# Protokol ke GUI aplikaci

**Předmět:** Úvod do programování a algoritmů

**Zadání:** Bilanční výpočty

**Vypracoval:** Marek Klíma

**Kruh:** 159

# Úvod

GUI aplikace byla vytvořena v rámci seminární práce z *Úvodu do programování a algoritmů* v *programu Microsoft Visual Studio 2017*. Zadáním bylo převést předmět bilanční a chemické výpočty do programu. Rozhodl jsem se vzhledem ke složitosti udělat pouze výpočty potřebné pro přípravu roztoků s limitací na pouze jeden uzel, což pokrývá většinu daného předmětu.

# Výpočty na přípravu roztoků

Rovnice, týkající se přípravy roztoků, lze rozdělit na dva typy – celkové bilance a rovnice, díky kterým budeme schopni vypočítat hodnoty pro celkové bilance.

Celkové bilance jsou celková hmotnost, celkové látkové množství, celková hmotnost složky a celkové látkové množství složky. Tyto čtyři hodnoty se musí shodovat na konci i na začátku. Dále jsem si musel přidat jednu rovnici do celkových bilancí z důvodu, že se vyskytla ve cvičném testu a žádná z aktuálních rovnic ji nemohla zastoupit. Touto rovnicí je součet hmotností, ze kterých proud vzniká v případě složky výstupního proudu a v případě vstupního proudu je to součet složek, které kam jdou.

U rovnic, které nám pomůžou určit hodnoty pro celkové bilance, jsme museli rozlišit, zda se jedná o plyn či nikoliv, jelikož některé rovnice platí pouze pro plyny, přičemž jsem všechny plyny bral jako dokonalé plyny.

# Popis řešení

Po spuštění se ukáže uživateli okno, na kterém má pět tlačítek, z toho tři zajišťují vstup uživatele. Čtvrté započne výpočet a poslední tlačítko vyresetuje vstupy uživatele.

Tlačítko *Nastavit* otevře okno, kde může uživatel nastavit počet složek a vstupních/výstupních proudů. Při zavření se uživatele program zeptá, jestli chce uživatel hodnoty uložit.

Uživatel následně kliknutím na vstup/výstup otevře okno, ve kterém nastaví hodnoty proměnných ve vstupních/výstupních proudech skrze interaktivní seznam.

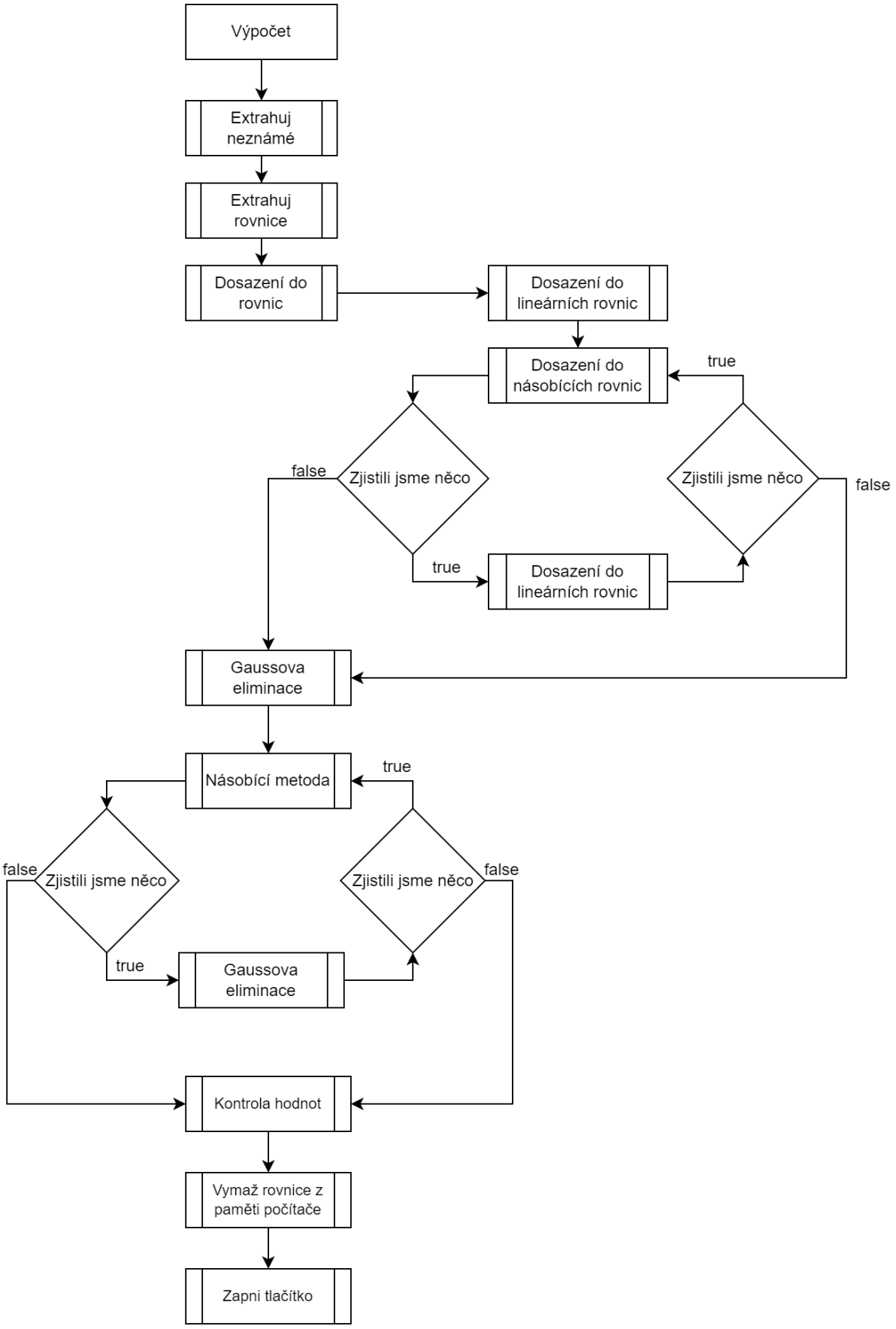
Po nastavení může uživatel kliknout na *Vypočítat* a program vypočítá všechno, čeho je jeho algoritmus schopen. Uživatel si může výsledky zobrazit v rámci interaktivního seznamu, do nějž před tím hodnoty zadal. Dříve než vůbec začne algoritmus počítat hodnoty neznámých, se uzamknou všechna tlačítka, aby uživatel nemohl nic náhodou spustit a narušit tím postup výpočtu.

Výpočet používá kombinaci *Gaussovy eliminace* a *metody pro úpravu soustavy násobících rovnic* o nezáporných činitelích (více v sekci matematika). Nejdříve si připraví neznámé a rovnice. Poté program začne dosazovat do rovnic. Nejdříve dosadí do lineárních rovnic, poté do násobících. Po prvním dosazení do obou se začne ptát, jestli bylo dosazení k něčemu a zdali jsme něco zjistili.

Rovnice pro *Gaussovu eliminaci* a *metodu úpravy násobících rovnic* beru z učiva předmětu Bilanční a chemické výpočty.

Jakmile již z dosazováním známých hodnot nelze dále nic zjistit, tak se pokusíme upravit rovnice *Gaussovou eliminaci* a *metodou pro úpravu násobících rovnic*. Tyto metody jsou použity podobným způsobem jako dosazování – nejdříve se provedou obě a pak se vždy po nich ptáme, jestli něco zjistily; pokud ano, tak pokračujeme; pokud nikoliv, tak pokračujeme dále.

Nakonec jen zkontrolujeme, jestli jsou hodnoty neznámých v přijatelných mezích (mezi maximem a minimem, které je od nich očekáváno) a uvolníme místo v paměti počítače vymazáním rovnic z paměti.


Výpočet
Extrahuj neznámé
Extrahuj rovnice
Dosazení do rovnic
Dosazení do lineárních rovnic
Dosazení do násobících rovnic
true
Zjistili jsme něco
false
Zjistili jsme něco
false
true
Dosazení do lineárních rovnic
Gaussova eliminace
Násobící metoda
true
false
Zjistili jsme něco
false
Zjistili jsme něco
true
Gaussova eliminace
Kontrola hodnot
Vymaž rovnice z paměti počítače
Zapni tlačítko

# Metoda pro úpravu násobících rovnic

Pro úpravu lineárních rovnic jsem si vybral *Gaussovu eliminaci*. Bohužel se zde nachází mnoho rovnic, ve kterých se vyskytuje násobení. Proto jsem si upravil *Gaussovu eliminaci*, aby řešila i tyto problémy. Tato metoda funguje pouze pro nezáporné základy. Naštěstí všechny neznámé jsou buďto kladné, nebo *0*.

Funguje na principu, že si převedeme rovnici, kde se nachází pouze násobení. Uložíme si do matice exponent u příslušné neznámé, přičemž u neznámých, které se nevyskytují, máme exponent automaticky nastaven na *0*. Do pole si uložíme výsledek této rovnice. Ukážeme si to na následné rovnici:

$$x \cdot 2 \cdot y^3 = z$$

Nejdříve ji upravíme do tvaru:

$$x \cdot y^3 \cdot z^{-1} = \frac{1}{2}$$

Následně uložíme koeficient u *x* jako *1*, u *y* jako *3*, u *z -1* (u všech, které nejsou v této rovnici zmíněny, nastavíme jako *0*) a jako výsledek dáme hodnotu *½*. Však si můžeme všimnout, že touto úpravou jsme si přidali podmínku, že *z* se nesmí rovnat *0*. Toto se stává velice často u našich rovnic a všechny tyto podmínky jsou tímto způsobeny (mimo podmínku nenulovosti termodynamické teploty. Zde je důvod podmínky fyzikální.) Toto budeme zohledňovat dále v metodě.

Jakmile máme matici s exponenty a pole s výsledky připravené, započneme samostatnou metodu. Postupně projíždíme všechny sloupce a koukneme se, jestli se zde nachází nenulový exponent. Pokud ano, tak tuto rovnici i s výsledkem dáme na začátek pod již takto uložené rovnice. Načež rovnicí i s výsledkem vydělíme všechny ostatní rovnice a výsledky tak, aby exponent u naší neznáme byl *0* ve všech rovnicích mimo naší uložené.

Zde si však musíme dát pozor, abychom nenásobili rovnicí, která se rovná *0*, jelikož bychom výsledky rovnic dělili *0*. Tyto rovnice upravujeme, ale jimi nic neupravujeme (v mém kódu si je pro přehlednost ukládám jako poslední rovnice v matici).

Z rovnic, které se rovnají *0*, můžeme vyškrtat záporné exponenty, jelikož víme, že čitatel musí být *0* a v tomto tvaru nevíme nic o jmenovateli.

Po této úpravě chceme dostat hodnoty jednotlivých koeficientů. Pokud máme pouze jeden nenulový exponent v rovnici, tak hodnota dané proměnné je rovna:

$$neznámá = výsledek^{\frac{1}{exponent}}$$

Pokud máme více nenulových exponentů, tak nemůžeme nic dělat.

Často se nám také vyskytuje sčítání i násobení v jedné rovnici. Zde si pomáháme substitucí. Uděláme si novou neznámou a vložíme ji do dané rovnice. Zároveň i však substituce bereme jako samostatnou rovnici, viz příklad:

$$W = \frac{w_i}{1 - w_i}$$

$$x = 1 - w_i$$

$$W = \frac{w_i}{x}$$

Když dosazujeme za neznámou, která se nerovná *0*, tak se nám výsledek změní dle následného vzorce:

$$nový\ výsledek = \frac{starý\ výsledek}{hodnota^{exponent}}$$

Ještě musíme dát pozor, když dosazujeme do rovnic čísla rovná *0*. Pokud dosazujeme do jmenovatele, dochází nám k chybě způsobené přidanou podmínkou, tudíž se můžeme v úpravě vrátit zpět a říct, že když se nám jmenovatel rovná *0*, tak musí i čitatel.

Avšak když se dosazuje za čitatel *0*, tak nastává problém, jelikož dle předchozího vzorce bychom dělili *0*. Nejdříve tedy umocníme celou rovnici na *-1* a tím efektivně převedeme problém na první případ.

Nesmíme zapomenout, že u této operace nám vyjde *rovnice = 0* a zase nevíme nic o jmenovateli.

# Program

## Neznama

První třída je *Neznama*. Z této třídy čerpá celý program. Je takovým základním kamenem. Udržuje nám důležité informace o každé neznámé – přesněji maximální a minimální hodnotu, aktuální hodnotu, jestli známe hodnotu (hodnota může být základní *0* a můžeme jí znát), jméno neznámé, jednotka, index proudu, ke kterému patří, index složky, ke které patří (pokud se jedná obecnou neznámou proudu, tak je *0*), jestli ji chci vidět ve výpisu pro vstup uživatele (pomocné proměnné pro substituci tam nechci), jestli jsou specifické pouze pro plynné proudy, index v poli neznámých a Proud *doProudu* (využívá se u neznámých, které nám udávají zlomek, kolik dané složky nám jde do určitého proudu, jinak *null*).

Dále třída obsahuje přetížený konstruktor a metodu *GetName*, která nám vygeneruje jméno přívětivé pro uživatele i s indexem složky, který se automaticky převede na písmeno v abecedě.

## ReseniSoustavyRovnic

Tato statická třída se stará o potřebné výpočty. Nejdříve bych popsal statická pole a matice, které se zde nachází.

Pole neznámých nám drží všechny neznámé v jednom poli.

Lineární matice a pole výsledků lineárních nám drží matici s rovnicemi pro *Gaussovu eliminaci*.

Násobící matice a pole výsledků násobících nám drží násobící rovnice pro naší násobící metodu.

*UpravaLinearniRovnice* je metoda aplikující *Gaussovu eliminaci*. Metoda nám vrací, jestli něco úpravou zjistila, přičemž kontroluje během každé úpravy, jestli nelze něco zjistit, jelikož když dochází k úpravě více neznámých, než máme rovnic.

*ExtrahujHodnotyLinearni* koukne, jestli je v nějaké lineární rovnici pouze jedna neznámá a zjistí její hodnotu. Rovnou danou hodnotu dosadí do všech lineárních rovnic a vrátí *bool*, jestli něco zjistil.

*PredPripravLinearni* dosadí za všechny známé neznámé do všech lineárních rovnic.

Pomocná metoda *DosazeniNeznameLinearni* dosadí za určitou neznámou do všech lineárních rovnic.

*UpravaNasobneRovnice* je aplikace zde zmíněné násobící metody. Podobně jako u *UpravyLinearniRovnice* vrací *bool*, zda něco zjistil. Také kontroluje po každé úpravě, jestli nelze něco zjistit.

*ExtrahujHodnotyNasobiciRovnice* se podívá, zdali je v nějaké rovnici pouze jedna neznámá, zjistí její hodnotu a dosadí její hodnotu do rovnic. Vrátí *bool*, jestli něco zjistil.

*PredPripravNasobici* dosadí za všechny známé neznámé do všech rovnic.

Pomocná metoda *DosazeniNeznameNasobici* dosadí za určitou neznámou do všech rovnic její hodnotu.

Pomocná metoda *VhodnySloupec* vrací index vhodné rovnice pro určitou neznámou, přičemž jsou mu dány rovnice, o kterou neznámou se jedná, od které rovnice má hledat a kolik na konci má vynechat.

*DosazeníDoRovnic* je součást výpočtu a snaží se zjistit co nejvíce neznámých z pouhého dosazování již známého.

*RESET* odindexuje neznámé a všechny pole s maticemi, které jsou v této třídě uloženy jsou vynullovány.

## Proud

Proud nám spravuje všechny neznámé spojené s ním a složkami v něm. Defaultně jsem nastavil plyn na false, jelikož předpokládám, že většinou budou uživatelé chtít proud, který není plyn. Jediná molární hmotnost je statické pole, jelikož neplánuji, aby se měnila molární hmotnost proud od proudu.

