2. feladat: Gyorsulás

Egy *m* tömegű vonat mozdonya *F* vonóerővel gyorsítja a szerelvényt. Mennyi idő alatt tesz meg *s* utat és mekkora sebességet ér el, ha kezdetben v_k sebességgel haladt?

A megoldáshoz fizikából ismert összefüggések:

$a = \frac{F}{m}$; a vonat gyorsulása. A gyorsító (húzó-vonó) erő a mozgás irányával azonos, ezért *F* pozitív, az *m* mindig pozitív, a gyorsulás is pozitív.

$s = \frac{a}{2} \cdot t^2 + v_k \cdot t$; egyenletesen gyorsuló mozgás útja az időtől négyzetesen függ; *s*, *t* és v_k is pozitív

$v_t = a \cdot t + v_k$; az egyenletesen gyorsuló mozgás sebessége lineárisan változik.

Feladat megértése

Felhasználva az előző feladat tapasztalatait, célszerűnek látszik az összefüggések áttekintése, értelmezése. Az *a* értékét nem érdemes kiszámolni, jobb, ha behelyettesítjük a második és harmadik képletbe az *F*/*m* értéket. Ezt követően látható, hogy a második képletben az idő, t az egyetlen ismeretlen, erre másodfokú. Ha ebből *t*-t kiszámoljuk, az eredményt a harmadik egyenletbe behelyettesítve a v_t elért sebesség is kiszámítható.

A bemenet minden adata pozitív. SI mértékegységekkel számolva m-re és F-re az egész (természetes) szám elegendően pontos érték; a tömeget – vasúti kocsik esetén – tonnában szokás megadni, de most kg-os pontossággal számolunk. A sebességet km/h-ban egész értékkel szoktuk megadni, de most a m/s miatt valós számot érdemes használni. Az út hosszára – méterben – megfelelő az egész szám. A kimenet adatai – a képlet alapján –valós értékek lesznek.

A specifikációban csak a be- és kimenet közötti kapcsolatot kell megadni, nem a kiszámítás szabályát. De annak eldöntéséhez, hogy tényleg van-e megoldás, hány megoldás van, érdemes átrendezni a képleteket, kifejezni a kiszámítandó *t*-t

$$\frac{a}{2} \cdot t^2 + v_k \cdot t - s = 0 \quad \overset{a=\frac{F}{m}}{\Longrightarrow} \quad \frac{F}{2 \cdot m} \cdot t^2 + v_k \cdot t - s = 0$$

A másodfokú egyenlet megoldóképlete alapján:

$$t = \frac{-v_k \pm \sqrt{{v_k}^2 + 4 \cdot \frac{F}{2 \cdot m} \cdot s}}{2\frac{F}{2 \cdot m}} \quad \Rightarrow \quad t = \frac{-v_k \pm \sqrt{{v_k}^2 + 2 \cdot \frac{F \cdot s}{m}}}{\frac{F}{m}}$$

Látható, hogy az F/m kétszer szerepel a képletben, ráadásul még a v_k kiszámításához is szükség lesz rá, praktikusabbnak tűnik segédadatként kiszámolni. Ellene szól, hogy – ahogy azt az előző feladatnál láthattuk, a számítás pontosságát a számított hányadossal számolás jelentősen befolyásolja, de most ennél nagyobb problémát jelent a gyökvonás. Nézzük meg emeletes tört nélkül, illetve a gyorsulás értékét használva a számítás módját:

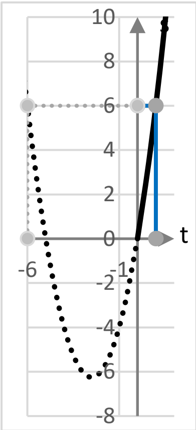
$$t = \frac{-v_k \cdot m \pm \sqrt{(v_k \cdot m)^2 + 2 \cdot F \cdot m \cdot s}}{F} \quad vagy \quad t = \frac{-v_k \pm \sqrt{{v_k}^2 + 2 \cdot a \cdot s}}{a}$$

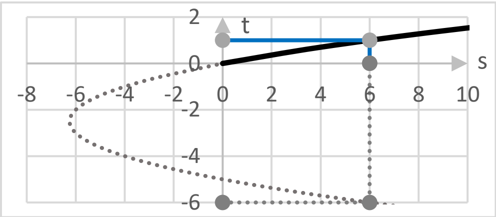
Hm … *F* és *m* egész, de *a* valós, ezért a baloldali képlet még gyökvonással együtt is pontosabbnak tűnik …

Mivel *m* és *F* is pozitív, a másodfokú egyenlet főegyütthatója nem lehet nulla. A v_k, az *F, az s* és az *m* is pozitív, ezért a diszkrimináns az előfeltételeket teljesítő adatokkal pozitív lesz, nem okozhat problémát a gyökvonás. A feladatban előírt feltételek elégségesek ahhoz, hogy a diszkrimináns pozitív legyen, nem szükséges további vizsgálat.

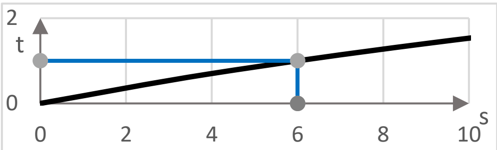
A diszkriminánst vizsgálva, az is látszik, hogy négyzetgyöke nagyobb lesz v_k-nál, ezért t csak a + (pozitív) ágon lehet pozitív. Fizikai hátterét tekintve: a diszkrimináns kivonásával kapott negatív *t* idő a mérés kezdete előtti „végső állapotot” jelentené: valamikor, régen, a vonat s úton v_t nagyságú sebességről v_k nagyságú sebességre lassított; ezután megállt, visszaindult és elérte a v_k sebességet.

Matematikai absztrakcióval, az s(t) függvény grafikonja egy parabola, ami t = 0 kezdőidőpontban s(0) = 0 értékű. Ebben a pontban a pozitív kezdősebesség a grafikonhoz t = 0 ponthoz húzott érintőjének pozitív meredekségével adható meg. A feladatban megadott *s* úthoz két *t* időpont tartozik. Az egyik a kezdőidőpont előtt, a „múltban”, a másik a kezdő időpont után, a „jövőben” van.





Egy húzó-vonó erőnek a jövőre van csak hatása. Jól látszik a függvény inverzén a t(s) függvény értelmezési tartománya.



Specifikáció

Két specifikációt írunk most. Az első az eredeti összefüggések alapján készül, a második a számításainkat is figyelembe veszi.

Eredeti összefüggések alapján

Be: m ∈ N, f ∈ Z, s ∈ N, vk ∈ R
Sa: a ∈ R
Ki: t ∈ R, vt ∈ R
Ef: m > 0 és f > 0 és vk > 0 és s > 0
Uf: a = f/m és
s = (a / 2) * t * t + vk * t és
vt = a * t + vk

```
m: 1000
f: 2000
s: 6
vk: 5
t: 1
vt: 7
a: 2
```

Link:

https://progalap.elte.hu/specifikacio/?data=H4sIAAAAAAAACk2NQUoDQRBFr%2FKplcaGzAyoUDBZCK4CWRjcaFwUsRuGcUZJlb0J2ecqnsObeBKpVicuinr%2F%2F%2BLXnvQtbrvUbcW615GYbiJjwNfxiFVAKvAQoH9O7gvdbXabcS0MOcllx7AfGZDtFNwmr1ygwueHIk2U%2BwnVyW%2Fvk3e2SPPBA7cAQNHiTDBHc44ZrMyFFzj8u8uGFjLFFMiimhI%2F7ulZTIhpYNRVVZ4lRvOLy-rjynXvGpYMx6mIY49pBGA0FGmWIxESBdlHfX4y4PjwdvgHQiRuLSAEAAA%3D%3D

Számított összefüggések alapján

Be: m ∈ N, f ∈ Z, vk ∈ R, s ∈ N
Ki: t ∈ R, vt ∈ R
Ef: m > 0 és f > 0 és vk > 0 és s > 0
Uf: t = (-vk * m + sqrt(vk * vk * m * m + 2 * f * m * s)) / f és
vt = f * t / m + vk

```
m: 1000
f: 2000
s: 6
vk: 5
t: 1
vt: 7
```

Link:

https://progalap.elte.hu/specifikacio/?data=H4sIAAAAAAAACkWOQWoCQRBFr%2FKplZoWZ4REKDALIatAFoIbYxaDdsPQmVGnKr0R914l58hNPEmodmJW9f7%2Fxa86kRz8tg71ttJ63xLTwjMaXC8XvDmEDGsH%2BXNSzLTcdJv2tWboTTok%2FQ9egnU8o8DPtyDcKcU7ipHtroKVzDEYp4gRGjxAjp0Osuqtmz3FCK-HXMhxigmBVVgLAHpjnXDHJ%2BymSI%2FWiQvx%2Bol2lFTE1jLIo8unAmPYojCebKTIeDZRRZkMZM3LUVo0nJnLUefn6VOLy%2FHH%2BBZPKaOA%2FAQAA

Önálló feladat

> Egészítsük ki a megoldást további esetekkel, tesztekkel. Legyen közötte olyan is, amelyik nem teljesíti az előfeltételt és olyan is, ahol a gyorsulás nem egész szám és nem is egy egész szám reciprokja. Kezdősebességnek se csak egész számot válasszunk, például 75 km/h ~ 20,833 m/s

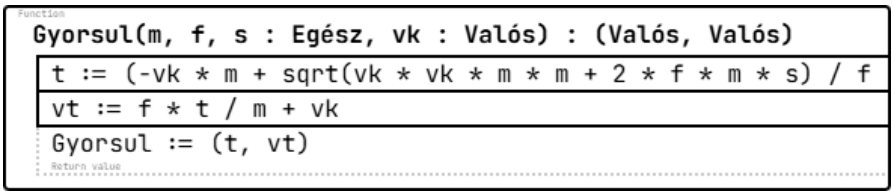
Visszavezetés

A teljes másodfokú egyenlet $ax^2 + bx + c = 0$ megoldásának képlete, azaz az x kiszámításának szabálya: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.
Ez a képlet mutatja meg, hogy a feladatban megadott összefüggésből hogyan képezzük majd az algoritmusban a kiszámítást. A t kiszámítását visszavezetjük a másodfokú egyenlet megoldóképletére.

a	~ **a/2** (vagy **f/m/2**)
b	~ **vk**
c	~ **-s**
x_1	~ **t**
x_2	~ nem reális, t > 0 miatt

Algoritmus

```
Adatok:
   m, s  :  Természetes  //nem nulla
   f  :  Egész  // pozitív
   vk, t, vt :  Valós  //vk > 0
```



https://progtlap.elte.hu/stuki/?data=H4sIAAAAAAAACq2T3W7TMBiGb8X6IqF2GGZIaZoYkQASPzvgZEOTYOqBa39uoj6ZOZzuBrcoFscwl7MaQna5NNHEER3792H6%2FVk2BQAGwUc2OxL2axIByzNsImep0mRAYV11aC9UsBN0XZYUH/YoPaoTMZVCB%2FwAX67L%2BVbVKuhoeAQU9sKI8adH6415aIBrUTqk4B%2F2CBwc3tdoJAI-Fud6UyqXSUg74HRQ3mLzL849AwX67%2B4Y4%2BSWHR0hMfYo4unmKOMK2FILZCB7tB4oODxpwcOnvC3ZPKq2ZzuMv4bu3fhi3PepwQs6i7vduSi6ihnaUSJDXEfxzXk1MLiT05CKm0GyhbSk4b2vpq5UVu9CzAxsCx4vzNLt33x6gKyry8mOEk2Jl5x8W039c08I0MPjrSiffnsp4WTSSTqg7JQaITdiGUZhg1hDIX1rAi3rn7cng71tQT6OYVCC9zbuIfX6GStrvsaIOmpHBPp8l4xA8/Y2FQru-gIKuysvFz6I9pxB2daMBYYUDqLq9ZBa52PbMlmR6WjmrNLJqKSKHQN2BhNDKX3t6C9le2qo3qjZfJMgnXwjoYoxwYzqUYIlSKpZIhEujzCX%2FET7UazfeYqpZSxFinl4KqTgaFWsmNF0ZOyIdtiFhQjpiobh6Ft%2FwCBLZOpDQQAAAN3DN3D

Másképp is elkészíthetjük az algoritmust. Lehet ehhez hasonlóan, de eljárásként vagy két függvényben kiszámolva az eredményeket. Lehet segédadattal vagy anélkül … a kódban pedig még nagyobb változatosságra ad lehetőséget a beolvasás és kiírás megvalósítása.

A specifikációban az `m` és `s` értékét a természetes számok halmazából ($\mathbb{N}$) választottuk ki. A természetes számok halmazába itt a nullát is beleértjük, de ez nem magától értetődő. Az aritmetika alapjait leíró Peano-axiómákat a XIX. században fogalmazták meg, ezek közül az első mondja ki, hogy a nulla természetes szám. A többszázéves hagyomány szerint a nulla nem természetes szám. Pontosabban, egy megtanult ismeret, hogy a nulla természetes szám, de természetes nyelvi és fogalmi környezetben a nullát nem gondoljuk természetes számnak. Ezért praktikus lehet a természetes számok halmazát ($\mathbb{N}$) is Egész típusú adatnak jelölni és előfeltételként megadni, hogy melyik tartományból származhatnak az értékek.

Kód és tesztelés

Kicsit unalmas már a négy szám egymás után megadása, ezért most egy sorban, szóközzel elválasztva adjuk meg a bemenet adatait m, f, s, vk sorrendben, az elvárt formátumot kérő kiírás a másodlagos (log, error) konzolon jelenik meg. Az adatokat gyorsan ellenőrizzük. A változók globálisak, ezért nem szükséges megadni sem paraméterként, sem függvény visszatérési értékeként.

A megoldást egy tizedesjegy pontossággal írjuk ki, mert ennyinek van értelme. A soktizedesjegyes eredmény a specifikációbéli teszteléséhez szükséges.

A tesztesetek a kód végén, megjegyzésben szerepelnek, így nem kell keresgélni.

Csapda: Amikor nagy egész számokat szorzunk össze a gyök alatti második tagban, a szorzat túllépheti az `int`.MaxValue-t, ilyenkor először negatív lesz… de semmiképp sem lesz helyes az eredmény. Mivel

gyököt double értékből tud számolni a C#, praktikus eleve double-béli szorzást végezni, ehhez 2 helyett 2.0 az első szorzótényező.

```
using System;
namespace Gyorsul
{
   internal class Program
   {
      #region bemenet és kimenet deklarációk
      static uint m, s;
      static int f;
      static double vk, t, vt;
      #endregion
      static void Main(string[] args)
      {
         bool sikeres = Be();
         if (sikeres)
         {
            Gyorsul();
            Ki();
         }
      }
      private static void Ki()
      {
         Console.WriteLine(t.ToString("F1") + " " + vt.ToString("F1"));
      }
      private static void Gyorsul()
      {
         t = (-vk * m + Math.Sqrt(vk * vk * m * m + 2.0 * f * m * s)) / f;
         vt = f * t / m + vk;
      }
      private static bool Be()
      {
         Console.Error.WriteLine("Adja meg szóközökkel elválasztva
                                  az m, f, s és vk értékeket!");
         string[] adatok = Console.ReadLine().Split(' '); //régebbi fordítóval is jó
         bool jo = adatok.Length == 4; //pontosan 4 adat van megadva
         jo = jo && uint.TryParse(adatok[0], out m) && m > 0
            && int.TryParse(adatok[1], out f) && f > 0
            && uint.TryParse(adatok[2], out s) && s > 0
            && double.TryParse(adatok[3], out vk) && vk > 0;
         return jo;
      }
   }
   /************* Tesztek *************
1000 2000 6 5
Ki: 1,0 7,0
t: 1
vt: 7
a: 2

7517 875 600 8,611111111
Ki: 51,6 14,6
t: 51.648009995383426
vt: 14.623085136004722
a : 0.1164028202740455

4506 11887 220 20,83333333
Ki: 7,2 39,9
t: 7.240672412578616
vt: 39.93450354045761
a: 2.6380381713271195
    */
}
```

A specifikációból és az algoritmusból pontosan vettük át az adattípust, ezért lett az `s` és az `m` típusa `uint`. A `.TryParse()` így kiszűri a negatív értékeket is, de a 0-ra külön kell figyelni. Emiatt nem lett rövidebb az előfeltétel vizsgálata. Az `uint` használata a számítások során is problémát okozhat. Egyrészt a kivonás korlátozottan értelmes erre a típusra. Értelmes az m <= 0 reláció vizsgálata? Másrészt, szorzás esetén az `int.MaxValue` kétszerese az `uint.MaxValue`. Vajon hogyan értelmezi a szorzást a programunk, ha `int.MaxValue` értéknél nagyobb `uint` értéket szorzunk össze `int` típusú értékkel?

Mivel az `uint` használatából hasznunk nincs, de a számítások kiértékelésében bizonytalanságot jelent a keveredés, ezért a kódba nagyon ritkán vesszük át a specifikációban vagy az algoritmusban jelzett természetes szám típust.

Az `int`, `uint` mellett több más egész típus érhető el C#-ban, amelyek az ábrázolt (értelmezési) tartományban különböznek egymástól.

Már *a specifikáció során figyeljünk arra*, hogy a lehetséges *bemeneti értékekre az alsó és a felső korlát is meg legyen határozva*, illetve *a valós* (tizedesjegyeket is tartalmazó) *értékek esetén legyen megadva az elvárt pontosság*. Ennek megfelelően válasszuk meg az adathoz megfelelő típust.

Mérés rekord és a kiszámítás függvényei

A bemenet adatai egy dologhoz, egy méréshez tartoznak. Akár adatbázisban tároljuk, akár egy Excel munkalapon, egy dologról tárolunk négy adatot. Egy mérés, egy feljegyzés, azaz egy rekord.

Fizika és matematika

Egy rekordba négy halmazból kerül be egy-egy halmazelem: A tömeg értéke – mivel kg-ban vannak megadva – a természetes számok halmazából kerül ki. A gyorsító erő szintén egy eleme a természetes számok halmazának, de mértékegysége Newton…

Az összes lehetséges mérési adatnégyes halmaza a négy halmaz minden lehetséges kombinációjából álló halmaz. A $\mathbb{H}$almaz, ami az összes mérést tartalmazza, az egyes halmazok Descartes-szorzata, azaz

$$\mathbb{H}almaz = \mathbb{T}\ddot{o}meg \times \mathbb{E}rő \times \mathbb{U}t \times \mathbb{S}ebesség$$

Egy mérés a $\mathbb{H}$*almaz* eleme: *mérés* ∈ $\mathbb{H}$*almaz*.

A mérés *m*, *f*, *s* és *vk* adatairól tudjuk, hogy *m* ∈ $\mathbb{T}$*ömeg*, f ∈ $\mathbb{E}$*rő*, s ∈ $\mathbb{U}t$ és vk ∈ $\mathbb{S}ebesség$

Az egyes adatok mérőszámai egy-egy számhalmaz elemei, a mértékegységeik adottak.

$$\mathbb{H}almaz = \mathbb{N}[kg] \times \mathbb{N}[N] \times \mathbb{N}[m] \times \mathbb{R}\left[\frac{m}{s}\right]$$

Specifikáció

A specifikáció alapja a matematikai halmazelmélet és logika. A fenti leírás adja a specifikációban is a leírás alapját, de nagyon nagy kavarodás lenne, ha a halmaz neve és a hozzá tartozó elem neve ennyire más lenne és mindig le kellene írni, hogy melyik halmaznak melyik az eleme. Arra is figyelni kell, hogy ne legyen félreérthető a megnevezés, nem adhatjuk a már foglalt halmazjelöléseket: $\mathbb{R}$, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{L}$, $\mathbb{S}$ $\mathbb{C}$, $\mathbb{K}$.

Ezért legyen a *mérés* ∈ $\mathbb{M}$, amelyben a $\mathbb{S}$ebesség

$$\mathbb{M} = \mathbb{T}\text{ömeg} \times \mathbb{E}rő \times \mathbb{U}\text{t} \times \mathbb{S}\text{ebesség}$$

Az egyes halmazokról tudjuk, hogy a mérőszámaik melyik halmazból származnak:

$\mathbb{T}$ömeg ⊆ $\mathbb{N}$, $\mathbb{E}$*rő* ⊆ $\mathbb{N}$, $\mathbb{U}t$ ⊆ $\mathbb{N}$ és $\mathbb{S}$ebesség ⊆ $\mathbb{R}$

Miért lehet, hogy csak része? Például azért, mert az erő nem lehet 0 (F > 0), de azért is, mert a kezdősebesség nem lehet fénysebesség közeli érték. Ez utóbbit nem jeleztük a specifikációban, ha szükséges az előfeltételben adhatunk meg korlátot.

A *mérés* négy adatáról ez alapján már tudjuk, hogy

mérés.kg ∈ 𝕋ömeg, *mérés.erő* ∈ 𝔼rő, *mérés.ut* ∈ 𝕌t és *mérés.vk* ∈ 𝕊ebesség

Fontos hangsúlyozni, hogy a 𝕋ömeg nem létezik a 𝕄-en kívül, annak egyik „tényezője", ezért a *kg* sem létezik halmazelem (ez most a mérés) nélkül, aminek ő egy adattagja.

A specifikációban amennyire lehet tömörítjük az információkat, ezért nem írjuk ki minden adattagra, hogy minek az eleme. Az elnevezéseink érthetőségének és tömörségének optimális kombinációját próbáljuk megalkotni. Mindent kiírva a mérés típusának definiálásakor, de röviden, az eredeti képletet megadva utófeltételként ezt írhatjuk (1. megoldás):

Be: mérés ∈ M,
M = (kg:Tömeg x ero:Ero x ut:Ut x vk:Sebesség),
Tömeg = N, Ero = N, Ut = N, Sebesség = R
Sa: a ∈ R
Ki: t ∈ R, vt ∈ R
Ef: mérés.kg > 0 és mérés.ero > 0 és mérés.ut > 0 és mérés.vk > 0
Uf: a = mérés.ero/mérés.kg és
mérés.ut = (a / 2) * t * t + mérés.vk * t és
vt = a * t + mérés.vk

```
mérés:
  kg: 1000
  ero: 2000
  ut: 6
  vk: 5
a: 2
t: 1
vt: 7
```

