Podpůrné materiály pro výuku programování na středních školách

Veronika Vyvlečková

Bakalářská práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Veronika Vyvlečková**
Osobní číslo:	**A17161**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Podpůrné materiály pro výuku programování na středních školách**
Téma práce anglicky:	**Support Materials for the Tuition of Programming in Secondary Schools**

Zásady pro vypracování

1. Nastudujte si jazyk C# a vývojové prostředí Microsoft Visual Studio.
2. V teoretické části práce popište vývojové prostřední Microsoft Visual Studio.
3. Součástí teoretické části práce bude i popis programovacího jazyka C# včetně jeho syntaxe a sémantiky.
4. V praktické části pak vytvořte výukové materiály ve formě tutoriálů (prezentací), které budou doplněny o řešené úlohy (zdrojové kódy); úlohy na sebe budou logicky navazovat s ohledem na Bloomovu taxonomii učení.
5. Vytvořené výukové materiály doplňte o možnost ověření získaných znalostí.

Forma zpracování bakalářské práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. NAGEL, Christian. Professional C# 7 and .NET Core 2.0. Indianapolis: Wrox, John Wiley & Sons, [2018], xviii, 1368 s. ISBN 9781119449270.
2. BORY, Pavel. C# bez předchozích znalostí. Brno: Computer Press, 2016, 255 s. ISBN 9788025146866.
3. PECINOVSKÝ, Rudolf. OOP – learn object oriented thinking and programming. Řepín: Tomáš Bruckner, 2013. Academic series. ISBN isbn978-80-904661-8-0.
4. Dokumentace k jazyku C#. Vývojářské nástroje, technická dokumentace a příklady kódování [online]. [cit. 2020-11-09]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/
5. Rámcové vzdělávací programy, MŠMT ČR. MŠMT ČR [online]. [cit. 2020-11-09]. Dostupné z: https://www.msmt.cz/vzdelavani/skolstvi-v-cr/skolskareforma/ramcove-vzdelavaci-programy

Vedoucí bakalářské práce: **Ing. Martin Strmiska**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **15. ledna 2021**
Termín odevzdání bakalářské práce: **17. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN

prof. Mgr. Roman Jašek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 17. 5. 2021

Veronika Vyvlečková, v.r
podpis studenta

ABSTRAKT

Práce je zaměřena na vytvoření výukových materiálů pro podporu výuky programování na středních školách. Cílem práce je vytvoření výukových materiálů v podobě praktických cvičení v programovacím jazyce C#. Výukové materiály jsou sestaveny podle požadavků pro výuku Informatického semináře na střední škole. V teoretické části bude popsán programovací jazyk C# včetně jeho syntaxe a sémantiky. Praktická část potom popisuje jednotlivé vyučovací hodiny a k nim přiřazené výukové materiály v podobě prezentací a zdrojových kódů.

Klíčová slova: C#, výukové materiály, základy programování, algoritmus, vývojové prostředí

ABSTRACT

This work is focused on creating educational materials, which will support programming lessons in secondary schools. This work aims to create these educational materials in form of practical exercises in programming language C#. Educational materials are compiled according to requirements for the Informatics seminar in secondary school. In theoretical part is described programming language C#, including its syntax and semantics. The practical part then describes individual lessons and assigned presentations and source codes.

Keywords: C#, educational materials, basics of programming, algorithm, integrated development environment

Chtěla bych poděkovat panu Ing. Martinu Strmiskovi za vedení práce, pomoc a poskytnutí rad během jejího vypracování.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato bakalářská práce by měla sloužit učitelům informatiky na středních školách jako podpůrný materiál pro přípravu vyučovacích hodin.

Bakalářská práce je rozdělena na teoretickou a praktickou část. V teoretické části je definován algoritmus, vlastnosti algoritmu a jeho zápis. Dále je stručně popsáno vývojové prostředí Microsoft Visual Studio a jeho prvky. Největší pozornost je pak věnována jazyku C# a jeho syntaxi a sémantice. Praktická část se zabývá jednotlivými vyučovacími hodinami, pro které jsou k dispozici praktická cvičení zahrnující prezentace a zdrojové kódy.

Cílem této práce je vytvoření výukových materiálů v podobě praktických cvičení do jednotlivých hodin programování na středních školách v jazyce C#. Výsledkem jsou sestavená cvičení na základě studijního programu žáků středních škol v odpovídající náročnosti.

I. TEORETICKÁ ČÁST

1 ALGORITMUS, VLASTNOSTI ALGORITMU A JEHO ZÁPIS

1.1 Definice algoritmu

Algoritmus je přesně definovaná konečná řada správně určených výroků, také nazývaných jako instrukce nebo příkazy, které poskytují řešení problému nebo specifické kategorie více problémů pro jakoukoliv přijatelnou sadu vstupních hodnot, pokud nějaké jsou. Jinými slovy algoritmus popisuje návod na řešení problému krok za krokem s tím, že na svém konci dosáhne řešení a neběží věčně. Určité kroky musí následovat v určitém pořadí, aby algoritmus mohl dosáhnout cíle. [1]

Počítač potřebuje být schopen interpretovat algoritmus, aby mohl uskutečnit proces, který daný algoritmus popisuje. Proto počítač jako takový musí být schopen porozumět v jaké formě je algoritmus vyjádřen a vykonat operace popisované algoritmem.

Pokud jde o formu, v jaké je algoritmus vyjádřen, sada gramatických pravidel, které určují, jak jednotlivé symboly v jazyce mohou být použity, se nazývá **syntaxe** jazyka. Program, který dodržuje syntaxi jazyka, ve kterém je vyjádřen, je považován za syntakticky správný a v případě odchylky od správné syntaxe, se jedná o syntaktickou chybu. Syntaktická správnost je obvykle předpokladem pro to, aby byl počítač schopen program vykonat.

Interpretace jednotlivých forem vyjádření v jazyce, se nazývá **sémantika** jazyka. [8]

1.2 Vlastnosti algoritmu

V kontextu zpracování dat počítačem, je nutné, aby algoritmus splňoval tyto vlastnosti:

1.2.1 Jednoduchost

Algoritmus je složen z jednoduchých kroků.

1.2.2 Obecnost

Také univerzálnost nebo hromadnost algoritmu znamená, že algoritmus neřeší jeden konkrétní problém pro konkrétní hodnoty, ale spíše množinu obdobných problémů.

1.2.3 Konečnost (finitnost)

Algoritmus musí skončit po určitém počtu kroků a jeho výsledek musí být poskytnut v rozumném čase, ať už je vstup jakýkoliv.

1.2.4 Jednoznačnost (determinovanost)

Zápis algoritmu je složen z několika jednoduchých kroků, které na sebe navazují. V kterémkoliv místě v algoritmu musí být jasné, co konkrétní krok právě řeší a jaký krok následuje po něm.

1.2.5 Resultativnost

Algoritmus musí produkovat výstup na základě vstupních dat. Hodnoty tohoto výstupu se považují za správné řešení problému. [2] [3] [4]

1.3 Zápis algoritmu

Algoritmus lze vyjádřit mnoha způsoby, například přirozeným jazykem, vývojovým diagramem, pseudokódem, matematickým zápisem a programovacím jazykem.

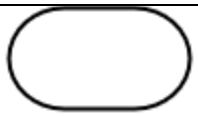
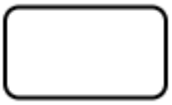
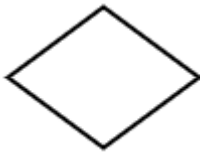
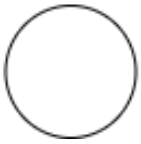
Přirozeným jazykem lze sice vyjádřit kroky algoritmu tak, aby je chápalo široké publikum, nicméně má své nevýhody, mezi něž patří hlavně jeho nejednoznačnost, neboť nemá přesně danou strukturu, a to ztěžuje dodržování jeho správnosti. Vývojové diagramy a pseudokódy jsou více strukturované formáty, v nichž může být algoritmus přesně vyjádřen. [5] [4]

1.3.1 Vývojové diagramy

Skrze vývojový diagram lze vyobrazit proces, systém nebo počítačový algoritmus. Je široce využíván v různých oborech k dokumentaci, plánování a vylepšení často složitých procesů tak, aby byly jasné a snadno srozumitelné. [6]

V kontextu programování se vývojový diagram používá jako nástroj na vytvoření plánu pro vyřešení problému. Využívají se v něm symboly, které jsou mezi sebou propojeny tak, aby byl naznačen směr proudu zpracování informací. [7]

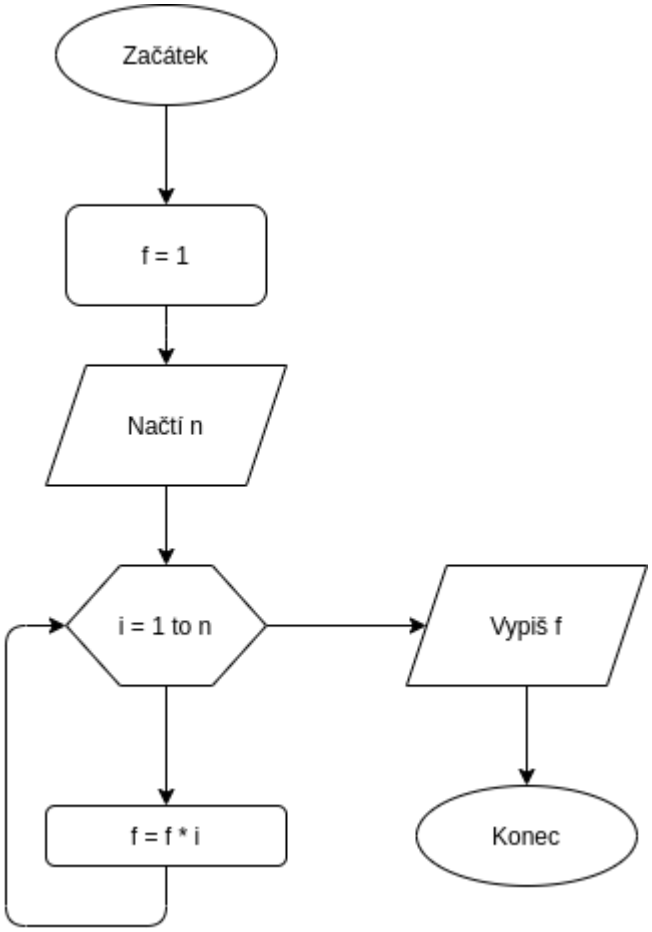
Tabulka 1: Základní symboly vývojových diagramů

Název	Symbol	Význam
Terminál		Toto je první a poslední symbol ve vývojovém diagramu, znázorňuje začátek a konec směru programovací logiky. Kromě konce programu, může také značit konec v případě, že narazí na chybovou podmínku.
Vstup a výstup		Tento symbol označuje jakoukoliv funkci typu vstup/výstup. Zapisují se do něj programové instrukce, které berou vstup ze vstupních zařízení nebo zobrazují výstup na výstupní zařízení.
Zpracování		Symbol čtverce představuje instrukci pro aritmetické procesy, jako jsou sčítání, odčítání, násobení a dělení.
Rozhodování		Toto je rozhodovací bod pro operace založené na rozhodovaní, pro něž je odpověď ano/ne nebo pravda/nepravda.
Spojka		V případě, kde se jedná o více komplexní vývojový diagram, který by přesahoval více než jednu stranu nákresu, se používá spojka na propojení rozdělených částí diagramu.

Šipky

Šipky naznačují přesné pořadí, v jakém mají být kroky algoritmu provedeny. Také reprezentují směr a vztah mezi ostatními symboly ve vývojovém diagramu. [7]

Následující vývojový diagram znázorňuje průběh algoritmu pro faktoriál. Vstupem pro algoritmus je jedno číslo *n* a výstupem bude vypsání výpis vypočteného faktoriálu *f*. Čísla od 1 do *n* projdou v cyklu a každá jeden cyklus se provede aritmetická operace *f * i*.