Metoda NastavNezname inicializuje všechny proměnné neznámé a nastaví jejich maxima, minima, jména, jednotky, indexProudu, index složky a eventuálně jestli ji chci vypsat či se vztahuje pouze na plyny.

Pro zjednodušení inicializace je zde metoda, která inicializuje rovnou celá pole, jelikož mají neznámé v poli podobné hodnoty.

Podobná metoda existuje i pro doProudu neznámé. Jen se změnou, že se jedná o pole polí a doProudu se musí nastavovat specificky.

Metoda *ProtejsiProudy* vrátí odkazy na proudy opačné, než kterých je součástí aktuální proud.

*NeznameDoListu* nám vrátí list obsahující všechny proměnné týkající se daného proudu (s prvním proudem se vrátí i molární hmotnosti). Můžeme nechat neznámé i naindexovat, ale to ne vždy chceme, jelikož občas chceme pouhý výpis. Pokud je však chceme naindexovat, tak můžeme nastavit počáteční index, aby se nám indexy nepřekrývaly s neznámými z ostatních proudů.

K této metodě je pomocná metoda *PoleNeznámýchDoListu*, která se stará o to, aby se pole neznámých vložilo do listu a pokud chceme, tak i naindexovalo.

*VnitroproudniRovnice* vrátí listy rovnic, které využívají pouze neznámé z aktuálního proudu (toto je drtivá většina rovnic). Zde pro lehčí zadávání je hned několik pomocných metod. Například *Summa,* do které zadáme pole neznámých a pokud chceme tak i koeficient/exponent a vrátí nám rovnici s koeficienty/exponenty se zadaným koeficientem a pokud není zadán, tak koeficient/exponent bude *1*. *SoucetRovnice* sečte dvě nebo více rovnic a jako vstupní argument je zde pole rovnic.

Pomocná metoda *Rovnice* nám ze zadaných neznámých a jim odpovídajícím koeficientům udělá rovnici. Pokud se však zadá bez vstupních argumentů, vrátí se pouze prázdná rovnice plná nul.

Občas potřebujeme změnit délku pole neznámých, jelikož ubereme či přidáme složku. Přesně od toho je tu metoda *Rozsirit*, jelikož se mi nechtělo zahodit staré hodnoty pokaždé při přenastavení. Odsud se zavolají pomocné metody *ZmenitDelku* a *ZmenitDelkuProudu* (doProudu).

Pomocná metoda *ZmenitDelku* funguje tak, že dokud jsou dostupné staré hodnoty, tak je to kopíruje do nového pole; pokud dojdou, tak začne inicializovat nové na základě parametrů těch předchozích.

*ZmenitDelkuProudu* funguje velice podobně, jen kontroluje, jaké protější Proudy již program použil.

## Uzel

Třída *uzel* má na starosti spravování Proudů a udržuje i statický odkaz na svou jedinou instanci. Drží list všech, vstupních a výstupních Proudů.

Při inicializaci se nejdříve inicializují proudy a až po nich neznámé v nich, jelikož je všechny potřebují doProudu.

*NastavProudy* je pomocná proměnná, která inicializuje rovnou celý list proudů.

*ExtrahujNezname* vezme všechny neznámé z proudů do listu, který následně nastaví jako *ReseniSoustavyRovnic.nezname*.

*ExtrahujRovnice* je metoda, která extrahuje rovnice z proudů i celkové bilance a uloží je do třídy *ReseniSoustavyRovnic*.

*CelkoveBilance* vrátí všechny rovnice celkové bilance.

*RozsirProudy* změní počet proudů dle aktuálního nastavení podobným stylem, jako se rozšiřují pole neznámých v třídě Proud, následně nadefinuje proměnné nových Proudů a rozšíří proměnné starých Proudů.

Pomocná metoda *Rozsirit* změní délku listu Proudů při zachování co nejvíce starých proudů na požadovanou délku.

*PrenastavProudIndexy* nataví proudům nové indexy podle toho, jak jsou seřazeny v poli celkoveProudy.

## NastaveniSlozek

Tento *Form* se skládá ze 3 *numericalupdown* a tlačítka uložit.

Během inicializace nastavím hodnoty *numericalupdown* na aktuální hodnoty a nastavím event, který se spustí při zavírání *Formu*.

Proměnná typu *bool ulozit* slouží k tomu, abych věděl, jestli jsem aplikaci chtěl vypnout přes tlačítko uložit nebo kliknutím na křížek.

*Ulozit* je metoda, která se vyvolá po kliknutí na tlačítko *Uložit*. Nastaví *bool ulozit* na true, čímž dá vědět, že uživatel chtěl uložit a zavře *Form*.

UlozitHodnoty nastaví aktuální hodnoty z *numericalupdown* u proudů, vyvolá přenastavení a reindexaci.

*PriZavreni* se vyvolá během zavírání, což pokud uživatel klikl na tlačítko uložit tak se uloží hodnoty a *Forms* se zavře. Pokud jsme neuložili, tak vyskočí dialogové okno a na základě odpovědi zavře a uloží *Forms*.

## Krmitko

Pole proudů nám drží odkazy na proudy, které chceme nastavovat. Pole neznámých nám drží aktuálně upravované neznámé. Pole *checkBoxů* a *numericalupandown*, které nám drží odkaz na tyto komponenty. Pole indexů nám přiřazuje indexy *checkBoxů* a *numericalupandown* k neznámým. *Indexík* nám udává aktuální index proudu.

Do konstruktoru je nutno zadat pole Proudů, ze kterých se má brát neznámé a jak pojmenovat tento Form (jelikož jméno se mění, jestli se jedná o vstupní či výstupní proudy). Rovnou zavolám generování dotazníku, po čemž ještě nastavím *dropdown,* ze kterého si může uživatel vybrat proud, ve kterém chce nastavovat.

Metoda *Vygeneruj* nejdříve smaže obsah panelu a požádá o všechny neznámé z daného Proudu. Následně si udělá listy *boxů* a *numericalupandown*. Zkontroluje, jestli je daný proud plynný, načež projede postupně všechny neznámé. Nejdříve to zkontroluje, jestli chceme danou proměnnou vůbec vypsat. Následně vytvoříme label s jménem, numericalupandown, checkbox a label s jednotkou. Přičemž *checkBoxu* a *numericalupandown* nastavím event, který nastane, když se jejich hodnota změní. Nakonec program uloží index neznámé na seznam indexů.

Tyto eventy nejdříve vyhledají, kdo jim to vlastně poslal a pomocí seznamu indexů naleznou neznámou, ke které to patří. Toto je nutno dělat, jelikož změny spravují pouze dvě metody – jedna pro *numericalupandown* a druhá pro *checkboxy*. Následně to uloží hodnoty do neznámé. Nesmím zapomenout nastavit z *numericalupandown* i *checkbox* na to, že hodnotu znám.

Pokud se nám však změní nastavení plynu, musíme znovu vygenerovat celý obsah panelu, jelikož najednou máme nové stavové rovnice.

Dále jsou zde tlačítka *další* a *předchozí* Proud. Tyto tlačítka jsou reliktem bývalého stylu fungování UI, ale byly jednoduché přenastavit na nové fungování skrze *dropdown*, takže jsem je zde nechal pro možnost volby.

Samozřejmě, že jakmile si uživatel jakoukoli cestou vybere jiný proud, je nutné tabulku znovu vygenerovat.

## Form1

Tento *Form* sjednocuje vše. Volá ostatní formy a spravuje výpočty.

Když uživatel klikne na tlačítko tak se zkontroluje, jestli není již něco jiného otevřeno či se nepočítá (při výpočtech by nemělo být možno kliknout na tlačítko). Pokud uživatel nevhodně klikne tak se mu otevře messagebox se zprávou, že nelze během jiné akce.

Mimo tlačítek na otevírání *Formů* a započetí výpočtu je zde tlačítko *Reset*. Toto tlačítko vyresetuje celý uzel s aktuálním nastavením počtu proudů a složek (pro uživatele to vypadá, že se hodnoty vynulovaly).

Nejdůležitější tlačítko celého programu je *vypočítat*. Toto tlačítko po zmáčknutí zkontroluje, jestli nedochází k úpravám (nejsou otevřené ostatní *Formy*) – pokud ne tak, asynchronně zavolá výpočet a vypne všechny tlačítka.

Na konci výpočtu ještě proběhne kontrola, jestli hodnoty zjištěné jsou v mezích možných hodnot. Pokud ne, tak v *messageboxu* dá vědět uživateli.

# Kód programu

## Neznama

```
1.  namespace BilancniVypocty
2.  {
3.      public class Neznama // tato třída uchovává v sobě všechno potřebné o nějaké proměnné
4.      {
5.          public int indexVPoli = -1; // index v poli proměnných (-1 znamená, že není zařazen)
6.          public float max; // maximální hodnota
7.          public float min; // minimální hodnota
8.          public float value; // hodnota aktuální
9.          public bool known; // známe hodnotu danné proměnné
10.         public string jmeno; //značka danné proměnné
11.         public string jednotka; // jednotka
12.         public int indexProudu; // v jakém proudu se daná proměnná nachází
13.         public int indexSlozky; // jaké složky se danná neznámá týká (0 znamená celého proudu)
14.         public bool chciVypsat; // jestli chci, aby uživatel viděl hodnoty
15.         public bool pozeProPlyn; // jestli danná neznámá je duležitá pouze pro plyn
16.         public Proud doProudu; // pouze pro do Proudu proměnné, aby měly odkaz na danný proud
17.
18.         public Neznama(float maximum, float minimum, string jmeno, string jednotka, int indexProudu, int indexSlozky, bool chciVeVypisu, bool plyn) // rozšířený konstruktor
19.         {
20.             this.max = maximum;
21.             this.min = minimum;
22.             this.jmeno = jmeno;
23.             this.indexProudu = indexProudu;
24.             this.indexSlozky = indexSlozky;
25.             this.jednotka = jednotka;
26.             known = false;
27.             chciVypsat = chciVeVypisu;
28.             pozeProPlyn = plyn;
29.
30.             if (minimum > 0) // aby byla hodnota mezi maximem a minimem (nikdy by nemělo nastat, aby toto bylo potřeba)
31.             {
32.                 value = minimum;
```

```
33.             }
34.             else
35.             {
36.                 value = 0;
37.             }
38.
39.             doProudu = null;
40.         }
41.
42.         public Neznama(float maximum, float minimum, string jmeno, string jednotka, int indexProudu, int indexSlozky) // krátký konstruktor
43.         {
44.             this.max = maximum;
45.             this.min = minimum;
46.             this.jmeno = jmeno;
47.             this.indexProudu = indexProudu;
48.             this.indexSlozky = indexSlozky;
49.             this.jednotka = jednotka;
50.             known = false;
51.             chciVypsat = true;
52.             pozeProPlyn = false;
53.             if (minimum > 0)
54.             {
55.                 value = minimum;
56.             }
57.             else
58.             {
59.                 value = 0;
60.             }
61.
62.             doProudu = null;
63.         }
64.
65.         public Neznama(float maximum, float minimum, string jmeno, string jednotka, int indexProudu, int indexSlozky, bool chciVeVypisu, bool plyn, Proud proud) // krátký konstruktor
66.         {
67.             this.max = maximum;
68.             this.min = minimum;
69.             this.jmeno = jmeno;
70.             this.indexProudu = indexProudu;
71.             this.indexSlozky = indexSlozky;
72.             this.jednotka = jednotka;
73.             known = false;
74.             chciVypsat = chciVeVypisu;
75.             pozeProPlyn = plyn;
76.
```

```
77.             if (minimum > 0) // aby byla hodnota mezi maximem a minimem (nikdy by nemělo nastat, aby toto bylo potřeba)
78.             {
79.                 value = minimum;
80.             }
81.             else
82.             {
83.                 value = 0;
84.             }
85.
86.             doProudu = proud;
87.         }
88.
89.         public string GetName() // nutno přidat i číslo složky
90.         {
91.             string name = jmeno;
92.
93.             if (doProudu != null)
94.             {
95.                 name = Uzel.alpha[indexSlozky - 1] + jmeno + (doProudu.indexProudu + 1);
96.             }
97.             else if (indexSlozky != 0) // jelikož nula je pro celý proud tak není nutno vypisovat
98.             {
99.                 name += Uzel.alpha[indexSlozky - 1];
100.             }
101.
102.             return name;
103.         }
104.     }
105. }
```

