

# Programování

Chyby v programu, ovládání VS, algoritmy a diagramy, datové struktury

# Typová konverze

- ▶ Dle datového typu proměnné víme, jaký typ hodnoty můžeme očekávat
- ▶ Datový typ nám i omezuje, jakou hodnotu můžeme do proměnné vložit
  - ▶ Např. nemůžeme do proměnné, která drží číselnou hodnotu vkládat řetězec. Stejně tak nemůžeme do proměnné, která nám drží celé číslo, vložit číslo desetinné
- ▶ Pokud to opravdu silou chceme, používáme **KONVERZE**
- ▶ Konverze nám umožňuje změnit datový typ příslušné hodnoty
- ▶ V obecnosti má tvar: **(požadovaná hodnota) výraz**
  - ▶ Výrazem může být samotná hodnota případně libovolný výpočet nebo volání funkce
- ▶ Během konverze může docházet ke ztrátě informace
- ▶ Speciálně lze nad objekty a proměnnými volat funkci **ToString()** k převedení na řetězec

# Pole [ ]

- ▶ Statická homogenní datová struktura
  - ▶ umožňuje uchovávat více hodnot stejného datového typu
  - ▶ Po stanovení velikosti pole již nelze v průběhu měnit počet prvků
- ▶ Dle datového typu, jednotlivých položek pole lze využívat speciální funkce - řazení, suma, průměr, vyhledávání, ...
- ▶ Pro práci s konkrétní položkou musíme uvést její index (**začínáme od 0**)

```
double[] novePole = new double[4];

bool[] pravdivostniPole = { true, false, true, true, false };

string[] poleSlov = { "ahoj", "svete", "jsem", "tady" };

int[] poleCisel = new int[3];
poleCisel[0] = 1;
poleCisel[2] = 4;
```

# List < >

- ▶ Dynamická homogenní datová struktura
- ▶ Oproti poli nedefinujeme jeho velikost - ta se mění podle počtu prvků
- ▶ Pro práci s listem využíváme funkce **Add()** pro přidání a **Remove()** případně **RemoveAt()** pro odebrání - nová položka se přidává vždy na konec
- ▶ Pro editaci prvku na příslušném poli využíváme indexace stejně jako u pole
- ▶ Pokud nám stačí zjistit hodnotu na příslušném indexu, lze využít funkce **ElementAt()**

```
List<int> listCisel = new List<int>();  
  
listCisel.Add(1);  
listCisel.Add(17);  
listCisel.Add(1);  
listCisel.Add(3);  
  
listCisel.Remove(1);  
listCisel.RemoveAt(2);  
  
listCisel[1] = 500;
```

Odstranění prvního výskytu

Odstranění na pozici

# Ošetření výjimek v běhu programu

- ▶ Pro kritickou sekci kódu je vhodné mít vymyšleno, co se má stát pokud se vyskytne problém
- ▶ **try-catch** funguje jako pískoviště, kde si v sekci **try** necháme proběhnout kód, kde se může objevit výjimka (Exception) v části **catch** pak říkáme, co se má stát
- ▶ Mimo výjimky, které již existují lze vyhazovat i výjimky vlastní pomocí **throw**

```
int[] array = { 1, 2, 3, 4 };  
  
try  
{  
    Console.WriteLine(array[5]);  
}  
catch (IndexOutOfRangeException e)  
{  
    Console.WriteLine("Nedostupný index pole");  
}  
  
throw new Exception("Stejně vytvoříme chybu");
```



Pole nemá takový index !!!



Jaký je čas?  
Čas na Kahoot!

