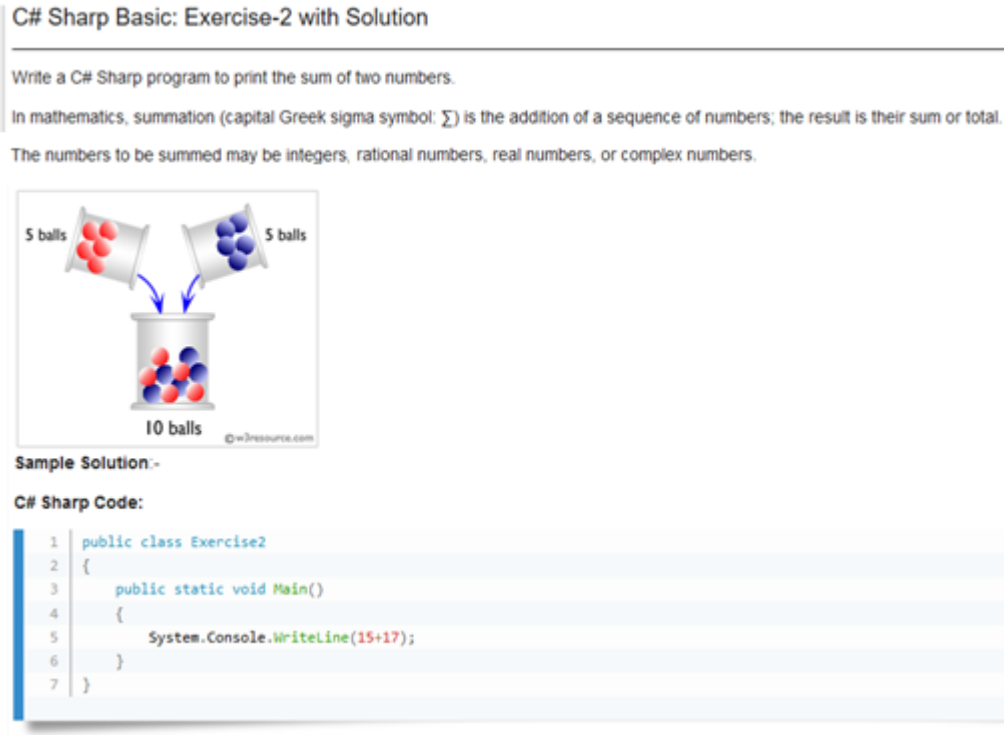
Dále se autor zabývá modifikátory přístupu ("access modifiers"), kde jednotlivé modifikátory, jako jsou "private", "public" a další, vysvětluje na bázi příkladu "rodinný dům".

Další větší téma je dědičnost. Ta na sebe následně váže abstrakci, přepisování ("override + virtual"), nebo vícenásobnou dědičnost. Až po těchto tématech přichází vysvětlení abstraktních tříd/metod ("abstract class/method"), a to samé platí pro virtuální třídy/metody ("virtual class/method"). Po virtuálních třídách jsou popisovány třídy uzavřené ("sealed class") a statické ("static class"), kde se autor také zabývá rozdíly mezi těmto dvěma třídami.

Jako poslední důležité téma autor popisuje rozhraní ("interface").

3.4 Pramen č.4 – 1. webový odkaz − w3resource.com

Tento pramen obsahuje pouze příklady z programování matematických funkcí v jazyce C#. Sbírka příkladů je rozdělena do několika skupin podle zaměření příkladů. Tyto skupiny jsou: základní cvičení, datové typy, funkce loop, příklady na pole, algortitmy, práce s textem, rekurzí, funkcemi nebo strukturami. Každá ze skupin obsahuje minimálně 10 různých příkladů.



Obr. 3-4 − ukázka výkladu látky čtvrtého pramene [11]

3.5 Pramen č.5 − 2. webová stránka − sanfoundry.com

Tak jako předchozí webový odkaz, i tento obsahuje pouze programy řešící různé matematické problémy v jazyce C#. Zde je dělení více rozvětvené, avšak ke každému problému je uveden pouze jediný příklad. Obsah se dělí na příklady se speciálními matematickými čísly (Fibonacci, faktoriál, a další), trigonometrické funkce (hodnota sin(x), cos(x), úhly v radiánech nebo stupních a další), příklady na dělitelnost a bitové operace, cvičení na kvadratické rovnice, příklady na permutace a kombinace. Posledním okruhem jsou součtové funkce, kam spadá příklad na nalezení maxima či minima nebo kalkulace.

```
/*
 * C# Program to Generate the Factorial of  Given Number
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace factorial
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, number, fact;
            Console.WriteLine("Enter the Number");
            number = int.Parse(Console.ReadLine());
            fact = number;
            for (i = number - 1; i >= 1; i--)
            {
                fact = fact * i;
            }
            Console.WriteLine("\nFactorial of Given Number is: "+fact);
            Console.ReadLine();

        }
    }
}
```

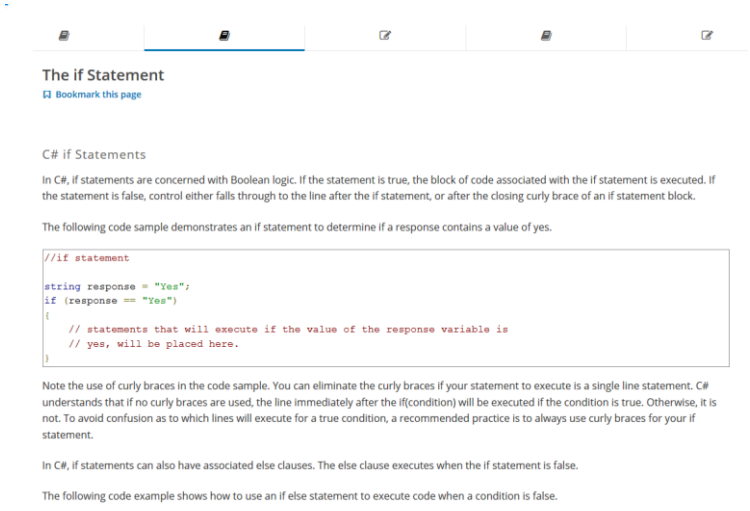
Here is the output of the C# Program:

```
Enter the Number
6
Factorial of Given Number is: 720
```

Obr. 3-5 – ukázka výkladu pátého pramene [8]

3.6 Pramen č.6 – Kurz edX – Introduction to C#

V tomto případě se jedná o internetový kurz pod záštitou Microsoftu. Konkrétněji se jedná o kurz pro začátečníky v programování jazykem C#. Využívají se zde jak videozáznamy s vysvětlením látky od profesionálního programátora, tak se po účastníkovi kurzu požaduje odpovídání na otázky a plnění úloh.



The if Statement

Bookmark this page

C# if Statements

In C#, if statements are concerned with Boolean logic. If the statement is true, the block of code associated with the if statement is executed. If the statement is false, control either falls through to the line after the if statement, or after the closing curly brace of an if statement block.

The following code sample demonstrates an if statement to determine if a response contains a value of yes.

```
//if statement

string response = "Yes";
if (response == "Yes")
{
    // statements that will execute if the value of the response variable is
    // yes, will be placed here.
}
```

Note the use of curly braces in the code sample. You can eliminate the curly braces if your statement to execute is a single line statement. C# understands that if no curly braces are used, the line immediately after the if(condition) will be executed if the condition is true. Otherwise, it is not. To avoid confusion as to which lines will execute for a true condition, a recommended practice is to always use curly braces for your if statement.

In C#, if statements can also have associated else clauses. The else clause executes when the if statement is false.

The following code example shows how to use an if else statement to execute code when a condition is false.

Obr. 3-6 – ukázka výkladu šestého pramene [2]

Kurz se dělí na tři důležité moduly. V prvním modulu je představován samotný jazyk C# a následuje výuka datových typů a jejich konverze nebo uvedení identifikátorů a operátorů. Ke každému tématu je uveden teoretický popis doplněný názorným krátkým příkladem či tabulkou. První modul zakončuje samostatná práce, kde je za úkol ve Visual Studiu napsat jednoduchý kód, který si lze následně zkontrolovat podle předlohy.

Druhý modul začíná představením látky pomocí videozáznamu, která se bude v této části probírat. První látkou v tomto modulu je popisování podmínek "if" a "switch". Ke každé jsou uvedeny jednoduché příklady. Modul pokračuje vysvětlováním podmínek "for", "while" a "do-while" a je zakončen znovu samostatnými úlohami, kde je úkolem napsat kódy k zmíněným podmínkám v tomto modulu.

Poslední modul začíná textovým uvedením. První probíranou látkou je deklarace metod, kde se zároveň zmiňují i modifikátory jako je "private", "public", a další. Následuje vysvětlení, jak se metody volají, jak získat data z metod, přetížení metod ("overloading") a využití parametrů u metod. Ke každé z těchto témat je uveden teoretický popis s názorným příkladem. V tomto modulu se jako poslední látka probírají výjimky a jejich způsoby řešení pomocí "try/catch/finally", případně vyhození podmínky "throw". Modul je ukončen samostatnými úkoly.

3.7 Dodatečně využitý pramen

K vypracování předběžné osnovy byl kromě již zmíněných pramenů v rozhodovací fázi využit pramen nový [10]. Dokument byl použit z důvodu obsáhlosti látky týkající se programování v programovacím jazyce C#. Například témata Lambda výrazy a LINQ byla zahrnuta pouze v tomto dokumentu. V dokumentech použitých v rozhodovací analýze byla témata nezmíněna. Navíc témata společná jsou podrobněji popsána, a proto byl tento dokument zařazen mezi hlavní zdroje.

4 Rozhodovací analýza

Rozhodování je proces, ve kterém je úkolem výběr nejlepší varianty z dostupných možností. V rozhodovacím procesu se postupně dopracovává k určitému rozhodnutí, které je učiněno rozhodovacím subjektem (autor práce) z několika dalších variant řešení daného problému. Objekt rozhodování může být charakterizován jako problém, na který je rozhodovací proces zaměřen. Jednotlivé varianty rozhodování představují možné výsledky v závislosti na jednání řešitele.

Kritéria slouží k definování té míry, do jaké je cíl splněn. Zároveň posuzují jednotlivé varianty a rozdíly mezi nimi. Tato analýza je provedena z hlediska individuálního. Když by byly brány v úvahu varianty rozhodovací analýzy obdržené od kantorů, jedná se poté o hledisko skupinové. Podle počtu jednotlivých kritérií můžeme říci, že se jedná o analýzu multikriteriální. V případě hlediska individuálního se řešení musí pokládat za subjektivní z pohledu studenta. Rozhodování v BP je za nejistoty, že můžeme uvažovat pouze o pravděpodobnosti jednotlivých scénářů. Z hlediska strukturovanosti problému se v případě BP jedná o špatně strukturovaný problém, jelikož se proces opakovat již nebude a je zde existence mnoha kritérií. Dle využité teorie se jedná o model rozhodování kvantitativně orientovaný. V případě způsobu rozhodování

se zde mluví o deskriptivní modelu, kde se popisují výhody a nevýhody řešení a následně se vybere ta nejlepší varianta. Váhy kritérií v BP byly stanoveny pomocí metody založené na párovém srovnávání. Varianty jsou hodnoceny výnosovým způsobem a vyjádřeny slovně vůči kritériím, viz *Tab. č. 4-3*.

Rozhodovací analýza se může rozdělit do několika fází. První fází je identifikace problému, následují analýza faktoru rozhodování (kritéria), zápis hodnot do tabulky k jednotlivým řešením (dokumentům), stanovení rizik (weby bez teoretického vysvětlení, látka v kurzu nepříliš obsáhlá), hodnocení navržených variant, stanovení vah kritérií, následuje přepočet a dostáváme se k finálním hodnotám analýzy. Podle účelu rozhodování se může konstatovat, že je použit model optimalizační, při kterém se hledají lepší řešení daného problému v závislosti na kritériích. V případě metody strom nastává seskupení kritérií do dílčích skupin. Pro tvorbu kritérií byl použit ordinální přístup, kde se může určit pořadí hodnot. Pro bakalářský projekt byla stanovena kritéria:

K1 – Návaznost témat

K2 – Rozčlenění kapitol

K3 – Srozumitelnost příkladů

K4 – Srozumitelnost výkladu textu

K5 – Úroveň látky vhodná pro porozumění cílové skupiny

K6 – Odbornost obsahu

K7 – Aktuálnost obsahu

K8 – Zaměření příkladů

K9 – Realizace ukázky příkladů

K10 – Zajímavost příkladů

K11 – Paleta příkladů k procvičení látky

K12 – Možnost tvorby řešení příkladů od žáka

K13 – Variace příkladů pro řešení problému

K14 – Využití příkladů žáky mimo vyučování

K15 – Nástroje k upoutání pozornosti žáka

K16 – Přehlednost textové úpravy

K17 – Vyzdvižení důležitých pojmů a definic

K18 – Stručnost teoretické části výkladu

K19 – Využívání komentářů

K20 – Odlišnost od výuky programování na vybraných technických univerzitách v ČR

K21 – Odlišnost od výuky na ZČU.

Podoba kritérií byla stanovena tak, aby bylo možné co nejlépe posoudit dané prameny z hlediska jak studenta, tak kantora. Množství kritérií bylo definováno pro citlivější hodnocení. Několik kritérií bylo převzato z použité literatury [6]. Kritéria mohou být rozdělena do několika skupin, jak bylo použito v metodě “strom krtiérií". Byly popsány skupiny S1 – Obsahové prvky, S2 – Srozumitelnost, S3 – Využitelnost, S4 – Příklady, S5 – Výukové prvky, S6 – Rozdíl oproti univerzitám.

4.1 Saatyho metoda

Tato metoda byla preferována díky její jednoduchosti a dobré srozumitelnosti. Jedná se o metodu preferenčních vztahů dvojic kritérií. Díky této metodě je možnost zjistit míru preference jednotlivých kritérií.

Počet bodů	Deskriptor
1	Kritéria jsou **stejně** významná
3	První kritérium je **slabě** významnější než druhé
5	První kritérium je **dosti** významnější než druhé
7	První kritérium je **prokazatelně** významnější než druhé
9	První kritérium je **absolutně** významnější než druhé

Obr. 4-1 – vysvětlení škály hodnocení kritérií [1]

Tato matice má na diagonále jednotky Sii = 1, sloupce a řádky se ohodnotí ve smyslu Sji = 1/Sij , kde Sij je odhad podílů vah kritérií vi a vj, přičemž platí Sij ~ vi / vj. Tyto váhy lze určit buď exaktním způsobem nebo určitou aproximací. V tomto případě se použila metoda aproximativní. Tento způsob určení kritérií může být hrubým odhadem, nebo je zde možnost provést přesnější odhad. Je více možností, jak přesnější odhad provést, ale v této práci je použit systém takový, že se stanoví geometrické průměry řádků matice. Následuje normalizace těchto řádkových geometrických průměrů, což znamená jejich vydělení součtem geometrických průměrů. Z této normalizace se získají přesnější odhady kritérií.

Kritérium	K1	K2	K3	K4	K5	K6	K7	Geomean	Váha
K1	1	1/2	2	6	2	3	3	1,84	0,22
K2	2	1	2	7	2	4	2	2,39	0,29
K3	1/2	1/2	1	1/3	1	2	1/2	0,96	0,11
K4	1/6	1/7	1/3	1	1/3	1/2	1/4	0,32	0,04
K5	1/2	1/2	1	3	1	2	1/2	0,96	0,11
K6	1/3	1/4	1/2	2	1/2	1	1/2	0,58	0,07
K7	1/2	1/2	2	4	2	2	1	1,35	0,16

Tab. č. 4-1 – ukázka jednoduchého příkladu Saatyho matice [1]

4.2 Metoda "strom kritérií"

Druhou možností řešení rozhodovacího procesu může být metoda určená pro řešení multikriteriálního rozhodování. Prvním krokem je určení počtu kritérií a jejich definic. Nadále se uvažuje, jaká kritéria jsou si podobná v rámci určitého okruhu (obsah, text, příklady a další). Kritéria se následně umístí do skupin na principu této podobnosti. Těmto skupinám subjektivně přiřadí váhy důležitosti. Ty se vzájemně násobí s jednotlivými vahami kritérií. Výsledkem je finální váha, se kterou se nadále počítá v rozhodovacím procesu. Metoda byla do BP autorem implementována z důvodu lepší přehlednosti pro kantory, kteří se účastnili na porovnávání ohodnocení kritérií studenta vůči kantorovi.

4.3 Rozhodovací postup autora

Zde budou popsány všechny tabulky, které autor zpracoval v doprovodném souboru Excel, viz *příloha 3*. Budou uvedeny neúplné verze tabulek z důvodu rozsáhlosti obsaženého textu. Celý text je uveden v *příloze 3* této práce.

Skupina kritérií	Váhy skupin	Kritéria	Váhy kritérií	Výsledná váha
S1	0,1	K1	0,3	0,03
		K2	0,3	0,03
		K3	0,4	0,04

Tab. č. 4-2 – ukázka metody "strom kritérií" [1]

Tabulka č.1

Jako první krok byla zvolena tvorba tabulky č. 1, ve které autor bakalářské práce (BP) uvádí použité prameny (varianty), viz *Tab. č. 4-3*, ve které od 3. do 8. sloupce jsou také pojmenovány.

Tabulka č.1

Rozhodovací analýza							
	Parametr	"CHEESE" Rob+Miles	C# intro Poul Klausen	Beginning with C# Syed Shanu	1. web. odkaz	2. web. odkaz	Kurz edX - Introduction to C#
Návaznost témat	K1	Obecně o programech	Knížka rozdělena do kap	Osnovu autor sepsal v jednom s	Jedná se o soul	jedná o sérii př	V tomto kurzu návaznost byl
Rozčlenění kapitol	K2	U 3. kapitoly bych vyne	autor žádné rozčlenění h	bez podkapitol	Pod názvy hlav	Rozčlenění - hla	Tento kurz se člení do modulů

Tab. č. 4-3 – vizualizace podoby první tabulky dle autora

Z počátku tvorby BP autor pracoval pouze s pěti prameny. Dodatečně byl přidán šestý pramen pro zajištění širší diverzity formátů výkladu. Můžeme konstatovat, že byly použity 3 tyto formáty: digitální forma učebnic (PDF), webový odkaz s repertoárem příkladů zaměřených na matematické funkce a internetový výukový kurz zaštítěný Microsoftem. Dále se zde kritéria, podle kterých se budou parmeny porovnávat. Pro tuto BP bylo sepsáno celkem 21 kritérií a každé kritérium bylo pro daný pramen charakterizováno, viz *Tab. č. 4-3*. Celá charakteristika kritérií je dostupná v přiloženém dokumentu Excel. Několik kritérií bylo použito z externího zdroje na internetu.

K posouzení pramenů je důležité se seznámit s jejich strukturou (osnovy), tedy jak části osnovy na sebe navazují či se dělí. Důležitými parametry jsou také srozumitelnost textu a příkladů. Z pohledu studenta může být důležitým kritériem zaměření na příklady, jejich zajímavost, přehlednost a stručnost a pro zapálené žáky též paleta příkladů a jejich variace.

Z kantorského hlediska se může předpokládat, že důležitými kritérii jsou nástroje k upoutání studentovi pozornosti k výkladu, osnovy pramenů, úroveň obtížnosti výuky, odbornost a celkově studentova upřednostněná kritéria k lepšímu porozumění ve vztahu student-kantor-látka.

Tabulka č.2

Tato tabulka slouží k informování o jednotlivých složkách osnov výuky programování na různých univerzitách v ČR. Na webových stránkách jednotlivých univerzit byly dohledány informace k předmětům o programování v 1. ročnících a nadále předměty zabývající se výukou programovacího jazyka C# nebo .NET v jakémkoliv ročníku. Tyto informace sloužily k vyplnění charakteristik jednotlivých variant pro kritéria K20 a K21. K této části se může vázat i doplňující informační tabulka "Tabulka pro zahraniční univerzity". Zde jsou uvedeny vybrané zahraniční univerzity, ke kterým též byly hledány informace o výuce programování.

Tabulka č.2

Osnova výuky ZČU x ČR				
	ČR			ZČU
	ČVUT - Pra	VUT - Brno	LIBEREC	
1. semestr	1. semestr	1. semestr - v jazyce C	1. semestr - FS - KSA/PP	1. semestr -JAVA
ostatní	5. semestr	4. semestr - FIT - http:/	1., 2. semestr - FA - pouz	3. semestr - programování v C/
	FS - str. 37	4.semestr - FIT - http://	4. semestr - MTI/VAW -	.

Tab. č. 4-4 – vizualizace podoby druhé tabulky dle autora

Tabulka č. 3

V této části rozhodovací analýzy se volí hodnoty kritérií pro jednotlivé prameny. Díky detailnějšímu prostudovaní každého z pramenů byl autor BP schopen hodnocení posoudit a následně sepsat, viz *Tab. č. 4-5*

Tabulka č.3

	Parametr	"CHEESE" Rob+Miles	C# intro Poul Klausen	Beginning with C# Syed Shanu	1. web. odkaz	2. web. odkaz	Kurz edX - Introduction to C#
Návaznost témat	K1	40	35	20	10	10	25
Rozčlenění kapitol	K2	35	35	0	40	30	30

Tab. č. 4-5 – vizualizace podoby třetí tabulky dle autora

Zde můžete u jednotlivých kritérií vidět, jak se autor mezi jednotlivými prameny rozhodoval. U prvního kritéria jsou racionálně lépe obodovány první tři varianty v podobě PDF souborů. Ty jsou psány jako učebnice, a tedy mají i svou osnovu. Zato webové odkazy jsou pouze ve formě sbírky příkladů, a osnovu nemají. Kurz svoji osnovu měl, a proto byl lépe obodován než webové odkazy.

V některých případech, jako je například kritérium K4, se vyskytuje ohodnocení 0, což znamená že tento pramen toto určité kritérium zcela nesplňuje či není v tomto případě způsob, jak ho posoudit.

Z této části je možnost také posoudit, jak jednotlivé prameny zapůsobily na řešitele analýzy.

U prvního pramene je čitelné, že nejvíce na "študákovu duši" udělalo dojem kritérium K17 – vyzdvižení důležitých pojmů a definic. U tohoto jediného pramene se toto shrnutí zakomponovalo. Bylo shledáno velmi užitečným, a proto si vysloužilo nejlepší ohodnocení. Pomine-li se absence využívání komentářů a možnost tvorby řešení příkladů žákem, nejslabším článkem ve smyslu obsahu pramene je úroveň látky vhodná k porozumění cílové skupiny. Je to z toho důvodu, že přibližně za polovinou obsahu vykládané látky se začne probírat pokročilé programování, které již nezahrnuje zaměření této BP.

Druhý pramen se liší zaměřením příkladů. Autor BP studuje FST ZČU, a proto ho více oslovily příklady s matematickým zaměřením, které v předchozím prameni nebyly. Je to názorný příklad subjektivnosti hodnocení. Za předpokladu řešitele ekonomického zaměření by ohodnocení pravděpodobně dopadlo jinak. Nejslabší oblastí u tohoto pramene je přehlednost textové úpravy kvůli velmi rozsáhlým reklamám obsaženým v tomto dokumentu. V některých případech přesahují až půl stránky obsahu a při detailnější četbě jsou tyto reklamy velmi nevhodné.

Třetí pramen je z PDF dokumentů nejkratší. A to také hrálo velkou roli při hodnocení. Předností v tomto případě je srozumitelnost příkladů. Díky aplikaci příkladů na rodinný dům i začátečník po velmi krátké chvíli dokáže pochopit autorův záměr. Jak již bylo zmíněno, tento pramen postrádá dostatečný rozsah látky, a proto nejhůře hodnocený vůči ostatním pramenům je kritérium K11 – paleta příkladů k procvičení látky.

Webové odkazy se můžou zhodnotit naráz, jelikož jen pár kritérií se liší o více než 5 bodů. Přednostmi webových odkazů jsou především v kritériu K8 – zaměření příkladů a kritériu K11 – paleta příkladů k procvičení. Když by se nebrala v potaz absence možnosti splnění kritéria,

tak nejhorším hodnoceným kritériem týkajícím se obsahu je K15 – nástroje k upoutání žáka. Avšak tyto webové odkazy nebyly zamýšleny k použití ve výuce, a proto relevantním nejslabším článkem je kritérium K13 – variace příkladů pro řešení jednoho problému.

Kurz nebyl tak rozsáhlý jako první dva prameny, ale určitá procenta látky pokrýval. Nejlépe hodnoceným kritériem bylo K3 – srozumitelnost příkladů. Kurz byl navržen tak, aby se i začátečník orientoval v probírané problematice. Nedostatkem tohoto typu řešení výkladu látky je opět malý rozsah variací příkladů a paleta příkladů nebyla také nikterak rozsáhlá. Z autorova pohledu tato dvě kritéria bohužel tento kurz citelně bodově znehodnotila.

Tabulka č. 4

V této tabulce autor BP dle dříve uvedené Saatyho metody sestrojil matici 21*21 dle počtu kritérií, ke kterým byly následně přiřazovány body s cílem určit výsledné váhy pro jednotlivá kritéria. Subjektivně, podle názoru autora BP, bylo provedeno toto bodové ohodnocení. Autor BP v této práci využil rozšířenou škálu hodnocení kritérií. Namísto doporučené škály 1-3-5-7-9 autor využil 1-2-3-4-5-6-7-8-9, a to z důvodu většího počtu kritérií a větší citlivosti. Vezme-li se v potaz názorný příklad z tohoto BP, tak ideálním příkladem by mohla být matice 3x3 pro poslední 3 kritéria v této tabulce.

Sestupné seřazení kritérií podle vah (tab.č 4)

Kritérium	Váha
K17	0,149
K15	0,144
K3	0,142
K8	0,1
K7	0,0093
K21	0,0091
K20	0,008

Tab. č. 4-7 – sestupné seřazení některých vah kritérií z tabulky č.4

		K19	K20	K21	geomean	váha
Využívání komentářů	K19	1	3	3	0,604563	0,018117
Odlišnost od výuky programování na vybraných technických univerzitách v ČR (*)	K20	1/3	1	1/2	0,267286	0,00801
Odlišnosti od výuky na ZČU (**)	K21	1/3	2	1	0,302374	0,009061

Tab. č. 4-6 – vizualizace podoby čtvrté tabulky dle autora

V *Tab. č. 4-7* je k vidění část matice z bakalářského projektu. Jak je již uvedeno v obecném popise postupu, na diagonále se nachází pouze hodnota 1. V případě prvního řádku se porovnávají důležitosti kritérií K20 a K21 vůči kritériu K19. Dle autorova ohodnocení kritérium K19 je lehce důležitější než dvě zbylá kritéria (červeně označeno). To odpovídá bodovému ohodnocení 3. V případě druhého řádku se již porovnávají jen 2 kritéria K20 a K21. Zde můžete vidět příklad, když porovnávané kritérium (K21) je nepatrně důležitější než kritérium výchozí (K20). Proto vidíme bodové ohodnocení v podobě zlomku (oranžově označeno). V poslední řádku nemáme co hodnotit, jelikož převrácené hodnoty řádků ve sloupcích nám již bodování obstaraly.

