Učební text a sbírka úloh pro výuku základů programování v C# na střední škole navazující na znalost jazyka Scratch

Bc. Martin Buriánek

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Martin Buriánek**
Osobní číslo:	**A21132**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Učitelství informatiky pro střední školy**
Forma studia:	**Prezenční**
Téma práce:	**Učební text a sbírka úloh pro výuku základů programování v C# na střední škole navazující na znalost jazyka Scratch**
Téma práce anglicky:	**A Textbook and a Set of Exercises for Teaching Basics of C# Programming in High School Following Knowledge of Scratch**

Zásady pro vypracování

1. Vytvořte literární rešerši na téma Metody výuky programovacích jazyků.
2. Popište stručně historii a specifika programovacích jazyků Scratch a C#.
3. V učebním textu vysvětlete základní teoretické znalosti. Předpokládejte, že čtenář textu umí programovat v jazyce Scratch a tyto znalosti využívá při učení se jazyka C#.
4. Porovnejte řešení několika různých úloh v jazyce Scratch a v jazyce C#. Uveďte vzájemné podobnosti a analyzujte odlišnosti v přístupu k řešení úloh v obou jazycích.
5. Sestavte sbírku úloh s příklady na procvičení získaných poznatků při tvorbě jednoduchých praktických aplikací.
6. Vytvořte pracovní listy využívající programování v C# použitelné při výuce v matematice.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. BORY, Pavel. C# bez předchozích znalostí. 2. vydání. V Brně: Computer Press, 2022. ISBN 978-80-251-5061-0.
2. VYSTAVĚL, Radek. Moderní programování: učebnice pro začátečníky. 5. vydání. Ondřejov: moderníProgramování, 2019. ISBN 978-80-903951-9-0.
3. HYLMAR, Radek. Programování pro úplné začátečníky. 2. vydání. V Brně: Computer Press, 2022. ISBN 978-80-251--5059-7.
4. VIRIUS, Miroslav. Programování v C#: od základů k profesionálnímu použití. Praha: Grada Publishing, 2021. Knihovna programátora. ISBN 978-80-271-1216-6.
5. KNUTH, Donald Ervin. Umění programování. 1. díl, Základní algoritmy. Brno: Computer Press, 2008. ISBN 978-80--251-2025-5.
6. POLÁK, Josef. Přehled středoškolské matematiky. 10. vydání. Praha: Prometheus, 2015. ISBN 978-80-7196-458-2.
7. HOROVÁ, Ivana a Jiří ZELINKA. Numerické metody. 2., rozš. vyd. Brno: Masarykova univerzita v Brně, 2004. ISBN 80--210-3317-7.

Vedoucí diplomové práce: **Ing. Tomáš Sysala, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

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

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové 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 diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou 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 – diplomovou 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 diplomové 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 diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové 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 diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 10. 5. 2024

Martin Buriánek, v. r.
podpis studenta

ABSTRAKT

Hlavním cílem mé diplomové práce je vytvoření učebního textu a sbírky úloh zaměřených na základy programovacího jazyka C#. Tyto materiály jsou určené pro střední školy a vycházejí ze znalosti blokového programovacího jazyka. V teoretické části je popsána současná výuka programování na základních a středních školách, je zde uveden přehled vhodných materiálů pro studium jazyka C# a také metodika k vytvořeným pracovním listům. Obsahem praktické části je vytvořený učební text a sbírka úloh. Součástí praktické části je také příloha – pracovní listy propojující vybraná témata z matematiky s programováním v C#.

Klíčová slova: Scratch, C#, výuka programování, učební text, sbírka úloh, pracovní listy

ABSTRACT

The main goal of my thesis is to create a textbook and a set of exercises focused on basics of programming in C#. These materials are supposed for students of secondary school and are based on knowledge of block programming language. The theoretical part describes current teaching of programming in primary and secondary schools, shows overview of available suitable source for studying programming in C# and methods to created worksheets. The practical part includes a textbook, a set of exercises and worksheets connecting chosen topics from math with programming in C#.

Keywords: Scratch, C#, teaching programming, textbook, set of exercise, worksheets

Rád bych na tomto místě poděkoval panu Ing. Tomáši Sysalovi, Ph.D., za vedení mé práce a za užitečné rady.

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

OBSAH

ÚVOD

Výuka informatiky prochází v současnosti změnami, které reagují na rychlý rozvoj digitálních technologií a jejich začlenění do soudobého každodenního života. Revidovaný rámcový vzdělávací program zavádí výuku algoritmizace a programování už na základních školách. Ve většině případů se tam žáci setkají s určitou formou vizuálního (blokového) programovacího jazyka.

Cílem mé práce je vytvořit učební text a sbírku úloh, které naleznou uplatnění na středních odborných školách v oboru Informační technologie při výuce programování v jazyce C#. Pro jazyk C# jsem se rozhodl z více důvodů. Jedná se o moderní objektově orientovaný programovací jazyk, po kterém je na trhu poměrně velká poptávka. Základy jazyka C# jsem se sám naučil v maturitním semináři na střední škole a od té doby v něm rád programuji. A v budoucnu bych se na výuku tohoto programovacího jazyka rád zaměřil.

Existuje několik kvalitních zdrojů, které se programování v C# věnují, ale žádný z nich nesplňuje současně tři podmínky pro to, aby byl vhodný k použití ve školní výuce: je volně dostupný, nejlépe v elektronické verzi, je v češtině a obsahuje dostatek příkladů na procvičení. Cílem mé práce je vytvořit text, který tyto podmínky splňovat bude.

V učebním textu budu předpokládat, že žáci přicházejí na střední školu se základními znalostmi algoritmizace a programování v blokovém jazyce už ze základní školy a že tyto znalosti vhodně využijí při učení se jazyka C#.

Teoretická část práce je rozdělena do tří částí. V první kapitole uvádím metody výuky programování a popisuji proměnu výuky informatiky v současné době, která vychází z revize rámcových vzdělávacích programů. Druhá kapitola je recenzí dostupných zdrojů zaměřených na programování v C#. Třetí kapitola je metodikou k pracovním listům vytvořeným v praktické části.

Praktická část je vypracována jako učební text a sbírka úloh. Je členěna do 11 kapitol, které jsou záměrně číslované od 1 – pokud někdo bude chtít použít tyto výukové materiály, může praktickou část práce z dokumentu jednoduše „vystřihnout“ pomocí některého volně dostupného nástroje pro úpravu PDF. Součástí praktické části práce je také příloha – pracovní listy, které propojují vybraná témata z matematiky s programováním v C# a vytvářejí tak mezipředmětové vztahy.

I. TEORETICKÁ ČÁST

1 VÝUKA PROGRAMOVÁNÍ

1.1 Metody výuky

Pojmem metody výuky označujeme způsoby dosažení výchovně vzdělávacího cíle. Tento pojem vymezuje také způsob, jak učitel vyučuje a jak se žák učí, a způsoby uspořádání činnosti učitele i žáků, které směřují k vytyčeným cílům. [1]

Pojem výuka označuje společnou interakci učitele a žáka, zatímco vyučování je činnost vykonávaná učitelem a učení činnost vykonávaná žákem při této interakci. [2]

Účinná didaktická metoda by měla splňovat několik kritérií: [1]

- informační přesnost (předává informace a dovednosti v nezkreslené podobě),
- informační účinnost (rozvíjí poznávací procesy),
- racionální a emotivní působivost (vede žáka k aktivitě, učení a poznávání),
- respektování systému vědy a poznání (používá ověřené údaje),
- výchovnost (rozvíjí morální, sociální, pracovní a estetický profil žáka),
- přirozenost v průběhu i v důsledcích,
- použitelnost v praxi (přibližuje školu reálnému životu),
- adekvátnost žákům i učiteli,
- didaktická ekonomičnost (umožňuje stihnout probrat potřebný rozsah učiva),
- hygieničnost.

Metod výuky existuje velké množství. Metody a jejich klasifikaci rozsáhle popisují například zdroje [1] a [2]. Volba výukových metod by neměla být stereotypní – kvalitní a efektivní výuky dosáhneme pouze jejich promyšlenou kombinací.

V seznamu níže uvádím vybrané [1] výukové metody a možnost jejich použití při výuce programování:

- Vysvětlování (výklad) – Jedná se o monologickou metodu předpokládající zvýšenou pozornost ze strany žáků. Je vhodná pro úvod do nového tématu nebo k seznámení žáků s definicemi pojmů. Výklad by neměl být příliš dlouhý a měla by po něm následovat činnost, která bude žáky aktivizovat.
- Řízená diskuse – Jde o dialogickou metodu, při jejímž použití žáci a učitel vzájemně komunikují. Tato metoda je vhodná například při rozboru řešeného příkladu k porovnání různých přístupů navržených žáky.

- Instruktáž – Tato monologická metoda je teoretickým úvodem před praktickou činností. Je vhodná například k seznámení žáků s algoritmem (tj. s jeho kroky), který mají naprogramovat.
- Práce s textem (odbornými zdroji) – Upevňuje schopnost žáků vyhledávat informace a pracovat s nimi. V programování ji považuji za velmi důležitou, protože vyhledávání (např. hledání příkazu pro určitou operaci v dokumentaci programovacího jazyka) je častou činností programátora.
- Samostatná práce – Tato metoda je důležitá, protože umožňuje žákům procvičit si učivo. Každý žák může postupovat vlastním tempem a obtížnost úloh může být diferencována pro různé žáky. Metoda je vhodná k procvičování dovednosti praktické tvorby programu.
- Skupinová práce – Žáci se při ní učí vzájemně spolupracovat. Tato metoda je vhodná ke tvorbě skupinového projektu. V případě výuky programování její zařazení vnímám jako velmi důležité, protože většina programátorů pracuje v týmech.
- Badatelsky orientovaná výuka – Žáci při ní samostatně objevují určité poznatky. I v tomto případě se jedná o metodu s velkým potenciálem při výuce programování. Využití badatelského přístupu také částečně předpokládají pracovní listy, které jsem vytvořil jako přílohu této práce.
- Názorně demonstrační metody – Při výuce programování je lze využít k předvedení programu nebo části kódu, k ukázce, jak se změní funkce programu při změně určité části kódu, nebo k předvedení, jak se bude aplikace chovat vlivem chyby v kódu.
- Brainstorming – Spočívá v bezprostředním sdělení nápadů na řešení problému. Postupně se touto cestou hledá optimální řešení. Ve výuce programování často nachází místo při řešení úloh, které rozvíjí schopnost logického myšlení.

1.2 Výuka algoritmizace a programování ve školách

Informatika je ve srovnání s některými tradičními vědními obory poměrně mladá disciplína. Vzhledem k tomu její výuka nemá takovou tradici jako například výuka matematiky nebo fyziky. V informatice musíme navíc zohledňovat její rychlý vývoj, na nějž je třeba brát zřetel i při plánování obsahu výuky a výběru vhodných metod a výukových materiálů.

V současnosti probíhá revize rámcových vzdělávacích programů (RVP). Hlavním cílem revize je inovace výuky informatiky a nově zařazení digitálních kompetencí napříč předměty. Změny reagují na rychlý vývoj v oblasti technologií v průběhu posledních let

a nedostatečný obsah výuky informatiky, která se mnoho let věnovala především uživatelské práci na počítači a práci s kancelářskými aplikacemi.

V této podkapitole popíšu nejprve postavení informatiky v RVP před změnami, poté se zaměřím na postavení informatiky v RVP po revizi. Budu se při tom soustředit na oblast algoritmizace a programování a míru zařazení této oblasti do výuky informatiky.

1.2.1 Algoritmizace a programování v informatice před revizí RVP

Nejprve se zaměřím na situaci na základních školách. Budu vycházet z Rámcového vzdělávacího programu pro základní vzdělávání (RVP ZV) vydaného v roce 2013 [3]. Oblast Informační a komunikační technologie (IKT) byla v RVP ZV v podstatě beze změny už od roku 2007, protože změny v roce 2013 se týkaly převážně zařazení druhého cizího jazyka jako povinného předmětu.

Minimální časová dotace pro oblast IKT byla podle tohoto RVP ZV pouze 1 hodina na 1. stupni ZŠ a 1 hodina na 2. stupni ZŠ.

V RVP ZV pro 1. stupeň se vyskytovaly tyto tematické celky:

- Základy práce s počítačem,
- Vyhledávání informací a komunikace,
- Zpracování a využití informací.

V RVP ZV pro 2. stupeň se vyskytovaly tyto tematické celky:

- Vyhledávání informací a komunikace,
- Zpracování a využití informací.

Povinná výuka oblasti IKT tak na 1. stupni ZŠ pokrývala jen nejběžnější uživatelské znalosti a činnosti – základy práce v operačním systému, znalost základního hardwaru a běžných periferií, ochranu a zálohování dat, bezpečnost při práci na počítači, vyhledávání informací na internetu a komunikaci jeho prostřednictvím a základy práce v textovém a grafickém editoru.

Na 2. stupni ZŠ se k uvedeným výstupům přidávalo ověřování věrohodnosti informací a informačních zdrojů, základní estetická a typografická pravidla, práce s informacemi v souladu se zákony o duševním vlastnictví a prezentace údajů v textové, grafické a multimediální formě.

Z výše uvedeného plyne, že oblast algoritmizace a programování se v RVP ZV vůbec nevyskytovala. Školy sice měly možnost při tvorbě školního vzdělávacího programu (ŠVP) posílit výuku oblasti IKT z disponibilní časové dotace a vybraná témata zařadit, v praxi však ne vždy tato možnost byla využita.

Pro zjištění obsahu výuky v oblasti informačních a komunikačních technologií na středních školách jsem prostudoval Rámcový vzdělávací program pro gymnázia (RVP G) a Rámcový vzdělávací program středního odborného vzdělávání (RVP SOV).

Původní RVP G byl vydán v roce 2007 a v oblasti Informatika a informační a komunikační technologie se vyskytují tyto tematické celky:

- Digitální technologie,
- Zdroje a vyhledávání informací, komunikace,
- Zpracování a prezentace informací. [4]

Algoritmizace úloh a úvod do programování se zde sice vyskytují, ale pouze okrajově jako jedna ze tří součástí tematického celku Zpracování a prezentace informací.

U středního odborného vzdělávání je situace složitější na prostudování, protože každý obor vzdělávání má svůj RVP [5]. Prozkoumáním několika z nich jsem ale zjistil, že oblast Informatické vzdělávání se v nich výrazně neliší. V RVP SOV před revizí se vyskytovaly tyto tematické celky:

- Práce s počítačem, operační systém, soubory, adresářová struktura, souhrnné cíle,
- Práce se standardním aplikačním programovým vybavením,
- Práce v lokální síti, elektronická komunikace, komunikační a přenosové možnosti Internetu,
- Informační zdroje, celosvětová počítačová síť Internet.

Ani v průběhu středního odborného vzdělávání se tedy většina žáků s algoritmizací a možnostmi tvorby vlastních programů nesetkala.

Výjimkou je obor Informační technologie, který je ze své povahy k rozšířené výuce informatiky předurčen a nabízel kromě výše uvedených tematických celků v oblasti Informatické vzdělávání i předměty v oblasti Programování a vývoj aplikací a v dalších odborných oblastech.

1.2.2 Algoritmizace a programování v informatice po revizi RVP

Výuka informatiky s probíhajícím rozvojem v oblasti digitálních technologií stále více zaostávala za reálnými potřebami dnešního světa a současné společnosti. V roce 2007, kdy byl obsah IKT v RVP vytvořen, nebyly například běžně rozšířené chytré telefony a připojení k internetu z mobilu bylo spíše okrajovou záležitostí.

V současnosti se stávají stále běžnějšími prvky chytré domácnosti. Do lokální sítě už nepřipojujeme jen počítače, notebooky a mobily. Na dálku můžeme ovládat třeba chytrou žárovku nebo vytápění. Pro trend digitalizace a automatizace v průmyslu se vžil pojem Průmysl 4.0. Kromě uživatelské znalosti práce s počítačem a obsluhy programů je také stále více potřeba hlouběji porozumět principu práce počítače a fungování programů. Vzrostly také nároky na práci s daty. Z velkého množství zdrojů, kterými jsme dnes obklopeni, je třeba umět vybrat relevantní informace. Revize RVP reaguje na popsaný vývoj.

V případě RVP ZV se revize označuje jako tzv. Malá revize. Dochází nejen k aktualizaci tematických celků ve výuce informatiky, ale také k zavedení tzv. digitálních kompetencí. Tyto kompetence žákům pomáhají orientovat se v digitálním prostředí, vedou je k bezpečnému a tvořivému využívání digitálních technologií při práci, při učení, ve volném čase i při zapojování do společnosti a občanského života. [6]

V praxi to znamená, že se přesouvá výuka programového vybavení (aplikací) do jiných předmětů tak, aby se žáci s programy seznamovali na praktických příkladech. Například formátování textu si vyzkouší při psaní slohové práce v českém jazyce a s grafickým editorem se seznámí ve výtvarné výchově při tvorbě počítačové grafiky. Formálně je oblast IKT v RVP ZP nahrazena oblastí Informatika (INF) a obsah této oblasti se změnil tak, aby nešlo o základy uživatelské práce s počítačem a programy, ale o práci s daty a hlubší pochopení funkce počítače a programů.

Minimální časová dotace pro oblast INF je nově na 1. stupni ZŠ 2 hodiny na 2. stupni ZŠ 4 hodiny.

V RVP ZV [6] pro 1. stupeň se vyskytují tyto tematické celky:

- Data, informace a modelování,
- Algoritmizace a programování,
- Informační systémy,
- Digitální technologie.

Tematické celky pro 2. stupeň ZŠ jsou pojmenované shodně jako celky pro 1. stupeň, ale obsahují pokročilejší očekávané výstupy a uvádí více učiva.

Vidíme, že časová dotace na informatiku byla citelně zvýšena a informatika už nemá roli pouze „okrajového“ předmětu. Algoritmizace a programování jsou navíc přímo jedním z tematických celků. Toto téma tak získává náležitou pozornost.

Základní školy měly povinnost přepracovat své ŠVP v souladu s revidovaným RVP ZV a podle upravených ŠVP začít vyučovat nejpozději od 1. 9. 2023 ve všech třídách na 1. stupni a od 1. 9. 2024 ve všech třídách na 2. stupni. Pro podporu těchto změn vznikly stránky Informatické myšlení, kde jsou uvedeny vzorové ŠVP a množství materiálů, které můžou vyučující využít při přípravě na výuku.

Výuka algoritmizace a programování na základní škole předpokládá využití blokového programovacího jazyka. Poměrně rozšířený je jazyk Scratch, pro který je na stránkách Informatické myšlení vypracována rozsáhlá série výukových materiálů, např. [7]. Kromě Scratche je však dostupná řada alternativ. Každopádně to znamená, že žáci budou na střední školy již přicházet se znalostí základů algoritmizace.

Aktualizací prošly i rámcové vzdělávací programy pro střední školy. Aktualizovaný RVP G, podle kterého musejí gymnázia začít vyučovat nejpozději od 1. 9. 2025, uvádí nově tyto tematické celky:

- Data, informace a modelování,
- Algoritmizace a programování,
- Informační systémy,
- Digitální technologie.

RVP SOV uvádějí stejné tematické celky jako RVP G, pouze celek Algoritmizace a programování je nahrazen celkem Tvorba, testování a provoz softwaru.

2 DOSTUPNÉ ZDROJE PRO VÝUKU JAZYKA C#

V této kapitole zpracovávám přehled zdrojů, které se zabývají výkladem programování v jazyce C#, a vybrané zdroje podrobněji popisuji. Pro lepší orientaci jsem rozdělil zdroje do dvou kategorií – tištěné publikace a webové stránky. U každé kategorie nejprve stručně shrnu její výhody a nevýhody a poté uvedu konkrétní zástupce.

Jelikož se moje práce zabývá výukou začátečníků ve škole, zaměřuji se při výběru materiálů primárně na česky psané zdroje, zmíním však i některé anglické.

Při výběru výukového materiálu je potřeba klást patřičný důraz na úroveň pokročilosti. Pokud člověk s programováním začíná, učení mu výrazně usnadní, pokud je v materiálu důkladně popsáno vývojové prostředí a teorie je ilustrována na množství názorných příkladů. Některé publikace nejsou vhodné pro výuku, jedná se zejména o programátorské příručky, které sice podrobně popisují všechny možnosti jazyka a věnují se i různým pokročilým nuancím v kódu, ale pro začátečníka jsou příliš náročné, málo srozumitelné a neobsahují dostatek úloh na procvičení.

2.1 Tištěné publikace

Největší slabinou tištěných publikací je obtížná možnost jejich aktualizace. Samotné napsání knihy je dlouhodobou záležitostí, další čas zabere korektura a vydání knihy. Proto se snadno může stát, že návod v knize už v době jejího vydání nepracuje s nejnovější verzí jazyka a vývojového prostředí.

I tak si zastánci tištěných publikací mohou vybrat z několika titulů. Zvolil jsem trojici nejaktuálnějších a popisuji je níže.

2.1.1 C# bez předchozích znalostí

Autorem knihy [8] je Pavel Bory a vyšla v nakladatelství Computer Press. Je členěna do 10 kapitol, každá kapitola obsahuje výklad určitého tématu doplněný o ukázky kódů nebo částí programů. Na závěr kapitoly jsou uvedeny kontrolní otázky, řešená cvičení a neřešená cvičení určená k dalšímu procvičení.

Tato kniha je určena začátečníkům a je přístupná všem zájemcům o programování, kteří mají běžné uživatelské znalosti práce s počítačem. Knihu je možné zakoupit v tištěné podobě nebo jako e-knihu. Ukázkové programy použité v knize je možné stáhnout z webových stránek nakladatelství.

V první kapitole je podrobně popsáno vývojové prostředí a jeho instalace, druhá až šestá kapitola postupně vysvětluje datové typy, proměnné, podmínky, cykly, pole, ladění programu a metody. Sedmá a osmá kapitola podává základy objektově orientovaného programování. V deváté kapitole je vyložena práce se soubory a desátá kapitola shrnuje vysvětlenou problematiku na praktických komplexních příkladech.

Celá kniha se věnuje programování v konzoli, možnost tvorby okenních aplikací je zmíněna až v závěru knihy a pouze jako jeden z možných směrů, kterými se čtenář může ubírat. Jedná se o „klasický“ postup, který se vyskytuje ve většině zdrojů zaměřených na výuku programovacího jazyka C#.

Na této publikaci oceňuji srozumitelně a čtivě vysvětlenou problematiku, dostatek praktických příkladů a ukázek kódu i uvedení množství řešených cvičení. Kniha je vhodně členěná, obsahuje rejstřík a v jejím závěru nalezneme odpovědi na kontrolní otázky uvedené na konci každé kapitoly.

Malou nevýhodu shledávám v tom, že kniha neprošla od svého prvního vydání v roce 2016 výraznější aktualizací. Osobně mám k dispozici druhé vydání z roku 2022, které využívá verzi Microsoft Visual Studio 2015. Pokud si začátečník stáhne nejnovější verzi Visual Studia, může být zpočátku mírně zmatený (zejména mírně odlišným vzhledem některých částí prostředí).

Kniha nemá přímé pokračování, autor v jejím závěru pouze nastiňuje tři možné cesty, kterými se čtenář může vydat. V případě zájmu o studium některé z těchto oblastí je proto třeba vybrat jiný vhodný zdroj. Ačkoliv tato kniha není koncipována jako učebnice pro školní výuku, považuji ji za vhodný materiál pro vyučujícího, kterému může při výuce poskytnout oporu.

2.1.2 Moderní programování

Autorem této série učebnic je Radek Vystavěl a vydala ji autorova vlastní firma moderníProgramování s. r. o. Série se skládá ze tří dílů, přičemž k prvním dvěma je k dispozici i sbírka úloh. [9] [10] [11] [12]

První díl je určen pro začátečníky, druhý pro středně pokročilé a třetí díl pro pokročilé. Tyto učebnice používají netradiční způsob výkladu, protože se od samého začátku věnují tvorbě okenních aplikací Windows Forms. Při tom postupně vysvětlují všechny potřebné pojmy a syntaxi jazyka. Velkou výhodou tohoto přístupu je, že se čtenář od samého začátku

s problematikou seznamuje při tvorbě vizuálně atraktivních a zajímavých aplikací. Knihy je možné zakoupit v tištěné i elektronické podobě a ukázkové programy je možné stáhnout ze stránek autora.

První díl je členěn do 12 kapitol. Nejprve seznamuje čtenáře s vývojovým prostředím a jeho instalací a se základy tvorby okenních aplikací. Následují kapitoly o datových typech, proměnných, grafice, podmínkách a cyklech. V závěrečné kapitole čtenář aplikuje získané znalosti při tvorbě dvou her. V pátém, upraveném vydání se navíc objevuje bonusová kapitola, ve které jsou vyloženy základy tvorby aplikací pomocí modernější technologie WPF. V každé kapitole je učivo vysvětlované pomocí množství příkladů.

Na této sérii publikací oceňuji netradiční a poutavé pojetí, přehlednost učebnic a úzké propojení vysvětlované teorie s jejími praktickými aplikacemi. Užitečné je také doplnění učebnic sbírkami, které nabízejí k procvičení množství řešených i neřešených úloh. Ty jsou rozličné – nejčastěji se vyskytují úlohy na tvorbu zadaného programu, úpravu vytvořeného programu nebo na nalezení a odladění chyb v programu.

Za zmínku také stojí, že první díl byl při pátém dotisku v roce 2019 aktualizován a doplněn o nejnovější informace. Druhý (vydán v roce 2008) a třetí (vydán v roce 2011) díl už aktualizovány nebyly, což nepovažuji za velký problém, protože lze předpokládat, že člověk, který již základní znalosti programování zvládl v prvním díle, si s případnými drobnými odchylkami zvládne poradit.

Určitou nevýhodou knihy může být úplná absence práce s konzolí. To může způsobovat problémy začátečníkovi, který bude hledat informace například v jiném zdroji. Základy práce s konzolí je však možné se rychle doučit z mnoha tutoriálů na internetu.

Tyto knihy se prezentují v názvu jako učebnice. Ačkoliv nemají schvalovací doložku MŠMT, dokážu si dobře představit jejich využití jako výukového materiálu například v semináři na střední škole. Výhodou v tomto ohledu je rozdělení obsahu do tří dílů (nevzniká tak jedna obsáhlá publikace, ale tři tenčí díly) a také existence propracovaných sbírek úloh.

2.1.3 Programování v C# od základů k profesionálnímu použití

Autorem knihy je Miroslav Virius a vyšla v nakladatelství Grada. Rozhodl jsem se ji zmínit zejména z toho důvodu, že je mezi publikacemi nejnovější – byla vydána v roce 2021. Je členěna do 18 kapitol a o řadě témat pojednává vyčerpávajícím způsobem.

Jako výukový materiál pro běžného žáka na střední škole však není vhodná právě kvůli své vysoké odbornosti. Velké množství základní, ale důležité problematiky je shrnuto v několika prvních kapitolách. Kniha sice uvádí demonstrační příklady, neobsahuje však žádné kontrolní otázky nebo úlohy na procvičení. Jedná se proto tedy spíše o kvalitní programátorskou příručku než o vhodný výukový zdroj pro začátečníka.

2.2 Výukové weby

Nespornou výhodou webových materiálů je snadná možnost jejich aktualizace. Webové kurzy mohou pružně reagovat na novinky v jazyce a ve vývojovém prostředí.