# Struktury

- ▶ Hodnotový typ, který zapouzdřuje jiné hodnoty
- ▶ Vhodné pro reprezentaci jednoduchých objektů
- ▶ Nemají zbytečnou složitost oproti objektům vytvořené z třídy
- ▶ Výhodou je vyšší rychlost při manipulaci oproti třídám
- ▶ Pro získání a úpravu hodnot jednotlivých položek struktur využíváme příslušných funkcí - **get**, **set**
  - ▶ Položky struktury musí začínat velkým písmenem pro odlišení od proměnných
- ▶ Oproti proměnné nemáme možnost nejprve strukturu deklarovat a následně ji přiřadit hodnotu - iniciální hodnota je vložena konstruktorem
- ▶ Pokud chceme vytvořit proměnnou, která bude obsahovat strukturu využijeme tzv. **konstruktoru**, který nám strukturu vytvoří a vyplní příslušnými hodnotami
- ▶ Struktury mohou mít své vlastní vnitřní funkce

# Enumerace

- ▶ = typ hodnoty definovaný sadou pojmenovaných konstant reprezentované jako celočíselná hodnota int
- ▶ Vhodné pokud chceme vytvořit výčet hodnot, které se budou v programu objevovat
- ▶ Přestože si jednotlivé proměnné pojmenujeme podle naší potřeby, kompilátor je čte jako hodnoty int - pokud skutečně chceme číselnou hodnotu je nutné provést konverzi (int) **Den.Pondeli**

```
public enum Den { Pondeli, Utery, Streda, Ctvrtek, Patek }
```

- ▶ Hodnoty jsou podle pořadí reprezentovány číslicemi od 0
- ▶ Pokud chceme můžeme reprezentované hodnoty změnit případně posunout

```
public enum Den { Utery = 2, Streda, Nedele = 7 }
```

- ▶ Volání výčtového typu: **Den.Utery**



# Postup vytváření algoritmu

- ▶ Formulace problému
  - ▶ Formulace požadavků, určení vstupů/výstupu a požadavků na přesnost
- ▶ Analýza úlohy
  - ▶ Ověření řešitelnosti a počtu řešení dle zadaných vstupů
- ▶ Vytvoření algoritmu
  - ▶ Sestavení sledu operací, které vedou k požadovanému výsledku
- ▶ Sestavení programu
  - ▶ Vytvoření zdrojového kódu v příslušném programovacím jazyce
- ▶ Odladění programu
  - ▶ Odstraňování logických a syntaktických chyb v programu

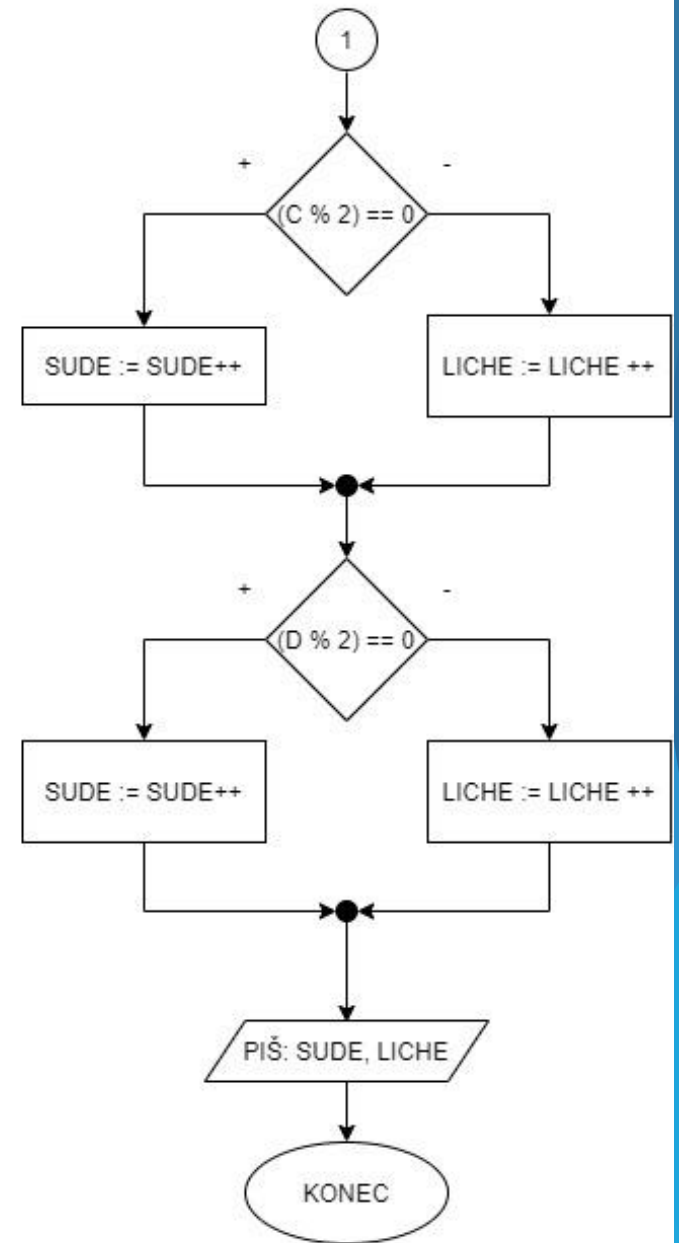
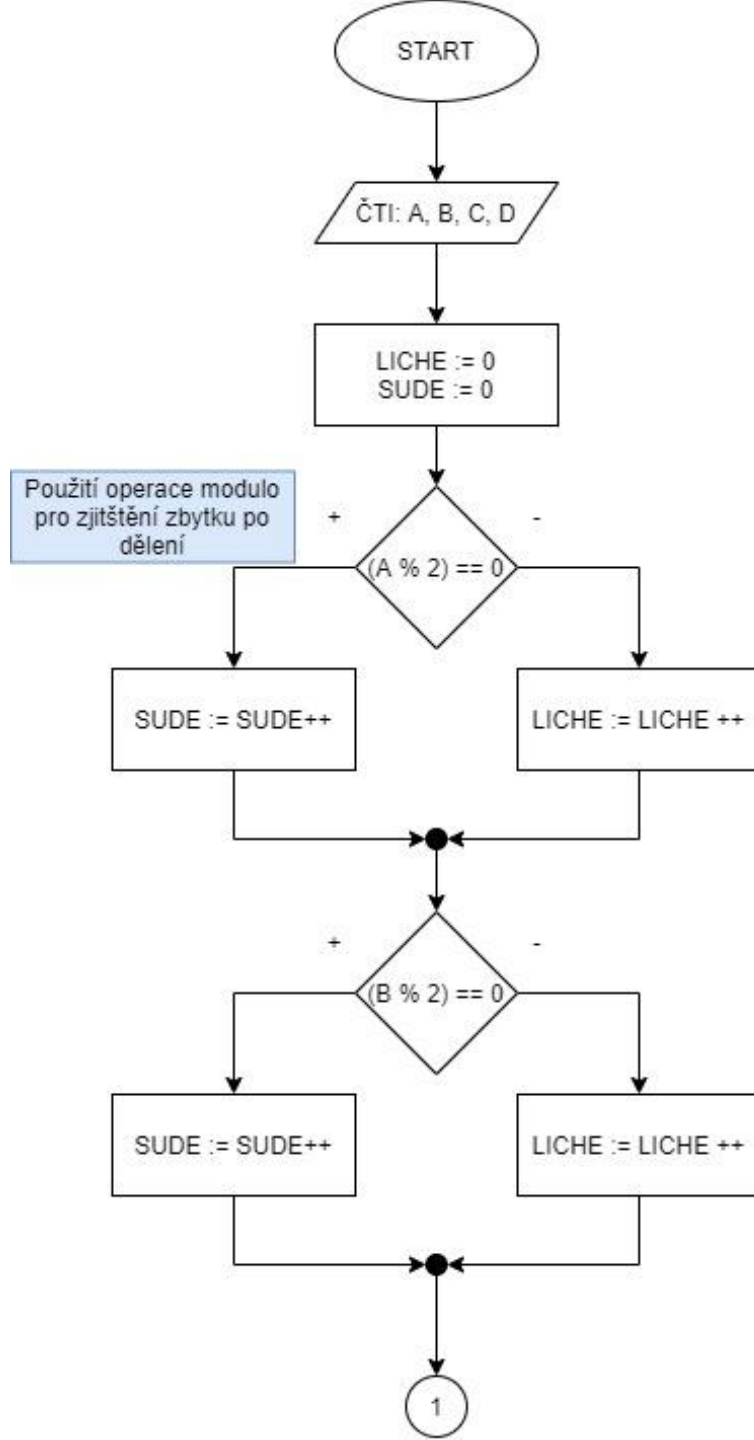
# Přehled používaných symbolů