Dva poslední sloupce slouží k určení vah. Sloupec s pojmenováním "geomean" slouží pro výpočet geometrických průměrů řádků matice podle vzorce v MS Excel "=geomean (kritérium

K19 až kritérium K21)". V *Tab. č. 4-7* jsou zachovány hodnoty vah z celkového nezkráceného řešení. Díky předchozímu výpočtu lze nyní vyřešit vzorec pro výpočet vah, který je zadáván v MS Excel jako "=geomean/suma ($první až poslední geomean$)".

Dodatečně byla dodělána tabulka seřazených vah pro lepší přehlednost, viz *Tab. č. 4-6*. Pořadí bylo seřazeno sestupným modelem. Z této tabulky je viditelné, že autor BP nejvíce preferuje celkové shrnutí důležitých pojmů (K17) a v otázce protikladu autor přidělil nejmenší váhu odlišnosti od výuky programování na vybraných technických univerzitách v ČR (K20).

Tabulka č.5

V poslední tabulce rozhodovacího procesu Saatyho metodou jsou váhy kritérií násobeny hodnotami. Výsledky jsou zaznamenány a následně sečteny v řádku Suma, díky kterému se určí nejlepší varianta pro další postup.

V této BP bylo autorem rozhodnuto, že nejlepší variantou je pramen č.1 – C# Programming Yellow Book, Rob Miles, viz *Tab. č. 4-8*.

Tabulka č.5

	Váha	"CHEESE" Rob+Miles	C# intro Poul Klausen	Beginning with C# Syed Shanu	1. web. odkaz	2. web. odkaz	Kurz edX - Introduction to C#
Návaznost témat	0,014474	0,578965	0,506594	0,289483	0,144741	0,144741	0,361853141
Rozčlenění kapitol	0,011192	0,391721	0,391721	0,000000	0,447682	0,335761	0,335761261
Odlišnost od výuky programování na vybraných technických univerzitách v ČR (*)	0,00801	0,160194	0,160194	0,080097	0,120146	0,120146	0,120145529
Odlišnosti od výuky na ZČU (**)	0,009061	0,181224	0,181224	0,090612	0,135918	0,135918	0,135917734
SUMA	**1**	**44,886661**	**33,119955**	**23,407033**	**26,711281**	**24,693842**	**29,9760927**

Tab. č. 4-8 – vizualizace podoby páté tabulky dle autora

Tabulka č.6

Tabulka nahrazuje rozhodovací Saatyho matici (tabulka č.4) v první možnosti řešení. Zde je využit postup “stromu kritérií“. Název metody se může chápat jako větvení ze skupin do samostatných kritérií. Definice kritérií zůstávají stejné, jedinou změnou jsou rozdílné váhy a způsob porovnávání, viz *Tab. č. 4-9*. Autor nejdříve definoval počet skupin na bázi podobnosti kritérií. Dále následovalo subjektivní ohodnocení těchto skupin (oranžová), kde byla za nejdůležitější zvolena skupina S4 a nejméně důležitou S6. Následovalo subjektivní ohodnocení kritérií ve skupinách (červená), kde suma vah kritérií ve všech skupinách musí být rovna jedné. Vynásobením (zelená) vah kritérií váhou skupiny, do které kritéria spadají, se docílí výsledných (fialové) hodnot, které budou důležité pro pokračování rozhodovacího procesu.

Tabulka č.6

	Parametr	Váha kritérií	Skupina	Váha skupin	Výsledná váha
Návaznost témat	K1	0,75	S1	0,10	0,075
Rozčlenění kapitol	K2	0,25			0,025

Tab. č. 4-9 – naznačení postupu metodou "stromu kritérií" pro skupinu S1

Dle seřazených hodnot (viz *Tab. č. 4-10)* může být konstatováno, že jako nejlépe autorem ohodnocené kritérium skončilo K17 – vyzdvižení důležitých pojmů a definic a nejhůře skončilo kritérium K19 – využívání komentářů.

Sestupné seřazení kritérií podle vah (tab.č.6)

Kritérium	Váha
K17	0,135
K15	0,105
K11	0,1
K7	0,0165
K18	0,015
K19	0,015

Tab. č. 4-10 – sestupné seřazení některých vah kritérií z tabulky č.6

Tabulka č.7

Tabulka je identická jako tabulka č.5, jen jsou zde použity váhy kritérií (fialově z *Tab. č.* 4-9) vypočtené pomocí metody "strom kritérií" ze 6. tabulky. Zde se vynásobí váhy kritérií s obodováním variant, které bylo převzato z tabulky č. 3, kde žádná hodnota nebyla změněna a varianty jsou též stejné. Jsou pozorovatelné přibližně stejné výsledné hodnoty variant, viz *Tab. č. 4-1*.

Tabulka č.7

	Váha	"CHEESE" Rob+Miles	C# intro Poul Klausen	Beginning with C# Syed Shanu	1. web. odkaz	2. web. odkaz	Kurz edX - Introduction to C#
Návaznost témat	0,075	3	2,625	1,5	0,75	0,75	1,875
Rozčlenění kapitol	0,025	0,875	0,875	0	1	0,75	0,75
Odlišnost od výuky programování na vybraných technických univerzitách v ČR (*)	0,0225	0,45	0,45	0,225	0,3375	0,3375	0,3375
Odlišnosti od výuky na ZČU (**)	0,0275	0,55	0,55	0,275	0,4125	0,4125	0,4125
SUMA	**1**	**42,4825**	**31,9225**	**19,9125**	**26,24**	**23,55**	**28,0175**

Pozn.: Bodové ohodnocení variant převzato z tabulky č.3

Tab. č. 4-1- vizualizace podoby sedmé tabulky dle autora

5 Analýza výsledků

Analýza výsledků se bude týkat pouze první citlivější rozhodovací metody. Druhá metoda uvedena pouze pro snazší provedení porovnání názoru studenta s názory kantorů.

Výsledky vycházející z tabulky č. 5 byly z větší části očekáváné, hlavně pro první tři prameny v PDF formátu. Dle očekávání první pramen – C# Programming Yellow Book skončil na nejlepším místě. Tento pramen na autora nejlépe zapůsobil uvedením celkového shrnutí pojmů, zajímavým a chytlavým stylem výkladu a také strukturou příkladů.

Druhý pramen C# 1 – Introduction to programming and the C# language po přečtení prvního již nezapůsobil takovým dojmem. Tento dokument je napsán běžným výkladem látky, nikterak chytlavým pro běžného čtenáře. Obsahově je však tento dokument velmi dobrý. Je zde vysvětleno mnoho látky i příkladů, a proto se umístil na druhém místě.

Třetí a poslední dokument PDF se umístil na této pozici hlavně díky absenci rozmanitosti probírané látky. Dokument obsahuje pouze 35 stran a na tak krátkém rozsahu nelze popsat dostatečné množství látky. Avšak kladně byl hodnocen styl výkladu látky a snaha přiblížit odvětví programovaní běžnému člověku skrze přirovnání k příkladu rodinného domu a objektech v něm.

Zbylé webové formáty již tak předvídatelné nebyly. První webový odkaz podle provedené analýzy byl bodově ohodnocen lépe než webový odkaz druhý. Paleta příkladů u prvního webového odkazu byla o něco širší. Zaměření jednotlivých sekcí příkladů je zde z pohledu autora lépe organizované. Zadání příkladů jsou tu specifičtější, což autor ocenil.

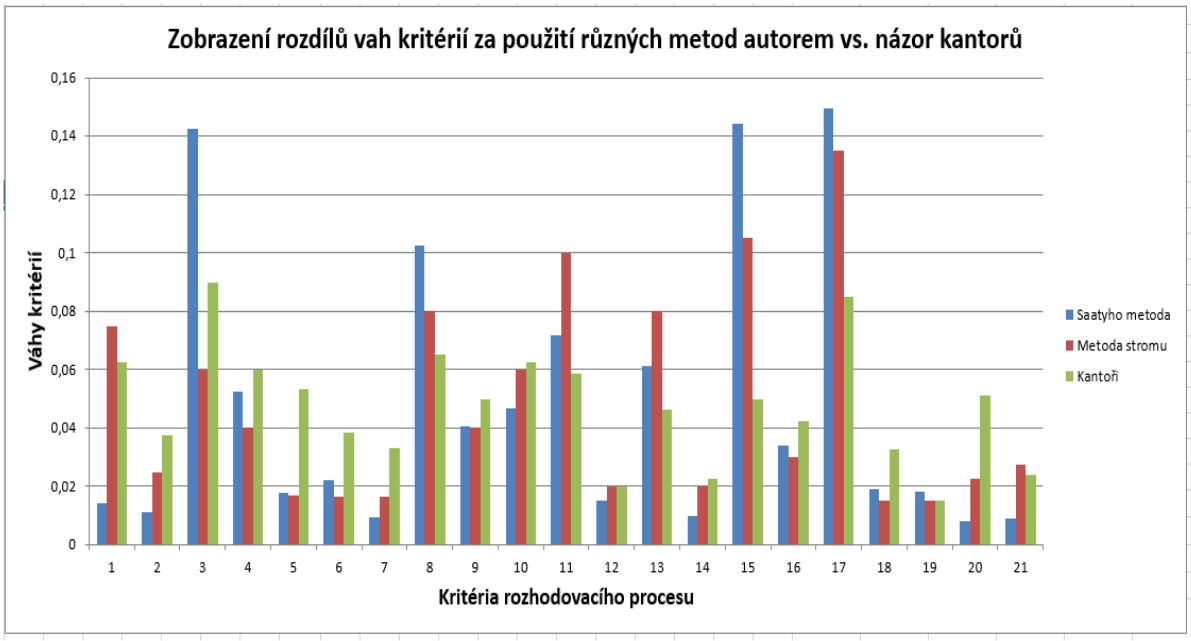
Druhý webový odkaz působí informativněji než odkaz první. Jsou zde uvedeny příklady i složitějších matematických funkcí, které v našem případě nebudou s velkou pravděpodobností nijak využity. Druhý webový odkaz má své hodnoty, avšak ty nedosahují kvalitě hodnot prvního webového odkazu.

Kurz byl přidán dodatečně do této rozhodovací analýzy pro větší diverzitu druhů pramenů. Kurz byl shledán autorem BP přínosný pro osobní rozvoj v programování. Je zde po Vás velmi často požadováno vymyslet vlastní řešení daného problému. To vede dotyčnou osobu k soběstačnosti a samostatnosti v rámci práce programování. Dobrým výukovým komponentem obsahující tento kurz jsou také videa, ve kterých profesionál hovoří o zrovna řešených tématech, poukazuje na důležité věci, a především názorně představuje krátké vzorové programy k tématu. Těmito dobrými vlastnostmi si kurz vysloužil dokonce třetí místo v pořadí. Pro autora BP to bylo i malým překvapením, protože tento kurz překonal v hodnocení i první webový odkaz.

Výsledky metodou "stromu kritérií" slouží pouze jako orientační. Metoda byla implementována pro možnost snadného porovnání názoru autora BP s názory dotázaných kantorů.

5.1 Porovnání metod

Bylo provedeno porovnání vah kritérií vycházejících z použitých dvou přístupů řešení pro stanovení vah. Z grafického znázornění (viz *Obr. 5-1*) jsou viditelné patrné rozdíly mezi metodami Saatyho matice (modře) a metodou "stromu kritérií" (červeně), ačkoliv je zpracovával stejný řešitel (autor BP).



Obr. 5-1- grafické znázornění výsledných vah kritérií

Největším rozdílem disponuje kritérium K3 – srozumitelnost příkladů. Naopak nejmenší rozdíl v hodnotách je pozorován u kritéria K9 – realizace ukázky příkladů.

Z důvodu nepříliš velké citlivosti metody "strom kritérií" pro volbu vah autor upřednostnil metodu Saatyho matice. Zde byly jednotlivé váhy detailněji porovnávány, a proto se tyto hodnoty pokládají za přesnější a lépe odrážející autorův názor.

5.2 Porovnání přístupů studenta a kantorů

Graf, viz *Obr. 5-1*, vykresluje porovnání jednotlivých metod použitých autorem BP k získání vah pro kritéria, ale zároveň též porovnává názor autora skrze jednotlivé metody s názory kantorů (zeleně) dosažených metodou "strom kritérií" pro lepší přehlednost. Do srovnání byly dosazeny zprůměrované váhy jednotlivých kritérií. Zprůměrované hodnoty byly vypočteny z dat obdržených od kantorů, viz *Tab. č. 5-2*. S autorem spolupracovali čtyři kantoři z katedry KPV na FST ZČU. Každý z nich vyučuje či se podílí na vyučování technické informatiky na FST ZČU.

Zprůměrování výsledků

	1. Výsledn	2. Výsledn	3. Výsledn	4. Výsledn	Průměr vah
K1	0,075	0,05	0,05	0,075	0,0625
K2	0,025	0,05	0,05	0,025	0,0375

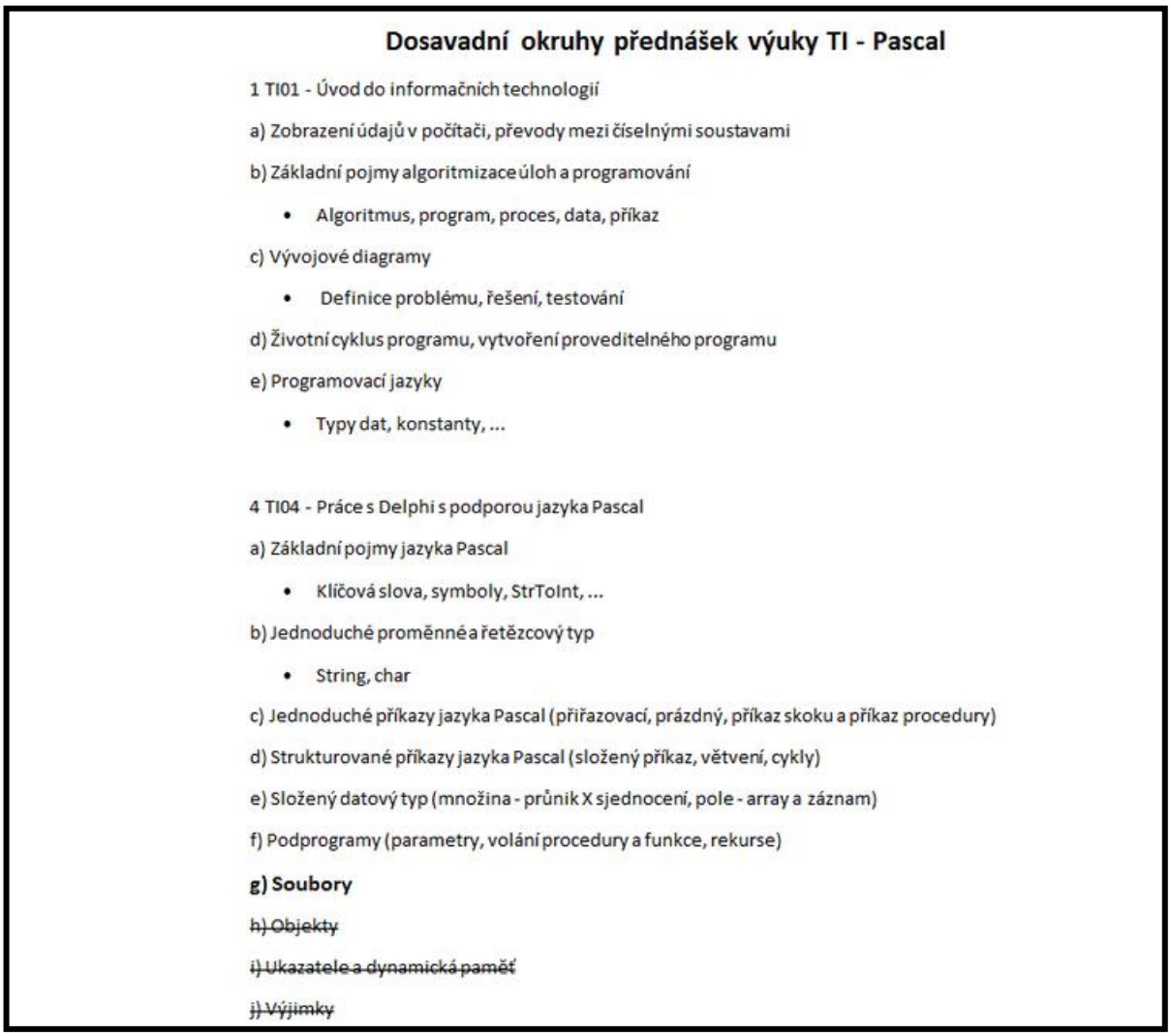
Tab. č. 5-2 – tabulka získaných dat od kantorů s výslednými vahami kritérií

Z grafického porovnání je čitelné, že se priority studenta s prioritami kantorů často neshodují. U porovnávání verzí metody "strom kritérií" studenta (červeně) a kantorů (zeleně) jsou téměř shodné názory u kritérií K12 – možnost tvorby řešení příkladů od žáka, K14 – využití příkladů žáky mimo vyučování nebo K10 – zajímavost příkladů. Autor se však velice míjí s názory kantorů v případech kritérií K17 – vyzdvižení důležitých pojmů a definic, K11 – paleta příkladů k procvičení látky nebo K3 – srozumitelnost výkladu textu. Rozdílnost názorů se samozřejmě předpokládá. Kritérium K17 bylo nejlépe hodnoceným z pohledu studenta, zatímco z pohledu kantorů se jedná o kritérium důležité, avšak ne tak dobře bodově ohodnocené. U kritéria K1 je možné pozorovat subjektivitu, s jakou autor kritéria ohodnotil a zároveň velký rozdíl výsledků mezi použitými metodami. Zde byl předpoklad vyšší váhy z kantorského pohledu hodnocení než od studenta. Tím samým případem je kritérium K20, které autor považoval za méně důležité pro jeho zájmy, ale kantory bylo kritérium ohodnoceno velmi vysoko v porovnání s autorovým ohodnocením.

6 Sestavení osnovy

Jako první byla sestavena předběžná osnova. Autor BP dle svého uvážení vybral důležitá témata, která se týkají programování, z využitých dokumentů. Předběžně byla vybrána ta, která jsou nezbytně nutná k pochopení vlastního principu programovacího jazyka a programování obecně. Tato základní témata jsou obsažena ve všech PDF dokumentech, které byl využity, s výjimkou dokumentu "Beginning with C#" od autora Syed Shanu.

Autor prostudoval přibližnou dosavadní koncepci, dle které se vyučuje předmět TI v jazyce Pascal, aby měl alespoň hrubou představu, jak sestavit osnovu novou v jazyce C#. Podoba současné osnovy je vidět na *Obr. 6-1*.



Dosavadní okruhy přednášek výuky TI - Pascal

1 TI01 - Úvod do informačních technologií

a) Zobrazení údajů v počítači, převody mezi číselnými soustavami

b) Základní pojmy algoritmizace úloh a programování

- Algoritmus, program, proces, data, příkaz

c) Vývojové diagramy

- Definice problému, řešení, testování

d) Životní cyklus programu, vytvoření proveditelného programu

e) Programovací jazyky

- Typy dat, konstanty, ...

4 TI04 - Práce s Delphi s podporou jazyka Pascal

a) Základní pojmy jazyka Pascal

- Klíčová slova, symboly, StrToInt, ...

b) Jednoduché proměnné a řetězcový typ

- String, char

c) Jednoduché příkazy jazyka Pascal (přiřazovací, prázdný, příkaz skoku a příkaz procedury)

d) Strukturované příkazy jazyka Pascal (složený příkaz, větvení, cykly)

e) Složený datový typ (množina - průnik X sjednocení, pole - array a záznam)

f) Podprogramy (parametry, volání procedury a funkce, rekurse)

g) Soubory

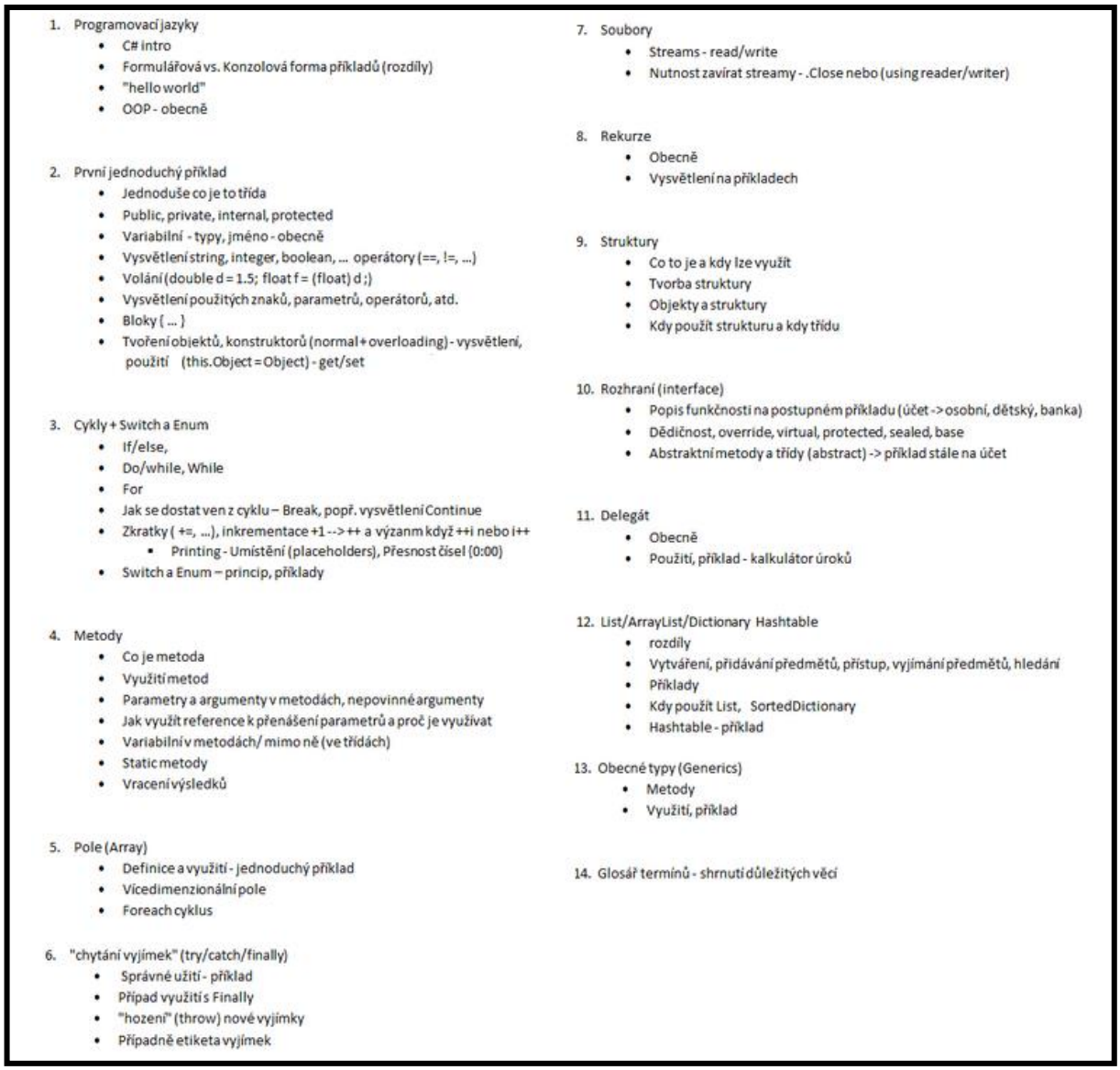
~~h) Objekty~~

~~i) Ukazatele a dynamická paměť~~

~~j) Výjimky~~

Obr. 6-1 – dosavadní okruhy výuky TI [5]

Autorem nadále byla vytvořena hrubá předběžná osnova. Témata v této fázi byla vybírána čistě autorem dle jeho uvážení založeného na důležitosti jednotlivých témat. Výsledkem byla osnova, viz *Obr. 6-2*.



1. Programovací jazyky
   - C# intro
   - Formulářová vs. Konzolová forma příkladů (rozdíly)
   - "hello world"
   - OOP - obecně

2. První jednoduchý příklad
   - Jednoduše co je to třída
   - Public, private, internal, protected
   - Variabilní - typy, jméno - obecně
   - Vysvětlení string, integer, boolean, ... operátory (==, !=, ...)
   - Volání (double d = 1.5; float f = (float) d ;)
   - Vysvětlení použitých znaků, parametrů, operátorů, atd.
   - Bloky { ... }
   - Tvoření objektů, konstruktorů (normal + overloading) - vysvětlení, použití (this.Object = Object) - get/set

3. Cykly + Switch a Enum
   - If/else,
   - Do/while, While
   - For
   - Jak se dostat ven z cyklu – Break, popř. vysvětlení Continue
   - Zkratky (+=, ...), inkrementace +1 --> ++ a význam když ++i nebo i++
     - Printing - Umístění (placeholders), Přesnost čísel (0:00)
   - Switch a Enum – princip, příklady

4. Metody
   - Co je metoda
   - Využití metod
   - Parametry a argumenty v metodách, nepovinné argumenty
   - Jak využít reference k přenášení parametrů a proč je využívat
   - Variabilní v metodách/ mimo ně (ve třídách)
   - Static metody
   - Vracení výsledků

5. Pole (Array)
   - Definice a využití - jednoduchý příklad
   - Vícedimenzionální pole
   - Foreach cyklus

6. "chytání vyjímek" (try/catch/finally)
   - Správné užití - příklad
   - Případ využití s Finally
   - "hození" (throw) nové vyjímky
   - Případně etiketa vyjímek

7. Soubory
   - Streams - read/write
   - Nutnost zavírat streamy - .Close nebo (using reader/writer)

8. Rekurze
   - Obecně
   - Vysvětlení na příkladech

9. Struktury
   - Co to je a kdy lze využít
   - Tvorba struktury
   - Objekty a struktury
   - Kdy použít strukturu a kdy třídu

10. Rozhraní (interface)
    - Popis funkčnosti na postupném příkladu (účet -> osobní, dětský, banka)
    - Dědičnost, override, virtual, protected, sealed, base
    - Abstraktní metody a třídy (abstract) -> příklad stále na účet

11. Delegát
    - Obecně
    - Použití, příklad - kalkulátor úroků

12. List/ArrayList/Dictionary Hashtable
    - rozdíly
    - Vytváření, přidávání předmětů, přístup, vyjímání předmětů, hledání
    - Příklady
    - Kdy použít List, SortedDictionary
    - Hashtable - příklad

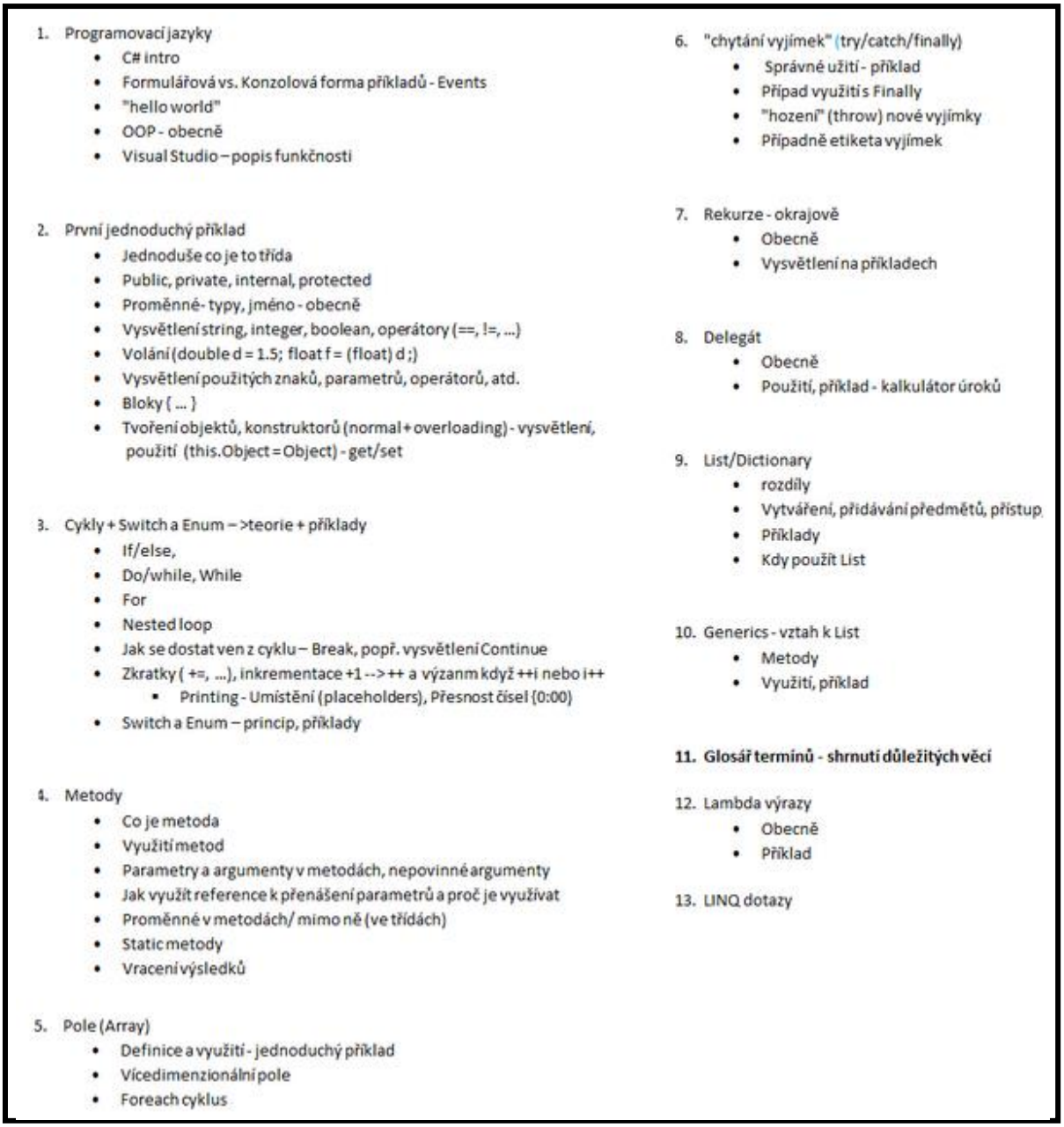
13. Obecné typy (Generics)
    - Metody
    - Využití, příklad

14. Glosář termínů - shrnutí důležitých věcí

Obr. 6-2 − hrubá předběžná osnova dokumentu

Předběžná osnova následně byla prokonzultována s vedoucím bakalářské práce a konzultantem. Následně byla upravena dle kompromisu názorů kantorů a autora BP. Vycházelo se z hrubé předběžné osnovy. Výsledkem byla osnova, viz *Obr. 6-3*.