## ReseniSoustavyRovnic

```
1.  namespace BilancniVypocty
2.  {
3.      static class ReseniSoustavyRovnic
4.      {
5.          public static Neznama[] nezname; // seznam všech neznámých
6.          public static float[][] linearniMatice; // lineární rovnice (koeficienty)
7.          public static float[][] nasobiciMatice; // rovnice násobící (exponenty)
8.          public static float[] vysledkyLinearni; // konstanty
9.          public static float[] vysledkyNasobici; // konstaty
10.
```

```
11.         public static bool UpravaLinearniRovnice() // gausova eli-
    minace; vrací, jestli něco zjistila
12.         {
13.             int skipnuto = 0; // počítá kolik neznámých se v rov-
    nících vůbec nevyskytuje (vyruší se nebo jsou už známy)
14.
15.             bool staloSeNeco = false; // kontroluje jestli se něco
    zjistilo
16.
17.             for (int i = 0; i < linearniMatice[0].Length && i <
    linearniMatice.Length + skipnuto - 1; i++) // jede dokud nedojdou
    rovnice nebo neznámé i udává index neznámé
18.             {
19.                 staloSeNeco = ExthrahujHodnotyLinearni() || stalo-
    SeNeco; // zkusí něco získat z dané úpravy?
20.                 int radek = VhodnySloupec(linearniMatice, i, i -
    skipnuto, 0); // zjistí nenulový koeficient u určité neznámé
21.
22.                 if (radek == -1) // pokud všechny neznámé mají
    koeficient 0
23.                 {
24.                     skipnuto++;
25.                     continue;
26.                 }
27.
28.                 if (i - skipnuto != radek) // poku není aktuální
    rovnice rovnice, kterou chci využít
29.                 {
30.                     // vymění aktuální rovnici za vybranou rovnici
31.                     float[] podrzRovnici = linearniMatice[i - ski-
    pnuto];
32.                     linearniMatice[i - skipnuto] = linearniMa-
    tice[radek];
33.                     linearniMatice[radek] = podrzRovnici;
34.
35.                     float podrzVysledek = vysledkyLinearni[radek];
36.                     vysledkyLinearni[radek] = vysledkyLinearni[i -
    skipnuto];
37.                     vysledkyLinearni[i - skipnuto] = podrzVysle-
    dek;
38.                 }
39.
40.                 for (int j = 0; j < linearniMatice.Length; j++) //
    projedu všechny rovnice
41.                 {
42.                     if (j == i - skipnuto) // nechci odečítat tuto
    rovnici od této rovnice jelikož by mi vyšlo 0 = 0
43.                     {
```

```
44.                         continue;
45.                     }
46.
47.                     float koeficient = linearniMatice[j][i] / linearniMatice[i - skipnuto][i]; // zjistí kolikrát je nutno odečíst aktuální rovnici od rovnice j, aby koeficient u nezname[i] 0
48.
49.                     vysledkyLinearni[j] -= vysledkyLinearni[i - skipnuto] * koeficient; // odečte výsledek aktuální rovnice od výsledku j
50.
51.                     for (int k = i + 1; k < linearniMatice[j].Length; k++) // projede všechny koeficienty za aktuální zenámou i
52.                     {
53.                         linearniMatice[j][k] -= linearniMatice[i - skipnuto][k] * koeficient; // odečte koeficient v aktální rovnici * koeficient od koeficientu v danné rovnici
54.                     }
55.                     linearniMatice[j][i] = 0; // aby jsem předešel zaokrouhlovací chybě nastavím na 0
56.                 }
57.             }
58.
59.             staloSeNeco = ExthrahujHodnotyLinearni() || staloSeNeco;
60.             return staloSeNeco; // vrátí jestli to k něčemu vůbec bylo
61.         }
62.
63.         public static bool ExthrahujHodnotyLinearni() // zjistí jestli nelze z matice zjistit hodnotu nějaké neznámé
64.         {
65.             bool upraveno = false; // zjistili jsme něco v této metodě?
66.
67.             bool necoSeStalo; // zjistili jsme něco v této iteraci?
68.             do
69.             {
70.                 necoSeStalo = false;
71.
72.                 for (int i = 0; i < linearniMatice.Length; i++) // projede všechny rovnice
73.                 {
74.                     int posledniNeznama; // index posledního nenulového koeficientu
75.                     if (NeznamychVRovnici(linearniMatice[i], out posledniNeznama) == 1) // zjistí jestli je v rovnici pouze jedna neznámá
```

```
76.                     {
77.                         if (nezname[posledniNeznama].known) // po-
   kud již známe hodnotu dané neznámé (nemělo by nastat, ale jistota
   je jistota)
78.                         {
79.                             continue;
80.                         }
81.
82.                         nezname[posledniNeznama].value = vysled-
   kyLinearni[i] / linearniMatice[i][posledniNeznama]; // hodnota ne-
   známé je výsledek rovnice / koeficient u neznámé
83.                         nezname[posledniNeznama].known = true; //
   nyní známe
84.
85.                         vysledkyLinearni[i] = 0; // nastaví výsle-
   dek rovnice na 0
86.                         linearniMatice[i][posledniNeznama] = 0; //
   nastaví koeficient na nula
87.
88.                         DosazeniNeznameLinearni(posledniNeznama);
89.
90.                         // něco jme zjistili
91.                         upraveno = true;
92.                         necoSeStalo = true;
93.                     }
94.                 }
95.             } while (necoSeStalo);
96.
97.             return upraveno;
98.         }
99.
100.         public static void PredPripravLinearni() // dosaď známé
   neznámé do rovnic
101.         {
102.             for (int i = 0; i < nezname.Length; i++)
103.             {
104.                 if (nezname[i].known)
105.                 {
106.                     DosazeniNeznameLinearni(i); // dosaď za ur-
   čitou neznámou
107.                 }
108.             }
109.         }
110.
111.         private static void DosazeniNeznameLinearni(int inde-
   xDosayovaneho) // dosadí za určitou hodnotu neznámé do rovnic
112.         {
113.             for (int j = 0; j < linearniMatice.Length; j++)
```

```
114.             {
115.                 vysledkyLinearni[j] -= nezname[indexDosayovaneho].value * linearniMatice[j][indexDosayovaneho];
116.                 linearniMatice[j][indexDosayovaneho] = 0;
117.             }
118.         }
119.
120.         public static bool UpravaNasobneRovnice() // úprava násobících rovnic
121.         {
122.             int skipnuto = 0; // počítá kolik neznámých se v rovnících vůbec nevyskytuje (vyruší se nebo jsou už známy)
123.
124.             int rovnaSeNula = 0; // kolik rovnic je tvaru x * y = 0
125.
126.             bool staloSeNeco = false; // víme něco
127.
128.             for (int i = 0; i < nasobiciMatice[0].Length && i < nasobiciMatice.Length + skipnuto - rovnaSeNula; i++) // jede dokud nedojdou rovnice nebo neznámé a nevyužívala rovnice, které jsou nulové
129.             {
130.                 staloSeNeco = ExtrahujHodnotyNasobiciRovnice() || staloSeNeco; // zjistili jsme něco z rovnic
131.
132.                 int radek = VhodnySloupec(nasobiciMatice, i, i - skipnuto, rovnaSeNula); // zjistí index rovnice s nenulovým exponentem u neznámé i
133.
134.                 if (radek == -1) // nic jsme nenašli => další neznámá
135.                 {
136.                     skipnuto++;
137.                     continue;
138.                 }
139.
140.                 if (vysledkyNasobici[radek] == 0) //pokud je výsledek 0 tak se výsledek uloží na konec rovnic, aby nepřekážela
141.                 {
142.                     float[] podrzRovnici = nasobiciMatice[nasobiciMatice.Length - 1];
143.                     nasobiciMatice[nasobiciMatice.Length - 1] = nasobiciMatice[radek];
144.                     nasobiciMatice[radek] = podrzRovnici;
145.
146.                     float podrzVysledek = vysledkyNasobici[radek];
```

```
147.                     vysledkyNasobici[radek]    =    vysledkyNaso-
    bici[nasobiciMatice.Length - 1];
148.                     vysledkyNasobici[nasobiciMatice.Length - 1]
    = podrzVysledek;
149.
150.                     rovnaSeNula++;
151.                     i--; // danná iterace nám nic neřekla
152.                     continue;
153.                 }
154.
155.                 if (i - skipnuto != radek) // pokud vybraný řádek
    není aktuální řádek tak prohoď
156.                 {
157.                     float[] podrzRovnici = nasobiciMatice[i -
    skipnuto];
158.                     nasobiciMatice[i - skipnuto] = nasobiciMa-
    tice[radek];
159.                     nasobiciMatice[radek] = podrzRovnici;
160.
161.                     float podrzVysledek = vysledkyNasobici[ra-
    dek];
162.                     vysledkyNasobici[radek]    =    vysledkyNaso-
    bici[i - skipnuto];
163.                     vysledkyNasobici[i - skipnuto] = podrzVysle-
    dek;
164.                 }
165.
166.                 for (int j = 0; j < nasobiciMatice.Length; j++)
    // projeď všechny rovnice
167.                 {
168.                     if (j == i - skipnuto) // kdyby vydělila sama
    sebou vyjde nám 0 = 0
169.                     {
170.                         continue;
171.                     }
172.
173.                     float koeficient = nasobiciMatice[j][i] /
    nasobiciMatice[i - skipnuto][i]; // koeficient kolikrát je
174.
175.                     vysledkyNasobici[j] = vysledkyNasobici[j] /
    (float)Math.Pow(vysledkyNasobici[i - skipnuto], koeficient); //
    vyděl výsledek j výsledkem aktuální rovnice na koeficient
176.
177.                     for (int k = i + 1; k < nasobiciMatice[j].Len-
    gth; k++)
178.                     {
```

```
179.                         nasobiciMatice[j][k] -= nasobiciMatice[i
    - skipnuto][k] * koeficient; // odečti od exponentu exponent aktu-
    alní rovnice * koeficient
180.                     }
181.                     nasobiciMatice[j][i] = 0; // pro zamezení za-
    okrouhlovací chyby nastav na 0
182.                 }
183.
184.                 for (int k = 1; k <= rovnaSeNula; k++) // vyházej
    záporné koeficietny z rovnic x/y = 0; jelikož nemohou být nula
185.                 {
186.                     VyhazejZaporneHodnotyZNasobiciRovnice(naso-
    biciMatice.Length - k);
187.                 }
188.             }
189.
190.             staloSeNeco = ExtrahujHodnotyNasobiciRovnice() ||
    staloSeNeco; // extrahuj hodnoty, které můžeme zjistit
191.
192.             return staloSeNeco;
193.         }
194.
195.         public static bool ExtrahujHodnotyNasobiciRovnice() //
    zjisti hodnoty, které můžeš z aktuálního stavu rovnice
196.         {
197.             bool upraveno = false; // zjistil jsi něco
198.
199.             bool necoSeStalo; // zjistil jsi něco tuto iteraci
    do while
200.             do
201.             {
202.                 necoSeStalo = false;
203.
204.                 for (int i = 0; i < nasobiciMatice.Length; i++)
    // projeď všechny rovnice
205.                 {
206.                     int posledniNeznama; // index poslední ne-
    známé
207.                     if     (NeznamychVRovnici(nasobiciMatice[i],
    out posledniNeznama) == 1) // je v danné rovnici pouze jedna neznámá
208.                     {
209.                         nezname[posledniNeznama].value        =
    (float)Math.Pow(vysledkyNasobici[i], 1 / nasobiciMatice[i][posled-
    niNeznama]); // odmocníme výsledek rovnice exponentem poslední ne-
    známé
210.                         nezname[posledniNeznama].known = true;
211.
```

```
212.                            vysledkyNasobici[i] = 0; // nastaví rov-
    nici na 0 = 0
213.                            nasobiciMatice[i][posledniNeznama] = 0;
214.
215.                            DosazeniNeznameNasobici(posledniNe-
    znama); // Dosadí do všech ostatních rovnic hodnotu
216.
217.                            upraveno = true;
218.                            necoSeStalo = true;
219.                        }
220.                    }
221.                } while (necoSeStalo);
222.
223.                return upraveno;
224.            }
225.
226.            public static void PredPripravNasobici() // dosaď za
    známé hodnoty neznámých
227.            {
228.                for (int i = 0; i < nezname.Length; i++)
229.                {
230.                    if (nezname[i].known)
231.                    {
232.                        DosazeniNeznameNasobici(i);
233.                    }
234.                }
235.            }
236.
237.            public static void DosazeniNeznameNasobici(int inde-
    xDosayovaneho) // Dosadí za neznámou do násobících rovnic
238.            {
239.                for (int j = 0; j < nasobiciMatice.Length; j++) //
    projeď všechny rovnice
240.                {
241.                    if (nasobiciMatice[j][indexDosayovaneho] == 0)
    // pokud je exponent 0 nic nedělej
242.                    {
243.                        continue;
244.                    }
245.
246.                    if (nezname[indexDosayovaneho].value == 0) //
    pokud hodnota dosazované neznámé je 0
247.                    {
248.                        if (nasobiciMatice[j][indexDosayovaneho] >
    0) // pokud se jedná o kladný exponet nastává dělení nulou
249.                        {
```

```
250.                        for (int i = 0; i < nasobiciMa-
    tice[j].Length; i++) // otočíme znaménka exponentu jelikož nám
    všechny podmínky vzniky úpravou rovnice (takže to nakonec výjde)
251.                        {
252.                            nasobiciMatice[j][i] = -nasobiciMa-
    tice[j][i];
253.                        }
254.                        vysledkyNasobici[j] = 1 / vysledkyNaso-
    bici[j];
255.                    }
256.                    VyhazejZaporneHodnotyZNasobiciRovnice(j);
    // odebereme jmenovatele jelikož by se měl rovnat nule
257.                    vysledkyNasobici[j] = 0; // nastavíme pravou
    stranu na 0
258.                }
259.                else // pokud hodnota není nula tak normálně
260.                {
261.                    vysledkyNasobici[j] = vysledkyNasobici[j] /
    (float)Math.Pow(nezname[indexDosayovaneho].value,      nasobiciMa-
    tice[j][indexDosayovaneho]);
262.                }
263.                nasobiciMatice[j][indexDosayovaneho] = 0; // ex-
    ponent za známého = 0
264.            }
265.        }
266.
267.        private static int VhodnySloupec(float[][] matice, int
    sloupec, int pocatecniRadek, int vynechatRadku) // Nalezne z matice
    z určitého sloupce první vhodnou rovnici
268.        {
269.            for (int i = pocatecniRadek; i < matice.Length -
    vynechatRadku; i++)
270.            {
271.                if (matice[i][sloupec] != 0) // není nula? tak
    to jsme předci hledali
272.                {
273.                    return i;
274.                }
275.            }
276.
277.            return -1; // nenašel jsi nic tak vrať -1 ať to víme
278.        }
279.
280.        private static int NeznamychVRovnici(float[] rovnice,
    out int indexPosledniNezname) // vrátí počet nenulových koefi-
    cientů/exponentů v rovnici a i index poslední nenulové hodnoty
281.        {
282.            int neznamych = 0;
```

```
283.             indexPosledniNezname = -1; // z debug důvodů -1 a navíc musí být definováno mimo if
284.             for (int i = 0; i < rovnice.Length; i++)
285.             {
286.                 if (rovnice[i] != 0)
287.                 {
288.                     neznamych++;
289.                     indexPosledniNezname = i;
290.                 }
291.             }
292.             return neznamych;
293.         }
294.
295.         private static void VyhazejZaporneHodnotyZNasobiciRovnice(int indexRovnice) // vymění záporné exponenty násobící rovnice z rovnice (voláno když se rovnice rovná nule)
296.         {
297.             for (int j = 0; j < nasobiciMatice[indexRovnice].Length; j++) // projede všechny členy rovnice
298.             {
299.                 if (nasobiciMatice[indexRovnice][j] < 0) // menší než nula
300.                 {
301.                     nasobiciMatice[indexRovnice][j] = 0; // vynulovat
302.                 }
303.             }
304.         }
305.
306.         public static void DosazeniDoRovnic() // první část (dosazování do rovnic)
307.         {
308.             // nejdříve zavolám pouze dosazování a snažím se zjistit co nejvíce můžu bez úprav
309.
310.             LinDosazeniDoRovnic(); // nejdříve zavolám mimo loop, abych před možným ukončením udělal obě metody
311.
312.             while (true)
313.             {
314.                 if (!NasDosazeniDoRovnic())
315.                 {
316.                     break;
317.                 }
318.
319.                 if (LinDosazeniDoRovnic())
320.                 {
321.                     break;
```

```
322.                 }
323.             }
324.         }
325.
326.         private static bool LinDosazeniDoRovnic()
327.         {
328.             ReseniSoustavyRovnic.PredPripravLinearni();  // do-
    sadˇ
329.             return      ReseniSoustavyRovnic.ExthrahujHodnotyLi-
    nearni(); // víme z toho něco
330.         }
331.
332.         private static bool NasDosazeniDoRovnic()
333.         {
334.             ReseniSoustavyRovnic.PredPripravNasobici();  // do-
    sadˇ
335.             return  ReseniSoustavyRovnic.ExtrahujHodnotyNasobi-
    ciRovnice(); // jsme z toho o něco chytřejší
336.         }
337.
338.         public static void VypisMatici(float[][] matice, float[]
    hodnoty)  // debug; metoda vypíše všechny hodnoty v matici s vý-
    sledky
339.         {
340.             Console.WriteLine();
341.             for (int i = 0; i < matice.Length; i++)
342.             {
343.                 for (int j = 0; j < matice[i].Length; j++)
344.                 {
345.                     Console.Write(matice[i][j] + "; ");
346.                 }
347.                 Console.WriteLine("= " + hodnoty[i]);
348.             }
349.
350.             Console.WriteLine();
351.         }
352.
353.         public static void VypisMaticiChytre(float[][] matice,
    float[] hodnoty)  // debub; vypíše hodnotu z matice a jméno, ale
    jen u nenulových
354.         {
355.             Console.WriteLine();
356.             for (int i = 0; i < matice.Length; i++)
357.             {
358.                 for (int j = 0; j < matice[i].Length; j++)
359.                 {
360.                     if (matice[i][j] != 0)
361.                     {
```

```
362.                         Console.Write(matice[i][j] + " " + ne-
    zname[j].GetName());
363.
364.                         if (matice == nasobiciMatice)
365.                         {
366.                             Console.Write(" * ");
367.                         }
368.                         else if (matice == linearniMatice)
369.                         {
370.                             Console.Write(" + ");
371.                         }
372.                         else
373.                         {
374.                             Console.Write("; ");
375.                         }
376.                     }
377.                 }
378.                 Console.WriteLine("= " + hodnoty[i]);
379.             }
380.
381.             Console.WriteLine();
382.         }
383.
384.         public static void VypisNezname() // debug; vypiš hodnoty
    pole neznámých
385.         {
386.             Console.WriteLine();
387.             for (int i = 0; i < nezname.Length; i++)
388.             {
389.                 Console.WriteLine(nezname[i].GetName() + " = " +
    nezname[i].value);
390.             }
391.         }
392.
393.         public static void RESET() // uvolni nepotřebné místo v
    paměti a odindexuj pole neznámých
394.         {
395.             foreach (Neznama item in nezname)
396.             {
397.                 item.indexVPoli = -1;
398.             }
399.
400.             nezname = null;
401.
402.             linearniMatice = null;
403.             nasobiciMatice = null;
404.             vysledkyLinearni = null;
405.             vysledkyNasobici = null;
```

```
406.          }
407.      }
408.  }
```

