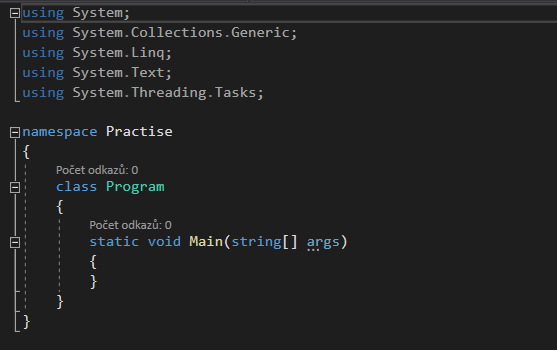
Pracovní list – ONBOARDING

Následující dokument slouží jako počáteční dokument pro ztracené duše a nováčky v předmětu programování. Obsahuje popis jednotlivých části pro vytváření konzolových aplikací v jazyce C# v prostředí Visual Studio 2019. Po každé kapitole následuje krátké cvičení s řešením.

# Úvod

* Popis základní struktury kódu aplikace
* Jednořádkové a více řádkové komentáře
* Deklarace proměnné
* Přiřazení hodnoty
* Zobrazení proměnné na výstup
* Načtení hodnoty ze vstupu

## Základní struktura kódu aplikace



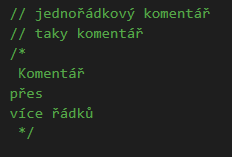
Hlavní funkce programu – do jejího těla píšeme kód

Definice třídy konzolové aplikace

Seznam knihoven obsahující použitelné funkce

## Komentáře

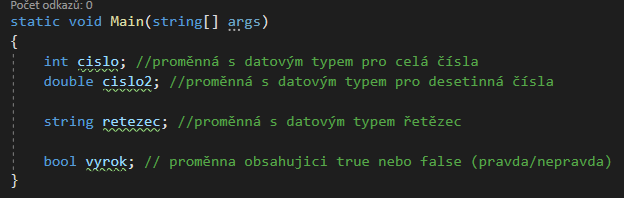
Komentáře nám umožňuje vytvořit poznámky v kódu, kterou kompilátor při překladu ignoruje a nemá vliv na výslednou aplikaci. Pro jednořádkový komentář použijeme symbolu //. Pro více řádkový komentáře je text v komentáři třeba ohraničit pomocí /\* a \*/.



## Deklarace proměnné

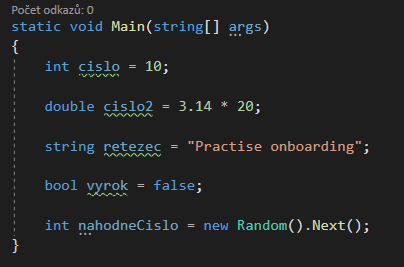
Hodnoty nebo informace, se kterými pracujeme potřebujeme v průběhu programu ukládat, abychom pak s nimi mohli dále pracovat. Využíváme k tomuto účelu proměnné. Při deklaraci proměnné je nutné uvést jakého typu je (jednoduchý datový typ, struktura, enumerace, třída, …) a její název, na který se budeme odkazovat. Název proměnné se nesmí opakovat a musí dodržovat pravidla pro pojmenování.

Poznámka: Symbol středníku ; používáme na konci každého jednoduchého nebo složeného příkazu. Středník nepíšeme v případě, že část kódu je ohraničena složenými závorkami { }.