Obrázek 1: Vývojový diagram pro faktoriál

1.3.2 Pseudokód

Psaní pseudokódu je metoda, která umožňuje programátorům reprezentovat implementaci algoritmu. Taková implementace je ve formě informativního textu napsaného v prostém jazyce. Nemá syntax programovacího jazyka, a proto nemůže být zkompilován a počítač ho není schopen interpretovat. Algoritmy často bývají vyjádřeny pomocí pseudokódu, protože je v takové formě může přečíst každý programátor bez ohledu na to, jaký programovací jazyk umí nebo jak je v něm zběhlý.

Hlavním cílem pseudokódu je vysvětlit, co přesně má každý řádek programu dělat, což usnadňuje tvorbu konstrukce kódu programátorovi. Kromě toho se dá také využít jako hrubá dokumentace kódu. [9]

1.3.3 Matematický zápis

Tato forma zápisu je vhodná u popisování matematických vztahů, vzorců nebo rovnic. Je jednoznačná, ale v rámci jednoho zadání se může jednat o větší množství matematických vztahů, nicméně člověk znalý úpravy matematických výrazů je schopen určit, za jakých podmínek lze problém vyřešit. Nevýhodou matematického zápisu je, že jej nelze přímo předat počítači kvůli tomu, že bývá málo podrobný. [21]

1.3.4 Rozhodovací tabulky

Rozhodovací tabulky jsou kompaktní a přesný způsob modelování logiky, která může být použita v počítačovém programu. Tabulky nám usnadňují vidět všechny možné kombinace případů, které mohou nastat a jejich případné řešení. Obsahují podmínky, akce a stavy, které se odškrtávají. Stavy v tabulkách bývají většinou popsány jako *true* a *false* nebo 1 a 0.

Výhodou tabulek je, že jsou přehledné, jednoznačné a umožňují nám pozorovat každý stav, a to i ty, které by se jinak mohly přehlédnout, tudíž by nemusely být zavedeny v algoritmu. Na druhou stranu rozhodovací tabulky neobsahují instrukce krok za krokem, a proto musí být dále rozvedeny před svou implementací do programu. [21] [22]

1.3.5 Programovací jazyk

Programovací jazyky jsou umělé jazyky vytvořené pro zápis počítačového programu. Jsou strukturované a mají přesně definovanou syntaxi a sémantiku. Platí v nich omezující pravidla, například jaké slova v nich můžeme používat a jakým způsobem můžeme skládat věty. Mají více stupňů abstrakce a dají se rozdělit na strojově orientované jazyky a vyšší programovací jazyky, mezi něž patří právě například jazyk C#. [4]

2 VÝVOJOVÉ PROSTŘEDÍ MICROSOFT VISUAL STUDIO

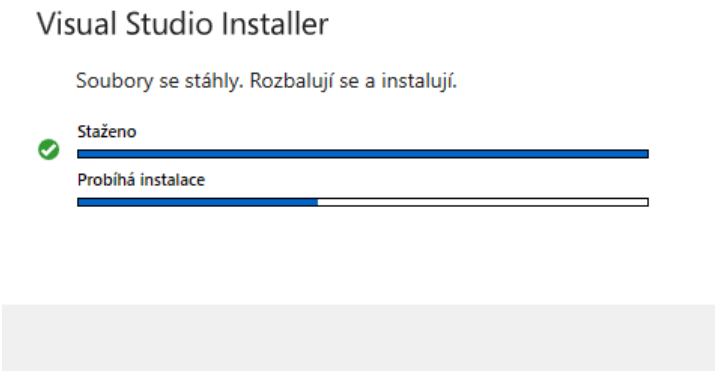
Visual Studio je vývojové prostředí od společnosti Microsoft, které je možno využít na úpravu, debugging, tvorbu kódu a publikování aplikací. Pracuje na operačních Windows 7 SP1 a všech pozdějších verzích. Nejnovější verze prostředí je *2019 version 16.10.0 Preview 1*. Podporuje třináct světových jazyků, včetně češtiny. Využívá se na tvorbu počítačových programů, webových stránek, webových aplikací, webových služeb a mobilních aplikací. [10][11][12]

Visual Studio podporuje třicet šest různých programovacích jazyků a umožňuje jejich editaci a debugging pro téměř každý z nich. Zabudované programovací jazyky zahrnují například C, C++, C# JavaScript, HTML, CSS, a další. Podporuje také jiné jazyky jako Python nebo Ruby pomocí dostupných pluginů. [13][14]

Základní edice Visual Studia je *Community Edition*, která je dostupná zdarma. [15]

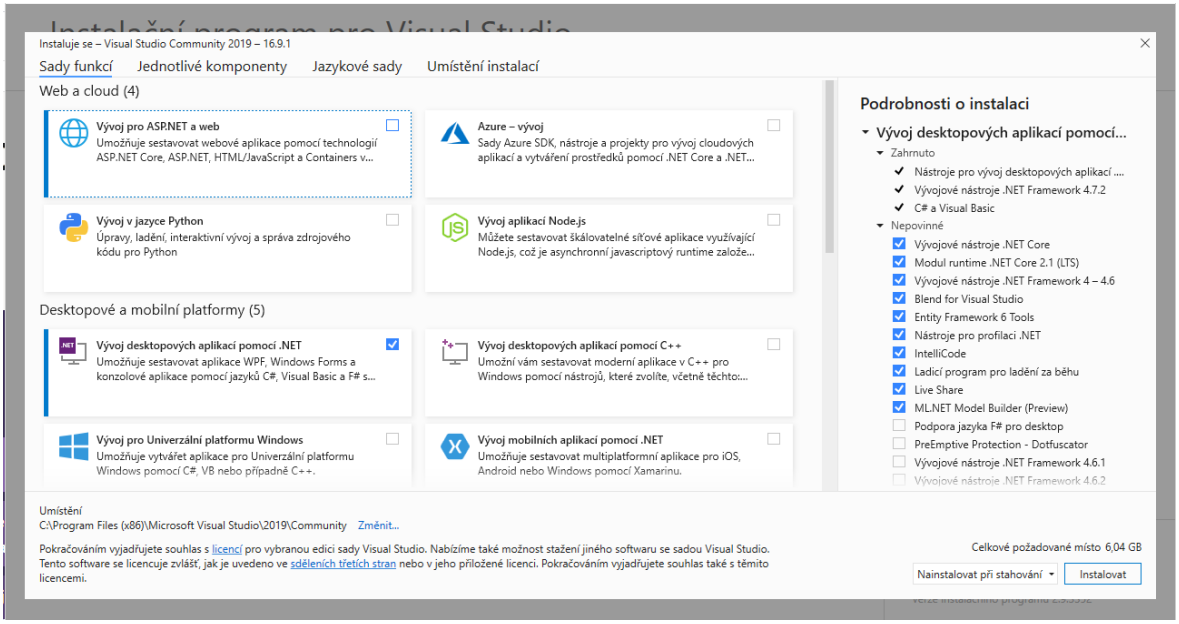
2.1 Instalace Visual Studio Community edice

Před instalací Visual Studia je vhodné zkontrolovat, zda počítač, na kterém jej instalujeme, splňuje systémové požadavky, má nejnovější aktualizace a dostatek místa na disku.



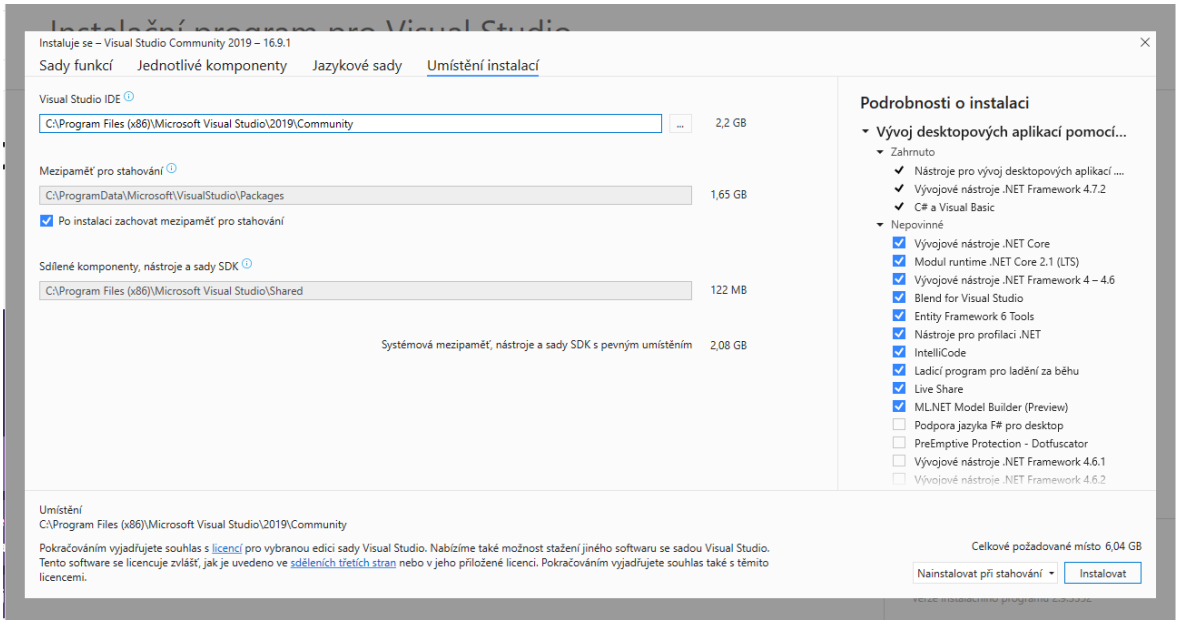
Obrázek 2: Instalace Visual Studio Installer

Po stažení instalačního programu, jej spustíme. Sám se nainstaluje a poté nám zobrazí dialogové okno, kde v záložce *Sady funkcí* zvolíme *Vývoj desktopových aplikací pomocí .NET*. Tato sada obsahuje mimo jiné tvorbu konzolových aplikací pomocí jazyku C#, což pro účely střední školy stačí. Ostatní sady se v případě potřeby dají přidat buď v nástrojích v již nainstalovaném Visual Studiu nebo přes tento Installer (viz obrázek níže).



Obrázek 3: Záložka Sady funkcí ve Visual Studio Installer

Výběr komponentů v další záložce pro naše účely není nutný. V záložce *Jazykové sady* bude automaticky navolen jazyk, který se shoduje s jazykem systému, na kterém program instalujeme a můžeme ho v případě potřeby změnit. V záložce *Umístění instalace* lze navolit jiné místo pro instalaci programu, automaticky Installer zvolí disk C.



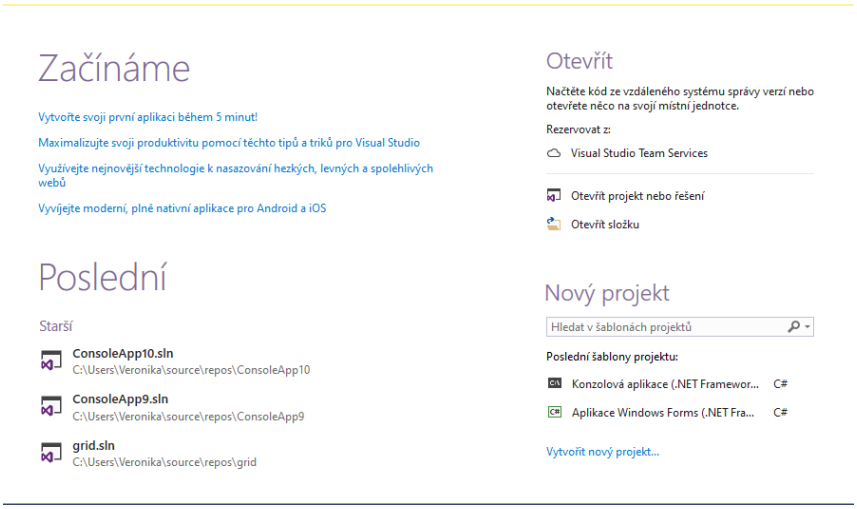
Obrázek 4: Záložka Umístění instalací ve Visual Studio Installer

Po zvolení instalačních kritérií, klikneme na tlačítko Instalovat, nyní se stáhnou příslušné sady a komponenty a Visual Studio se nainstaluje.