Link:

https://progalap.elte.hu/specifikacio/?data=H4sIAAAAAAAACmVQQUrDUBC9ymNWrQabBlQYiAuhK6kLa1bGxbf%2BhhDbSP4kBEr3vUou4AVyE08ik9Q2kMVn5r1582bm78l923W6SddG0nxHTI%2BWY9m2TdE2Dr%2FHI5ZeXMS7WJYIMckSfm1%2FtjZBDVvkvChy1CiFI0GNKuOV%2FbDOtU0yPbWd5CGePai6SyLp41mNEC%2BqX-xmG6cZ28CnlWKTHHig5VBaby5o3WYIH%2BNCN%2Fylb5COulBFVZUgpYaSGBuHQYTYY0DauP2jgFmJiMEMwxRWke9dDZyXObZXKzUikRfJIrBNH%2FLanTyOGmE4K1jqQJYy57%2Fs90o9HclalMO76tMoYt5oaRqBRGHONlTDuyaOd2Vpilo8K68ovlZ4f3g9%2FFNGtSAYCAAA%3D

Rövidített rekord definícióval és kimenetekre megfogalmazott utófeltétellel így írjuk (2. megoldás):

Be: mérés ∈ M,
M = (kg:N x ero:N x ut:N x vk:R)
Ki: t ∈ R, vt ∈ R
Ef: mérés.kg > 0 és mérés.ero > 0 és
mérés.ut > 0 és mérés.vk > 0
Uf: t = (-mérés.vk * mérés.kg +
sqrt(mérés.vk * mérés.vk * mérés.kg * mérés.kg + 2 * mérés.ero * mérés.kg * mérés.ut)) /
mérés.ero és
vt = mérés.ero * t / mérés.kg + mérés.vk

```
mérés: {kg: 1000, ero: 2000, ut: 6, vk: 5}
t: 1
vt: 7
```

Link

https://progalap.elte.hu/specifikacio/?data=H4sIAAAAAAAACnVRzUrDQBB%2BlY85JbrapKDCQD0InqQeCp6Mh1A3IcQmmp0EIeTeV8lz5E18EtltionV0yzfz3wfOy2Zd73NkmwbS1YWxHSnGbuhr4be4Gu%2Fx1pFVVREssYKXp7ylz6hq9LNWtxoct74VvWQcSTibBuFZnxZ5j7hSMa1l3mKWwSwCUdIV%2BUJ-VssJ1OQOshufEpe1gncxYc8wCTk%2FNI%2FEfFTi%2Fan6bZn7sZwAtuI%2F2lp8H4tj2tQw9OYAN7bqfJVgMU%2F7KWU9pEi0EUP83NJrLDExjQpGm6eMMAgC5T4D7iRYOqAWxrWyV8FVZ1lhhHY2wrghRUW808REiipt6jchDruX7hu4UOcfCwIAAA%3D%3D

Függvény

Az utófeltétel és a függvény között az a nagy különbség, hogy az utófeltételbe logikai kifejezéseket kell írni, a függvénybe viszont hozzárendelési szabályt, azaz lényegében értékadást. Ezért az első megoldás, a mérés.ut = (a / 2) * t * t + mérés.vk * t utófeltételt csak a mérés.ut kiszámításához lehetne függvényként felírni, a t-re nem ad közvetlenül kiszámítási, hozzárendelési szabályt.

A második megoldás alapján már t -re és vk-ra is tudunk függvényt írni. Mivel specifikációról van szó, matematikai jelöléseket és értelmezést használunk. A függvénynek van értelmezési tartománya (domain), értékkészlete (range) és hozzárendelési szabálya.

$$fgv = D_f \to R_{f,}\ x \mapsto fgv(x)\ ahol\ \forall x \in D_f:$$

Másképp:

$$fgv = D_f \to R_f \text{ és } \forall x \in D_f \text{ esetén } \exists y \in R_f \text{ olyan, hogy } y = fgv(x)$$

Például:

$$sqr: R \to R, x \mapsto x^2 \text{ vagy } sqr: R \to R,\ sqr(x) = x^2 \text{ vagy } y = x^2$$

A másodikként felírt specifikációból az idő és sebesség kiszámítását függvénybe kiszervezhetjük. A sebesség esetén az értelmezési tartomány három valós számhalmaz Descartes-szorzatával képzett halmazrendszer, az értékkészlete a valós számhalmaz. Vizuálisan: kiválasztjuk a 3D koordinátarendszer egy tetszőleges pontját, a számhármas meghatároz egy negyedik számot.

Fv: sebesseg: R x R x R → R,
sebesseg(a, t, vk) = a * t + vk

Az előfeltételek miatt a 3D koordinátarendszerrel kifeszített térnek csak az 1/8-a lesz az értelmezési tartomány, de ezt programozásban nem jelöljük. Matematikusan:

$$\text{sebesseg}: \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+, \text{sebesseg}(a, t, vk) = a \cdot t + vk$$

Az idő a mérési adatokból számolható. Szép megoldás, ha ezt a függvényben is így adjuk meg:

Fv: ido: M → R,
ido(m) = (-m.vk * m.kg + sqrt(m.vk * m.vk * m.kg * m.kg + 2 * m.ero * m.kg * m.ut))/m.ero

Ezután, az utófeltételben azt adjuk meg, hogy a kimenet paramétere a függvényértékkel egyenlő:

Uf: t = ido(mérés) és
vt = sebesseg(mérés.ero / mérés.kg, t, mérés.vk)

Link

https://progplap.elte.hu/specifikacio/?data=H4sIAAAAAAAACnWRwWrCQBCGX%2BVnTomuGoW2MKCHQnsp9hDogekh1TWENKbNThZBvPsqPodv0icpu9Fo28Myy%2F%2FM7N8uyXqjRfSKI%2BkkIdrYrXjPd4qiBHg%2B%2F9HnDV1Mk6kTmmCIqMn7GBritfG%2FHFFhyHLvWUcyLJ22iFe7o559EyjH7XxuIMEWNzOoMZYt8AQBcIUgVReEWIKVLDlDJb4jwmX1aM%2Bx%2FOff-kPoUsHg3JoC%2FRQDosMfZvWoJOujhdVOXvuq6unUbCcORVt%2B8hxYmckLjED8EcnbPkmvf6qEyTbOEIN%2FfiSTiN23e3fujCUWrrrA7G9ZoRLu%2FwjC7D%2FQeQJtFGDPHrIpappMROijC2RcYYR1GK0HJz34UFxph3DrkjJudc4UedtUK444LrdNSExMpgrVpPoR4vHvb%2FQAYPHHPwIAAAlL3D%3D

Látható, hogy a függvény bemeneti paramétere szimbolikus. A felhasználás során behelyettesítjük a bemenet adataival, a kiszámított segédadattal, az ezekből kiszámítható kifejezéssel.

A specifikációban a bemenet globális paraméter, ezért a függvényen belül tudunk rá hivatkozni, nem kell megadni értelmezési tartományként.

Fv: idő: **?** → R,
idő() = (-mérés.vk * mérés.kg +
sqrt(mérés.vk * mérés.vk * mérés.kg * mérés.kg + 2 * mérés.ero * mérés.m * mérés.ut)) /
mérés.ero

Az imperatív nyelvekben írt függvények, alprogramok esetén nem okoz gondot, ha nincs bemeneti paraméter. De egy matematikai függvény megadásánál elképzelhetetlen, hogy nincs értelmezési tartomány. Mihez rendelem hozzá az időt? Nagyon csúnya trükk: bármihez, például, valamilyen egész számhoz:

Be: mérés ∈ M,
M = (kg:N x ero:N x ut:N x vk:R)
Ki: t ∈ R, vt ∈ R
Fv: ido: Z → R,
ido(z) = (-mérés.vk * mérés.kg + sqrt(mérés.vk * mérés.vk * mérés.kg * mérés.kg +
2 * mérés.ero * mérés.kg * mérés.ut))/mérés.ero
Ef: mérés.kg > 0 és mérés.ero > 0 és mérés.ut > 0 és mérés.vk > 0
Uf: t = ido(0) és
vt = mérés.ero * t / mérés.kg + mérés.vk

Link

https://progplap.elte.hu/specifikacio/?data=H4sIAAAAAAAACnVRwU7CQBD9IZc5UV2gJVGS5eBgoheD8kvWg8NLK5ptNpON0bCnV%2FhO%2FqTvBTMWkJB3cts3rz3Zvbtmqo3OOsX6SyRtMiJ6cYyVvtdud9V%2BNpuMTFxGeexTDBCJ1vyAz5gy8LXWnxxGU8DZd2mHt842dTANTft3DlGOiBYT%2BlOMfWeAB-TrFAZqJWZGBlyGiBMGvWyJ51TvpXT%2BBi3jzz45HKcczaFFsWfynriUI%2BiZeGt0uOJYWe4wQjGILb7hyrSRfMMq%2Bp4%2BPCxXyKY5B0n6ydggerinon77t6KcaMiSZkor4eU3zRBiIahiMdbZkRGEYmiYR%2FTBMPFAL49roP%2BJqo11hRFqdMIZKkE9WIpjiUGmr%2BIWlo83L5hIZRCe6OAIAAA%3D%3D

Persze, ugyanennyi erővel lehetne a 0 helyett 3, -456 vagy bármilyen egész szám. Működik? Igen. Jó? Nem. A specifikáció a feladat értelmezése. Mit keres benne egy olyan szám, ami nem kapcsolódik a feladathoz?

Amellett, hogy az algoritmusban vagy a kódban tetszés szerint – vagy a már korábban megírt kódokhoz kapcsolódva – átszervezhetjük a változók globális és lokális értelmezését, a specifikációban is

próbáljunk értelmes megoldást adni. Ehhez megoldás lehet az is, ahogy a sebesség-függvényt adtuk meg: az értelmezési tartomány a $\mathbb{M}$ helyett a rekord halmazainak a szorzata lesz.

De választhatunk általánosabb megoldást is: a függvény egy méréshez megadja az időt és a végsebességet is. Ehhez azonban a vk értékét is az eredeti mérés adataiból kell megadni:

$$v_t = a \cdot t + v_k = \frac{F}{m} \cdot \frac{-v_k \cdot \mathrm{m} \pm \sqrt{(v_k \cdot m)^2 + 2 \cdot F \cdot m \cdot s}}{F} + v_k$$
$$= \frac{-v_k \cdot \mathrm{m} \pm \sqrt{(v_k \cdot m)^2 + 2 \cdot F \cdot m \cdot s}}{m} + v_k$$

Be: mérés ∈ M,
M = (kg:N x ero:N x ut:N x vk:R)
Ki: t ∈ R, vt ∈ R
Fv: fgv: M → R x R,
fgv(m) = (
(-m.vk*m.kg + sqrt(m.vk*m.vk * m.kg*m.kg + 2 * m.ero * m.kg * m.ut)) / m.ero
,
(-m.vk*m.kg + sqrt(m.vk*m.vk * m.kg*m.kg + 2 * m.ero * m.kg * m.ut)) / m.kg + m.vk
)
Ef: mérés.kg > 0 és mérés.ero > 0 és mérés.ut > 0 és mérés.vk > 0
Uf: (t, vt) = fgv(mérés)
Link

https://progtlap.elte.hu/specifikacio/?data=H4sIAAAAAAAACrWRwUrDQBRFPk2BKyVolOa1JQ4YFdCLqRuAi4M1SCnYQD2rvAfhRK9%2F2Vfk0%2FxC%2BRND2rtGtn842Zr79wHd1bUfdpZVVSzXKrFnJjuLaPZbdvdtsP3ZpPEZGD2zyTBHYK6SGcsYduFn7344WpOQ3U9V2yI%2BGepgRtuqjw6RIE6RoLRFCmWSH0uAOVBE2r8gQwnGDVjV%2BMCz-bguc7nuqSXgiH6VoZHir7Zdf%2FFV6CUNc7fnIAsDgf3dBi7472%2BwLeyq4l6Ft9U4RQYs%2FIl0f92b2c1Vd7pkvmhiltq%2BV%2Bm73rpAMie2k03Sd0XsuOTENGmNV44w4iqJDfi%2FrHmHjQC%2BPG6DfjeqZqMGKdThi32GieNSaYyFBru%2F5DiOP12%2FoHYK9jV1wCAAA%3D

Ezzel a megoldással a *t*-t és a *vt*-t egyetlen függvénnyel számoltuk ki, de a függvényen belül nem tudunk „segédváltozót” tárolni, egyik eredményt átvenni a másik kiszámításához. Ráadásul a lépésenkénti ellenőrzés sem mutatja a belső műveleteket.

A függvénybe kiszervezett képlet áttekinthetőbbé teszi a specifikációt. Ugyanakkor a rekord használata nem az olvashatóságot növeli, hanem az értelmezhetőséget: ami összetartozó, annak közös neve legyen a specifikációban is.

Algoritmus

Amikor a specifikáció paraméterei alapján megadjuk a programunk adatait, a halmazok helyett adattípusokat adunk meg. A halmaznak eleme van, az adattípusnak reprezentánsa, példánya. A halmazból kiválasztunk egy elemet, az adattípus meghatározza, hogy mit jelentenek a bitek a példány tárolásához lefoglalt memóriában. Az átalakításnál már nem fontos, hogy minek neveztük a halmazt, elég annyit megadnunk, ami a memóriakezeléshez szükséges.

```
Típus:
   M = Rekord(kg, erő, út : Egész, vk : Valós)
Változó:
   mérés : M //vagy
   mérés : Rekord(kg, erő, út : Egész, vk : Valós)
```

Az `M` egy rekord, aminek adattagjai vannak. Az `M` épp olyan adattípus, mint az `Egész` vagy a `Valós`, ezért nem lehet lokális változóként értelmezni, mindig a program adatait (változóit) megelőzően kell deklarálni (megnevezni) és definiálni, azaz megadni az adattagjait. A `mérés` egy `M` típusú adat.

Vagy: a mérés egy rekord jellegű adat, a megadott nevű és típusú adattagokkal. Ebben az esetben nincs neve a típusnak. Névtelen rekordot (`struct`-ot) C nyelven létre lehet hozni, de nem jellemző a használata. Az OOP térhódításával a rekord nemhogy névvel rendelkezik, de osztályhoz (`class`) hasonlóan használható.

A névtelen rekord kicsit hasonlít az egyes nyelvekben megtalálható tuple-hoz, a specifikáció utófeltételének bal oldalán is alkalmazott többszörös adathoz. Azonban a tuple és a rekord között lényeges

különbség, hogy a rekorddal adatstruktúrát képezünk, az adattagok (legalábbis az adattagokra mutató pointerek) a memóriában „egyvégtében” helyezkednek el, a tuple viszont egymástól független változókat fog egybe, csak az értékadás kódolását rövidíti. Ezért nem tehetünk egyenlővé egy tuple-t egy rekorddal.

A halmazrendszer (Descartes-szorzat) elemének a szintaktikája megegyezik a megfelelő rekord típusú adattal. A *mérés.ero* az algoritmusban is `mérés.ero`; adat.adattag formában írjuk.

Az algoritmusban az összetett adattípus használata többféleképpen történhet. Ha az algoritmus egy kódrészlet, akkor „egyben”, egy darab változóban lesz tárolva a bemenet mind a négy adata.

Kódrészlet algoritmusa, lokális változóval

```
Típus:
   M = Rekord(kg, ero, ut : Egész, vk : Valós)
Változó (globális):
   t, vt : Valós
```

```
Local declarations
m: M
---------------------------------------------------------------
Be: m
t := (-m.vk * m.kg + sqrt(m.vk * m.vk * m.kg * m.kg + 2 * m.ero * m.kg * m.ut) / m.ero
vt := m.ero * t / m.kg + m.vk
Ki: (t, vt)
```

https://progalap.elte.hu/stuki/?data=H4sIAAAAAAAACq1UXW%2FTMBT9K9GdhDowJU0%2FY8QDg7Wr2CRAopo09cGxr9usid05TgdU%2Be%2FIyfKBhoRQyYuPju89Puc6yRFiART8fjAahH44G4TBeDQJxtMRENjqA5qlAKryJCGQYYLcomgZpQVmQI%2Bwwtsvy7kscSnYEAT2zKCyrgnSyeLa%2Fky%2BOe1YCFRAJUsyJGB%2F7BEoKK33UJC2sNbrdLZ6lYk%2FK2X4kKPi-CAT4Nk6EQbUUGdA72JrF5fv5JAUC8nJ8Pxh%2BnAOBr4P7z4vr6RUQ0PntcDubrGBdkLa6dtJp%2F5dkmWUWU1QWCFj8boHCBVIvdWkbG%2FUZHV8nnmE9%2Bs7rvU77h5330kv7u433yssejO01VLvTFAQlRKO7O7k9995U9AsVZfu3QCBifLcxOlfig060AQpnsnxcrGakdazOjE%2BMdShz1RZt6at07uL8zVdzv7WvhjjZ16eYej1LvIM9h6IgkFmTc6s3hqXu3TuCYqlrvWGxAgIHZmI-WJe4bgpR6N0DAaP3s%2BGyrH1dtqTU5VuQVMoHmyVexJsBd2vIrrAzdudy%2B7%2FtA4MytUjokJW%2B4yB83SJZo6g99ViKOTM6msK7yXTRDfRKOgihwZW6thZHXHE45q5EskRhxHwMn534c%2F0uONHe7%2Fm3gz%2FQ557xMxTmv9WVQcygFr7gwrHZ9f8bDsUOIOBKT7kSK4heZAXkrOgUAAA%3D%3D

Függvény használata rekorddal

A függvénynek tuple kimenetet adva a specifikáció két függvényét egyben meg lehet írni.

```
Típus:
   M = Rekord(kg, ero, ut : Egész, vk : Valós)
Változó
   m : M
```

```
Function
Main
   Local declarations
   t, vk : Valós
   ------------------------------------------
   Be: m
   (t, vt) := Gyorsul(m.kg, m.ero, m.ut, m.vk
   Ki: (t, vt)
```

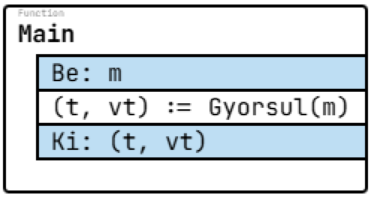
```
Function
Gyorsul(m, f, s : Egész, vk : Valós) : (Valós, Valós)
   Local declarations
   t, vt : Valós
   ------------------------------------------
   t := (-vk * m + sqrt(vk * vk * m * m + 2 * f * m * s) / f
   vt := f * t / m + vk
   Gyorsul := (t, vt)
   Return value
```

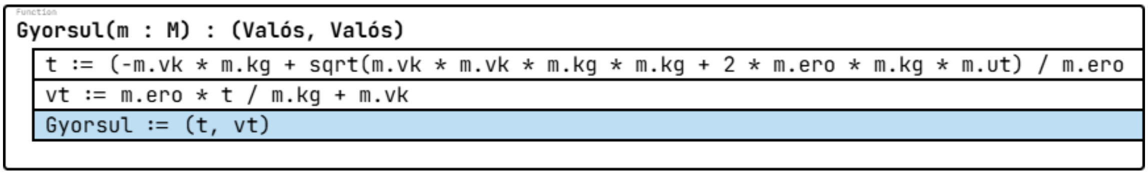
https://progalap.elte.hu/stuki/?data=H4sIAAAAAAAACq1V3U7bMBR%2BFesgTYV5EPqfTLsYAwoaSIAYQkO9cO3jtjQ%2FxXYKFOWBdr1H4MUmO00aRhGaulzEJ198vvOdnzhPMBYQgLfd9Vq%2B3%2Bi0mr7fbTfbbaAwSmaojgUEcRqGFDSGyA2KJRInAjUETxC1eydmHv6wtuMrAQpTpjA2S6fRWAiMIZAs1EjBPE4RAtB4l2LMESjw0TgUCuNjoSG4AXnQut1t7B8ChYvd27Pe-SecI%2Bhld4kXMysZlzKqSNyIbZjDC2AAFgw8GAjAk%2BEJqn2YTskUi8pHoO2Vq7mkB5XCdbBG5eNabZIfID%2FFATz8DhQHjk6FK0lh8S8JEQQAb0l2QVdIolJfA2spnTroVZciOEzmbvCdHXh1fn4vJ%2FrKQBfBCThxND78e1B%2FelBMnydTylRsLvornusNwf9mbn%2FAzHyicDtOf1w3BgML5XDceLsXUzUW5pQhf8fmXdFZUdw8DEtkUy9hFjIqYNWPUDCUzs2nb2HtMlE7DWrQ9GVIS-baNK7JIae3%2B%2Fr2VVCpElsLbI7%2BOALIRClIHQRqXcJEPFItumJ4hZZF3LDCiRlGgSkIPh8y89p%2FZDCsgVC59%2F600SkFpu0gICCjOmxmwQ2gMGXKjSASioJHn1kehRcn%2B1dDIqxRw8QiZQVZELNKmKL10yOaoqSCHbnQJFlrRM6pSNY%2FjLY4XeyUq9lWqvlrYyg6pe15ysT4HblrvjN%2B%2FKjW2%2B53keUNiwq5TWkpKX2MBrlZZ0Vsd-reMxZHJnsdqCfN3mvnKwF8aA%2BqNttdi2IkRcYdjgrLOks0eQe1i2d%2FUv8LzpaDnj%2FxdS94uecc5cV57zgl%2FUCQyl4jvl%2B%2FtbzutxvWQsRm6JdrUiW%2FQHl3S7mJgcAAA%3D%3D

Függvény rekord paraméterrel

Az algoritmusban felhasználhatjuk a korábban kiszámolt értéket, ezért nem a specifikációban megadott számítással számoljuk a sebességet. Ebben a megoldásban minden adatot globálisan deklarálunk.

```
Típus:
   M = Rekord(kg, ero, ut : Egész, vk : Valós)
Változó
   m : M
   t, vt : Valós
```





https://progalap.elte.hu/stuki/?data=H4sIAAAAAAAACq1V7W7aMBR9FetWmmDzOpdCAO9DGhvtqrWi2jS6quKH49xARhJTx2FrqzzVHmEvNjkQk36pmhh%2FfHV8fc6959rhBqIAOLBdr%2Bd19lo9r9P3GGOdNICYqSXqowB4mscxhQxjlAaDDZKqADPgN5B4h8fmOv5m45LPARQWQmNqNodmURBgCjwUcYYUzNUCgUOGlzmmEoG-CnEVxoDE9CjLgFzCOTt%2Frk7M9oBAOOz%2F29j8eAAWVf9%2Bf9bwxTIoaXsnXEjfy9aIeKcIlgwmmBigY%2FGWAQ8NQsjRNwt%2BSwyulszxuJE2g4As5n2qVp8EHFSsNHHbC8gdFrbiqHgdsXc%2FniJN1TVbpfHjw5Tyad52SA24pfW2PuqPBKHtUKVVqYflcYsVXO7ntIHuLqdSHp0OgcDZYTEfd2Iajy6vjUev8uhykS6nka2f%2BpZ0HjDN2hI2Xye5yTp6TZHc%2BJS9IdqlNw0GbHZfQKk-PUqr6TmyZ5tYKfugmuu6ohB2zd0Po6lm1tLoRzthKsWb2I4LK0sHLDlBaUJlnnnjLCveKqrtqz3vJJDJCTBIqCQmZ0Lo2aapHYG3cDqUjsoRMRpUBhKXQk%2FNh%2BsuCNr98BBa3UPeFspn6ON6lrfYt%2BQhGgBm50jgV19O7DQDg5aRJOGmMR%2F%2FmdUbJam89SP1u8vlNBXb02jrs6D9dTTChI63L5UldGXFi%2FGWMMKOzYNQxtFIbSYT7ruCg-soy7bZ6KMJlqw14XJyteBG%2Baa2G%2F5LZtm14oYZYVhV4oqCssoaEuGLUtn%2FyX%2BFx11d2pya9z3%2BKWUsuxKSlnxh60KwzCQK6zfX%2B0y1pP9jo0QsR14dUeK4i8IMyc7JgcAAA%3D%3D

Kód (C# nyelven)

A C# nyelvben lehetőségünk van adattípusok, osztályok, névterek más néven történő használatára, bár a különböző nevek nem jelentenek filozófiai különbséget. Például:

```
using System;
namespace Gyorsul
{
    using Kg = Int32;
    using Ero = int;
    using Vk = Double;
    internal class Program
    {
        struct H
        {
            public Kg kg;
            public Ero ero;
            public int ut;
            public Vk vk;
        }
    }
}
```

Az `int` helyett a speciális Ero megnevezést egyetlen helyen használjuk, a `struct` típus deklarálásakor. Ráadásul, ez tényleg csak egy másik név. Az `int` típus minden tulajdonságát és függvényét is el lehet érni az Ero típusban is. Annyira olyan az Ero, mint az `int`, hogy bármilyen művelet elvégzésére is alkalmas: össze lehet adni egy ero-t egy kg-val, az eredmény `int` lesz. Miért jó a típus átnevezése? Például, ha egy adattípusnak hosszú a neve, ezzel le lehet rövidíteni.

Ha szükség van arra, hogy ne keverjük össze az erőt a tömeggel, hogy az összeadás-kivonást csak „azonos nemű", azaz azonos mértékegységű mennyiségek között végezzük el és a szorzás-osztást a fizika törvényeinek megfelelően lehessen csak megvalósítani a programkódban, akkor ehhez – például OOP alapokon – újra kell értelmezni a feladatot: minden adatnak van mérőszáma (vagy vektor esetén mérőszámai), SI mértékegysége, megjelenített mértékegységei, amelyeknek ismert az SI-re átváltás módja, illetve definiált az SI alapmértékegységeivel való műveleti viszonya.

A programozás ilyen mélységeibe most nem megyünk bele, maradunk az egyszerűbb megvalósításnál.

A specifikációban meghatározott Descartes-szorzathoz a programozásban leginkább az adatbáziskezelők rekord fogalma illeszkedik. Ezt sok programozási nyelvben a `struct`, azaz adatstruktúra valósította meg. Azért múltidőben, mert az OOP megjelenésével a `struct` adatösszetétel egyre közelebb kerül az osztály, `class`, objektum leíróhoz. Az eredeti `struct`/`record` és `class` között az elvi különbségek:

- a `struct` csak adattagokat tartalmazhatott, míg a `class` metódusokat (eljárást, függvényt) is;

- a `struct` minden adatának a láthatósága olyan, mintha önmagában lenne az adat, a `class` adattagjai viszont csak belülről láthatók, interface-en keresztül; getter/setter függvények segítségével, tulajdonságként láthatók más adatok számára;
- a `struct` OOP előtti időkből származó nyelvi eszköz, inkább egy osztályon belül írandó, a `class` viszont inkább társa a többi osztálynak;
- a `struct`-ban az adatok a memóriában pontosan elrendezettek, a `class` – a mindenféle megvalósítási kiegészítésekhez is kapcsolódóan – sokszor használ hivatkozásokat, mutatókat.

A gyakorlatban ezen különbségek közül már csak néhány létezik és ezek is nyelvenként eltérő módon. *C# nyelvben az adatok tárolásának módja a szignifikáns különbség*. A `struct` *Value, azaz érték típusú*, ezért a függvények argumentumába az eredeti adat másolata kerül. A `class` objektumai ezzel szemben *Reference, azaz hivatkozás típusú*ak, a függvény paramétere az objektumra történő hivatkozás, az eredeti hivatkozás másolata, ami ugyanarra az objektumra mutat.

Az összes `.TryParse()` függvény egyszerű adatot értelmez, ezek mind Value típusúak, ezért kell az `out` az érték kiírásához. A `struct`-ként definiált adattípusoknál is kell az `out`, ha módosítani szeretnénk az eredeti adatot. Egy Reference adattípus esetén a hivatkozás másolatát kapja meg a függvény, ami ugyanarra az adatra hivatkozik. Ezért a hivatkozott helyen lévő adatot a függvény tudja módosítani. Ilyenek a `class`-ként definiált adattípusok.

Hagyományos `struct` megoldás

```
namespace Gyorsul
{
   internal class Program
   {
      struct M
      {
         public int kg, ero, ut;
         public double vk;
      }
      static void Main()
      {
         M mérés;
         double t, vt;
         Be(out mérés);
         (t, vt) = Gyorsul(mérés);
         Ki(t, vt);
      }
      private static void Be(out M m)
      {
         string[] adatok = Console.ReadLine().Split(' ');
         m.kg = int.Parse(adatok[0]);
         m.ero = int.Parse(adatok[1]);
         m.ut = int.Parse(adatok[2]);
         m.vk = double.Parse(adatok[3]);
      }
      private static (double, double) Gyorsul(M m)
      {
         double t=(-m.vk*m.kg+Math.Sqrt(m.vk*m.vk*m.kg*m.kg
                                        +2.0*m.ero*m.kg*m.ut))/m.ero;
         double vt = m.ero * t / m.kg + m.vk;
         return (t, vt);
      }
      private static void Ki(double t, double vt)
      {
         Console.WriteLine(t.ToString("F1") + " " + vt.ToString("F1"));
      }
   }
}
```

*Konstruktoros **struct** megoldás*

Ha a megoldásunkban `struct`-ot használunk, akkor érdemes lehet a `class` használatából „átvenni” a konstruktor megírását. Ez annyit jelent, hogy a `struct` definiálásakor megadjuk, hogy milyen adatokból, hogyan kapnak kezdőértéket az adattagok. Ha nem írunk konstruktort, akkor minden `struct` típusú adat adattagjai először alapértelmezett értékkel vagy definiálatlanul jönnek létre és ezeket egyenként az adott helyen kell módosítani. Majdnem mindegy, kivéve, hogy ahol a `struct`-ot definiáljuk, ott fel vannak sorolva az adatok, átlátható a kód, míg az adat létrehozásánál már oda kell figyelni, hogy milyen adatokat kell megadni. Elsőre szinte biztosan jó lesz… de például a bemenet adatainak változása miatt, ha módosítjuk a `struct` definícióját, már jobban oda kell figyelni, hogy a program minden olyan helyén, ahol ilyen adattípust hozunk létre, módosítsuk az értékadást is.

```
namespace Gyorsul
{
    internal class Program
    {
        struct M
        {
            public int kg, ero, ut;
            public double vk;
            public M(string[] adatok)
            {
                kg = int.Parse(adatok[0]);
                ero = int.Parse(adatok[1]);
                ut = int.Parse(adatok[2]);
                vk = double.Parse(adatok[3]);
            }
        }
        static void Main()
        {
            M mérés;
            double t, vt;
            mérés = Be();
            (t, vt) = Gyorsul(mérés);
            Ki(t, vt);
        }
        private static M Be()
        {
            return new M(Console.ReadLine().Split(' '));
        }
…
    }
}
```