Z *Obr. 6-3* jsou viditelné jak organizační rozdíly, tak rozdíly týkající se šíře obsahu oproti osnově z *Obr. 6-2*. Po domluvě s kantory byla definitivně zahrnuta témata „Lambda výrazy“ a „LINQ“. Autor tato témata považoval za pokročilejší způsob programování, a tudíž nevhodné pro zahrnutí do tvorby osnovy. Avšak po osobní zkušenosti autora z pracovního prostředí autor usoudil, že i tato témata vlastní stejnou prioritu k zahrnutí jako témata základní.



1. Programovací jazyky
   - C# intro
   - Formulářová vs. Konzolová forma příkladů - Events
   - "hello world"
   - OOP - obecně
   - Visual Studio – popis funkčnosti

2. První jednoduchý příklad
   - Jednoduše co je to třída
   - Public, private, internal, protected
   - Proměnné - typy, jméno - obecně
   - Vysvětlení string, integer, boolean, operátory (==, !=, ...)
   - Volání (double d = 1.5; float f = (float) d ;)
   - Vysvětlení použitých znaků, parametrů, operátorů, atd.
   - Bloky { ... }
   - Tvoření objektů, konstruktorů (normal + overloading) - vysvětlení, použití (this.Object = Object) - get/set

3. Cykly + Switch a Enum – >teorie + příklady
   - If/else,
   - Do/while, While
   - For
   - Nested loop
   - Jak se dostat ven z cyklu – Break, popř. vysvětlení Continue
   - Zkratky (+=, ...), inkrementace +1 -->++ a význam když ++i nebo i++
     - Printing - Umístění (placeholders), Přesnost čísel {0:00}
   - Switch a Enum – princip, příklady

4. Metody
   - Co je metoda
   - Využití metod
   - Parametry a argumenty v metodách, nepovinné argumenty
   - Jak využít reference k přenášení parametrů a proč je využívat
   - Proměnné v metodách/ mimo ně (ve třídách)
   - Static metody
   - Vracení výsledků

5. Pole (Array)
   - Definice a využití - jednoduchý příklad
   - Vícedimenzionální pole
   - Foreach cyklus

6. "chytání vyjímek" (try/catch/finally)
   - Správné užití - příklad
   - Případ využití s Finally
   - "hození" (throw) nové vyjímky
   - Případně etiketa vyjímek

7. Rekurze - okrajově
   - Obecně
   - Vysvětlení na příkladech

8. Delegát
   - Obecně
   - Použití, příklad - kalkulátor úroků

9. List/Dictionary
   - rozdíly
   - Vytváření, přidávání předmětů, přístup
   - Příklady
   - Kdy použít List

10. Generics - vztah k List
    - Metody
    - Využití, příklad

11. **Glosář terminů - shrnutí důležitých věcí**

12. Lambda výrazy
    - Obecně
    - Příklad

13. LINQ dotazy

Obr. 6-3 – předběžná osnova dokumentu

Za základní témata je autorem považováno:

- určité uvedení do problematiky programovacího jazyka,
- popis rozhraní, ve kterém bude program zobrazen jako výstup tzv. IDE,
- vysvětlení definice základních procesů a pojmů a jejich použití,
- cykly,
- pole nebo list.

Jako okrajová témata, byla po konzultaci s kantory zvolena:

- zachycení výjimek.
- rekurze.

Tato témata budou stručně teoreticky vysvětlena, aby studenti o existenci této látky věděli. Doložena budou krátkým, jednoduchým příkladem. Pro studenty technických oborů může být i docela zajímavým tématem "Rekurze", jelikož se díky rekurzi snadno vypočte příklad na faktoriál.

Témata, jež byla vyřazena, jsou:

- soubory,
- rozhraní,
- struktura.
- hashtable.

Témata byla vynechána z důvodu přílišné komplexnosti. Výuka těchto témat by byla myslitelná ve výuce pokročilejšího programování.

1 Programovací jazyk C# ... 5
2 OOP ... 6
2.1 Co je objekt? ... 6
2.2 Visual studio (VS) ... 7
2.2.1 CLR ... 8
2.3 První program ... 9
2.3.1 První náš kód ... 10
3 Formulář vs. Konzole ... 12
3.1 WinForms a WPF (GUI) ... 12
3.2 WinForms ... 13
3.3 WPF ... 13
3.4 Rozdíly ... 14
3.5 Výhody ... 14
4 Uvedení do programování ... 15
4.1 Co je to třída ... 15
4.2 Co je to objekt ... 15
4.3 Tvoření objektů, konstruktorů ... 16
4.3.1 Konstruktory ... 16
4.3.2 Nový objekt ... 16
4.4 Public, private, internal, protected členy ... 18
4.5 Proměnné ... 18
4.6 Typy ... 19
4.6.1 Boolean ... 19
4.6.2 string ... 20
4.6.3 Využití Placeholderů pro možnost tisku ... 20
4.6.4 integer ... 20
4.7 Operátor ... 21
4.7.1 Operátorové zkratky ... 23
4.8 Závorky ... 24
4.9 Volání (Casting) ... 26
5 Podmínka IF ... 26
6 Cykly ... 28
6.1 DO – WHILE cyklus ... 28
6.1.1 WHILE cyklus ... 29
6.2 FOR cyklus ... 29
6.3 Vnořené cykly ... 30
6.4 Únik z cyklů ... 31
6.5 Vracení se na začátek cyklu ... 32
7 Výčtový typ (Enumeration) ... 32
8 Switch ... 33
9 Metody ... 35
9.1 Vracení hodnot ... 37
10 Pole ... 37
10.1 Vícedimenzionální pole ... 39
10.2 Foreach ... 39
11 Výjimky ... 40
12 Rekurze ... 41
13 Delegát a Event ... 43
14 List/Dictionary ... 45
14.1 List ... 45
14.2 Obecné typy (Generics) ... 46
14.3 Obecné typy a List ... 46
14.4 Slovník (Dictionary) ... 47
15 Lambda výrazy ... 48
15.1 Lambda výrazy jako Delegát ... 48
16 LINQ ... 49
17 Glosář ... 50
18 Literatura ... 53

Obr. 6-4 – finální osnova dokumentu

Při samotné tvorbě dokumentu byla osnova nadále upravována. Bylo přidáno několik podtémat, avšak hlavní body byly zachovány, viz *Obr. 6-4*.

6.1 Tvorba jednotlivých témat osnovy

Níže popisovaná témata jsou vázána k osnově z *Obr. 6-4*.

Jako úvodní téma je rozumné uvést určité seznámení s tímto programovacím jazykem. V seznámení se student dozví vlastnosti C# programovacího jazyka nebo stručné vysvětlení, co je to objekt, kompilátor a za jakým účelem se využívají.

V dalším bodu se vysvětluje princip C# programovacího jazyka, čímž je objektivně orientované programování (OOP). Zde je podrobněji definován objekt. Aby bylo možno pochopit co se pod pojmem "objekt" myslí, je zde vysvětlován na přirovnání abstraktního a reálného objektu. Je zde také uvedeno rozlišení objektu dle charakteristik. V této části je popsáno i rozhraní, ve kterém se bude C# kód psát. Nazývá se Visual Studio a na obrázcích budou ukázány a následně okomentovány základní funkce tohoto rozhraní.

V následujícím tématu je již čas se patřičně seznámit s prvním programem, a tudíž i jeho kódem. Aby bylo možno vůbec kód psát, je nutno založit nový projekt a vybrat, jakou formu tento projekt bude mít. Postup vytvoření projektu je popsán obrázky s krátkými komentáři. Jako "startovní" příklad je nejen v českých, ale i světových učebnicích využíván příklad typu "AhojSvěte". Úkolem tohoto "startovního" programu není zamotat studentovi hlavu, ale osvětlit mu opravdové základy zápisu kódu. Jsou zde vysvětleny principy použití jednotlivých termínů jako je definice třídy (class) nebo metody Main() a jejího obsahu.

Důležitý k podotknutí je fakt, v jaké formě se příklady budou studentům představovat. Doposud se upřednostňoval na hodinách Technické Informatiky formát formulářový. Avšak ve všech využitých dokumentech se upřednostňuje forma konzolová, kterou upřednostňuje i autor této práce. Konzole sice ukáže pouze napsaný kód a jeho funkčnost bez rozvinutějšího uživatelského rozhraní (user interface), ale k pochopení logiky kódu není lepší metody. Formuláře již obsahují určité uživatelské rozhraní, kde lze programovat například tlačítka. V této části tyto dvě verze budou obecně popsány a vzájemně porovnány, kde u formulářové formy nadále bude porovnání "staré" (WinForms) a "nové" (WPF) verze formulářů.

V dalším bloku je popsán pojem "třída" (class). Je zde uveden jednoduchý příklad k pochopení principu fungování tříd. V návaznosti na třídu je zde krátce popsán objekt s jeho charakteristickými vlastnostmi a pravidly a následuje vysvětlení, jakým způsobem se objekty tvoří. K třídám vztažené je zde vysvětlení pojmů public, internal a další, které definují, jak se třída bude chovat vůči zbytku programu. Dále jsou zde popsány stručně proměnné. Na ně navazuje definice typů. Pod typy jsou základní tvary proměnných například jako string nebo integer. S hodnotami proměnných budeme ve většině případů určitým způsobem pracovat. K tomuto účelu nám slouží operátory. Ty mají za úkol dát například metodě vědět, že se jedná o sčítání, a tudíž má metoda vrátit výchozí hodnotu jako součet těchto dvou čísel. Aby toho nebylo málo, existují i operátorové zkratky, které mají ulehčit práci s psaním kódu. Je obecně známo, že programátor je člověk líný, a proto si rád ulehčuje práci všude, kde je to možné.

Nyní začínáme pronikat do programovacího jazyka hlouběji. V této části obsahu se krátce definuje pojem cyklus, a následně se podrobněji vysvětlují jednotlivé druhy cyklů doprovázené jednoduchým příkladem. Blok je uzavřen stručným vysvětlením, jakým způsobem cyklus předčasně opustit nebo jej vrátit zpět na počáteční pozici z daného místa.

Dále je popisováno, co si lze představit pod pojmem "výčtový typ". Je uveden krátký a srozumitelný příklad na dny v týdnu s následným vytištěním vybraného výsledku.

Podobným stylem je vysvětlen "SWITCH". Tématika příkladu na dny v týdnu je zde taktéž využita. Je zde příklad a pod ním vysvětlení funkčnosti s principy, na jakých SWITCH pracuje.

Velmi důležitým tématem jsou Metody, které jsou nutné k funkčnosti téměř každého programu. Slouží ke strukturaci kódu na menší části, a tím pádem k lepší přehlednosti a orientaci v kódu. Jsou zde uvedeny lehké příklady na vizualizaci, jak se metody volají, že mohou být zavolány s parametrem/parametry nebo že existuje volání přetížené metody. Stručnou teorií je na závěr vysvětlen způsob vracení hodnot z metod.

Pro výpočet určité řady čísel a příkladů podobných je dobré znát pár informací o polích (Arrays). Je zde vysvětleno, jak se pole vytváří a jakými způsoby se definuje jejich velikost, což je ukázáno na příkladu. Autor se zde nadále zmiňuje o vícedimenzionálních polích, kde je možné kromě definice počtu řádek nadefinovat i počet sloupců v poli. Pro práci s poli existuje jeden velmi šikovný cyklus s názvem FOREACH (pro každého). Tento cyklus umožňuje programátorovi projít celé pole a pro každou položku v poli vykonat určitou akci.

Následujícím tématem jsou výjimky a práce s nimi. Je stručně vysvětleno, co si pod pojmem výjimka představit, jak s výjimkami pracovat, a především jak je zachytit tak, aby nám program v případě neočekávané chyby nepřestal pracovat, ale například pouze zobrazil okno se zprávou popisující tuto chybu a možná řešení této chyby.

Velmi krátce po podmínkách je definována Rekurze. Ta může sloužit například k výpočtu faktoriálu. Tento způsob zápisu metody je zapotřebí, když je metoda definována sama sebou. Rekurze je obtížná na pochopení, a ještě těžší je odhadnout případ, kdy ji lze využít, a proto se v podkladech nebude příliš rozvádět. Více popsána by rekurze měla být v případném navazujícím předmětu na Technickou Informatiku.

V případě práce ve formulářové formě bude nutné řízení událostí. S tímto problémem nám pomohou Delegáti. V této části jsou delegát a událost stručně definovány. Je vysvětleno, jakou funkci delegát zastupuje a následně je termín vysvětlen na lehkém příkladu z bankovního prostředí.

V dalším okruhu jsou popisovány kolekce List a Slovník (Dictionary). U Listu je vysvětleno, jak se vytváří, jak jsou definovány hlavní operace, které v rámci Listu provádět, a následně na příkladu jsou některé z nich uvedeny. Velmi často využívané v praxi jsou Listy s obecným typem, a proto je krátce popsán princip, jak obecný typ listu přidělit. To samé platí i pro Slovník. U slovníku se definují do závorek <…> klíč spolu s určitou hodnotou, dle kterých se následně provádí různé operace.

Dodatečně bylo dodáno téma "Lambda výrazy" a s ním spojené téma "LINQ". Lambda výrazy slouží jako operátor. Užití tohoto výrazu je ukázáno v příkladu s kolekcí List. Lambda výrazem je například v kolekcích možno nahradit Delegáta. LINQ nám dovoluje lépe "pronikat do kódu". Převážně v kolekcích a polích, kde by se musel použít k získání určitého výsledku velmi rozsáhlý kód, díky LINQ lze tento kód napsat na jeden řádek (tyto řádky ale nebývají krátké rovněž, ale stále jsou výhodnějšími).

Jako poslední téma byl vytvořen glosář pojmů. Zde student bude schopen rychlého dohledání významu jednotlivých pojmů z programování, které v dokumentu byly zmíněny. Jedná se o stručný popis vhodný k připomenutí podstaty hledaného pojmu.

7 Hrubé podklady k výuce

Autorem, po konzultaci s kantory, byly sestaveny hrubé podklady k podpoře výuky Technické Informatiky v rámci Západočeské univerzity v Plzni. Podklady jsou dostupné v příloze této práce, viz *Příloha 1*. Zde bude zobrazena malá část dokumentu, a to konkrétně vysvětlování cyklů. Ty budou sloužit i jako podklady k tvorbě již zmíněné vzorové prezentace.

Výňatek části dokumentu obsahující cykly je zobrazen od nadpisu *č.7.1* a končí nadpisem č. *7.1.6*.

V osnově dokumentu se téma "Cykly" nachází pod číslovkou 6, viz *Obr. 6-4*.

7.1 Cykly

Podmínkové tvrzení umožňuje provést část kódu, pokud bude daná podmínka splněna, tedy bude True. Nicméně velmi často je nutné určitý proces několikrát zopakovat za předpokladu, že podmínka bude pravdivá/nepravdivá. Pro tento případ nám poslouží cykly.

C# má několik cest, jak výsledku docílit. Závisí to převážně na cíli, kterého se chce dosáhnout. Několik cest existuje proto, aby v různých případech byla práce s kódem zjednodušena použitím jedné nebo více cest naráz.

7.1.1 DO – WHILE cyklus

Do – while konstrukce vypadá přibližně takto:

```
do
  Tvrzení nebo blok
while (podmínka) ;
```

Zde je možné opakovat kus kódu do té doby, než podmínka hodnoty bude False. Dobré by bylo si také zapamatovat to, že cyklus DO-WHILE proběhne vždy alespoň jedenkrát, i když test je již předurčen k neúspěchu. Názorný příklad [7]:

```
using System;
class Forever
{
  public static voidMain ()
        {
              do
```

```
                    Console.WriteLine ( "Mám hlad" ) ;
                while( true);
            }
        }
```

Tento kód je naprosto v pořádku. Jak dlouho by tento kód běžel, je dobrá otázka. Kód obsahuje části lidské psychologie, energie a možná i trochu té kosmologie.

Program s tímto kódem poběží tak dlouho dokud:

- Vás to přestane bavit,
- Vám dojde elektřina,
- vesmír nevybuchne a nezůstane nic,
- přestanete mít hlad.

7.1.2 WHILE cyklus

Někdy je zapotřebí rozhodnout, zda smyčku zopakovat předtím, než bude spuštěna. K tomu nám poslouží cyklus WHILE. Pro lepší pochopení lze uvést příklad, kde je třeba se zeptat na hodnotu ještě předtím, než bude rozhodnuto, zda je dotázaná hodnota přijatelná cyklem (true) či nikoliv.

Obecný příklad:

```
while(podmínka)
        Tvrzení nebo blok
```

7.1.3 FOR cyklus

Často bude potřeba zopakovat určitou věc daným počtem smyček. Cyklus FOR je pro tento případ jako dělaný.

```
1  using System;
2  public class Exercise1
3  {
4      public static void Main()
5      {
6      int i;
7      Console.Write("\n\n");
8      Console.Write("Display the first 10 natural numbers:\n");
9      Console.Write("---------------------------------------");
10     Console.Write("\n\n");
11
12      Console.WriteLine("The first 10 natural number are:");
13
14     for (i=1;i<=10;i++)
15     {
16      Console.Write("{0} ",i);
17     }
18     Console.Write("\n\n");
19   }
20 }
```

Obr. 7-1 – příklad na cyklus FOR [11]

C# umožňuje konstrukci cyklu FOR a jeho podmínek na jeden řádek tímto způsobem:

```
for (vstup; mez ; update)
{
  Věci od kterých je vyžadováno, aby se několikrát zopakovali
}
```

Vstup vloží hodnotu do kontrolní proměnné, kterou cyklus začne. Mez je podmínka, která musí platit pro cyklus FOR, aby mohl dále pokračovat ve smyčce. Update je podmínka, kterou se provádí v každé smyčce například inkrementace/dekrementace kontrolní proměnné. Všimněte si, že podmínky jsou děleny pomocí znaku ";" a ne čárkou. Přesný postup cyklu FOR zní:

a. Vložení počáteční hodnoty do kontrolní proměnné
b. Stanovení meze a ověření, zda cyklus po dokončení smyčku uzavřel
c. Podmínky budou opakovány
d. Provedení update
e. Opakování od bodu (b.)

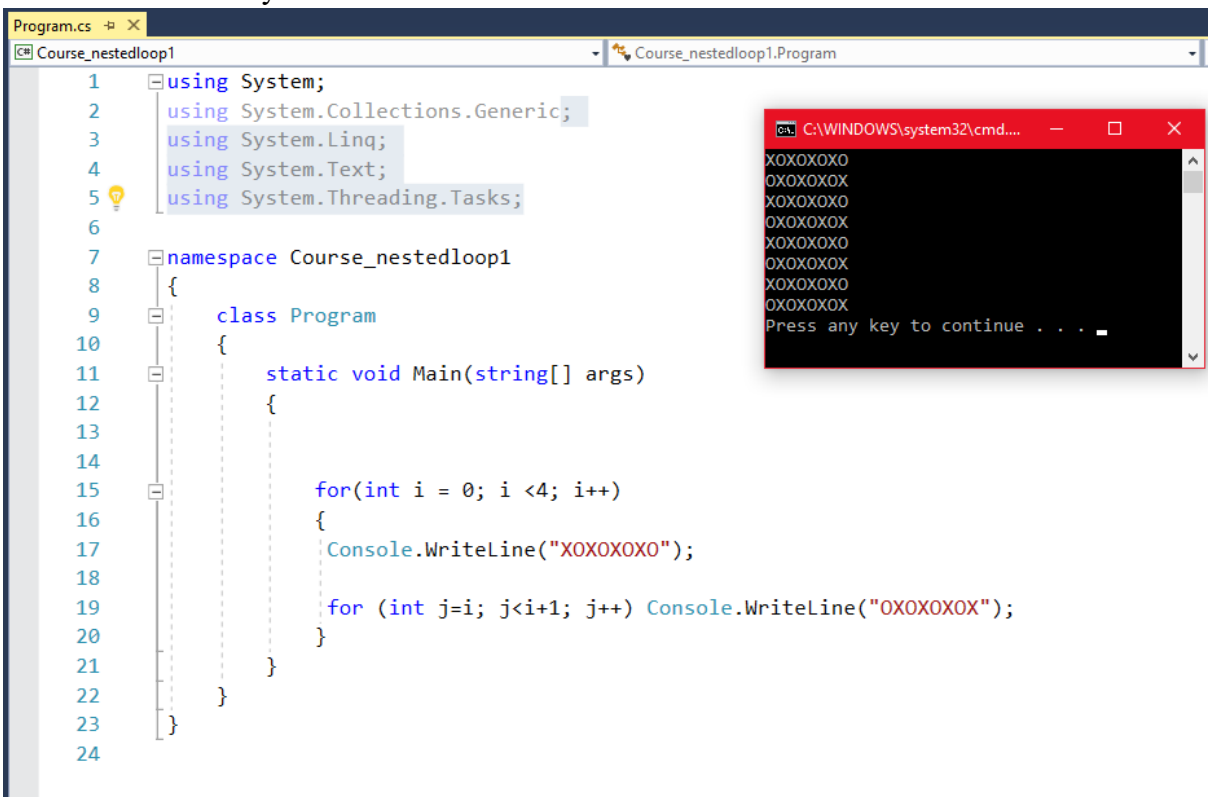
7.1.4 Vnořené cykly

Vnořený cyklus může být chápán jako "cyklus v cyklu", kde máme jeden vnější cyklus a v něm vnořený druhý, vnitřní cyklus. Vnitřní cykly proběhnou vícekrát než cykly vnější. Zde je názorný příklad vnořeného cyklu:

```
for (vstup, mez, update)
{
  for (vstup, mez, update)
  {
        Proveditelný kód
  }
  ...
}
```

Po dosazení vstupního parametru pro vnější cyklus bude proveden jeho obsah mezi {…}. Zde se nachází vnořený vnitřní cyklus. Jeho vstupní parametr bude inicializován, jeho podmínky budou zkontrolovány a kód uvnitř závorek {…} vnitřního cyklu bude proveden a následně vstupní proměnná vnitřního cyklu bude "updatována". To vše, mimo vstupu počátečního parametru, se bude opakovat, dokud proměnná nevrátí hodnotu True. Poté bude provedena druhá iterace prvního cyklu. Jeho proměnná bude "updatetována" a celý vnitřní cyklus bude proveden znovu, a to minimálně tolikrát, kolikrát bude zopakován cyklus vnější.

Příklad vnořeného cyklu:



Obr. 7-2 – příklad vnořeného cyklu FOR [2]

7.1.5 Únik z cyklů

Někdy by bylo třeba z cyklu předčasně "utéct", tzn. program rozhodne, že není už třeba dále ve smyčce pokračovat a následně bude provedeno tvrzení uvnitř cyklu k opuštění smyčky. Toho se může docílit takzvaným "break" tvrzením. Je to příkaz k okamžitému opuštění smyčky.

Příklad na využití "break":

```
while (beziOK)
        {
          kus kódu
           ....
          if (preruseno)
          {
                  break ;
          }
          .... další kus kódu
        }
```

7.1.6 Vracení se na začátek cyklu

Někdy bude potřeba se vrátit na začátek cyklu a udělat ho celý znovu. Tento případ může nastat, když se cyklus dostane dále, než je potřeba. Pro řešení zmíněného problému C# poskytuje řešení klíčové slovo continue. To má význam přibližně takovýto:

Prosím nepokračuj již dále a vrať se zpět. Jdi zpět na začátek cyklu a udělej všechny "updaty" a další věci zcela znovu.

V následujícím programu bude bool proměnná "Vse_bylo_splneno" nastavena jako pravda (True), jakmile dosáhneme požadované hloubky cyklu.

```
for ( predmet = 1 ; predmet<Celkem_predmetu ; predmet = predmet +1 )
{
 .... práce s předmětem ....

  if (Vse_bylo_splneno ) continue ;

… další práce s předmětem ....
}
```

"Continue" způsobí, že program "restartuje" cyklus s další hodnotou předmětu, pokud hodnota bude OK.