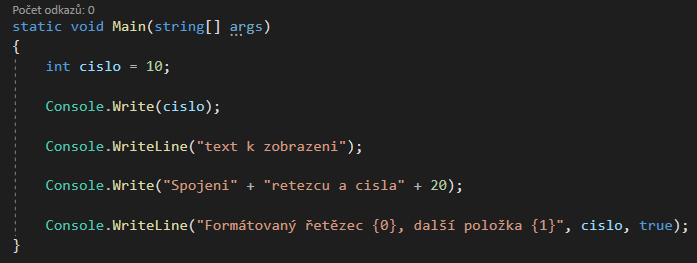
## Přiřazení hodnoty do proměnné

Deklarovaná proměnná v lepším případě obsahuje nějakou výchozí hodnotu, v horším případě obsahuje nějaký zbylá data z posledního spuštění. Pro přiřazení hodnoty do proměnné užíváme operátoru **=**. Přiřadit můžeme konkrétní hodnotu/informaci, složený výraz případně do ní můžeme vrátit výsledek volání funkce.

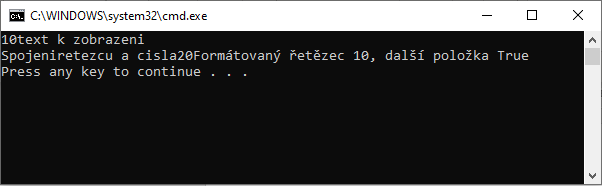


## Zobrazení na výstup

Tím, že vytváříme konzolovou aplikaci jako komunikační médium nám slouží klasická konzole případně terminál. Pro práci s konzolí využíváme funkcí třídy **Console**, které je již součástí základní knihovny, která se nám vkládá při vytvoření nového projektu. Pro vypsání používáme příkazu **Console.WriteLine();** pro vypsání textu na řádek a následném odřádkování nebo **Console.Write();** pro výpis na řádek bez odřádkování. Do kulatých závorek, vkládáme vstupní parametry jednotlivých funkcí. V tomto případě je vstupním parametrem je text v uvozovkách případně proměnná typu string. Ostatní datové typy se v tomto případě automaticky převedou na řetězec. Případně můžeme využít spojování pomocí operátoru + nebo formátovaného řetězce, kdy jsou po kompilaci nahrazené pozice indexu {X} příslušnou hodnotou.



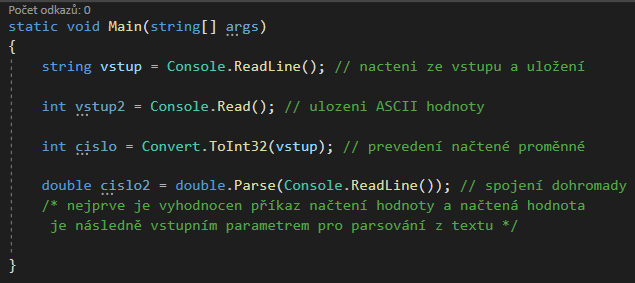
Na konzoli se nám po spuštění zobrazí předešlá kód následovně (všimněte si rozdílu, kdy používáme funkci s odřádkováním a kde ne):



Poznámka: Pro spuštění můžeme využít zeleného tlačítka PLAY v horní části, konzole se nám ovšem po doběhnutí programu sama zavře. Případně pomocí CTRL + F5 kdy se po doběhnutí zobrazí hláška jako na ukázce výše.

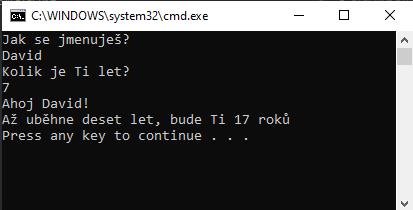
## Načtení hodnoty ze vstupu

Obdobně jako pro výpis budeme pracovat s třídou **Console**. Pro načtení lze použít příkazů **Console.ReadLine();** nebo **Console.Read();** Funkce ReadLine nám přečte celý vstup až do odřádkování pomocí Enter. Funkce načtený vstup vrací jako řetězec. V případě, že jsme zadali číslo a chceme s ním pracovat jako s číslem je třeba řetězec převést. Funkce Read nám přečte jeden znak ze vstupu a uloží jeho hodnotu z ASCII tabulky – celé číslo (datový typ int). Pro převedení používáme funkci **ConvertToX()**, kde X je nahrazeno konkrétním datovým typem případně **X.Parse()**, kde X je rovněž potřebný datový typ do kterého chceme hodnotu převést. Vstupním parametrem funkcí na převedení je řetězec (string).