Po dokončení instalace se zobrazí dialogové okno s volbou motivu pro Visual Studio, kde je automaticky navolen modrý motiv. Po potvrzení motivu se Visual Studio spustí a můžeme vytvořit nový projekt. [16]

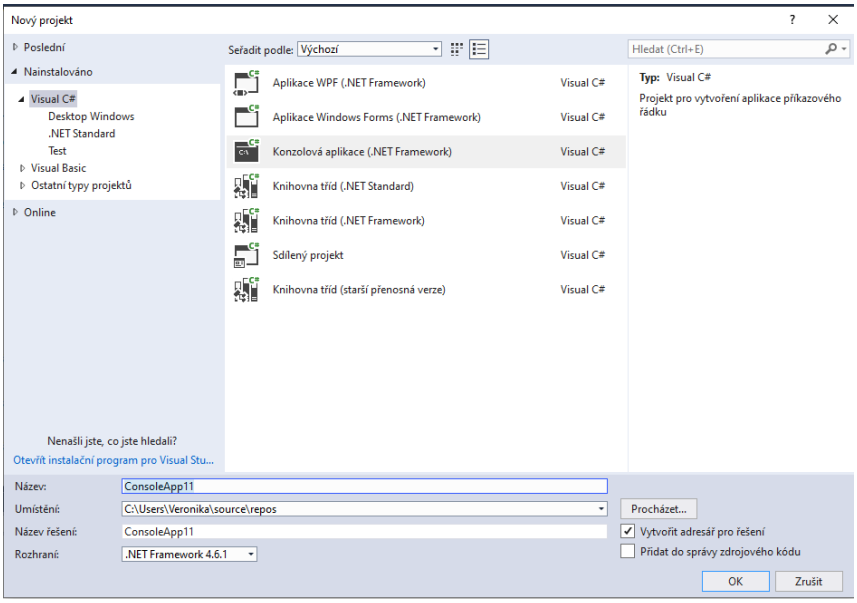
2.2 Tvorba projektu

Po spuštění Visual Studia se zobrazí uvítací obrazovka, kde je možné otevřít již existující projekt nebo založit nový. Názvy jednotlivých sekcí jsou intuitivní. Sekce *Začínáme* obsahuje odkazy na oficiální dokumentaci návodů od společnosti Microsoft. V sekci *Poslední* je nabídnuto několik projektů, na kterých uživatel pracoval naposledy a pro rychlý přístup je může hned otevřít. V sekci *Otevřít* lze procházet všechny existující projekty na disku.



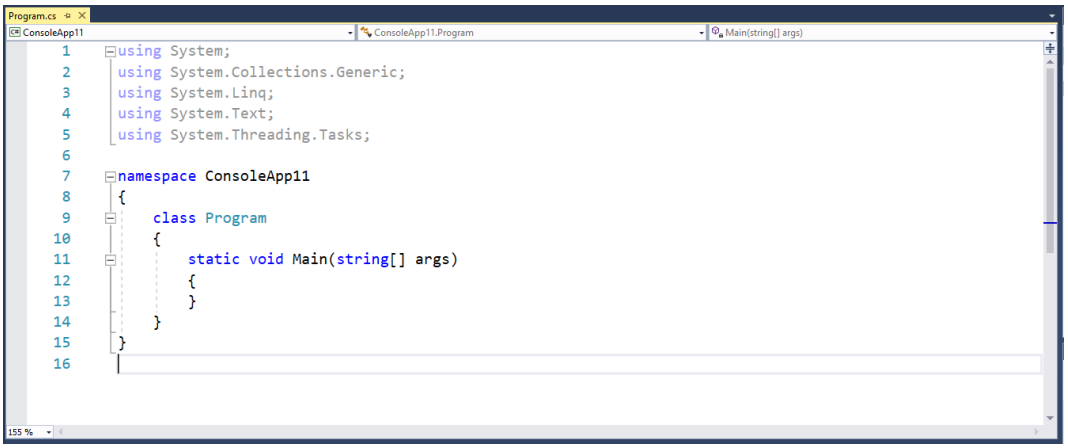
Obrázek 5: Uvítací obrazovka Visual Studia

Při tvorbě nového projektu, je třeba zvolit šablonu pro projekt, v našem případě je to šablona konzolové aplikace. Dále se zvolí název projektu, případně jeho umístění a dialogové okno se potvrdí.



Obrázek 6: Okno vytváření nového projektu

Nový projekt se vygeneruje s připraveným jmenným prostorem, třídou a hlavní metodou *Main*. Projekt je připraven na úpravu a psaní vlastního kódu.



Obrázek 7: Vygenerovaný nový projekt

2.3 Prvky Visual Studia

2.3.1 Editor kódu

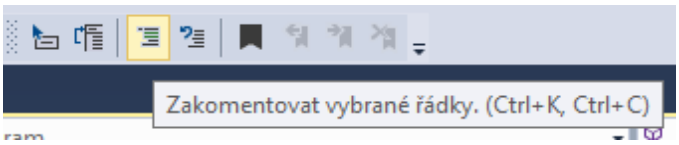
a) Šablony kódu

Visual Studio nabízí vkládání šablon kódu, což jsou krátké bloky kódu, které se často a opakovaně využívají. Mezi takové šablony patří například bloky jako *try-finally* nebo *if-else*. Šablony kódu se ale taky dají využít na vložení celých tříd nebo metod. Šablony jsou dostupné pro několik programovacích jazyků, včetně jazyka C#. Do přehledu všech šablon

se dá nahlédnout v záložce *Nástroje* a pak *Správce fragmentace kódu*. Vložit se dají přes záložku *Upravit*, pak *IntelliSense* a *Vložit fragmentaci kódu*. [17]

b) Komentování kódu

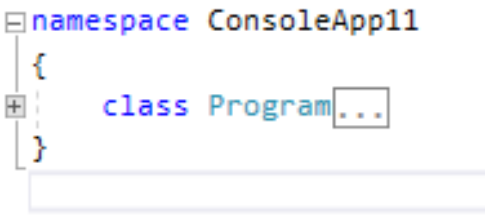
Visual Studio umožňuje kód zakomentovat znaky `//` pro jednořádkové komentování, anebo `/* zakomentovaný kód */` pro víceřádkové komentování. Visual Studio má také příkaz pro komentování kódu ve svém panelu nástrojů.



Obrázek 8: Tlačítko pro zakomentování řádků kódu

c) Zabalení kódu

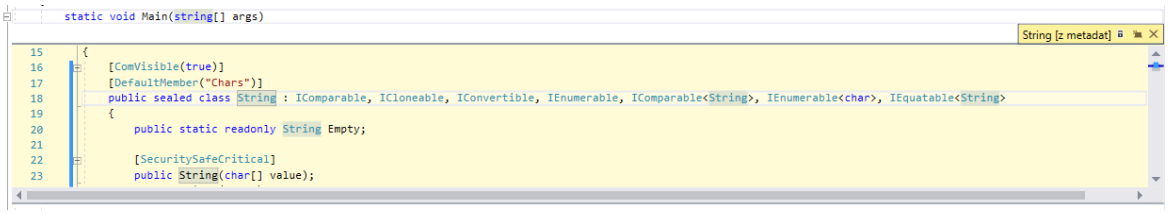
Pro přehlednější kód je možné zabalit a skrýt konstruktory celých tříd.



Obrázek 9: Zabalená třída Program

d) Definice symbolů

Editor ve Visual Studiu umožňuje nahlédnout do definicí typů, metod a dalších. To lze udělat kliknutím pravým tlačítkem na jejich výskyt a poté *Náhled definice* nebo zkratkou alt+12. Zobrazí se okno s definicí.



Obrázek 10: Náhled do definice datového typu string

e) Refaktorování

Refaktorování je proces modifikování kódu za účelem vytvořit ho jednodušší na údržbu a porozumění, aniž by se změnila jeho funkcionalita. Umožňuje například hromadně přejmenovat všechny proměnné se stejným názvem. [19]

f) IntelliSense

IntelliSense je důležitý nástroj při programování, který umí zobrazovat informace o dostupných typech nebo parametrech různých metod. Také se dá využít na automatické doplňování slov. Tyto prvky nám pomáhají lépe porozumět kódu, který píšeme, sledují parametry, které píšeme a přidávají volání vlastnostem a metodám několika stisky kláves.

Mnoho aspektů technologie IntelliSense jsou specifické pro jednotlivé jazyky. [18]

g) Debugger

Debbuger je nástroj, který nám pomáhá ladit a odhalit chyby v kódu při jeho vývoji. Cílem tohoto nástroje je spustit program s nastavenými podmínkami, které nám pomohou sledovat operaci ve svém procesu a monitorovat tak provedené změny, které mohou ukázat na chybný kód.

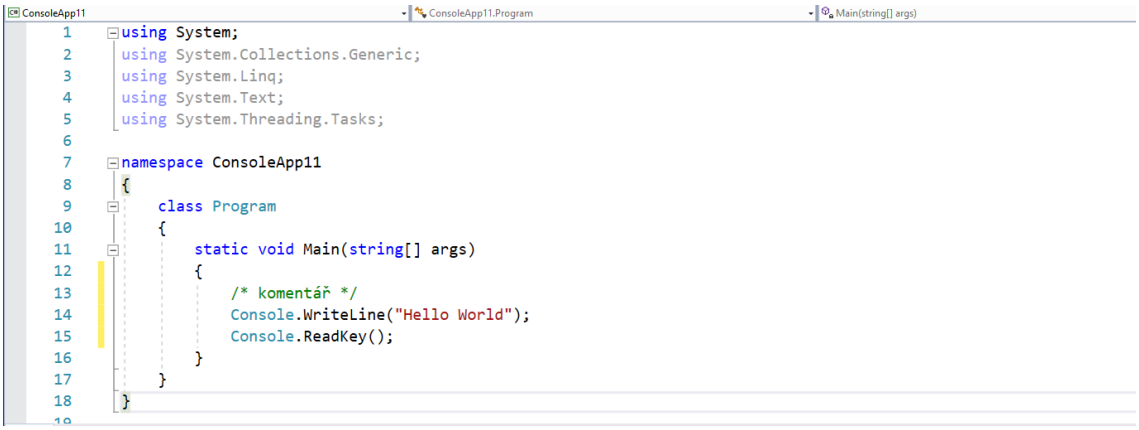
Ve Visual Studiu se do módu ladění vstupuje buď stisknutím tlačítka F5 nebo přes panel nástrojů *Ladit* a *Spustit ladění*. Součástí ladění je také využití zarážek (breakpoints), což jsou indikátory toho, kde má Visual Studio pozastavit běh kódu, abychom se mohli podívat na aktuální hodnoty v proměnných nebo na chování paměti. Zarážky se nastaví buď v levém panelu vedle řádku kódu nebo umístěním kurzoru na řádek a stisknutím klávesy F9. [20]

3 PROGRAMOVACÍ JAZYK C#

3.1 Základní struktura zdrojového kódu jazyka C#, překlad zdrojového kódu, ladění, profilace, preprocessing zdrojového kódu

3.1.1 Základní struktura zdrojového kódu jazyka C#

Základní části programu v jazyce C# se mohou skládat z následujících částí: jmenný prostor, třída, metoda třídy, vlastnosti třídy, hlavní metoda, příkazy, výrazy a komentáře.