7.2 Styl autorovy tvorby podkladů

Z vytažené části dokumentu zde uvedené je vidět, jakým stylem autor práce navázal na jednotlivé body utvořené osnovy. Byla zde snaha o stručnost jednotlivých definic témat a pojmů. Autor se též pokoušel co nejvíce textu definovat příklady, jelikož z osobního hlediska je vidí jako lepší alternativu pro výklad látky (možný vliv stylu výuky na FST – matematika, mechanika, atd.).

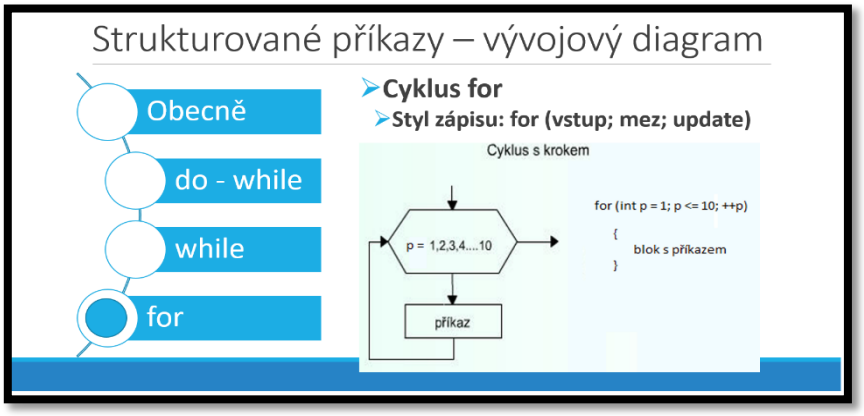
Jednotlivá klíčová slova byla zvýrazněna z důvodu lepšího chápání v návaznosti na samotné pracovní prostředí (Visual Studio), kde se obdobné barvy ke značení klíčových slov využívají. Student by díky této skutečnosti mohl lépe chápat příklady uvedené v dokumentu. Příklady jsou uvedeny jak ve formě pouhého textu, tak i v pracovním prostředí Visual Studio. Bylo převzato několik příkladů z pramenů a následně se autor odkazuje na jejich původ. Tyto převzaté příklady jsou ideální pro pochopení látky i studenty, kteří se doposud s programováním nesetkali. Několik příkladů bylo vytvořeno autorem bakalářské práce. Vytvořené příklady se zaměřují převážně na strojní tématiku a jejich obtížnost zůstává stále vna úrovni prozatímních znalostí studenta.

Autor se snažil psát dokument podobnou formou, jakou je sepsán pramen od autora Roba Milese. Je velmi těžké psát dokument se zaměřením, jako je programování. K napodobení tohoto nadhledu R. Milese byly v dokumentu využity obrázky s vtipným kontextem ohledně programování v jazyce C#. Tento prvek by mohl zachytit studentovu pozornost, jelikož dnešní

dobou jsou tyto obrázky nazývané "meme" ve velké oblibě a zároveň studenta upozorní na nějaký důležitý fakt k právě probíranému tématu.

8 Tvorba předběžné formy prezentace dokumentu

Po konzultaci s vedoucím bakalářské práce byl dohodnut další výstup této práce, a tím je část výukové prezentace předmětu TI. Prezentace, jak již bylo zmíněno, bude pouze na určitou tématiku. Od autora BP bylo vyžadováno upravení dosavadní prezentace v oblasti obsahu (Pascal na C#). Layout prezentace byl převzat z již existující prezentace [3]. Cílem je hlavně přehlednost, aby studenti nemuseli zdlouhavě hledat na jednotlivých listech prezentace právě probíraný bod. Zároveň musí listy být graficky poutavé. Obsah listů musí být stručný, nejlépe v jednotlivých bodech, které budou obsahovat jen nejdůležitější informace, viz *Příloha 2*.

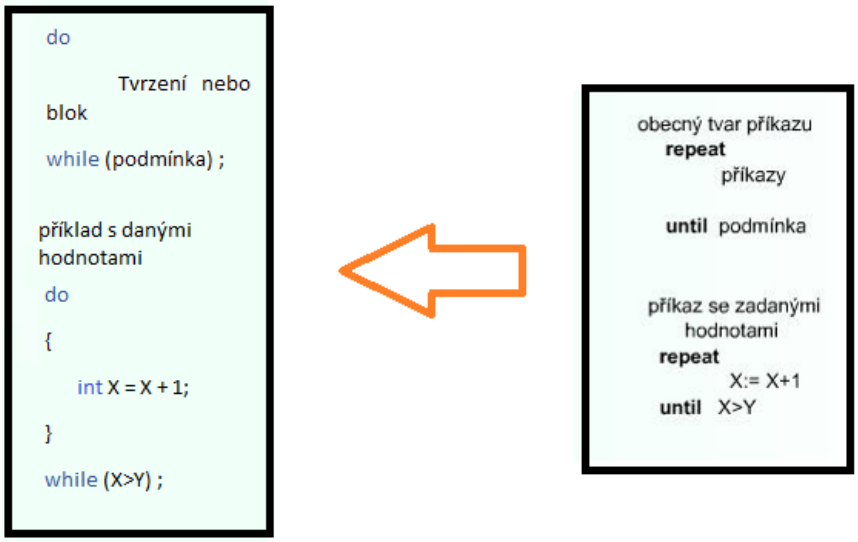


Obr. 8-1 – vizualizace stylu předběžné prezentace

8.1 Změny provedené v prezentaci

Z originální prezentace bylo vytaženo téma týkající se cyklů. Zbytek obsahu prezentace nebyl využit.

V prezentaci byl nahrazen cyklus "repeat-until" cyklem obdobným v jazyce c# "do-while". Funkce těchto dvou cyklů jsou prakticky totožné. Byl změněn názorný příklad, kde se uvádí obecný tvar cyklu a nadále příklad s hodnotami, viz *Obr. 8-2*, kde je možno vidět i blízkou podobnost napsaného kódu. Grafické znázornění zůstává identické.

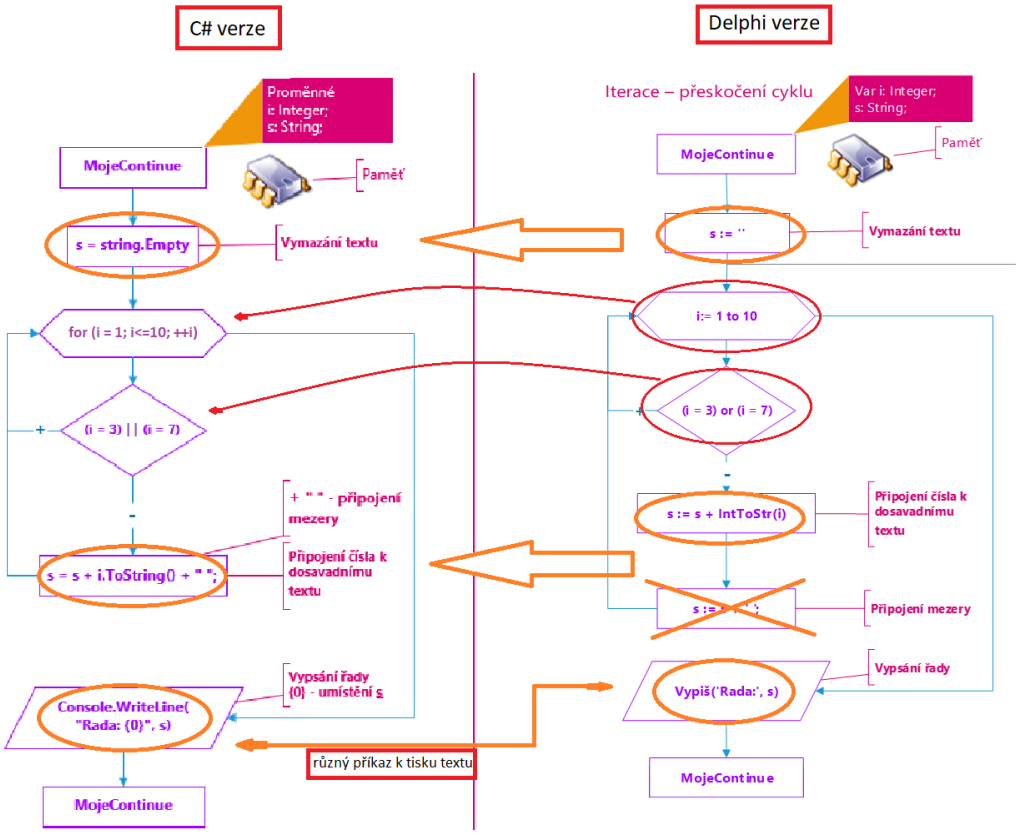


Obr. 8-2 – přepsání cyklu "repeat-until"

U cyklu "for" musel být změněn zápis cyklu. V C# je odlišný, než jak je tomu v dosavadně využívanému softwaru Delphi, viz *Obr. 8-3*. Byl dodán i obecný zápis cyklu pro lepší srozumitelnost. Grafické znázornění zůstává identické.

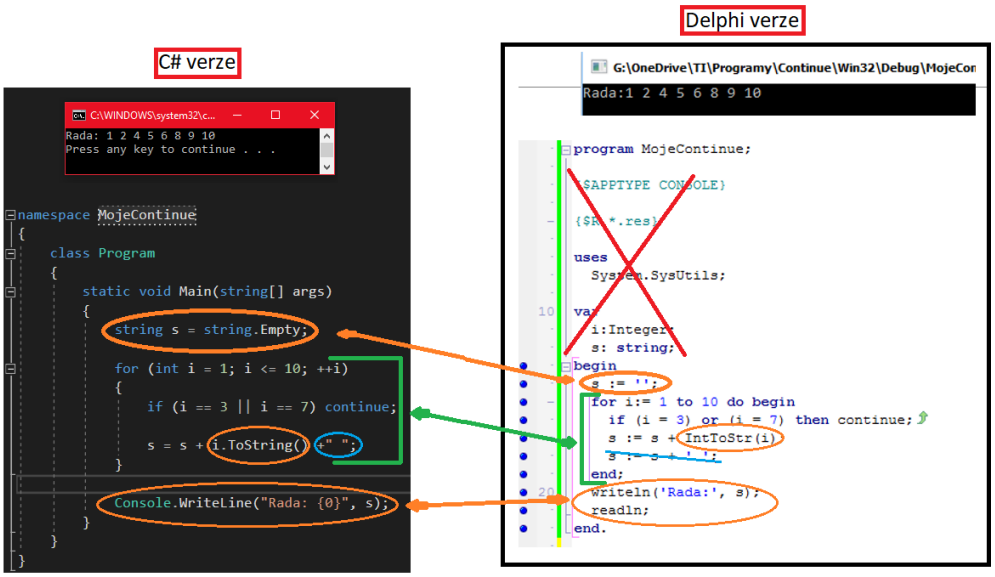
Obr. 8-3 – změna zápisu cyklu "for"

Poslední část tématu tvoří názorný příklad na přeskočení cyklu a iteraci. Zde byly provedeny úpravy jak grafické, tak v kódu. Kód se musel značně upravit, aby byl spustitelný a funkční v jazyce C#. Z grafické stránky byl upraven diagram, který popisuje kroky daného cyklu [3]. Z *Obr. 8-4* je možné vidět rozdíly, které se vážou na využití jiného programovacího jazyka. Pro prázdný textový řetězec (string) byl změněn zápis, jelikož pravidla pro jazyk C# by zápis ve formě jazyka Delphi nepovolil a kompilátor by nahlásil chybu. Plnění proměnné "s" čísly je v případě C# řešeno odlišným, ale nejspíše i přehlednějším způsobem. Navíc je zde možnost eliminace řádky přidávající mezeru za čísla v proměnné "s". Nakonec se upozorňuje na odlišný zápis tisku výstupu příkladu.



Obr. 8-4 – změny v diagramu cyklu

Porovnání jednotlivých kódů lze vidět na *Obr. 8-5*. V kódu jazyka Delphi se variabilní definují na začátku kódu. Musí být definována jak proměnná typu "string" tak "int". V C# verzi je definována pouze proměnná "string", a navíc rovnou je deklarováno, že se jedná o prázdnou proměnnou. Cyklus "for" se zde liší formou zápisu, jak již bylo řečeno výše a nadále svým obsahem, viz *Obr. 8-4*. Konečným prvkem kódů je příkaz k vytisknutí výstupu (výsledné řady čísel). V *Obr. 8-5* je očividný rozdíl v zápisu příkazu k tisku.



Obr. 8-5 – rozdíl mezi kódy C# a Delphi vztažených k diagramu

9 Závěr

Byla provedena rozhodovací analýza týkající se dostupných pramenů, ze které byl stanoven základ pro tvorbu struktury dokumentu, kterým je pramen C# Programming Yellow Book, napsaný Robem Milesem. K provedení analýzy byly vypsány možné varianty a sepsána odpovídající kritéria k možnosti porovnání variant. Byla vytvořena Saatyho matice za účelem určení preference kritérií. Následovalo ohodnocení variant. Konečným krokem byla výstupní tabulka pro vyjádření finálního výsledku rozhodovacího procesu. Proces a výsledky rozhodování autor i vedoucí práce shledává racionálními, a proto výsledné hodnoty budou použity k dalšímu postupu v této práci. Pro lepší vizualizaci možných postupů rozhodování byla zpracována druhá metoda „strom kritérií“. Srovnání rozhodovacích metod provedené studentem nebo kantorem by mělo sloužit k odůvodnění autorova rozhodování a porovnání pohledu studenta na výukové materiály oproti pohledu kantorů.

Na základě provedené analýzy byla postupně sestavena osnova dokumentu po konzultacích s vedoucím bakalářské práce. Následně byl utvořen obsah hrubého dokumentu na základě využívaných pramenů. Jedná se o hrubé podklady, tudíž teoretická část dokumentu není tolik rozvedena. V dokumentu bylo využito rozsáhlé množství příkladů k dobrému procvičení látky. Některé příklady byly vytvořeny se zaměřením na strojní tématiku. Dokument je dostupný v příloze této práce.

Předběžná prezentace byla vytvořena na základě podkladů poskytnutých od konzultanta této práce. Byl využit layout prezentace a do určité míry upraven grafický obsah prezentace. Textově poskytnutá prezentace byla více než dostačující, tudíž byly změněny pouze odborné pojmy pro jiný programovací jazyk.

Cílem bylo navrhnout dokument, který by zaujal studenty a umožnil jim pochopit obsah výuky TI lépe než doposud. Podklady přiložené k této práci byly tvořeny autorem BP, tedy studentem, a proto lze předpokládat, že se jedná o subjektivní názor studenta na podobu podpůrných výukových materiálů k předmětu TI. Zároveň ale dokument poskytuje náhled, jak by si někteří studenti představovali vhodné zpracování podkladů k výuce předmětu TI.

10 Bibliografie

[1] EDL, Doc. Ing. Milan, Ph.D. Plánovací a rozhodovací techniky. In: *Přednáška z předmětu PRT* [online]. 2017. Dostupné z: https://portal.zcu.cz/portal/studium/courseware/kpv/prt.

[2] G. O´BRIEN a MICROSOFT. Introduction to C#. *edX* [online]. 2017. Dostupné z: https://www.class-central.com/mooc/8823/edx-introduction-to-c. 2017.

[3] RAŠKA, Ing. Pavel, Ph.D. 04_prednaska_2018_03_08., 2018.

[4] KLAUSEN, Poul. *C# 1 Introduction to programming and the C# language*. B.m.: Poul Klausen & bookboon.com, 2012. ISBN 978-954-400-773-7.

[5] KOPEČEK, Ing. Pavel, CSc. *Okruhy Technické Informatiky*. Kopeček Pavel. 2016.

[6] MAŇÁK, J., KNECHT, P. *Hodnocení učebnic* [online]. 2017. Dostupné z: http://www.paido.cz/pdf/hodnoceni_ucebnic.pdf.

[7] MILES, Rob. *C# Programming Yellow Book*. B.m.: Rob Miles, 2016. ISBN 150-930-115-7.

[8] SANFOUNDRY.COM. C# Examples on Mathematics. *sanfoundry.com* [online]. 2018. Dostupné z: http://www.sanfoundry.com/csharp-programming-examples-on-mathematics/.

[9] SHANU, Syed. *Beginning C# Object Oriented Programming*. B.m.: C# Corner, 2016.

[10] SVETLIN, Nakov. *Fundamentals of Computer Programming with C#*. Sofia: Nakov Svetlin & Co., 2013. ISBN 978-954-400-773-7.

[11] W3RESOURCE.COM. *w3resource* [online]. 2018. Dostupné z: https://www.w3resource.com/csharp-exercises/for-loop/csharp-for-loop-exercise-1.php.

PŘÍLOHA č. 1

Hrubý podpůrný dokument k výuce technické informatiky v programovacím jazyce C#

Podpůrné podklady k výuce Technické Informatiky

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

„EASY-PEASY“

Verze 1.0.6

Autor: Pompl Michael

Rok sepsání: 2018

Obsah:

1 Programovací jazyk C# ... 4
2 OOP ... 5
2.1 Co je objekt? ... 5
2.2 Visual studio (VS) ... 6
2.2.1 CLR ... 7
2.3 První program ... 8
2.3.1 První náš kód ... 9
3 Formulář vs. Konzole ... 11
3.1 WinForms a WPF (GUI) ... 11
3.2 WinForms ... 12
3.3 WPF ... 12
3.4 Rozdíly ... 13
3.5 Výhody ... 13
4 Uvedení do programování ... 14
4.1 Co je to třída ... 14
4.2 Tvoření objektů, konstruktorů ... 15
4.2.1 Konstruktory ... 15
4.2.2 Nový objekt ... 15
4.3 Public, private, internal, protected, static členy ... 17
4.4 Proměnné ... 17
4.5 Typy ... 18
4.5.1 Boolean ... 18
4.5.2 string ... 19
4.5.3 Využití Placeholderů pro možnost tisku ... 19
4.5.4 integer ... 19
4.6 Operátor ... 20
4.6.1 Operátorové zkratky ... 22
4.7 Závorky ... 23
4.8 Přetypování (Casting) ... 25
5 Podmínka IF ... 25
6 Cykly ... 27
6.1 DO – WHILE cyklus ... 27
6.1.1 WHILE cyklus ... 28
6.2 FOR cyklus ... 28

6.3 Vnořené cykly ... 29
6.4 Únik z cyklů ... 30
6.5 Vracení se na začátek cyklu ... 31
7 Výčtový typ (Enumeration) ... 31
8 Switch ... 32
9 Metody ... 34
9.1 Vracení hodnot ... 36
10 Pole ... 36
10.1 Vícedimenzionální pole ... 38
10.2 Foreach ... 38
11 Výjimky ... 39
12 Rekurze ... 40
13 Delegát ... 42
14 List/Dictionary ... 44
14.1 List ... 44
14.2 Obecné typy (Generics) ... 45
14.3 Obecné typy a List ... 45
14.4 Slovník (Dictionary) ... 46
15 Lambda výrazy ... 47
15.1 Lambda výrazy jako Delegát ... 47
16 LINQ ... 48
17 Glosář ... 49
18 Literatura ... 52

Seznam obrázků:

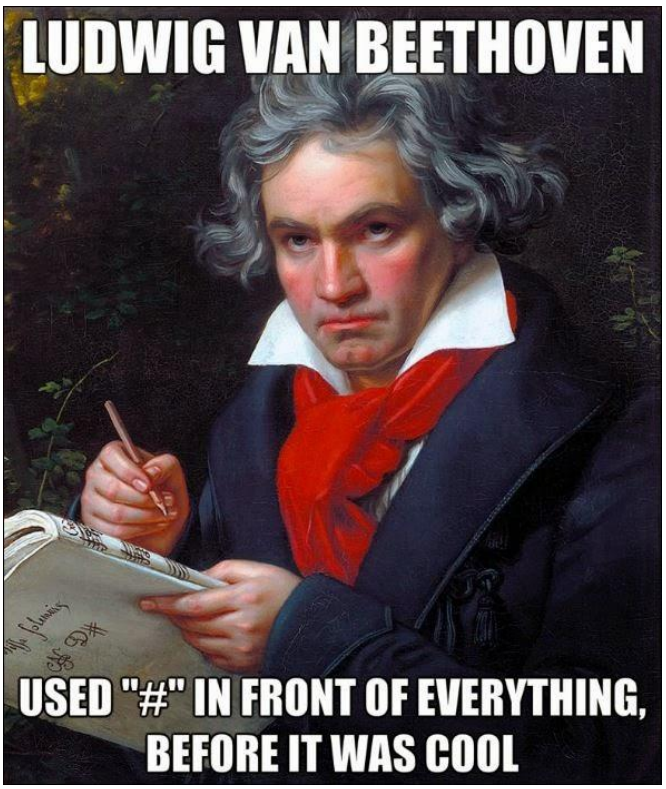
Obr. 1-1 - využívání znaku "#" [6] 4
Obr. 2-1 - startovní okno Visual Studio 6
Obr. 2-2 - znázornění jednotlivých oken VS 7
Obr. 2-3 - vytváření nového projektu 8
Obr. 2-4 - výběr podoby nového programu, přejmenování 9
Obr. 3-1 - ukázka konzolového okna 11
Obr. 3-2 - ukázka vzorového výstupu WPF [11] 13
Obr. 4-1 - příklad na třídu Auto 16
Obr. 4-2 - výsledné konzolové okno pro inkrementaci [1] 23
Obr. 4-3 - nutnost uzavírání bloků [9] 24
Obr. 6-1 – příklad na cyklus FOR [10] 28
Obr. 6-2 – příklad vnořeného cyklu FOR [5] 30
Obr. 10-1 – výstupní konzole příkladu na pole v1, v2 [1] 37
Obr. 11-1 – ukázka špatného způsobu přemýšlení ohledně aplikace výjimek [8] 40
Obr. 12-1 – důležitost uvedení únikové podmínky u rekurzivních funkcí [7] 41

Seznam tabulek:

Tab. č. 4-1 - tabulka s datovými typy [4] 18
Tab. č. 4-2 - druhy operátorů [4] 20
Tab. č. 4-3 - druhy operátorů dle priority [4] 21
Tab. č. 4-4 - příklad operátorových zkratek [2] 22
Tab. č. 10-1 – vizualizace vícedimenzionálního pole [2] 38

1 Programovací jazyk C#

C# je moderní, objektově orientovaný jazyk. Jestli jste někdy udělali chybu, že jste jazyk nazvali C "hash", tak nejspíše jste tento symbol již několikrát zneužili na sociálních sítích. Správný název je C "sharp".



Obr. 1-1 - využívání znaku "#" [6]

C# je velmi flexibilní a mocný programovací jazyk se zajímavou historií. Byl vyvinut firmou Microsoft pro řadu důležitých odvětví lidské společnosti, jako je strojírenství, politika nebo marketingové účely. Při popisování jazyka bychom mohli začít hlavní částí, a to jsou objekty. Objekt je organizační mechanismus, který Vás nechá program "rozkouskovat" do menších dílků, kde každý dílek je součástí celkového systému. Object Oriented Design umožňuje snazší navrhnutí velkých projektů z pohledu designu nebo textu. Umožňuje též tvorbu vysoce spolehlivých a stabilních programů.

C# programy obsahují jednu nebo více složek se soubory s příponou ".cs", které obsahují definice tříd (classes) a ostatních typů (types). Tyto soubory jsou kompilovány pomocí C# kompilátoru (compiler) (csc) na spustitelný kód. Výsledkem jsou sestavení (assemblies), které typicky představují soubor se stejným jménem akorát s jinou příponou (.exe nebo .dll). Například když zkompilujeme HelloWorld.cs, může být vytvořen spustitelný soubor se jménem HelloWorld.exe (popř. další soubory).

Kompilátor je velice rozsáhlý program, který je speciálně napsán pro určitý počítač a programovací jazyk. Kompilátory pracují na principu několika fází. První fází je preprocesor, který si bere zdroj napsaný uživatelem a nadále hledá jednotlivá klíčová sova, identifikátor a symboly vytvářející podstatu zdroje programu. Ten je poslán do překladače, který zajistí, že zdroj dodržuje zásady využívaného programovacího jazyka. Poseldní fáze obsahuje generátor kódu, který vytváří spustitelný soubor, který bude spouštěn uživatelem.

Kompilátor kontroluje chyby. Pokud kompilátor žádné nenajde, vyprodukuje určitý výstup (output). Kompilátor také zvýrazní varování kódu, který není chybný, avšak po spuštění programu může v určité části tohoto kódu dojít k chybě anebo je navíc.

Počítač nemůže rozumět námi psanému jazyku přímo, a proto je volán kompilátor, který převede C# text na nízko-úrovňové instrukce. Ty jsou pak převedeny do příkazů řídícího hardwaru, na kterém je spuštěn program. Kompilovaný kód může být spuštěn jako každý jiný spustitelný program na počítači (dvojklikem). Jestliže se pokusíme zpustit zkompilovaný C# kód na počítači, kde není nainstalován .NET Framework, tak jedinou zobrazenou věcí bude zpráva ohlašující chybu (errormessage).

Aspekty, jak C# programy pracují, budou více rozvedeny v dalších částech tohoto dokumentu. Nyní jedinou věcí k zapamatování je to, že musíte svůj nádherný funkční C# program ukázat kompilátoru (compiler) než bude spuštěn.

C# jazyk je nadále vybaven mnoho dalšími "vychytávkami", které programům umožňují věci, jako čtení textu z klávesnice, tisk textu na displej a další.

2 OOP

Objektivně orientované programování (OOP) je logický přístup, který využívá objektů, vlastností a tzv. metod k utvoření počítačového programu.

2.1 Co je objekt?

Některé věci, které píšeme v programování, jsou objekty, které jsou součástí softwarové knihovny (frameworku), kterou využíváme.

Softwarové objekty modelují objekty z reálného světa nebo jejich abstraktní koncepty. Jako příklad reálného objektu jsou lidé, auta, psi atd. Abstraktní objekty můžou být koncepty v oblasti objektů, kterou musíme vytvořit a použít v našem počítači. Příklad abstraktního objektu může být struktura (structure), řada (queue), list nebo strom (tree). Mezi objekty se rozlišují dvě skupiny dle jejich charakteristik [4]

- Stavy (states) – charakteristiky objektů, které je definují a popisují v obecném nebo specifickém momentu.
- Chování – specifické akce, které můžou být provedeny objektem

Vezme-li jako příklad reálného objektu "pes". Stavy objektu “pes“ může být jméno, barva srsti nebo plemeno a jeho chování může být štěkání, chůze nebo jiná aktivita.