## Proud

```
1.namespace BilancniVypocty
2.{
3.    public class Proud
4.    {
5.        private const float plynovaKonstanta = 8.31446261815324f;
6.
7.        public int indexProudu; // o jaký proud se jedná
8.
9.        public bool plyn; // je danný proud plynný
10.
11.           // proud si spravuje stavové veličiny
12.
13.           // celkove proměnné
14.           public Neznama celkovaHmotnost;
15.
16.           public Neznama hustota;
17.
18.           public Neznama objem;
19.
20.           public Neznama molarniHmotnostSmesi;
21.
22.           public Neznama celkemMolu;
23.
24.           // u jednotlyvých složek
25.           public Neznama[] hmotnostiSlozek;
26.
27.           public Neznama[] hmotnostniZlomky;
28.
29.           public Neznama[] relativnihmotnostniZlomky;
30.
31.           public Neznama[] molarniZlomky;
32.
33.           public Neznama[] relativniMolarniZlomky;
34.
35.           public Neznama[] molarniKoncentrace;
36.
37.           public Neznama[] hmotnostniKoncentrace;
38.
39.           public Neznama[][] koeficientDoJakehoProudu;
40.
41.           public Neznama[] latkoveMnozstvi;
42.
```

```
43.         public Neznama[] pocetMolekul;
44.
45.         public Neznama[] objemJednotlivychLatek;
46.
47.         public Neznama[] relativniObjemovaKoncentrace;
48.
49.         public Neznama[] objemovaKoncentrace;
50.
51.         public Neznama[] hustotaSlozky;
52.
53.         // staticke promene
54.         public static Neznama[] molarniHmotnost;
55.
56.         // u plynu
57.         public Neznama tlak;
58.
59.         public Neznama teplota;
60.
61.         public Neznama[] parcialniTlak;
62.
63.         // pomocene promene
64.
65.         public Neznama[] pomocnaSummaMolHmotnosti;
66.
67.         public Neznama[] pomocnaSummaProVypocetmolarnichZlomku;
68.
69.         public   Neznama   pomocnaCelkovaSummaProVypocetmolar-
  nichZlomku;
70.
71.         public Neznama[] pomocnyRelativniHmotnostniZlomek;
72.
73.         public Neznama[] pomocnyRelativniMolarniZlomek;
74.
75.         public Neznama[] pomocnaRelativniObjemovaKoncentrace;
76.
77.         public Neznama[][] pomocnaKoeficientDoProudu;
78.
79.         public Neznama[][] pomocnaLatkoveKoeficientDoProudu;
80.
81.         public Proud(int indexProudu)
82.         {
83.             this.indexProudu = indexProudu;
84.             plyn = false;
85.         }
86.
87.         public void NastavNezname(int slozek, int prouduNaDruhe-
  Strane)
88.         {
```

```
89.             // definice Neznama
90.
91.             celkovaHmotnost = new Neznama((float)(decimal.MaxVa-
  lue * (decimal)0.99), 0, "m", "kg", indexProudu, 0); // slozka 0 -
  celkova proudu
92.
93.             hustota = new Neznama((float)(decimal.MaxValue *
  (decimal)0.99), 0, "ϱ", "kg*m^(-3)", indexProudu, 0);
94.
95.             objem = new Neznama((float)(decimal.MaxValue * (de-
  cimal)0.99), 0, "V", "m^3", indexProudu, 0);
96.
97.             molarniHmotnostSmesi = new Neznama((float)(deci-
  mal.MaxValue * (decimal)0.99), 0, "M", "kg*mol^(-1)", indexProudu,
  0);
98.
99.             celkemMolu = new Neznama((float)(decimal.MaxValue *
  (decimal)0.99), 0, "n", "mol", indexProudu, 0);
100.
101.            tlak = new Neznama((float)(decimal.MaxValue * (deci-
  mal)0.99), 0, "p", "Pa", indexProudu, 0, true, true);
102.
103.            teplota = new Neznama((float)(decimal.MaxValue *
  (decimal)0.99), 0, "T", "K", indexProudu, 0, true, true);
104.
105.            hmotnostiSlozek = NastavPoleNeznamych(slozek,
  (float)(decimal.MaxValue * (decimal)0.99), 0, "m", "kg");
106.
107.            relativnihmotnostniZlomky = NastavPoleNe-
  znamych(slozek, (float)(decimal.MaxValue * (decimal)0.99), 0, "W",
  "1");
108.
109.            hmotnostniZlomky = NastavPoleNeznamych(slozek, 1, 0,
  "w", "1");
110.
111.            molarniZlomky = NastavPoleNeznamych(slozek, 1, 0,
  "x", "1");
112.
113.            relativnihmotnostniZlomky = NastavPoleNe-
  znamych(slozek, (float)(decimal.MaxValue * (decimal)0.99), 0, "W",
  "1");
114.
115.            molarniKoncentrace = NastavPoleNeznamych(slozek,
  (float)(decimal.MaxValue * (decimal)0.99), 0, "c", "mol*m^(-3)");
116.
117.            hmotnostniKoncentrace = NastavPoleNeznamych(slozek,
  (float)(decimal.MaxValue * (decimal)0.99), 0, "ϱ(m)", "kg*m^(-3)");
118.
```

```
119.            latkoveMnozstvi     =     NastavPoleNeznamych(slozek,
  (float)(decimal.MaxValue * (decimal)0.99), 0, "n", "mol");
120.
121.            pocetMolekul      =      NastavPoleNeznamych(slozek,
  (float)(decimal.MaxValue * (decimal)0.99), 0, "N", "1");
122.
123.            if (molarniHmotnost == null)
124.            {
125.                molarniHmotnost = NastavPoleNeznamych(slozek,
  (float)(decimal.MaxValue * (decimal)0.99), 0, "M", "kg*mol^(-1)");
126.            }
127.
128.            koeficientDoJakehoProudu  =  NastavDoProudu(slozek,
  prouduNaDruheStrane);
129.
130.            pomocnaKoeficientDoProudu = NastavDoProudu(slozek,
  prouduNaDruheStrane, false);
131.
132.            pomocnaLatkoveKoeficientDoProudu   =   NastavDoPro-
  udu(slozek, prouduNaDruheStrane, false);
133.
134.            relativniMolarniZlomky  =  NastavPoleNeznamych(slo-
  zek, (float)(decimal.MaxValue * (decimal)0.99), 0, "X", "1");
135.
136.            objemJednotlivychLatek  =  NastavPoleNeznamych(slo-
  zek, (float)(decimal.MaxValue * (decimal)0.99), 0, "V", "m^3");
137.
138.            relativniObjemovaKoncentrace     =     NastavPoleNe-
  znamych(slozek,  (float)(decimal.MaxValue  *  (decimal)0.99),  0,
  "Vol", "1");
139.
140.            objemovaKoncentrace = NastavPoleNeznamych(slozek, 1,
  0, "vol", "1");
141.
142.            pomocnaSummaMolHmotnosti = NastavPoleNeznamych(slo-
  zek,  (float)(decimal.MaxValue  *  (decimal)0.99),  0,  "(xi  *  Mi)",
  "kg/mol", false, false);
143.
144.            pomocnaSummaProVypocetmolarnichZlomku = NastavPole-
  Neznamych(slozek,  (float)(decimal.MaxValue  *  (decimal)0.99),  0,
  "(wi/Mi)", "mol/kg", false, false);
145.
146.            pomocnaCelkovaSummaProVypocetmolarnichZlomku  =  new
  Neznama((float)(decimal.MaxValue  *  (decimal)0.99),  0,  "(wi/Mi)",
  "mol/kg", indexProudu, 0, false, false);
147.
```

```
148.            pomocnyRelativniHmotnostniZlomek = NastavPoleNe-
  znamych(slozek, (float)(decimal.MaxValue * (decimal)0.99), 0, "(m-
  mi)", "kg", false, false);
149.
150.            pomocnyRelativniMolarniZlomek = NastavPoleNe-
  znamych(slozek, (float)(decimal.MaxValue * (decimal)0.99), 0, "(n-
  ni)", "mol", false, false);
151.
152.            pomocnaRelativniObjemovaKoncentrace = NastavPoleNe-
  znamych(slozek, (float)(decimal.MaxValue * (decimal)0.99), 0, "(V -
  V i)", "m^3", false, false);
153.
154.            parcialniTlak = NastavPoleNeznamych(slozek,
  (float)(decimal.MaxValue * (decimal)0.99), 0, "pi", "Pa", false,
  true);
155.
156.            hustotaSlozky = NastavPoleNeznamych(slozek,
  (float)(decimal.MaxValue * (decimal)0.99), 0, "ϱ", "kg*m^(-3)");
157.        }
158.
159.        private Neznama[] NastavPoleNeznamych(int pocetSlozek,
  float maximum, float minimum, string jmeno, string jednotka) //
  nastavení pole neznámých
160.        {
161.            Neznama[] nezname = new Neznama[pocetSlozek];
162.
163.            for (int i = 0; i < pocetSlozek; i++)
164.            {
165.                nezname[i] = new Neznama(maximum, minimum,
  jmeno, jednotka, indexProudu, i + 1);
166.            }
167.
168.            return nezname;
169.        }
170.
171.        private Neznama[] NastavPoleNeznamych(int pocetSlozek,
  float maximum, float minimum, string jmeno, string jednotka, bool
  chciVypsat, bool plyn) // nastavení pole neznámých pro rozšířený
  konstruktor
172.        {
173.            Neznama[] nezname = new Neznama[pocetSlozek];
174.
175.            for (int i = 0; i < pocetSlozek; i++)
176.            {
177.                nezname[i] = new Neznama(maximum, minimum,
  jmeno, jednotka, indexProudu, i + 1, chciVypsat, plyn);
178.            }
179.
```

```
180.                return nezname;
181.            }
182.
183.            private Neznama[][] NastavDoProudu(int pocetSlozek, int
    pocetProudu)
184.            {
185.                return   NastavDoProudu(pocetSlozek,   pocetProudu,
    true);
186.            }
187.
188.            private Neznama[][] NastavDoProudu(int pocetSlozek, int
    pocetProudu, bool chciVypsat) // nastaví Neznámé do Proudu
189.            {
190.                Neznama[][] pomocna = new Neznama[pocetSlozek][];
191.
192.                Proud[]   protejsiProudy   =   ProtejsiProudy(index-
    Proudu);
193.
194.                for (int i = 0; i < pocetSlozek; i++)
195.                {
196.                    pomocna[i] = new Neznama[pocetProudu];
197.                }
198.                for (int i = 0; i < pocetSlozek; i++)
199.                {
200.                    for (int j = 0; j < pocetProudu; j++)
201.                    {
202.                        pomocna[i][j] = new Neznama(1, 0, "->", "1",
    j, i + 1, chciVypsat, false, protejsiProudy[j]);
203.                    }
204.                }
205.
206.                return pomocna;
207.            }
208.
209.            private static Proud[] ProtejsiProudy(int indexProudu)
    // pošli protější proudy
210.            {
211.                foreach (Proud item in Uzel.uzel.vystupniProudy)
212.                {
213.                    if (item.indexProudu == indexProudu)
214.                    {
215.                        return Uzel.uzel.vztupniProudy.ToArray();
216.                    }
217.                }
218.
219.                return Uzel.uzel.vystupniProudy.ToArray(); // pokud
    není tam tak je tady
220.            }
```

```
221.
222.         public List<Neznama> NeznameDoListu(int pocatecnyIndex,
  bool chciINaindexovane) // vrátí všechny neznámé z proudu a pokud
  chceme tak je i naindexuje
223.         {
224.             List<Neznama> vratit = new List<Neznama>();
225.
226.             celkovaHmotnost.indexVPoli = vratit.Count + pocatec-
  nyIndex;
227.             vratit.Add(celkovaHmotnost);
228.
229.             hustota.indexVPoli = vratit.Count + pocatecnyIndex;
230.             vratit.Add(hustota);
231.
232.             objem.indexVPoli = vratit.Count + pocatecnyIndex;
233.             vratit.Add(objem);
234.
235.             molarniHmotnostSmesi.indexVPoli = vratit.Count + po-
  catecnyIndex;
236.             vratit.Add(molarniHmotnostSmesi);
237.
238.             celkemMolu.indexVPoli = vratit.Count + pocatecnyIn-
  dex;
239.             vratit.Add(celkemMolu);
240.
241.             pomocnaCelkovaSummaProVypocetmolarnichZlomku.inde-
  xVPoli = vratit.Count + pocatecnyIndex;
242.             vratit.Add(pomocnaCelkovaSummaProVypocetmolar-
  nichZlomku);
243.
244.             vratit.AddRange(PoleNeznamychDoListu(hmotnostiSlo-
  zek, vratit.Count + pocatecnyIndex, chciINaindexovane));
245.
246.             vratit.AddRange(PoleNeznamychDoListu(latkoveMnoz-
  stvi, vratit.Count + pocatecnyIndex, chciINaindexovane));
247.
248.             vratit.AddRange(PoleNeznamychDoListu(hmotnost-
  niZlomky, vratit.Count + pocatecnyIndex, chciINaindexovane));
249.
250.             vratit.AddRange(PoleNeznamychDoListu(relativnihmot-
  nostniZlomky, vratit.Count + pocatecnyIndex, chciINaindexovane));
251.
252.             vratit.AddRange(PoleNeznamychDoListu(molarniZlomky,
  vratit.Count + pocatecnyIndex, chciINaindexovane));
253.
254.             vratit.AddRange(PoleNeznamychDoListu(relativ-
  niMolarniZlomky, vratit.Count + pocatecnyIndex, chciINaindexo-
  vane));
```

```
255.
256.            vratit.AddRange(PoleNeznamychDoListu(molarniKoncen-
  trace, vratit.Count + pocatecnyIndex, chciINaindexovane));
257.
258.            vratit.AddRange(PoleNeznamychDoListu(hmotnostniKon-
  centrace, vratit.Count + pocatecnyIndex, chciINaindexovane));
259.
260.            vratit.AddRange(PoleNeznamychDoListu(pocetMolekul,
  vratit.Count + pocatecnyIndex, chciINaindexovane));
261.
262.            vratit.AddRange(PoleNeznamychDoListu(molarniHmot-
  nost, vratit.Count + pocatecnyIndex, chciINaindexovane));
263.
264.            vratit.AddRange(PoleNeznamychDoListu(pomocnaSumma-
  MolHmotnosti, vratit.Count + pocatecnyIndex, chciINaindexovane));
265.
266.            vratit.AddRange(PoleNeznamychDoListu(pomocnaSum-
  maProVypocetmolarnichZlomku,   vratit.Count   +   pocatecnyIndex,
  chciINaindexovane));
267.
268.            vratit.AddRange(PoleNeznamychDoListu(pomocnyRela-
  tivniHmotnostniZlomek, vratit.Count + pocatecnyIndex, chciINainde-
  xovane));
269.
270.            vratit.AddRange(PoleNeznamychDoListu(pomocnyRela-
  tivniMolarniZlomek, vratit.Count + pocatecnyIndex, chciINaindexo-
  vane));
271.
272.            vratit.AddRange(PoleNeznamychDoListu(objemJednotli-
  vychLatek, vratit.Count + pocatecnyIndex, chciINaindexovane));
273.
274.            vratit.AddRange(PoleNeznamychDoListu(relativniObje-
  movaKoncentrace,  vratit.Count  +  pocatecnyIndex,  chciINaindexo-
  vane));
275.
276.            vratit.AddRange(PoleNeznamychDoListu(pomocnaRela-
  tivniObjemovaKoncentrace, vratit.Count + pocatecnyIndex, chciINain-
  dexovane)); // už mě nebaví komentovat
277.
278.            vratit.AddRange(PoleNeznamychDoListu(pomocnyRela-
  tivniMolarniZlomek, vratit.Count + pocatecnyIndex, chciINaindexo-
  vane));
279.
280.            vratit.AddRange(PoleNeznamychDoListu(objemovaKon-
  centrace, vratit.Count + pocatecnyIndex, chciINaindexovane));
281.
282.            vratit.AddRange(PoleNeznamychDoListu(objemovaKon-
  centrace, vratit.Count + pocatecnyIndex, chciINaindexovane));
```

```
283.
284.             vratit.AddRange(PoleNeznamychDoListu(hustotaSlozky,
  vratit.Count + pocatecnyIndex, chciINaindexovane));
285.
286.             foreach (Neznama[] item in koeficientDoJakehoProudu)
287.             {
288.                 vratit.AddRange(PoleNeznamychDoListu(item, vra-
  tit.Count + pocatecnyIndex, chciINaindexovane));
289.             }
290.
291.             foreach (Neznama[] item in pomocnaKoeficientDoPro-
  udu)
292.             {
293.                 vratit.AddRange(PoleNeznamychDoListu(item, vra-
  tit.Count + pocatecnyIndex, chciINaindexovane));
294.             }
295.
296.             foreach (Neznama[] item in pomocnaLatkoveKoeficient-
  DoProudu)
297.             {
298.                 vratit.AddRange(PoleNeznamychDoListu(item, vra-
  tit.Count + pocatecnyIndex, chciINaindexovane));
299.             }
300.
301.             if (plyn)
302.             {
303.                 teplota.indexVPoli = vratit.Count;
304.                 vratit.Add(teplota);
305.
306.                 tlak.indexVPoli = vratit.Count;
307.                 vratit.Add(tlak);
308.
309.                 vratit.AddRange(PoleNeznamychDoListu(parcial-
  niTlak, vratit.Count + pocatecnyIndex, chciINaindexovane));
310.             }
311.
312.             return vratit;
313.         }
314.
315.         private List<Neznama> PoleNeznamychDoListu(Neznama[] ne-
  zname, int pocatecnyIndex, bool chciInaindexovane) // vrátí list
  neznámých z pole neznznámých a pokud chceme tak i naindexuje
316.         {
317.             List<Neznama> vratit = new List<Neznama>();
318.
319.             for (int i = 0; i < nezname.Length; i++)
320.             {
321.                 if (chciInaindexovane)
```

```
322.                 {
323.                     vratit.Add(nezname[i]);
324.                 }
325.                 else if (nezname[i].indexVPoli == -1)
326.                 {
327.                     nezname[i].indexVPoli = pocatecnyIndex + i;
328.                     vratit.Add(nezname[i]);
329.                 }
330.             }
331.
332.             return vratit;
333.         }
334.
335.         public void VnitroProudniRovnice(out List<float[]> li-
  nearniRovnice, out List<float> vysledkyLinearni, out List<float[]>
  nasobiciRovnice, out List<float> vysledkyNasobici) // připraví rov-
  nice které obsahují pouze složky z jednoho proudu
336.         {
337.             // místo na uložení rovnic
338.
339.             linearniRovnice = new List<float[]>();
340.
341.             vysledkyLinearni = new List<float>();
342.
343.             nasobiciRovnice = new List<float[]>();
344.
345.             vysledkyNasobici = new List<float>();
346.
347.             // E wi = 1
348.             linearniRovnice.Add(Summa(hmotnostniZlomky));
349.             vysledkyLinearni.Add(1);
350.
351.             // M snesi = E (xi * Mi)
352.             float[] podrzRovnici = Summa(pomocnaSummaMolHmot-
  nosti);
353.             podrzRovnici[molarniHmotnostSmesi.indexVPoli] = -1;
354.             linearniRovnice.Add(podrzRovnici);
355.             vysledkyLinearni.Add(0);
356.
357.             // E xi = 1
358.             linearniRovnice.Add(Summa(molarniZlomky));
359.             vysledkyLinearni.Add(1);
360.
361.             // E (wi / Mi) = E(wi / Mi)
362.             podrzRovnici = Summa(pomocnaSummaProVypocetmolar-
  nichZlomku);
363.             podrzRovnici[pomocnaCelkovaSummaProVypocetmolar-
  nichZlomku.indexVPoli] = -1;
```

```
364.             linearniRovnice.Add(podrzRovnici);
365.             vysledkyLinearni.Add(0);
366.
367.             // E mi = m celk
368.             podrzRovnici = Summa(hmotnostiSlozek);
369.             podrzRovnici[celkovaHmotnost.indexVPoli] = -1;
370.             linearniRovnice.Add(podrzRovnici);
371.             vysledkyLinearni.Add(0);
372.
373.             // E ni = n celk
374.             podrzRovnici = Summa(latkoveMnozstvi);
375.             podrzRovnici[celkemMolu.indexVPoli] = -1;
376.             linearniRovnice.Add(podrzRovnici);
377.             vysledkyLinearni.Add(0);
378.
379.             // ro i = hustota
380.             podrzRovnici = Summa(hmotnostniKoncentrace);
381.             podrzRovnici[hustota.indexVPoli] = -1;
382.             linearniRovnice.Add(podrzRovnici);
383.             vysledkyLinearni.Add(0);
384.
385.             // m = hustota * V
386.             nasobiciRovnice.Add(Rovice(new Neznama[] { celkova-
    Hmotnost, hustota, objem }, new float[] { 1, -1, -1 }));
387.             vysledkyNasobici.Add(1);
388.
389.             // m = M * n
390.             nasobiciRovnice.Add(Rovice(new Neznama[] { celkova-
    Hmotnost, molarniHmotnostSmesi, celkemMolu }, new float[] { 1, -1,
    -1 }));
391.             vysledkyNasobici.Add(1);
392.
393.             // mi = Mi * ni
394.             for (int i = 0; i < Uzel.slozek; i++)
395.             {
396.                 nasobiciRovnice.Add(Rovice(new Neznama[] { hmot-
    nostiSlozek[i], molarniHmotnost[i], latkoveMnozstvi[i] }, new
    float[] { 1, -1, -1 }));
397.                 vysledkyNasobici.Add(1);
398.             }
399.
400.             // (wi/Mi) = xi * E (wi/Mi)
401.             for (int i = 0; i < Uzel.slozek; i++)
402.             {
403.                 nasobiciRovnice.Add(Rovice(new Neznama[] { po-
    mocnaSummaProVypocetmolarnichZlomku[i], pomocnaCelkovaSummaProVy-
    pocetmolarnichZlomku, molarniZlomky[i] }, new float[] { 1, -1, -1
    }));
```

```
404.                 vysledkyNasobici.Add(1);
405.             }
406.
407.             // mi = wi * m celk
408.             for (int i = 0; i < Uzel.slozek; i++)
409.             {
410.                 nasobiciRovnice.Add(Rovice(new Neznama[] { hmot-
  nostiSlozek[i], hmotnostniZlomky[i], celkovaHmotnost }, new float[]
  { 1, -1, -1 }));
411.                 vysledkyNasobici.Add(1);
412.             }
413.
414.             // (xi * Mi) = wi * E (xi * Mi)
415.             for (int i = 0; i < Uzel.slozek; i++)
416.             {
417.                 nasobiciRovnice.Add(Rovice(new Neznama[] { hmot-
  nostniZlomky[i], pomocnaSummaMolHmotnosti[i], molarniHmotnostSmesi
  }, new float[] { -1, 1, -1 }));
418.                 vysledkyNasobici.Add(1);
419.             }
420.
421.             // (xi * Mi) = xi * Mi
422.             for (int i = 0; i < Uzel.slozek; i++)
423.             {
424.                 nasobiciRovnice.Add(Rovice(new Neznama[] { po-
  mocnaSummaMolHmotnosti[i], molarniZlomky[i], molarniHmotnost[i] },
  new float[] { 1, -1, -1 }));
425.                 vysledkyNasobici.Add(1);
426.             }
427.
428.             // mi = ro i * V
429.             for (int i = 0; i < Uzel.slozek; i++)
430.             {
431.                 nasobiciRovnice.Add(Rovice(new Neznama[] { hmot-
  nostniKoncentrace[i], hmotnostiSlozek[i], objem }, new float[] { -
  1, 1, -1 }));
432.                 vysledkyNasobici.Add(1);
433.             }
434.
435.             // ni = ci * V
436.             for (int i = 0; i < Uzel.slozek; i++)
437.             {
438.                 nasobiciRovnice.Add(Rovice(new    Neznama[]    {
  molarniKoncentrace[i], latkoveMnozstvi[i], objem }, new float[] { -
  1, 1, -1 }));
439.                 vysledkyNasobici.Add(1);
440.             }
441.
```

```
442.             // ci * Mi = wi * hustota
443.             for (int i = 0; i < Uzel.slozek; i++)
444.             {
445.                 nasobiciRovnice.Add(Rovice(new Neznama[] { molarniKoncentrace[i], molarniHmotnost[i], hustota, hmotnostniZlomky[i] }, new float[] { 1, 1, -1, -1 }));
446.                 vysledkyNasobici.Add(1);
447.             }
448.
449.             // (m - mi) = m - mi
450.             for (int i = 0; i < Uzel.slozek; i++)
451.             {
452.                 linearniRovnice.Add(Rovice(new Neznama[] { celkovaHmotnost, pomocnyRelativniHmotnostniZlomek[i], hmotnostiSlozek[i] }, new float[] { 1, -1, -1 }));
453.                 vysledkyLinearni.Add(0);
454.             }
455.
456.             // (m - mi) * W i = mi
457.             for (int i = 0; i < Uzel.slozek; i++)
458.             {
459.                 nasobiciRovnice.Add(Rovice(new Neznama[] { hmotnostiSlozek[i], pomocnyRelativniHmotnostniZlomek[i], relativnihmotnostniZlomky[i] }, new float[] { 1, -1, -1 }));
460.                 vysledkyNasobici.Add(1);
461.             }
462.
463.             // (n - ni) = n - ni
464.             for (int i = 0; i < Uzel.slozek; i++)
465.             {
466.                 linearniRovnice.Add(Rovice(new Neznama[] { celkemMolu, pomocnyRelativniMolarniZlomek[i], latkoveMnozstvi[i] }, new float[] { 1, -1, -1 }));
467.                 vysledkyLinearni.Add(0);
468.             }
469.
470.             // (n - ni) * X i = ni
471.             for (int i = 0; i < Uzel.slozek; i++)
472.             {
473.                 nasobiciRovnice.Add(Rovice(new Neznama[] { latkoveMnozstvi[i], pomocnyRelativniMolarniZlomek[i], relativniMolarniZlomky[i] }, new float[] { 1, -1, -1 }));
474.                 vysledkyNasobici.Add(1);
475.             }
476.
477.             // (V - V i) = V - V i
478.             for (int i = 0; i < Uzel.slozek; i++)
479.             {
```

```
480.                 linearniRovnice.Add(Rovice(new Neznama[] { objem, pomocnaRelativniObjemovaKoncentrace[i], objemJednotlivychLatek[i] }, new float[] { 1, -1, -1 }));
481.                 vysledkyLinearni.Add(0);
482.             }
483.
484.             // V * vol i = V i
485.             for (int i = 0; i < Uzel.slozek; i++)
486.             {
487.                 nasobiciRovnice.Add(Rovice(new Neznama[] { objemJednotlivychLatek[i], objem, objemovaKoncentrace[i] }, new float[] { 1, -1, -1 }));
488.                 vysledkyNasobici.Add(1);
489.             }
490.
491.             // (V - V i) * Vol i = V i
492.             for (int i = 0; i < Uzel.slozek; i++)
493.             {
494.                 nasobiciRovnice.Add(Rovice(new Neznama[] { objemJednotlivychLatek[i], pomocnaRelativniObjemovaKoncentrace[i], relativniObjemovaKoncentrace[i] }, new float[] { 1, -1, -1 }));
495.                 vysledkyNasobici.Add(1);
496.             }
497.
498.             // (V - V i) * Vol i = V i
499.             for (int i = 0; i < Uzel.slozek; i++)
500.             {
501.                 nasobiciRovnice.Add(Rovice(new Neznama[] { objemJednotlivychLatek[i], pomocnaRelativniObjemovaKoncentrace[i], relativniObjemovaKoncentrace[i] }, new float[] { 1, -1, -1 }));
502.                 vysledkyNasobici.Add(1);
503.             }
504.
505.             // Vi * ro i = mi
506.             for (int i = 0; i < Uzel.slozek; i++)
507.             {
508.                 nasobiciRovnice.Add(Rovice(new Neznama[] { objemJednotlivychLatek[i], hustotaSlozky[i], hmotnostiSlozek[i] }, new float[] { 1, 1, -1 }));
509.                 vysledkyNasobici.Add(1);
510.             }
511.
512.             if (plyn)
513.             {
514.                 // p * V = R * T * n
515.                 nasobiciRovnice.Add(Rovice(new Neznama[] { tlak, objem, teplota, celkemMolu }, new float[] { 1, 1, -1, -1 }));
516.                 vysledkyNasobici.Add(plynovaKonstanta);
```

```
517.
518.                   //pi * V = R * T * ni
519.                   for (int i = 0; i < Uzel.slozek; i++)
520.                   {
521.                       nasobiciRovnice.Add(Rovice(new Neznama[] { parcialniTlak[i], objem, teplota, latkoveMnozstvi[i] }, new float[] { 1, 1, -1, -1 }));
522.                       vysledkyNasobici.Add(plynovaKonstanta);
523.                   }
524.
525.                   // p * Vi = R * T * ni
526.                   for (int i = 0; i < Uzel.slozek; i++)
527.                   {
528.                       nasobiciRovnice.Add(Rovice(new Neznama[] { tlak, objemJednotlivychLatek[i], teplota, latkoveMnozstvi[i] }, new float[] { 1, 1, -1, -1 }));
529.                       vysledkyNasobici.Add(plynovaKonstanta);
530.                   }
531.
532.                   // V = E Vi
533.                   podrzRovnici = Summa(objemJednotlivychLatek);
534.                   podrzRovnici[objem.indexVPoli] = -1;
535.                   linearniRovnice.Add(podrzRovnici);
536.                   vysledkyLinearni.Add(0);
537.
538.                   // ro i = xi
539.                   for (int i = 0; i < Uzel.slozek; i++)
540.                   {
541.                       linearniRovnice.Add(Rovice(new Neznama[] { objemovaKoncentrace[i], molarniZlomky[i] }, new float[] { 1, -1 }));
542.                       vysledkyLinearni.Add(0);
543.                   }
544.
545.                   // p i = xi * p
546.                   for (int i = 0; i < Uzel.slozek; i++)
547.                   {
548.                       nasobiciRovnice.Add(Rovice(new Neznama[] { parcialniTlak[i], molarniZlomky[i], tlak }, new float[] { 1, -1, -1 }));
549.                       vysledkyNasobici.Add(1);
550.                   }
551.               }
552.           }
553.
554.           public static float[] Summa(Neznama[] summa) // funguje pro udělání rovnice jako summy neboli všem vloženým neznámým dá koeficient 1
555.           {
```

```
556.             float[] rovnice = new float[ReseniSoustavyRovnic.ne-
  zname.Length];
557.
558.             foreach (Neznama item in summa)
559.             {
560.                 rovnice[item.indexVPoli] = 1;
561.             }
562.
563.             return rovnice;
564.         }
565.
566.         public static float[] Summa(Neznama[] summa, float koe-
  ficient) // podobné jako klasická summa jen se zde může nastavit
  jiný koeficient než 1
567.         {
568.             float[] rovnice = new float[ReseniSoustavyRovnic.ne-
  zname.Length];
569.
570.             foreach (Neznama item in summa)
571.             {
572.                 rovnice[item.indexVPoli] = koeficient;
573.             }
574.
575.             return rovnice;
576.         }
577.
578.         public static float[] SloucetRovnice(float[][] rovnice)
  // sečte 2 nebo více rovnic
579.         {
580.             float[] vyslednarovnice = new float[rovnice[0].Len-
  gth];
581.
582.             for (int i = 0; i < rovnice.Length; i++)
583.             {
584.                 for (int j = 0; j < rovnice[i].Length; j++)
585.                 {
586.                     vyslednarovnice[j] += rovnice[i][j];
587.                 }
588.             }
589.
590.             return vyslednarovnice;
591.         }
592.
593.         public static float[] Rovice(Neznama[] dosazovat,
  float[] koeficienty) // nastaví rovnici neznámých s určitými koefi-
  cienty (pole Neznámých a koeficientů musí být stejně dlouhé a není
  ošetřeno jelikož vztup jde pouze ode mě)
594.         {
```

```
595.             float[] rovnice = new float[ReseniSoustavyRovnic.ne-
  zname.Length];
596.
597.             for (int i = 0; i < dosazovat.Length; i++)
598.             {
599.                 rovnice[dosazovat[i].indexVPoli]    =    koefi-
  cienty[i];
600.             }
601.
602.             return rovnice;
603.         }
604.
605.         public static float[] Rovice() // vrátí prázdné pole
  floatů o délce potřebné pro jednu rovnici
606.         {
607.             float[] rovnice = new float[ReseniSoustavyRovnic.ne-
  zname.Length];
608.
609.             return rovnice;
610.         }
611.
612.         private static Neznama[] ZmenitDelku(Neznama[] puvodni,
  int novaDelka) // Změní délku pole držící neznámé pro složky proudu,
  když dojde k rozšíření
613.         {
614.             Neznama[] vratit = new Neznama[novaDelka];
615.
616.             for (int i = 0; i < novaDelka; i++)
617.             {
618.                 if (i < puvodni.Length) // pokud už existují tak
  přenesu dál (abych nerušil zadané hodnoty)
619.                 {
620.                     vratit[i] = puvodni[i];
621.                 }
622.                 else // pokud ne tak je nutno vytvořit po vzoru
  předchozího členu
623.                 {
624.                     vratit[i] = new Neznama(vratit[i - 1].max,
  vratit[i - 1].min, vratit[i - 1].jmeno, vratit[i - 1].jednotka,
  vratit[i - 1].indexProudu, i + 1, vratit[i - 1].chciVypsat, vratit[i
  - 1].pozeProPlyn);
625.                 }
626.             }
627.
628.             return vratit;
629.         }
630.
```

```
631.         private static Neznama[][] ZmenitDelkuProudu(Neznama[][] puvodni, int slozek, int proudy, int indexProudu) // změní délku polí při přenastavení počtu proudá či složek
632.         {
633.             Neznama[][] vratit = new Neznama[slozek][];
634.
635.             Proud[] protejsiProudy = ProtejsiProudy(indexProudu);
636.             List<Proud> protejsiProudyCopy = new List<Proud>();
637.
638.             for (int i = 0; i < slozek; i++)
639.             {
640.                 protejsiProudyCopy.AddRange(protejsiProudy);
641.
642.                 vratit[i] = new Neznama[proudy];
643.                 if (i < puvodni.Length) // existuje
644.                 {
645.                     for (int j = 0; j < proudy; j++)
646.                     {
647.                         if (j < puvodni[i].Length) // existuje
648.                         {
649.                             vratit[i][j] = puvodni[i][j];
650.                             protejsiProudyCopy.Remove(puvodni[i][j].doProudu);
651.                         }
652.                         else // vytvoř nový
653.                         {
654.                             vratit[i][j] = new Neznama(vratit[i][j - 1].max, vratit[i][j - 1].min, vratit[i][j - 1].jmeno, vratit[i][j - 1].jednotka, vratit[i][j - 1].indexProudu, i + 1, vratit[i][j - 1].chciVypsat, vratit[i][j - 1].pozeProPlyn, protejsiProudyCopy[0]);
655.                             protejsiProudyCopy.RemoveAt(0);
656.                         }
657.                     }
658.                 }
659.                 else // vytvoř nový
660.                 {
661.                     for (int j = 0; j < proudy; j++)
662.                     {
663.                         vratit[i][j] = new Neznama(vratit[i - 1][j].max, vratit[i - 1][j].min, vratit[i - 1][j].jmeno, vratit[i - 1][j].jednotka, vratit[i - 1][j].indexProudu, i + 1, vratit[i - 1][j].chciVypsat, vratit[i - 1][j].pozeProPlyn, protejsiProudyCopy[0]);
664.                         protejsiProudyCopy.RemoveAt(0);
665.                     }
666.                 }
```

```
667.                }
668.
669.                return vratit;
670.            }
671.
672.            public void Rozsirit(int slozek, int protiproudu) // ini-
  cializuje rozšíření
673.            {
674.                hmotnostiSlozek = ZmenitDelku(hmotnostiSlozek, slo-
  zek);
675.
676.                relativnihmotnostniZlomky   =   ZmenitDelku(relativ-
  nihmotnostniZlomky, slozek);
677.
678.                hmotnostniZlomky   =   ZmenitDelku(hmotnostniZlomky,
  slozek);
679.
680.                molarniZlomky = ZmenitDelku(molarniZlomky, slozek);
681.
682.                relativnihmotnostniZlomky   =   ZmenitDelku(relativ-
  nihmotnostniZlomky, slozek);
683.
684.                molarniKoncentrace   =   ZmenitDelku(molarniKoncen-
  trace, slozek);
685.
686.                hmotnostniKoncentrace  =  ZmenitDelku(hmotnostniKon-
  centrace, slozek);
687.
688.                latkoveMnozstvi = ZmenitDelku(latkoveMnozstvi, slo-
  zek);
689.
690.                pocetMolekul = ZmenitDelku(pocetMolekul, slozek);
691.
692.                molarniHmotnost = ZmenitDelku(molarniHmotnost, slo-
  zek);
693.
694.                koeficientDoJakehoProudu = ZmenitDelkuProudu(koefi-
  cientDoJakehoProudu, slozek, protiproudu, indexProudu);
695.
696.                pomocnaKoeficientDoProudu   =   ZmenitDelkuProudu(po-
  mocnaKoeficientDoProudu, slozek, protiproudu, indexProudu);
697.
698.                pomocnaLatkoveKoeficientDoProudu    =    ZmenitDel-
  kuProudu(pomocnaKoeficientDoProudu,  slozek,  protiproudu,  index-
  Proudu);
699.
700.                relativniMolarniZlomky    =    ZmenitDelku(relativ-
  niMolarniZlomky, slozek);
```

```
701.
702.                objemJednotlivychLatek = ZmenitDelku(objemJednotli-
  vychLatek, slozek);
703.
704.                relativniObjemovaKoncentrace = ZmenitDelku(relativ-
  niObjemovaKoncentrace, slozek);
705.
706.                objemovaKoncentrace = ZmenitDelku(objemovaKoncen-
  trace, slozek);
707.
708.                pomocnaSummaMolHmotnosti = ZmenitDelku(pomocnaSum-
  maMolHmotnosti, slozek);
709.
710.                pomocnaSummaProVypocetmolarnichZlomku = Zmenit-
  Delku(pomocnaSummaProVypocetmolarnichZlomku, slozek);
711.
712.                pomocnyRelativniHmotnostniZlomek = ZmenitDelku(po-
  mocnyRelativniHmotnostniZlomek, slozek);
713.
714.                pomocnyRelativniMolarniZlomek = ZmenitDelku(pomoc-
  nyRelativniMolarniZlomek, slozek);
715.
716.                pomocnaRelativniObjemovaKoncentrace = Zmenit-
  Delku(pomocnaRelativniObjemovaKoncentrace, slozek);
717.
718.                parcialniTlak = ZmenitDelku(parcialniTlak, slozek);
719.
720.                hustotaSlozky = ZmenitDelku(hustotaSlozky, slozek);
721.            }
722.        }
723.    }
```

## Uzel

```
1.namespace BilancniVypocty
2.{
3.    class Uzel
4.    {
5.        public static char[] alpha = "ABCDEFGHIJKLMNOPQRS-
  TUVWXYZ".ToCharArray(); // abeceda
6.
7.        public static Uzel uzel; // zde drží odkaz na jedinou in-
  stanci
8.
9.        public static List<Proud> celkemProudu = new List<Proud>();
  // zde jsou drženy všechny proudy
10.
```

```
11.         public List<Proud> vztupniProudy = new List<Proud>(); //
  vztupní proudy uzle
12.         public List<Proud> vystupniProudy = new List<Proud>(); //
  vystupní proudy uzle
13.         public static int slozek;
14.
15.         public Uzel(int vztupneProudy, int vystupneProudy)
16.         {
17.             uzel = this;
18.
19.             vztupniProudy = NastavProudy(vztupneProudy, celkem-
  Proudu.Count);
20.             celkemProudu.AddRange(vztupniProudy);
21.
22.             vystupniProudy = NastavProudy(vystupneProudy, celkem-
  Proudu.Count);
23.             celkemProudu.AddRange(vystupniProudy);
24.
25.             foreach (Proud item in vystupniProudy)
26.             {
27.                 item.NastavNezname(slozek, vztupneProudy);
28.             }
29.
30.             foreach (Proud item in this.vztupniProudy)
31.             {
32.                 item.NastavNezname(slozek, vystupneProudy);
33.             }
34.         }
35.
36.         private List<Proud> NastavProudy(int pocet, int pocatec-
  niIndex) // inicializuje proudy a vrátí je
37.         {
38.             Proud[] proudy = new Proud[pocet];
39.             for (int i = 0; i < pocet; i++)
40.             {
41.                 proudy[i] = new Proud(i + pocatecniIndex);
42.             }
43.             return proudy.ToList();
44.         }
45.
46.         public void ExtrahujNezname() // extrahuje neznámé z proudů
  a uloží je do pole
47.         {
48.             List<Neznama> nezname = new List<Neznama>();
49.
50.             foreach (Proud item in vztupniProudy)
51.             {
```

```
52.                 nezname.AddRange(item.NeznameDoListu(ne-
  zname.Count, false));
53.             }
54.
55.             foreach (Proud item in vystupniProudy)
56.             {
57.                 nezname.AddRange(item.NeznameDoListu(ne-
  zname.Count, false));
58.             }
59.
60.             ReseniSoustavyRovnic.nezname = nezname.ToArray();
61.         }
62.
63.         public void ExtrahujRovnice() // extrahuje rovnice z proudů
  i rovnice mezi jednotlivými proudy
64.         {
65.             List<float[]> linearniMatice = new List<float[]>(),
  nasobiciMatice = new List<float[]>();
66.             List<float> rovnaseLinearni = new List<float>(), rov-
  naseNasobici = new List<float>();
67.
68.             List<float[]> linearniRovnice, nasobiciRovnice;
69.             List<float> vysledkyLinearni, vysledkyNasobici;
70.
71.             foreach (Proud item in vztupniProudy)
72.             {
73.                 item.VnitroProudniRovnice(out     linearniRovnice,
  out vysledkyLinearni, out nasobiciRovnice, out vysledkyNasobici);
74.                 linearniMatice.AddRange(linearniRovnice);
75.                 nasobiciMatice.AddRange(nasobiciRovnice);
76.                 rovnaseLinearni.AddRange(vysledkyLinearni);
77.                 rovnaseNasobici.AddRange(vysledkyNasobici);
78.             }
79.             foreach (Proud item in vystupniProudy)
80.             {
81.                 item.VnitroProudniRovnice(out     linearniRovnice,
  out vysledkyLinearni, out nasobiciRovnice, out vysledkyNasobici);
82.                 linearniMatice.AddRange(linearniRovnice);
83.                 nasobiciMatice.AddRange(nasobiciRovnice);
84.                 rovnaseLinearni.AddRange(vysledkyLinearni);
85.                 rovnaseNasobici.AddRange(vysledkyNasobici);
86.             }
87.
88.
89.             CelkoveBilance(out linearniRovnice, out vysledkyLi-
  nearni, out nasobiciRovnice, out vysledkyNasobici);
90.             linearniMatice.AddRange(linearniRovnice);
91.             nasobiciMatice.AddRange(nasobiciRovnice);
```

```
92.             rovnaseLinearni.AddRange(vysledkyLinearni);
93.             rovnaseNasobici.AddRange(vysledkyNasobici);
94.
95.             ReseniSoustavyRovnic.linearniMatice = linearniMa-
  tice.ToArray();
96.             ReseniSoustavyRovnic.nasobiciMatice = nasobiciMa-
  tice.ToArray();
97.             ReseniSoustavyRovnic.vysledkyLinearni = rovnaseLi-
  nearni.ToArray();
98.             ReseniSoustavyRovnic.vysledkyNasobici = rovnaseNaso-
  bici.ToArray();
99.         }
100.
101.        private void CelkoveBilance(out List<float[]> linear-
  niRovnice, out List<float> vysledkyLinearni, out List<float[]> naso-
  biciRovnice, out List<float> vysledkyNasobici) // rovnice mezi jed-
  notlivými proudy (celkové bilance)
102.        {
103.            linearniRovnice = new List<float[]>();
104.            nasobiciRovnice = new List<float[]>();
105.            vysledkyLinearni = new List<float>();
106.            vysledkyNasobici = new List<float>();
107.
108.            float[] podrzRovnici = Proud.Rovice();
109.
110.            // celková bilance hmotností
111.            foreach (Proud item in vztupniProudy)
112.            {
113.                podrzRovnici[item.celkovaHmotnost.indexVPoli] =
  1;
114.            }
115.
116.            foreach (Proud item in vystupniProudy)
117.            {
118.                podrzRovnici[item.celkovaHmotnost.indexVPoli] =
  -1;
119.            }
120.
121.            linearniRovnice.Add(podrzRovnici);
122.            vysledkyLinearni.Add(0);
123.
124.            // hmotnostní bilance jednotlivých složek
125.            for (int i = 0; i < slozek; i++)
126.            {
127.                podrzRovnici = Proud.Rovice();
128.                foreach (Proud item in vztupniProudy)
129.                {
```

```
130.                     podrzRovnici[item.hmotnostiSlozek[i].inde-
  xVPoli] = 1;
131.                 }
132.
133.                 foreach (Proud item in vystupniProudy)
134.                 {
135.                     podrzRovnici[item.hmotnostiSlozek[i].inde-
  xVPoli] = -1;
136.                 }
137.
138.                 linearniRovnice.Add(podrzRovnici);
139.                 vysledkyLinearni.Add(0);
140.             }
141.
142.             // celková bilance látkového množství
143.             podrzRovnici = Proud.Rovice();
144.             foreach (Proud item in vztupniProudy)
145.             {
146.                 podrzRovnici[item.celkemMolu.indexVPoli] = 1;
147.             }
148.
149.             foreach (Proud item in vystupniProudy)
150.             {
151.                 podrzRovnici[item.celkemMolu.indexVPoli] = -1;
152.             }
153.
154.             linearniRovnice.Add(podrzRovnici);
155.             vysledkyLinearni.Add(0);
156.
157.             // bilance látkového množství jedlotlivých složek
158.             for (int i = 0; i < slozek; i++)
159.             {
160.                 podrzRovnici = Proud.Rovice();
161.                 foreach (Proud item in vztupniProudy)
162.                 {
163.                     podrzRovnici[item.latkoveMnozstvi[i].inde-
  xVPoli] = 1;
164.                 }
165.
166.                 foreach (Proud item in vystupniProudy)
167.                 {
168.                     podrzRovnici[item.latkoveMnozstvi[i].inde-
  xVPoli] = -1;
169.                 }
170.
171.                 linearniRovnice.Add(podrzRovnici);
172.                 vysledkyLinearni.Add(0);
173.             }
```

```
174.
175.             // ámen
176.             for (int i = 0; i < slozek; i++)
177.             {
178.                 NastavRovniceDoProudu(i, vztupniProudy, vystup-
  niProudy, linearniRovnice, vysledkyLinearni, nasobiciRovnice, vy-
  sledkyNasobici);
179.
180.                 NastavRovniceDoProudu(i, vystupniProudy, vztup-
  niProudy, linearniRovnice, vysledkyLinearni, nasobiciRovnice, vy-
  sledkyNasobici);
181.             }
182.         }
183.
184.         private    void    NastavRovniceDoProudu(int    slozka,
  List<Proud> zProudu, List<Proud> doProudu, List<float[]> linear-
  niRovnice,  List<float> vysledkyLinearni,  List<float[]> nasobi-
  ciRovnice,  List<float> vysledkyNasobici) // rovnice obsahující do-
  Proudu
185.         {
186.             float[] podrzRovnici = Proud.Rovice();
187.             for (int j = 0; j < doProudu.Count; j++)
188.             {
189.                 // týkající se hmotností
190.                 podrzRovnici = Proud.Rovice();
191.                 foreach (Proud item in zProudu)
192.                 {
193.                     podrzRovnici = Proud.Rovice();
194.                     podrzRovnici[item.koeficientDoJakeho-
  Proudu[slozka][j].indexVPoli] = 1;
195.
196.                     podrzRovnici[item.hmotnostiSlo-
  zek[slozka].indexVPoli] = 1;
197.
198.                     podrzRovnici[item.pomocnaKoeficientDoPro-
  udu[slozka][j].indexVPoli] = -1;
199.
200.                     nasobiciRovnice.Add(podrzRovnici);
201.                     vysledkyNasobici.Add(1);
202.                 }
203.
204.                 podrzRovnici = Proud.Rovice();
205.
206.                 foreach (Proud item in zProudu)
207.                 {
208.                     podrzRovnici[item.pomocnaKoeficientDoPro-
  udu[slozka][j].indexVPoli] = 1;
209.                 }
```

```
210.
211.                 podrzRovnici[doProudu[j].hmotnostiSlo-
  zek[slozka].indexVPoli] = -1;
212.
213.                 linearniRovnice.Add(podrzRovnici);
214.                 vysledkyLinearni.Add(0);
215.
216.                 // týkající se látkového množství
217.                 podrzRovnici = Proud.Rovice();
218.                 foreach (Proud item in zProudu)
219.                 {
220.                     podrzRovnici = Proud.Rovice();
221.                     podrzRovnici[item.koeficientDoJakeho-
  Proudu[slozka][j].indexVPoli] = 1;
222.
223.                     podrzRovnici[item.latkoveMnoz-
  stvi[slozka].indexVPoli] = 1;
224.
225.                     podrzRovnici[item.pomocnaLatkoveKoeficient-
  DoProudu[slozka][j].indexVPoli] = -1;
226.
227.                     nasobiciRovnice.Add(podrzRovnici);
228.                     vysledkyNasobici.Add(1);
229.                 }
230.
231.                 podrzRovnici = Proud.Rovice();
232.
233.                 foreach (Proud item in zProudu)
234.                 {
235.                     podrzRovnici[item.pomocnaLatkoveKoeficient-
  DoProudu[slozka][j].indexVPoli] = 1;
236.                 }
237.
238.                 podrzRovnici[doProudu[j].hmotnostiSlo-
  zek[slozka].indexVPoli] = -1;
239.
240.                 linearniRovnice.Add(podrzRovnici);
241.                 vysledkyLinearni.Add(0);
242.             }
243.         }
244.
245.         private List<Proud> Rozsirit(List<Proud> proudy, int no-
  vaDelka, int delkaDruheStrany) // nastaví délku pole proudů na chtě-
  nou délku
246.         {
247.             if (proudy.Count > novaDelka)
248.             {
249.                 int delka = proudy.Count;
```

```
250.                 for (int i = delka; i > novaDelka; i--)
251.                 {
252.                     celkemProudu[proudy[i - 1].indexProudu] =
  null; // nutno odstranit i z celkového seznamu proudů
253.                     proudy.RemoveAt(i - 1);
254.                 }
255.                 proudy.RemoveRange(novaDelka, proudy.Count - no-
  vaDelka);
256.             }
257.             else
258.             {
259.                 for (int i = proudy.Count; i < novaDelka; i++)
260.                 {
261.                     proudy.Add(new Proud(celkemProudu.Count));
262.                     celkemProudu.Add(proudy[proudy.Count - 1]);
263.                 }
264.             }
265.             return proudy;
266.         }
267.
268.         public void RozsirProudy(int slozek, int vztup, int vy-
  ztup) // nastaví délky polí na novou délku podle nastavení
269.         {
270.             int puvodniVztup = vztupniProudy.Count;
271.             int puvodniVystup = vystupniProudy.Count;
272.
273.             vystupniProudy = Rozsirit(vystupniProudy, vyztup,
  vztup);
274.
275.             vztupniProudy = Rozsirit(vztupniProudy, vztup, vy-
  ztup);
276.             // nadefinuj nové
277.             for (int i = puvodniVystup; i < vystupniProudy.Count;
  i++)
278.             {
279.                 vystupniProudy[i].NastavNezname(slozek, vztup-
  niProudy.Count);
280.             }
281.
282.             for (int i = puvodniVztup; i < vztupniProudy.Count;
  i++)
283.             {
284.                 vztupniProudy[i].NastavNezname(slozek, vystup-
  niProudy.Count);
285.             }
286.
287.             // předefinuj staré
```

```
288.             for (int i = 0; i < puvodniVystup && i < vystupni-
  Proudy.Count; i++)
289.             {
290.                 vystupniProudy[i].Rozsirit(slozek, vztup);
291.             }
292.
293.             for (int i = 0; i < puvodniVztup && i < vztupni-
  Proudy.Count; i++)
294.             {
295.                 vztupniProudy[i].Rozsirit(slozek, vyztup);
296.             }
297.         }
298.
299.         public static void PrenastavProudIndexy() // nastaví od-
  povídající index v poli proudů
300.         {
301.             for (int i = 0; i < celkemProudu.Count; i++)
302.             {
303.                 if (celkemProudu[i] == null)
304.                 {
305.                     celkemProudu.RemoveAt(i);
306.                 }
307.                 else
308.                 {
309.                     celkemProudu[i].indexProudu = i;
310.                 }
311.             }
312.         }
313.     }
314. }
```

## NastaveniSlozek

```
1.namespace BilancniVypocty
2.{
3.    public partial class nastaveniSlozek : Form
4.    {
5.        private bool ulozit;
6.
7.        public nastaveniSlozek()
8.        {
9.            // nastaví výchozí hodnoty
10.            InitializeComponent();
11.            ulozit = false;
12.            numPocetSlozek.Value = Uzel.slozek;
13.            numVystoupProud.Value      =      Uzel.uzel.vystupni-
  Proudy.Count;
14.            numVztupProud.Value = Uzel.uzel.vztupniProudy.Count;
```

```
15.             FormClosing += new FormClosingEventHandler(PriZa-
  vreni);
16.         }
17.
18.         private void PriZavreni(object sender, System.ComponentMo-
  del.CancelEventArgs e)
19.         {
20.             if (ulozit) // klikl na tlačítko uložit
21.             {
22.                 UlozitHodnoty();
23.                 e.Cancel = false;
24.             }
25.             else
26.             {
27.                 DialogResult odpoved = MessageBox.Show("Chcete na-
  stavení uložit?", "Odejít?", MessageBoxButtons.YesNoCancel,
  MessageBoxIcon.Question, MessageBoxDefaultButton.Button1); // ze-
  ptáme se uživatele
28.
29.                 if (odpoved.Equals(DialogResult.Yes))
30.                 {
31.                     UlozitHodnoty();
32.                     e.Cancel = false;
33.                 }
34.                 else if (odpoved.Equals(DialogResult.No))
35.                 {
36.                     e.Cancel = false;
37.                 }
38.                 else if (odpoved.Equals(DialogResult.Cancel))
39.                 {
40.                     e.Cancel = true;
41.                 }
42.             }
43.
44.             if (!e.Cancel) // pokud vypínáme tak odebereme refe-
  renci z Form1
45.             {
46.                 Form1.nastaveni = null;
47.             }
48.         }
49.
50.         private void Ulozit_Click(object sender, EventArgs e) //
  uživatel klikne na uložit
51.         {
52.             ulozit = true;
53.             Close();
54.         }
55.
```

```
56.         private void UlozitHodnoty() // ulož hodnoty a vše nastav
57.         {
58.             Uzel.slozek = (int)numPocetSlozek.Value;
59.             Uzel.uzel.RozsirProudy(Uzel.slozek,           (int)nu-
  mVztupProud.Value, (int)numVystoupProud.Value);
60.             Uzel.PrenastavProudIndexy();
61.         }
62.     }
63. }
```