Obrázek 11: Ukázka jednoduchého programu

Výše uvedená ukázka po zkompilování a spuštění zobrazí text `Hello World`.

První řádek programu *using System;* obsahuje klíčové slovo *using*, které má funkci zahrnout jmenný prostor *System* do programu. Programy mívají vícero příkazů *using.* Zašedlé *using* výrazy znamenají, že jsou zahrnuty do programu, ale nikde se nevyužívají.

Dále je deklarace *jmenného prostoru*, což je kolekce tříd. Náš *jmenný prostor* se jmenuje *ConsoleApp11* a obsahuje třídu *Program*.

Na dalším řádku je deklarace *třídy Program*, která obsahuje data a definice metod, které program používá. Metody definují chování tříd a třída může mít vícero metod, nicméně naše třída má právě jednu metodu *Main*.

Následuje hlavní metoda *Main*, což je vstupní bod pro každý program v jazyku C#. Hlavní metoda vyjadřuje, co třída dělá po spuštění.

Řádek s komentářem `/* komentář */` bude kompilátorem ignorován a slouží pro zapsání poznámek a orientaci v kódu programátorům.

Hlavní metoda specifikuje své chování příkazem *Console.WriteLine("Hello World");*, což je metoda pro třídy *Console*, která je definována jmenným prostorem *System*. Tento příkaz vypíšu zprávu „Hello, World!" na konzoli na obrazovce.

Na posledním řádku je příkaz *Console.ReadKey();*, který způsobuje, že program čeká na uživatele, až stiskne libovolnou klávesu a tím zabraňuje tomu, aby se program hned po svém otevření sám zavřel.

Mezi další pravidla, na která je třeba myslet, patří například, že C# je citlivý na velikost písmen, každý výraz a příkaz musí končit středníkem, program se spouští v hlavní metodě. [23]

3.1.2 Překlad zdrojového kódu

Jazyk C# je navrhnut jako vyšší programovací jazyk, to znamená, že je více abstraktní a programátor se při psaní kódu vyhne péči o věci, jako například správu paměti nebo jak jeho program bude fungovat na různém hardwaru.

O překlad mezi kódem napsaném v C# do strojového kódu, se stará *překladač* (compiler). Při spuštění překladač vezme náš kód jako vstup, zpracuje ho a jako výstup bude program přepsaný do *Intermediate Language* (což je nejnižší člověkem čitelný programovací jazyk) a ten pak uloží například jako spustitelný soubor (.exe). Ten ale procesor stále nemůže přečíst, takže se konvertuje dál na *Common Language Runtime*. Ten používá *Just-In-Time* kompilátor pro překlad do strojového kódu, který už procesor přečíst dokáže. [24]

3.1.3 Ladění

Ladění je metoda odstraňování chyb v kódu. Nejjednodušší způsob, jakým lze kontrolovat kód, je takzvaný *print debugging*, což je postup, při kterém naše aplikace někam vytiskne text nebo číslo, abychom viděli, kam až se kód bezchybně dostal a jaké jsou hodnoty v proměnných. V C# se k tomuto postupu používá metoda *Console.Write()*, což je metoda, jejíž výstup je vytiskut na konzoli. To do určité míry může stačit, nicméně ve Visual Studio, je pro tyto účely zabudovaný debbuger, který byl popsán v předchozí kapitole. [25]

3.1.4 Profilace

Profilace je metoda analýzy kódu, u které se měří například místo v paměti nebo časová složitost programu. Tradiční profilovací nástroje se soustředí na měření času, který spotřebuje každá funkce, nebo kolik paměti běžící aplikace využije. Kromě měření také

monitoruje aplikaci při spuštění. Z toho důvodu API (Application Programming Interface), která profiluje, by neměla být využívána samotnou aplikací a její spuštění by na ni nemělo být závislé.

Visual Studio má pro C# zabudované nástroje pro profilaci, které pomáhají diagnostikovat různé typy problémů s výkonem v závislosti na typu aplikace. K profilovacím nástrojům lze přistoupit už během ladění skrze dialogové okno *Diagnostic Tool*. Během debbugingu tedy lze analyzovat využití procesu a paměti. [26] [27]

3.1.5 Preprocessing zdrojového kódu

Koncept preprocesoru náleží oblasti kompilovaných jazyků, jako jsou C a C#, jejichž kód je pomocí kompilátoru přeložen do strojového kódu. Preprocessor negeneruje strojový kód, ale připravuje kód pro kompilátor. Obsahuje direktivy, které jsou vyhodnoceny předem a mají dopad na kompilační proces. Preprocesory se většinou berou jako oddělené entity od kompilátorů. [28]

3.2 Základní datové typy jazyka C# a operace nad nimi, proměnné a rozsah jejich platnosti

C# je silně typovaný jazyk. Každá proměnná a konstanta mají typ, stejně jako každý výraz, u kterého se vyhodnocuje hodnota. Každý typ má specifikované vstupní parametry a návratovou hodnotu. Knihovna .NET má definovanou řadu zabudovaných jak numerických typů, tak i složitějších typů, které reprezentují širokou škálu logických konstruktů jako jsou kolekce, pole objektů a jiných. Typický program v jazyku C# využívá typy z knihoven, ale také typy definované uživatelem, které jsou vytvořeny, aby odpovídaly konceptu, který je specifický pro daný problém, jež program řeší.

3.2.1 Základní datové typy

Tabulka 2: Přehled datových typů [29] [30] [31]

Typ	Rozsah	Velikost v paměti
byte	0 až 255	1 bajt
int	-2,147,483,648 až 2,147,483,647	4 bajty
long	-9,223,372,036,854,775,808 až 9,223,372,036,854,775,807	8 bajtů
float	-1.5 x 1045 až 3.4 x 1038	4 bajty
double	-5 x 10324 až 1.7 x 10308	8 bajtů
decimal	-1028 až 7.9 x 1028	24 bajtů
char	0 až 65535	2 bajty
bool	True nebo False	1 bit
string	až 2 biliony bajtů	4 bajtová adresa

3.2.2 Operace s datovými typy

C# nabízí mnoho operátorů. Spoustu z nich jsou podporovány zabudovanými datovými typy a dovolují nám s nimi provádět základní operace. Tyto operátory zahrnují následující skupiny: [32]

a) Aritmetické operace

Tento typ se dále rozděluje na:

1. **Unární**

U unárních operátorů se pracuje s jednou proměnnou. Při jejich použití se proměnná zvýší nebo sníží o jednu jednotu. Patří mezi ně inkrementace a dekrementace.

2. **Binární**

Tyto operátory fungují stejně, jak by se očekávalo v matematických operacích. Pro jejich použití jsou potřeba vždy alespoň dvě proměnné. Patří mezi ně součet, součin, násobení, dělení a modulo. [33]

b) Složené přiřazení

Tyto operátory jsou kombinací přiřazení hodnoty do proměnné a aritmetické operace.

c) Relační operátory

Tyto operátory porovnávají vztah mezi dvěma proměnnými. Některé z nich jsou převzaté z matematiky a mají zde stejnou funkčnost.

Tabulka 3: Přehled relačních operátorů

X a Y se rovnají	x == y
X a Y se nerovnají	x != y
X je menší než Y	x < y
X je větší než Y	x > y
X je menší nebo rovno Y	x <= y
X je větší nebo rovno Y	x >= y

d) Logické operátory

Porovnávají vztah mezi dvěma proměnnými, vycházejí z Booleanovi logiky. [32]

Konjunkce – výrok je pravdivý, pokud X a Y mají pravdivou hodnotu.

Tabulka 4: Konjunkce X&&Y

X	**Y**	**X ∧ Y**
0	0	0
0	1	0
1	0	0
1	1	1

Disjunkce – výrok je pravdivý, pokud alespoň jedna proměnná má pravdivou hodnotu

Tabulka 5: Disjunkce X||Y

X	**Y**	**X ∨ Y**
0	0	0
0	1	1
1	0	1
1	1	1

3.2.3 Proměnné a rozsah jejich platnosti

Proměnná je jméno, které zadáváme místu v paměti, se kterým náš program manipuluje. Každá proměnná v C# má konkrétní typ, který určuje její velikost, rozsah, v jakém může být uložena a řadu operací, které na takovou proměnnou mohou být aplikovány. [35]

V C# existují pouze lokální proměnné, která se dají využít v oblasti, ve které byly deklarovány, například v metodě nebo třídě. Mohou být využity pouze výrazy, které jsou uvnitř oné funkce nebo bloku kódu. [34]

3.3 Řídící struktury

Řídící struktura je pořadí, v jakém jsou jednotlivé příkazy, instrukce a funkce zavolány.

3.3.1 Sekvence

Sekvence je případ, kde jsou všechny příkazy provedeny postupně jeden po druhém. Tento případ nevyžaduje žádné speciální příkazy a celá jeho konstrukce je vykonána lineárně. [36]

3.3.2 Větvení

V případě větvení, nebo také podmíněného příkazu, se určitý příkaz buď provede nebo neprovede v závislosti na splnění podmínky.

a) If

Syntaxe:

```
if (x > y)
{
    Console.WriteLine("X je větší než Y.");
}
```

Sémantika: *If* je příkaz, který pomáhá určit, zda je podmínka splněna nebo ne. Pokud je podmínka splněna, tak se provede kód, který je v tom konkrétním bloku, pro daný *if* příkaz.

b) If-else

Syntaxe:

```
if (x > y)
{
   Console.WriteLine("X je větší než Y.");
}
else
{
   Console.WriteLine("X je menší než Y.");
}
```

Sémantika: Pokud chceme specifikovat, co se stane v případě, že podmínka není splněna, tak za blok kódu s *if* následuje *else*. Je možno použít několik *if-else* příkazů za sebou, nicméně čím delší je takový kód, tím horší je na údržbu a lze místo něj použít *switch-case*. [37]

c) Switch a case

Syntaxe:

```
switch (promenna)
{
    case 1:
        Console.WriteLine("Case 1");
        break;
    case 2:
        Console.WriteLine("Case 2");
        break;
    case 3:
        Console.WriteLine("Case 3");
        break;
    default:
        Console.WriteLine("Default");
        break;
}
```

Sémantika: *Switch-case* funguje na stejném principu, jako *if-else*, ale je ho vhodné použít, pokud potřebujeme zkontrolovat více podmínek. Příkazu *switch* se přiřadí proměnná, jejíž změnu bude příkaz kontrolovat a jednotlivá východiska případů se vypíší do příkazů *case.* Každý *case* se zakončí příkazem *break*. Pokud nebylo nalezeno žádné východisko podle příkazů *case*, podmínka spadne do příkazu *default.* [38]

3.3.3 Cykly

Cykly jsou příkazy, které se opakují určitý počet iterací. Tento počet může být pevný, ale je možné, aby se cyklus opakoval do nekonečna.