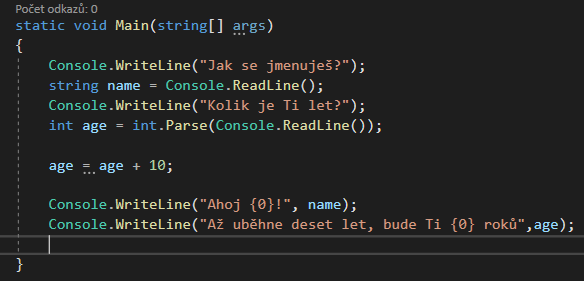


### Cvičení 1 - zadání:

Vytvořte aplikaci, která ze vstupu načte jméno uživatele a jeho věk. Následně jej pozdraví s oslovením a napíše kolik roků bude mít za deset let. Příklad výstupu z aplikace na konzoli:



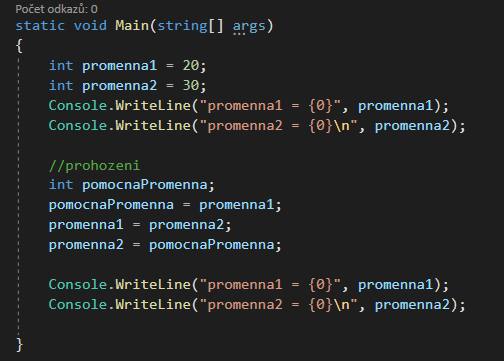
### Cvičení 1 – řešení:



### Cvičení 2 – zadání:

Vytvořte aplikaci, která prohodí obsah v proměnných promenna1 a promenna2. Aplikaci můžete rozšířit o načtení hodnot od uživatele.

### Cvičení 2 – řešení:

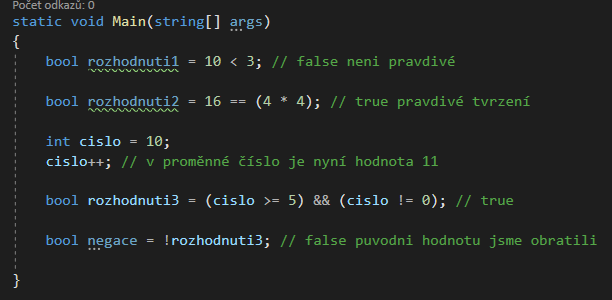


# Základní konstrukce a datové struktury

* Operátory
* Pole a seznamy
* Rozhodovací blok (if-else)
* Cykly s neznámým počtem opakování
* Cykly s pevným počtem opakování