Objekty v OOP slučují data a náležitosti k jejich společnému zpracování. Odkazují se na objekty v reálném světě a obsahují data a akce:

- Datové členy – vestavěny do variabilních, které popisují jejich Stavy
- Metody – nástroje k vytvoření objektů

2.2 Visual studio (VS)

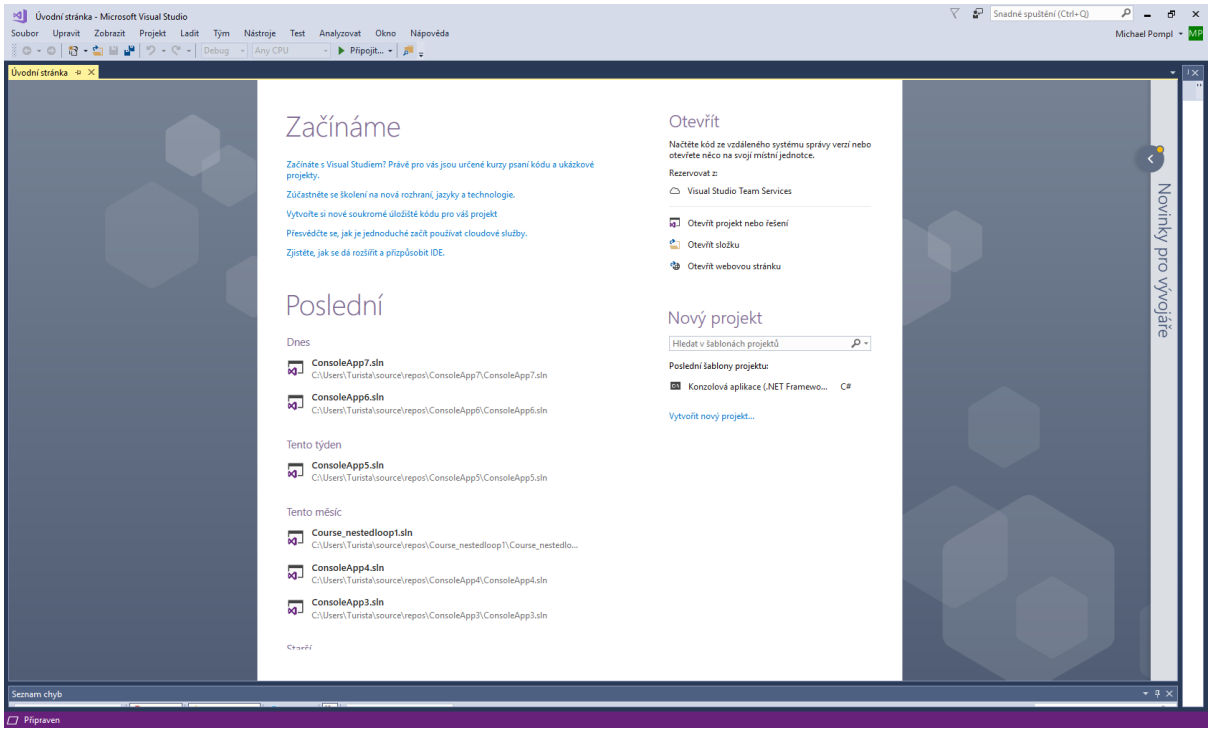
Microsoft vyvinul nástroj s názvem Visual Studio (dále VS), kde lze lehce psát program. Obsahuje kompilátor spolu integrovaným editorem a debuggerem.

Debbuger slouží k opravě kódu, pokud byla zobrazena chyba nebo byl očekáván jiný výsledek. Pomocí debuggeru lze procházet napsaný kódem, hledat případné chyby a následně je opravit. Pomocí „breakpointů“, které říkají debuggeru, na jaké řádce se má průběh kontroly pozastavit, lze zobrazit průběžnou hodnotu proměnných a posoudit zda v některém z kroků nenastala chyba.

Programovací jazyk C# by neměl být brán jako samostatná „jednotka“, ale jako jedna z částí Microsoft .NET Framework platformy, která obsahuje prostředí k vývoji a spouštění programů napsaných v C# jayzce nebo jiném kompatibilním jazyce s platformou .NET. Obsahuje .NET programovací jazyky, CLR, vývojové nástroje a knihovny.

VS je integrované prostředí (IDE – integrated environment) pro vývoj softwarových aplikací Windows a .NET Framework platforem. IDE slouží jako nástroj k psaní kódu, ke kompilaci, ke spuštění programu, testování nebo k debuggování. Vše je zahrnuto na jednom místě. VS podporuje různé druhy programovacích jazyků (C#, VB.NET nebo C++) a různé druhy technologií pro vývoj softwaru (Win32, COM, ASP.NET, ADO.NET Entity Framework, Windows Forms, WPF, Silverlight, a další). Při kompilaci je kód překládán do jazyka CIL (Common Intermediate Language). Díky této kompilaci lze kód spustit v jakémkoliv prostředí, které podporuje CLI (Common Language Infrastructre), kteým jsou již zmíněné programovací jazyky. CLI je v informatice nejnižší člověkem čitelný programovací jazyk definovaný specifikací CLI používaný projekty .NET Framework.

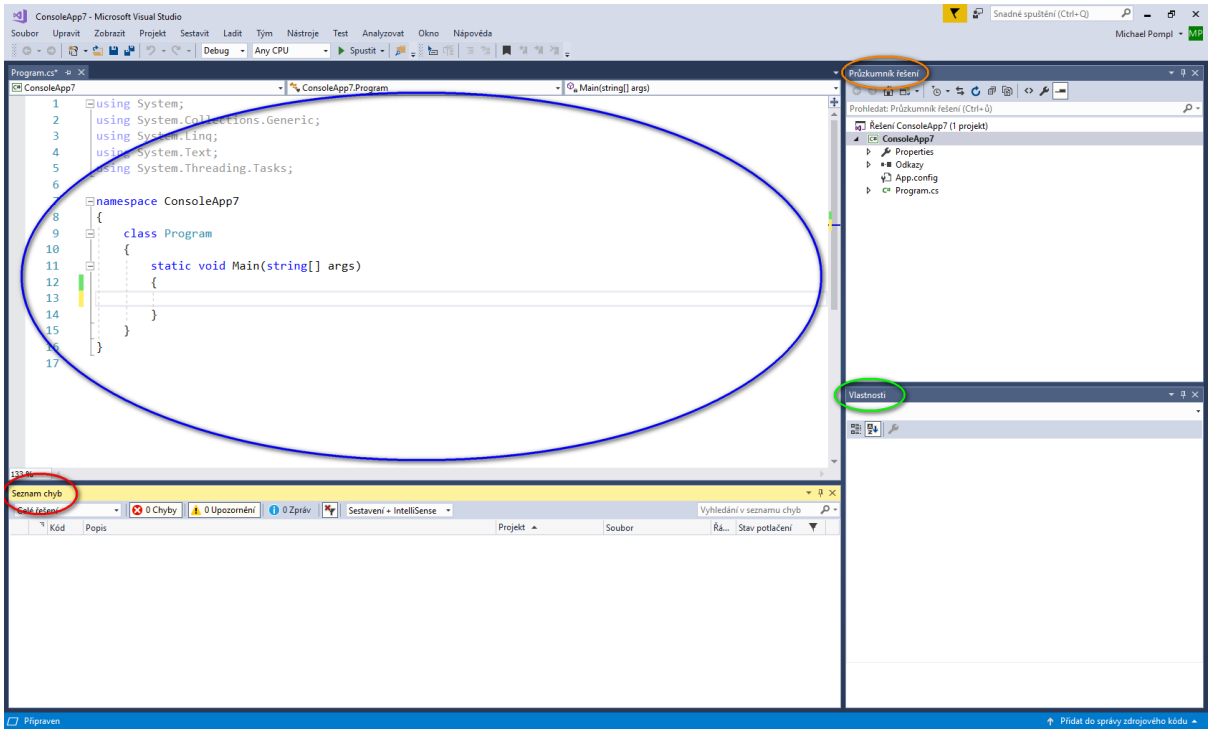
Než začneme s vytvářením programu, bylo by dobré si představit základní uživatelské rozhraní VS (UI - user interface). Na *Obr. **2-1*** je k vidění, jak VS vypadají po běžné instalaci a následné konfiguraci:



Obr. 2-1 - startovní okno Visual Studio

VS má několik oken, které si teď popíšeme:

- Startovní stránka (start page) - odtud můžeme snadno otevírat jakýkoliv z našich aktuálních projektů nebo vytvořit zcela nový, viz *Obr. 2-1*.
- Kód editor (code editor) – umožňuje editaci zdrojového kódu programu a umožňuje otevírání a editování několika složek.
- List chyb (error list) - ukazuje seznam chyb v programu, který jsme vytvořili.
- Průzkumník řešení (Solution Explorer) – jakmile žádný projekt není nahrán, Explorer je prázdný. Jakmile bude nahrán určitý projekt, ukáže nám strukturu projektu tzn. Všechny složky obsažené v projektu bez ohledu na to, zda se jedná o kód, obrázek čí jiný typ kódu nebo zdroje.
- Vlastnosti - obsahují list aktuálních vlastností objektu. Jsou používány převáženě v komponentně založeném programování (componet-based). Tím se myslí vývoj WPF, ASP.NET nebo WinForms aplikací.



Obr. 2-2 - znázornění jednotlivých oken VS

2.2.1 CLR

CLR (common language runtime) je prostředí, které je obsaženo v .NET Framework. CLR zprostředkovává "spuštění kódu" a servis, který usnadňuje developerům práci při pasní kódu.

CLR umožňuje snadné upravování komponent a aplikací, které fungují v rámci vícero programovacích jazyků. Objekty, které jsou napsány v jiném programovacím jazyku, mohou komunikovat s objekty v jazyce právě využívaném. Mezi-jazyková komunikace je umožněna díky jazykovým kompilátorům. K umožnění funkce runtime je nutné, aby jazykové kompilátory poskytovaly metadata poskytující popis typů, členů nebo referencí v našem kódu. Metadata jsou využívána runtime funkcí k nalezení a následnému nahrání tříd, zajišťování bezpečnosti našeho kódu a nastavení runtime kontextových vazeb.

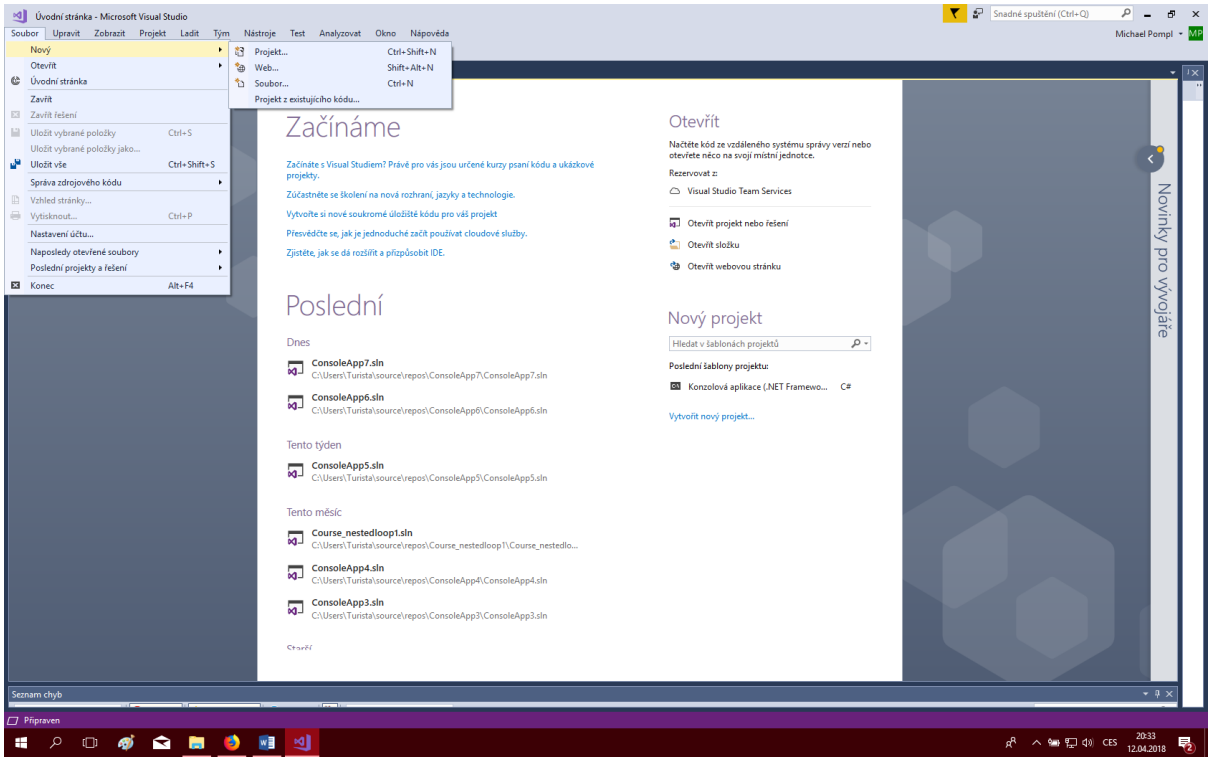
2.3 První program

První věcí, kterou se začíná, je vytvoření nového projektu v prostředí VS. Existuje i možnost nahrát již existující projekt.

Projekt je ucelené logické uspořádání jednoho nebo více souborů s kódem (z důvodů přehlednosti mohou být tyto soubory organizovány ve složkách). Je doporučeno vytvoření nového projektu pro každý nový program.

Projekt se může vytvořit ve VS následujícími kroky:

- Soubor (File) -> Nový -> Projekt (New project)

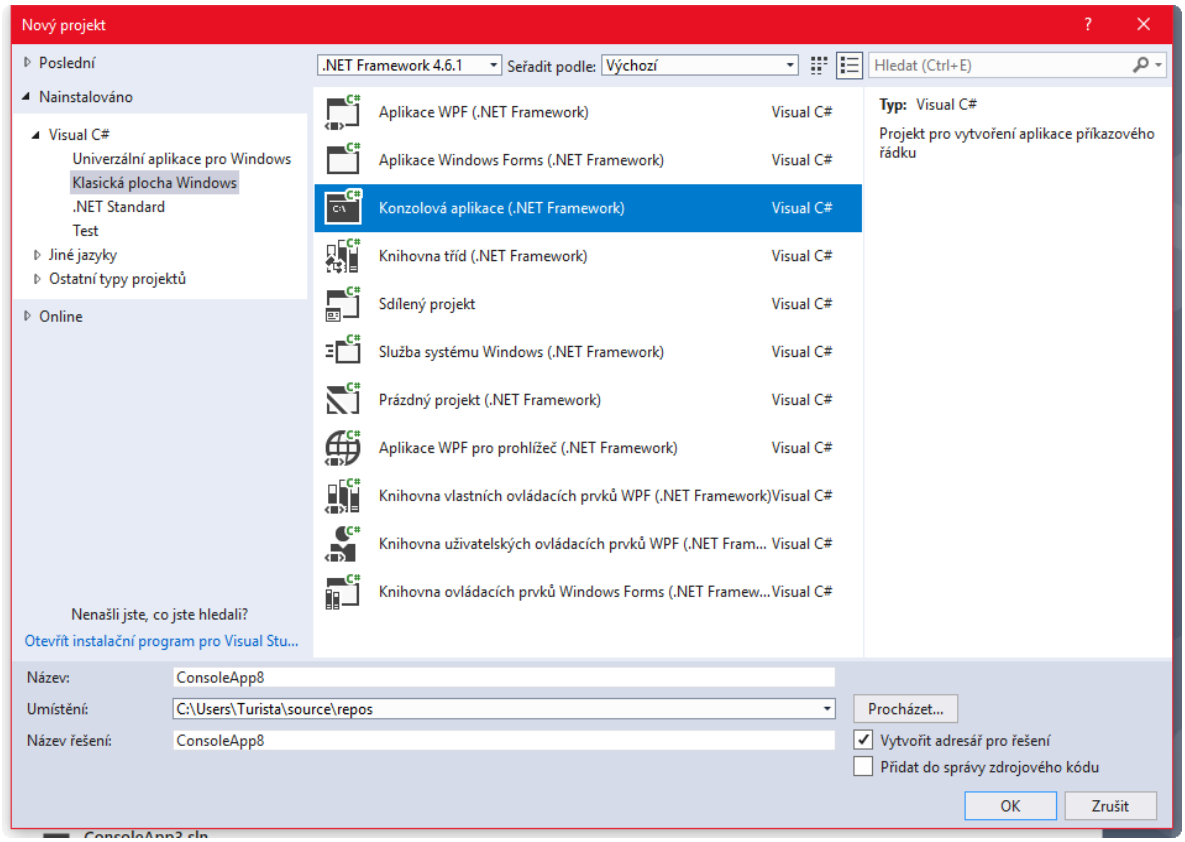


Obr. 2-3 - vytváření nového projektu

Vybere se požadovaná aplikace (konzole, forms, WPF a další), v tomto případě konzolová aplikace, což je program, který využívá konzoli jako výchozí vstup i výstup (input, output). Data jsou vložena pomocí klávesnice a po zadání příkazu vytištění výsledku se v konzoli objeví.

Přejmenovat projekt lze následovně:

- Přejmenovat (Rename) -> zde je možno přejmenovat aplikaci na požadované jméno např. z "ConsoleApp4" na "AhojSvete"
- Soubor -> Uložit ConsoleApp4.cs jako ...



Obr. 2-4 - výběr podoby nového programu, přejmenování

Pozn.: Vytvořený projekt se založí to tzv. Solution – v rámci Solution totiž můžeme řídit více projektů. Při tvorbě „větších“ aplikací je to běžná praxe.

2.3.1 První náš kód

(1) Než se začne s podrobnějším popisováním C# jazyka a .NET platformy, bylo by dobré ukázat jednoduchý příklad, který ilustruje, jak program v C# jazyce vlastně vypadá [2]:

```
using System
class PrvniProgram
{                                   // začátek třídy
static void Main()
        {
        Console.WriteLine("Ahoj světe");
        }
}                                   // konec třídy
```

Jedinou věcí co tento program dělá, je vytištění zprávy "Ahoj Svete" na výstupu (konzoli). Je ale stále příliš brzo k jeho spuštění, a proto se nejdříve nahlédne jen do jeho struktury.

Jak náš první program vlastně funguje?

Program obsahuje tři základní logické části:

- Definice třídy AhojSvěte
- Definice metody Main()
- Obsah metody Main()

Definice třídy

Na první řádce našeho programu definujeme třídu zvanou PrvniProgram. Nejjednodušší definice třídy obsahuje klíčové slovo třída (class), kde následuje její jméno. Obsah třídy se nachází v bloku ohraničeného vlněnými závorkami {…} (curly brackets).

Definice metody Main()

Na třetí řádce definujeme metodu s názvem Main(), která je počátkem pro náš program. Každý program napsán v C# začíná od metody Main() s následujícím typickým zápisem:

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

Metoda musí být deklarována, jak je ukázáno výše. Musí být označena jako static a void, musí mít jméno Main. List parametrů může mít pouze jeden parametr typu pole (array) nebo string (textový řetězec). V našem případě je parametr nazván args. Tento název není povinný a může se lišit nebo být i vypuštěn. V tom případě počátek programu může být zjednodušen a vypadá takto:

```
static void Main()
```

Jestliže některá ze zmíněných podmínek není splněna, program bude zkompilován, ale nebude funkční, protože počátek programu není definován správně.

Obsah metody Main()

Obsah každé metody je k vidění za jejím názvem opět ve vlnitých závorkách {…}. Na další řádce našeho programu se použije systémový objekt System.Console a jeho metoda WriteLine() (napiš řádek) k vytištění zprávy na výchozí výstup (konzoli). V našem případě nám konzole ukáže vytištěný text "Ahoj Světe". Z důvodu využití zápisu “using System“ není nutno před metodu již psát System.

V metodě Main() můžeme napsat náhodnou sekvenci výrazů, která bude následně spuštěna ve stanoveném pořadí.

Více informacích o třídách a metodách se dočtete dále v dokumentu.

Poslední důležitá informace pro začátek:

C# rozlišuje velká a malá písmena v názvech! Existuje několik konvencí, dle kterých by se měl řídit každý programátor, aby kód byl pro ostatní čitelný.

System

Z této knihovny si program může vzít předměty v ní obsažené, jako je například metoda Console. Kdyby na začátku programu nebylo deklarováno ‘‘using‘‘ + název knihovny, musel by zápis vypadat následovně:

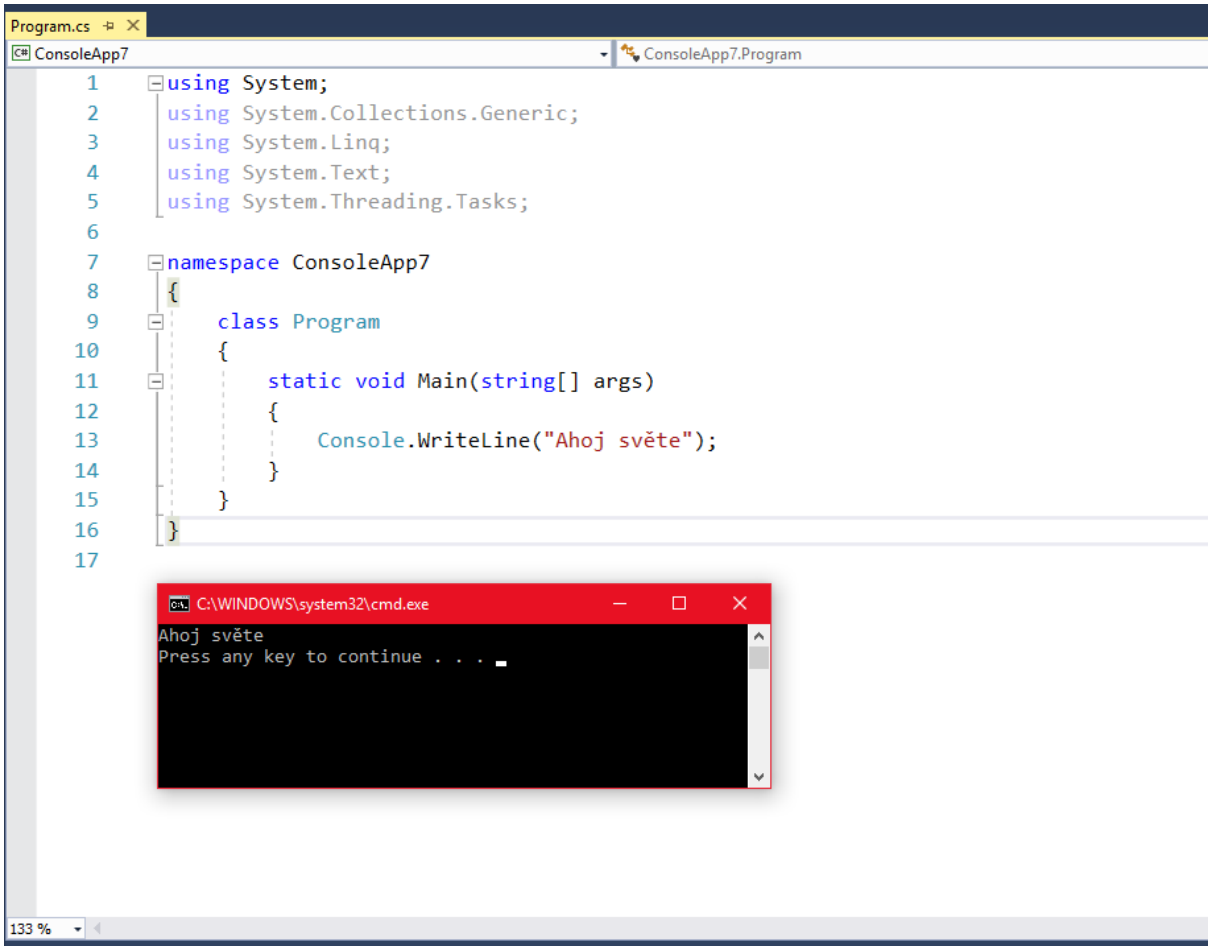
```
System.Console.WriteLine(" ");
```

Lomítka //

Dvě lomítka ihned za sebou znamenají komentář. Označená část kódu slouží pouze jako informace pro developera/programátora.

3 Formulář vs. Konzole

Konzole je okno operačního systému, přes které uživatel může komunikovat s programy systému nebo dalšími konzolovými aplikacemi. Interakce obsahuje vstupní text zadaný přes vstup (klávesnicí) nebo text zobrazený na standardním výstupu, jako je počítačová obrazovka. Tyto akce jsou známy jako vstup-výstup (input/output) operace. Text napsán v konzoli přináší určitou informaci a je zároveň sekvencí charakterů (chars), kterou nám posílá jeden nebo více programů. Mnoho programů komunikuje určitým směrem se svým uživatelem, který má povinnost programu zadávat určité instrukce. V dnešní době existuje spousta dalších „lepších" komunikačních metod (metoda komunikace grafickou cestu, webově založené rozhraní nebo konzole). Počítačově založené programy využívají ke komunikaci grafické uživatelské rozhraní.



Obr. 3-1 - ukázka konzolového okna

3.1 WinForms a WPF (GUI)

Co vlastně grafické uživatelské rozhraní je?

Z anglického významu Graphical User Interface, ve zkratce GUI, vyplývá, že se jedná o komunikaci mezi uživatelem a určitým programem. GUI umožňuje vytvářet aplikace se širokou škálou prvků využívaných právě tímto rozhraním. Jedná se popisky (labels), textové boxy a dalšími prvky. Bez GUI by se všechny tyto prvky musely vytvářet/malovat ručně.

3.2 WinForms

WinForms je grafické rozhraní, které zobrazuje data a umožňuje lehčí a bezpečnější interakci uživatele s programem. Poskytuje několik užitečných prostředků, jako jsou text-boxy, tlačítka a možnost vytvoření vlastního specifického prostředku pro ovládání aplikace. WinForms obsahuje také třídy pro definici fontů, ikon a dalších grafických objektů. WinForms Designer je nástroj, který ve VS umožňuje vložení a umístění například tlačítka do námi vytvořeného layoutu. Jediná věc co zbývá, je naprogramovat tyto prvky (tlačítka, …) tak, aby fungovaly, jak je od nich vyžadováno.

3.3 WPF

WPF (Windows PresentationFoundation) je posledním výtvorem společenosti Microsoft, jak zlepšit GUI framework. Windows má své vlastní GUI, které funguje na každém počítači, aby například uživatel mohl počítač vypnout přes nabídku Start a ne přes hardwarové tlačítko. Své vlastní GUI mají i webové prohlížeče, které pak umožňují "surfovat" po webu.

Nyní si popíšeme jak takové WPF vytvořit ve VS:

- Vytvořte Nový Projekt z menu Složka (File)

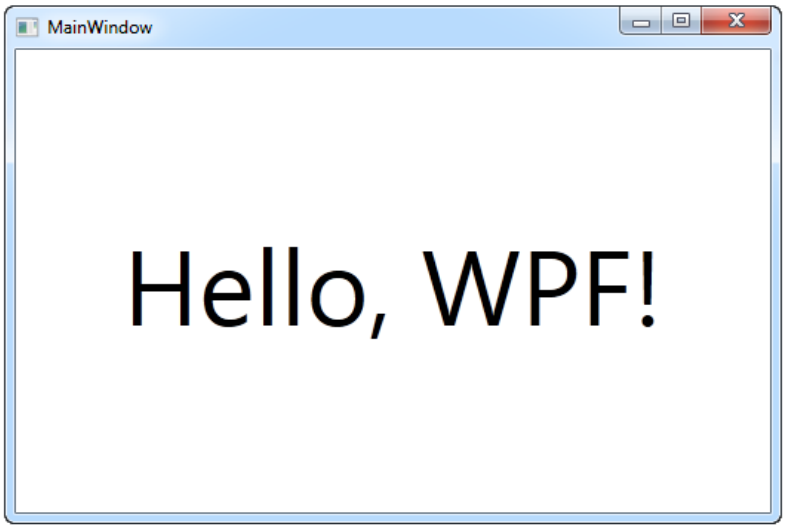
Na levé straně by se měl nacházet strom s kategoriemi.