Při výběru výukového webu je třeba zohlednit, zda je pravidelně aktualizován a zda pokrývá problematiku v dostatečném rozsahu. Můžeme narazit na stránky, o které se autor už dlouhou dobu nestará nebo u kterých zpracoval jen malou část témat a pak se tvorbě webu přestal věnovat. Takové stránky jsou k účelům výuky obvykle nevhodné. Níže uvádím dva kvalitní weby, jeden v češtině a druhý v angličtině.

2.2.1 Stránky www.itnework.cz

Tyto stránky se prezentují jako sociální síť pro programátory a zájemce o programování a stojí za nimi společnost ITnetwork s. r. o. Kromě blogu a nabídky prezenčních kurzů na nich v záložce „IT e-learning“ najdeme i webové kurzy mnoha programovacích jazyků.

Sekce pro výuku jazyka C# [13] je propracovaná a strukturovaná do několika kapitol. Pro začátečníka je vhodná kapitola „Základní konstrukce jazyka C# .NET“, která se skládá celkem ze 17 lekcí. V nich se čtenář seznámí postupně se všemi základními dovednostmi (vývojové prostředí, datové typy, proměnné, podmínky, pole, cykly a výjimky). Lekce základního kurzu jsou zdarma, registrovaný účastník si k nim může zakoupit i cvičení.

Po absolvování základního kurzu je možné vybrat si z množství dalších, v tomto případě už placených (např. objektově orientované programování, tvorba aplikací ve Windows Forms nebo WPF, propojení aplikací s databázemi).

Výhodou stránek je přehledné zpracování učiva a uvedení množství ukázek, které si čtenář může ihned spustit a vyzkoušet přímo ve webovém prohlížeči (webové stránky jsou tomu uzpůsobeny), případně je i drobně modifikovat a testovat. Jazyk stránek je místy hovorový a některá témata jsou vysvětlována trochu odlehčenou formou, například na

situačních příkladech z populárních seriálů. Úlohy k procvičení nabízejí okamžité vyhodnocení, přístup k nim je však placený.

Využití základního kurzu si dokážu představit ve školní výuce. Další kurzy jsou vzhledem k jejich zpoplatnění vhodné spíše pro zájemce k individuálnímu samostudiu.

2.2.2 Stránky www.w3schools.com

Jedná se o stránky v angličtině, které jsou známé zejména mezi tvůrci webů, protože se detailně zabývají problematikou HTML, CSS a JavaScriptu. Základy Programování v C# jsou na těchto stránkách ale také kvalitně vysvětlené. [14]

Kurz programování v C# je přehledně rozdělen do sekcí, každá sekce se dělí do kapitol. V každé kapitole je nejprve vysvětlena teorie, poté následuje ukázka, kterou si lze vyzkoušet a modifikovat přímo ve webovém prohlížeči. Kladně hodnotím cvičení uvedené na závěr každé kapitoly, pomocí kterého je možné si ověřit pochopení dané problematiky.

Kurz je zdarma a považuji ho za vhodný k využití ve výuce, pokud žáci i vyučující umí anglicky na dostatečné úrovni.

3 METODIKA K PRACOVNÍM LISTŮM

V této kapitole se nachází metodika k vytvořeným pracovním listům. Jejich cílem je posílit mezipředmětové vazby a ukázat žákům i jiné možnosti řešení úloh, než jsou běžně zvyklí používat v matematice.

Pracovní listy jsou určeny pro žáky ve vyšších ročnících střední školy, kteří už základy programování v C# bezpečně ovládají a jsou schopní je využít k řešení matematických úloh. Jeden pracovní list je koncipován jako náplň dvou vyučovacích hodin.

Tato metodika je určena pro vyučujícího, který zde najde komentáře k úkolům v pracovních listech. Vyučujícímu by tato metodika měla usnadnit orientaci v pracovních listech.

3.1 Prvočísla a složená čísla

- Úkol 1 je zařazen, aby si žáci při řešení příkladů zopakovali význam pojmů prvočíslo a složené číslo. Zároveň tento úkol poskytuje údaje pro testování programu v úkolu 2.
- Úkol 2 rozvíjí pochopení programových konstrukcí a schopnost tyto konstrukce vhodně využít v praxi. Žáci se v tomto úkolu setkávají s numerickým přístupem k řešení matematické úlohy.
- Úkol 3 je zařazen k ověření, zda žáci kód v úkolu 2 pochopili. Tento úkol vede také k rozvoji vyjadřovacích schopností žáků (správné a výstižné interpretace algoritmu).
- V úkolu 4 žáci poznají nový datový typ vhodný pro ukládání velkých čísel. Smyslem tohoto úkolu je upozornit na to, že při práci s velkými čísly je třeba použít jiný datový typ než *int*. Je vhodné, aby při plnění tohoto úkolu žáci mohli vyhledávat na webu.
- V úkolech 5 a 6 žáci pracují s pojmem efektivita algoritmu. Smyslem těchto úkolů je, aby si žáci uvědomili, že kvalitní program by měl být navržen tak, aby zvládl efektivně zpracovat i velké množství dat.
- Úkol 7 je zařazen, aby si žáci procvičili početní rozklad na prvočinitele. Zároveň tento úkol poskytuje údaje pro testování programu v úkolu 8.
- Úkol 8 rozvíjí pochopení programových konstrukcí a schopnost tyto konstrukce vhodně využít v praxi. Žáci se v tomto úkolu opět setkávají s numerickým přístupem k řešení matematické úlohy.
- Úkol 9 rozvíjí schopnost modifikovat algoritmus a popsat důsledky této modifikace.
- Úkol 10 je zařazen k ověření, zda žáci pochopili kód v úkolu 8. Tento úkol vede také k rozvoji vyjadřovacích schopností žáků (správné a výstižné interpretace algoritmu).

Při tvorbě tohoto pracovního listu byly použity zdroje [15], [16], [17] a [18].

3.2 Krácení zlomků a největší společný dělitel

- Úkoly 1 a 2 rozvíjí pochopení programových konstrukcí a schopnost tyto konstrukce vhodně využít v praxi. Žáci se v těchto úkolech setkávají s numerickým přístupem k řešení matematické úlohy. Úkol 1 také rozvíjí komunikační schopnosti žáků.
- Smyslem úkolu 3 je, aby si žáci procvičili početní krácení zlomků. Zároveň tento úkol poskytuje údaje pro testování programu z úkolu 1.
- V úkolu 4 se žáci seznámí s pojmem efektivita algoritmu. Smyslem úkolu je, aby si žáci uvědomili, že kvalitní program by měl zvládnout efektivně zpracovat i velké množství dat.
- V úkolu 5 si žáci procvičí početní nalezení největšího společného dělitele. Zároveň tento úkol poskytuje údaje pro testování programu v úkolu 6.
- Úkol 7 u žáků rozvíjí schopnost vytvořit program podle zadaného algoritmu.
- Úkol 8 navazuje na úkol 4 a rozvíjí u žáků schopnost navrhovat efektivní programy.

Při tvorbě tohoto pracovního listu byly použity zdroje [15], [16], [17] a [18].

3.3 Vzdálenost bodů a počítání s vektory

- Řešením úkolu 1 si žáci zopakují výpočet vzdálenosti dvou bodů v rovině podle vzorce nebo použitím Pythagorovy věty. Zároveň tento úkol poskytuje údaje pro testování programu v úkolu 2.
- Úkol 2 představuje žákům možnost, jak je možné algoritmizovat výpočet vzdálenosti dvou bodů v rovině.
- Úkol 3 vede žáky k objevení nového příkazu a ukazuje jim, že dva kódy s různým zápisem mohou mít tutéž funkci.
- V úkolu 4 si žáci otestují porozumění programu z úkolu 2 při jeho modifikaci.
- Úkol 5 slouží k ověření funkčnosti programu vytvořeného v úkolu 4.
- Řešením úkolů 6, 8 a 10 si žáci zopakují výpočet délky vektoru, skalárního součinu vektorů a odchylky vektorů. Zároveň tento úkol poskytuje údaje pro testování programů v úkolech 7, 9 a 11.
- Úkoly 7, 9 a 11 u žáků rozvíjí schopnost vytvořit program podle zadaného algoritmu.

- V úkolu 11 si žáci procvičí na vhodném příkladu využití metod ke zpřehlednění zápisu kódu programu.
- V úkolu 12 si žáci procvičí přetěžování metod na typickém příkladu použití této programovací techniky.

Při tvorbě tohoto pracovního listu byly použity zdroje [19], [16], [17] a [18].

3.4 Aritmetická posloupnost

- Řešením úkolů 1a, 2a a 3 si žáci zopakují vzorce pro počítání s aritmetickou posloupností. Zároveň úkoly poskytují údaje pro testování programů v úkolech 1b, 2b, 4 a 5.
- Úkoly 1b a 2b rozvíjejí pochopení programových konstrukcí a schopnost tyto konstrukce vhodně využít v praxi a modifikovat je. Žáci se v těchto úkolech setkávají s numerickým přístupem k řešení matematické úlohy.
- Úkoly 4 a 5 navazují na úkoly 1b a 2b, ale na rozdíl od nich vedou žáky ke tvorbě univerzálních a znovupoužitelných metod.
- V úkolu 6 žáci ověří funkčnost metod vytvořených v úkolech 4 a 5.
- Řešením úkolů 7 a 8 si žáci vyzkouší početní i numerické řešení složitějších úloh o posloupnostech a přesvědčí se o tom, že při programování je výhodné používat metody, aby byl kód znovupoužitelný.

Při tvorbě tohoto pracovního listu byly použity zdroje [20], [16], [17] a [18].

3.5 Faktoriál a kombinační číslo

- Úkol 1 je zařazen, aby si žáci při řešení příkladů zopakovali výpočet faktoriálu. Zároveň tento úkol poskytuje údaje pro testování programu v úkolu 2.
- Úkol 2 rozvíjí pochopení programových konstrukcí a schopnost tyto konstrukce vhodně využít v praxi. Žáci se v tomto úkolu setkávají s numerickým přístupem k řešení matematické úlohy.
- V úkolu 3 si žáci vyzkouší význam použití různých datových typů pro čísla. Zjistí také řádově maximální hodnotu, kterou dokáže daný datový typ uložit.
- V úkolu 4 mají žáci formulovat, co se dozvěděli v úkolu 3. Úkol trénuje vyjadřovací schopnosti žáků.
- V úkolu 5 žáci zjišťují přesné maximální hodnoty, které lze uložit do proměnných s daným datovým typem. Úkol může sloužit také jako kontrola správnosti řešení úkolu 3.

- Úkol 6 je zařazen, aby si žáci na praktickém příkladu zkusili využít definici kombinačního čísla. Zároveň tento úkol poskytuje hodnotu pro testování programu v úkolu 7.
- Úkol 7 rozvíjí pochopení programových konstrukcí a schopnost tyto konstrukce vhodně využít v praxi. Žáci se v tomto úkolu setkávají s numerickým přístupem k řešení matematické úlohy.
- V úkolu 8 si žáci vyzkouší řešení nestandardního problémového zadání. Zároveň tento úkol žáky vede k uvědomění, že při využití numerického přístupu je třeba zohledňovat, jak se bude program chovat při zadání vysokých čísel.
- V úkolech 9 a 10 si žáci vyzkouší aplikaci kombinatoriky při řešení slovních úloh.

Při tvorbě tohoto pracovního listu byly použity zdroje [21], [22], [16], [17] a [18].

II. PRAKTICKÁ ČÁST

1 ZÁKLADNÍ POJMY

1.1 Programovací jazyk

Programovací jazyky nám umožňují vytvářet počítačové programy, tedy využít počítač k řešení určitého problému.

Přístup k řešení problému z pohledu člověka a počítače se značně liší. Počítač je elektrické zařízení a jeho procesor ve své podstatě pracuje se dvěma stavy – přítomnost a nepřítomnost elektrického napětí (1 nebo 0). Využívá tedy binární (dvojkovou) soustavu.

Na strojové úrovni se instrukce procesoru zadávají obvykle v hexadecimální (šestnáctkové) soustavě a závisejí na konkrétním typu procesoru. Z pohledu člověka je tento přístup velmi nepraktický, protože instrukce v této podobě jsou lidsky špatně čitelné (nevyčteme z nich jednoduše jejich funkci). [13]

Z tohoto důvodu vznikají vyšší programovací jazyky. Zápis programu v těchto jazycích je lidsky dobře čitelný (poznáme z něj jeho funkci), avšak není srozumitelný procesoru. Proto musí být každý kód napsaný ve vyšším jazyce před spuštěním přeložen do strojového kódu. Prakticky při programování ve vyšším jazyce se strojovým kódem vůbec nepřijdeme do styku. Je však dobré mít o něm alespoň základní povědomí, abychom programování správně chápali v souvislosti s principem práce počítače.

Můžeme se setkat s rozdělením programovacích jazyků na nižší a vyšší. Zjednodušeně můžeme říct, že nižší jazyky jsou srozumitelnější procesoru počítače, a naopak méně srozumitelné člověku (jsou bližší strojovému kódu). Jejich výhodou je vysoká rychlost (jsou rovnou prováděny – nemusejí být překládány). Nevýhodou pak je, jak už bylo zmíněno, velmi špatná čitelnost pro člověka. U vyšších jazyků je tomu přesně naopak. [13]

Samostatnou kapitolou jsou blokové jazyky, které se používají nejčastěji při výuce programování. Programy v těchto jazycích vytváříme tak, že je složíme z bloků připravených v nabídce. Daní za velkou uživatelskou přívětivost a srozumitelnost je u těchto jazyků nízká rychlost (samy fungují jako programy v některém vyšším jazyce) a omezená nabídka bloků. Přesto jsou tyto jazyky velmi vhodné pro výuku v začátcích programování, protože nám díky své srozumitelnosti dávají možnost soustředit se více na řešení samotného problému než na technické detaily zápisu programu.

1.2 Algoritmus a program

Pojem algoritmus nemá jednoznačnou definici. Můžeme si jej představit jako postup nebo návod, jak řešit určitý problém. Příkladem jednoduchého neprogramátorského algoritmu může být recept v kuchařce nebo návod na sestavení nábytku. [23]

Všechny algoritmy mají společné tyto vlastnosti:

- Konečnost (finitnost) – musí skončit v libovolně velkém, ale konečném počtu kroků.
- Určitost (determinovanost) – každý krok algoritmu musí být jednoznačně a přesně definován, v každé situaci musí být zřejmé, co se má provést.
- Korektnost – algoritmus musí být věcně správný (např. při technickém výpočtu musí být použity správné vzorce).
- Obecnost (univerzálnost) – algoritmus neřeší jeden konkrétní problém, ale skupinu obdobných problémů.
- Vstup a výstup – jsou definovány hodnoty, které je třeba algoritmu zadat, a hodnoty, které jsou výstupem na konci běhu algoritmu.

Správný algoritmus by měl být také efektivní – chceme získat řešení problému použitím současné výpočetní techniky v rozumném čase. Jako počítačový program označujeme zápis algoritmu v libovolném konkrétním programovacím jazyce. [24]

1.3 Syntaxe

Pojem syntaxe označuje soubor pravidel pro zápis formálního jazyka. Mezi tato pravidla patří například způsob zápisu proměnných (znaky povolené v jejich názvu) a všech dalších programových konstrukcí, se kterými se postupně seznámíme.

O syntaxi hovoříme v souvislosti s konkrétním programovacím jazykem. Každý programovací jazyk má definovanou syntaxi, kterou je nutné znát a dodržovat. Pro představu si pojem syntaxe ilustrujme na příkladu pojmenování souboru v operačním systému Windows – název nesmí obsahovat určité znaky a nesmí překročit stanovenou délku. [22]

1.4 Sémantika

Sémantika se zabývá významem konstrukcí používaných v programovacích jazycích. Týká se tedy více pojmu algoritmus, ale částečně závisí i na konkrétním programovacím jazyku (jednotlivé jazyky se mohou lišit v možnosti použití různých programových konstrukcí a tomu je třeba přizpůsobit výběr těchto konstrukcí). [22]

1.5 Programovací jazyk Scratch

Programovací jazyk Scratch patří mezi vizuální jazyky. Vzhledem ke své jednoduchosti, názornosti, dobrému grafickému zpracování a snadné dostupnosti je v současnosti pro účely výuky velmi oblíbený. Je k dispozici v řadě světových jazyků, i v češtině.

Vývoj jazyka Scratch odstartoval v roce 2003 a první verze byla vydána v roce 2007. Jednalo se o verzi desktopovou, kterou bylo nutné nainstalovat na počítač. K většímu rozšíření jazyka došlo po roce 2013, ve kterém byla uvolněna verze 2.0, která nevyžadovala instalaci na počítač a bylo ji možné spustit i ve webovém prohlížeči. [25]

V současnosti je jazyk Scratch dostupný v desktopové i webové verzi. Obě verze jsou k dispozici zdarma. Výhodou webové verze je možnost zřídit si uživatelský účet, ukládat si své projekty a mít tak možnost se k nim rychle dostat z kteréhokoliv místa, kde je možnost připojení k internetu.

1.6 Programovací jazyk C#

Jazyk C# je vyšší objektově orientovaný programovací jazyk vyvinutý firmou Microsoft. Jeho vývoj odstartoval v roce 2000. Od té doby proběhla řada aktualizací, jazyk je tak stále vylepšován. Jazyk C# je založen na technologii .NET, která poskytuje aplikacím běhové rozhraní. [22]

C# je nepřímým potomkem jazyka C, ze kterého čerpá syntaxi. Na rozdíl od jazyka C má ale automatickou správu paměti, což programování v něm zjednodušuje. Původně byl C# určen pouze pro platformu Windows, v poslední době se ale začíná rozšiřovat i na jiné platformy.

U kompilovaných programovacích jazyků musí být nejprve přeložen celý kód a poté je možné program spustit. Interpretovaný jazyk provádí kód postupně za běhu programu. C# je jazyk s virtuálním strojem – kombinuje výhody kompilace a interpretace. [13]

2 VÝVOJOVÉ PROSTŘEDÍ

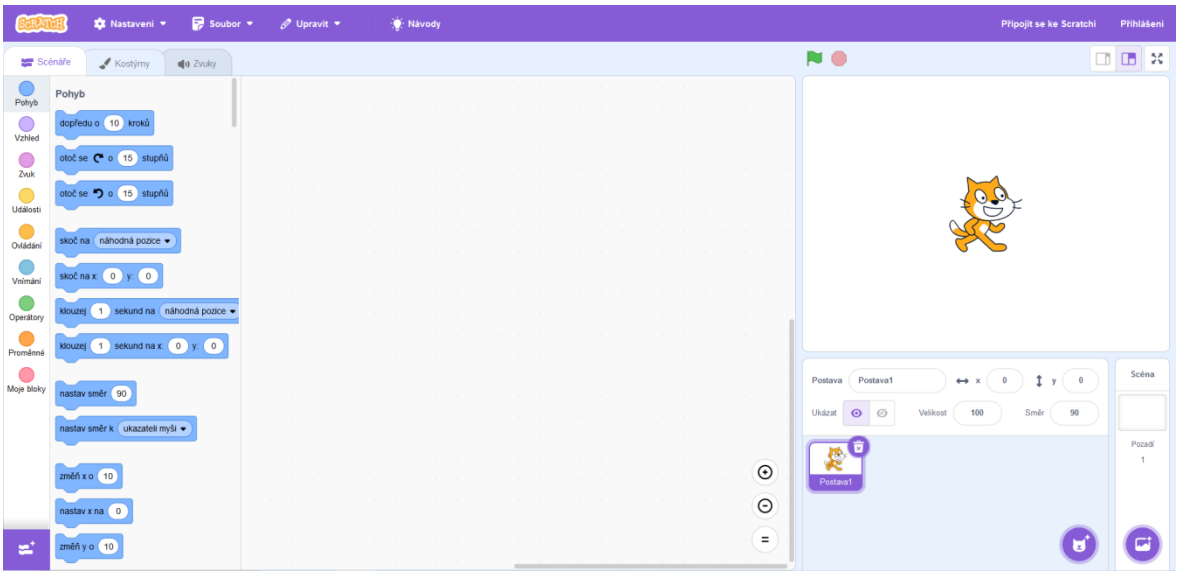
Nezbytným nástrojem ke tvorbě vlastních programů je vývojové prostředí. Proto se nejprve seznámíme s možnostmi, které máme při jeho výběru. Můžeme si volit mezi vývojovými prostředími vyžadujícími instalaci na náš počítač a prostředími dostupnými online.

2.1 Jazyk Scratch

Připojení k internetu je v současnosti standardem, proto se k programování v jazyce Scratch obvykle využívá prostředí, které nalezneme na těchto webových stránkách: https://scratch.mit.edu. Existuje také aplikace, kterou je možné stáhnout a nainstalovat na počítač. Jelikož online prostředí nabízí všechny potřebné funkce a programování v něm je pohodlné, doporučuji ho čtenářům a v dalším výkladu ho budu také používat.

Jazyk Scratch je vytvořený speciálně pro výuku programování, práce s ním je proto velmi jednoduchá a intuitivní. Připomeňme si, že v prostředí Scratche si lze vytvořit účet, v němž můžeme ukládat své projekty. Druhou možností je ukládat soubory do složky v počítači a v případě potřeby je opět načíst (obě operace provedeme kliknutím na *Soubor* v horní části prostředí a výběrem příslušné možnosti).

Na následujícím obrázku je ukázka prostředí Scratche.



Vlevo je nabídka bloků. Bloky mají různou barvu podle funkce, kterou vykonávají. Pomocí barevných teček úplně vlevo můžeme pohodlně přecházet mezi jednotlivými kategoriemi bloků. Uprostřed okna je plocha pro vytváření scénářů (blok jednoduše přetáhneme z nabídky na plochu, kde ho můžeme připojit k jinému bloku). Vpravo nahoře vidíme okno

programu (s postavou), ve kterém vytvořený program po spuštění probíhá (okno můžeme zvětšit na celou obrazovku kliknutím na ikonku s šipkami nad oknem zcela vpravo). Vpravo dole se nachází okno, ve kterém můžeme do projektu přidávat nebo z něj odebírat postavy, a ještě vpravo od něj vidíme malé okno, kde scéně nastavujeme pozadí. [26]

Jazyk Scratch je blokový, proto „pokyny“, co má program provést, označujeme jako bloky. Program s požadovanou funkcí vytvoříme tak, že spojíme potřebné bloky k sobě. Několik takto spojených bloků označujeme jako scénář. Scénáře můžeme spouštět jednoduše kliknutím na jejich první blok. Tento postup se využívá při testování scénáře nebo obsahuje-li náš program pouze jeden scénář.

Druhou možností je zařadit na počátek scénáře blok, který scénář spustí po provedení určité akce (např. po kliknutí na zelenou vlaječku nebo po stisknutí zvolené klávesy). Tento postup se používá, potřebujeme-li spustit více scénářů současně nebo pokud chceme scénář spouštět v okně programu zvětšeném na celou obrazovku. Při běhu programu jsou scénáře, které právě běží, orámovány žlutě. Červený terčík (osmiúhelník) nad oknem programu slouží k zastavení všech probíhajících scénářů. [26]

Při seznamování se s programovacím jazykem je zvykem vytvořit jako první program, který pozdraví slovy „Ahoj, světe!“. Toho docílíme tak, že na plochu pro vytváření scénářů přesuneme blok *bublina* (z kategorie *Vzhled*), dovnitř bloku napíšeme „Ahoj, světe!“ a na blok klikneme. Vidíme, že postava v okně programu hlášku zobrazí.

2.2 Jazyk C#

Pro programování v C# je online dostupná celá řada vývojových prostředí, která jsou pro začátečníka dostačující. Žádné online vývojové prostředí však nenabízí takové možnosti a množství funkcí jako prostředí Visual Studio od firmy Microsoft. Toto prostředí je možné ve verzi Community stáhnout a používat zdarma pro nekomerční účely. Úplný začátečník jeho výhody možná zprvu neocení a bude mu připadat příliš složité. Při tvorbě větších programů a zejména okenních aplikací je ale toto prostředí nezbytné.

Nejprve se seznámíme s vybraným online prostředím a potom s Visual Studiem.

2.2.1 Online vývojová prostředí pro jazyk C#

Použít online vývojové prostředí je výhodné ve chvíli, kdy potřebujeme otestovat funkci malé části kódu nebo pokud potřebujeme vytvořit krátký kód například na počítači, kde není Visual Studio nainstalované a nemáme možnost si ho doinstalovat (např. ve školní studovně, kde k instalaci nemáme potřebná práva).

Online vývojových prostředí existuje více, osobně mám dobrou zkušenost s prostředími Replit (https://replit.com/) a GDB Online (http://www.onlinegdb.com). Replit vyžaduje přihlášení (např. účtem od Googlu) a umožňuje ukládat si vytvořené programy. Prostředí GDB Online lze používat i bez přihlášení a budu se na něj odkazovat v dalším výkladu (pro většinu programů v tomto textu bude online prostředí dostačující).

Po otevření GDB online je nejprve nutné vybrat vpravo nahoře z nabídky *Language* položku *C#*. Prostředí nám připraví strukturu programu – je uvedena na obrázku níže. Aniž bychom něco sami vytvářeli, máme svůj první program. První, světle hnědé řádky jsou komentář a na náš program nemají žádný vliv (můžeme je smazat). Všimněte si lišty s barevnými tlačítky v horní části okna.

Kliknutím na zelené tlačítko *Run* program spustíme. V případě, že program neobsahuje žádné syntaktické chyby, otevře se v dolní části okna konzole, ve které se program bude provádět. Pokud se vyskytnou syntaktické chyby, vypíšou se i s číslem řádku, na němž se pravděpodobně chyba objevila.

Spustíme-li program s kódem výše, vypíše se do konzole text „Hello world“. Povedlo se nám spustit náš první program.

Doporučuji udržovat si v kódu podobnou úpravu jako v ukázce (odsazení řádků od kraje atd.) kvůli snadné orientaci. Je také vhodné si udržovat přehled o příslušných párech složených závorek. Strukturu kódu si zatím jen stručně popíšeme. S podrobnostmi se seznámíme později, až budeme mít k jejich pochopení potřebné znalosti.

Zpočátku budeme všechny příkazy psát dovnitř hlavní metody `static void Main()`, tedy do složených závorek za názvem této metody. Ve výše uvedeném vzoru je v této metodě jediný příkaz – `Console.WriteLine("Hello World")` – sloužící k vypsání hlášky (textu, který je uvedený v uvozovkách v závorce).

Hlavní metoda (stejně jako všechny jiné metody) se nachází uvnitř třídy (*class*), která je důležitá pro pokročilou techniku zvanou objektově orientované programování. Nad definicí třídy se nachází ještě příkaz `using System`. Příkazy začínající slovem *using* se používají k přidání rozšíření (knihoven apod.), se kterými budeme v projektu pracovat (větší projekty mívají těchto příkazů na začátku více).

V jazyce C# se ukončují středníkem všechny řádky, za kterými nenásleduje blok složených závorek (podívejte se na příklad na obrázku výše a uvědomte si, že každý řádek má za sebou buď blok složených závorek, nebo je ukončen středníkem).

Příklady kódu budou v textu uvedeny strojopisným písmem na šedém pozadí a pro jednoduchost bude uveden pouze kód hlavní metody programu (nebude-li výslovně řečeno jinak). Podle tohoto pravidla bude příklad na obrázku výše prezentován tímto způsobem:

```
static void Main()
{
    Console.WriteLine("Hello World");
}
```

Je však třeba vždy pamatovat na to, že pokud chceme uvedený kód spustit, je nutné ve vývojovém prostředí na začátku uvést příkaz `using System` a hlavní metodu umístit dovnitř třídy (tzn. do bloku složených závorek za definicí třídy s určitým názvem – např. `class Program`). Do vývojového prostředí tak napíšeme tento kód:

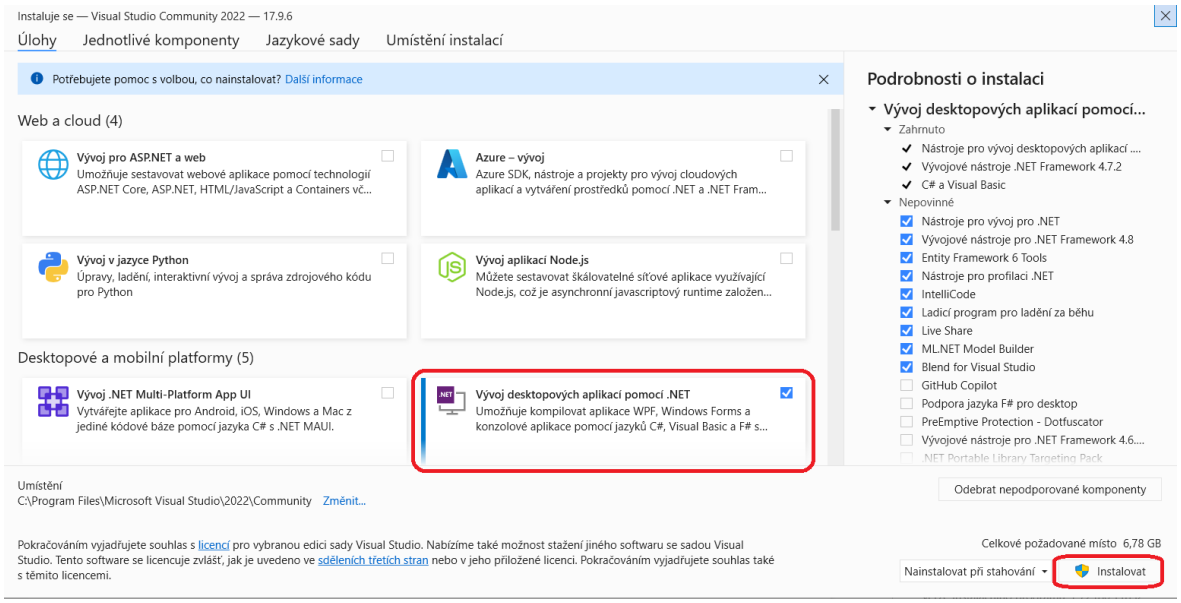
```
using System;
class Program
{
    static void Main()
    {
        Console.WriteLine("Hello World");
    }
}
```

2.2.2 Vývojové prostředí Visual Studio Community

Nyní se seznámíme s Visual Studiem, jehož potenciál plně využijeme při tvorbě větších projektů nebo budeme-li chtít vytvářet okenní aplikace. Visual Studio je nutné před prvním použitím nainstalovat.

Instalační soubor stáhneme z těchto stránek: https://visualstudio.microsoft.com/cs/. Na stránce najdeme Visual Studio a vybereme verzi Community (v době vzniku této práce je nejnovější verze Visual Studio Community 2022). Spuštěním instalačního souboru dojde k nainstalování programu Visual Studio Installer.

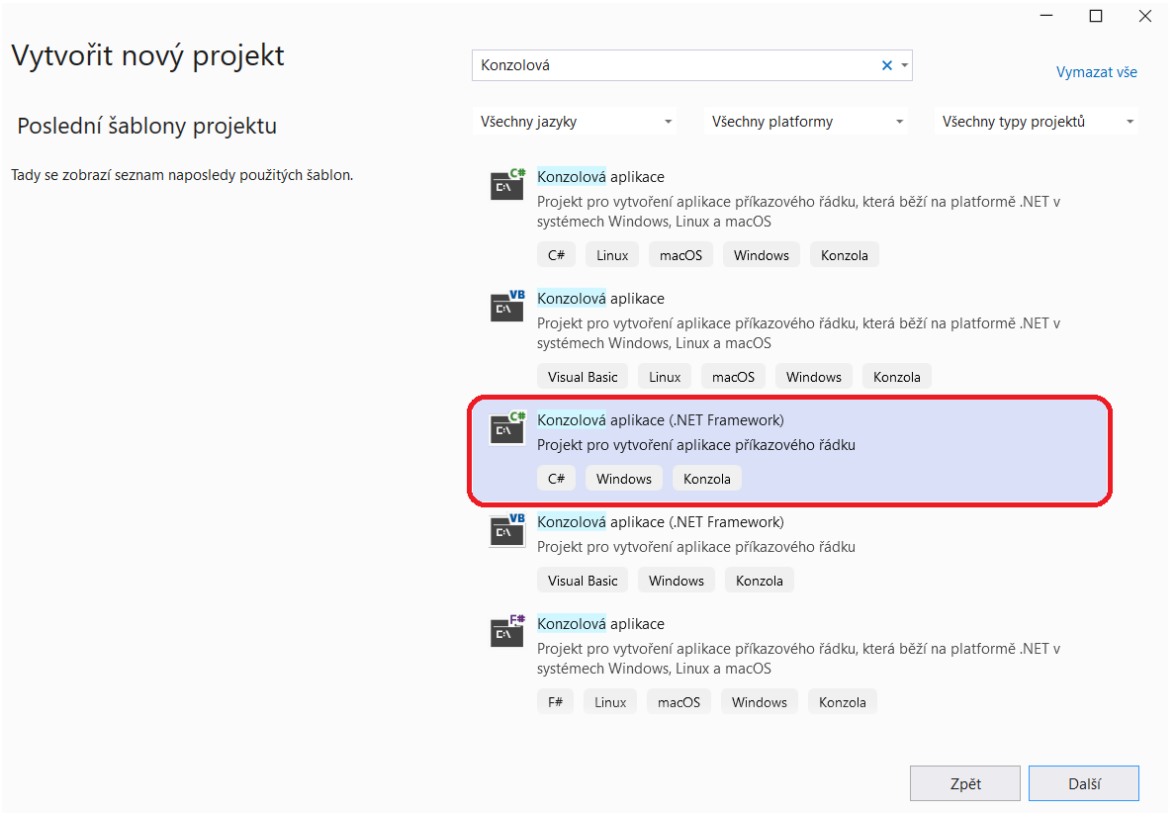
V nabídce, která se nám poté otevře, vybereme *Vývoj desktopových aplikací pomocí .NET* a klikneme na *Instalovat*. Instalace bude trvat několik minut a na její dokončení budeme upozorněni.



Nyní máme vše připravené na to, abychom mohli ve Visual Studiu vytvořit a spustit svůj první program. Spustíme Visual Studio a v okně, které se nám otevře, z nabídky vybereme *Vytvořit nový projekt*. Poté vybereme možnost *Konzolová aplikace (.NET Framework)* s logem C# (viz další obrázek). V dalším okně zadáme název projektu (např. *PrvniAplikace*) a zvolíme jeho umístění. Doporučuji zaškrtnout políčko *Umístit řešení a projekt do stejného adresáře*. V kolonce *Architektura* zvolíme nejnovější verzi *.NET Framework 4.8*. Po potvrzení bude projekt vytvořen.

Při vytváření nového projektu jste si možná všimli, že kromě možnosti *Konzolová aplikace (.NET Framework)* je v nabídce i *Konzolová aplikace*. První možnost je určena

k tvorbě aplikací pouze pro Windows. Druhá varianta umožňuje tvořit aplikace i pro jiné platformy než Windows. [9]



Do hlavní metody doplníme dva příkazy – `Console.WriteLine("Hello World")` a `Console.ReadKey()`. Kód projektu tak bude vypadat jako na ukázce níže. Chceme-li program spustit, stiskneme na klávesnici klávesu *F5* nebo klikneme na ikonu *Spustit* (se zeleným trojúhelníčkem) v nabídce programu.

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

namespace PrvniAplikace
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            Console.ReadKey();
        }
    }
}
```

Kód se mírně liší od toho, který připravilo prostředí GDB online. Jako první si zřejmě všimneme, že je zde více příkazů začínajících slovem *using*. Ve výchozím stavu jsou do projektu přidána některá další často používaná rozšíření. Rozšíření skutečně využitá v projektu mají černou barvu písma. Rozšíření nevyužitá v projektu mají světle šedou barvu písma (máme-li projekt hotový, lze rozšíření se světle šedou barvou písma smazat).

Dalším rozdílem je, že třída (*class*) je umístěna uvnitř jmenného prostoru (*namespace*). Tento způsob rozdělení projektu se používá u velkých aplikací, které se skládají z mnoha dílčích součástí a vyvíjí je tým několika lidí. Pro nás je podstatné pouze to, že jmenný prostor uvádět vůbec nemusíme, na rozdíl od třídy, která musí být přítomna v každém programu. Drobné odchylky v zápisu třídy a hlavní metody v různých prostředích nejsou podstatné.

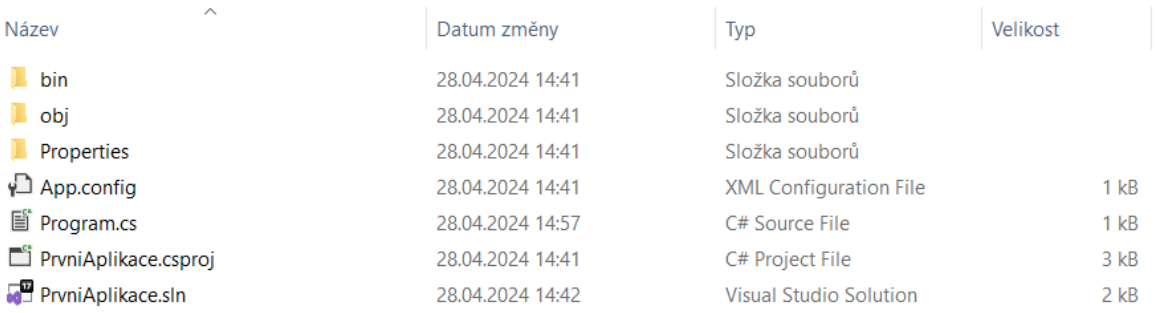
Kromě příkazu `Console.WriteLine("Hello World")`, o němž již víme, že vypisuje text do konzole, je v kódu navíc příkaz `Console.ReadKey()`. Tento příkaz způsobí, že program čeká na stisknutí libovolné klávesy na klávesnici a až poté se ukončí. Pokud bychom tento příkaz nepoužili, nestihli bychom si vypsaný text ani přečíst, protože konzole by se ihned zavřela (jedná se o další odlišnost mezi Visual Studiem a GDB online). [8]

Důležité je, že obě možnosti zápisu kódu jsou vzájemně zaměnitelné. Pokud zkopírujeme kód vytvořený ve Visual Studiu do GDB online, program se zkompiluje a spustí bez problémů. Stejně tak tomu bude v opačném případě, kdy kód vytvořený v GDB online zkopírujeme do Visual Studia.

Při psaní uvedených dvou příkazů jste si zřejmě všimli výhod Visual Studia – jeho nástroje IntelliSense, který nám našeptává příkazy, a kontroly správnosti příkazů (syntaktické chyby jsou ihned podtrhovány červeně).

Podívejme se ještě do složky s naším projektem umístěné na ploše (umístění jsme volili při vytváření projektu). Ve složce vidíme několik složek a souborů. Soubor *PrvniAplikace.sln* je řešení (editovatelný projekt) a standardně se otevírá ve Visual Studiu. Použijeme ho, chceme-li rozpracovaný nebo hotový projekt později upravit. Soubor *Program.cs* obsahuje zdrojový kód našeho programu a je možné ho otevřít i v textovém editoru (např. Poznámkovém bloku). Tento soubor využijeme například ve chvíli, kdy chceme kód programu poslat někomu, kdo nemá na počítači nainstalované Visual Studio. Poslední důležitý soubor je *PrvniAplikace.exe*. Najdeme ho, pokud otevřeme složku *bin* a následně složku *Debug*. Jedná se o výslednou aplikaci – pokud vytvoříme program a chceme někomu poslat pouze

funkční aplikaci, aniž by měl přístup ke zdrojovému kódu, pošleme mu právě tento soubor (soubor vznikne až po prvním spuštění vytvořeného programu).



Název	Datum změny	Typ	Velikost
bin	28.04.2024 14:41	Složka souborů	
obj	28.04.2024 14:41	Složka souborů	
Properties	28.04.2024 14:41	Složka souborů	
App.config	28.04.2024 14:41	XML Configuration File	1 kB
Program.cs	28.04.2024 14:57	C# Source File	1 kB
PrvniAplikace.csproj	28.04.2024 14:41	C# Project File	3 kB
PrvniAplikace.sln	28.04.2024 14:42	Visual Studio Solution	2 kB

V příkladech v tomto textu budu preferovat zápis, který generuje prostředí GDB online, sami se ale můžete rozhodnout, zda si budete zkoušet kódy spouštět v tomto prostředí, nebo ve Visual Studiu.

2.2.3 Komentáře v kódu

Často narazíme na situace, v nichž bude žádoucí přidat do kódu komentář nebo si v něm udělat poznámku. Tato poznámka bude určená pouze pro nás programátory a bude vysvětlovat význam některé části kódu. Kompilátor bude tuto poznámku ignorovat. [8]

Víceřádkovou poznámku začneme pomocí symbolů `/*` a ukončíme ji pomocí symbolů `*/`. Text uvedený mezi těmito symboly kompilátor přeskočí. Komentář na konci řádku začíná dvěma lomítky `//` a platí do konce řádku, na němž je uveden (text od dvou lomítek po konec řádku kompilátor ignoruje). Využití poznámek a komentářů ilustruje příklad níže.

```
using System;
class Program
{
    /* Toto je hlavní metoda
       Zatím budeme veškeré příkazy psát dovnitř této metody */
    static void Main()
    {
        Console.WriteLine("Hello World"); //Příkaz vypíše text do konzole
    }
}
```

V GDB online i ve Visual Studiu komentáře snadno rozeznáme na první pohled díky jiné barvě písma, pomocí které je vývojová prostředí odlišují od zbytku kódu.

3 PROMĚNNÉ, VÝPIS A NAČTENÍ ÚDAJŮ

Načtení a výpis údajů, označované někdy jako uživatelský vstup a výstup, patří mezi základní schopnosti programu. Jejich zvládnutí je nutným předpokladem pro učení se dalších dovedností. Jednoduše si operace načtení a výpisu údajů můžeme přiblížit na příkladu kalkulačky. Té musíme zadat (obvykle) dvojici čísel a operaci, kterou s nimi chceme provést. Kalkulačka poté vypíše výsledek.

Údaje ale nestačí pouze načíst. Abychom s nimi později mohli pracovat, je třeba si je někam uložit.

3.1 Proměnné a výpis údajů

Každého z nás jistě napadne nespočet situací, kdy potřebujeme v počítači uložit určitá data. Při vytváření referátu potřebujeme uložit výsledný soubor, při hraní hry je třeba ukládat jména hráčů a jejich skóre, při objednávání z e-shopu se musí uložit na server prodejce naše objednávka apod.

Proměnnou si můžeme představit jako místo v paměti počítače, kam lze uložit určitá data (určitou informaci). Podle toho, jak s proměnnými programovací jazyky pracují, je dělíme na staticky a dynamicky typované.

Staticky typované jazyky vyžadují při založení proměnné definování typu proměnné, který je dále neměnný. Pokud vytvoříme proměnnou, která je typu celé číslo, už nikdy do této proměnné nemůžeme uložit textový řetězec (a naopak).

Jiný přístup nabízejí dynamicky typované jazyky, u kterých programátor s datovými typy vůbec nepracuje (datové typy řeší vnitřně samotný jazyk). Nemusíme předem specifikovat, jaká hodnota bude v proměnné uložena, a do téže proměnné můžeme teoreticky uložit nejprve číslo a poté jej přepsat slovem (prakticky je však vhodné z důvodu přehlednosti kódu každou proměnnou používat k účelu, za jakým byla vytvořena). [13]

Oba přístupy mají své výhody i nevýhody. Dynamicky typované jazyky jsou vhodnější pro začátečníky, protože u nich odpadá nutnost přemýšlet nad správnou volbou datového typu, proměnnou pouze jednoduše definujeme. Nevýhodou jsou menší možnosti kontroly kódu – některé chyby se mohou projevit až po spuštění programu. U staticky typovaných jazyků nás naopak v řadě případů na chybu upozorní samotné vývojové prostředí. Nevýhodou tohoto přístupu je objemnější kód a větší složitost pro začátečníka.

Každá proměnná musí mít své jméno. Je vhodné proměnné pojmenovávat výstižně, abychom podle jejich názvu poznali, co je v nich uloženo. Dodržování tohoto pravidla nám jednak usnadní orientaci, pokud budeme chtít vytvořený program v budoucnu upravit, a zároveň je nezbytné, pokud pracujeme v týmu, aby ostatní členové mohli na naši práci hladce navázat.

3.1.1 Proměnné a výpis údajů v jazyce Scratch

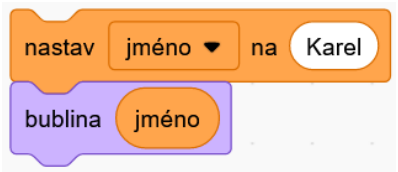
Jazyk Scratch je dynamicky typovaný, práce s proměnnými je v něm proto poměrně jednoduchá. Proměnnou je třeba nejprve vytvořit. To učiníme tak, že vlevo klikneme na kategorii *Proměnné* a poté na bílý obdélníček *Vytvoř proměnnou*. Do okna, které se nám zobrazí, zadáme název proměnné, rozhodneme, zda bude proměnná dostupná pouze pro konkrétní postavu, nebo pro všechny postavy, a potvrdíme.

Na názvy proměnných ve Scratchi nejsou kladeny žádné zvláštní požadavky. Lze v nich používat diakritiku, mezery i speciální znaky. Speciálních znaků je však dobré se spíše vyvarovat, protože se nejedná o dobrou praxi v programování. [26]

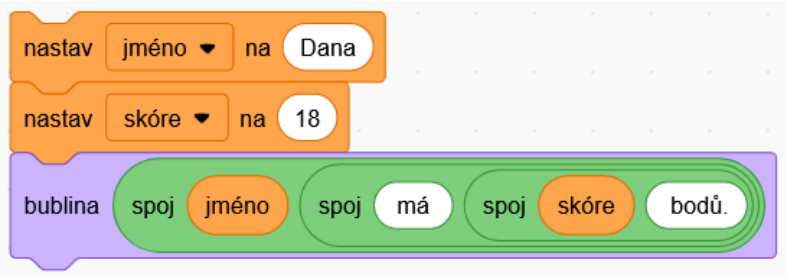
Zvažovat omezení dostupnosti proměnné pouze pro konkrétní postavu má smysl zejména u větších projektů s více postavami. Pokud proměnné, s nimiž pracuje pouze jedna postava, nastavíme jako dostupné pouze pro tuto postavu, jiné postavy je vůbec neuvidí. To nám jednak usnadní orientaci při tvorbě programu, jednak nám to zabrání provést nechtěnou změnu proměnné, což by mělo za následek vznik chyby v programu.

Zaškrtnutím nebo odškrtnutím checkboxu u názvu proměnné můžeme nastavit její viditelnost v okně programu. Pro dobrou přehlednost je vhodné skrýt proměnné, které není nutné zobrazovat.

Na ukázce níže je scénář, ve kterém nejprve uložíme do proměnné *jméno* hodnotu „Karel“ (proměnnou jsme si nejprve založili výše popsaným způsobem) a poté hodnotu uloženou do proměnné vypíšeme pomocí bloku *bublina*. Připomeňme si, že scénář můžeme spustit (provést) kliknutím na jeho první blok. Provedení scénáře sledujeme v okně programu.



Samotné vypsání proměnné nám většinou nestačí. Často potřebujeme vypisovanou proměnnou vložit do nějakého kontextu, tj. nějaké věty, případně do vypisované hlášky vložit více proměnných. Toto ve Scratchi provedeme pomocí operátoru *spoj*, viz následující ukázka.



Při tvorbě scénáře nesmíme zapomenout před i za „má“ a před „bodů“ zapsat mezeru, aby byl zobrazený výsledek dobře čitelný a typograficky správně.

3.1.2 Proměnné a výpis údajů v jazyce C#

Jazyk C# je staticky typovaný, a proto je v něm situace složitější. Při tvorbě proměnné musíme zvolit datový typ, který bude ve zbytku programu neměnný. Přehled základních datových typů uvádí tabulka níže. V posledním sloupci tabulky vidíme příklad vytvoření datového typu a jeho naplnění vhodnou hodnotou. [8]

Datový typ	Velikost	Uchovávaná hodnota	Příklad
int	4 bajty	celé číslo od −2 147 483 648 do 2 147 483 648	`int vek = 32;`
double	8 bajtů	desetinné číslo uchovávající 15 desetinných míst	`double delka = 8.26;`
bool	1 bit	pravdivostní hodnota (*true* nebo *false*)	`bool plnolety = true;`
char	2 bajty	jeden znak (jedno písmeno)	`char pismeno = 'a';`
string	2 bajty na každý znak	sekvence znaků (slovo nebo věta)	`string jmeno = "Adam";`

Výčet datových typů není úplný, ale pro začátečníka je dostačující. Pro práci s velkými celými čísly se nám může hodit datový typ *long* a někdy se také můžeme setkat s uložením desetinných čísel do proměnné s datovým typem *float*, který má ale přesnost jen 6 desetinných míst. Zvláště při numerických výpočtech je proto vhodné používat přesnější datový typ *double*. Vyčerpávající pojednání o datových typech proměnných najdeme například ve zdroji [13] v lekci 4.

Název proměnné se smí skládat pouze z písmen, číslic a podtržítek, nesmí však číslicí začínat. Kromě podtržítka nesmí obsahovat jiné speciální znaky. V názvech proměnných je vhodné využívat pouze znaky anglické abecedy (vyvarovat se použití diakritiky).

K vypsání textu do konzole slouží příkazy `Console.Write()` nebo `Console.WriteLine()`. Do závorek se uvádí název proměnné nebo text v uvozovkách. První z příkazů text pouze vypíše, druhý příkaz text vypíše a odřádkuje (nastaví kurzor na začátek nového řádku). Při opakovaném použití prvního z příkazů se tak bude všechen text vypisovat na týž řádek, při opakovaném použití druhého z příkazů bude text z každého příkazu vypisován na nový řádek. [8]

Níže je ukázka programu, který vypíše do konzole hodnotu uloženou v proměnné (analogie s prvním příkladem v předchozí podkapitole).

```
static void Main()
{
    string jmeno = "Karel";
    Console.Write(jmeno);
}
```

Chceme-li vypsat určitou hlášku doplněnou o hodnoty proměnných, můžeme využít buď spojování (konkatenaci) řetězců, nebo vložení zástupných značek, které budou nahrazeny hodnotami proměnných.

Obě možnosti si ukážeme na analogii druhého příkladu z předchozí podkapitoly. Spojování řetězců a proměnných provádíme pomocí znaménka plus.

```
static void Main()
{
    string jmeno = "Dana";
    int skore = 18;
    Console.Write(jmeno + " má " + skore + " bodů.");
}
```

Zástupné značky jsou čísla od nuly (včetně) výše, která jsou uvedená ve složených závorkách. Za řetězcem následují vypsané proměnné oddělené čárkou, jejichž hodnoty budou vloženy namísto zástupných značek. Níže je uvedeno, jak by výpis hlášky v příkladu výše vypadal s použitím zástupných značek (zbytek kódu zůstane beze změny).

```
Console.Write("{0} má {1} bodů.", jmeno, skore);
```

3.2 Načtení údajů

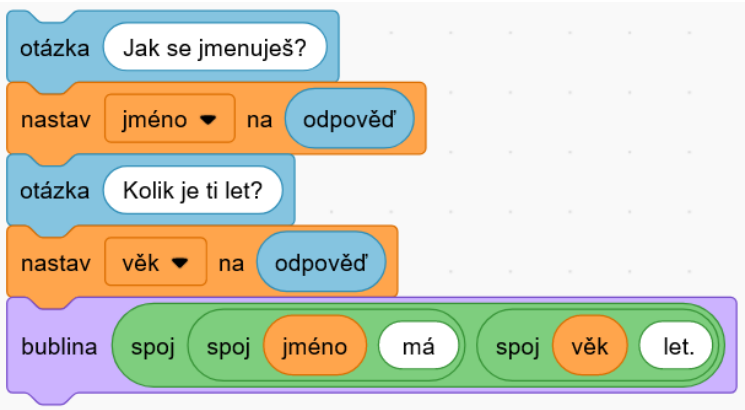
Již jsme se seznámili s tím, jak nějaký údaj uložit a vypsat. K tomu, abychom byli schopní vytvářet praktické aplikace, potřebujeme umět také údaje programu zadat. Opět se nejprve podíváme na situaci v jazyce Scratch a následně se budeme věnovat jazyku C#.

3.2.1 Načtení údajů v jazyce Scratch

Vzhledem k tomu, jak Scratch zjednodušuje práci s proměnnými, je jednoduché i načítání dat od uživatele. Slouží k tomu blok *otázka* (z kategorie *Vnímání*). Do něj napíšeme hlášku zobrazující se v bublině a vyzývající uživatele k zadání proměnné. V okně programu se pak zobrazí pole pro zadání hodnoty. Po zadání a potvrzení je hodnota uložena do proměnné *odpověď* (tento blok najdeme rovněž v kategorii *Vnímání*), která je ve Scratchi vytvořena ve výchozím nastavení. [26]

Téměř vždy je dobré hodnotu z proměnné *odpověď* uložit do vlastní vytvořené proměnné, protože při dalším použití bloku *otázka* bude hodnota v proměnné *odpověď* přepsána. Pouze v situaci, kdy chceme načtenou hodnotu použít okamžitě a v programu ji už nikdy více nebudeme potřebovat, nemusíme hodnotu z proměnné *odpověď* ukládat do vlastní proměnné.

Níže je ukázka scénáře, který se uživatele zeptá na jméno a věk a odpovědi uživatele uloží do proměnných *jméno* a *věk* (které jsme si vytvořili). Poté vypíše hlášku „*jméno* má *věk* let.“ (místo proměnných v kurzívě budou uvedeny konkrétní hodnoty zadané uživatelem).



Podobně jako ve Scratchi nerozlišujeme datové typy proměnných, přistupujeme stejně k načtení textového řetězce i k načtení číselné hodnoty.

3.2.2 Načtení údajů v jazyce C#

V jazyce C# existuje příkaz `Console.ReadLine()`, který načte textový řetězec z konzole. Před jeho použitím je vhodné pomocí příkazu `Console.Write()` informovat uživatele o tom, co má zadat, a zadanou hodnotu je třeba ihned uložit do proměnné datového typu *string*, jak ukazuje následující příklad.

```
static void Main()
{
    Console.Write("Jak se jmenuješ? ");
    string jmeno = Console.ReadLine();
}
```

Pokud bychom se stejným způsobem pokusili načíst celé číslo do proměnné s datovým typem *int*, objeví se chybová hláška. Je tomu tak z toho důvodu, že příkaz `Console.ReadLine()` načítá (vrací) vždy textový řetězec. Chceme-li načíst proměnnou datového typu *int*, je potřeba ji po načtení příkazem `Console.ReadLine()` ještě přetypovat (tzn. říct programu, že načtenou hodnotu chceme uložit do datového typu *int*). [8]

```
static void Main()
{
    Console.Write("Kolik je ti let? ");
    int vek = int.Parse(Console.ReadLine());
}
```

Problém nastane, pokud program očekává zadání číselné hodnoty, ale uživatel zadá textový řetězec (slovo). V takovém případě dojde k chybě. Tyto situace je třeba ošetřovat například tak, aby v případě zadání neplatné hodnoty byl uživatel vyzván k jejímu opakovanému zadání. Jak to můžeme provést, si ukážeme později.

S pomocí získaných informací můžeme vytvořit stejný program jako ve Scratchi. Do závorek příkazů `Console.Write()` a `Console.WriteLine()` můžeme zadat textový řetězec i číslo, příkaz si s vypsáním poradí.

```
static void Main()
{
    Console.Write("Jak se jmenuješ? ");
    string jmeno = Console.ReadLine();
    Console.Write("Kolik je ti let? ");
    int vek = int.Parse(Console.ReadLine());
    Console.WriteLine(jmeno + " má " + vek + " let.");
}
```

Přetypování desetinného čísla se provádí podobně jako přetypování celého čísla.

```
double hmotnost = double.Parse(Console.ReadLine());
```

Je možné načíst i jeden znak (tento příkaz ale použijeme spíše zřídka).

```
char volba = Console.ReadKey().KeyChar;
```

Na závěr si uvedeme důležitou poznámku k terminologii. Jazyk C# je objektový a tato skutečnost se velmi projevuje na tom, jak jazyk vypadá. Příkazy `Console.Write()`, `Console.WriteLine()` a `Console.ReadLine()` jsou ve skutečnosti statické metody třídy *Console*. O metodách si povíme později. Výklad objektově orientovaného programování není náplní tohoto textu, s pojmem třída se můžete seznámit v jiném zdroji, například v knize [8]. Pro zjednodušení budu uvedené metody dále v textu označovat jako „příkazy“.

3.3 Operátory a základní operace

V této kapitole si uvedeme přehled základních operací, které můžeme provádět s proměnnými. Kapitola je rozdělena do pěti podkapitol, v každé z nich je dané téma nejprve popsáno ve Scratchi a poté vysvětleno v C#.

3.3.1 Početní operace s čísly

Už víme, že Scratch nerozlišuje datové typy proměnných. Při práci s ním proto nemusíme řešit, zda pracujeme s celými nebo desetinnými čísly. Všechny bloky zmíněné v této podkapitole nalezneme v kategorii *Operátory*. [26] Pro provádění základních početních operací slouží tyto bloky:

Dovnitř těchto bloků můžeme zapsat čísla, proměnné nebo jejich kombinaci. Pokud je výsledkem dělení periodický desetinný rozvoj, Scratch dokáže zobrazit až 16 desetinných míst. Dávejte si ale pozor na to, že bloky *bublina* a *myšlenka* zaokrouhlují automaticky desetinná čísla na 2 desetinná místa. Toto zaokrouhlení je možné obejít použitím bloku *spoj*. Zkuste si ve Scratchi sestavit a spustit dva níže uvedené scénáře.

Zbytek po dělení můžeme zjistit pomocí tohoto bloku:

K provádění pokročilých matematických operací (odmocnina, goniometrické funkce apod.) je určený tento blok (z nabídky si vybereme požadovanou operaci):

Pamatujte na to, že při zápisu desetinných čísel ve Scratchi i v C# je třeba používat desetinnou tečku. Při použití desetinné čárky budou tuto hodnotu oba jazyky vnímat jako textovou.

K provádění početních operací s čísly a proměnnými slouží běžně používaná znaménka pro tyto operace, jak ilustruje ukázka níže. K určení zbytku po dělení se používá operátor `%`.

```
static void Main()
{
    int x = 15;
    int y = 4;
    int soucet = x + y;
    int rozdil = x - y;
    int soucin = x * y;
    int podil = x / y;
    int zbytekPoDeleni = x % y;
}
```

Obezřetní musíme být při dělení. Dělí-li C# dvě čísla, z nichž alespoň jedno je datového typu *double* (desetinné číslo), bude i výsledek datového typu *double*. Pokud však C# dělí dvě čísla, obě datového typu *int* (celé číslo), výsledkem bude vždy celé číslo – dolní celá část desetinného čísla, které bychom vydělením čísel získali. Chceme-li, aby výsledek dělení dvou celých čísel byl vyjádřen desetinným číslem, musíme alespoň jedno z nich přetypovat na desetinné číslo (viz poslední řádek ukázky níže). Samozřejmě nesmíme zapomenout, že proměnná, do níž výsledek ukládáme, musí být datového typu *double*. [22]

```
static void Main()
{
    int x = 21;
    int y = 6;
    int podil1 = x / y; //výsledkem bude číslo 3
    double podil2 = x / y; //výsledkem bude také číslo 3
    double podil3 = (double)x * y; //výsledkem bude číslo 3.5
}
```

Odmocninu určíme příkazem `Math.Sqrt()`. Pokročilé matematické funkce (např. goniometrické funkce) poskytuje třída *Math*. Zápis příkazů pro tyto funkce najdeme v dokumentaci jazyka na webu.

3.3.2 Inkrementace a dekrementace

Pojmem inkrementace se označuje zvýšení hodnoty v proměnné, pojmem dekrementace pak snížení hodnoty v proměnné. Tyto operace mají samozřejmě smysl pouze pro proměnné obsahující číselné hodnoty.

Ve Scratchi ke změně hodnoty proměnné slouží níže uvedený blok. Název tohoto bloku je záměrně zvolen „změň“, protože jeho použitím můžeme hodnotu v proměnné zvýšit (uvedením kladné hodnoty), nebo snížit (uvedením záporné hodnoty).

Příklad níže ukazuje, jak provedeme zvýšení hodnoty v proměnné (pro snížení bychom uvedli znaménko mínus) v C#. Příkaz `cislo = cislo + 10` se označuje jako přiřazení a je třeba nechápat ho jako rovnici (pak by neměl smysl). Operace přiřazení přepíše hodnotu v proměnné novou (větší nebo menší) hodnotou. [8]

```
static void Main()
{
    int cislo = 50;
    cislo = cislo + 10;
}
```

Vzhledem k tomu, že zvýšení nebo snížení hodnoty proměnné o 1 se provádí často, existují pro tyto operace speciální příkazy:

- zvýšení hodnoty v proměnné *i* o 1: `i++;`
- snížení hodnoty v proměnné *i* o 1: `i--;`

Je samozřejmě nutné, aby v proměnné, kterou chceme inkrementovat nebo dekrementovat, byla uložená nějaká (číselná) hodnota. Pokud takovou operaci provedeme pro prázdnou proměnnou, dojde v programu k chybě. (Tuto situaci je třeba hlídat zejména v C#, ve Scratchi se do každé proměnné po jejím vytvoření automaticky uloží 0.)

3.3.3 Zaokrouhlování

K zaokrouhlování slouží ve Scratchi blok *zaokrouhli…* z kategorie *Operátory*. Tento blok zaokrouhluje vždy na celé číslo (podle matematických pravidel). Blok umožňující zaokrouhlit číslo na daný počet desetinných míst ve Scratchi dostupný není. Existuje ale způsob, jak lze zaokrouhlení na požadovaný počet desetinných míst dosáhnout.

Ukažme si, jak můžeme zaokrouhlit na dvě desetinná místa číslo 0,4283. Nejprve ho vynásobíme 100, poté použijeme blok pro zaokrouhlení a získanou hodnotu na závěr vydělíme 100. Obdržíme tak číslo zaokrouhlené na setiny.

V C# zaokrouhlujeme pomocí příkazu `Math.Round( , )`. Příkaz přijímá dva parametry – zaokrouhlované číslo a počet desetinných míst, na něž bude zaokrouhlovat. Pokud zadáme pouze číslo (tzn. vynecháme počet míst), dojde k zaokrouhlení na celé číslo.

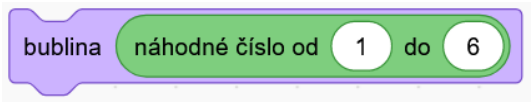
Funkci příkazu si představíme na příkladu, který převede zlomek na desetinné číslo zaokrouhlené na požadovaný počet desetinných míst.

```
static void Main()
{
    Console.Write("Zadejte čitatel: ");
    int citatel = int.Parse(Console.ReadLine());
    Console.Write("Zadejte jmenovatel: ");
    int jmenovatel = int.Parse(Console.ReadLine());
    Console.Write("Zadejte počet desetinných míst: ");
    int mista = int.Parse(Console.ReadLine());
    double podil = (double)citatel/jmenovatel;
    double zaokrouhleno = Math.Round(podil, mista);
    Console.Write("Výsledek: " + zaokrouhleno);
}
```

3.3.4 Náhodná čísla

Náhodná čísla využijeme například při losování hráče, který bude hrát jako první, nebo při výběru otázek z databáze do kvízu.

Ve Scratchi slouží k vygenerování náhodného čísla blok *náhodné číslo od ... do ...* z kategorie *Operátory*. Tento blok vygeneruje náhodné celé číslo z daného rozsahu včetně krajních mezí (obě z nich mohou být vygenerovány). Program napodobující šestistěnnou hrací kostku bude mít ve Scratchi velmi jednoduchý scénář:



Chceme-li generovat náhodná čísla v C#, musíme na začátek programu přidat generátor náhodných čísel příkazem `Random generator = new Random()`. V odborné terminologii se jedná o vytvoření instance třídy *Random* s názvem *generator*, podrobnostem ale opět porozumíte až při hlubším studiu objektově orientovaného programování. [9]

Vygenerování náhodného čísla provedeme příkazem `generator.Next(x,y)`, kde x, y jsou celá čísla. Dejte si pozor na to, že na dolní mez (x) vygenerována být může, zatímco horní mez (y) nikdy vygenerována nebude (v tomto se příkaz liší od bloku ve Scratchi). Vygenerované náhodné číslo buď uložíme do proměnné, nebo ho rovnou vypíšeme.

Program napodobující šestistěnnou hrací kostku bude mít v C# tento kód:

```
static void Main()
{
    Random generator = new Random();
    int nahodneCislo = generator.Next(1,7);
    Console.Write(nahodneCislo);
}
```

3.3.5 Délka textového řetězce

Poslední důležitou operací, kterou si zmíníme, je zjištění délky (tzn. počtu znaků) textového řetězce uloženého do proměnné. Tato operace se nám může hodit například při tvorbě aplikace, ve které si má uživatel nastavit heslo s minimálním počtem znaků, nebo při šifrování zprávy.

Ve Scratchi ke zjištění délky textového řetězce používáme níže uvedený blok. Textový řetězec do něj přímo zapíšeme, nebo do něj vložíme blok s proměnnou, jejíž délku chceme zjišťovat.

Zjištění délky textového řetězce ukazuje následující příklad (pokud chceme navíc délku slova vypsat, musíme ještě doplnit příkaz pro vypsání proměnné do konzole – v tomto případě `Console.WriteLine(pocetZnakuSlova)`). Délka textového řetězce je vždy celé číslo, tuto hodnotu proto můžeme uložit do proměnné datového typu `int`. [8]

```
static void Main()
{
    string slovo = "počítač";
    int pocetZnakuSlova = slovo.Length;
}
```

4 VĚTVENÍ PROGRAMU

Podmínky patří mezi základní stavební kameny každého většího programu. Slouží k řízení jeho toku. To znamená, že určují, za jakých předpokladů se vykonají určité příkazy.

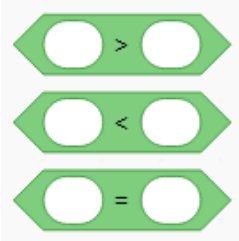
Ukažme si pro ilustraci několik jednoduchých příkladů: Systém nám umožní přihlášení, pouze pokud zadáme správné heslo. Kvízová aplikace nám přičte bod, odpovíme-li správně na otázku. Zvýší-li se teplota v lednici nad určitou mez, spustí se chlazení. Chceme-li zavřít textový dokument, ve kterém jsme prováděli úpravy, zeptá se nás program, zda chceme uložit změny, a podle naší volby provede příslušnou operaci.

4.1 Logické výrazy

Jako logický výraz označujeme tvrzení, o jehož pravdivosti lze jednoznačně rozhodnout. Pomocí logických výrazů můžeme podmiňovat vykonání některých příkazů – tyto příkazy se vykonají, pouze pokud je logický výraz splněný (pravdivý). Každý logický výraz v programu je při běhu programu vyhodnocen – výsledkem vyhodnocení je buď *true* (pravda, splněno), nebo *false* (nepravda, nesplněno). Součástí logického výrazu je obvykle proměnná, na jejíž hodnotu klademe určité podmínky.

4.1.1 Logické výrazy v jazyce Scratch

Logické výrazy jsou ve Scratchi poměrně intuitivní. Pro porovnání čísel máme možnost použít tři operátory – *menší*, *rovno*, *větší*. K porovnání identičnosti textových hodnot používáme operátor *rovno* (tentýž operátor, který slouží k porovnání čísel).

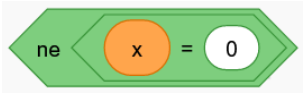


Pokud potřebujeme podmínku negovat (tj. změnit na opačnou), využijeme operátor *ne*. Uveďme si příklady několika podmínek:

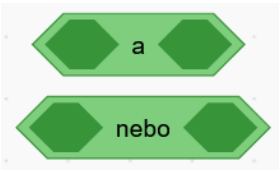
- podmínka pro zjištění, zda teplota klesla pod bod mrazu:

- podmínka pro zjištění, zda je uživatel starší 18 let:

- podmínka pro otestování, zda se proměnná nerovná nule:



Podmínky je možné spojovat také použitím operátorů *a*, *nebo*. Operátor *a* testuje, zda platí obě v něm zapsané podmínky současně. Operátor *nebo* testuje, zda platí alespoň jedna z podmínek, které jsou v něm zapsané.



Chceme-li pro porovnání čísel použít podmínky *menší nebo rovno* nebo *větší nebo rovno*, musíme příslušné dva bloky spojit pomocí operátoru *a*. [26]

4.1.2 Logické výrazy v jazyce C#

Níže uvedená tabulka [8] představuje přehled nejčastěji používaných logických výrazů pro číselné hodnoty:

Výraz	Význam
`x == y`	*x* je rovno *y*
`x < y`	*x* je menší než *y*
`x <= y`	*x* je menší nebo rovno *y*
`x > y`	*x* je větší než *y*
`x >= y`	*x* je větší nebo rovno *y*
`x != y`	*x* není rovno *y*

Všimněme si, že v logickém výrazu pro testování rovnosti se používají dvě rovná se. Je tomu tak proto, aby se logický výraz odlišil od přiřazení hodnoty do proměnné. V jazyce C# je možné, na rozdíl od jazyka Scratch, zapsat podmínku *menší nebo rovno* nebo *větší nebo rovno* pomocí jediného logického výrazu.

Pro porovnání shodnosti textových řetězců použijeme dvě rovná se, stejně jako při zjišťování rovnosti čísel. Pro textové řetězce je možné použít i operátor `!=` (*není rovno*).

Podmínky v C# spojujeme tak, že je vypíšeme za sebe a vložíme mezi ně operátor pro spojení. Chceme-li testovat, zda platí obě podmínky současně, použijeme operátor `&&` (*a zároveň*). Chceme-li testovat, zda platí alespoň jedna z uvedených podmínek, použijeme operátor `||` (*nebo*).

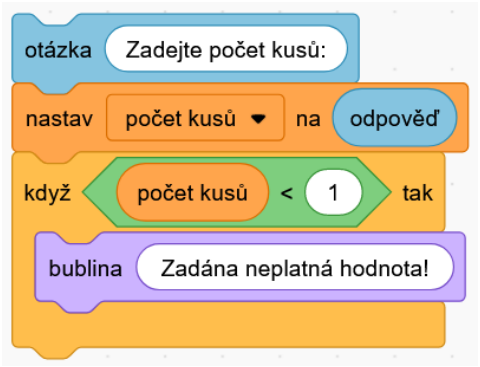
4.2 Podmínky

4.2.1 Podmínky v jazyce Scratch

Ve Scratchi k zápisu podmínky slouží blok *když (podmínka) tak…* z kategorie *Ovládání*. Místo pro podmínku má šestiúhelníkový tvar a vkládáme do něj odpovídající blok (logický výraz) z kategorie *Operátory*. Bloky vložené dovnitř bloku *když (podmínka) tak…* se vykonají, jen pokud je splněna podmínka (logický výraz je vyhodnocen jako pravdivý).

Využití podmínky si představíme na programu, jehož obdoba by mohla být využita například v e-shopu ke kontrole správnosti vyplnění políčka „počet objednaných kusů“.

Příklad: Vytvořte program, který po kliknutí na vlaječku načte číslo a v případě zadání záporného čísla nebo nuly uživatele upozorní, že zadal neplatnou hodnotu.



Připomeňme si, že scénář spustíme kliknutím na jeho první blok. Můžeme tak otestovat jeho funkčnost.

K načtení čísla program používá blok *otázka*. Načtená hodnota (*odpověď*) se uloží do proměnné *počet kusů*. Podmínkou *když (podmínka) tak…* se ověřuje, jestli je zadané číslo záporné nebo nula. Pokud ano, vypíše se hláška pomocí bloku *bublina*. Pro jednoduchost předpokládáme, že uživatel zadá celé číslo (ne desetinné).

V takto krátkém programu není uložení načtené hodnoty do proměnné nezbytné (v podmínce bychom mohli porovnávat přímo *odpověď*). Uložení do proměnné však zvyšuje přehlednost programu a při načítání více hodnot se mu stejně nevyhneme.

Chceme-li, aby se některé bloky vykonaly i při nesplnění podmínky, použijeme blok *když (podmínka) tak… / jinak…* Jeho použití si ukážeme na ověření plnoletosti. [26]

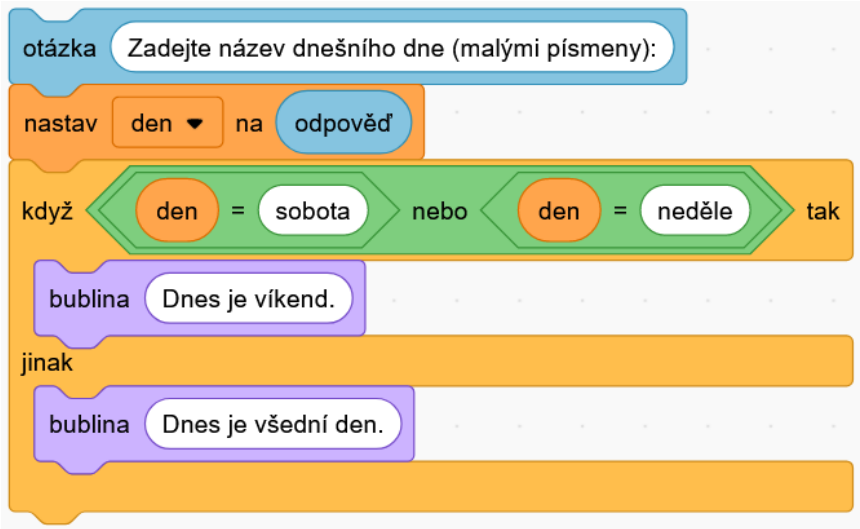
Příklad: Vytvořte program, který se uživatele po kliknutí na vlaječku zeptá na jeho věk a podle zadané odpovědi rozhodne, zda je uživatel plnoletý (tedy zda má 18 nebo více let).

Řešení je obdobné jako u předchozího příkladu s tím rozdílem, že používáme blok *když (podmínka) tak… / jinak…* Jsou uvedeny dva možné scénáře, které řeší tento příklad. Opět předpokládáme, že uživatel zadá věk jako celé číslo. Proto má v prvním případě podmínka tvar *věk > 17*. Pokud bychom podmínku zapsali jako *věk > 18*, program by v případě zadání věku 18 let vypsal, že uživatel zatím není plnoletý (což by nebylo správně).

Uvědomte si, že v programování mnoho problémů nemá jediný způsob řešení. Mějte to na paměti zejména při kontrole a porovnávání svých nápadů a vzorových řešení – i když se vaše řešení mírně liší, nemusí to okamžitě znamenat, že je špatné.

Poslední příklad zaměřený na podmínky, který si představíme ve Scratchi, bude popisovat skládání podmínek pomocí operátoru *nebo*.

Příklad: Vytvořte program, který si nechá zadat název dnešního dne a poté vyhodnotí, jestli je dnes všední den, nebo víkend.



Program po zadání údajů využívá skládání podmínek pomocí operátoru *nebo*. Aby byl scénář co nejkratší, ověří program nejprve, zda je sobota nebo neděle (tj. víkend). V opačném případě je všední den. Program předpokládá zadání správného údaje (tj. že uživatel zadá skutečně název dne).

4.2.2 Podmínky v jazyce C#

V jazyce C# zapisujeme podmínky pomocí příkazu *if*, za nímž následuje logický výraz a blok složených závorek. Je-li logický výraz vyhodnocen jako pravdivý, provedou se příkazy uvnitř bloku složených závorek. Pokud logický výraz jako pravdivý vyhodnocen není, blok složených závorek se přeskočí. [22]

Podmínky si v C# představíme na stejných modelových příkladech jako ve Scratchi. Budete tak moci porovnat zápis podmínek v obou jazycích. Jako první je uveden příklad, který modeluje kontrolu zadání počtu kusů v e-shopu.

Příklad: Vytvořte program, který nechá uživatele zadat přirozené číslo. Pokud uživatel zadá záporné číslo nebo nulu, program ho upozorní, že zadal neplatnou hodnotu.

```
static void Main()
{
    Console.Write("Zadejte počet kusů: ");
    int pocet = int.Parse(Console.ReadLine());
    if (pocet <= 0)
    {
        Console.WriteLine("Zadána neplatná hodnota!");
    }
}
```

Program nás nejprve nechá zadat číslo, které uloží do proměnné *pocet* datového typu *int*. Následuje podmínka, která je splněna, pokud je v proměnné *pocet* uloženo záporné číslo nebo nula. V tomto případě je vypsána hláška upozorňující uživatele na zadání neplatné hodnoty. Uvedený zápis podmínky odpovídá bloku *když (podmínka) tak…* ve Scratchi.

Všimněme si, že řádek s klíčovým slovem *if* a logickým výrazem se neukončuje středníkem – protože za tímto řádkem následuje blok složených závorek.

Chceme-li, aby se určitý příkaz vykonal při splnění podmínky a jiný se vykonal při nesplnění podmínky, použijeme klíčové slovo *else* (obdoba bloku *když (podmínka) tak… / jinak…* ve Scratchi). [22]

Příklad: Vytvořte program, který se uživatele zeptá na jeho věk a podle odpovědi rozhodne, jestli je uživatel plnoletý.

```
static void Main()
{
    Console.Write("Zadejte svůj věk: ");
    int vek = int.Parse(Console.ReadLine());
    if (vek < 18)
    {
        Console.WriteLine("Zatím nejste plnoletý/plnoletá.");
    }
    else
    {
        Console.WriteLine("Už jste plnoletý/plnoletá.");
    }
}
```

Program se nás nejprve dotáže na věk a naši odpověď uloží do proměnné *vek* datového typu *int*. Na základě hodnoty v této proměnné je pak vypsána hláška – pokud je věk uživatele nižší než 18 let, vypíše program, že není plnoletý. V opačném případě vypíše, že uživatel plnoletý je.

Pokud je v bloku složených závorek za příkazem *if* nebo *else* pouze jeden příkaz, není zápis složených závorek povinný (za *if* nebo *else* se středník nepíše ani v tomto případě). V ukázkách v tomto textu budou složené závorky ale uváděné ve všech případech, kvůli maximální přehlednosti.

Další příklad nám ukazuje, že logický výraz v podmínce může testovat i shodnost textového řetězce. [13]

Příklad: Vytvořte program, který uživateli položí kvízovou otázku a vyhodnotí jeho odpověď. V případě správné odpovědi zvýší počet bodů o 1.

```
static void Main()
{
    int pocetBodu = 0;

    Console.WriteLine("Jaké je hlavní město Francie?");
    string odpoved = Console.ReadLine();
    if (odpoved == "Paříž")
    {
        Console.WriteLine("Správně!");
        pocetBodu++;
    }
    else
    {
        Console.WriteLine("Chyba!");
    }
}
```

Na začátku programu je založena proměnná *pocetBodu* datového typu *int* a je do ní uložena hodnota 0 (tato proměnná má funkci počítadla získaných bodů).
Poté program vypíše otázku, načte zadanou odpověď a uloží ji do proměnné *odpoved* datového typu *string*. Následuje podmínka, ve které program porovnává zadanou hodnotu se správnou odpovědí. Je-li zadaná hodnota stejná jako správná odpověď, program vyhodnotí, že jsme odpověděli správně, a zvýší hodnotu v proměnné *pocetBodu* o 1 (pomocí příkazu `pocetBodu++`). V opačném případě program vyhodnotí naši odpověď jako chybnou.
Zkuste si kvíz samostatně rozšířit přidáním dalších otázek a jejich vyhodnocení. Na závěr vypište celkový počet bodů, které uživatel získal.

Často narazíme na situace, v nichž je třeba použít rozvětvení na více než dvě možnosti. Ukažme si, jak to lze provést pomocí vnořování podmínek.

Příklad: Vytvořte program, pomocí kterého si zákazník dokončující v e-shopu objednávku zvolí způsob platby. Program se zákazníka nejprve zeptá, zda je majitelem bankovního účtu. V případě kladné odpovědi nabídne platbu převodem nebo kartou. V případě záporné odpovědi nabídne platbu složenkou nebo hotově při převzetí. [8]

```
static void Main()
{
    Console.WriteLine("Jste majitelem bankovního účtu? (ano/ne)");
    string odpoved = Console.ReadLine();
    if (odpoved == "ano")
    {
        Console.Write("Pro platbu převodem zadejte 1, pro platbu
                       kartou zadejte 2: ");
        int volba = int.Parse(Console.ReadLine());
        if(volba == 1)
        {
            Console.WriteLine("Vybrali jste si platbu převodem.");
        }
        else
        {
            Console.WriteLine("Vybrali jste si platbu kartou.");
        }
    }
    else
    {
        Console.Write("Pro platbu složenkou zadejte 1, pro platbu
                       v hotovosti při převzetí zadejte 2: ");
        int volba = int.Parse(Console.ReadLine());
        if(volba == 1)
        {
            Console.WriteLine("Vybrali jste si platbu složenkou.");
        }
        else
        {
            Console.WriteLine("Vybrali jste si platbu v hotovosti
             při převzetí.");
        }
    }
}
```

Na tomto příkladu vidíme, že uvnitř bloku složených závorek následujícího za jednou podmínkou může následovat další podmínka. Tímto způsobem může být vnořený libovolný počet podmínek. Je ale třeba brát v potaz, že opakované vnořování podmínek snižuje přehlednost kódu. Vypisovaný text je rozdělen do dvou řádků pouze kvůli nedostatku místa na řádku (v programu rozdělení není třeba provádět).

Jinou možností je použít příkaz *else if*, kterým můžeme vytvořit více možností rozvětvení. Použití tohoto příkazu ilustruje následující příklad.

Příklad: Vytvořte program, který po zadání teploty vody vypíše její skupenství. [13]

```
static void Main()
{
    Console.Write("Zadejte teplotu vody: ");
    int teplota = int.Parse(Console.ReadLine());
    if (teplota < 0)
    {
        Console.WriteLine("Led");
    }
    else if (teplota < 100)
    {
        Console.WriteLine("Kapalina");
    }
    else
    {
        Console.WriteLine("Pára");
    }
}
```

Program se nás zeptá na teplotu vody a zadanou hodnotu uloží do proměnné *teplota* datového typu *int*. Potom prochází jednotlivé větve podmínky. Ve chvíli, kdy program narazí na splněnou podmínku, provede příkaz uvnitř této podmínky (výpis skupenství) a v procházení dalších větví už nepokračuje.

Při využití *else if* musíme pamatovat na to, že *if* a všechny bezprostředně za ním následující bloky *else if* nebo *else* tvoří spolu jednu část. Jakmile program narazí na splněnou podmínku, vykoná jí odpovídající kód a další příkazy *else if* nebo *else* přeskočí (vůbec je nebude vyhodnocovat). Pokud řešíme dvě situace, které se vzájemně nevylučují, musíme pro každou z nich vytvořit samostatnou podmínku začínající příkazem *if*.

Příklad: Vytvořte program, který uživatele nechá zadat číslo a následně o zadaném čísle rozhodne, jestli je liché a jestli je dělitelné pěti.

```
static void Main()
{
    Console.Write("Zadejte číslo: ");
    int cislo = int.Parse(Console.ReadLine());
    if(cislo % 2 != 0)
    {
        Console.WriteLine("Zadané číslo je liché.");
    }
    if(cislo % 5 == 0)
    {
        Console.WriteLine("Zadané číslo je dělitelné pěti.");
    }
}
```

Dvě situace, které v programu testujeme, se vzájemně nevylučují, a proto musíme použít dvě samostatné podmínky. Pokud bychom u druhé podmínky místo *if* uvedli *else if*, pak by např. v případě čísla 15 program vypsal pouze informaci, že je číslo liché, ale informaci o tom, že je dělitelné pěti, by nevypsal (vyhodnocením této situace by se vůbec nezabýval).

Na závěr si představíme dva příklady na skládání podmínek. V prvním z nich využijeme operátor `&&` s významem „a zároveň“.

Příklad: Restaurace má otevřeno od 10 do 20 hodin. Vytvořte program, který po zadání aktuálního času vypíše, zda má restaurace otevřeno, nebo zavřeno. [8]

```
static void Main()
{
    Console.Write("Zadejte údaj hh z aktuálního času ve tvaru hh:mm: ");
    int hodiny = int.Parse(Console.ReadLine());
    if(hodiny >= 10 && hodiny < 20)
    {
        Console.WriteLine("Restaurace má otevřeno.");
    }
    else
    {
        Console.WriteLine("Restaurace má zavřeno.");
    }
}
```

Po zadání aktuálního času program testuje pomocí operátoru *a zároveň*, zda aktuální čas patří do doby, kdy má restaurace otevřeno. Podle toho vypíše odpovídající hlášku.

Druhý příklad představuje využití operátoru `||` s významem „nebo“.

Příklad: Vytvořte program, který si nechá zadat název dnešního dne a poté vyhodnotí, jestli je dnes všední den, nebo víkend.

```
static void Main()
{
    Console.Write("Zadejte název dnešního dne (malými písmeny): ");
    string den = Console.ReadLine();
    if(den == "sobota" || den == "neděle")
    {
        Console.WriteLine("Dnes je víkend.");
    }
    else
    {
        Console.WriteLine("Dnes je všední den.");
    }
}
```

Program po zadání údajů využívá skládání podmínek pomocí operátoru *nebo*. Abychom minimalizovali kód, ověřuje program nejprve, zda je sobota nebo neděle (tj. víkend).

V opačném případě je všední den. Program předpokládá zadání správného údaje (tj. že uživatel zadá skutečně název dne).

5 CYKLY

Další situací, s níž se v programování velmi často setkáváme, je opakování určitého kroku. Při filtrování zaměstnanců v databázi podle počtu odpracovaných let porovnáváme u každého zaměstnance počet odpracovaných let s požadovaným číslem. Při hraní piškvorek program po tahu hráče prochází okolní políčka jedno po druhém, aby zjistil, jestli hráč vytvořil v některém směru pět stejných symbolů za sebou. Při šifrování zprávy Caesarovou šifrou zprávu procházíme a šifrujeme písmeno po písmenu. Epizoda počítačové hry probíhá, dokud neztratíme všechny životy. Všechny tyto situace se řeší pomocí cyklů.

V oblasti cyklů jsou mezi jazyky Scratch a C# výraznější rozdíly. Začneme proto vysvětlením cyklů v C# a poté si uvedené informace porovnáme se situací ve Scratchi.

5.1 Cykly v jazyce C#

V jazyce C# existují tři typy cyklů: cyklus s podmínkou na začátku, cyklus s podmínkou na konci a cyklus s řídicí proměnnou. [8]

Proměnné vytvořené uvnitř cyklu existují pouze po dobu jeho provádění. Chceme-li s proměnnou pracovat i po opuštění cyklu, musíme ji založit před tímto cyklem (vytvoříme buď prázdnou proměnnou, nebo do ní uložíme vhodnou startovací hodnotu).

5.1.1 Cyklus s podmínkou na začátku

Tento cyklus se opakuje tak dlouho, dokud je splněna podmínka uvedená v jeho zápisu. Podmínka se ověřuje před prvním a před každým dalším průchodem cyklem. Někdy bývá tento cyklus označován jako cyklus *while*. Představíme si ho na příkladu. [22]

Příklad: Vytvořte program, který vyzve uživatele k nastavení nového hesla s délkou nejméně 8 znaků. Při zadání kratšího hesla bude výzvu opakovat tak dlouho, než uživatel vytvoří heslo s požadovanou délkou.

```
static void Main()
{
    string heslo = "";
    while(heslo.Length < 8)
    {
        Console.WriteLine("Nastavte si heslo dlouhé alespoň 8 znaků: ");
        heslo = Console.ReadLine();
    }
    Console.WriteLine("Heslo bylo úspěšně nastaveno.");
}
```

Proměnnou *heslo* datového typu *string* si vytvoříme před vstupem do cyklu a uložíme do ní jako startovací hodnotu prázdný řetězec (pokud bychom do ní žádnou hodnotu neuložili, došlo by v programu k chybě při vyhodnocování délky tohoto řetězce).
Při zápisu cyklu uvedeme podmínku pro délku řetězce. Program nás bude k zadání hesla vyzývat tak dlouho, dokud námi vytvořené heslo nebude mít požadovanou délku. Poté nás program informuje o úspěšném nastavení hesla.

Cyklus s podmínkou na začátku můžeme použít také k řešení některých matematických úloh. Uplatní se zejména u úloh, ve kterých pracujeme s posloupnostmi, jak ilustruje následující příklad.

Příklad: Jitka začala spořit. První vklad 200 Kč provedla v lednu 2020. Každý další měsíc vkládala o 20 Kč více než předchozí. Zjistěte, ve kterém měsíci překročila její naspořená částka 12 000 Kč.

```
static void Main()
{
    //zápis počátečních údajů
    int vklad = 200;
    int prirustek = 20;
    int limit = 12000;
    int mesic = 1;
    int rok = 2020;

    //výpočet
    int nasporenaCastka = vklad;
    while (nasporenaCastka  < limit)
    {
        vklad = vklad + prirustek;
        nasporenaCastka = nasporenaCastka + vklad;
        mesic++;
        if(mesic == 13)
        {
            mesic = 1;
            rok++;
        }
    }

    //výpis výsledku
    Console.WriteLine("Hranice {0} bude překročena v {1}. měsíci
                        roku {2}", limit, mesic, rok);
}
```

Po spuštění programu zjistíme, že Jitka částku 12 000 Kč překročila v březnu 2022.
Kód je rozdělen do tří částí. V první části zadáváme potřebné údaje. Všechny údaje ukládáme do proměnných datového typu *int* (celé číslo).

V druhé části si definujeme proměnnou *nasporenaCastka*, do které na počátku uložíme první vklad a která se bude s každým dalším měsícem zvyšovat. Následuje cyklus *while*, který se opakuje tak dlouho, než naspořená částka překročí stanovený limit.
Při každém opakování cyklu se nejprve navýší vklad, poté je tento vklad přičten k naspořené částce a počítadlo měsíců se zvýší o 1. Dosáhne-li počítadlo měsíců 13 (tj. byl překročen konec roku), nastaví se počítadlo měsíců znovu na 1 a zvýší se o 1 počítadlo let.
Ve třetí části dochází k výpisu výsledku. Proměnné jsou do vypisovaného řetězce vložené pomocí zástupných značek. Vypisovaný text je rozdělen do dvou řádků pouze kvůli nedostatku místa na řádku (v programu rozdělení není třeba provádět).
Vstupní údaje jsou v tomto programu vloženy přímo do kódu, ale nebylo by obtížné program přepracovat tak, aby potřebné údaje od uživatele načetl v konzoli.

5.1.2 Cyklus s podmínkou na konci

Tento cyklus je velmi podobný právě vysvětlenému cyklu s podmínkou na začátku. Jediná odlišnost je v tom, že cyklus s podmínkou na konci se vždy provede alespoň jednou. Je tomu tak proto, že podmínka se ověřuje až na konci cyklu, poté, co jím projdeme. Někdy bývá tento cyklus označován jako cyklus *do-while*. Jeho funkci si představíme na již uvedeném příkladu, abychom obě řešení porovnali. [22]

Příklad: Vytvořte program, který vyzve uživatele k nastavení nového hesla s délkou nejméně 8 znaků. Při zadání kratšího hesla bude výzvu opakovat tak dlouho, než uživatel vytvoří heslo s požadovanou délkou.

```
static void Main()
{
    string heslo;
    do
    {
        Console.WriteLine("Nastavte si heslo dlouhé alespoň 8 znaků: ");
        heslo = Console.ReadLine();
    } while(heslo.Length < 8);
    Console.WriteLine("Heslo bylo úspěšně nastaveno.");
}
```

Všimněte si, že při použití cyklu s podmínkou na konci není třeba do proměnné *heslo* při jejím vytváření zadat žádnou hodnotu (ani prázdný řetězec). Cyklus se vždy provede alespoň jednou, a proto bude při vyhodnocování podmínky v proměnné *heslo* vždy nějaká hodnota uložená. Jinak se princip řešení a úloha cyklu nijak neliší od řešení, při kterém jsme využívali cyklus s podmínkou na začátku.

Cyklus s podmínkou na konci je výhodné využít, pokud chceme, aby se program prováděl ve smyčce (tzn. že po svém proběhnutí se nás program dotáže, zda ho chceme ukončit, nebo znovu spustit od začátku).

Příklad: Vytvořte program, který bude provádět hody kostkou. Po spuštění program vygeneruje náhodné číslo od 1 do 6. Zároveň se zeptá, zda chceme hod opakovat (stisknutím klávesy „o“), nebo program ukončit (stisknutím libovolné jiné klávesy). [8]

```
static void Main()
{
    Random generator = new Random();
    char opakovat;
    do
    {
        Console.Clear();
        Console.WriteLine(generator.Next(1,7));
        Console.WriteLine("klávesa o - další hod, jiná klávesa - konec");
        opakovat = Console.ReadKey().KeyChar;
    } while (opakovat == 'o');
}
```

Nejprve přidáme do programu generátor náhodných čísel a založíme proměnnou *opakovat* datového typu *char*. Uvnitř cyklu si všimněte příkazu `Console.Clear()`, který vymaže konzoli (tzn. smaže výpis předchozího hodu).
Následuje generování náhodného čísla, které je hned poté programem vypsáno (lze použít úsporný zápis bez uložení čísla do proměnné, protože číslo se dál v programu nevyužívá).
Program pak čeká na stisknutí klávesy. Hodnota stisknuté klávesy se uloží do proměnné *opakovat* a podle této hodnoty se rozhoduje, zda dojde k novému hodu kostkou, nebo k ukončení programu. Proměnnou je třeba založit před cyklem (pokud ji založíme až uvnitř cyklu, proměnná bude existovat jen v něm a při vyhodnocování podmínky dojde k chybě).

5.1.3 Cyklus s řídicí proměnnou

Tento cyklus se někdy označuje také jako cyklus *for* a je užitečný zejména v případech, kdy potřebujeme provést předem známý počet opakování cyklu. Seznámíme se s ním pomocí vzorové ukázky. [8]

```
static void Main()
{
    for(int i = 0; i < 5; i++)
    {
        Console.WriteLine(i);
    }
}
```

Pokud kód spustíme, vypíšou se nám čísla od 0 do 4 (včetně). Proměnnou *i* označujeme jako řídicí proměnnou cyklu. Zápis cyklu se skládá ze tří částí:

- `int i = 0` je výraz pro založení a nastavení počáteční hodnoty řídicí proměnné.
- `i < 5` je podmínka, při jejímž splnění se bude cyklus opakovat (ve chvíli, kdy se podmínka vyhodnotí jako nesplněná, opakování skončí).
- `i++` popisuje změnu řídicí proměnné v každém kroku cyklu (je možné měnit *i* sestupným směrem pomocí výrazu `i--` nebo iterovat (posouvat se) o krok větší než 1 např. pomocí výrazu `i=i+2`).

Příklad: Vytvořte program, který vypíše prvních deset kladných násobků sedmi.

```
static void Main()
{
    for(int i = 0; i < 10; i++)
    {
        Console.WriteLine(7*(i+1));
    }
}
```

Program je přímou analogií vzorové ukázky.

Ukažme si, jak zapsat cyklus *for* tak, aby počet jeho opakování (tj. podmínku pro ukončení) určoval uživatel zadáním požadovaného čísla.

Příklad: Vytvořte program, který se uživatele zeptá, kolik „plusů“ má vypsat, a po zadání hodnoty tento počet „plusů“ vypíše. [9]

```
static void Main()
{
    Console.WriteLine("Kolik + si přejete vypsat?");
    int pocet = int.Parse(Console.ReadLine());
    for(int i = 0; i < pocet; i++)
    {
        Console.Write("+");
    }
}
```

Nejprve necháme uživatele zadat, kolik „plusů“ si přeje vypsat. Tuto hodnotu načteme do proměnné *pocet* datového typu *int* a poté tuto proměnnou použijeme v podmínce pro ukončení opakování cyklu, v němž se nachází příkaz pro vypsání znaku plus.

Podobně bychom postupovali, pokud bychom chtěli vytvořit kvíz, ve kterém si uživatel sám volí počet vygenerovaných otázek.

Následující příklad nám nejen ukazuje další situaci, kdy počet opakování cyklu závisí na hodnotě zadané uživatelem, ale také nám opět připomíná, že pokud chceme používat proměnnou i po ukončení cyklu, musíme ji vytvořit před začátkem cyklu.

Příklad: Vytvořte program analyzující úhrn srážek. Program se nejprve uživatele zeptá, kolik hodnot bude zadávat, a poté nechá uživatele tato čísla zadat. Nakonec zobrazí celkový úhrn srážek a průměrný denní úhrn srážek za dané období zaokrouhlený na setiny. [8]

```
static void Main()
{
    Console.WriteLine("Program pro analýzu srážek v daném období");
    Console.Write("Kolik hodnot budete zadávat? ");
    int pocet = int.Parse(Console.ReadLine());
    double soucet = 0;
    Console.WriteLine("Zadejte úhrny srážek v mm v daných dnech:");
    for(int i = 0; i < pocet; i++)
    {
        Console.Write("{0}. den: ", i+1);
        double uhrn = double.Parse(Console.ReadLine());
        soucet = soucet + uhrn;
    }
    double prumer = Math.Round(soucet/pocet, 2);
    Console.WriteLine("Celkový úhrn srážek: " + soucet);
    Console.WriteLine("Průměrný denní úhrn srážek: " + prumer);
}
```

Program se nejprve zeptá na počet zadávaných hodnot a tento údaj uloží do proměnné *pocet* datového typu *int*. Poté je vytvořena proměnná *soucet* datového typu *double*. Do této proměnné se budou ukládat součty úhrnů srážek v jednotlivých dnech.

Následuje cyklus, který nás vybízí k zadání úhrnu srážek v každém z dnů. Zadávané hodnoty mají datový typ *double* (desetinné číslo). Proměnná *soucet* je po každém zadání hodnoty o tuto hodnotu navýšena.

Po ukončení cyklu (zadání všech hodnot) program spočítá průměrný denní úhrn tak, že součet všech úhrnů vydělí počtem dnů (a výsledek dělení zaokrouhlí na setiny). Na závěr vypíše celkový úhrn srážek za dané období a průměrný denní úhrn srážek.

Cyklus *for* můžeme využít také k řešení některých matematických úloh numerickým přístupem. Následující příklad ukazuje možnost využití příkazů *return* a *break*. Příkaz *return* ukončí běh celého programu a příkaz *break* ukončí běh cyklu, v němž se tento příkaz nachází.

Příklad: Vytvořte program, který nechá uživatele zadat přirozené číslo větší než 1 a rozhodne, zda se jedná o prvočíslo.

```
static void Main()
{
    //načtení čísla
    Console.Write("Zadejte přirozené číslo: ");
    int cislo = int.Parse(Console.ReadLine());
    bool prvocislo = true;
    if(cislo <= 1)
    {
        Console.WriteLine("Zadané číslo musí být větší než 1!");
        return;
    }

    //ověření, zda je zadané číslo prvočíslo
    for(int i = 2; i < cislo/2; i++)
    {
        if(cislo % i == 0)
        {
            prvocislo = false;
            break;
        }
    }

    //výpis výsledku
    if(prvocislo)
    {
        Console.WriteLine("Zadané číslo je prvočíslo.");
    }
    else
    {
        Console.WriteLine("Zadané číslo je složené číslo.");
    }
}
```

Program je rozdělen do tří částí – každá z nich je nadepsána komentářem. Úlohou první části je načtení čísla od uživatele. Je-li zadané číslo menší než 2, program nás na tuto skutečnost upozorní a ukončí se pomocí příkazu *return* (šlo by použít i konstrukci *if/else*, ale vedlo by to ke zkomplikování kódu). V první části je vytvořena také proměnná *prvocislo* datového typu *bool*, ve které se uchovává informace o tom, zda je zadané číslo prvočíslo. Proměnná *prvocislo* je ve výchozím stavu nastavená na pravdivý stav, ve chvíli objevení jiného dělitele, než je 1 nebo číslo samotné, se proměnná změní na nepravdivý stav.

Ve druhé části program postupně prochází čísla větší než 2 a těmito čísly zkouší dělit zadané číslo. Je-li zadané číslo dělitelné některým ze zkoušených čísel (tj. zbytek po dělení je nula), nejedná se o prvočíslo a nemá smysl zdržovat se zkoušením dalších čísel – cyklus je opuštěn příkazem *break*. V cyklu program zkouší dělit čísly od 2 po nejvýše polovinu zadaného čísla

(tak je nastavena podmínka pro ukončení cyklu), protože dělitel čísla nikdy není větší než jeho polovina (neuvažujeme-li jako dělitele čísla toto číslo samotné).
Úkolem třetí části je výpis výsledku. Všimněte si, že pokud je proměnná datového typu *bool*, můžeme do podmínky jako logický výraz zapsat pouze její jméno.

Cyklus *for* můžeme vložit dovnitř jiného cyklu *for*. Toto vnořování cyklů se využívá, pokud potřebujeme pracovat s dvojrozměrným objektem, například k vyhodnocování sítě při hře piškvorky.

Příklad: Vytvořte program, který se uživatele zeptá na výšku a šířku obrazce a podle zadaných hodnot vykreslí obrazec z „plusů“. Níže je vzor s čtyřmi řádky a sedmi sloupci. [9]

```
+ + + + + + +
+ + + + + + +
+ + + + + + +
+ + + + + + +
```

```
static void Main()
{
    Console.WriteLine("Zadejte výšku obrazce: ");
    int vyska = int.Parse(Console.ReadLine());
    Console.WriteLine("Zadejte šířku obrazce: ");
    int sirka = int.Parse(Console.ReadLine());
    for(int i = 0; i < vyska; i++)
    {
        for(int j = 0; j < sirka; j++)
        {
            Console.Write("+ ");
        }
        Console.WriteLine();
    }
}
```

Nejprve necháme uživatele zadat oba rozměry obrazce. Poté následuje cyklus s řídicí proměnnou *i*, který se stará o vypisování řádků. Uvnitř něj se nachází cyklus s řídicí proměnnou *j*, který se stará o výpis jednotlivých znaků v řádcích (pro lepší přehlednost vkládáme za každý vypsaný znak mezeru).
Uvnitř cyklu s řídicí proměnnou *j* používáme příkaz `Console.Write("+ ")`, protože tento cyklus vypisuje všechny znaky vždy na jeden řádek. Až za tímto cyklem následuje příkaz `Console.WriteLine()`, který po dokončení výpisu řádku nastaví kurzor na nový řádek.

5.2 Cykly v jazyce Scratch

Ve Scratchi existují dva cykly – *opakuj dokud nenastane* (obdoba cyklu s podmínkou na začátku) a cyklus *opakuj…krát* (obdoba cyklu s řídicí proměnnou). [26]

Založení proměnné ve Scratchi nikdy není součástí scénáře (proměnné zakládáme tlačítkem v kategorii *Proměnné* a každá proměnná musí být založena před prvním použitím). Ve Scratchi proto odpadá nutnost dbát na založení proměnných před začátkem cyklu jako v C# (i pokud vytvoříme proměnnou jen pro konkrétní postavu, může s ní tato postava pracovat z libovolného místa v každém svém scénáři).

5.2.1 Opakování s podmínkou

Cyklus *opakuj dokud nenastane* je analogií cyklu s podmínkou na začátku používaného v C# (analogie cyklu s podmínkou na konci se ve Scratchi nevyskytuje).

Abychom si C# a Scratch porovnali, vytvoříme si ve Scratchi program, s nímž jsme se seznámili v kapitole o cyklu s podmínkou na začátku. Podmínka pro ukončení je ve Scratchi formulována opačně než v C# (v C# opakujeme tak dlouho, dokud uvedená podmínka platí, zatímco ve Scratchi opakujeme do doby, než dojde ke splnění podmínky).

Příklad: Vytvořte program, který vyzve uživatele k nastavení nového hesla s délkou nejméně 8 znaků. Při zadání kratšího hesla bude výzvu opakovat tak dlouho, než uživatel vytvoří heslo s požadovanou délkou.

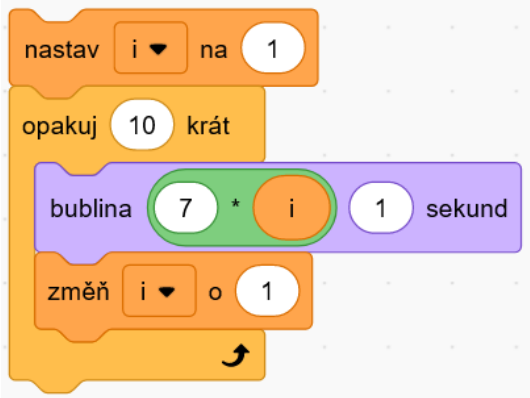
Po spuštění programu se proměnná *heslo* vyprázdní, aby v ní nezůstala uložená hodnota z minula. Následuje cyklus, v němž zadáváme nové heslo do doby, než je splněna podmínka pro jeho délku. Poté program vypíše hlášku o úspěšném nastavení hesla. Pro zvýraznění efektu vyhodnocení délky hesla je ve scénáři zařazen blok *čekej 1 sekund*.

Příklad zaměřený na spoření by vypadal analogicky. Můžete si ho zkusit ve Scratchi sestavit samostatně.

5.2.2 Opakování s daným počtem

Cyklus *opakuj …krát* je analogií cyklu s řídicí proměnnou. Hlavním rozdílem je, že ve Scratchi tento cyklus ve výchozím stavu řídicí proměnnou neobsahuje. Můžeme si ale vytvořit proměnnou, která bude její funkci plnit. Jak to provedeme, ukazuje následující příklad, obdoba příkladu již řešeného v C#.

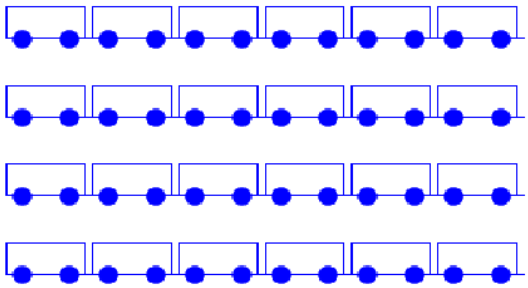
Příklad: Vytvořte program, který vypíše prvních deset kladných násobků sedmi.



Proměnná *i* zde má funkci řídicí proměnné. Po každém průchodu cyklem je její hodnota zvýšena o 1. Jednotlivé násobky jsou vypisovány postupně a jsou zobrazovány vždy na 1 sekundu. Bylo by možné si také založit proměnnou, do níž by program násobky postupně připisoval a na svém konci by tuto proměnnou vypsal (zobrazily by se tak všechny násobky současně, nikoliv jeden po druhém).

Pokud si chceme procvičit práci s dvourozměrným polem ve Scratchi, můžeme využít projekt Vlak ze studia iMYŠLENÍ (https://scratch.mit.edu/projects/268054172/). [7]

Příklad: Vlevo uvedený scénář (sestavený v projektu Vlak) vykreslí obrázek vpravo.



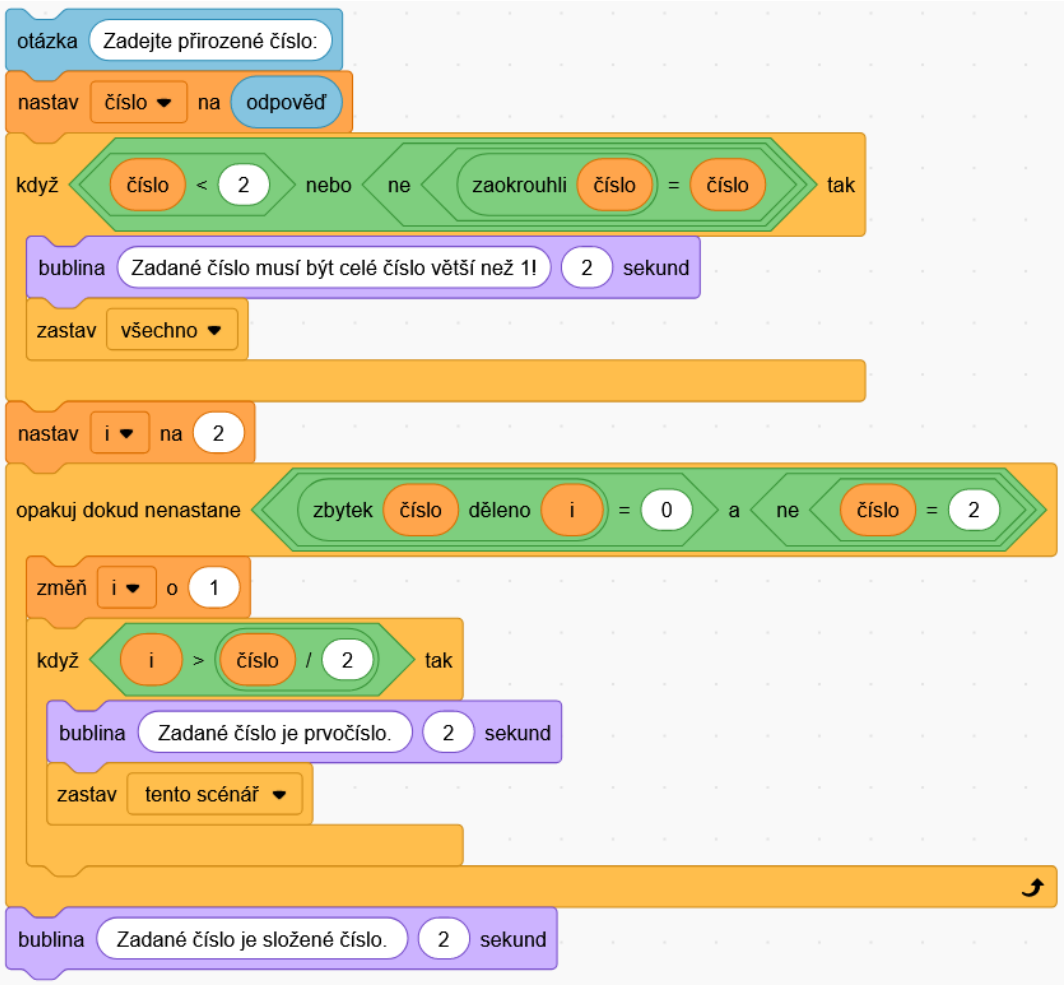
Vzhledem k tomu, že při vypisování textu pomocí bloků *bublina* nebo *myšlenka* Scratch automaticky zalamuje řádky podle délky zprávy, je práce s dvourozměrným polem ve Scratchi tímto limitovaná. Pro práci s dvourozměrným polem se ale výborně hodí úlohy na vykreslování obrazců, jak bylo ukázáno výše.

5.2.3 Zajímavost na závěr

Podívejme se ještě na program testující, zda je číslo prvočíslo. Zjistíme, že zmíněné dva cykly ve Scratchi nejsou přímou analogií cyklů v C#. Je tomu tak i díky tomu, že Scratch nemá blok, který by umožňoval opuštění aktuálně probíhajícího cyklu (obdoba příkazu *break*). Někdy proto může být výhodnější ve Scratchi použít cyklus *opakuj dokud nenastane* na místě, kde bychom v C# použili cyklus *for*.

Dodejme ještě, že blok pro ukončení aktuálně probíhajícího scénáře nebo všech scénářů (obdoba příkazu *return*) ve Scratchi naopak existuje.

Příklad: Vytvořte program, který nechá uživatele zadat přirozené číslo větší než 1 a rozhodne, zda se jedná o prvočíslo.



Program nechá uživatele zadat číslo a uloží ho do proměnné *číslo*. Logický výraz v bloku *když…tak…* se skládá ze dvou dílčích podmínek – testuje, jestli nebylo zadáno číslo menší než 2 nebo číslo desetinné (pouze pro celé číslo platí, že zaokrouhlení ho nezmění). Pokud dojde k jedné z jmenovaných situací, nemá smysl vyhodnocovat, zda se jedná o prvočíslo, program nás na tuto situaci upozorní a ukončí se.

Pokud bylo zadáno celé číslo větší než 1, dojde ke spuštění cyklu, který zkouší dělit zadané číslo postupně celými čísly *i* nabývajícími hodnoty od 2 (včetně) výše. Pokud zjistíme, že je zadané číslo dělitelné některým z čísel *i*, cyklus je ukončen s výsledkem, že se jedná o složené číslo.

Pokud číslo *i* dosáhlo hodnoty větší než polovina zadaného čísla, je zřejmé, že žádného dělitele tohoto čísla nenajdeme (kromě čísla samého). Program vypíše, že zadané číslo je prvočíslo, a ukončí se.

Logický výraz *nerovná se 2* je v podmínce pro opakování uveden kvůli číslu 2 – aby bylo správně vyhodnoceno jako prvočíslo.

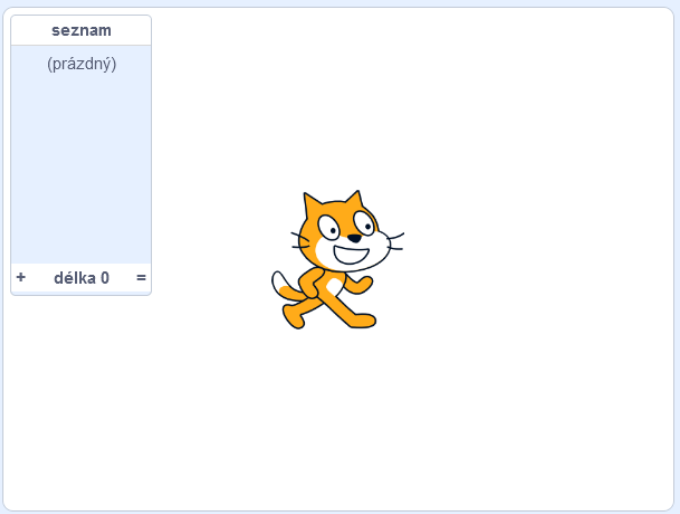
6 POLE A SEZNAM

Představte si, že chcete v programu uložit hodnoty srážek v jednotlivých dnech v průběhu týdne. Se současnými znalostmi byste museli pro každý den založit proměnnou. Tento problém je ale možné řešit mnohem efektivněji. K uložení mnoha hodnot stejného významu se používají pole a seznam.

6.1 Seznam v jazyce Scratch

Pokud chceme pracovat se seznamem, musíme ho nejprve vytvořit. To provedeme tak, že klikneme na tlačítko *Vytvoř seznam* v kategorii *Proměnné*. V okně, které se otevře, nastavíme název seznamu a potvrdíme. Seznamů můžeme samozřejmě vytvořit více, podobně jako proměnných.

Pokud se podíváme do okna aplikace, uvidíme v něm zobrazený zatím prázdný seznam. Prvky do seznamu přidáváme pomocí ikony + v levém dolním rohu seznamu v okně aplikace. Viditelnost seznamu v okně aplikace řídíme pomocí zaškrtávacího políčka u názvu seznamu v kategorii *Proměnné*.

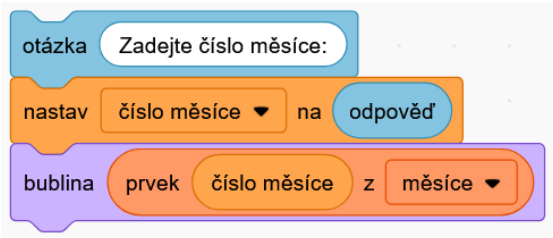


Po vytvoření seznamu se v kategorii *Proměnné* objeví bloky pro práci se seznamy. Najdeme zde bloky, kterými můžeme prvek do seznamu přidat, prvek z něj odebrat, vyhledat pořadí prvku v seznamu nebo zjistit, jestli seznam určitý prvek obsahuje. [26]

Seznamy se často využívají k práci s dny v týdnu nebo s měsíci, jak ukazuje následující příklad.

Příklad: Vytvořte program, který po zadání čísla měsíce vypíše jeho název (tzn. např. při zadání čísla 5 program vypíše „květen“).

Nejprve vytvoříme seznam *měsíce* a vložíme do něj všech 12 názvů měsíců. Poté z bloků sestavíme tento scénář:



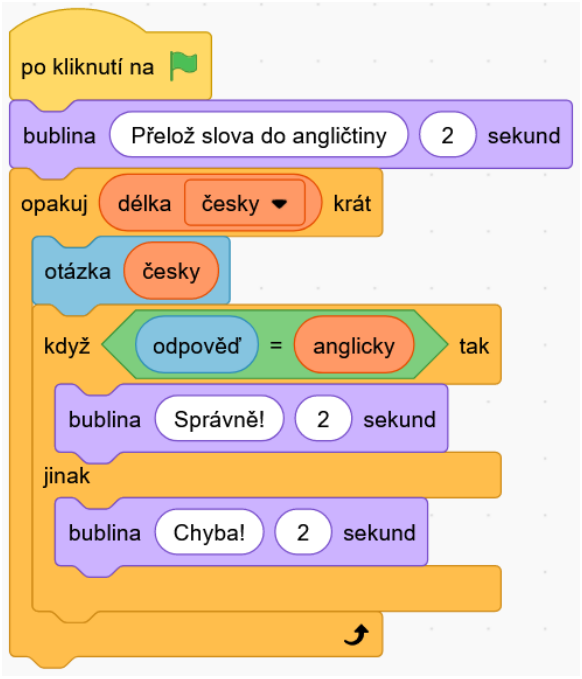
Pomocí bloku *otázka* nechá program uživatele zadat číslo, které uloží do proměnné *číslo měsíce*. Poté zobrazí název měsíce nacházející se na pozici určené zadaným číslem.

Seznamy nabízejí velké množnosti využití. Lze je použít například při tvorbě databáze osob, programu s funkcí žákovské knížky nebo aplikace, která bude testovat naše znalosti z určité oblasti.

Příklad: Vytvořte kvíz, který uživatele vyzkouší z překladu 5 českých slov do angličtiny.

Nejprve vytvoříme dva seznamy s názvy *česky* a *anglicky*. Vložíme 5 slov do seznamu *česky* a jejich překlady vložíme do seznamu *anglicky* tak, aby sobě odpovídající slova byla na stejných řádcích (na téže pozici od začátku seznamu) – viz ukázka vpravo.

Z bloků potom sestavíme uvedený scénář:



Před spuštěním kvízu zrušíme zaškrtnutí políček u názvů seznamů v kategorii *Proměnné*, aby uživatel neviděl správné odpovědi. Počet opakování cyklu se řídí délkou seznamu *česky*. Při každém průchodu cyklem je zobrazeno české slovo a program čeká na zadání překladu. Poté odpověď vyhodnotí a vyhodnocení vypíše.

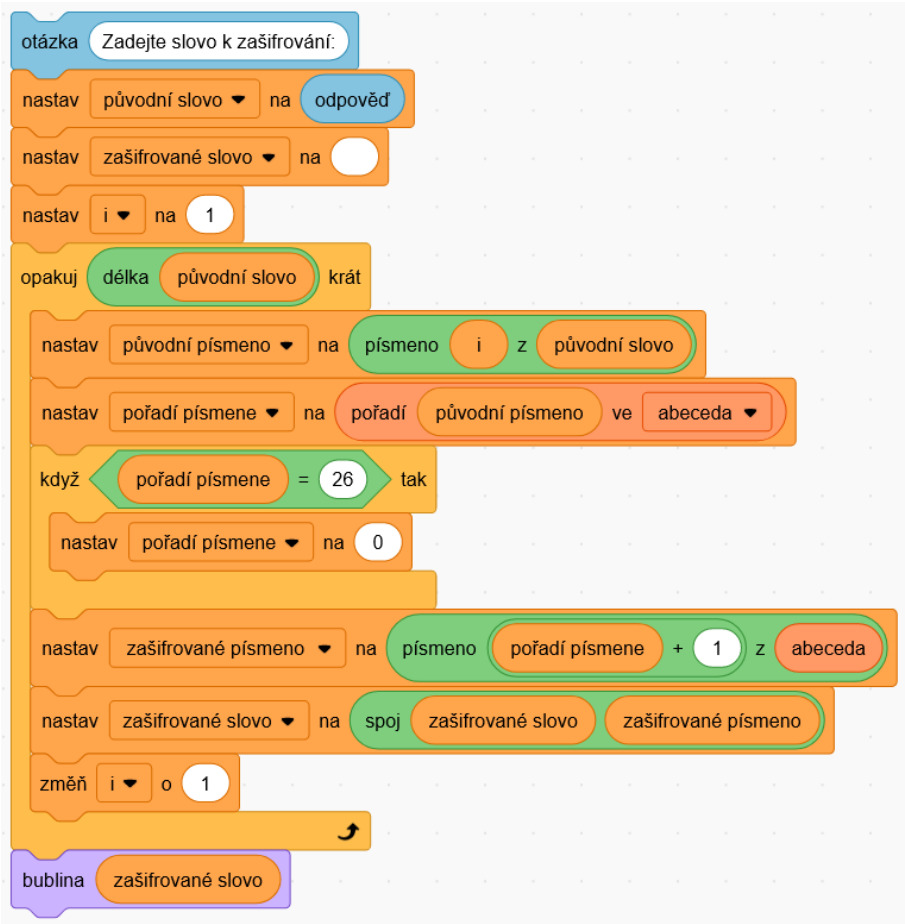
Pokud bychom chtěli program vylepšit, můžeme do něj přidat počítadlo správných a špatných odpovědí (proměnné *správné odpovědi* a *chyby*, které bychom ponechali zobrazené zaškrtnutím políček u jejich názvů v kategorii *Proměnné*).

Při vytváření kvízu (na jiné téma než překlady slov) je třeba vymýšlet otázky, na které lze odpovědět jednoznačně a jedním slovem.

Na textovou hodnotu uloženou v proměnné se můžeme dívat také jako na seznam složený z jednotlivých písmen. Tuto znalost využijeme, pokud chceme zašifrovat slovo.

Příklad: Vytvořte program, který zašifruje slovo posunem všech jeho znaků o jedno písmeno dopředu v anglické abecedě („z“ se posune na „a“). Program bude předpokládat zadání slova bez diakritiky (tj. pouze ze znaků anglické abecedy).

Nejprve vytvoříme seznam *abeceda* a vložíme do něj po řadě všech 26 písmen anglické abecedy. Pak sestavíme tento scénář:



Program nejprve nechá uživatele zadat slovo. Toto slovo uloží do proměnné *původní slovo*, proměnnou *zašifrované slovo* nastaví na prázdnou hodnotu a do iterační proměnné *i* uloží hodnotu 1. Následuje cyklus, který se opakuje tolikrát, kolik znaků má zadané slovo. V cyklu program bere postupně jednotlivá písmena slova, každé z nich najde v seznamu *abeceda* a písmeno nacházející se v tomto seznamu těsně za nalezeným písmenem přidá do proměnné *zašifrované slovo* (v případě písmene „z“ dojde k jeho změně na „a“ – toto řeší blok *když…tak*). Na závěr je programem zašifrované slovo zobrazeno.

6.2 Pole a seznam v jazyce C#

V jazyce C# je situace komplikovanější. Existují dva způsoby, jak můžeme uložit hromadně více dat stejného typu.

První možností je pole. Pole musí mít pevně definovanou velikost už při vytvoření a dále ji není možné měnit. Hodnotu na konkrétním místě můžeme kdykoliv v průběhu programu změnit, ale nelze ji odstranit (museli bychom přepsat všechny hodnoty následující za touto hodnotou na předchozí místo a poslední prvek pole nechat prázdný), podobně jako nelze mezi dvě hodnoty za běhu programu vložit třetí. Díky tomu program při práci s polem dokáže k uloženým hodnotám přistupovat velmi rychle. Pole má blíže ke strojovému zpracování kódu, v nižších jazycích (např. C) je často jedinou možností, jak ukládat hromadně více dat. [13]

Druhou možností je seznam (list), který nemá při spuštění programu pevně definovanou velikost a do něhož můžeme položky přidávat nebo je z něj mazat kdykoliv za běhu programu. I seznam přitom vnitřně pracuje s polem – při založení seznamu je vytvořeno pole s určitou kapacitou; pokud do seznamu vložíme více prvků, než je kapacita pole, program vytvoří nové, větší pole a všechny hodnoty z původního pole do něj překopíruje. Seznam můžeme tedy vnímat jako určitou nadstavbu nad pole, která nás odstiňuje od technických nepříjemností při práci s ním. Daní za tyto výhody je ale nižší rychlost přístupu k datům ve srovnání s polem.

Podobně jako proměnné i pole a seznamy musí mít v C# definovaný datový typ hodnot, které v nich budou uložené. Je třeba pamatovat na to, že pole i seznam indexují (počítají) prvky od 0 – tedy to, co považujeme za první prvek, je „ve skutečnosti“ nultý prvek. V tomto se C# liší od Scratche, který pro jednoduchost indexuje seznamy od 1. Pokud překročíme

hranice pole (tzn. přistoupíme k pozici s vyšším indexem, než je velikost pole), dojde v programu k chybě.

Jestliže chceme v programu pracovat se seznamem, je nutné uvést na začátek programu (pod příkaz `using System;`) příkaz `using System.Collections.Generic;` – bez použití tohoto příkazu by došlo v programu k chybě a nespustil by se.

Nejprve se seznámíme se základními příkazy, které se používají v souvislosti s polem: [11]

- vytvoření pole s danými hodnotami:

```
string barvy = { "černá", "bílá", "modrá", "žlutá" };
```

- vypsání hodnoty na dané pozici v poli:

```
Console.Write(barvy[2]);
```

- uložení hodnoty na danou pozici pole (příp. přepsání hodnoty na této pozici):

```
barvy[2] = "hnědá";
```

- zjištění délky pole:

```
int velikostPole = barvy.Length;
```

- vytvoření prázdného pole dané velikosti a naplnění pole daty zadanými do konzole:

```
int[] znamky = new int[10];
for(int i = 0; i < znamky.Length; i++)
{
    znamky[i] = int.Parse(Console.ReadLine());
}
```

Nyní si ukážeme, jak se stejné a některé další (ty, které umožňuje seznam na rozdíl od pole) operace provedou se seznamem (listem): [8]

- vytvoření seznamu s danými hodnotami:

```
List<string> barvy = new List<string>(){ "černá", "bílá", "modrá"};
```

- vypsání hodnoty na dané pozici v seznamu:

```
Console.Write(barvy[2]);
```

- uložení hodnoty na danou pozici seznamu (příp. přepsání hodnoty na této pozici):

```
barvy[2] = "hnědá";
```

- zjištění délky seznamu:

```
int velikostSeznamu = barvy.Count;
```

- přidání prvku do seznamu:

```
barvy.Add("červená");
```

- odebrání prvku ze seznamu:

```
barvy.Remove("bílá");
```

- vytvoření prázdného seznamu a jeho naplnění daty zadanými do konzole:

```
List<int> znamky = new List<int>();
Console.Write("Kolik známek budete zadávat? ");
int pocet = int.Parse(Console.ReadLine());
for(int i = 0; i < pocet; i++)
{
    int znamka = int.Parse(Console.ReadLine());
    znamky.Add(znamka);
}
```

Využití pole v C# si představíme na stejných příkladech jako ve Scratchi. Ve všech příkladech v C# budeme používat pole, práci se seznamem si můžete zkusit samostatně.

Příklad: Vytvořte program, který po zadání čísla měsíce vypíše jeho název (tzn. např. při zadání čísla 5 program vypíše „květen“).

```
static void Main()
{
    string[] mesice = {"leden","únor","březen","duben","květen","červen",
           "červenec","srpen","září","říjen","listopad","prosinec" };
    Console.Write("Zadejte číslo měsíce: ");
    int cisloMesice = int.Parse(Console.ReadLine());
    Console.WriteLine(mesice[cisloMesice - 1]);
}
```

V programu nejprve definujeme pole s názvy všech měsíců. Program nás napřed vyzve k zadání čísla měsíce a poté zobrazí název měsíce, který se nachází na požadované pozici (nesmíme zapomenout odečíst 1, protože pole indexuje pozice prvků od 0).

Další příklad ukazuje jednoduchý kvíz na překládání slovíček z češtiny do angličtiny.

Příklad: Vytvořte kvíz, který uživatele vyzkouší z překladu 5 českých slov do angličtiny.

```
static void Main()
{
    string[] cesky = { "pes", "kočka", "prase", "ryba", "včela", "lev" };
    string[] anglicky = { "dog", "cat", "pig", "fish", "bee", "lion" };
    Console.WriteLine("Přeložte do angličtiny:");
    for(int i = 0; i < cesky.Length; i++)
    {
        Console.WriteLine(cesky[i]);
        string odpoved = Console.ReadLine();
        if (odpoved == anglicky[i])
            Console.WriteLine("Správně!");
        else
            Console.WriteLine("Chyba!");
    }
}
```

V kódu jsou nejprve definovány dva seznamy – jeden s českými slovy a druhý s jejich anglickými překlady. Při průchodu cyklem *for* program postupně zobrazuje slovíčka ze seznamu česky a nechává uživatele zadat odpovědi (překlady), které porovnává s hodnotami seznamu anglicky. Podle toho vyhodnotí, jestli je zadaná odpověď správná, nebo chybná.

Textový řetězec, tedy proměnná datového typu `string`, je vlastně pole znaků, proměnných datového typu `char`. Využití této skutečnosti si ukážeme na následující aplikaci, která zašifruje zadané slovo.

Příklad: Vytvořte program, který zašifruje slovo posunem všech jeho znaků o jedno písmeno dopředu v anglické abecedě („z“ se posune na „a“). Program bude předpokládat zadání slova bez diakritiky (tj. pouze ze znaků anglické abecedy).

```
static void Main()
{
    Console.Write("Zadejte slovo k zašifrování: ");
    string slovoPuvodni = Console.ReadLine();
    string slovoZasifrovane = "";
    for(int i = 0; i < slovoPuvodni.Length; i++)
    {
        char znakPuvodni = slovoPuvodni[i];
        int cisloZnaku = (int)znakPuvodni;
        cisloZnaku++;
        if(cisloZnaku == 123)
        {
            cisloZnaku = cisloZnaku - 26;
        }
        char znakZasifrovany = (char)cisloZnaku;
        slovoZasifrovane = slovoZasifrovane + znakZasifrovany.ToString();
    }
    Console.WriteLine(slovoZasifrovane);
}
```

Program nechá uživatele zadat slovo a uloží ho do proměnné *slovoPuvodni*. Proměnnou *slovoZasifrovane* nastaví na prázdnou hodnotu. Pak prochází hodnotu v proměnné *slovoPuvodni* znak po znaku. Pro každý znak zjistí program jeho číselný kód, toto číslo zvětší o 1 a opět ho převede na znak. Následně tento znak přidá do proměnné *slovoZasifrovane*. Tyto kroky program opakuje, dokud nedojde na konec slova.

7 OŠETŘENÍ UŽIVATELSKÝCH VSTUPŮ

V předchozích kapitolách jsme vytvořili několik různých praktických programů. Zatím ale naše aplikace předpokládaly, že hodnota zadaná uživatelem bude mít smysl (např. že pokud je uživatel vyzván k zadání čísla, opravdu zadá číslo, a ne slovo). Chybné údaje zadané uživatelem jsou jedním z nejčastějších důvodů, proč dochází k chybám v programu.

V této krátké kapitole se proto podíváme na to, jak ošetřit situace, kdy hodnota zadaná uživatelem bude neplatná (nesmyslná). Smyslem ošetřování chyb je vytvořit program, který nás v případě zadání neplatné hodnoty na tuto skutečnost upozorní, případně nás vyzve k opakovanému zadání hodnoty. Pokud v programu chyby nejsou ošetřené, povede chyba k jeho vynucenému ukončení a ke ztrátě všech neuložených dat. [8]

V programu může dojít vlivem uživatelského vstupu k chybě ze dvou různých důvodů. V prvním případě uživatel zadá očekávaný datový typ a hodnota se do proměnné načte, nemá ale smysl v daném kontextu. Jde například o zadání záporného počtu kusů při tvorbě objednávky nebo o zadání čísla, které při určitém výpočtu vede k dělení nulou.

Tyto chyby se ošetřují pomocí podmínek (situaci se zadáním záporného počtu kusů jsme si již zkoušeli ošetřit). Při tvorbě aplikace musí programátor důkladně promyslet různé situace, k nimž může dojít vlivem zadání neplatných dat, a tyto situace vhodně podmínkami ošetřit. V případě, že chceme nejen vypsat upozornění na chybu, ale také uživatele vyzvat k opakovanému zadání hodnoty, umístíme podmínku do cyklu *while*.

V druhém případě chyby způsobuje zadání jiného datového typu, než program očekává – např. je zadán textový řetězec ve chvíli, kdy program očekává zadání čísla. Příkazy `int.Parse()` a `double.Parse` neumějí text zpracovat, proto dojde k chybě a program je nuceně ukončen. Tyto chyby nelze ošetřit pomocí podmínek, používají se proto výjimky.

V následující ukázce [13] je uvedeno, jak ošetřit načtení hodnoty datového typu *int* do proměnné *x* hláškou o chybě a výzvou k opakovanému zadání hodnoty.

```
static void Main()
{
    Console.Write("Zadejte celé číslo: ");
    int x;
    while( !int.TryParse(Console.ReadLine(), out x) )
    {
        Console.WriteLine("Neplatná hodnota, opakujte zadání: ");
    }
}
```

Správně vytvořená aplikace by měla mít ošetřené všechny možné chybové stavy, aby v případě chyby na ni uživatele upozornila, případně mu dala možnost opravy, ale nebyla nuceně ukončena bez uložení dat.

Ošetření chyb si ukážeme na příkladu aplikace, která bude počítat druhou odmocninu ze zadaného čísla. [13]

Příklad: Vytvořte program, který vypočítá druhou odmocninu zadaného čísla zaokrouhlenou na dvě desetinná místa. V programu ošetřete možné chybové stavy.

```
static void Main()
{
    Console.WriteLine("Program pro výpočet druhé odmocniny");
    Console.Write("Zadejte číslo: ");
    double cislo;
    while( !double.TryParse(Console.ReadLine(), out cislo) )
    {
        Console.WriteLine("Neplatná hodnota, opakujte zadání: ");
    }
    if(cislo >= 0)
    {
        double odmocnina = Math.Round(Math.Sqrt(cislo), 2);
        Console.Write("Druhá odmocnina čísla je: " + odmocnina);
    }
    else
    {
        Console.Write("Záporné číslo není možné odmocnit.");
    }
}
```

V programu je použita jedna výjimka k ošetření načtení desetinného čísla. V případě zadání jiné než číselné hodnoty nás program na tuto chybu upozorní a bude nás opakovaně vyzývat k zadání hodnoty tak dlouho, než ji zadáme správně.

Výpočet odmocniny je ošetřen podmínkou. V případě zadání záporného čísla nás program na tento problém upozorní a ukončí se.

K výpočtu druhé odmocniny slouží příkaz `Math.Sqrt()`, do závorek uvedeme odmocňovanou hodnotu.

8 METODY

Dosud jsme psali všechny příkazy dovnitř metody *Main()*, která byla jedinou metodou v našem programu. Tento postup si můžeme dovolit, pokud se s programováním teprve seznamujeme nebo pokud vytváříme malý program s krátkým kódem.

Jako metoda se označuje část kódu s určitou funkcí. Metod můžeme v programu vytvořit libovolné množství. Abychom si vysvětlili princip metod, představme si, že vytváříme hru, ve které se postavy pohybují v rovině. Poloha postav je popsaná souřadnicemi a v určitých chvílích potřebujeme určit vzdálenost postav. Za tímto účelem si vytvoříme metodu, která bude přebírat souřadnice dvou postav a z těchto souřadnic vypočítá jejich vzdálenost. V hlavní metodě *Main()* pak v případě potřeby výpočtu vzdálenosti postav pouze voláme vytvořenou metodu, které předáme souřadnice obou postav, a obdržíme výsledek. Přitom se nemusíme zabývat tím, jak metoda určující vzdálenost vnitřně funguje.

Použitím metod docílíme přehlednosti kódu a také zmenšíme jeho množství. Pokud potřebujeme v kódu řešit na více různých místech tentýž problém, namísto duplikování stejného kódu pouze voláme metodu. Kód větších programů je třeba vždy rozdělovat do metod. Pokud budete ve studiu programování pokračovat, pravděpodobně se seznámíte i s přístupem zvaným objektově orientované programování a naučíte se program dělit do tříd (což je ještě vyšší jednotka, než metoda). Téma tříd ale přesahuje rozsah tohoto textu.

Existují dva typy metod: metody bez návratové hodnoty (např. pro výpis do konzole) a metody s návratovou hodnotu (např. provedou výpočet a vrátí jeho výsledek). Příklady obou metod si ukážeme a vysvětlíme si na nich, jak zápis metod vypadá.

Začneme příkladem metody bez návratové hodnoty – metody, která přijme jméno a věk osoby a tyto údaje vypíše do konzole. Funkčnost metody ověříme jejím voláním z hlavní metody.

```
static void PredstavOsobu(string jmeno, int vek)
{
    Console.WriteLine("Toto je {0} a má {1} let.", jmeno, vek);
}

static void Main()
{
    PredstavOsobu("Adam", 28);
}
```

Zápis metody se skládá z těchto částí (v tomto pořadí):

- Modifikátor přístupu: V našich příkladech budeme používat vždy `static`. Jeho přesný význam pochopíte a další modifikátory přístupu se naučíte, pokud se rozhodnete studovat objektově orientované programování (např. podle [8]).
- Návratová hodnota: U metod s návratovou hodnotou se uvede datový typ proměnné, kterou metoda vrací (*int*, *double*, *string*, *bool* atd.). U metod, které žádnou hodnotu nevrací, se uvede *void*.
- Název metody: Je zvykem psát ho s velkým počátečním písmenem. Díky tomu snadno rozlišíme název metody od názvu proměnné.
- Parametry: Jedná se o proměnné, které metoda přijímá. Zapisujeme je do závorky za název metody s označeními jejich datových typů.
- Tělo metody: Jedná se o kód ve složených závorkách následující za řádkem s definicí metody. Pokud má metoda návratovou hodnotu, musí být na posledním řádku v jejím těle příkaz `return` spolu s názvem proměnné, kterou metoda vrací.

Je třeba si uvědomit, že veškeré proměnné vytvořené uvnitř metody existují pouze při provádění této metody. Chceme-li proměnnou využívat i po proběhnutí metody, musíme tuto proměnnou z metody předat jako její návratovou hodnotu.

Příkladem metody s návratovou hodnotou je metoda, která při zadání odvěsen pravoúhlého trojúhelníka dopočítá jeho přeponu a tuto hodnotu vrátí zaokrouhlenou na dvě desetinná místa. Kromě metody je opět uvedeno i její volání z hlavní metody.

```
static double VypocitejPreponu(double a, double b)
{
    double c = Math.Sqrt(a*a + b*b);
    c = Math.Round(c, 2);
    return c;
}

static void Main()
{
    VypocitejPreponu(12, 5);
}
```

Poslední užitečnou technikou, kterou si v souvislosti s metodami zmíníme, je jejich přetěžování. C# nám umožňuje vytvořit více metod se stejným názvem lišících se pouze počtem přijímaných parametrů. Při volání metody pak C# sám najde podle počtu parametrů odpovídající metodu a provede ji.

Přetěžování metody si ukážeme na příkladu výpočtu obsahu trojúhelníka.

```
static double UrciObsahTrojuhelnika(double a, double v_a)
{
    double S = 0.5 * a * v_a;
    return S;
}

static double UrciObsahTrojuhelnika(double a, double b, double gama)
{
    double S = 0.5 * a * b * Math.Sin(gama);
    return S;
}

static void Main()
{
    Console.WriteLine(UrciObsahTrojuhelnika(8,6));
    Console.WriteLine(UrciObsahTrojuhelnika(4,7,Math.PI/6));
}
```

V kódu výše jsou uvedené dvě metody – první k výpočtu obsahu využívá délku strany a délku k ní příslušné výšky trojúhelníka. Druhá metoda využívá goniometrický vzorec, ve kterém násobíme délky dvou stran trojúhelníka a sinus úhlu, který tyto strany svírají. Úhel je nutné do programu zadat v radiánech (30° = π/6).

9 ÚVOD DO TVORBY OKENNÍCH APLIKACÍ

Až do této chvíle jsme se zaměřovali pouze na tvorbu konzolových aplikací. Díky tomu jsme se mohli plně soustředit na porozumění vysvětlovaným problémům. Koho by ale nelákalo vytvořit aplikaci grafické rozhraní a přiblížit ji tak zase o krok profesionálnímu výtvoru?

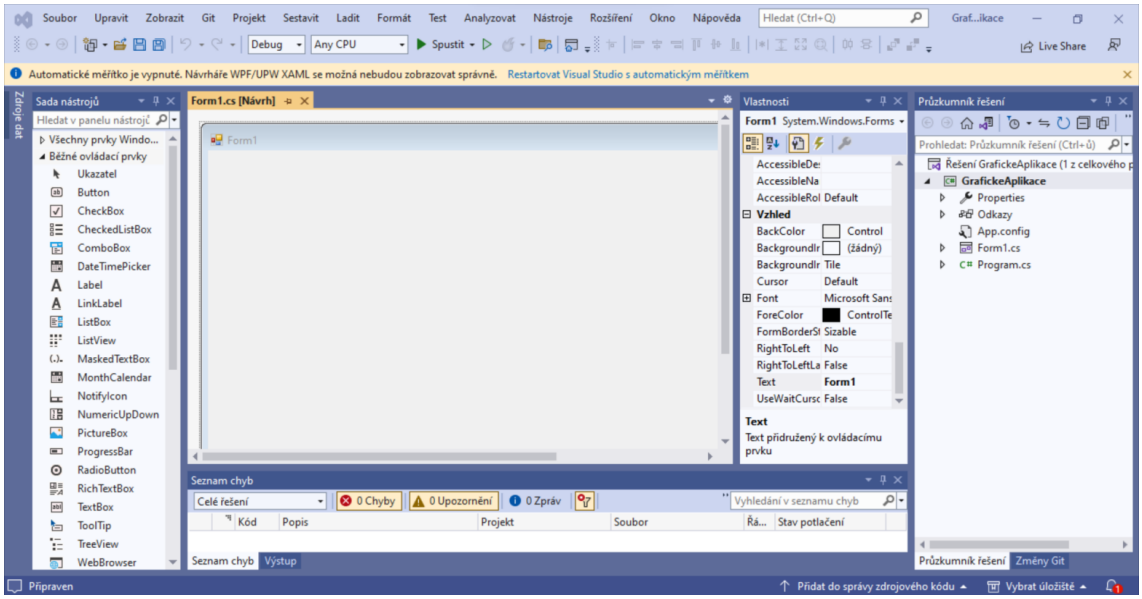
Tato kapitola rozhodně nebude vyčerpávající, jedná se spíše o bonus na závěr. Dozvíme se základní principy tvorby okenních aplikací a zaměříme se pouze na nejdůležitější informace. I s těmito nevelkými znalostmi však dokážeme vytvářet vizuálně i funkčně zajímavé programy.

Vzhledem k tomu, že GDB online tvorbu okenních aplikací nepodporuje, představíme si v této kapitole podrobněji prostředí Visual Studia.

9.1 Založení nového projektu ve Windows Forms

Nejprve si ukážeme, jakým způsobem se vytváří nový projekt Windows Forms, a poté se seznámíme s prostředím pro tvorbu okenních aplikací. [9]

Otevřeme si Visual Studio a klikneme na *Vytvořit nový projekt*. Z nabídky, která se ukáže, vybereme *Aplikace Windows Forms (.NET Framework)*. Zadáme název programu (např. *GrafickeAplikce*) a vybereme umístění, kam se má projekt uložit. Doporučuji zaškrtnout políčko *Umístit řešení a projekt do stejného adresáře*. Potvrdíme kliknutím na tlačítko *Vytvořit*. Otevřelo se nám řešení našeho projektu s podobným vzhledem:



Funkce vývojového prostředí jsou uspořádané do panelů. Panelů je velké množství, pro nás budou důležité zejména tyto:

- Průzkumník řešení (na obrázku vpravo, najdeme v něm všechny soubory, ze kterých se skládá náš projekt),
- Sada nástrojů (na obrázku vlevo, umožňuje nám do vytvářené aplikace tažením přidávat zobrazovací a ovládací prvky – textová pole, tlačítka apod.),
- Vlastnosti (na obrázku vpravo, v tomto okně nastavujeme vlastnosti zobrazovacích a ovládacích prvků – název, rozměry, umístění na formuláři, barvu, styl písma atd.),
- Seznam chyb (v dolní části, oceníme ho zejména při kompilaci před spuštěním programu, kdy nás upozorňuje na syntaktické chyby, kterých jsme se v kódu dopustili),
- Návrhář zobrazení / Editor kódu (uprostřed okna, nejdůležitější část – v návrháři utváříme vzhled, v editoru kódu programujeme ovládání a funkčnost aplikace). [9]

Polohu a rozložení panelů na obrazovce je možné měnit tažením myší. Pokud se nám některá z uvedených částí sama nezobrazí nebo si ji omylem zavřeme, zobrazíme ji tímto způsobem:

- Průzkumník řešení: v horní liště klikneme na *Zobrazit → Průzkumník řešení*.
- Sada nástrojů: v horní liště klikneme na *Zobrazit → Panel nástrojů*.
- Vlastnosti: v horní liště klikneme na *Zobrazit → Okno vlastností*.
- Seznam chyb: v horní liště klikneme na *Zobrazit → Seznam chyb*.
- Návrhář zobrazení / Editor kódu: v průzkumníku řešení klikneme pravým tlačítkem na *Form1.cs → Zobrazit kód* nebo *Návrhář zobrazení*.

Pokud nám Visual Studio nabízí možnost restartu se 100% měřítkem, je vhodné ho provést. Rozměry aplikace v návrháři zobrazení potom budou shodné s rozměry spuštěné hotové aplikace (toto souvisí s měřítkem monitoru; při měřítku jiném než 100% se rozměry připravované aplikace v návrháři zobrazení a rozměry hotové aplikace mohou lišit).

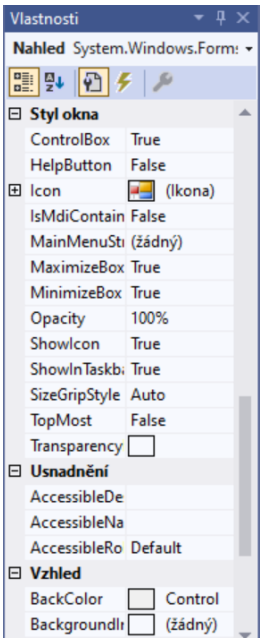
Ve vývojovém prostředí máme vytvořený a otevřený prázdný nový projekt aplikace Windows Forms. Stisknutím klávesy *F5* zkompilujeme a spustíme program. Otevře se nám prázdné okno (protože žádné ovládací prvky ani funkcionality jsme zatím nevytvořili).

Podívejme se ještě do složky našeho projektu (na ploše nebo v umístění, kde jsme projekt vytvořili). Složka bude obsahovat další složky a několik souborů:

bin	21.06.2023 12:15	Složka souborů	
obj	21.06.2023 12:15	Složka souborů	
Properties	21.06.2023 12:15	Složka souborů	
App.config	21.06.2023 12:15	Soubor CONFIG	1 kB
Form1.cs	21.06.2023 12:15	Soubor CS	1 kB
Form1.Designer.cs	21.06.2023 12:15	Soubor CS	2 kB
GrafickeAplikace.csproj	21.06.2023 12:15	VisualStudio.Launche...	4 kB
GrafickeAplikace.sln	21.06.2023 12:15	Visual Studio Solution	2 kB
Program.cs	21.06.2023 12:15	Soubor CS	1 kB

Soubory s příponou *.cs* jsou zdrojové soubory jazyka C#. Při tvorbě okenních aplikací vytváří vývojové prostředí vstupní soubor *Program.cs* s metodou *Main* a pro každé okno, které aplikace obsahuje, dvojici souborů – soubor s kódem obsahující zápis funkcionalit (výchozí název *Form1.cs*) a soubor s informacemi o vzhledu okna (výchozí název *Form1.Designer.cs*). Kromě souborů s příponou *.cs* složka s řešením obsahuje další systémové soubory. Pro nás jsou nejpodstatnější:

- Řešení (soubor s příponou *.sln*) – umožňuje nám otevřít všechny soubory projektu ve vývojovém prostředí a editovat je.
- Soubor *GrafickeAplikace.exe* ve složce *bin* → *Debug* – obsahuje námi vytvořený program jako spustitelný soubor (chceme-li někomu dát jen hotový funkční program bez zdrojových kódů a možnosti jej upravovat, pošleme mu tento soubor).



Ve vývojovém prostředí máme nyní otevřený zatím stále prázdný projekt aplikace Windows Forms, konkrétně návrhář zobrazení okna *Form1* (pokud jej vývojové prostředí neotevřelo automaticky, v průzkumníku řešení klineme pravým tlačítkem na *Form1.cs* a vybereme *Návrhář zobrazení*). Klikneme v návrháři na formulář (návrh okna aplikace) a podíváme se do okna vlastností (zde vidíme všechny vlastnosti, které lze nastavit nebo upravit).

Vlastnost *Text* se zobrazuje jako popisek okna. Tuto vlastnost změníme na *Aplikace*. Mezi další vlastnosti, které lze nastavit, patří rozměry (*Size*) a barva pozadí okna (*Backcolor*). Najděte si tyto vlastnosti a zkuste si změnit jejich hodnoty.

9.2 Hrací kostka

Naším cílem bude vytvořit okenní aplikaci, která po kliknutí na tlačítko zobrazí náhodně vygenerované celé číslo od 1 do 6. Způsobem popsaným v předchozí podkapitole si vytvoříme nový projekt Windows Forms. Název zvolíme *HraciKostka*. [9]

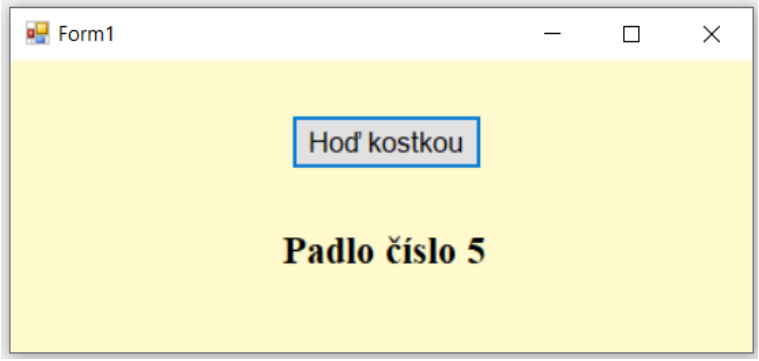
Popisek okna nastavíme na *Hrací kostka* a pozadí aplikace si obarvíme na námi zvolenou barvu. Ze sady nástrojů přeneseme v návrháři zobrazení do formuláře tlačítko (*button*) a popisek (*label*). Oba prvky umístíme na střed formuláře, tlačítko bude výše a pod ním bude popisek.

Klikneme na tlačítko a v okně vlastností mu nastavíme vlastnost *Name* na *buttonHod* a vlastnost *Text* na *Hoď kostkou*. Rozměry tlačítka (*Size*) a vlastnosti písma (*Font*) nastavíme podle vlastního uvážení. Popisku nastavíme vlastnost *Name* na *labelVysledek* a vlastnost *Text* ponecháme prázdnou. Vlastnosti písma (*Font*) opět nastavíme podle vlastního uvážení, velikost (*Size*) nenastavujeme, protože u prvku *label* se velikost přizpůsobuje textu uvnitř tohoto prvku.

V návrháři zobrazení dvakrát klikneme na tlačítko a budeme přesměrováni do editoru kódu. V editoru kódu se vytvořila metoda, která se provede po kliknutí na tlačítko. Dovnitř metody vložíme příkazy podle vzoru níže tak, aby celá metoda měla tuto podobu:

```
private void buttonHod_Click(object sender, EventArgs e)
{
    Random generator = new Random();
    int nahodneCislo = generator.Next(1,7);
    labelVysledek.Text = "Padlo číslo " + nahodneCislo;
}
```

Význam prvních dvou příkazů uvnitř metody je zřejmý, třetí příkaz provádí výpis výsledku do popisku *labelVysledek*. Hotová aplikace by měla mít tento vzhled:



9.3 Získání údajů od uživatele

Většina aplikací potřebuje pro smysluplnou funkci získat údaje od uživatele. Jak to provedeme u okenních aplikací, si ukážeme na příkladu programu, kterému zadá uživatel své jméno a svůj věk a který pak vyhodnotí, zda uživatel smí řídit auto (tj. je starší 18 let).

Známým způsobem vytvoříme nový projekt Windows Forms. Název zvolíme *Auto*. Popisek okna nastavíme na *Můžeš řídit auto?* a pozadí aplikace obarvíme zvolenou barvou.

Ze sady nástrojů přeneseme v návrháři zobrazení do formuláře tři popisky (*label*), textové pole (*textBox*), pole pro číslo (*numericUpDown*) a tlačítko (*button*). Názvy prvků (vlastnost *Name*) nastavíme podle tabulky níže a prvky ve formuláři rozmístíme podle ukázky na konci této podkapitoly. Velikost formuláře, velikosti jednotlivých prvků a vlastnosti písma vhodně nastavíme podle vlastního uvážení.

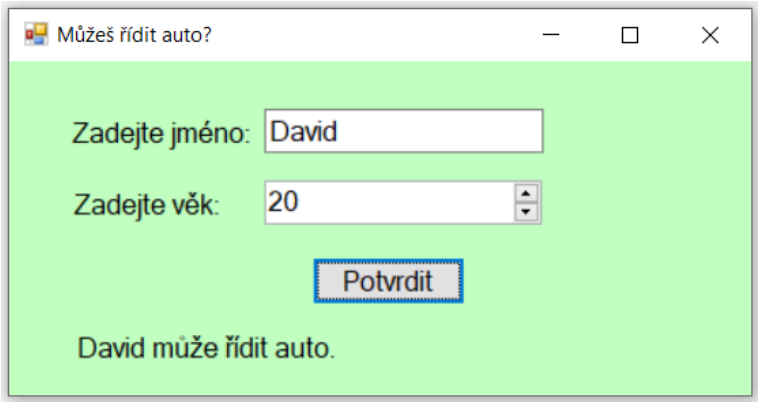
Typ prvku	**Vlastnost *Name***
label (vložen třikrát)	*labelJmeno*, *labelVek*, *labelVysledek*
textBox	*textBoxJmeno*
numericUpDown	*textBoxVek*
button	*buttonPotvrdit*

Popisku *labelJmeno* nastavíme zobrazovaný text (vlastnost *Text*) na *Zadejte jméno:*, popisku *labelVek* nastavíme zobrazovaný text (vlastnost *Text*) na *Zadejte věk:* a popisek *labelVysledek* necháme prázdný, program do něj vloží obsah až za běhu.

V návrháři zobrazení dvakrát klikneme na tlačítko a budeme přesměrováni do editoru kódu. V editoru kódu se vytvořila metoda, která se provede po kliknutí na tlačítko (stejně jako tomu bylo při programování hrací kostky v předchozí podkapitole). Dovnitř metody vložíme příkazy podle vzoru níže tak, aby celá metoda měla tuto podobu: [9]

```
private void buttonPotvrdit_Click(object sender, EventArgs e)
{
    string jmeno = textBoxJmeno.Text;
    int vek = (int)numericUpDownVek.Value;
    if (vek < 18)
    {
        labelVysledek.Text = jmeno + " nesmí řídit auto.";
    }
    else
    {
        labelVysledek.Text = jmeno + " může řídit auto.";
    }
}
```

První dva příkazy ukládají údaje z formuláře do proměnných. V případě věku je třeba provést přetypování na *int*, protože výchozím datovým typem prvku *numericUpDown* je *decimal*. Poté následuje podmínka, která řídí výpis textu do popisku *labelVysledek*. Vzhled okna aplikace je pro inspiraci uveden na obrázku níže.



10 SBÍRKA ÚLOH K PROCVIČOVÁNÍ

10.1 První program

Úloha 1: Vytvořte program, který po spuštění vypíše do konzole vaše jméno a po stisknutí libovolné klávesy se ukončí.

10.2 Proměnné, výpis a načtení údajů

Úloha 1: Programátor chce vypsat údaje do konzole, ale v kódu se dopustil chyb. Odhalte tyto chyby a opravte je.

a)
```
Console.WriteLine(Vítejte v matematickém kvízu);
```

b)
```
string jmeno = "David";
int uspesnost = 2;
Console.WriteLine("(0) dosáhl úspěšnosti (1) %" jmeno uspesnost);
```

c)
```
string hora = "Sněžka";
int vyska = 1603;
Console.WriteLine("hora" + " je vysoká " + "vyska" + " metrů.");
```

Úloha 2: Programátor zapsal příkaz pro načtení údaje z konzole a jeho uložení do proměnné, ale dopustil se v příkazu chyby. Odhalte tuto chybu a opravte ji.

a)
```
string jmeno = ReadLine();
```

b)
```
string nazevKnihy = Console.ReadLine;
```

c)
```
int cena = Console.ReadLine();
```

d)
```
string jmeno&prijmeni = Console.ReadLine();
```

e)
```
int 1den = int.Parse();
```

Úloha 3: Vytvořte program podle pokynů:

a) Program se uživatele zeptá na jeho jméno a na jeho oblíbený film. Poté program vypíše hlášku „*Jméno* má rád *film*.“ (místo slov vyznačených kurzívou program do věty doplní údaje zadané uživatelem).

b) Program se uživatele zeptá na jméno soutěžícího a jeho pořadí. Poté program vypíše hlášku „*Soutěžící* skončil na *pořadí*. místě.“ (místo slov vyznačených kurzívou program do věty doplní údaje zadané uživatelem). Pořadí bude mít datový typ *int*.

Úloha 4: Nejprve určete výsledek uvedeného dělení zpaměti. Svůj výsledek poté ověřte provedením tohoto příkazu a vypsáním výsledné hodnoty.

a) `double c = 30/4;`

b) `double c = 30.0/4;`

c) `int c = 30/4;`

d) `int c = 30.0/4;`

Úloha 5: Vytvořte program, který se uživatele zeptá na tři údaje: jeho jméno, rok narození a myšlený rok. Poté program vypíše hlášku „*Jméno* bude mít v roce *myšlený rok věk* let.“ (místo slov vyznačených kurzívou program do věty doplní konkrétní hodnoty).

Úloha 6: Vytvořte program, který nechá uživatele postupně zadat dvě čísla. Poté vypíše jejich součet, rozdíl, součin a podíl zaokrouhlený na dvě desetinná místa. (Rozdíl a podíl budou vypočítány v zadaném pořadí.)

Úloha 7: Vytvořte program, který postupně načte koeficienty *a*, *b*, *c*, *d* lineární rovnice $ax + b = cx + d$ a určí její řešení zaokrouhlené na dvě desetinná místa. Předpokládáme, že platí $a \neq c$. (Nápověda: řešení rovnice má tvar $x = \frac{d-b}{a-c}$).

Úloha 8: Doplňte na vyznačené místo část kódu tak, aby došlo k záměně číselných hodnot v proměnných *a*, *b*.

a) Vytvořte a použijte pomocnou proměnnou *c*.

b) Úkol vyřešte bez vytváření jakékoliv další proměnné (pouze s proměnnými *a*, *b*).

```
static void Main()
{
    Console.Write("Zadej a: ");
    int a = int.Parse(Console.ReadLine());
    Console.Write("Zadej b: ");
    int b = int.Parse(Console.ReadLine());
    Console.WriteLine();

    Console.WriteLine("Původní stav: ");
    Console.WriteLine("a = " + a + "; b = " + b);

    //zde doplňte svůj kód

    Console.WriteLine("Stav po záměně: ");
    Console.WriteLine("a = " + a + "; b = " + b);
}
```

10.3 Podmínky

Úloha 1: Vytvořte program, který bude simulovat čidlo monitorující teplotu v lednici. Program si nechá zadat teplotu (jako desetinné číslo). Podle teploty vypíše:

- „Teplota je příliš vysoká.“, pokud je teplota vyšší než 6 °C.
- „Teplota je v pořádku.“, pokud je teplota mezi 1 °C až 6 °C (včetně těchto hodnot).
- „Teplota je příliš nízká.“, pokud je teplota nižší než 1 °C.

Úloha 2: V následujícím kódu se nachází jedna chyba. Najděte a opravte ji.

```
static void Main()
{
    string heslo = Console.ReadLine();
    if (heslo = "a7P4q6@BV")
    {
        Console.WriteLine("Vyčkejte, probíhá přihlašování.");
    }
    else
    {
        Console.WriteLine("Bylo zadáno nesprávné heslo.");
    }
}
```

Úloha 3: David měl za úkol vytvořit program, který postupně zjistí, jestli je zadané číslo dělitelné 2, 5 a 7, a výsledky vypíše (tzn. např. při zadání čísla 28 program vypíše „Číslo je dělitelné dvěma.“ a Číslo je dělitelné sedmi.“).

David vytvořil uvedený kód, který ale nefunguje tak, jak předpokládal. Zjistěte, co je v kódu špatně, a chybu opravte.

```
static void Main()
{
    Console.Write("Zadejte celé číslo: ");
    int cislo = int.Parse(Console.ReadLine());
    if (cislo % 2 == 0)
    {
        Console.WriteLine("Číslo je dělitelné dvěma.");
    }
    else if (cislo % 5 == 0)
    {
        Console.WriteLine("Číslo je dělitelné pěti.");
    }
    else if (cislo % 7 == 0)
    {
        Console.WriteLine("Číslo je dělitelné sedmi.");
    }
}
```

Úloha 4: Je dán rovnoběžník *ABCD.* Vytvořte program, který požádá uživatele o zadání čtyř hodnot – *a, b, alfa, beta* (platí $a = |AB|$, $b = |BC|$, *alfa* = $|\sphericalangle BAD|$, *beta* = $|\sphericalangle ABC|$). Podle zadaných hodnot program rozhodne, jestli se jedná o čtverec, obdélník, kosočtverec, nebo kosodélník.

Úloha 5: Obchod má otevřeno od 8 do 17 hodin. Vytvořte program, který po zadání aktuálního času zobrazí, zda má obchod otevřeno. Program uživatele vyzve nejprve k zadání hodiny a poté minut. Úlohu řešte dvakrát, poprvé využitím operátoru *&&* (*a zároveň*), podruhé využitím operátoru || (*nebo*).

Úloha 6: Vytvořte program, který nechá uživatele zadat dvě celá čísla – horní a dolní mez intervalu. Poté program vygeneruje náhodné číslo z daného intervalu (včetně horní i dolní meze). V případě chybného zadání od uživatele, kdy zadaná dolní mez bude větší než horní, program vypíše upozornění na chybu a náhodné číslo generovat nebude.

Úloha 7: Vytvořte program, který náhodně vygeneruje jeden příklad z malé násobilky, nechá uživatele zadat výsledek a odpověď vyhodnotí.

Úloha 8: Vytvořte program, který po zadání čísla měsíce zobrazí roční období, v němž se tento měsíc nachází. V případě zadání neplatné hodnoty (menší než 1 nebo větší než 12) nás program na to upozorní. Úlohu řešte dvakrát, poprvé převážně využitím operátoru *&&* (*a zároveň*), podruhé převážně využitím operátoru || (*nebo*).

Úloha 9: Vytvořte program, který postupně načte koeficienty *a*, *b*, *c* kvadratické rovnice $ax^2 + bx + c = 0$ a určí její řešení v oboru reálných čísel. Program nejprve vypíše diskriminant. Podle hodnoty diskriminantu rozhodne o počtu řešení rovnice. Pokud řešení existuje, program tyto hodnoty vypíše zaokrouhlené na dvě desetinná místa. Pokud řešení ne-existuje, program nás na tuto skutečnost upozorní. Předpokládáme, že platí $a \neq 0$. (Nápověda: Diskriminant určíme vztahem $D = b^2 - 4ac$. Pokud platí $D > 0$, rovnice má dvě řešení ve tvaru $x_1 = \frac{-b+\sqrt{D}}{2a}$, $x_2 = \frac{-b-\sqrt{D}}{2a}$. Pokud platí $D = 0$, rovnice má jedno řešení ve tvaru $x = \frac{-b}{2a}$. Pokud platí $D < 0$, rovnice nemá žádné řešení.)

10.4 Cykly

Úloha 1: Použitím cyklu *while* vyřešte tyto úlohy:

a) Adam si spořil do kasičky. Poprvé do ní v září 2018 vhodil 30 Kč. Každý další měsíc vhazoval o 3 Kč více než předchozí. Jakou částku měl naspořenou v květnu 2023?

b) Marcela se rozhodla pravidelně spořit. První vklad v listopadu 2019 činil 40 Kč. Každý další měsíc vkládala o 5 Kč více než předchozí. Ve kterém měsíci překročila naspořenou částku 3000 Kč?

c) Karel si založil stavební spoření. První vklad v lednu 2023 činil 2300 Kč. Každý další měsíc vkládal o 80 Kč více než předchozí. V prosinci každého roku se mu připisuje státní podpora 2000 Kč. Ve kterém měsíci bude mít naspořenou částku 180 000 Kč na nové auto? (Úroky z vkladů v této úloze neuvažujte.)

Úloha 2: Použitím cyklu *for* vypočítejte součet:

a) všech přirozených čísel od 1 do 10 (včetně).

b) všech přirozených čísel od 40 do 60 (včetně).

c) všech lichých čísel ležících mezi 0 a 100.

d) všech přirozených čísel dělitelných 7 ležících mezi 0 a 100.

e) všech přirozených čísel dělitelných 11 ležících mezi 200 a 400.

f) všech prvočísel ležících mezi 150 a 250.

Úloha 3: Vytvořte program, který nechá uživatele zadat číslo a vypíše jeho prvních 10 kladných násobků.

Úloha 4: Vytvořte program, který vypíše prvních 10 násobků čísel od 1 do 10 (na jednom řádku bude vždy 10 těchto násobků oddělených čárkami). Viz náznak výstupu níže. [13]

```
Násobky 1: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Násobky 2: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20
...
Násobky 10: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
```

Úloha 5: Vytvořte program, který bude náhodně generovat příklady z malé násobilky. Po každém vygenerování příkladu nechá uživatele zadat výsledek a odpověď vyhodnotí.

a) Počet vygenerovaných příkladů zadá na počátku uživatel.

b) Po každém vyhodnocení odpovědi bude program čekat na stisk klávesy. Při stisku klávesy „k“ se ukončí, při stisku jiné klávesy bude pokračovat generováním dalšího příkladu.

Úloha 6: Vytvořte program, který po zadání čísla vypíše všechny jeho dělitele.

Úloha 7: Tomáš se snažil vytvořit program, který vypočítá průměr zadaných známek zaokrouhlený na dvě desetinná místa. Dopustil se v něm ale tří chyb. Odhalte a opravte je.

```
static void Main()
{
    Console.WriteLine("Vítejte v programu pro výpočet průměru známek!");
    Console.Write("Kolik známek budete zadávat? ");
    int pocet = int.Parse(Console.ReadLine());
    int soucet;
    for(int i = 0; i < pocet; i++)
    {
        int znamka = int.Parse(Console.ReadLine());
        if (znamka > 0 || znamka < 6)
        {
            soucet = soucet + znamka;
        }
        else
        {
            Console.WriteLine("Zadali jste neplatnou hodnotu!");
        }
    }
    double prumer = soucet/pocet;
    double zaokrouhlenyPrum = Math.Round(prumer, 2);
    Console.WriteLine("Průměr zadaných známek je: " + zaokrouhlenyPrum);
}
```

Úloha 8: Vykreslete uvedené obrazce. Použijte cyklus *for* vnořený do cyklu *for*:

a)	b)	c)	d)
+ + + o + +	+ o + o + o	+ o + o + o	+
+ + + o + +	o + o + o +	o o o o o o	+ +
o o o o o o	+ o + o + o	+ o + o + o	+ + +
+ + + o + +	o + o + o +	o o o o o o	+ + + +
+ + + o + +	+ o + o + o	+ o + o + o	+ + + + +
+ + + o + +	o + o + o +	o o o o o o	+ + + + + +

Úloha 9: Vyřešte úlohu 8d použitím jediného cyklu *for*.

Úloha 10: Fibonacciho posloupnost začíná čísly 0 a 1 a každý další člen je součtem dvou předchozích členů ($a_1 = 0, a_2 = 1, a_n = a_{n-1} + a_{n-2}$).

a) Vytvořte program, který vypíše prvních 15 členů této posloupnosti.

b) Vytvořte program, který vypíše část Fibonacciho posloupnosti od *m*-tého do *n*-tého členu (včetně těchto členů). Hodnoty *m*, *n* bude zadávat uživatel. Otestujte program pro hodnoty *m* = 30, *n* = 35.

Úloha 11: Žáci měli za úkol vytvořit program, který nechá uživatele zadat čitatel a jmenovatel zlomku, poté zlomek zkrátí do základního tvaru a vypíše ho. Eva vytvořila program, který funguje jen pro některé zlomky. Najděte příklad zlomku, pro který program funguje, a příklad zlomku, pro který program nepracuje správně (nezkrátí ho do základního tvaru). Pokuste se odhalit v programu chybu a opravit ji co nejjednodušším způsobem.

```
static void Main()
{
    Console.Write("Zadejte čitatel zlomku: ");
    int citatel = int.Parse(Console.ReadLine());
    Console.Write("Zadejte jmenovatel zlomku: ");
    int jmenovatel = int.Parse(Console.ReadLine());

    int mensi;
    if(citatel > jmenovatel)
    {
        mensi = jmenovatel;
    }
    else
    {
        mensi = citatel;
    }

    for(int i = 2; i <= mensi; i++)
    {
        bool deleniCitatele = (citatel % i == 0);
        bool deleniJmenovatele = (jmenovatel % i == 0);
        if (deleniCitatele && deleniJmenovatele)
        {
            citatel = citatel/i;
            jmenovatel = jmenovatel/i;
        }
    }

    Console.WriteLine("Zkrácený zlomek: {0}/{1} ", citatel, jmenovatel);
}
```

Úloha 12: Vytvořte program pro převod čísla z desítkové soustavy do dvojkové a naopak. Uživatel si po spuštění programu vybere, zda chce provést převod z dvojkové soustavy do desítkové nebo naopak. V případě zadání neplatné hodnoty program uživatele na tuto skutečnost upozorní a ukončí se.

10.5 Pole a seznam

Úloha 1: Vytvořte program, který po zadání číselného vyjádření měsíce vypíše jeho slovní vyjádření (zadáme-li např. 4, program vypíše „duben“).

Úloha 2: Programátor zapsal příkaz pro vytvoření pole hodnot, ale dopustil se v něm chyby. Odhalte tuto chybu a opravte ji.

a) `string[] predmety = ( "Český jazyk", "Matematika", "Chemie" );`

b) `int teploty = { 20, 17, 15, 18, 23 };`

c) `string[] otazky = string[8];`

d) `string[] jmena = { Karel, Jana, Markéta, David };`

e) `int[] znamky = new int(10);`

Úloha 3: V kódu uvnitř hlavní metody se nachází na každém řádku jedna syntaktická chyba. Odhalte tyto chyby a opravte je.

```
static void Main()
{
    string obdobi = { "jaro", "leto", "podzim", "zima" };
    Console.WriteLine(Zadejte číslo ročního období: );
    int cislo = Console.ReadLine();
    string zvoleneObdobi = obdobi(cislo - 1);
    Console.WriteLine("{0}. roční období je {1}", cislo, zvoleneObdobi)
}
```

Úloha 4: Klára se snažila vytvořit kód, který vyzkouší uživatele z překladu jednoho náhodně vybraného slovíčka. Kód níže, který vytvořila, ale nefunguje správně. Odhalte v něm chybu a opravte ji.

```
static void Main()
{
    string[] cesky = { "pes", "kočka", "prase", "ryba", "včela", "lev" };
    string[] anglicky = { "dog", "cat", "pig", "fish", "bee", "lion" };
    Random generator = new Random();
    string slovicko = cesky[generator.Next(0,6)];
    Console.WriteLine("Přeložte do angličtiny: ");
    Console.WriteLine(slovicko);
    string odpoved = Console.ReadLine();
    if (odpoved == anglicky[generator.Next(0,6)])
        Console.WriteLine("Správně!");
    else
        Console.WriteLine("Chyba!");
}
```

Úloha 5: Upravte program z předchozího úkolu takovým způsobem, aby vyzkoušel uživatele ze všech dostupných slovíček tak, že po každém spuštění se budou slovíčka zobrazovat v jiném (náhodném) pořadí. Pokuste se vymyslet alespoň dvě různá řešení tohoto úkolu.

Úloha 6: Zjednodušte níže uvedený kód tak, aby plnil stejnou funkci, ale obsahoval pouze jeden *for* cyklus. Proč je řešení s použitím jednoho cyklu *for* výhodnější?

```
static void Main()
{
    Console.WriteLine("Program pro analýzu denních teplot");
    Console.Write("Kolik hodnot budete zadávat? ");
    int pocet = int.Parse(Console.ReadLine());
    double[] teploty = new double[pocet];

    for(int i = 0; i < pocet; i++)
    {
        Console.Write("{0}. den: ", i+1);
        teploty[i] = double.Parse(Console.ReadLine());
    }

    double minimum = teploty[0];
    for(int i = 1; i < pocet; i++)
    {
        if(teploty[i] < minimum)
        {
            minimum = teploty[i];
        }
    }

    double maximum = teploty[0];
    for(int i = 1; i < pocet; i++)
    {
        if(teploty[i] > maximum)
        {
            maximum = teploty[i];
        }
    }
}
```

Úloha 7: Vytvořte program, který po zadání jména, příjmení a roku narození vygeneruje kód, jenž bude postupně obsahovat: počáteční tři písmena z příjmení, poslední dvě číslice z roku narození a poslední tři písmena ze jména.

Úloha 8: Vytvořte program, který zašifruje slovo posunem všech písmen o daný počet znaků. Po spuštění si program nechá zadat slovo a číslem vyjádřenou velikost posunu. Program bude kromě zašifrování slova schopen také dešifrovat slovo zašifrované tímto způsobem (sami navrhněte vhodný způsob, jak je možné úkol vyřešit).

Úloha 9: Vytvořte generátor náhodného hesla. Vytvořené heslo bude dlouhé 14 znaků a bude obsahovat alespoň dvě malá písmena, alespoň tři velká písmena, alespoň čtyři číslice a alespoň jeden speciální znak.

10.6 Metody

Poznámka: Funkčnost všech metod po vytvoření otestujte jejich voláním z hlavní metody.

Úloha 1: Doplňte tělo metody tak, aby vracela vzdálenost dvou bodů $A[a_x; a_y]$, $B[b_x, b_y]$ v rovině zaokrouhlenou na 1 desetinné místo.

```
static double ZmerUsecku(double ax, double ay, double bx, double by)
{

}
```

Úloha 2: Při celkovém hodnocení školního prospěchu žáka platí:

- Má-li žák studijní průměr nejvýše 1,5 a nemá žádnou známku horší než 2, prospěl s vyznamenáním.
- Nesplňuje-li žák kritérium výše a není-li v žádném předmětu hodnocen známkou 5, prospěl.
- Je-li žák alespoň v jednom předmětu hodnocen známkou 5, neprospěl.

Doplňte tělo metody tak, aby vyhodnotila přijaté parametry (studijní průměr a nejhorší známku) a vrátila celkové hodnocení žáka.

```
static string ZjistiHodnoceni(double studijniPrumer, int nejhorsiZnamka)
{

}
```

Úloha 3: Vytvořte metodu, která se dotáže uživatele na datum jeho narození a podle toho vrátí současný věk uživatele jako celé číslo (zaokrouhlený dolů). Metoda bude přijímat pouze datum narození uživatele, současné datum zjistí automaticky.

Úloha 4: Využijte přetěžování metod a do souboru s řešením úlohy 1 doplňte metodu, která kromě souřadnic dvou bodů přijme i přirozené číslo *n*, které vyjadřuje počet desetinných míst, na které má být výsledek (vzdálenost bodů) zaokrouhlen. V případě, že číslo *n* nebude zadáno, provede se původní metoda (výsledek bude zaokrouhlen na 1 desetinné místo).

10.7 Okenní aplikace

Úloha 1: Vytvořte okenní aplikaci, která po zadání teploty vody a potvrzení kliknutím na tlačítko vypíše skupenství vody při této teplotě.

11 ŘEŠENÍ SBÍRKOVÝCH ÚLOH

11.2 Proměnné, výpis a načtení údajů

Úloha 1:

a) `Console.WriteLine("Vítejte v matematickém kvízu");`

Vypisovaný text se zapisuje v uvozovkách.

b)
```
string jmeno = "David";
int uspesnost = 2;
Console.WriteLine("{0} dosáhl úspěšnosti {1} %", jmeno, uspesnost);
```

Čísla s významem zástupných značek se uvádí ve složených závorkách.
Názvy dosazovaných proměnných oddělujeme čárkou.

c)
```
string hora = "Sněžka";
int vyska = 1603;
Console.WriteLine(hora + " je vysoká " + vyska + " metrů.");
```

Názvy proměnných vkládaných do vypisovaného textu uvádíme bez uvozovek.

Úloha 2:

a) `string jmeno = Console.ReadLine();`

Správný zápis příkazu pro načítání údajů z konzole je `Console.ReadLine()`.

b) `string nazevKnihy = Console.ReadLine();`

Příkazu pro načítání údajů z konzole je nutné uvést včetně složených závorek.

c) `int cena = int.Parse(Console.ReadLine());`

Správný zápis příkazu pro načítání údajů z konzole je `Console.ReadLine()`.

d) `string jmeno_prijmeni = Console.ReadLine();`

Znak & se nesmí vyskytovat v názvu proměnné. Název proměnné je lze zapsat pomocí podtržítka `jmeno_prijmeni` nebo využitím velbloudí konvence `jmenoPrijmeni`.

e) `int den1 = int.Parse();`

Název proměnné nesmí začínat číslicí. Místo `1den` zvolíme např. název `den1`.

Úloha 4:

a) c = 7 b) c = 7,5 c) c = 7

d) Program skončí s chybou. Výsledkem dělení je desetinné číslo, které nelze uložit do proměnné datového typu `int`.

Úloha 8: Uvedeny jsou pouze části kódu, které budou doplněny na označené místo.

a)
```
int c = a;
a = b;
b = c;
```

b)
```
a = a + b;
b = a - b;
a = (a - b)/2;
b = a + b;
```

11.3 Podmínky

Úloha 2:

```
    if (heslo == "a7P4q6@BV")
```

Podmínka pro testování rovnosti čísel nebo shody řetězců se zapisuje pomocí dvou rovná se. Řádek se zápisem podmínky bude vypadat tímto způsobem. Zbytek kódu je v pořádku.

Úloha 3:

```
static void Main()
{
    Console.Write("Zadejte celé číslo: ");
    int cislo = int.Parse(Console.ReadLine());
    if (cislo % 2 == 0)
    {
        Console.WriteLine("Číslo je dělitelné dvěma.");
    }
    else if (cislo % 5 == 0)
    {
        Console.WriteLine("Číslo je dělitelné pěti.");
    }
    else if (cislo % 7 == 0)
    {
        Console.WriteLine("Číslo je dělitelné sedmi.");
    }
}
```

Zadání úlohy vyžaduje využití trojice samostatných podmínek (tedy bez příkazu *else*). Pokud použijeme příkazy *else*, nedochází při splnění první (nebo druhé) podmínky k vyhodnocení dalších podmínek.

11.4 Cykly

Úloha 1: a) 435 Kč, b) v únoru 2022, c) v srpnu 2026

Úloha 2: a) 55, b) 1050, c) 2500, d) 735, e) 5445, f) 3554

Úloha 7:

```
static void Main()
{
    Console.WriteLine("Vítejte v programu pro výpočet průměru známek!");
    Console.Write("Kolik známek budete zadávat? ");
    int pocet = int.Parse(Console.ReadLine());
    int soucet = 0;
    for(int i = 0; i < pocet; i++)
    {
        int znamka = int.Parse(Console.ReadLine());
        if (znamka > 0 && znamka < 6)
        {
            soucet = soucet + znamka;
        }
        else
        {
            Console.WriteLine("Zadali jste neplatnou hodnotu!");
        }
    }
    double prumer = (double)soucet/pocet;
    double zaokrouhlenyPrum = Math.Round(prumer, 2);
    Console.WriteLine("Průměr zadaných známek je: " + zaokrouhlenyPrum);
}
```

Abychom mohli provádět inkrementaci proměnné, musíme do ní při založení uložit nějakou hodnotu.
V podmínce ověřující, zda jako známka bylo zadáno platné číslo, je třeba uvést operátor *a zároveň* (v případě využití operátoru *nebo* bude podmínka vždy splněna).
Pokud chceme, aby výsledkem dělení bylo desetinné číslo, musí být dělenec nebo dělitel také desetinné číslo – proto provedeme přetypování dělence na desetinné číslo.

Úloha 11:

```
    for(int i = mensi; i > 1; i--)
```

Řádek s inicializací cyklu *for* je třeba upravit tímto způsobem. V původní verzi např. při zadání zlomku 4/20 program zlomek nejprve zkrátí číslem 2 na 2/10, krácení číslem 2 neopakuje a snaží se krátit vyššími čísly – zlomek tak už zůstane nezkrácený. Opravíme-li uvedené řádek cyklu *for*, program uvedeným nedostatkem trpět nebude.

11.5 Pole a seznam

Úloha 2: Programátor zapsal příkaz pro vytvoření pole hodnot, ale dopustil se v něm chyby. Odhalte tuto chybu a opravte ji.

a) `string[] predmety = { "Český jazyk", "Matematika", "Chemie" };`
Při vytváření pole uvádíme prvky pole do složených závorek

b) `int[] teploty = { 20, 17, 15, 18, 23};`
V deklaraci pole musí být uvedené složené závorky.

c) `string otazky = new string[8];`

Vytváříme-li prázdné pole s daným počtem prvků, je třeba použít klíčové slovo `new`.

d) `string[] jmena = { "Karel", "Jana", "Markéta", "David" };`

Vytváříme-li pole textových řetězců, jednotlivé hodnoty musejí být v uvozovkách.

e) `int[] znamky = new int[10];`

Při vytváření pole uvádíme počet prvků do hranatých závorek.

Úloha 3:

```
static void Main()
{
    string[] obdobi = { "jaro", "leto", "podzim", "zima" };
    Console.WriteLine("Zadejte číslo ročního období: ");
    int cislo = int.Parse(Console.ReadLine());
    string zvoleneObdobi = obdobi[cislo - 1];
    Console.WriteLine("{0}. roční období je {1}", cislo, zvoleneObdobi);
}
```

Při deklaraci pole je třeba použít hranaté závorky.
Řetězec vypisovaný pomocí metody `Console.WriteLine()` uvádíme do uvozovek.
Pokud načítáme číselnou hodnotu, musíme ji po načtení převést na odpovídající dat. typ.
K prvku pole přistupujeme pomocí hranatých závorek.
Za příkazem na konci řádku uvádíme středník.

Úloha 4:

```
static void Main()
{
    string[] cesky = { "pes", "kočka", "prase", "ryba", "včela", "lev" };
    string[] anglicky = { "dog", "cat", "pig", "fish", "bee", "lion" };
    Random generator = new Random();
    int nahodneCislo = generator.Next(0,6);
    string slovicko = cesky[nahodneCislo];
    Console.WriteLine("Přeložte do angličtiny:");
    Console.WriteLine(slovicko);
    string odpoved = Console.ReadLine();
    if (odpoved == anglicky[nahodneCislo])
        Console.WriteLine("Správně!");
    else
        Console.WriteLine("Chyba!");
}
```

V původním kódu se generuje náhodné číslo dvakrát – při zobrazení českého slova i při kontrole překladu. Dochází tak k tomu, že překlad zadaného českého slova nekontrolujeme s odpovídajícím anglickým slovem. Řešením je vygenerovat náhodné číslo, které uložíme do proměnné. Hodnotu v proměnné použijeme pro vygenerování slova i pro následnou kontrolu odpovědi.

11.6 Metody

Úloha 1:

```
static double ZmerUsecku(double ax, double ay, double bx, double by)
{
    double delkaUsecky = Math.Sqrt( (bx-ax)*(bx-ax) + (by-ay)*(by-ay) );
    delkaUsecky = Math.Round(delkaUsecky, 1);
    return delkaUsecky;
}
```

Úloha 2:

```
static string ZjistiHodnoceni(double studijniPrumer, int nejhorsiZnamka)
{
    string hodnoceni;
    if(studijniPrumer <= 1.5 && nejhorsiZnamka == 2)
    {
        hodnoceni = "Prospěl s vyznamenáním";
    }
    else if(nejhorsiZnamka != 5)
    {
        hodnoceni = "Prospěl";
    }
    else
    {
        hodnoceni = "Neprospěl";
    }
    return hodnoceni;
}
```

ZÁVĚR

Účelem této práce bylo vytvořit učební text a sbírku pro výuku programovacího jazyka C# na střední škole. Sbírka byla vytvořena tak, aby žáci při učení se jazyku C# využili znalosti blokového programovacího jazyka, se kterým se seznámili na základní škole. Učební text pro lepší pochopení ilustruje popisovanou teorii na mnoha příkladech. Sbírka úloh obsahuje ke každé kapitole učebního textu několik úloh, na nichž si žáci téma procvičí a ověří, že ho pochopili.

Přílohou práce jsou také pracovní listy, ve kterých žáci aplikují znalost programování v jazyce C# při řešení úloh z matematiky. Při tvorbě práce jsem využil zkušenosti z vedení kroužku programování na Gymnáziu Ladislava Jaroše v Holešově a na ZŠ U Sýpek v Kroměříži. Vytvořené podklady jsem otestoval také na Gymnáziu Kroměříž a Obchodní akademii v Kroměříži. Rád bych těmto školám vyjádřil poděkování za příležitost vyzkoušet práci s vytvořenými materiály ve výuce.

SEZNAM POUŽITÉ LITERATURY

[1] ČANDÍK, Marek a Štefan CHUDÝ. *Didaktika informatiky* [online]. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005 [cit. 2024-05-09]. ISBN 80-7318-285-8.

[2] KAŠPÁRKOVÁ, Svatava. *Učení a vyučování* [online]. Zlín: Univerzita Tomáše Bati ve Zlíně, 2013 [cit. 2024-05-09]. ISBN 9788074542985. Dostupné z: http://hdl.handle.net/10563/25822

[3] *Rámcový vzdělávací program pro základní vzdělávání* [online]. Jednotný metodický portál MŠMT, Praha, 2013 [cit. 2024-03-06]. Dostupné z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcovy-vzdelavacici-program-pro-zakladni-vzdelavani-rvp-zv/

[4] *Rámcový vzdělávací program pro gymnázia* [online]. Jednotný metodický portál MŠMT, Praha, 2007 [cit. 2024-03-08]. Dostupné z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcove-vzdelavaci-programy-pro-gymnazia-rvp-g/

[5] Rámcové vzdělávací programy středního odborného vzdělávání. *Jednotný metodický portál MŠMT* [online]. c2022 [cit. 2024-03-10]. Dostupné z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcove-vzdelavaci-programy-stredniho-odborneho-vzdelavani-rvp-sov/

[6] *Rámcový vzdělávací program pro základní vzdělávání* [online]. Jednotný metodický portál MŠMT, Praha, 2023 [cit. 2024-03-06]. Dostupné z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcovy-vzdelavacici-program-pro-zakladni-vzdelavani-rvp-zv/

[7] Programování ve Scratch pro 2. stupeň základní školy. *Informatické myšleny* [online]. 2021-05-01 [cit. 2024-05-02]. Dostupné z: https://imysleni.cz/ucebnice/programovani-ve-scratchi-pro-2-stupen-zakladni-skoly

[8] BORY, Pavel. *C# bez předchozích znalostí*. 2. vydání. V Brně: Computer Press, 2022. ISBN 978-80-251-5061-0.

[9] VYSTAVĚL, Radek. *Moderní programování: učebnice pro začátečníky*. 5. vydání. Ondřejov: moderníProgramování, 2019. ISBN 978-80-903951-9-0.

[10] VYSTAVĚL, Radek. *Moderní programování: sbírka úloh k učebnici pro začátečníky*. 2. vyd. Ondřejov: moderníProgramování, 2019. ISBN 978-80-903951-5-2.

[11] VYSTAVĚL, Radek. *Moderní programování: učebnice pro středně pokročilé*. Ondřejov: moderníProgramování, 2008. ISBN 978-80-903951-2-1.

[12] VYSTAVĚL, Radek. *Moderní programování: sbírka úloh k učebnici pro středně pokročilé*. Ondřejov: moderníProgramování, 2009. ISBN 978-80-903951-3-8.

[13] Základní konstrukce jazyka C# .NET. *ITnetowork.cz* [online]. [cit. 2024-03-20]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady

[14] C# Tutorial. *W3Schools* [online]. [cit. 2024-05-03]. Dostupné z: https://www.w3schools.com/cs/index.php

[15] BUŠEK, Ivan a Emil CALDA. *Matematika pro gymnázia: základní poznatky*. 4. vyd. Praha: Prometheus, 2008. Učebnice pro střední školy. ISBN 80-7196-146-9.

[16] ČERMÁK, Libor a Rudolf HLAVIČKA. *Numerické metody*. Vydání třetí. Brno: Akademické nakladatelství CERM, 2016. ISBN 978-80-214-5437-8.

[17] POLÁK, Josef. *Přehled středoškolské matematiky*. 10. vydání. Praha: Prometheus, 2015. ISBN 978-80-7196-458-2.

[18] HOROVÁ, Ivana a Jiří ZELINKA. *Numerické metody*. 2., rozš. vyd. Brno: Masarykova univerzita v Brně, 2004. ISBN 80-210-3317-7.

[19] KOČANDRLE, Milan a Leo BOČEK. *Matematika pro gymnázia: Analytická geometrie*. 3. vydání. Praha: Prometheus, 2009. Učebnice pro střední školy (Prometheus). ISBN 978-80-7196-390-5.

[20] ODVÁRKO, Oldřich. *Matematika pro gymnázia*. 2., upravené vydání. Praha: Prometheus, 2004. ISBN 80-7196-195-7.

[21] CALDA, Emil a Václav DUPAČ. *Matematika pro gymnázia*. 5. vydání. Praha: Prometheus, 2008. Učebnice pro střední školy (Prometheus). ISBN 978-80-7196-365-3.

[22] VIRIUS, Miroslav. *Programování v C#: od základů k profesionálnímu použití*. Praha: Grada Publishing, 2021. Myslíme v.. ISBN 978-80-271-1216-6.

[23] PŠENČÍKOVÁ, Jana. *Algoritmizace*. Vydání druhé. Prostějov: Computer Media, 2021. ISBN 978-80-7402-414-6.

[24] HYLMAR, Radek. *Programování pro úplné začátečníky*. 2. vydání. V Brně: Computer Press, 2022. ISBN 978-80-251-5059-7.

[25] Scratch. *Scratch Wiki* [online]. last edited on 8 February 2024 [cit. 2024-04-06]. Dostupné z: https://en.scratch-wiki.info/wiki/Scratch

[26] Scratch Wiki: Program. *Scratch Wiki* [online]. [cit. 2024-04-12]. Dostupné z: https://en.scratch-wiki.info/wiki/Scratch_Wiki:Table_of_Contents/Program

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

RVP	Rámcový vzdělávací program.
ŠVP	Školní vzdělávací program.
RVP ZV	Rámcový vzdělávací program pro základní vzdělávání
RVP G	Rámcový vzdělávací program pro gymnázia
RVP SOV	Rámcový vzdělávací program středního odborného vzdělávání
IKT	Informační a komunikační technologie (oblast RVP)
INF	Informatika (oblast RVP)
C#	Programovací jazyk

SEZNAM PŘÍLOH