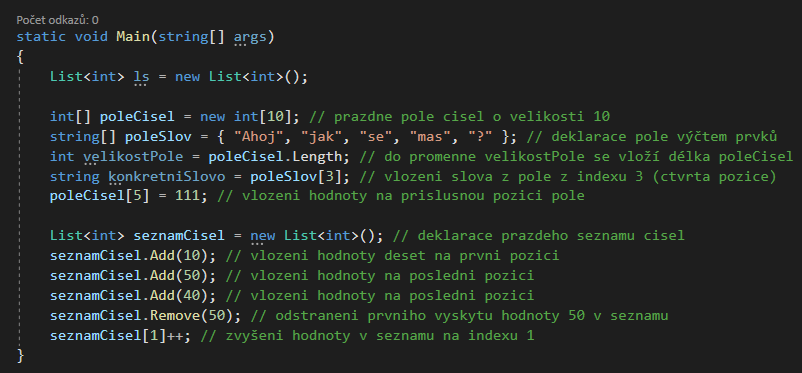
## Operátory

Pro operaci s hodnotami nebo proměnnými používáme operátory. Matematické operátory fungují stejně jako v matematice. Proto si představíme další operátory, které lze využít. Již v předešlé kapitole jsme se bavili o operátoru přiřazení = a spojování řetězců +. Dalšími známými operátory jsou porovnávací operátory <=, <, >=, > k těmto operátorům přidáme ještě == pro porovnání a != pro nerovnost výrazů a máme tímto pokryté operátory relační. Výsledkem relačních operátorů je hodnota datového typu bool, která nabývá dvou možností true/false. Dalšími typy operátorů jsou operátory logické, které nám rozhodují o pravdivosti několika tvrzení dohromady. Používáme && pro logický součin (AND) a || pro logický součet (OR) a jako poslední ! (negace), kterým obracíme platnost tvrzení. Posledními operátory, se kterými se často setkáváme je operátor inkrementu ++ a dekrementu --, které nám dané číslo v proměnné zvýšit/snížit o 1.



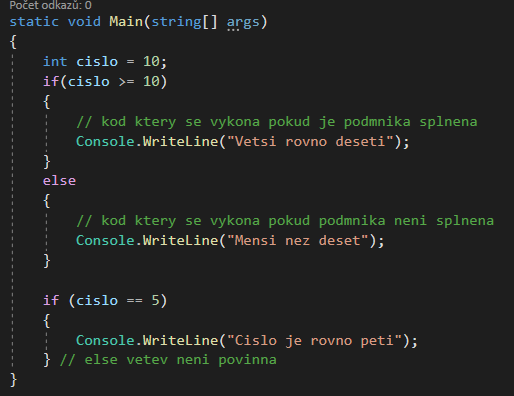
## Pole a seznamy

Oba výrazy nám reprezentují datové struktury, které nám pod jedním názvem umožňují uchovávat více hodnot stejného datového typu. Hlavní rozdíl je, že pole jsou statická a v momentě, kdy určíme jejich velikost se v průběhu programu jejich velikost nezmění. Zatímco u seznamů se velikost mění v závislosti na počtu hodnot, které obsahují. Pro pole i seznamy platí základní pravidlo, že indexujeme od 0. Tedy první prvek ať již v poli nebo seznamu se nachází na indexu 0. Pokud potřebujeme zjistit jaké je délka pole potažmo kolik položek je v seznamu, lze k tomu využít příslušné atributy. Pro pole se jedná o atribut **Length** a pro seznam atribut **Count**. Pro vkládání hodnot do pole využíváme přiřazovacího operátoru a lze přistoupit hned na libovolný prvek pole[X] kde X je příslušný index. U seznamu je třeba využít vlastních funkcí **Add()** přidání na poslední pozici a **Remove()** případně **RemoveAt()** pro odebrání položky. Na konkrétní položku v seznamu funkce jako taková neexistuje, ale můžeme na ni přistoupit, jako by se jednalo o pole tedy opět seznam[X], kde X je pozice v seznamu.



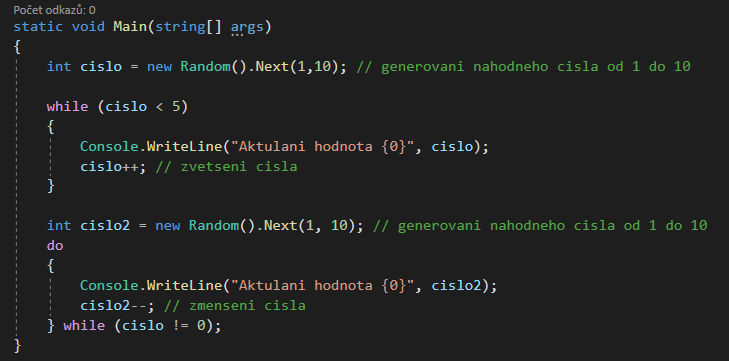
## Rozhodovací blok (if-else)

Základní prvek, který umožňuje na základě podmínky, která musí být vyhodnotitelná (výsledek je true nebo false), rozhodnout, která část kódu se má vykonat.