- Vyberte možnost Windows z tohoto stromu
- Objeví se nabídka možností k vytvoření různých Windows aplikací - vyberte WPF Aplikace
- Pojmenujte svůj nový projekt například "HelloWPF" v textovém poli Jméno (Name)
- Ujistěte se, že všechna nastavení ve spodní části dialogového okna jsou v pořádku a stiskněte tlačítko OK

Napište do editoru kódu následující kód [11]:

```
<Window x:Class =“WpfApplication1.MainWindow“
        xmlns =“http://schemas.microsoft.com/winfx/2006/xaml/presentation“
        xmlns:x =“http://schemas.microsoft.com/wnfx/2006/xaml“
        Title =“MainWindow“ Height =“350“ Width =“525“>
        <Grid>
           <TextBlock HorizontalAligment =“Center“ VerticalAlingment =“Center“ FontSize =“72“>
                Hello, WPF!
           </TextBlock>
        <Grid>
</Window>
```

Očekávaným výdledkem by mělo být takovéto okno:



Obr. 3-2 - ukázka vzorového výstupu WPF [11]

3.4 Rozdíly

Nejzřetelnější a nejvíce důležitý rozdíl mezi WinForms a WPF je fakt, že WinForms je jen pouhá vrstva na povrchu "Windows standard controls" (např. Text-box, tlačítko, a dále). WPF je stavěno od základu a nespoléhá se na "Windows standard controls" skoro ve všech případech. Dobrým příkladem může být tlačítko, na kterém je obrázek s textem. WinForms by toto tlačítko nedokázal vytvořit, ale v případě WPF se tlačítko musí namalovat a nastavit způsobem, aby přijímalo obrázek na svůj povrch nebo se dalo využití třetí stranou. Ve WPF může tlačítko obsahovat téměř cokoliv.

3.5 Výhody

WPF

- Je novější, a proto více vyhovuje dnešním standardům
- Microsoft WPF využívá ve velké míře ve svých nových projektech (nové verze VS)
- Je více flexibilní, možnost dělat více věcí bez potřeby tvoření nebo koupí nových ovladačů
- Databinding - lepší synchronizace dat a uživatelského rozhraní
- Využívá hardware k tvoření GUI, vyšší výkonnost
- Umožňuje vytvořit jak rozhraní pro Windows tak i pro webové aplikace

WinForms

- Je starší, a proto již řádně otestovaná možnost tvorby rozhraní
- Existuje již spousta ovladačů vydaných třetí stranou, které lze koupit nebo obdržet zdarma
- WinForms Designer je lépe ovladatelný než u WPF

4 Uvedení do programování

Úvodem by bylo dobré si blíže vysvětlit pojmy třída a objekt.

4.1 Co je to třída

Třída definuje abstraktní charakteristiky objektů. Poskytuje strukturu pro objekty nebo modely, kterými je popsána určitá povaha objektu.

Třídy staví bloky a vztahují se k objektům. Každý objekt je instance specifických tříd. C# program se skládá z jedné nebo více tříd. Třída může být přirovnána ke kontejneru, který obsahuje určitá data nebo kód k provedení určitého výkonu.

Jako příklad může být využita třída Pes a jméno Franky je instance třídy Pes. Třída Pes popisuje charakteristiku všech psů, ale Franky je pouze jeden určitý pes.

Charakteristiky musí dávat smysl v obecném kontextu, aby člověk, který není "v obraze", porozuměl danému problému. Například třída Pes nemůže mít charakteristiku Koza, protože v kontextu třídy Pes by tato charakteristika nedávala žádný smysl. Chování objektů je tvořeno metodami uvnitř třídy.

Co je to objekt

Objekt je charakterizován těmito základními atributy:

- Unikátním identifikátorem, který definuje určitý objekt ve vztahu vůči všem ostatním
- Aktuální hodnota objektu
- Chování, které definuje, co se s objektem může provést
- Životní cyklus, kde objekty tvořeny a také mazány

Objekt je vytvořen ze třídy a v ten okamžik je mu přiřazen odkaz (reference), který objekt identifikuje. Odkazy mají svá specifická jména dle objektu, na který se odkazují. Slouží k nalzení cesty k instancím třídy a využití jejích metod a dat.

"Ahh", a ještě jedna věc. Existuje určitá konvence, která říká, že jméno určité třídy by se mělo shodovat se jménem souboru, ve kterém se nachází definice třidy (konvence dále doporučuje, aby byl pro jednu dílčí třídu využit jeden soubor se jménem třidy). Z hlediska třídy Pes by se měla složka tedy jmenovat Pes.cs.(1)

4.2 Tvoření objektů, konstruktorů

V této části bude definován pojem konstruktor a následně jakým způsobem se tvoří objekty.

4.2.1 Konstruktory

Konstruktor je speciální metoda třídy, která je automaticky zavolána při vytváření nového objektu třídy. Plní také funkci inicializace dat objektu. Konstruktor nevrací žádný typ. Jméno konstruktorů není vybíráno náhodně, ale shoduje se s názvem třídy. Konstruktor může být definován s parametry, ale není to podmínkou. Jakmile parametry nejsou vloženy, jedná se o tzv. bezparametrový konstruktor.

4.2.2 Nový objekt

Nyní se zaměříme na vytváření a používání objektů v našem programu. Budeme pracovat již s existujícími třídami a převážně se systémem tříd z .NET Frameworku. Tvoření objektů z předběžně vytvořených tříd během zpuštění programu se uskuteční pomocí operátoru "new" (nový). Nově objekt je obvykle přiřazen k typu proměnné shodného s třídou objektu.

```
Auto nejakeAuto = new Auto();
```

Nyní vytvoříme konstruktor s parametry model a značka na základě příkladu (viz *Obr. 4-1*). Volání konstruktoru bude vypadat takto:

```
Auto nejakeAuto = new Auto("S", "Tesla");
```

V tomto případě je požadováno po objektu “nejakeAuto“, aby zastupoval určité auto značky "Tesla", modelu "S". Jak vyplývá z příkladu, toto zastoupení je dáno vloženými parametry do konstruktoru.

Přístup k vlastnostem (properties) daného objektu je umožněn operátorem "." (tečka) umístěného mezi jméno objektu a jméno pole nebo vlastnosti. Je možné vstupovat do polí a vlastností, jak z důvodu extrakce dat, tak jejich importování. Toho je docíleno klíčovými slovy "get" a "set" v definici vlastnosti, která provádí extrakci hodnoty a přiřadí hodnotu novou. V případě výše popsané třídy Auto se jedná o vlastnosti Model a Znacka.

```
string model = nejakeAuto.Model
```

```
public class Auto
{
    // pole s modelem
    private string model;

    // pole se značkou
    private string znacka;

    public string Model
    {
        // dostaň vlastnost od "model"
        get
        {
            return this.model;
        }


        // přepiš vlastnost pro "model"
        set
        {
            this.model = value;
        }
    }

    public string Znacka
    {
        // dostaň vlastnost od "znacka"
        get
        {
            return this.znacka;
        }


        // Setter of the property "znacka"
        set
        {
            this.znacka = value;
        }
    }


    // Výchozí konstruktor
    public Auto()
    {
        this.model = "Neuvedeno";
        this.znacka = "Neznámá";
    }


    // Konstruktor s parametry
    public Auto(string model, string znacka)
    {
        this.model = model;
        this.znacka = znacka;
    }

}
```

Obr. 4-1 - příklad na třídu Auto

4.3 Public, private, internal, protected, static členy

Public znamená, že člen (typicky metoda nebo vlastnost) je viditelný i z jiných tříd.

Private má význam takový, že člen může být využit pouze metodami ze stejné třídy. Private člen je viditelný pouze metodám uvnitř dané třídy. Využívá se tehdy, pokud nechceme, aby metoda z jiné třídy mohla tento člen měnit.

Internal znamená, že člen může být využit všemi metodami ve stejném sestavení (assembly).

Protected člen třídy je viditelný metodám ve třídě a třídám rozšiřujícím tuto třídu. Je to v podstatě něco mezi Private členem a Public členem. Umožňuje práci s členy v rodičovské třídě (parentclass) a viditelnost změn v potomcích (childclasses).

Static se v C# deklarují převážně metody, konstruktory nebo třídy. Klíčové slovo static deklaruje, že element je spíše členem části třídy než částí její instance. Statické elementy třídy mohou být využity, aniž by musel být vytvořen objekt dané třídy. Klíčové slovo static je využíváno v případě, že člen třídy má být sdílen všemi instancemi třídy.

4.4 Proměnné

Aplikace musí pracovat s daty. Aby to bylo proveditelné, potřebují určitý způsob, jak data uchovat. Pro tento účel mají programy proměnné, které poskytují možnost uchování hodnoty. Proměnné jsou definovány:

- Jménem
- Typem
- Operátorem

Proměnné musí mít jméno, aby se na ně mohlo odkazovat v programu C#. Proměnné mají typ, který indikuje, jaké hodnoty mohou být v nich uloženy. Zároveň je z jejich typu jednoznačně patrné, jak moc tato proměnná bude zatěžovat paměť. Typ také určuje, jaké operace mohou být s proměnnou provedeny.

Jakým způsobem by se proměnné měli pojmenovávat:

- Jméno proměnné by mělo začínat vždy malým písmenem
- Jméno nesmí obsahovat žádné mezery anebo speciální znaky
- Nelze začít číslem, číslo však může pojemnování proměnné obsahovat

Pokud se budou dodržovat tyto jednoduchá pravidla, nenastane nikdy problém s pojmenováním proměnných.

Důležitý je také fakt, že proměnná musí být nejdříve vytvořena, než bude použita. Lze vidět z následujícího kódu (obecně):

```
type jmeno = value;
```

Jako první se píše typ, poté jméno proměnné, a nakonec je přiřazena hodnota např.:

```
int cislo = 11;
```

Zde je deklarována proměnná nazvaná "číslo", která má typ int a hodnotu 11. Proměnné by měly být vždy využity, jinak se obdrží chyba při překladu, což je nežádoucí.

4.5 Typy

Základní datové typy v C# jsou rozděleny do následujících skupin:

- Integer typ – sbyte, byte, short, ushort, int, uint, long, ulong
- Reálný typ s plovoucí řádovou čárkou - float, double
- Reálný typ s desetinou přesností - decimal
- Boolean typ – bool
- Charakterový typ – char
- String (textový řetězec) - sting
- Objektový typ - object

Tyto datové typy jsou nazývány primitivními, protože jsou vztaženy k nejnižší hladině C# jazyka.

Tabulka s datovými typy s jejich velikostí a výchozími hodnotami:

Data Types	Default Value	Minimum Value	Maximum Value
sbyte	0	-128	127
byte	0	0	255
short	0	-32768	32767
ushort	0	0	65535
int	0	-2147483648	2147483647
uint	0u	0	4294967295
long	0L	-9223372036854775808	9223372036854775807
ulong	0u	0	18446744073709551615
float	0.0f	$\pm 1.5 \times 10^{-45}$	$\pm 3.4 \times 10^{38}$
double	0.0d	$\pm 5.0 \times 10^{-324}$	$\pm 1.7 \times 10^{308}$
decimal	0.0m	$\pm 1.0 \times 10^{-28}$	$\pm 7.9 \times 10^{28}$
bool	false	Two possible values: **true** and **false**	
char	'\u0000'	'\u0000'	'\uffff'
object	null	-	-
string	null	-	-

Tab. č. 4-1 - tabulka s datovými typy [4]

4.5.1 Boolean

Boolean, ve zkratce bool, je proměnná, která ukládá informaci, zda tvrzení je pravda nebo ne. Pokud se ukládá, zda bylo předplatné zaplaceno nebo ne, tak není nutno zbytečně využívat typy, které by zabíraly příliš mnoho místa v paměti. Dostatečným je pouze typ, který ukládá True/False - boolean. True/False jsou jediné hodnoty, které lze v typu boolean ukládat.

Příklad boolean proměnné na internetové připojení:

```
bool internetOK;

networkOK = true ;
```

4.5.2 string

String proměnné můžeme přirovnat k úložišti, které může uchovat textový řetězec (string). V C# může být string velmi krátký např. "Ota" nebo delší např. "Ota je strojař". Proměnná string může uložit řádku textu. Nicméně, je i možné, aby string ukládal i velké množství řádek textu.

4.5.3 Využití Placeholderů pro možnost tisku

Placeholder pouze označuje místo kam bude hodnota vytištěna pomocí závorek {}. Např. [2]:

```
int i = 50 ;
double f = 12.5899;
Console.WriteLine ( "i: {0} f: {1}", i, f ) ;
Console.WriteLine ( "i: {1} f: {0}", f, i ) ;
```

Výstup bude vypadat takto:

```
i: 50 f: 12.5899

i: 50 f: 12.5899
```

Jak je vidět z příkladu výše, namísto {X} byly vytisknuty výsledné hondoty v daném pořadí (funkce placeholderů). V druhém tištění textu bylo přehozeno pořadí čísel, ale zároveň bylo přehozeno i pořadí, v jakém byly parametry, a proto výstup je stejný jako u předchozího tisku.

Jestli by se udělalo něco bláznivého, jako umístění {99} namísto {1}, metoda WriteLine by proběhla, avšak ve výstupu by bylo ukázáno chybové okénko a program “spadne“. Kompilátor tento druh chyby nezaregistruje.

4.5.4 integer

Když se berou v potaz numerické hodnoty, jsou zde na výběr dva druhy dat:

- Pěkně rozkouskované hodnoty, např. počet krabic na paletě nebo jablek v košíku. Nazývají se integer (int), atd.
- Ošklivá čísla z reálného světa, jako např. aktuální teplota nebo rychlost auta. Nazývají se reálnými (reals)

V prvním případě může být hodnota uložena přesně. Vždy je k dispozici přesný počet čísel těchto položek.

V druhém případě nemůže být nikdy uloženo přesně "na co se zrovna díváme". I když délka textu (string) bude změřena na 100 míst, stále je nemožné vrátit přesnou délku onoho textu. Počítač je schopen ukládat reálné hodnoty pouze s limitovanou přesností.

Číslo (int, atd.) je nejsnazší typ hodnoty, který počítač dokáže uložit. Jediný problém je jeho délka. Čím větší je hodnota, tím více bitů je potřeba k jeho reprezentaci.

4.6 Operátor

Každý programovací jazyk využívá operátory, skrze které lze provádět operace s daty. Pojďme se podívat na tyto operátory a zjistit k jakému účelu slouží a jak se s nimi pracuje.

Operátory umožňují zpracování primitivních datových typů a objektů. Berou si jako vstup jeden nebo více operandů a vrací hodnotu jako výsledek. Operátory jsou využívány ke specifikaci operace, která má být vykonána operandy. Většina operátorů pracuje na bázi dvou operandů. Operátory v C# jazyce jsou speciální znaky ("+", ". ", "^", a další) a provedou změnu v jednom, dvou nebo třech operandech najednou. Příkladem může být násobení, dělení a operace, které provádí s číselnými typy (int, atd.) a reálnými čísly.

Category	Operators
arithmetic	`-, +, *, /, %, ++, --`
logical	`&&, ||, !, ^`
binary	`&, |, ^, ~, <<, >>`
comparison	`==, !=, >, <, >=, <=`
assignment	`=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=`
string concatenation	`+`
type conversion	`(type), as, is, typeof, sizeof`
other	`., new, (), [], ?:, ??`

Tab. č. 4-2 - druhy operátorů [4]

C# má následující operátory seřazeny podle jejich priorit následovně:

Priority	Operator
Nejvyšší priorita	(,)
	++, -- (přípony), new, (type), typeof, sizeof
	++, -- (předpony), +, - , !, ~
	*, /, %
	+ (textový řetězec)
	+, -
...	<<, >>
	<, >, <=, >=, is, as
	==, !=
	&, ^, |
Nejnižší priorita	&&
	||
	?:, ??
	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=

Tab. č. 4-3 - druhy operátorů dle priority [4]

Priority jsou důležité pouze tehdy, pokud je využit více než jeden operátor.

a + b*c

Násobení je vykalkulováno jako první, protože operátor "*" má vyšší prioritu než operátor "+". Když ale bude příklad dán takto:

(a + b) * c

bude jako první spočtena závorka, respektive její obsah. Většinou užití operátorů není nijak složité, ale některé potřebují trochu naší pozornosti, abychom věděli, jak je správně využít. Například operátor "++" bude vysvětlen na příkladu:

```
int a = 10 + 5;

a  ++ ;              // zde zvýšíme proměnnou o 1
```

Krátké vysvětlení dalších (logických) operátorů [2]:

==

Rovnost. Jestliže levá strana a pravá strana jsou si rovny, výraz má hodnotu True (Pravda). Jestli tomu tak neni, strany si nejsou rovny a výraz bude mít hodnotu False (Nepravda).

!=

Nerovnost. Opačný příklad rovnosti. Jestliže operandy si nejsou rovny, výraz bude mít hodnotu True, jinak False.

!

"Není". Tento operátor může být využit k převrácení určité hodnoty výrazu, například když víme, že výraz bude hodnoty True, ale my potřebujeme False. Jednoduše napíšeme !(x == y) za předpokladu že se operandy x a y sobě rovnají.

&&

"A". Jestliže operandy na obou stranách tohoto operátoru jsou True, tak poté i celkový výsledek/hodnota bude True, ale pokud alespoň jedna strana bude mít hodnotu False tak i celkový výsledek bude mít hodnotu False.

||

"Nebo". Má podobnou funkčnost jako operátor "A". Když jeden z operandů bude False, tak vrátí hodnotu operandu, který je True. Aby operátor "Nebo" vrátil hodnotu False, musí být oba operandy hodnoty False.

4.6.1 Operátorové zkratky

Možnost zkrátit inkrementaci zápisu:

```
window_count = window_count + 1
```

Na zápis:

```
window_count++
```

Oba dva zápisy jsou funkčností zcela stejné.

a += b	hodnota nahrazena pomocí	a + b
a -= b	hodnota nahrazena pomocí	a – b
a /= b	hodnota nahrazena pomocí	a / b
a *= b	hodnota nahrazena pomocí	a * b

Tab. č. 4-4 - příklad operátorových zkratek [2]

V tabulce *Tab. č. 4-4* je vysvětlována záměna hodnoty, která je uložnena pod proměnnou "a", například součtem dvou proměnných "a" s "b" (+=).

Když by bylo bráno v potaz, že operátor jako "++" může mít dvojsmysl a neví se, zda požadovaná hodnota je vypsána před nebo po inkrementaci (zvýšení), poslouží k objasnění následující vysvětlení:

```
i++
```

Význam zápisu znamená "Dej mi hodnotu před inkrementací"

```
++i
```

Význam zápisu znamená "Dej mi hodnotu po inkrementací"

Jako příklad můžeme napsat:

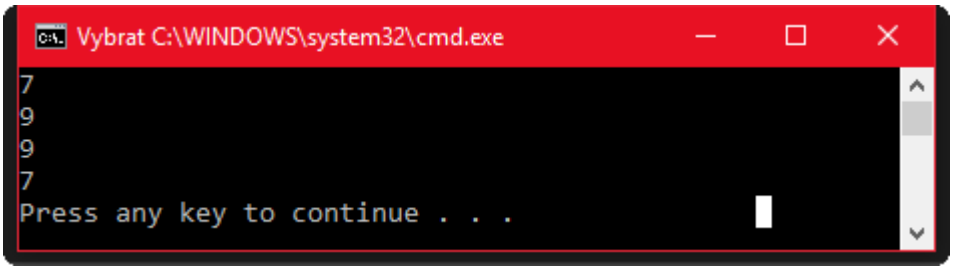
```
int i = 2;
int j;
j = ++i ;
```

Zde by se stalo, že “j“ by dostalo hodnotu 3.

Názorná ukázka příkladu s využitím zkratek:[1]

```
namespace ConsoleApp6
{
    class Program
    {
        static void Main(string[] args)
        {
            int n = 7;
            Console.WriteLine(n++);
            Console.WriteLine(++n);
            Console.WriteLine(n--);
            Console.WriteLine(--n);
        }
    }
}
```

Výsledkem je:



Obr. 4-2 - výsledné konzolové okno pro inkrementaci [1]

4.7 Závorky

Závorky se píší ve dvojicích tzn. pro každou otevřenou závorku "(" musí být i závorka uzavírající ")". Umožňují programátorům shlukovat kusy programu na jedno místo. Tento shluk je často nazýván blokem, který může obsahovat deklaraci proměnných využitých uvnitř bloku.

Operátor -závorky []

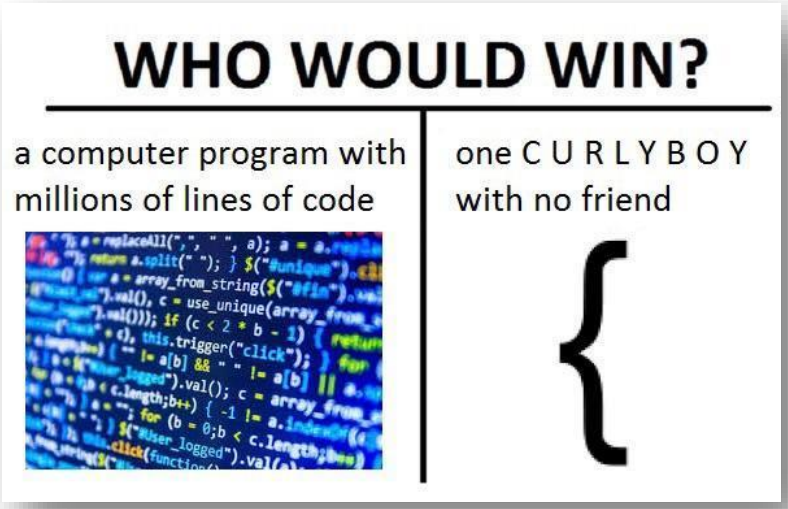
Hranaté závorky slouží ke vstupu elementu do pole pod určitým indexem ->indexer. Indexery jsou využívány také ke vstupu charakterů (char) do textových polí (string).

Operátor - závorky ()

Kulaté závorky se využívají k přepsání priorit provedení výrazů a operací. Již jsme se s nimi seznámili.

Operátor - závorky {}

Tyto závorky slouží k tvorbě bloku. Jakmile kompilátor uvidí, že blok je ohraničen a uzavřen závorkami stejného typu, začne se rozhlížet, kde je další počátek nového bloku.



Obr. 4-3 - nutnost uzavírání bloků [9]

Z *Obr. 4-3* je důležité si uvědomit nutnost tvoření páru závorek.

Je taky možné vytvoření vnořených bloků uvnitř jiného bloku (nestedblocks). Každý z vnořených bloků může mít své vlastní proměnné. Příklad vnořeného bloku:

```
{
        int i ;
                {
                        int j ;
                }
}
```

4.8 Přetypování (Casting)

Pomocí přetypování můžeme donutit C# považovat hodnotu jako součást jistého typu. Volání má formu dodatečné instrukce pro kompilátor, aby ji bral jako hodnotu v určitém směru, který je vyžadován. Hodnota se zavolá pomocí vložení potřebného typu do závorek před název např. proměnné, viz příklad:

```
double d = 1.5;

float f = (float) d ;
```

V kódu příkladu uvedeného výše, byla předána zpráva kompilátoru, aby ignoroval, že se jedná o typ double, i když může dojít ke ztrátě dat. Osoba, která kód napsala, bere na sebe veškerou zodpovědnost, že program bude pracovat správně. Přetypování je možné implicitně nebo explicitně. Implicitní způsob je typově bezpečný a je vztažen k integrálním typům. K explicitnímu přetypování je nutné uvést operátor přetypování, jelikož není typově bezpečné. Tento způsob je využíván, pokud se provádí číselný převod na typ, který má menší přesnost než typ předchozí.

Jak bylo možné vidět výše, každý typ proměnné má svůj rozsah možných hodnot. Rozsah "plovoucích bodů" (floating points) je o dost větší než disponuje číselný typ integer.

5 Podmínka IF

Obecný příklad:

```
if (podmínka)
        Tvrzení nebo blok kde podmínka je pravda (True)
else
        Tvrzení nebo blok kde podmínka pravda není (False)
```

Podmínka určuje, co přesně se stane v programu. C# obsahuje způsoby jak explicitně vyjádřit, zda podmínka je pravdivá či nikoliv, uvnitř programu. Již jsme s tímto způsobem byly obeznámeni při vysvětlování typu Boolean.

Můžeme vytvořit podmínku, která vrací logický výsledek. Takovéto podmínky jsou nazvány "logickými". Příklad:

```
if (true)
        Console.WriteLine ( "Mám hlad" ) ;
```

Toto je přijatelný, avšak úplně zbytečný zápis podmínky, jelikož bude pokaždé pravda (True). Tudíž bude text "Mám hlad" vždy vytištěn na výstupu.

Názorný příklad na podmínku IF z prostředí matematiky [10]:

```
using System;
public class Exercise11
{
    public static void Main()
    {
        int a, b, c;

        double d, x1, x2;
        Console.Write("\n\n");
        Console.Write("Calculate root of Quadratic Equation :\n");
        Console.Write("----------------------------------------");
        Console.Write("\n\n");

        Console.Write("Input the value of a : ");
        a = Convert.ToInt32(Console.ReadLine());
        Console.Write("Input the value of b : ");
        b = Convert.ToInt32(Console.ReadLine());
        Console.Write("Input the value of c : ");
        c = Convert.ToInt32(Console.ReadLine());

        d = b * b - 4 * a * c;
        if (d == 0)
        {
            Console.Write("Both roots are equal.\n");
            x1 = -b / (2.0 * a);
            x2 = x1;
            Console.Write("First  Root Root1= {0}\n", x1);
            Console.Write("Second Root Root2= {0}\n", x2);
        }
        else if (d > 0)
        {
            Console.Write("Both roots are real and diff-2\n");

            x1 = (-b + Math.Sqrt(d)) / (2 * a);
            x2 = (-b - Math.Sqrt(d)) / (2 * a);

            Console.Write("First  Root Root1= {0}\n", x1);
            Console.Write("Second Root root2= {0}\n", x2);
        }
        else
            Console.Write("Root are imeainary;\nNo Solution. \n\n");
    }
}
```

6 Cykly

Podmínkové tvrzení umožňuje provést část kódu, pokud bude daná podmínka splněna, tedy bude True. Nicméně, velmi často je nutné určitý proces několikrát zopakovat za předpokladu, že podmínka bude pravdivá/nepravdivá. Pro tento případ nám pouslouží cykly.

C# má několik cest, jak výsledku docílit. Závisí to převážně na věci, které se chce dosáhnout. Několik cest existuje z důvodu, aby v různých případech byla práce s kódem zjednodušena použitím jedné nebo více cest naráz.

6.1 DO – WHILE cyklus

Do – while konstrukce vypadá přibližně takto:

```
do
        Tvrzení nebo blok
while (podmínka) ;
```

Zde je možné opakovat kus kódu do té doby, než podmínka bude hodnoty False. Dobré by bylo si také zapamatovat to, že právě probíraný cyklus proběhne vždy alespoň jedenkrát, i když test je již předurčen k neúspěchu. Názorný příklad:

```
using System;
class Forever
{
        public static void Main ()
                {
                        do
                                Console.WriteLine ( "Mám hlad" ) ;
                        while( true);
                }
}
```

Tento kód je naprosto v pořádku. Jak dlouho by tento kód běžel, je dobrá otázka. Kód obsahuje části lidské psychologie, energie a možná i trochu té kosmologie. Program s tímto kódem poběží tak dlouho dokud:

- Vás to přestane bavit
- Dojde Vám elektřina
- Vesmír vybuchne a nezůstane nic

6.1.1 WHILE cyklus

Někdy chcete pouze rozhodnout, zda smyčku zopakovat předtím, než bude spuštěna. K tomu nám poslouží cyklus WHILE. Pro lepší pochopení lze uvést příklad, kde je třeba se zeptat na hodnotu ještě předtím, než bude rozhodnuto, zda je dotázaná hodnota přijatelná cyklem (true) či nikoliv.

Obecný příklad:

```
while (podmínka)
        Tvrzení nebo blok
```

6.2 FOR cyklus

Často bude potřeba zopakovat určitou věc daným počtem smyček. Cyklus FOR je pro tento případ jako dělaný.

```
1  using System;
2  public class Exercise1
3  {
4      public static void Main()
5      {
6      int i;
7      Console.Write("\n\n");
8      Console.Write("Display the first 10 natural numbers:\n");
9      Console.Write("---------------------------------------");
10     Console.Write("\n\n");
11
12      Console.WriteLine("The first 10 natural number are:");
13
14     for (i=1;i<=10;i++)
15     {
16      Console.Write("{0} ",i);
17     }
18     Console.Write("\n\n");
19   }
20 }
```

Obr. 6-1 – příklad na cyklus FOR [10]

C# umožňuje konstrukci cyklu FOR a jeho podmínek na jeden řádek tímto způsobem:

```
for (vstupní výraz, podmínka pro provádění, inkrerment/dektrement)
{
        Tvrzení od kterých chceme, aby se několikrát zopakovala
}
```

Vstup vloží hodnotu do kontrolní proměnné, kterou cyklus začne. Mez je podmínka, která musí platit pro cyklus FOR, aby mohl dále pokračovat ve smyčce. Update se provádí v každé smyčce inkrementace/dekrementace kontrolní proměnné. Všimněte si, že podmínky jsou děleny pomocí znaku ";" a ne čárkou. Přesný postup cyklu FOR zní:

a. Vložení počáteční hodnoty do kontrolní proměnné
b. Stanovení meze a ověření zda cyklus se po dokončení smyčku uzavřel
c. Podmínky budou opakovány
d. Provedení update
e. Opakování se od bodu (b.)

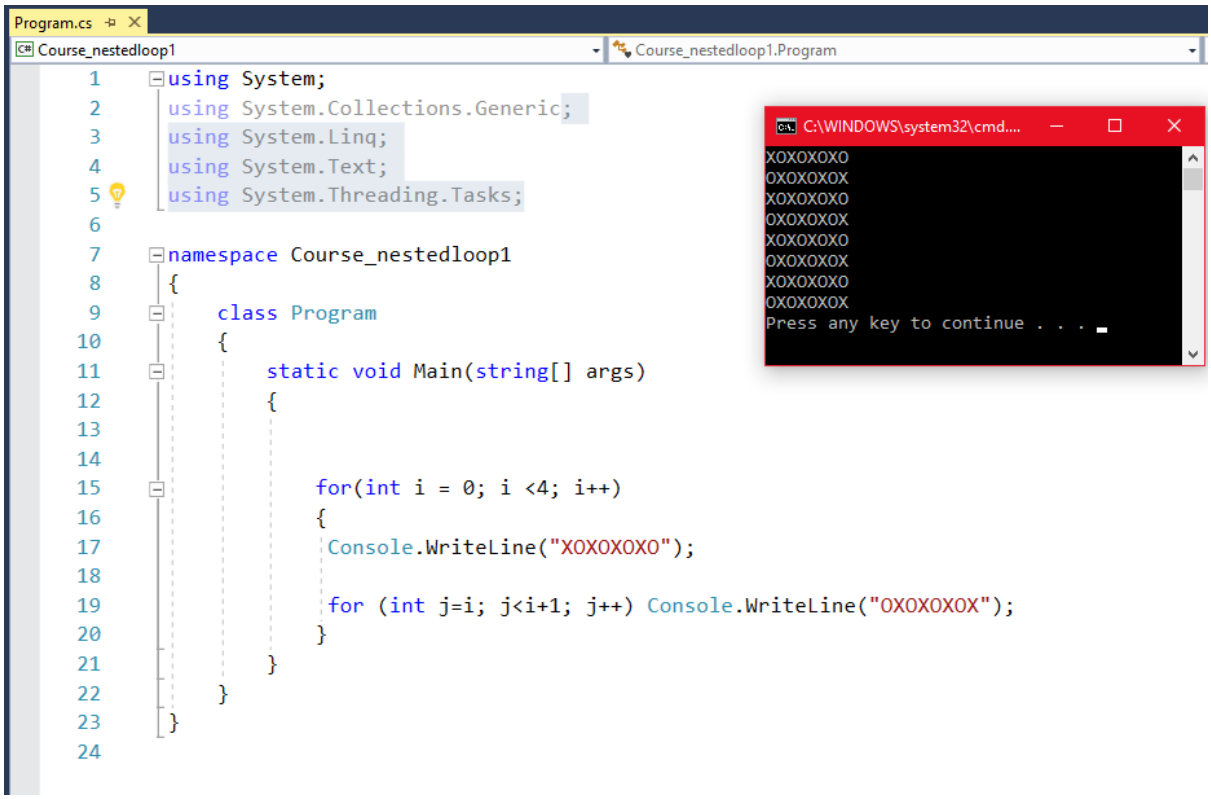
6.3 Vnořené cykly

Vnořený cyklus může být chápán, jako "cyklus v cyklu", kde máme jeden vnější cyklus a v něm vnořený druhý, vnitřní cyklus. Vnitřní cykly proběhnou vícekrát než cykly vnější. Zde je názorný příklad vnořeného cyklu:

```
for (vstup, mez, update)
{
        for (vstup, mez, update)
        {
                Proveditelný kód
        }
        ...
}
```

Po dosazení vstupního parametru pro vnější cyklus bude proveden jeho obsah mezi {…}. Zde se nachází vnořený vnitřní cyklus. Jeho vstupní parametr bude inicializován, jeho podmínky budou zkontrolovány a kód uvnitř závorek {…} vnitřního cyklu bude proveden. Vstupní proměnná vnitřního cyklu bude "updatována". To vše, kromě vstupu počátečního parametru, se bude opakovat, dokud proměnná nevrátí hodnotu True. Poté bude provedena druhá iterace vnějšího cyklu. Jeho proměnná bude "updatetována" a celý vnitřní cyklus bude proveden znovu a to minimálně tolikrát, kolikrát bude zopakován cyklus vnější.

Příklad vnořeného cyklu:



Obr. 6-2 – příklad vnořeného cyklu FOR [5]

6.4 Únik z cyklů

Někdy by bylo třeba z cyklu předčasně "utéct" tzn. program díky zápisu rozhodne, že není už třeba dále ve smyčce pokračovat a následně bude provedeno tvrzení uvnitř cyklu k opuštění smyčky. Toho se může docílit takzvaným "break" tvrzením. Je to příkaz k okamžitému opuštění smyčky.

Příklad na využití "break":

```
while (beziOK)
{
        kus kódu
        ....
        if (preruseno)
        {
                break ;
        }
        .... další kus kódu
}
....
```

```
Kousek kódu, který dostaneme, když cyklus přerušíme

....
```

6.5 Vracení se na začátek cyklu

Někdy bude potřeba se vrátit na začátek cyklu a udělat ho celý znovu. Tento případ může nastat, když se cyklus dostane dál, než je potřeba. Pro řešení zmíněného problému C# poskytuje klíčové slovo continue. To má význam přibližně takový:

Prosím nepokračuj již dále a vrať se zpět. Jdi zpět na začátek cyklu a udělej všechny "updaty" a další věci úplně znova.

V následujícím programu bude bool proměnná "Vse_bylo_splneno" nastavena jako pravda (True), jakmile dosáhneme požadované hloubky cyklu.

```
for ( predmet = 1 ; predmet<Celkem_predmetu ; predmet = predmet +1 )
{
 .... práce s předmětem ....

        if (Vse_bylo_splneno ) continue ;

... další práce s předmětem ....
}
```

"Continue" způsobí, že program "restartuje" cyklus s další hodnotou předmětu, pokud hodnota bude OK.

7 Výčtový typ (Enumeration)

Výčtový typ je struktura, která připomíná třídu, ale liší se od ní tím, že v "těle" výčtového typu lze využít pouze konstanty (constants). Výčtový typ může přijímat hodnoty pouze od konstant zařazených v listu daného typu. Výčtová proměnná může mít hodnotu jedné položky z daného typu listu konstant, ale nemůže mít hodnotu null.

Výčtový typ může být deklarován pomocí slova "enum" namísto třídy:

```
[<modifikátory>] enum<enum_jméno>
{
        konstanta1 [, konstanta2 [, [, … [, konstantaN]]
}
```

Pod pojmem <modifikátory> se rozumí vstup modifikátorů dle typu public, internal a private. Identifikátor <enum_jméno> se řídí dle pravidel pojmenování tříd v jazyce C#. Konstanty oddělené čárkami jsou deklarovány ve výčtovém bloku.

Jak ukazuje následující příklad, je možnost si definovat výčtový typ pro dny týdnu. Jak můžete hádat, konstanty budou v tomto výčtovém typu pojmenovány dle dnů v týdnu:

```
enum Dny
{
        Po, Út, St, Čt, Pá, So, Ne
}
```

Zde můžete vidět, že konstanty, obsažené ve výčtovém typu, jsou stejného typu, v jakém jsou deklarovány tzn. konstanta Po spadá pod typ Dny, tak jako všechny ostatní konstanty.

Jinými slovy, jestliže spustíme následující kód:

```
Console.WriteLine(Dny.Po is Dny);
```

Výsledek bude mít vytištěnou podobu:

```
True
```

Pojďme si ještě jednou zopakovat: výčtový typ je soubor konstant jednoho typu – výčtový typ listu.

Každá z konstant ve výčtovém typu je textovým zastoupením pro určité číslo, v jakém pořadí jsou konstanty ve výčtovém typu uloženy (indexy). Hlavním úklom výčtových typů je nahrazení číselných hodnot, které by bylo nutno použít v případě neexistence výčtových typů. Konstanty výčtového typu mohou být využity i ve struktuře switch, o které se píše v další kapitole.(2)

8 Switch

Cykly, jako jsou WHILE, FOR a DO, kontrolují struktury smyček nebo iterací. IF cyklus kontroluje strukturu pro větvení. Ve většině případů IF je používané pro selekci, ale je zde alternativa tzv. mnohonásobného větvení a tou je SWITCH. Nejlépe se SWITCH vysvětlí na příkladu dnech v týdnu.

Úkolem je napsat program, kde uživatel musí zadat určitý vstupní paramter – integer. Jestliže číslo bude 1, 2, 3, 4 nebo 5, program vytiskne jméno dne v týdnu přiřazeného k tomuto číslu a pokud bude číslo 6 nebo 7, program vytiskne zprávu, že se jedná o víkend. Pokud číslo nebude v rozmezí intervalu 1-7, vytiskne se pouze zpráva s chybou (errormessage), viz příklad na další stránce[1]:

```
static voidMain(string[] args)
{
Console.Write("Zadejte den v týdnu: ");
string text = Console.ReadLine();

switch (Convert.ToInt32(text))
        {
                case 1:
                        Console.WriteLine("Pondělí");
                        break;
                case 2:
                        Console.WriteLine("Úterý");
                        break;
                case 3:
                        Console.WriteLine("Středa");
                        break;
                case 4:
                        Console.WriteLine("Čtvrtek");
                        break;
                case 5:
                        Console.WriteLine("Pátek");
                        break;
                case 6:
                case 7:
                        Console.WriteLine("Víkend");
                        break;
                default:
                        Console.WriteLine("Neexistující den");
                        break;
        }
}
```

Konstrukce má počet možných "scénářů" - vstupů, kde "case" tyto scénáře rozlišuje v našem případě dle jejich očíslování.

Konstanty musí mít stejný typ jako výraz:

case 1: // případ 1

Jakmile je určité tvrzení provedeno, výraz je vyhodnocen a řízení je přesměrováno na "case" vstup, který je identický s hodnotou výrazu. Tvrzení a "case" vstupy jsou nadále spuštěny, dokud se nenarazí na již zmíněný únik z cyklů "break". Následně je vráceno tvrzení nebo SWITCH skončí. Jestliže se nenajde žádný "case" výstup, který odpovídá hodnotě hledaného výrazu nebo SWITCH podmínce, tak můžou nastat dvě situace. První je, že SWITCH má tzv. výchozí případ (default case) a řízení programu je přesměrováno na toto tvrzení. Jinak celý SWITCH může být jednoduše přeskočen. Výchozí případ (default case) není povinný.

V příkladu uvedeném výše, jakmile bude vstup číslo 6 nebo 7, program v obou případech ukáže výstup formou vytištěného textu "Víkend". Z kódu může být také vidět, že případ 6 není definován, avšak použitým zápisem případ 6 přebírá vše od případu 7. Jakmile uživatel zadá číslo 6 nebo 7, výsledek bude stejný.

SWITCH je tvrzení umožňující větvení. Ve vhodných případech může být nejlepším řešením, ale není tak často využíván jako ostatní tvrzení ke kontrole funkčnosti programu.

9 Metody

(2)K vyřešení určitého úkolu můžeme použít metody, jejichž funkcí je takzvané "rozděl a panuj" (při rozdělení do menších částí je docíleno lepší orientace v kódu a snáze se s ním nadále pracuje). Problém, který je řešen, je lepší rozdělit do několika částí. Části jsou brány jedna po druhé, a proto je kód obsažený v nich lépe přehledný, než kdyby se metody nepoužili. V okamžiku vyřešení všech malých částí, se teprve začne řešit kód jako celek. Ve zkratce se rozdělí úkol do sub-úkolů, najdou se jednotlivá řešení pro sub-úkoly a následně se spojí zpět do jednoho celkového úkolu. Tyto části úkolu se v jazyce C# nazývají metody.

Definice: metoda je základní částí programu, může řešit určitý problém, popřípadě vzít určitý parametr a vrátit výsledek.

Příklad metody s danými parametry:

```
static double ObsahObdelnika(double sirka, double vyska)
        {
                double obsah = sirka * vyska;
                return obsah;
        }
```

Pokaždé když je program tvořen, je dobré využít metody. Program následně bude lépe strukturován a snazší k pochopení ostatními lidmi, kteří se na tvorbě kódu nepodílí.

Deklaraci metod nazýváme registrací metody v programu, kde je lehce rozeznatelná od zbytku kódu. V C# jazyce můžou být metody deklarovány pouze mezi závorkami {…} třídy.

V objektově orientovaném programování jsou metody definovány párem elementů jejich deklarace, což je jméno a list parametrů. Tyto dva elementy definují takzvanou specifikaci metody neboli její podpis. Typ, který je vrácen, není součástí této specifikace, a to z důvodu odlišování dvou metod

se stejným typem na výstupu. Pro program by jinak situace byla nepřehledná a nevěděl by, jakou z metod zavolat.

Implementace metod je proces, kdy je nutno napsat její kód (funkčnost), který plní specifickou funkci. Tento kód v podstatě představuje metodu a její logiku. Metoda by měla být pojmenována dle funkce, kterou má plnit. Název metody musí začínat velkým písmenem.

Zavolání metody je proces, který "vzbudí" deklarovanou metodu z té části kódu, kde se nachází problém k vyřešení.

K předání informace je nutné pro metodu využití listů parametrů:

```
static výstupní_typ jméno_metody (list_parametrů)
{
        // Tělo metody
}
```

Pro lepší přehlednost bude ukázán specifický příklad:

```
static void NazevFirmy(string nazev)
{
        Console.WriteLine(nazev);
}
```

Při volání metody se hodnoty, zadané jako parametry v samotné metodě, nazývají argumenty.

Jinými slovy jméno a město jsou elementy nazývány parametry, jak může být viděno v následujícím příkladu:

```
static void TiskniFirmu(string jmeno, string mesto)
```

Zde již ale metodu voláme se specifickými argumenty:

```
TiskniFirmu("Jtekt", "Plzeň");
```

V případě, že uvnitř třídy je deklarována metoda a její jméno se shoduje se jménem jiné metody, ale liší se listem parametrů, jedná se o takzvané přetěžování metod (method overloading).

Příiklad na přetížení metod: [4]

```
static void Tisk (string text)
{
        // Tisk textového řetězce (string)
}
static void Tisk(int cislo)
{
        // Tisk čísla (int)
}
static void Tisk(float cislo)
{
        // Tisk čísla s plovoucí desetinou čárkou (float)
}
```

Jestliže bude změněn typ parametru, avšak jméno parametru zůstane stejné, jedná se zcela o dvě různé metody nebo přesněji řečeno a jinou variaci této metody.

9.1 Vracení hodnot

Vysvětlení na příkladu může znít:

Vezme-li se metoda na výpočet plochy čtverce, kde se místo vytištění výsledku, “vrátí“ výsledek v proměnné použitím klíčového slova "return". V tomto případě se počítá, že se s výsledkem (proměnnou) bude nadále manipulovat například v dalším výpočtu.

10 Pole

Pole může být definováno jako určitý počet objektů určitého typu, které jsou uloženy na jednom místě. Jednotlivé objekty je odkazováno jménem pole a indexem v onom poli. Indexem se myslí pozice, na které se objekt v poli nachází. V poli je vždy první index 0. V C# je možno vytvořit desetimístné pole pro deset objektů následovně:

```
int[] cisla = new int[10];
```

V závorkách […] může být číslo, které nám sděluje počet míst v poli. Všimněte si v příkladu níže, jak je vytvořeno pole pomocí klíčového slova "new", jelikož pole je referenčním typem.

Dvě pole typu int jsou vytvořena v následujícím příkladu: [1]

```
static voidMain(string[] args)
{
        int[] v1 = new int[10];
        int[] v2 = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };
        for (int i = 0; i <= v1.Length; ++i)
                {
                v1[i] = i + 1;
                Print(v1);
                }
}
Print(v2);

static void Print(int[] t)
{
        for (int i = 0; i <t.Length; ++i)
        {
        Console.Write("{0}", t[i]);
        }
        Console.WriteLine();
}
```

Obr. 10-1 – výstupní konzole příkladu na pole v1, v2 [1]

Je důležité si povšimnout, že pole v2 je definováno seznamem čísel. To znamená, že kompilátor může vytvořit pole a přímo ho inicializovat s elementy, zadanými v daném seznamu. U pole v1 lze vidět, že je použit cyklus FOR pro přiřazení hodnot ke každému indexu v poli. V parametrech tohoto cyklu je využita délka onoho pole jako mez pro cyklus.

10.1 Vícedimenzionální pole

Občas bude potřeba ukládat více než jednu řadu objeků např. čísel (int). Za daných okolností se využije vícedimenzionální pole, které si můžeme představit i jako tabulku se sloupci a řádky. Kdybychom chtěli vytvořit prostředí pro hraní piškvorek, pole by vypadalo následovně:

```
int[,] prostredi =new int[3,3];

prostredi [1,1] = 1;
```

Pole vypadá téměř stejně jako to jednodimenzionální. Jediný rozdíl je uvnitř závorek [,], kde se nyní vyskytuje čárka, která odděluje "sloupce" od "řádků ("2 dimenze"). Když udáváme velikost pole, musí být dány obě velikosti dimenzí (sloupců a řádků).

	0	1	2
0	0	0	0
1	0	1	0
2	0	0	0

Tab. č. 10-1 – vizualizace vícedimenzionálního pole [2]

10.2 Foreach

FOREACH cyklus (upravený cyklus FOR) je konstrukce sloužící k “prozkoumání“ všech elementů z pole, listu a další kolekcí. Cyklus projde všechny elementy kolekce, i když se jedná o kolekci bez indexů.

Tento cyklus je jednoduší než běžný FOR cyklus, a proto je často využíván developery, jelikož šetří čas při psaní kódu. Zde je jednoduchý příklad: [4]

```
int[] cisla= { 2, 3, 5, 7, 11, 13, 17, 19 };
foreach(int i in cisla)
{
        Console.Write(" " + i);
}
Console.WriteLine();
string[] mesta = { "Praha", "Plzeň", "Karlov Vary", "Brno" };
foreach (string mesto in mesta)
{
        Console.Write(" " + mesto);
}
```

Ve výše uvedeném příkladu bylo vytvořeno pole čísel, které následně prošlo cyklem FOREACH. Díky této funkci byly vytištěny všechny položky tohoto pole ve výstupu (konzole). Následně bylo vytvořeno pole s textem. Postup je totožný jako u prvního pole.

11 Výjimky

Výjimka slouží například k zobrazení upozornění, že určitá událost narušila normální běh programu. Zobrazení upozornění je jednou z výstupních možností v případě odchycení výjimky. Výjimka umožňuje detekci neočekávaných událostí následnou reakci vůči těmto událostem. Například se program může pokusit otevřít složku, avšak cesta k umístění složky je zadána chybně, zapojí se do "práce" výjimka, která nám řekne, z jakého důvodu nelze složku otevřít. V tomto případě by bylo ve zprávě psáno: "Složka nenalezena". Samozřejmě výjimek na otevírání složek může být více než jedna. Například složka by mohla být prázdná a program by si dále nevěděl rady co si počít, kdyby tato výjimka ošetřena nebyla.

Na příkladu níže [2] bude popsáno, jak předcházet chybám v programu, například z důvodu nesprávného použití uživatelem. Nesrpávným použitím může být rozuměno například vložení chybného vstupu. Od toho tu máme funkci Parse, která se využívá k přetypování vstupu. K aplikaci výjimek se používá postup "try-catch" a nepovinným dodatkem za částí "catch" je "fianlly". Po klíčovém slovu "try" navazuje blok s kódem, který se "zkouší" k nalezení podmínek ke spuštění výjimky. Následuje "catch", který nalezené výjimky "chytí" a vytiskne uživateli zprávu, o jaké výjimky se jedná.

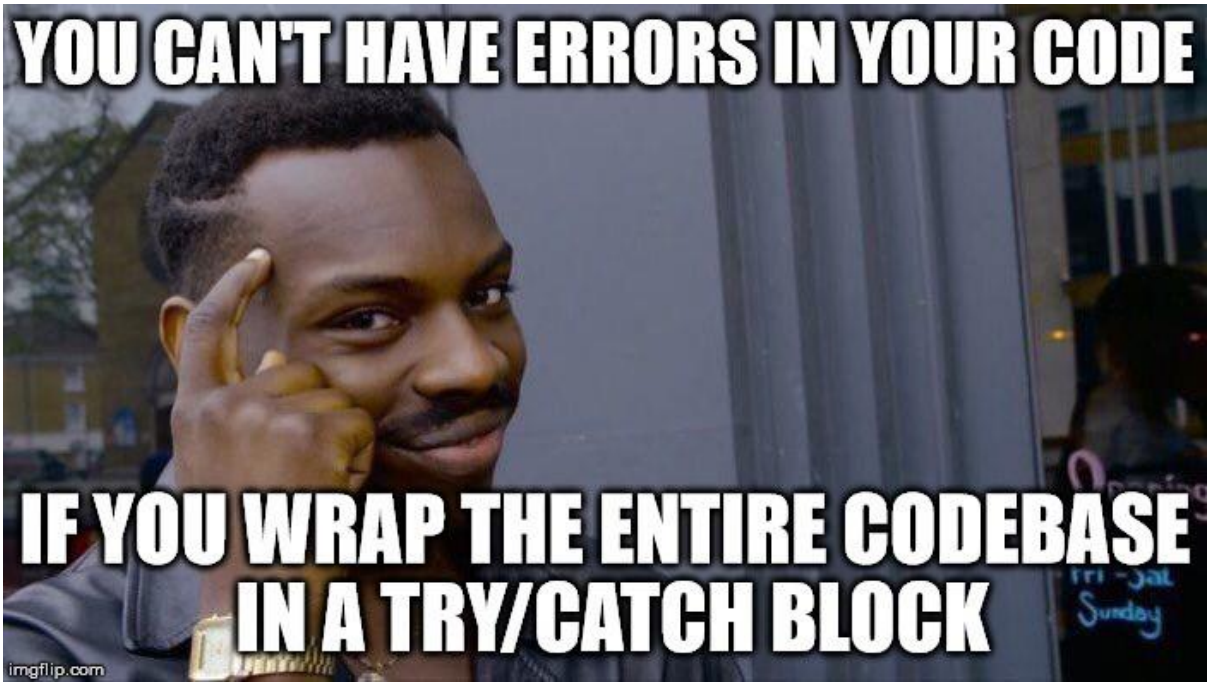
```
int vek;
try
{
        vek= int.Parse(vekString);
        Console.WriteLine("Děkuji");
}
catch (Exception e)
{
        // Zpráva o chybě
        Console.WriteLine(e.Message);
}
finally
{
        // nepovinné
}
```

"catch" nyní vypadá jako metoda s daným parametrem "e". A vlastně tak i aplikace výjimek funguje. Uvnitř bloku "catch" hodnota parametru "e" je nastavena výjimkou, která byla vytvořena funkcí "Parse". Typ výjimky má vlastnost nazvanou "Message", která obsahuje string popisující chyby (errors).

Pokud uživatel vloží špatný vstup do programu uvedeného výše, objeví se zpráva:

Vstupní text (string) není korektním vstupním formátem.

C# umožňuje přidání tzv. "finally" do "try-catch" konstrukce. Tvrzení uvnitř bloku "finally" je provedeno vždy, i když žádná podmínka není v "try" bloku nalezena.



Obr. 11-1 – ukázka špatného způsobu přemýšlení ohledně aplikace výjimek [8]

Ideálním avšak nemožným by bylo dle *Obr. 11-1* zahrnout celý kód složitého programu do bloku try a nadále se nestarat o aplikaci výjimek.

12 Rekurze

Objekt se může volat rekurzivně, pokud obsahuje sám sebe, nebo je sám sebou definován. Rekurze může být popsána jako metoda, která volá sama sebe, aby vyřešila určitý úkol.

Využití rekurze bude představeno příkladem na výpočet faktoriálu. Faktoriál n! Je produkt všech čísel (int) mezi 1 a "n" včetně, což vyplývá z definice tohoto faktoriálu 0! = 1.

n! = 1.2.3...n

Přítomnost vracející se závislosti není vždy tak očividná. V tomto případě lze rekurzi určit pomocí vyčíslení hodnoty faktoriálu pro prvních pár čísel (int).

0! = 1

1! = 1 = 1.1 = 1.0!

2! = 2.1 = 2.1!

3! = 3.2.1 = 3.2!

Odtud je čitelná vracející se závislost, která má obecný tvar:

n! = n*(n-1)!

Příklad kódu:

```
static decimal Faktorial(int n)
{
        if (n == 0)
        {
                return 1;
        }
        // následuje rekurzivní volání
        else
        {
                return n * Faktorial(n - 1);
        }
}
```

Obr. 12-1 – důležitost uvedení únikové podmínky u rekurzivních funkcí [7]

Z *Obr. 12-1* je možné vidět, jak důležité je uvedení podmínky výstupu z rekurzivní funkce. Pokud se tak neučiní, funkce se bude cyklit do nekonečna.

13 Delegát

Delegátem lze manipulovat s odkazy na metody. Můžeme jimi ovlivnit, jaká specifická metoda bude zavolána v dané situaci.

Delegát je speciální typ třídy, který obsahuje pouze informace ohledně určité metody. Mohou nastat situace, kdy je zapotřebí využití jisté metody v metodě jiné. V případě zmíněného problému se využije delegát.

Často se delegát využívá, jako základ k řešení modelu nastalých akcí (stisknutí tlačítka, atd.). Disponuje však také možností odkázání se na metodu formou parametru delegátu v metodě jiné. Jakákoliv metoda, která vyhovuje formě delegátu a vrací výsledek a parametry, může být přiřazena onomu delegátu.

Pro lepší zapamatování byl využit “vzenešený“ font.

Delegáte je typově bezpečný odkaz na metodu ve třídě. [2]

"Typově bezpečným" má význam takový, že pokud metoda přijímá dvě čísla (int) jako parametry a vrací string, delegát pro danou metodu bude vypadat přesně stejně jako metoda a nemůže být využit v jiném směru než pro danou metodu.

- Delegát umožní metodě být předána jako parametr
- Delegáté mohou být propojeni tzn. vícenásobné metody můžou být volány pomocí jediné události (eventu).

Jako příklad může být bráno vyčíslení poplatků našeho bankovního účtu. Banky mají několik takovýchto výpočtů (metod) pro různé typy zákazníků.

Delegát zde zastupuje za metody. Jestliže je zavolán delegát, je s ním zároveň zavolána metoda, na kterou se odkazuje delegát. Zde delegát funguje na principu rozhodovatele, jakou metodu použít. Typ delegate je vytvořen přibližně takto:

```
public delegate decimal VycisleniPoplatku(decimal zustatek);
```

Zde zatím nebyl vytvořen žádný delegát. Bylo jen řečeno kompilátoru, jak delegát "VycisleniPoplatku" vypadá.

Příklad na výpočet úroků: [2]

```
using System;
public delegate decimal VycisleniPoplatku(decimal zustatek);
public class DelegateDemo
{                                                   // začátek třídy
        public static decimal VysokyPoplatek(decimal zustatek)
        {
                Console.WriteLine("Volání metody ohledně vysokého poplatku");
                if (zustatek< 0)
                {
                        return 100;
                }
                else
                {
                        return 1;
                }
        }

        public static decimal MalyPoplatek(decimal zustatek)
        {
                Console.WriteLine("Volání metody ohledně malého poplatku");
                if (zustatek< 0)
                {
                        return 10;
                }
                else
                {
                        return 1;
                }
        }
```

```
        public static void Main()
        {
                VycisleniPoplatkucalc;

                calc = new VycisleniPoplatku(VysokyPoplatek);
                calc(-1); // zde se volá metoda ohledně vysokého poplatku
                calc = new VycisleniPoplatku(MalyPoplatek);
                calc(-1); // zde se volá metoda ohledně malého poplatku

}                                       // konec třídy
```

Důležitou věcí k zapamatování z výše uvedeného příkladu je, že v metodě Main() jsou dvě volání delegáta "calc". Každé z nich je výsledkem odlišného kódu.

14 List/Dictionary

V této části budou popisovány List a Dictionary, doplněné krátkým příkladem.

14.1 List

List si můžeme představit jako uspořádanou řádku elementů. Jako příklad můžou být brány platby v obchodě. V listu můžeme vidět každý z elementů (plateb). Zobrazují se jak platby již déle uskutečněné tak i platby právě příchozí. Je zde možnost i platbu vymazat nebo proházet jejich pořadí. List je lineární datová struktura, která obsahuje sekvenci elementů. Disponuje vlastností "Count", která nám řekne, jak rozsáhlý list je.

(1)Vytvoření listu je velmi jednoduché:

```
List ulozit= new List();
```

Základní metody definující list jsou:

- Add(object) - přidá element na konec listu
- Insert(int, object) - přidá element na předběžně vybranou pozici v listu
- Clear() - odstraní všechny elementy z listu
- Contains(object) - zkontroluje, zda v listu je obsažen dotazovaný element
- Remove(object) - odstraní element z listu
- RemoveAt(int) - odstraní element na dané pozici
- IndexOf(object) - vrátí pozici daného elementu
- This[int] - indexér, dovolí nám přístup k elementu na dané pozici

U listu se nemusí definovat velikost, jen se přidávají elementy a pokud je potřeba zjistit velikost listu zavolá se jeho vlastnost "Count“.

```
List ulozDeset= new List(10);

int delka = ulozDeset.Count();
```

List nazvaný ulozDeset je schopen pojmout 10 odkazů, ale není jeho limitací, pokud je nutno přidat/ubrat elementy. Tato možnost se však využívá pouze ojediněle. Nejčastěji se využívá list s neuvedeným počtem elementů uvnitř (new List( ))

Přidání předmětu do Listu je velmi snadné. Využije se metoda "Add" následujícím způsobem:

```
Ucet tomuvUcet = new Ucet();

ulozDeset.Add(tomuvUcet);
```

Metoda "Add" nepřidává položky do Listu, ale pouze přidává odkaz na onu položku.

Odstranění předmětů (odkazu) z Listu je možné využitím metody "Remove". Metoda odkazuje na předmět, který má být odstraněn následujícím způsobem:

```
ulozDeset.Remove(tomuvUcet);
```

Metoda odstraní první výskyt odkazu na daný předmět z Listu. Jestliže "uložit" obsahovalo více než jeden odkaz na "tomuvUcet", musí být každý z nich odstraněn zvlášť.

Poslední zajímavostí k vysvětlení je využití metody "Contains". Je to jednoduchá cesta k nalezení odkazu na předmět v Listu. Kód vypadá následovně:

```
If (ulozit.Contains(tomuvUcet))
{
        Console.WriteLine("Tom má v bance účet");
}
```

Metodě "Contains" je dán odkaz na předmět (objekt) a vrácená hodnota bude mít podobu bool True, pokud List tento odkaz obsahuje.

14.2 Obecné typy (Generics)

Obecné typy umožňují psát kód, který se vypořádá s objekty jako předměty určitého typu. Nezáleží, s jakým typem se pracuje. Typ bude definován, až bude potřeba. Lépe pochopitelná definice snad vyplyne z následujícího příkladu.

Příklad: pokud jste si ve skupině chtěli někdy rozdělit například cukroví, přišli jste na způsob, jak toho docílit, aby rozdělení bylo v rámci spravedlnosti. Možný postup byl takovýto: rozdělení cukroví podle počtu přátel ve skupině tak, aby každý člen měl stejný počet cukroví. Nyní vznikl systém na rozdělování cukroví. Tento systém se však může využít na rozdělení i úplně jiných věcí, než jsou sladké pochoutky. Tento systém je možno aplikovat například na rozdělení práce mezi pracovníky dle časové náročnosti. Tím je naznačováno, že nezáleží, jaký typ předmětu se bude rozdělovat.

14.3 Obecné typy a List

Jaké typy má List v sobě ukládat, není důležité, pokud existuje způsob sdělení Listu, co uložit může. V jazyce C# existuje prvek, který poskytuje obecným typům přidat nové zápisy.

Třída List je definována v "namespace" System.Collections.Generic a funguje následovně:

```
List<Ucet> ucty = new List<Ucet>();
List<int> skore = new List<int>();
```

Typ mezi závorkami <…> je způsob, jakým říkáme Listu, který typ předmětu může ukládat. Od tohoto okamžiku kompilátor ví, že do Listu "ucty" ukládá pouze odkaz "Ucet".

14.4 Slovník (Dictionary)

Každý element ve Slovníku má svůj klíč, ke kterému se váže hodnota. Klíč s hodnotou dohromady tvoří reprezentativní pár. Klíč je využíván k hledání potřebných hodnot uvnitř slovníků. Slovník ukládá pouze typově bezpečné předměty.

(1)Jako názorný příklad je uvedena situace, kde je myšlenka využít jméno majitele účtu jako cestu k nalezení účtu dotyčného. Z programátorského hlediska známe určitý text (string), který představuje klíč a odkaz Ucet, který zastupuje za hodnotu (value). Názorný příklad:

```
Dictionary<string,Ucet> uctySlovnik = new Dictionary<string,Ucet>();
```

Přidávání předmětů do Slovníku:

```
uctySlovnik.Add("Tom", tomuvUcet);
```

Slovník má tu výhodu, že je možné přidávat pouze hodnoty z odkazu Ucet, které mají podobu typu string.

Pokud nepotřebujeme výsledek, ale pouze vložit peníze na účet, lze kód napsat tímto způsobem:

```
d["Tom"].Vklad(50);
```

Slovník nedovolí dvěma různým elementům vlastnit stejný klíč.

Celý příklad na Dictionary: [4]

```
class DictionaryBanka
{                                       // začátek třídy
        Dictionary<string,IUcet> uctySlovnik= new Dictionary<string,IUcet>();
        public IUcet NajdiUcet(string jmeno)
        {
                If (uctySlovnik.ContainsKey(jmeno))
                        return uctySlovnik[jmeno];
                else
                        return null;
        }
        public bool Ulozit(IUcet ucet)
        {
                if (uctySlovnik.ContainsKey(ucet.GetName()))
                return false;
                accountDictionary.Add(ucet.GetName(), ucet);
                return true;
        }
}                                       // konec třídy   (1)
```

15 Lambda výrazy

Lambda výrazy (LV) jsou anonymní funkce, které obsahují výrazy operátorů. Všechny LV užívají lambda operátor "=>", který může být čten jako "jdi na". Levá strana operátoru definuje vstupní parametry, zato pravá strana má podobu výrazu nebo bloku kódu, který pracuje se vstupními parametry a vrací určitý výsledek. Hodně častým případem bývá nahrazení delegáta právě LV, které mohou být použity v kolekcích (collections), ke zpracování jejich elementů a vrácení určitého výsledku.

Jako příklad může být použita rozšiřující metoda FindAll(…), která má za úkol vyfiltrovat potřebné elementy. Musí být zahrnut "namespace" System.Linq, který umožňuje s LV pracovat. Úkolem příkladu je najít všechna sudá čísla z kolekce čísel (int).

Kód příkladu má podobu: [4]

```
List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6 };
List<int> sudaCisla= list.FindAll(x => (x % 2) == 0);
foreach (var cislo in sudaCisla)
{
        Console.Write("{0} ", cislo);
}
Console.WriteLine();
```

Výsledkem je: 2 4 6

Příklad prochází skrze kolekci čísel (int) a pro každý element (foreach) zkontroluje, zda číslo je dělitelné dvěma beze zbytku, viz 2. řádek z výše uvedeného příkladu.

15.1 Lambda výrazy jako Delegát

Lambda výrazy mohou být psány uvnitř delegátů. Delegát je typ proměnné, který odkazuje na metody. Typickým příkladem typu delegáta může být akce (Action), Func<out TVysledek> a další. Zmíněné dva delegáté jsou obecnými typy obsahující typ vracené hodnoty a typ parametrů funkcí (metod). Proměnné těchto typů jsou odkázány na funkce (metody). Níže je uveden příklad na přiřazování hodnot k těmto typům:

```
Func<bool> boolFunc = () => true;
Func<int, bool> intFunc = (x) => x < 10;
if (boolFunc() && intFunc(5))
{
        Console.WriteLine("5 < 10");
}
```

Výsledek je: 5 < 10

Ve výše uvedeném příkladu jsou definována dvojice delegátů. První je boolFunc, který je bezparametrovou funkcí a vrací výsledek typu boolean. V prvním řádku jsme použili lambda výraz, který nedělá nic jiného, než vrací True do funkce. Druhý delegát intFunc si bere argument typu integer (int) proměnné a vrací hodnotu boolean. True pokud x je menší než 10, jinak vrací False. Na konci kódu

voláme oba tyto delegáty, kde druhému delegátu přidělujeme argument, kterým je číslo 5. Konečným výsledkem porovnání je hodnota True.

16 LINQ

LINQ (Language-IntegratedQuery) obsahuje jazykově integrované dotazy a operace ohledně elementů týkajících se určitého zdroje dat. Velmi často tímto zdrojem bývají kolekce (collections) a pole (arrays). LINQ je velmi mocný nástroj. Obsahuje několik klíčových slov, jako jsou FROM, IN nebo SELECT. K jeho užití je potřeba deklarace "namespace" Systems.Linq na začátku programu.

K definici zdroje dat (kolekcí, polí, atd.) je možné použít klíčová slova FROM a IN. Dále je nutné použít proměnnou, která má na starosti iteraci kolekce. Příklad na dotaz (query):

```
from kultura

in InfoOKulture.GetKultura(KulturaTypes.VsechnyKultury)
```

Výše uvedený příklad může být čten takto: "pro každý element kolekce InfoOKulture.GetKultura(KulturaTypes.VsechnyKultury) přiřaď proměnnou "kultura" a využij ji v souvislosti s těmito předměty dále v dotazu."

Klíčová slova, kde mohou být využity podmínky, by měla být držena každým předmětem dané kolekce a to z důvodu dalšího pokračování dotazu. Doposud nezmíněná funkce WHERE slouží k filtrování elementů v kolekcích. Výraz, který následuje po WHERE, je vždy typu boolean.

K vybrání výstupních dat dotazů může být využito klíčové slovo SELECT. Výsledkem je objekt existující třídy nebo anonymního typu.

SELECT a vše co ho následuje, je vždy umístěno na koneci dotazu.

Ukázka příkladu LINQ:

```
List<int> cisla= new List<int>() {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
var sudaCisla = cisla.Where(cislo =>cislo % 2 == 0);
        foreach (var predmet in sudaCisla)
{
        Console.Write(item + " ");
}
```

Výsledkem bude:

```
2 4 6 8 10
```

Výše uvedený příklad zpustí dotaz vztažený na kolekci čísel (int) nazvaných "cisla" a vyfiltruje pouze čísla sudá v nové kolekci. Dotaz může být čten jako: "pro každé číslo v "cislo" z "cisla" zkontroluj, zda je "cislo" dělitelné dvěmi a pokud ano, přidej ho do nové kolekce".

Třídění pomocí LINQ dotazů je prováděno pomocí klíčového slova OrderBy. Podmínky pro třídění elementů jsou umístěny za klíčovým slovem. Třídění může mít směr vzestupný nebo sestupný (ascending/descending).

Příklad:

```
string[] slova = { "motor", "svíčka", "kliková hřídel" };
var slovaTridenaDleDelky = slova.OrderByDescending( slovo =>slovo.Length)
foreach (var slovo in slovaTridenaDleDelky )
{
        Console.WriteLine(slovo);
}
```

Výsledkem je:

I. Kliková hřídel
II. Svíčka
III. motor

Pokud nejsou dány instrukce, v jakém pořadí předměty třídit, budou vytištěny v takovém pořadí, v jakém byly vloženy do cyklu foreach.

17 Glosář

Volání [2]

Pokud je v úmyslu použít metodu, musí se nejdříve "zavolat". Při volání metody se začíná na počátku jejího bloku. Jakmile se dosáhne konce bloku metody nebo tvrzení return, je započata sekvence vracející se k tvrzení, která je následována voláním metody.

Třída [2]

Třída je kolekcí, která se skládá například z metod a různých vlastností. Třída může být využita i k popisu reálného předmětu v programu. Jakmile je zapotřebí shromáždit více různých proměnných na jednom místě, mělo by se uvažovat o vytvoření třídy, kam se proměnné uloží a kde zároveň budou i využívány.

Kolekce [2]

Kolekce můžou být definovány jako shluk věcí, od kterých je vyžadováno shlukování na jednom místě. Příkladem můžou být hráči na hřišti nebo všechny části motoru pod kapotou. Pod kolekce spadají Pole, Listy nebo Hashtable. Třída typu kolekce může obsahovat výčtový typ, což znamená, že lze od ní požadovat poskytnutí jednotlivých hodnot kolekce cyklem foreach.

Kolekce jsou k dispozici v "namespace" Systém.Collections.

Kompilátor [4]

Kompilátor vezme zdrojovou složku a upraví si ji dle svých potřeb. Následně vytvoří složku novou, kterou lze již zpustit. Kompilátory jsou různé pro každý typ programovacího jazyka.

Komponenta [2]

Je to třída, která ukazuje na své chování ve formě rozhraní. To znamená, že se musí na komponentu nahlížet spíše jako na třídu, která má vlastnost provést určitou věc než na třídu ze "vzhledové stránky".

Konstruktor [2]

Konstruktor může být popsán jako metoda, která volaná v podobě nové instance vytvořené třídy (ve které se konstruktor nachází). Je využíván ke kontrole nových instancí ve třídě a k definování hodnot uvnitř třídy.

Delegát [2]

Delegát je typově bezpečná metoda, která je tvořena pro určitou funkci (např. vstupní hodnoty jsou dvě čísla (int) a výstupní hodnota je float). Delegát je pak odkázán na metodu ve třídě, která danou funkci plní. Důležité k zapamatování je fakt, že delegát ve své moci drží dva předměty, a to referenci ke třídě obsahující metodu a referenci na samotnou metodu. Delegát je brán jako objekt, tudíž může být předáván dále jako každý jiný objekt. Důležitou funkcí delegátů jsou reakce na určité události, které mohou nastat například v uživatelském rozhraní (zmáčknutí tlačítka).

Událost [4]

Událost lze charakterizovat jako určité vnější ovlivnění, na které program musí určitým způsobem reagovat. Pod událostmi si lze představit pohyb myší, zmenšení/zvětšení okna prohlížeče, zmáčknutí tlačítka, a mnoho dalších příkladů. Jakmile nastane určitá událost, měla by se s ní objevit i metoda, která na tuto událost upozorní.

Knihovny [2]

Knihovna se skládá z několika tříd, které lze použít vícero programy. Rozdíl mezi třídou a knihovnou je způsob uložení v našem počítači. Je očividné, že vlastní různé "koncovky". Třída má ukončení ".cs" zatímco knihovna ".dll". Dalším rozdílem je, že knihovna nemůže obsahovat metodu Main().

Člen [2]

Je deklarován uvnitř tříd. Může mít podobu jak metody tak proměnné. V případě metody jsou zastoupeny jako jejich "chování". Proměnným se v této situaci říká "vlastnosti" metody.

Metadata [2]

Metadata jsou “data o datech“. Fakt, že počet pístů v motoru je uložen v programu jako int, je součástí metadat. Metadata jsou tvořena programátorem v rámci dohody se zákazníkem, pro kterého je program tvořen.

Metoda [2]

Metoda má svůj jedinečný název a může navracet určitou hodnotu. Je schopna též přijímat parametry, s kterými následně pracuje. Metody slouží k "rozkouskování" kódu z důvodu přehlednosti a lepšímu porozumění pro ostatní programátory, kteří na kódu nepracovali. Každá metoda v programu plní určitou část celkového úkolu onoho programu. Jestliže metoda je typu public, může být volána i jinými třídami, než v které je obsažena.

Namespace [2]

Namespace je oblast, v které má určité jméno určitý význam. V programu jako celku nelze použít stejné jméno více než jednou, ale díky "namespace" se tyto jména uvnitř těchto oblastí mohou opakovat. Namespace se deklaruje převážně na počátku kódu pomocí klíčového slova "using". Následuje jejich tělo/blok, kde mohou být vnořeny další "namespaces" podobně jako u vnořených cyklů.

Odkazy – reference [2]

Pod pojmem reference si lze představit štítek, který je připnut k instanci třídy. V programovacím jazyce C# se odkazy používají k nalezení způsobu využití metody obsažené v určité třídě. Odkaz může být pojen na další odkaz. V tomto případě byly vytvořeny dva odkazy "připnuté" k jednomu objektu v paměti.

Typově bezpečný [2]

Význam je doslovně: "nemíchej jablka a hrušky". V případě programování nelze přiřadit hodnotu float k proměnné int. Kompilátor by tento zápis nepřijal a oznámil by chybu. Důvod je takový, že vývojáři tohoto jazyka si všimli několika chyb v programování obecně a navrhli jazyk (C#), aby tyto chyby jazyk zaznamenal již předtím, než bude program spuštěn. Jeden z ukázkových příkladů může být ztráta dat z důvodu použití špatného typu, jako je přiřazení hodnoty double k proměnné byte.

18 Literatura

[1] KLAUSEN, Poul. *C# 1 Introduction to programming and the C# language*. B.m.: Poul Klausen & bookboon.com, 2012. ISBN 978-954-400-773-7.

[2] MILES, Rob. *C# Programming Yellow Book*. B.m.: Rob Miles, 2016. ISBN 150-930-115-7.

[3] PETRUSHA, Ron a OSTATNÍ. CLR. *Microsoft.com* [online]. 2017. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/clr

[4] SVETLIN, Nakov. *Fundamentals of Computer Programming with C#*. Sofia: Nakov Svetlin & Co., 2013. ISBN 978-954-400-773-7.

[5] G. O´BRIEN a MICROSOFT. Introduction to C#. *edX* [online]. 2017. Dostupné z: https://www.class-central.com/mooc/8823/edx-introduction-to-c. 2017

[6] Obrázek - Beethoven. *michigansilverback.com* [online]. Dostupné z: http://www.michigansilverback.com/2014/08/memes-of-week-july-27-august-2-2014.html

[7] DIPLYTECH. Obrázek - rekurze. *Facebook* [online]. 2018. Dostupné z: https://www.facebook.com/diplytech/photos/a.173689002772711.45685.173678916107053/1140116692796599/?type=3&theater

[8] BIRATKIRAT. Obrázek - Try/Catch. *medium.com* [online]. 2018. Dostupné z: https://medium.com/@biratkirat/step-21-distinguish-business-exceptions-from-technical-dan-bergh-johnsson-affdddd861d

[9] Obrázek - závorky {}. *reddit.com* [online]. 2018. Dostupné z: https://www.reddit.com/r/TechMaymays/

[10] W3RESOURCE.COM. *w3resource* [online]. 2018. Dostupné z: https://www.w3resource.com/csharp-exercises/for-loop/csharp-for-loop-exercise-1.php

[11] WPF-TUTORIAL.COM. WPF. *wpf-tutorial.ocm* [online]. 2018. Dostupné z: http://www.wpf-tutorial.com/getting-started/hello-wpf/

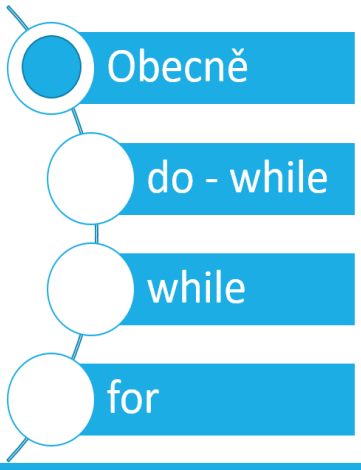
PŘÍLOHA č. 2

Krátká vzorová prezentace vztažená k části obsahu vytvořeného dokumentu

Předběžná forma prezentace dokumentu

AUTOR: MICHAEL POMPL

Strukturované příkazy – vývojový diagram



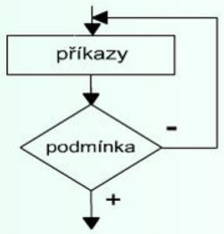
- **Cykly**
  - 3 základní typy:
    - **Do - while** - příkaz s podmínkou na konci
    - **while** - příkaz s podmínkou na začátku
    - **for** - příkaz s krokem (pevným počtem opakování)
  - Předčasné ukončení:
    - **continue** – přeskočení na další iteraci
    - **break** - přerušení cyklu

Strukturované příkazy – vývojový diagram

- Obecně
- do - while
- while
- for

Cyklus do - while

příkaz s podmínkou na konci



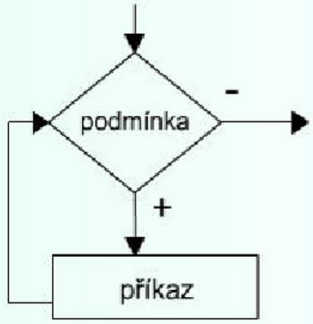

```
do
        Tvrzení  nebo
blok
while (podmínka) ;

příklad s danými
hodnotami
do
{
    int X = X + 1;
}
while (X>Y) ;
```

Strukturované příkazy – vývojový diagram

- Obecně
- do - while
- while
- for

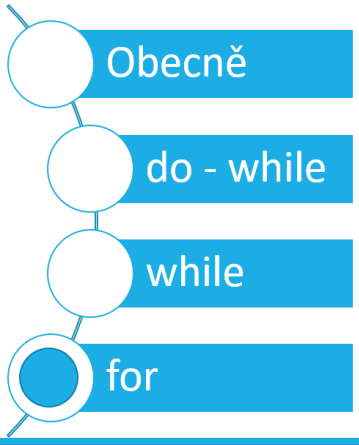
Cyklus while

příkaz s podmínkou na začátku



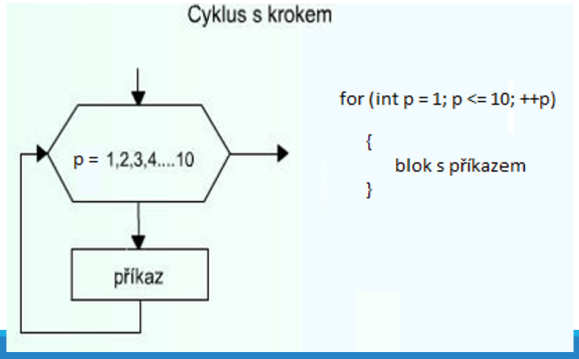

```
while(podmínka)
        Tvrzení nebo blok
```

Strukturované příkazy – vývojový diagram

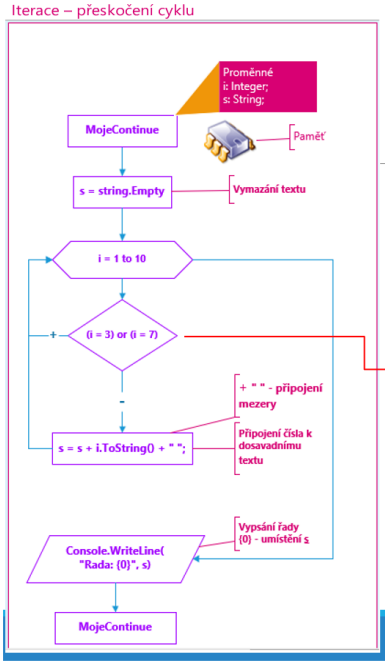


Cyklus for

Styl zápisu: for (vstup; mez; update)



Iterace – přeskočení cyklu



Dokážete napsat kód pro konzolovou aplikaci na základě vývojového diagramu?

```
namespace MojeContinue
{
    class Program
    {
        static void Main(string[] args)
        {
            string s = string.Empty;

            for (int i = 1; i <= 10; ++i)
            {
                if (i == 3 || i == 7) continue;

                s = s + i.ToString() +" ";
            }

            Console.WriteLine("Rada: {0}", s);
        }
    }
}
```

```
C:\WINDOWS\system32\c...
Rada: 1 2 4 5 6 8 9 10
Press any key to continue . . .
```

Výsledek po spuštění