a) For

Syntaxe:

```
for (int i = 0; i < 10; i++)
{
     Console.WriteLine(i);
}
```

Sémantika: Cyklus *for* má pevný počet opakování a vkládají se do něj následující části: proměnná s počáteční hodnotou, podmínka pro vykonání dalšího cyklu a příkaz, který říká, co se s řídící proměnnou má každý cyklus stát. Pak následuje tělo *for* cyklu, tedy blok kódu, který se vykoná každý jeden cyklus. Z cyklu lze kdykoliv vyskočit; cyklus lze kdykoliv zastavit pomocí příkazu *break* nebo přeskočit na další iteraci pomocí příkazu *continue*. Z *for* cyklu lze také vyskočit pomocí *goto, return* nebo *throw*. [39] [40]

b) While

Syntaxe:

```
while (x > 0)
{
    x--;
}
```

Sémantika: Cyklus *while* se na druhou stranu dá použít i v případě, kdy nevíme, kolik iterací budeme potřebovat, než se dostaneme k požadovanému výsledku. Do *while* se vkládá pouze podmínka pro vykonání cyklu a následně tělo cyklu s kódem. Tělo cyklu se vykoná za předpokladu, že vložená podmínka má pravdivý výsledek. Z cyklu lze kdykoliv vyskočit pomocí příkazu *break*. [41]

c) Do-while

Syntaxe:

```
do
{
  x++;
} while (x < 0);
```

Sémantika: Cyklus *do-while* funguje stejně jako příkaz *while* s tím rozdílem, že *do-while* se vykoná alespoň jednou a jeho podmínka se kontroluje až po vykonání cyklu. Protože u *while* se kontroluje podmínka předem, nemusí se vykonat vůbec. [42] [43]

d) Foreach

Syntaxe:

```
foreach (int prvek in pole)
{
     Console.WriteLine(prvek);
}
```

Sémantika: *Foreach* je příkaz, který projde každý prvek v kolekci, například v poli. Pro *foreach* se vkládá datový typ a proměnná, pak následuje kolekce, kterou chceme procházet. Cyklus projde každý prvek v kolekci, aniž by musel znát celou její délku, a v každé iteraci ho uloží do dané proměnné. [44] [45]

3.4 Pole a seznamy

Pole i seznamy jsou si podobné v tom, že oba ukládají větší množství proměnných daného typu a liší se v tom, jak se s nimi zachází.

a) Pole

Sémantika: U každého nového pole se deklaruje datový typ, jaký bude pole uchovávat, jeho jméno a délku pole. Každé pole má vlastnost *length*, kde je uložena jeho délka, dále .NET poskytuje třídu *array*, která obsahuje pomocné metody pro práci s poli, například *sort()*, *reverse()*, *copy()*, a další. [46] [47]

Syntaxe:

```
int[] pole = new int[5] { 1, 2, 3, 4, 5 };
```

b) Seznam

Sémantika: Seznam je kolekce, která má uložené prvky a je u ní možno je za běhu programu přidávat a mazat. Samotný seznam má implementované metody jako *AddRange(), Count, Find*. Má také implementovaný interface *IList*, který obsahuje metody pro další práci s prvky, jako například *add()*, *clear()*, *contains()*, a jiné. [48] [49]

Syntaxe:

```
List<int> seznam = new List<int>();
```

3.5 Struktury a třídy

Struktury a třídy jsou základní stavbou typového systému v .NET. Jsou to datové struktury, které shrnují sadu dat a chování, jež patří k sobě jako logický soubor. Data a chování jsou členy třídy nebo struktury a zahrnují metody, vlastnosti a události.

Deklarace třídy nebo struktury jsou jako návrh, který je pak použitý k vytvoření instancí nebo objektů za běhu.

a) Struktura

Sémantika: Struktura je uživatelem definovaný datový typ, který shrnuje data a související funkcionalitu. Může obsahovat konstruktor s parametry, statický konstruktor, konstanty, pole, metody, vlastnosti, události, a jiné. Struktura může být využita na uložení malých datových hodnot, po kterých nepožadujeme dědičnost, takže například body souřadnic. Struktura se deklaruje použitím slova *struct*. [50] [51] [52]

Syntaxe:

```
struct Souradnice
{
     public int x;
     public int y;
}
```

b) Třída

Sémantika: Třída je jako návrh pro specifický objekt, který definuje vícero charakteristik. Třída se definuje vlastnostmi, poli, událostmi, metodami, a dalšími. Definuje datové typy a funkcionality, které daný objekt má. Třída nám umožňuje vytvořit vlastní typ seskupením proměnných různých typů, metod a událostí. Může obsahovat modifikátor přístupu, pole, konstruktor, metodu, vlastnosti. Deklaruje se slovem *class.* [53] [54]

Syntaxe:

```
public class Trida
{
   public string PoleTridy;

   public Trida(string PoleTridy)
   {
       //konstruktor
       this.PoleTridy = PoleTridy;
   }

   public void Metoda(int parametr1, string parametr2)
   {
       //funkce metody
   }
```

```
    public int Vlastnost
    {
        get { return Vlastnost; }
        set { Vlastnost = value; }
    }
}
```

3.6 Metoda

Sémantika: Metody nám dovolují shrnout kus kódu do jednoho místa v programu a pak ho volat z jiné části kódu. Tím se eliminuje časté opakování kódu, který by byl použitý ve vícero místech v programu. Metodě se přiřadí její viditelnost, návratový typ, jméno a vstupní parametry, dovnitř se pak píše funkcionalita, kterou požadujeme. [55] [56] [57]

Syntaxe:

```
static double MojeMetoda(double a)
{
     double obvod = 4 * a;
     return obvod;
}
```

3.7 Objektově orientované programování

Objektově orientované programování je moderní metodika vývoje softwaru. Klade důraz na to, aby se kód skládal z jednotlivých komponentů, dal se znovu použít a snadno udržovat. Je to způsob myšlení, který vše interpretuje jako objekt, který může mít vlastnosti a vztahy s dalšími objekty.

3.7.1 Vlastnosti objektově orientovaného programování

a) Objekty

Objekty jsou výsledkem seskupení jednotlivých prvků, které jsou modelovány podle reality. Skládají se z vlastností a dat a mohou se také skládat z metod, pokud obsahují nějaké operace.

b) Abstrakce

S objektem se pracuje jako s černou skříňkou. Každý objekt má svou funkci, se kterou pracujeme, ale nemusíme nutně vědět, jak přesně danou funkci objekt vykonává.

c) Skládání

Objekt se může skládat z dalších objektů.

d) Delegování

Objekt může spolupracovat s jinými objekty a využívat jejich služeb.

e) Zapouzdření

Zapouzdření zaručuje skrytí vnitřního stavu objektů. S objekty lze komunikovat z venku, což zpřístupňuje rozhraní jako je modifikátor přístupu nebo jmenný prostor.

f) Dědičnost

Objekty jednoho druhu mohou dědit z objektů jiného druhu a přebrat tak nějakou schopnost, ke které přidají vlastní rozšíření. To se projevuje tak, že třída může být rodičovská a třídy, které od ní kód dědí, jsou její potomci.

g) Polymorfismus

S polymorfismem pracujeme, pokud máme více tříd, které jsou spolu spjaté skrze dědičnost. Používá metody dědičnosti k vykonání různých úloh. Touto cestou můžeme provést jednu akci mnoha způsoby.

3.7.2 Modifikátory přístupu

Modifikátory přístupu souvisí se zapouzdřením. Jde o klíčová slova, která se zapisují objektům při jejich deklaraci a určují tak, která část kódu může s objekty pracovat.

a) Public

Přístup není omezený a k objektu lze přistupovat odkudkoliv z kódu.

b) Protected

Přístup je omezený a k objektu může přistupovat pouze třída, kterou je objekt součástí.

c) Internal

Přístup je omezen pouze na rámec souborů, které jsou ve stejném sestavení.

d) Protected internal

Přístup je omezen pouze na rámec souborů, které jsou ve stejném sestavení a zároveň k objektu může přistupovat třída, kterou je objekt součástí.

e) Private

Přístup omezuje použití členů tak, že jsou přístupné pouze v těle své třídy nebo struktury.

f) Private protected

Přístup omezuje použití členů tak, že jsou přístupné pouze v těle své třídy nebo struktury, ale pouze v rámci nadřazeného sestavení.

3.7.3 Třída

V obsáhlých programech s mnoha objekty je třeba je nějak roztřídit nebo klasifikovat podle podobných charakteristik. K takovému rozdělení používáme třídy. Objekty patřící do konkrétní třídy se nazývají instancí třídy.

Třída může obsahovat:

a) Fields

Field je proměnná jakéhokoliv typu, která se deklaruje ve třídě. Dávají se jí modifikátory přístupu a může být použita ostatními členskými prvky třídy.

b) Konstruktor

Konstruktor třídy je metoda slouží k inicializaci objektů. Má stejný název jako třída a nemá návratový typ. Do jeho parametrů se vkládají fieldy.

c) Property

Property třídy obsahuje metody *get* a *set*, které umožňují nastavovat a vracet hodnoty. Tyto operace by měly být časově jednoduché.

d) Metody

Metoda je kus kódu, který obsahuje řadu příkazů, které se provedou při jejím zavolání. Metodám tříd se při deklaraci specifikuje modifikátor přístupu, návratová hodnota a parametry metody.

[58] [59]

II. PRAKTICKÁ ČÁST

4 VYUČOVACÍ HODINY

V této kapitole jsou popsány jednotlivé vyučovací hodiny, jejich obsah, vyhotovení a možné výstupy zadaných příkladů. Cílem vyučovacích hodin je seznámit žáky se základy programování v jazyku C#.

Seznam prezentací:

1. C# a vývojové prostředí
2. Proměnné
3. Výrazy a příkazy
4. Podmíněný příkaz
5. Cykly
6. Pole
7. Seznamy
8. Struktury
9. Třídy
10. Metody

4.1 Úvod do C# a prostředí

Cíle vyučovací hodiny: Žáci se stručně seznámí s popisem jazyka C# a jeho historií a vývojovým prostředím Visual Studio.

Obsah vyučovací hodiny: Na vyučovací hodině jsou žáci seznámeni s charakteristikami a využitím jazyka C#, s možnostmi Microsoft Visual Studia a částmi nově založené konzolové aplikace.

Vyhotovení: Žáci se seznámí s prezentací.

4.2 Proměnné

Cíle vyučovací hodiny: Žák se seznámí s pojmy proměnná a datový typ, dokáže je správně deklarovat a dosazovat hodnoty.

Obsah vyučovací hodiny: Na vyučovací hodině jsou žáci seznámeni s pojmem proměnná, její deklarací, s datovými typy, jejich využitím a rozsahem.

Vyhotovení: Žáci se seznámí s prezentací, zodpoví na otázky na konci prezentace společně s učitelem a na příkladech pracují samostatně.

4.3 Výrazy a operátory

Cíle vyučovací hodiny: Žáci se seznámí s různými typy operátorů a přiřazení a dokáže je správně využít.

Obsah vyučovací hodiny: Na začátku hodiny je opakování na téma proměnné. Dále se žáci seznámí s novým tématem operátorů a přiřazení, které jsou vysvětleny na konkrétních příkladech.

Vyhotovení: Žáci si na začátku prezentace zopakují téma proměnná a zodpoví na otázky, pak se seznámí se zbytkem obsahu prezentace, na konci zodpoví otázky a na příkladech pracují samostatně.

4.4 Podmíněný příkaz

Cíle vyučovací hodiny: Žáci se seznámí s podmíněnými příkazy *if*, *else-if*, *if-else* a *switch* a jejich zápisem a dokáže je správně využít.

Obsah vyučovací hodiny: Na začátku hodiny je opakování na téma výrazy a operátory. Dále se žáci seznámí s novým tématem podmíněných příkazů, který je vysvětlen na konkrétních příkladech.

Vyhotovení: Žáci si na začátku prezentace zopakují téma výrazy a operátory a zodpoví na otázky, pak se seznámí se zbytkem obsahu prezentace, na konci zodpoví otázky společně s učitelem a na příkladech pracují samostatně.