## Krmitko

```
1.namespace BilancniVypocty
2.{
3.    public partial class Krmitko : Form // žer ty hodnoty
4.    {
5.        Proud[] proudy;
6.        Neznama[] nezname;
7.        CheckBox[] chkBoxy;
8.
9.        NumericUpDown[] numeric;
10.
11.           int[] indexy; // udrzuje indexy neznámých, které byly
  použitelné
12.
13.           int indexik; // index aktuálního proudu
14.
15.           public Krmitko(Proud[] proudy, string jmeno)
16.           {
17.               this.proudy = proudy;
18.               indexik = 0;
19.               InitializeComponent();
20.               this.Name = jmeno;
21.               this.Text = jmeno;
22.               Vygeneruj(indexik);
23.               FormClosing += Zaviracka;
24.
25.               proudik.ItemHeight = proudy.Length;
26.
27.               foreach (Proud item in proudy) // výběr proudů
28.               {
29.                   proudik.Items.Add("Proud: " + (item.indexProudu
  + 1));
30.               }
31.               proudik.SelectedIndex = 0;
32.           }
33.
```

```
34.         private void Vygeneruj(int cisloProudu) // vygeneruje nastavení neznámých
35.         {
36.             panel.Controls.Clear(); // vyčistí panel
37.
38.             nezname = proudy[cisloProudu].NeznameDoListu(0, true).ToArray(); // extrahuje neznámé z určitého proudu
39.
40.             // vyresetuje pole
41.             List<CheckBox> boxy = new List<CheckBox>();
42.             List<NumericUpDown> num = new List<NumericUpDown>();
43.
44.             chkPlyn.Checked = proudy[cisloProudu].plyn; // nastaví jestli je daný proud plynný
45.
46.             int odestup = 0; // odestup při generování
47.
48.             List<int> listIndexu = new List<int>(); // reset
49.
50.             for (int i = 0; i < nezname.Length; i++)
51.             {
52.                 if (!nezname[i].chciVypsat) // pokud nechci vypsat přeskoč
53.                 {
54.                     continue;
55.                 }
56.                 else if (!proudy[cisloProudu].plyn && nezname[i].pozeProPlyn) // pokud jsi nejsi plyn a nechci plynné neznámé přeskoč
57.                 {
58.                     continue;
59.                 }
60.
61.                 odestup += 30; // přidá k odestupu
62.
63.                 boxy.Add(NastavChkbox(odestup, i, nezname[i].known)); // přidá nový box do listu
64.                 panel.Controls.Add(boxy[boxy.Count - 1]); // přidá na panel
65.
66.                 panel.Controls.Add(NastavLabel(odestup, nazev.Location.X, i, nezname[i].GetName()));
67.
68.                 panel.Controls.Add(NastavLabel(odestup, jednotka.Location.X, i, nezname[i].jednotka));
69.
```

```
70.                 num.Add(NastavNumericUpDown(odestup,  i,  nezname[i].value, nezname[i].min, nezname[i].max, nezname[i])); // přidá do listu
71.                 panel.Controls.Add(num[num.Count - 1]); // přidá na panel
72.
73.                 listIndexu.Add(i); // přidá použitelný index
74.             }
75.
76.             numeric = num.ToArray();
77.             chkBoxy = boxy.ToArray();
78.             indexy = listIndexu.ToArray();
79.         }
80.
81.
82.         private CheckBox NastavChkbox(int odestup, int poradi, bool known) // krátká varianta pro generické
83.         {
84.             return NastavChkbox(odestup, poradi, known, OnChkChanged);
85.         }
86.
87.         private CheckBox NastavChkbox(int odestup, int poradi, bool known, EventHandler postChange) // pro plyn checkbox
88.         {
89.             CheckBox check = new CheckBox();
90.             check.Location = new Point(zname.Location.X, odestup);
91.             check.Name = "ChkBox" + poradi;
92.             check.Text = "";
93.             check.Checked = known;
94.             check.CheckedChanged += postChange;
95.
96.             check.Show();
97.             return check;
98.         }
99.
100.        private Label NastavLabel(int odestup, int xPos, int poradi, string text)
101.        {
102.            Label label = new Label();
103.            label.Location = new Point(xPos, odestup);
104.            label.Name = text + poradi;
105.            label.Text = text;
106.
107.            label.Show();
108.            return label;
109.        }
```

```
110.
111.         private NumericUpDown NastavNumericUpDown(int odestup,
  int poradi, float value, float min, float max, Neznama neznama)
112.         {
113.             NumericUpDown num = new NumericUpDown();
114.             num.Location = new Point(hodnota.Location.X, ode-
  stup);
115.             num.Name = "Num" + poradi;
116.             num.Minimum = (decimal)min;
117.             num.Maximum = (decimal)max;
118.             try
119.             {
120.                 num.Value = (decimal)value;
121.             }
122.             catch // debug
123.             {
124.                 num.Value = num.Minimum;
125.             }
126.             num.DecimalPlaces = 7;
127.             num.ValueChanged += OnNumChanged;
128.
129.             return num;
130.         }
131.
132.         private void OnNumChanged(object sender, EventArgs e)
133.         {
134.             int index = IndexSenderu(sender, numeric); // vytra-
  suje sendera v poli objektu
135.             int indexNezname = indexy[index]; // prevede na index
  neznámé
136.             nezname[indexNezname].value = (float)numeric[in-
  dex].Value; // změní hodnotu neznámé
137.             chkBoxy[index].Checked = true; // změní hodnotu
  známe
138.         }
139.
140.         private void OnChkChanged(object sender, EventArgs e)
141.         {
142.             int index = IndexSenderu(sender, chkBoxy); // vytra-
  suje sendera v poli objektu
143.             int indexNezname = indexy[index]; // prevede na index
  neznámé
144.             nezname[indexNezname].known = chkBoxy[index].Chec-
  ked; // nastavi známe
145.         }
146.
147.         private void OnPlynChanged(object sender, EventArgs e)
148.         {
```

```
149.             proudy[indexik].plyn = chkPlyn.Checked;
150.
151.             Vygeneruj(indexik);
152.         }
153.
154.         private int IndexSenderu(object sender, object[] ob-
  jekty) // vrátí index v poli
155.         {
156.             for (int i = 0; i < objekty.Length; i++)
157.             {
158.                 if (sender.Equals(objekty[i]))
159.                 {
160.                     return i;
161.                 }
162.             }
163.
164.             return -1;
165.         }
166.
167.         private void btnDalsiProud_Click(object sender, Even-
  tArgs e)
168.         {
169.             indexik += 1;
170.             if (indexik == proudy.Length)
171.             {
172.                 indexik -= proudy.Length;
173.             }
174.
175.             proudik.SelectedIndex = indexik;
176.         }
177.
178.         private void btnPredchoziProud_Click(object sender,
  EventArgs e)
179.         {
180.             indexik -= 1;
181.             if (indexik < 0)
182.             {
183.                 indexik += proudy.Length;
184.             }
185.
186.             proudik.SelectedIndex = indexik;
187.         }
188.
189.         private void Zaviracka(object sender, System.Component-
  Model.CancelEventArgs e)
190.         {
191.             e.Cancel = false;
```

```
192.             Form1.krmitko = null; // zruš odkaz na krmítko ve
  forms
193.         }
194.
195.         private void OnProudChanged(object sender, EventArgs e)
  // když se změní proud
196.         {
197.             indexik = proudik.SelectedIndex;
198.             Vygeneruj(indexik);
199.         }
200.     }
201. }
```

## Form1

```
1.Name spáče BilancniVypocty
2.{
3.    public partial class Form1 : Form
4.    {
5.        public static nastaveniSlozek nastaveni = null; // udržuje
  odkaz na nastavení složek
6.        public static Krmitko krmitko = null; // udržuje odkaz na
  krmítko
7.        public static bool pocitam = false; // probíhají výpočty?
8.
9.        private static List<Button> buttony =  new List<Button>();
  // list všech čudlíků ve formu
10.
11.         public Form1()
12.         {
13.             InitializeComponent();
14.         }
15.
16.         private void Form1_Load(object sender, EventArgs e)
17.         {
18.             Uzel.slozek = 3;
19.             Uzel uzlik = new Uzel(2, 3);
20.
21.             buttony.Add(vypocet);
22.             buttony.Add(btnNastaveniSlozek);
23.             buttony.Add(vztup);
24.             buttony.Add(vyztup);
25.             buttony.Add(resetBTN);
26.         }
27.
28.         private  void  btnNastaveniSlozek_Click(object  sender,
  EventArgs e)
29.         {
```

```
30.             if (nastaveni == null && krmitko == null) // zkon-
  troluji jestli není něco otevřené
31.             {
32.                 nastaveni = new nastaveniSlozek();
33.                 nastaveni.Show();
34.             }
35.             else
36.             {
37.                 MessageBox.Show("Může být otevřeno pouze jedno
  okno!",   "Error",   MessageBoxButtons.OK,   MessageBoxIcon.Error,
  MessageBoxDefaultButton.Button1);
38.             }
39.         }
40.
41.         private void vztup_Click(object sender, EventArgs e)
42.         {
43.             if (nastaveni == null && krmitko == null && !pocitam)
44.             {
45.                 krmitko   =   new   Krmitko(Uzel.uzel.vztupni-
  Proudy.ToArray(), "Nastavení vztuních proudů");
46.                 krmitko.Show();
47.             }
48.             else
49.             {
50.                 MessageBox.Show("Může být otevřeno pouze jedno
  okno!",   "Error",   MessageBoxButtons.OK,   MessageBoxIcon.Error,
  MessageBoxDefaultButton.Button1);
51.             }
52.         }
53.
54.         private void vyztup_Click(object sender, EventArgs e)
55.         {
56.             if (nastaveni == null && krmitko == null && !pocitam)
57.             {
58.                 krmitko   =   new   Krmitko(Uzel.uzel.vystupni-
  Proudy.ToArray(), "Nastavení vztuních proudů");
59.                 krmitko.Show();
60.             }
61.             else
62.             {
63.                 MessageBox.Show("Může být otevřeno pouze jedno
  okno!",   "Error",   MessageBoxButtons.OK,   MessageBoxIcon.Error,
  MessageBoxDefaultButton.Button1);
64.             }
65.         }
66.
67.         private void Vypocet_Click(object sender, EventArgs e)
68.         {
```

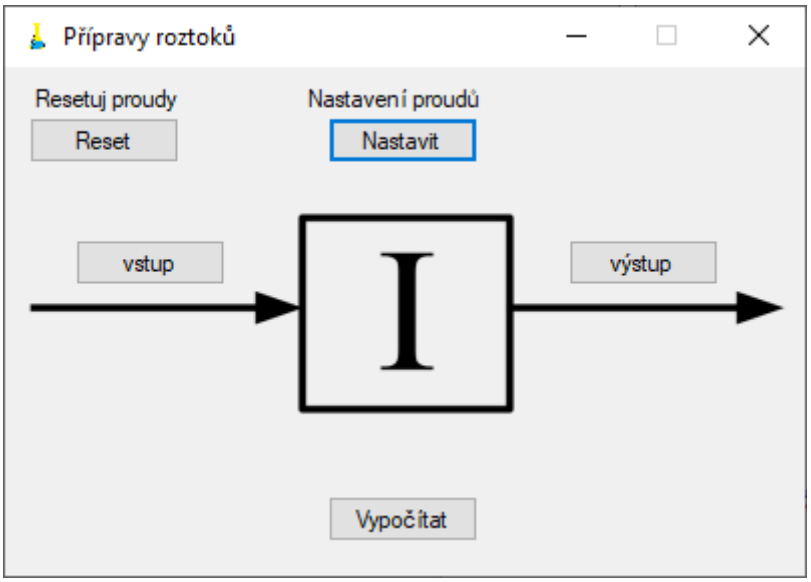
```
69.             if (nastaveni == null && krmitko == null && !pocitam)
  // zkontroluji jestli se něco neděje
70.             {
71.                 pocitam = true;
72.
73.                 async void karel () { await Task.Run(() => Vy-
  pocty()); }; // asychroní chování, aby nám to nelagovalo
74.
75.                 karel(); // co by to bylo za kód bez karla
76.
77.                 // zakáži uživateli vše
78.                 ZapniButtony();
79.             }
80.             else
81.             {
82.                 MessageBox.Show("Zavřete prosím nastavení.",
  "Error", MessageBoxButtons.OK, MessageBoxIcon.Error, MessageBoxDe-
  faultButton.Button1);
83.             }
84.         }
85.
86.         public static void Vypocty()
87.         {
88.             // získám neznámé a rovnice
89.             Uzel.uzel.ExtrahujNezname();
90.             Uzel.uzel.ExtrahujRovnice();
91.
92.             // pokusím se získat co nejvíce pouhým dosazením
93.             ReseniSoustavyRovnic.DosazeniDoRovnic();
94.
95.             // pokusím si pomoc gausovou metodou
96.             LinearniCast();
97.             while (true) // dělám dokud se něco zjišťuje
98.             {
99.                 // zkusím i úpravu násobné rovnice
100.                if (!NasobiciCast())
101.                {
102.                    break;
103.                }
104.
105.
106.                if (!LinearniCast())
107.                {
108.                    break;
109.                }
110.            }
111.
```

```
112.             Kontrola();  //  zkontroluje  jestli  jsou  hodnoty
  vrámci mezí
113.
114.             ReseniSoustavyRovnic.RESET(); // nastavím vše na pů-
  vodní stav se zachovanými výsledky
115.
116.             pocitam = false;
117.
118.             // povolím uživateli vše
119.             ZapniButtony();
120.         }
121.
122.         private static void ZapniButtony() // zakáže / povolíme
  buttony
123.         {
124.             foreach (Button item in buttony)
125.             {
126.                 // netuším proč, netuším jak, ale funguje to
  (nešmatat => křehké)
127.                 item.BeginInvoke(new Action(() => {
128.                     item.Enabled = !pocitam;
129.                     }));
130.             }
131.
132.             Application.UseWaitCursor = pocitam;
133.             Cursor.Current = Cursors.Default; // pokud není miší
  na aplikaci hrozí, že by mu zůstalo kolečko
134.         }
135.
136.
137.         private static bool LinearniCast() // Gausova eliminace
138.         {
139.             ReseniSoustavyRovnic.PredPripravLinearni();
140.             return      ReseniSoustavyRovnic.UpravaLinearniRov-
  nice();
141.         }
142.
143.         private static bool NasobiciCast() // metoda pro úpravu
  násobící rovnice
144.         {
145.             ReseniSoustavyRovnic.PredPripravNasobici();
146.             return ReseniSoustavyRovnic.UpravaNasobneRovnice();
147.         }
148.
149.         private void resetBTN_Click(object sender, EventArgs e)
150.         {
151.             if (nastaveni == null && krmitko == null && !pocitam)
152.             {
```

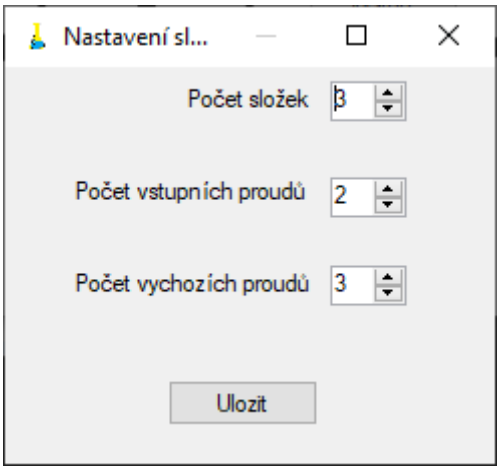
```
153.                 Uzel uzlik = new Uzel(Uzel.uzel.vztupni-
  Proudy.Count, Uzel.uzel.vystupniProudy.Count); // vyresetuje obsah
  všech proudů jelikož se znovu
154.             }
155.             else
156.             {
157.                 MessageBox.Show("Nelze udělat během provádějí
  jiné akce.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error,
  MessageBoxDefaultButton.Button1);
158.             }
159.         }
160.
161.         public static void Kontrola()
162.         {
163.             foreach (Neznama item in ReseniSoustavyRovnic.ne-
  zname)
164.             {
165.                 if (!item.known)
166.                 {
167.                     continue;
168.                 }
169.
170.                 if (item.value > item.max || item.value <
  item.min)
171.                 {
172.                     MessageBox.Show("Hodnota " + item.GetName()
  + " je mimo hranice možných hodnot" + Environment.NewLine + "Hodnota
  je " + item.value + ")", "out of bounds", MessageBoxButtons.OK,
  MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
173.                 }
174.             }
175.         }
176.     }
177. }
```

# Ukázky výsledné GUI aplikace

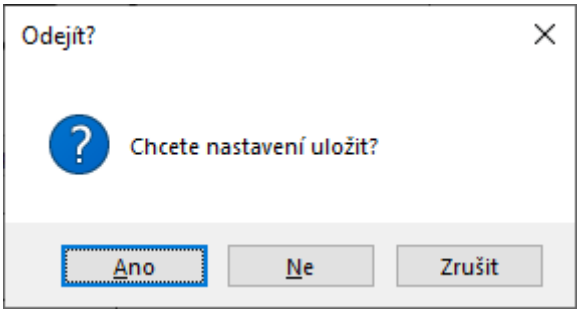
Základní okno (otevřeno celou dobu)



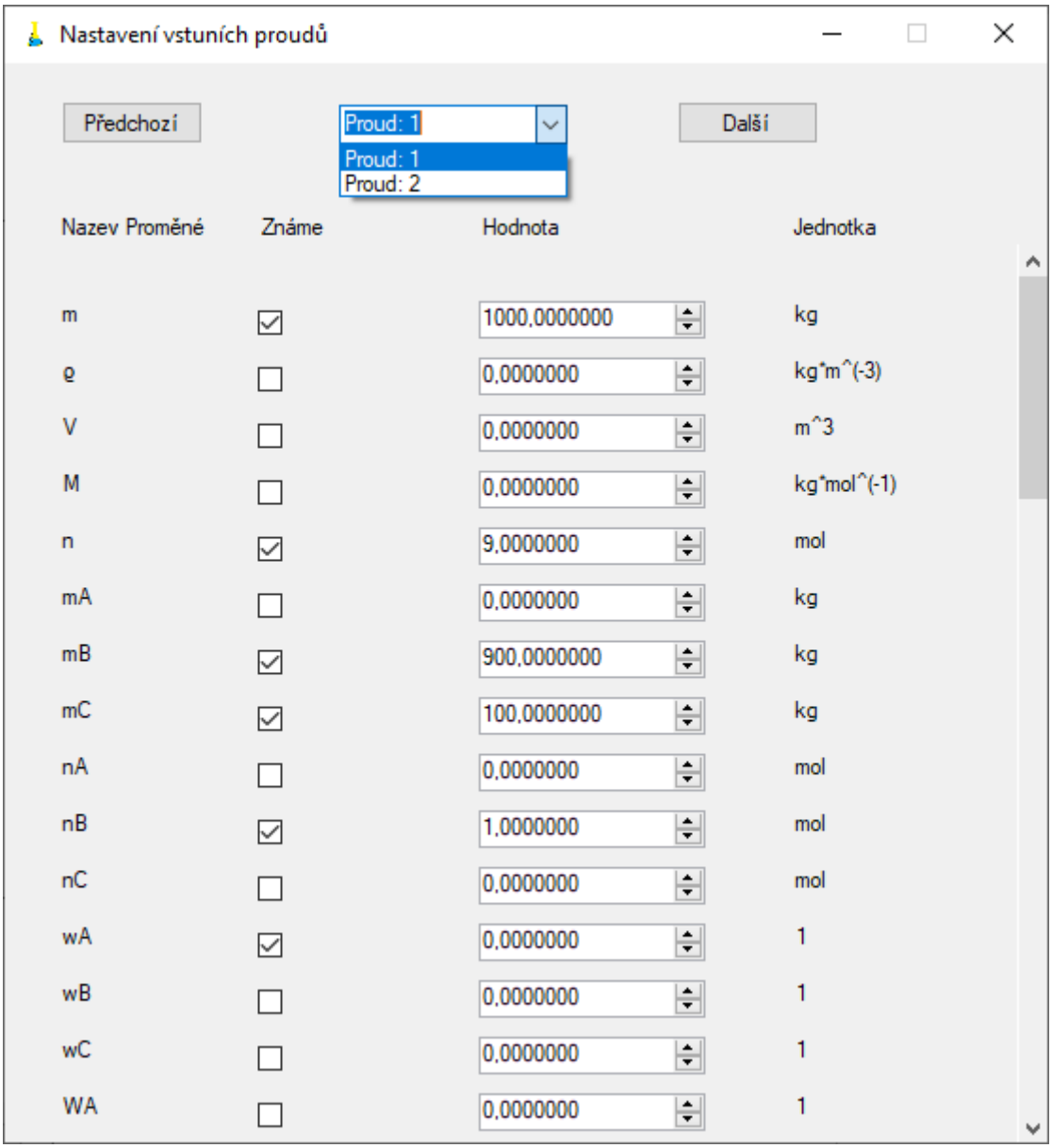

Nastavení proudů




Dialogové okno, které se otevře při zavírání tohoto okna




Okno na nastavení jednotlivých neznámých v proudech




# Závěr

Program lze používat na přípravu roztoků, separace a krystalizace. Většinu úloh vyřeší. Občas je však nutno zadat více hodnot, než by bylo nutné v ideálním případě. Toto je způsobeno špatnou schopností vytýkat a dosazovat mezi rovnicemi pro *Gaussovu eliminaci* a pro metodu úpravy násobících rovnic. Retrospektivně bych ohledně struktury kódu uvážil rozdělení třídy Proud do dvou tříd Proud a Složka, jelikož tato třída je nejméně přehledná ze všech. Rychlostí výpočtu jsem však příjemně překvapen, přestože nedochází k multithreadingu v kódu. Vygenerování interaktivního seznamu na slabších počítačích může trvat pouhou vteřinu. Zadávání do tohoto seznamu je velice intuitivní. Celkově bych ocenil praktičnost oken a informování uživatele ohledně problémů, které mohou během programu nastat.

# Zdroje

Panel Control (Windows Forms). *Microsoft dokumentace* [online]. [cit. 2022-01-06]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/controls/panel-control-windows-forms?view=netframeworkdesktop-4.8

How can I make the cursor turn to the wait cursor? *Stackoverflow* [online]. [cit. 2022-01-06]. Dostupné z: https://stackoverflow.com/questions/1568557/how-can-i-make-the-cursor-turn-to-the-wait-cursor

Form.Closing Event. *Microsoft dokumentace* [online]. [cit. 2022-01-06]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.form.closing?view=windowsdesktop-6.0