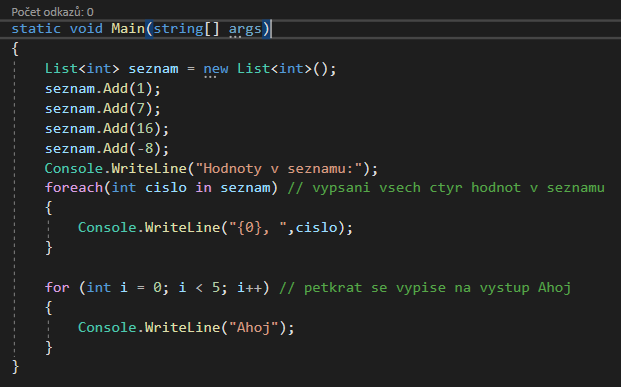
## Cykly s neznámým počtem opakování

V těchto cyklech není při kompilaci jisté, kolikrát se má tělo cyklu provést. Takovéto cykly známe dva: **do-while** a **while**. Rozdíl mezi těmito cykly je v tom, kdy se kontroluje podmínka, které rozhoduje, zda se má tělo programu vykonávat. Pokud je podmínka splněna, tělo cyklu se vykoná. Vykonávat se bude, dokud je podmínka platná. V opačném případě kód pokračuje dál za cyklem. Cyklus do-while má podmínku až na konci, a tudíž se vždy alespoň jednou vykoná tělo cyklu. Zatímco cyklus while má podmínku na začátku a tělo cyklu se tak nemusí nikdy vykonat.



## Cykly s pevným počtem opakování

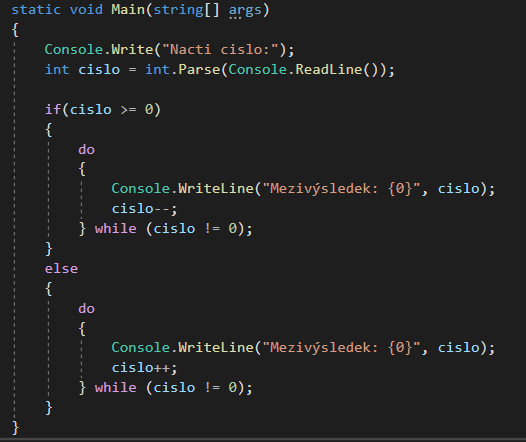
U těchto cyklu lze říct, kolikrát se tělo vykoná. V zásadě známe dva typy, kdy oba fungují stejně, ale každý je vhodnější na něco jiného. První obyčejný cyklus s pevným počtem opakování je **for**. Tento cyklus má ve své definici uvedenu řídící proměnou, dále podmínku související s řídící proměnnou a následně příkaz, co se má s řídící proměnnou stát po jednom cyklu. Druhým typem cyklu je **foreach**, který je vhodný na různé kolekce. Kolekcí je třeba pole případně seznam. Ve své definici říká vykonávej konkrétní tělo cyklu pro každou položku v nějaké kolekci. Tento cyklus tak nemá řídící proměnnou a počet opakování je dán právě počtem prvků v dané kolekci.



### Cvičení 3 – zadání:

Vytvořte aplikaci, která načte číslo. Pokud bude číslo kladné včetně nuly, tak budeme postupně číslo snižovat až do 0 a mezihodnoty vypisovat. Pokud číslo bude záporné, budeme je zase do 0 zvyšovat a opět mezivýsledky zobrazovat na výstup.

### Cvičení 3 – řešení:



### Cvičení 4 – zadání:

Vytvořte aplikaci, která nejprve načte číslo, které bude odpovídat velikosti nového pole řetězců. Dle načtené velikosti pole vytvořte nové pole a nechte jej uživatelem naplnit pomocí for cyklu. Jakmile je pole načtené, vypište pomocí foreach cyklu délky každého slova v poli.