Možné výstupy:

1. **příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace podminene_prikazy
{
    class Program
    {
        static void Main(string[] args)
        {
            // Program, který bude vypočítávat hodnotu BMI
            // a výsledek přiřadí do příslušné kategorie.
```

```
            double vyska = 189.0;
            double vaha = 140.1;
            double BMI = 0.0;

            BMI = vaha / Math.Pow(vyska / 100, 2);

            if (18.5 > BMI)
            {
                Console.WriteLine("Podvýživa");
            }
            else if (BMI >= 18.5 && BMI <= 24.9)
            {
                Console.WriteLine("Normální váha");
            }
            else if (BMI >= 25 && BMI <= 29.9)
            {
                Console.WriteLine("Nadváha");
            }
            else if (BMI >= 30)
            {
                Console.WriteLine("Obezita");
            }
        }
    }
}
```

2. příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace podminene_prikazy
{
    class Program
    {
        static void Main(string[] args)
        {
            // Program, který určí, zda předaná proměnná
            // je liché nebo sudé číslo.

            int cislo = 1;

            int vypocet = cislo % 2;

            if (vypocet == 0)
            {
                Console.WriteLine("Cislo sude");
            }
            else if (vypocet == 1)
            {
                Console.WriteLine("Cislo liche");
            }
        }
    }
}
```

3. příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace podminene_prikazy
{
    class Program
    {
        static void Main(string[] args)
        {
            // Program, který vyhodnotí číselné pořadí
            // písmena v abecedě.
            // Funguje pro první 4 písmena, jako ukázka.

            char pismenko = 'C';

            switch (pismenko)
            {
                case 'A':
                    Console.WriteLine("Prvni pismeno v abecede");
                    break;
                case 'B':
                    Console.WriteLine("Druhe pismeno v abecede");
                    break;
                case 'C':
                    Console.WriteLine("Treti pismeno v abecede");
                    break;
                case 'D':
                    Console.WriteLine("Ctvrte pismeno v abecede");
                    break;
                default:
                    Console.WriteLine("Nenaslo se v abecede");
                    break;
            }
        }
    }
}
```

4. příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace podminene_prikazy
{
    class Program
    {
        static void Main(string[] args)
        {
            // Program, který vypíše, o jaký geometrický tvar
            // se jedná podle počtu jeho stran.

            int pocet_stran = 5;
            switch (pocet_stran)
            {
                case 3:
```

```
                    Console.WriteLine("trojuhelnik");
                    break;
                case 4:
                    Console.WriteLine("ctverec");
                    break;
                case 5:
                    Console.WriteLine("petiuhelnik");
                    break;
            }
        }
    }
}
```

Umístění řešení: Vybrané ukázkové řešení je zpracováno v datové příloze ve složce prilohy/cv03_podminene_prikazy. Dílčí kódy jsou pak uloženy jako:

cv03_podminene_prikazy_01.cs

cv03_podminene_prikazy_02.cs

cv03_podminene_prikazy_03.cs

cv03_podminene_prikazy_04.cs

4.5 Cykly

Cíle vyučovací hodiny: Žáci se seznámí s cykly *for*, *while*, *do-while* a jejich zápisem a dokáže je správně využít.

Obsah vyučovací hodiny: Na začátku hodiny je opakování na téma podmíněné příkazy. Dále se žáci seznámí s novým tématem cyklů, které jsou vysvětleny na konkrétních příkladech.

Vyhotovení: Žáci si na začátku prezentace zopakují téma podmíněné příkazy a zodpoví na otázky, pak se seznámí se zbytkem obsahu prezentace, na konci zodpoví otázky společně s učitelem a na příkladech pracují samostatně.

Možné výstupy:

1. **Příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            // opakování
            // program, který bude hodnoty 0-100
            // přiřazovat k příslušným známkám
            int body = 90;

            if (body < 50)
            {
                Console.WriteLine("F");
            }
            else if (body >= 50 && body <= 59)
            {
                Console.WriteLine("E");
            }
            else if (body >= 60 && body < 69)
            {
                Console.WriteLine("D");
            }
            else if (body >= 70 && body < 79)
            {
                Console.WriteLine("C");
            }
            else if (body >= 80 && body < 89)
            {
                Console.WriteLine("B");
            }
            else if (body >= 90 && body <= 100)
            {
                Console.WriteLine("A");
            }
        }
    }
}
```

2. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
    class Program
    {
        static void Main(string[] args)
        {
            // program, který převede roky na dny

            int dny = 365;
            int rok = 5;
            int soucet = 0;

            for (int i = 1; i <= rok; i++)
            {
                if (i % 4 == 0)
                    soucet += 1;
                soucet += dny;
```

```
            }

            Console.WriteLine(soucet);
        }
    }
}
```

3. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
    class Program
    {
        static void Main(string[] args)
        {
            // manualni capslock
            char pismenko = 'g';
            bool vysledek = Char.IsLower(pismenko);

            switch (vysledek)
            {
                case true:
                    pismenko = Char.ToUpper(pismenko);
                    Console.WriteLine(pismenko);
                    break;
                case false:
                    Console.WriteLine(pismenko);
                    break;
            }
        }
    }
}
```

4. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
    class Program
    {
        static void Main(string[] args)
        {
            // cyklus while, tkerý bude vypisovat i
            // dokud i bude menší než 7

            int i = 15;
            while (i >= 7)
            {
                Console.WriteLine(i);
```

```
                i--;
            }
        }
    }
}
```

5. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
    class Program
    {
        static void Main(string[] args)
        {
            // cyklus do-while, který bude vypisovat i
            // dokud i bude větší než 4

            int i = 10;
            do
            {
                Console.WriteLine(i);
                i--;
            } while (i >= 4);
        }
    }
}
```

6. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
    class Program
    {
        static void Main(string[] args)
        {
            // vypište pomocí for cyklu slovo "Ahoj" petkrát

            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("Ahoj");
            }
        }
    }
}
```

7. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
    class Program
    {
        static void Main(string[] args)
        {
            // cyklus, který se zastaví ve chvíli,
            // kdy i bude rovno číslu 5

            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine(i);
                if (i == 5)
                    break;
            }
        }
    }
}
```

8. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
    class Program
    {
        static void Main(string[] args)
        {
            // cyklus, který vypíše sestupnou posloupnost čísel od 10 do 1

            for (int i = 10; i >= 1; i--)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

9. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace cykly
{
```

```
    class Program
    {
        static void Main(string[] args)
        {
            //cyklus, který vypíše každé sudé číslo od 0 do 100

            for (int i = 0; i <= 100; i++)
            {
                if (i % 2 == 0)
                    Console.WriteLine(i);
            }
        }
    }
}
```

Umístění řešení: Vybrané ukázkové řešení je zpracováno v datové příloze ve složce prilohy/cv04_cykly. Dílčí kódy jsou pak uloženy jako:

cv04_cykly_01.cs

cv04_cykly_02.cs

cv04_cykly_03.cs

cv04_cykly_04.cs

cv04_cykly_05.cs

cv04_cykly_06.cs

cv04_cykly_07.cs

cv04_cykly_08.cs

cv04_cykly_09.cs

4.6 Pole

Cíle vyučovací hodiny: Žáci se seznámí s poli, jejich deklarací, práci s prvky v poli a cyklem *foreach* a dokáže je správně využít.

Obsah vyučovací hodiny: Na začátku hodiny je opakování na téma cykly. Dále se žáci seznámí s novým tématem polí, které jsou vysvětleny na konkrétních příkladech.

Vyhotovení: Žáci si na začátku prezentace zopakují téma cykly a zodpoví na otázky, pak se seznámí se zbytkem obsahu prezentace, na konci zodpoví otázky spolu s učitelem a na příkladech pracují samostatně.

Možné výstupy:

1. **Příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pole
{
    class Program
    {
        static void Main(string[] args)
        {
            // opakování
            // cyklus, který bude vypisovat všechna
            // čísla od 1 do 10 kromě čísel 4 a 8

            for (int i = 1; i <= 10; i++)
            {
                if (i == 4 || i == 8)
                {
                    continue;
                }

                Console.WriteLine(i);
            }
        }
    }
}
```

2. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pole
{
    class Program
    {
        static void Main(string[] args)
        {
            //opakovani
            // cyklus, který bude vypisovat i, dokud i
            // nebude rovno číslu 20

            int j = 0;
            while (j <= 20)
            {
                Console.WriteLine(j);
                j++;
            }
        }
    }
}
```

3. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pole
{
    class Program
    {
        static void Main(string[] args)
        {
            //opakovani
            // cyklus, který bude vypisovat každé liché
            // číslo od 100 do 0 (sestupně)

            for (int i = 100; i >= 0; i--)
            {
                if (i % 2 == 1)
                    Console.WriteLine(i);
            }
        }
    }
}
```

4. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pole
{
    class Program
    {
        static void Main(string[] args)
        {
            //opakovani
            // cyklus, který bude vypisovat každý násobek
            // čísla 6 od 0 do 100

            for (int i = 0; i <= 100; i++)
            {
                if (i % 6 == 0 && i != 0)
                    Console.WriteLine(i);
            }
        }
    }
}
```

5. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace pole
{
    class Program
    {
        static void Main(string[] args)
        {
            // procvicovani
            // 1.
            int[] poleCisel = new int[8] { 2, 5, 8, 0, 3, 4, 1, 6 };

            // 2.
            Console.WriteLine(poleCisel[1]);
            Console.WriteLine(poleCisel[3]);

            // 3.
            int[] druhePole = poleCisel;
            Array.Reverse(druhePole);

            // 4.
            druhePole[2] = 10;

            // 5.
            double prumer = druhePole.Average();
            Console.WriteLine(prumer);

            // 6.
            int delka = poleCisel.Length;
            if (delka % 2 == 0)
            {
                Console.WriteLine("Pole má sudou délku.");
            }
            else
            {
                Console.WriteLine("Pole má lichou délku.");
            }
        }
    }
}
```

Umístění řešení: Vybrané ukázkové řešení je zpracováno v datové příloze ve složce prilohy/cv05_pole. Dílčí kódy jsou pak uloženy jako:

cv05_pole_01.cs

cv05_pole_02.cs

cv05_pole_03.cs

cv05_pole_04.cs

cv05_pole_05.cs

4.7 Seznamy

Cíle vyučovací hodiny: Žáci se seznámí se seznamy, deklarací seznamů a metodami pro práci se seznamy a dokáže je správně použít.

Obsah vyučovací hodiny: Na začátku hodiny je opakování na téma pole. Dále se žáci seznámí s novým tématem seznamů, které jsou vysvětleny na konkrétních příkladech.

Vyhotovení: Žáci si na začátku prezentace zopakují téma pole a zodpoví na otázky, pak se seznámí se zbytkem obsahu prezentace, na konci zodpoví otázky spolu s učitelem a na příkladech pracují samostatně.

Možné výstupy:

1. **Příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace seznamy
{
    class Program
    {
        static void Main(string[] args)
        {

            //opakování pole
            // 1.
            int[] pole1 = new int[5] { 4, 7, 9, 1, 0 };
            int[] pole2 = new int[5] { 3, 8, 1, 2, 9 };

            // 2.
            int prvky1 = pole1.Sum();
            int prvky2 = pole2.Sum();

            if (prvky1 > prvky2)
            {
                Console.WriteLine("První pole má vyšší součet");
            }
            else
            {
                Console.WriteLine("Druhé pole má vyšší součet");
            }

            // 3.
            int soucet = prvky1 + prvky2;

            //4.
            for (int i = 0; i < pole1.Length; i++)
            {
                int test = pole1[i] % 2;
                if (test == 1)
                {
```

```
                    pole1[i] = pole1[i] + 1;
                    Console.WriteLine(pole1[i]);
                }
            }
        }
    }
}
```

2. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace seznamy
{
    class Program
    {
        static void Main(string[] args)
        {
            // procvicovani
            List<int> seznam = new List<int>();

            for (int i = 1; i <= 10; i++)
            {
                seznam.Add(i);
            }

            seznam[1] = 15;
            seznam.Insert(seznam.Count, 15);

            foreach (var a in seznam)
            {
                Console.WriteLine(a);
            }
        }
    }
}
```

Umístění řešení: Vybrané ukázkové řešení je zpracováno v datové příloze ve složce prilohy/cv06_seznamy. Dílčí kódy jsou pak uloženy jako:

cv06_seznamy_01.cs

cv06_seznamy_02.cs

4.8 Struktury

Cíle vyučovací hodiny: Žáci se seznámí se strukturami, členskými prvky struktur, konstruktorem a metodou struktury a také deklarací a inicializací struktury a dokáže je správně použít.

Obsah vyučovací hodiny: Na začátku hodiny je opakování na téma seznamy. Dále se žáci seznámí s novým tématem struktur, které jsou vysvětleny na konkrétních příkladech.

Vyhotovení: Žáci si na začátku prezentace zopakují téma seznamy a zodpoví na otázky, pak se seznámí se zbytkem obsahu prezentace, na konci zodpoví otázky společně s učitelem a na příkladech pracují samostatně.

Možné výstupy:

1. **Příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace struktury
{
    class Program
    {
        static void Main(string[] args)
        {
            //opakovani seznamy
            List<int> seznam = new List<int>();
            Random random = new Random();

            for (int i = 1; i <= 10; i++)
            {
                seznam.Add(random.Next(1, 100));
            }

            seznam.Sort();
            seznam.RemoveRange(2, 4);
            seznam.Reverse();

            foreach (var a in seznam)
            {
                Console.WriteLine(a);
            }

        }
    }
}
```

2. **Příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace struktury
{
    class Program
    {
```

```
        struct Trojuhelnik
        {
            private readonly double _a;
            private readonly double _b;
            private readonly double _c;

            public Trojuhelnik(double a, double b, double c)
            {
                _a = a;
                _b = b;
                _c = c;
            }

            public void ObvodTrojuhelnik()
            {
                Console.WriteLine(_a + _b + _c);
            }
        }
        static void Main(string[] args)
        {

       Trojuhelnik trojuhelnik = new Trojuhelnik(3, 4, 5);
        trojuhelnik.ObvodTrojuhelnik();
        }
    }
}
```

3. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace struktury
{
    class Program
    {
        struct Obdelnik
        {
            private readonly double _a;
            private readonly double _b;

            public Obdelnik(double a, double b)
            {
                _a = a;
                _b = b;
            }

            public void ObsahObdelniku()
            {
                Console.WriteLine(_a * _b);
            }
        }


        static void Main(string[] args)
        {

            Obdelnik obdelnik = new Obdelnik(2, 8);
```

```
            obdelnik.ObsahObdelniku();
        }
    }
}
```

4. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace struktury
{
    class Program
    {
        struct Krychle
        {
            private readonly double _a;

            public Krychle(double a)
            {
                _a = a;
            }

            public void PovrchKrychle()
            {
                Console.WriteLine(6 * Math.Pow(_a, 2));
            }
        }

        static void Main(string[] args)
        {
            Krychle krychle = new Krychle(6);
            krychle.PovrchKrychle();
        }
    }
}
```

Umístění řešení: Vybrané ukázkové řešení je zpracováno v datové příloze ve složce prilohy/cv07_struktury. Dílčí kódy jsou pak uloženy jako:

cv07_struktury_01.cs

cv07_struktury_02.cs

cv07_struktury_03.cs

cv07_struktury_04.cs

4.9 Třídy

Cíle vyučovací hodiny: Žáci se seznámí s třídami a jejich deklarací a také s modifikátory přístupu, fieldy, konstruktory, property a metodami tříd.

Obsah vyučovací hodiny: Na začátku hodiny je opakování na téma strukury. Dále se žáci seznámí s novým tématem tříd, které jsou vysvětleny na konkrétních příkladech.

Vyhotovení: Žáci si na začátku prezentace zopakují téma struktury a zodpoví na otázky, pak se seznámí se zbytkem obsahu prezentace, na konci zodpoví otázky a na příkladech pracují samostatně.

Možné výstupy:

1. **Příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tridy
{
    struct Osoba
    {
        public string Jmeno;
        public string Prijmeni;

        public void VypsaniOsoby()
        {
            Console.WriteLine($"{Jmeno} {Prijmeni}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Osoba osoba;
            osoba.Jmeno = "Lukas";
            osoba.Prijmeni = "Novy";
            osoba.VypsaniOsoby();
        }
    }
}
```

2. **Příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tridy
{
    struct Prevod
    {
        public double Stopa;

        public void StopaDoMetru()
```

```
        {
            double metry = Stopa / 3.281d;

            Console.WriteLine($"{Stopa} stopa je {metry} metru");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Prevod prevod;
            prevod.Stopa = 3.0d;
            prevod.StopaDoMetru();
        }
    }
}
```

3. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tridy
{
    struct Kniha
    {
        public string Nazev;
        public int RokVydani;

        public void InfoKniha()
        {
            Console.WriteLine($"{Nazev} vysla v roce {RokVydani}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Kniha kniha;
            kniha.Nazev = "1984";
            kniha.RokVydani = 1949;
            kniha.InfoKniha();
        }
    }
}
```

4. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tridy
```

```
{
    class Osoba
    {
        public string Jmeno { get; set; }
        public DateTime DatumNarozeni { get; set; }

        public Osoba(string jmeno, DateTime datumNarozeni)
        {
            Jmeno = jmeno;
            DatumNarozeni = datumNarozeni;
        }

        public int VekOsoby()
        {
            return (int)((DateTime.Now - DatumNarozeni).TotalDays / 365);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // trida osoba
            Osoba osoba = new Osoba("Lukas", new DateTime(1999, 3, 1));
            Console.WriteLine(osoba.VekOsoby());
        }
    }
}
```

5. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tridy
{
    class Student
    {
        public string Jmeno { get; set; }
        public int Body { get; set; }

        public Student(string jmeno, int body)
        {
            Jmeno = jmeno;
            Body = body;
        }

        public void HodnoceniZnamky()
        {
            if (Body < 50)
            {
                Console.WriteLine($"Hodnoceni pro {Jmeno} je F ({Body} bodu)");
            }
            else if (Body >= 50 && Body <= 59)
            {
                Console.WriteLine($"Hodnoceni pro {Jmeno} je E ({Body} bodu)");
            }
            else if (Body >= 60 && Body < 69)
            {
```

```
                Console.WriteLine($"Hodnoceni pro {Jmeno} je D ({Body} bodu)");
            }
            else if (Body >= 70 && Body < 79)
            {
                Console.WriteLine($"Hodnoceni pro {Jmeno} je C ({Body} bodu)");
            }
            else if (Body >= 80 && Body < 89)
            {
                Console.WriteLine($"Hodnoceni pro {Jmeno} je B ({Body} bodu)");
            }
            else if (Body >= 90 && Body <= 100)
            {
                Console.WriteLine($"Hodnoceni pro {Jmeno} je A ({Body} bodu)");
            }
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            string[] jmena = { "Martin", "Lukas", "Daniel", "Isak", "Rostislav" };
            Random rnd = new Random();
            List<Student> students = new List<Student>();

            for (int i = 0; i < jmena.Length; i++)
            {
                students.Add(new Student(jmena[i], rnd.Next(1, 100)));
            }

            foreach (var student in students)
            {
                student.HodnoceniZnamky();
            }
        }
    }
}
```

Umístění řešení: Vybrané ukázkové řešení je zpracováno v datové příloze ve složce prilohy/cv08_tridy. Dílčí kódy jsou pak uloženy jako:

cv08_tridy_01.cs

cv08_tridy_02.cs

cv08_tridy_03.cs

cv08_tridy_04.cs

cv08_tridy_05.cs

4.10 Metody

Cíle vyučovací hodiny: Žáci se seznámí s tvorbou vlastních metod, jejich voláním a vstupními parametry a návratovými hodnotami.

Obsah vyučovací hodiny: Na začátku hodiny je opakování na téma třídy. Dále se žáci seznámí s novým tématem vlastních metod, které jsou vysvětleny na konkrétních příkladech.

Vyhotovení: Žáci si na začátku prezentace zopakují téma třídy a zodpoví na otázky, pak se seznámí se zbytkem obsahu prezentace, na konci zodpoví otázky a na příkladech pracují samostatně.

Možné výstupy:

1. **Příklad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace metody
{
    class Zavodnik
    {
        public string Jmeno { get; set; }
        public int Cas { get; set; }

        public Zavodnik(string jmeno, int cas)
        {
            Jmeno = jmeno;
            Cas = cas;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            string[] jmena = { "Martin", "Lukas", "Daniel", "Isak", "Rostislav" };
            List<Zavodnik> zavodniky = new List<Zavodnik>();

            Random rnd = new Random();
            for (int i = 0; i < jmena.Length; i++)
            {
                zavodniky.Add(new Zavodnik(jmena[i], rnd.Next(1, 100)));
            }

            int aktualniCas = 0;
            int nejCas = 0;
            string nejZavodnik = "";
            foreach (var zavodnik in zavodniky)
            {
                Console.WriteLine($"Zavodnik {zavodnik.Jmeno} s casem
{zavodnik.Cas}");
                aktualniCas = zavodnik.Cas;

                if (nejCas == 0)
                    nejCas = zavodnik.Cas;

                if (aktualniCas < nejCas)
                {
                    nejCas = aktualniCas;
```

```
                    nejZavodnik = zavodnik.Jmeno;
                }
            }

            Console.WriteLine($"Nejrychlejsi cas ma zavodnik {nejZavodnik} s casem
{nejCas}");
        }
    }
}
```

2. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace metody
{
    class Zvirata
    {
        public string Druh { get; set; }
        public string Vlasnost { get; set; }
        public string Ucel { get; set; }

        public Zvirata(string druh, string vlasnost, string ucel)
        {
            Druh = druh;
            Vlasnost = vlasnost;
            Ucel = ucel;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            string[] druh = { "kocka", "kun", "krava", "slepice" };
            string[] vlasnost = { "selma", "kopytnik", "kopytnik", "ptak" };
            string[] ucel = { "domaci mazlicek", "rekreace", "mleko", "vejce" };
            List<Zvirata> zvirata = new List<Zvirata>();
            for (int i = 0; i < druh.Length; i++)
            {
                zvirata.Add(new Zvirata(druh[i], vlasnost[i], ucel[i]));
            }

            foreach (var zvire in zvirata)
            {
                Console.WriteLine($"Zvire {zvire.Druh} ma vlasnost {zvire.Vlasnost}
za ucelem {zvire.Ucel}");
            }
        }
    }
}
```

3. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace metody
{
    class Program
    {
        public static void PorovnejCisla(int a, int b)
        {
            if (a > b)
            {
                Console.WriteLine($"Cislo {a} je vyssi nez cislo {b}");
            }
            else
            {
                Console.WriteLine($"Cislo {b} je vyssi nez cislo {a}");
            }
        }
        static void Main(string[] args)
        {
            PorovnejCisla(4, 2);
        }
    }
}
```

4. Příklad

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace metody
{
    class Program
    {
        public static void VraceniNasobku(int min, int max, int nasobek)
        {
            for (int i = min; i <= max; i++)
            {

                if (i % nasobek == 0)
                {
                    Console.WriteLine(i);
                }
            }
        }

        static void Main(string[] args)
        {
            VraceniNasobku(1, 100, 6);
        }
    }
}
```