*A **struct** jellegű **class**, elvi hibás megoldás*

Ha a megoldásunkban `class`-t használunk, akkor egyszerűsíthetjük úgy, hogy nem írjuk meg a gettert és a settert, hanem az eredetileg privát belső adatot publikussá tesszük. Lehet, de szakmailag hibás, mert ez az OOP minden elvárását figyelembe vevő programozási nyelv megerőszakolása.

```
namespace Gyorsul
{
    class M
    {
        public int kg, ero, ut;
        public double vk;
        public M(int _kg = 0, int _ero = 0, int _ut = 0, double _vk = 0.0)
        {
            kg = _kg;
            ero = _ero;
            ut = _ut;
            vk = _vk;
        }
    }
```

```
    internal class Program
    {
        static void Main()
        {
            M mérés;
            double t, vt;
            mérés = Be();
            (t, vt) = Gyorsul(mérés);
            Ki(t, vt);
        }
        private static M Be()
        {
            string[] adatok = Console.ReadLine().Split(' ');
            return new M(int.Parse(adatok[0]), int.Parse(adatok[1]),
                         int.Parse(adatok[2]), double.Parse(adatok[3]));
        }
…
    }
}
```

*Megoldás **class**-szal*

Ha már `class`, akkor legyen a C# nyelvi szabványnak megfelelően, nagybetűvel kezdődő Property is az adattaghoz: „`public` típus Adatnev {`get`; `set`;}”. A többivel, például a privát adatnev adattal nekünk már nem kell foglalkozni, a fordító minden mást elintéz.

```
namespace Gyorsul
{
    class M
    {
        public int Kg { get; set; }
        public int Ero { get; set; }
        public int Ut { get; set; }
        public double Vk { get; set; }
        public M(string adatsor)
        {
            string[] adatok = adatsor.Split(' ');
            Kg = int.Parse(adatok[0]);
            Ero = int.Parse(adatok[1]);
            Ut = int.Parse(adatok[2]);
            Vk = double.Parse(adatok[3]);
        }
    }
    internal class Program
    {
        static void Main()
        {
            #region bemenet és kimenet deklarációk
            M mérés;
            double t, vt;
            #endregion
            mérés = Be();
            (t, vt) = Gyorsul(mérés);
            Ki(t, vt);
        }
        private static M Be()
        {
            return new M(Console.ReadLine());
        }
        private static (double, double) Gyorsul(M m)
        {
            double t=(-m.Vk*m.Kg+Math.Sqrt(m.Vk*m.Vk*m.Kg*m.Kg+
                                           2.0*m.Ero*m.Kg*m.Ut))/m.Ero;
            double vt = m.Ero * t / m.Kg + m.Vk;
            return (t, vt);
        }
…
```

```
    }
}
```

Konstruktorok

Az egyes megoldásokban különböző konstruktorokat használtunk. Egy `class` vagy `struct` definiálásakor több konstruktor is létrehozható, csak arra kell ügyelni, hogy egyértelműen felismerhető legyen, melyiket kell alkalmazni. Ez a konstruktor paraméterezésétől függ. A fenti megoldások között van egy olyan, ahol egyetlen `string`-et adunk meg, van egy `string` tömbös megoldás és egy olyan konstruktor is szerepel, amelyben az adatoknak megfelelő számú és típusú paramétere van a konstruktornak.

```
public H(int _kg = 0, int _ero = 0, int _ut = 0, double _vk = 0.0)
```

Ez az egy konstruktor valójában öt különböző konstruktornak felel meg, mivel minden paraméterhez megadtunk egy kezdőértéket. Ha a jobb oldalról sorban elhagyjuk a paramétereket, a megadott értékek lesznek érvényesek.

Az más kérdés – és ezt ellenőrizni is kellene –, hogy a feladat előfeltétele miatt egyik adat értéke sem lehet nulla. Ezt megoldhatjuk a konstruktoron belül is vagy az osztálynak lehet a helyességet ellenőrző függvénye is.

Record – a nem használható legjobb

A C# nyelv fejlesztőinek célja egy általánosan (mindenre) használható, szakmai szempontokat a C-hez hasonlóan messzemenőkig figyelembe vevő, szakembert segítő nyelv készítése. Fontos, hogy könnyen, gyorsan lehessen jó kódot írni. A nyelv fejlődése során a programozók igényeit veszik figyelembe. Ezt láthatjuk a tuple megjelenésénél is, a C# 7-es verziójában, illetve a C# 9-es verziótól kezdődően a `record` használatának lehetőségénél.

Nagyon röviden: a `record` pont az akar lenni, amit rekord-nak hívunk az algoritmusban. Nincs külön deklaráció és konstruktor; a konstruktorral deklarálható; `class`, tehát referencia típusú, de értékként adódik át, mint a `struct`. A memóriakezelést, elérés módját, az adattagok módosíthatóságát úgy adták meg, hogy kényelmesen lehessen használni, sokkal rövidebben, érthetőbben lehessen megírni a kódot. A különböző igényeknek megfelelően, a C# 10-ben a `record struct` és `record class` még jobban illeszkedik különböző programozói igényekhez.

```
namespace Gyorsul
{
    internal class Program
    {
        record Rec(int Kg, int Ero, int Ut, double Vk);
        static void Main(string[] args)
        {
            Rec mérés;
            double t, vt;
            mérés = Be();
            (t, vt) = Gyorsul(mérés);
            Ki(t, vt);
        }
        private static Rec Be()
        {
            string[] adatok = Console.ReadLine().Split(' ');
            Rec m = new(int.Parse(adatok[0]), int.Parse(adatok[1]),
                        int.Parse(adatok[2]), double.Parse(adatok[3]));
            return m;
        }
        private static (double, double) Gyorsul(Rec m)
        {
            double t=(-m.Vk*m.Kg+Math.Sqrt(m.Vk*m.Vk*m.Kg*m.Kg+
                                           2.0*m.Ero*m.Kg*m.Ut))/m.Ero;
            double vt = m.Ero * t / m.Kg + m.Vk;
            return (t, vt);
        }
```

```
        private static void Ki(double t, double vt)
        {
            Console.WriteLine(t.ToString("F1") + " " + vt.ToString("F1"));
        }
…
    }
}
```

Bár a `record`-ot a Copilot is javasolja – és amúgy is, ez egyezik meg leginkább az algoritmusban szereplő rekorddal –, használata ebben a programozás tantárgyban nem javasolt, mondhatjuk, hogy saját érdekünkben tiltott. Az összes értékelt feladatot a biro.inf.elte.hu-n lehet (kell) tesztelni, a mester.inf.elte.hu-n lehet gyakorolni, ahol Linux alapon a C# kódot Mono fordítja futtatható állományra. A Mono a C# 7-es verziójáig követte a fejlődést, így a `record` számára ismeretlen szleng.

100 szónak is egy a vége

Ha az eddigiekből néhány dolog érthetetlen, akkor a programozás tárgy tanulása során

- lehet a `struct` és `class` egyikéhez ragaszkodni, elsősorban a `struct` javasolt.
- mindig választható olyan megoldás, hogy a függvények paraméterei csak olvashatók legyenek; amit módosítani szeretnénk, az vagy legyen a `Program` osztályra nézve globális adat (`static`-os), vagy legyen a függvény visszatérési értéke.

Önálló feladat

> A megoldás egyes részeire többféle megoldás vagy megoldás részlet szerepelt. Állítsunk össze legalább egy (de jobb, ha több) teljes megoldást és írjuk meg a fejlesztői dokumentációt.