	začátek/konec algoritmu		ruční vstup
	běžný příkaz		zobrazení výstupu
	podmíněné větvení		zpracování souboru
	cyklus s určeným počtem opakování		uložení dat do souboru
	cyklus s podmínkou na konci		podprogram
	cyklus s podmínkou na začátku		spojovací značka
			spojovací čára (tok algoritmu)

Používáme i pro vstupní instrukce

# Vývojový diagram

- ▶ Vždy obsahuje právě jeden začátek a konec
- ▶ Šipky nám pomáhají ve čtení toku programu
- ▶ Obsahem rozhodovacích podmínek jsou výrazy, o kterých lze říci zda jsou pravdivé
- ▶ Nejlépe užít výrazu stylu:  $X < Y$  nebo  $A == B$  apod.
- ▶ Proměnné, které v průběhu použijeme uvádíme až v momentě použití



# Práce s Visual Studiem

- ▶ Dle vytvářeného typu projektu se nám liší rozložení jednotlivých oken
- ▶ Rozdíl Konzolová vs. Formulářová aplikace
- ▶ Pokud nechceme, aby se nám konzole po doběhnutí programu sama zavřela - spouštíme pomocí CTRL + F5 místo zelené šipky spuštění
- ▶ V chybovém okně nás IDE upozorňuje na jakém řádku se chyba nachází včetně informace o jaký typ chyby se jedná
- ▶ Pokud si s chybou nevím rady hledám -> řešení je v 99% na internetu
  - ▶ Stačí řádek z okna chyb zkopírovat a vložit do vyhledávače
- ▶ Pokud se v programu nachází chyby -> kompilátor neumožní spuštění (maximálně nabídne spuštění poslední funkční sestavení), upozornění nevadí
- ▶ IDE nám hodně pomáhá (napovídá) co za kód má následovat nebo doporučí možnost úpravy
- ▶ Pokud nevíme, jak konkrétní funkce funguje, proklikem si zobrazím dokumentaci

Soubor Upravit Zobrazit Git Projekt Sestavit Ladit Test Hledat (Ctrl+Q) Con...pp4

Analyzovat Nástroje Rozšíření Okno Nápoředa

Debug Any CPU Spustit Live Share

Sada nástrojů Program.cs ConsoleApp4 ConsoleApp4.Program Main(string[] args)

```
20 case 1:
21     Console.WriteLine("ABC");
22     break;
23 case 2:
24     Console.WriteLine("BAC");
25     break;
26 case 3:
27     break;
28 case 4:
29     Console.WriteLine("BBC");
30     Console.WriteLine("CCA");
31     break;
32 }
33
34 double[] novePole = new double[4];
```

107 % 1 1 Řádek: 37 Znak: 13 SPC CRLF

Seznam chyb

Celé řešení 1 Chyba 1 Upozornění 0 Zpráv

Vyhledání v seznamu chyb

	Kód	Popis	Projekt	Soubor	Řá.
✖	CS1002	Očekával se středník (;).	ConsoleApp4	Program.cs	30
⚠	CS0168	Proměnná e je deklarovaná, ale nikdy se	ConsoleApp4	Program.cs	42

Seznam chyb Výstup

Okno kódu

Struktura projektu

Chybové okno,  
okno výstupu