5. Příklad

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace metody
{
        public static void DelkaReteze(string retez)
        {
            Console.WriteLine(retez.Length);
        }

        static void Main(string[] args)
        {
            DelkaReteze("konec");
        }
    }
}
```

Umístění řešení: Vybrané ukázkové řešení je zpracováno v datové příloze ve složce prilohy/cv09_metody. Dílčí kódy jsou pak uloženy jako:

cv09_metody_01.cs

cv09_metody_02.cs

cv09_metody_03.cs

cv09_metody_04.cs

cv09_metody_05.cs

ZÁVĚR

V teoretické části tato bakalářská práce seznámila čtenáře s definicí a zápisem algoritmu. Objasnila, jaké prvky nabízí vývojové prostředí Microsoft Visual Studio. Následně popsala programovací jazyk C# včetně jeho syntaxe a sémantiky.

V praktické části bylo vytvořeno několik výukových materiálů ve formě prezentací a zdrojových kódů. Výukové materiály odpovídají požadavkům pro výuku Informatického semináře na středních školách.

Hlavním cílem bakalářské práce bylo vytvořit výukové materiály pro hodiny programování v Informatickém semináři na středních školách v jazyce C#. Obsahem výukových materiálů jsou logicky uspořádané prezentace, které obsahují jednotlivě popsanou probíranou látku. Látka je vysvětlena na konkrétních ukázkách a prezentace jsou doplněny o opakování a procvičovací příklady. Řešení příkladů je také kromě prezentací přiloženo v souborech se zdrojovými kódy.

SEZNAM POUŽITÉ LITERATURY

[1] BOURAS, Aristides S. *C# and Algorithmic Thinking for the Complete Beginner: Learn to Think Like a Programmer*. 2nd ed. USA: CreateSpace Independent Publishing Platform, 2016. ISBN 978-1671594364.

[2] JANDOVÁ, Radoslava. *Algoritmus a jeho vlastnosti* [online]. In: . [cit. 2021-02-18]. Dostupné z: http://www.vrstevnice.com/akce/grandaction/vskola/2semestr/ted/yd14ted_clanek_jandora1.pdf

[3] KRÁTKÝ, Michal a Jiří DVORSKÝ. *Úvod do programování* [online]. In: . 2004 [cit. 2021-02-18]. Dostupné z: http://www.cs.vsb.cz/kratky/courses/200405/udp/presentation/udp-1_6.pdf

[4] *Algoritmus a jeho vlastnosti* [online]. In: . [cit. 2021-02-18]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7316

[5] FOX, Pamela. *Expressing an algorithm* [online]. [cit. 2021-02-23]. Dostupné z: https://www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/building-algorithms/a/expressing-an-algorithm

[6] *What Is a Flowchart* [online]. [cit. 2021-02-24]. Dostupné z: https://www.lucidchart.com/pages/what-is-a-flowchart-tutorial/#section_1

[7] *An Introduction to Flowcharts* [online]. GeeksforGeeks, 2020 [cit. 2021-03-01]. Dostupné z: https://www.geeksforgeeks.org/an-introduction-to-flowcharts/

[8] *Syntax and Semantics* [online]. In: . [cit. 2021-03-07]. Dostupné z: https://www.codeit-project.eu/syntax-and-semantics/?chapter1

[9] How to Write a Pseudo Code? *GeeksforGeeks* [online]. 2021 [cit. 2021-03-07]. Dostupné z: https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/

[10] *Visual Studio 2017 Product Family System Requirements: Visual Studio 2017 System Requirements* [online]. Microsoft, 2018 [cit. 2021-03-13]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/productinfo/vs2017-system-requirements-vs#visual-studio-2017-system-requirements

[11] *Visual Studio build numbers and release dates* [online]. Microsoft, 2021 [cit. 2021-03-13]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/install/visual-studio-build-numbers-and-release-dates?view=vs-2019

[12] *Microsoft Visual Studio 2015 Language Pack* [online]. Microsoft [cit. 2021-03-13]. Dostupné z: https://www.microsoft.com/en-us/download/details.aspx?id=48157

[13] *C99 library support in Visual Studio 2013* [online]. Microsoft, 2013 [cit. 2021-03-13]. Dostupné z:http://blogs.msdn.com/b/vcblog/archive/2013/07/19/c99-library-support-in-visual-studio-2013.aspx

[14] *BEST PYTHON IDE FOR PYTHON PROGRAMMING* [online]. PythonicQuest, 2017 [cit. 2021-03-13]. Dostupné z: https://web.archive.org/web/20170118054009/http://www.pythonicquest.com/article/best-python-ide/]

[15] *Soubory ke stažení: Visual Studio 2019* [online]. Microsoft [cit. 2021-03-13]. Dostupné z: https://visualstudio.microsoft.com/cs/downloads/

[16] *Install Visual Studio* [online]. Microsoft, 2019 [cit. 2021-03-13]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019

[17] *Code snippets* [online]. Microsoft, 2016 [cit. 2021-03-13]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/ide/code-snippets?view=vs-2019

[18] *IntelliSense in Visual Studio* [online]. Microsoft, 2018 [cit. 2021-03-13]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/ide/code-snippets?view=vs-2019

[19] *Learn to use the code editor: Refactor a name* [online]. Microsoft, 2017 [cit. 2021-03-13]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/get-started/tutorial-editor?view=vs-2019#create-a-new-code-file

[20] *How to debug for absolute beginners* [online]. Microsoft, 2020 [cit. 2021-03-13]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/debugger/debugging-absolute-beginners?view=vs-2019

[21] PŠENČÍKOVÁ, Jana. *Algoritmizace*. 2. vydání. Computer Media, 2009. ISBN 978-80-7402-034-6.

[22] DUDDEL, Dave a Sylvia LANGFIELD. *Cambridge International AS and A Level Computer Science Coursebook*. Cambridge, 2016. ISBN 9781107546738.

[23] C# - Program Structure. *Tutorialspoint* [online]. 2021 [cit. 2021-04-07]. Dostupné z: https://www.tutorialspoint.com/csharp/csharp_program_structure.htm

[24] C# compilation process: Compilers, Runtime, and the .NET Framework. In: *Codeeasy.net* [online]. [cit. 2021-04-07]. Dostupné z: https://codeasy.net/lesson/c_sharp_compilation_process

[25] Debugging: Introduction to debugging. In: *C# Tutorial* [online]. [cit. 2021-04-07]. Dostupné z: https://csharp.net-tutorials.com/debugging/introduction/

[26] First look at profiling tools. In: *Visual Studio Docs* [online]. Microsoft, 2021 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/profiling/profiling-feature-tour?view=vs-2019

[27] Profiling Overview. In: *Visual Studio Docs* [online]. Microsoft, 2017 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/unmanaged-api/profiling/profiling-overview

[28] SZABÓ, Dániel. Preprocessor Directives in C#: The Preprocessor. In: *PLURALSIGHT* [online]. 2020 [cit. 2021-04-07]. Dostupné z: https://www.pluralsight.com/guides/preprocessor-directives-in-c

[29] C# - Data Types: Value Type. In: *Tutorialspoint* [online]. [cit. 2021-04-07]. Dostupné z: https://www.tutorialspoint.com/csharp/csharp_data_types.htm

[30] Types, variables, and values. In: *Visual Studio Docs* [online]. Microsoft, 2016 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/basic-types

[31] *C# Primitive Datatypes* [online]. In: . [cit. 2021-04-07]. Dostupné z: https://condor.depaul.edu/sjost/nwdp/notes/cs1/CSDatatypes.htm

[32] C# operators and expressions (C# reference). In: *Visual Studio Docs* [online]. Microsoft, 2020 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/

[33] Arithmetic operators (C# reference). In: *Visual Studio Docs* [online]. Microsoft, 2020 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators

[34] RAJU, Giri. Global and Local Variables in C#. In: *Tutorialspoint* [online]. 2018 [cit. 2021-04-07]. Dostupné z: https://www.tutorialspoint.com/global-and-local-variables-in-chash

[35] C# - Variables. In: *Tutorialspoint* [online]. [cit. 2021-04-07]. Dostupné z: https://www.tutorialspoint.com/csharp/csharp_variables.htm

[36] Statements (C# Programming Guide): Types of statements. In: *Visual Studio Docs* [online]. Microsoft, 2015 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/statements

[37] If-else (C# Reference). In: *Visual Studio Docs* [online]. Microsoft, 2015 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/if-else

[38] Switch (C# reference). In: *Visual Studio Docs* [online]. Microsoft, 2019 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/switch

[39] For (C# reference). In: *Visual Studio Docs* [online]. Microsoft, 2018 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/for

[40] ČÁPKA, David. Lekce 7 - Cykly v C# .NET - for a while: for cyklus. In: *IT network* [online]. [cit. 2021-04-07]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-cykly-for-while

[41] While (C# Reference). In: *Visual Studio Docs* [online]. Microsoft, 2018 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/while

[42] Do (C# Reference). In: *Visual Studio Docs* [online]. Microsoft, 2018 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/do

[43] PAVLÁTKA, Zdeněk. Lekce 7 - Cykly v C++ (while, do while): do-while cyklus. In: *IT network* [online]. [cit. 2021-04-07]. Dostupné z: https://www.itnetwork.cz/cplusplus/zaklady/c-plus-plus-tutorial-cykly-while-do-while

[44] C# | foreach Loop. In: *GeeksforGeeks* [online]. 2019 [cit. 2021-04-07]. Dostupné z: https://www.geeksforgeeks.org/c-sharp-foreach-loop/

[46] ČÁPKA, David. Lekce 9 - Pole v C# .NET. In: *IT network* [online]. [cit. 2021-04-07]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-pole

[47] Arrays (C# Programming Guide). In: *Visual Studio Docs* [online]. Microsoft, 2021 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/

[48] ČÁPKA, David. Lekce 2 - Seznam (List) pomocí pole v C#. In: *IT network* [online]. [cit. 2021-04-07]. Dostupné z: https://www.itnetwork.cz/csharp/kolekce-a-linq/c-sharp-tutorial-seznamy-kolekce-list

[49] List<T> Class. In: *Visual Studio Docs* [online]. Microsoft [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-5.0

[50] Classes and structs (C# programming guide). In: *Visual Studio Docs* [online]. Microsoft, 2021 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/

[51] C# - Struct. In: *TutorialsTeacher* [online]. 2020 [cit. 2021-04-07]. Dostupné z: https://www.tutorialsteacher.com/csharp/csharp-struct

[52] Structure types (C# reference). In: *Visual Studio Docs* [online]. Microsoft, 2020 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct

[53] C# Class. In: *TutorialsTeacher* [online]. [cit. 2021-04-07]. Dostupné z: https://www.tutorialsteacher.com/csharp/csharp-class

[54] Classes and structs (C# programming guide). In: *Visual Studio Docs* [online]. Microsoft, 2021 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/

[55] The Basics: Functions. In: *C# Tutorial* [online]. 2021 [cit. 2021-04-07]. Dostupné z: https://csharp.net-tutorials.com/basics/functions/

[56] Local functions (C# Programming Guide). In: *Visual Studio Docs* [online]. Microsoft, 2020 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/local-functions

[57] Methods (C# Programming Guide). In: *Visual Studio Docs* [online]. Microsoft, 2021 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods

[58] ČÁPKA, David. Lekce 1 - Úvod do objektově orientovaného programování v C#: Objektově orientovaný přístup. In: *IT network* [online]. [cit. 2021-04-07]. Dostupné z: https://www.itnetwork.cz/csharp/oop/c-sharp-tutorial-uvod-do-objektove-orientovaneho-programovani

[59] PECINOVSKÝ, Rudolf. *OOP – Learn Object Oriented Thinking and Programming*. Academic Series, 2013. ISBN 978-80-904661-9-7.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

C# Programovací jazyk C-sharp

API Application Programming Interface

HTML HyperText Markup Language

CSS Cascading Style Sheets

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD-ROM

V příloze se nachází složka prezentace, která obsahuje jednotlivé prezentace k vyučovacím hodinám.

Dále příloha obsahuje několik složek určeným k prezentacím 03 – 09, v nichž jsou uloženy zdrojové kódy s hotovými příklady z prezentací.

Struktura složky *prilohy*: