NAGY KRISZTIÁN

Programozás

a Modellező informatikus szakirányon

1. KÖTET

4. kiadás

Tartalomjegyzék

0.1 Szimbólum táblázat

A jegyzet olvasása közben az alábbi szimbólumokkal és jelölésekkel találkozhatsz. Ezen jelöléseket a jelentésükkel az alábbi táblázat foglalja össze:

Jelölés	Jelentés
$\mathbb{N}$	természetes számok halmaza
$\mathbb{N}_0$	természetes számok halmaza a 0-ával együtt
$\mathbb{Z}$	egész számok halmaza
$\mathbb{Z}^+$	a pozitív egészek halmaza
$\mathbb{Q}$	racionális számok halmaza
$\mathbb{R}$	valós számok halmaza
$\mathbb{R}_0^+$	a pozitív valós számok halmaza a 0-ával együtt
$\mathbb{C}$	komplex számok halmaza
$\mathbb{K}$	karakter halmaz
$\mathbb{K}^*$	karakterekből álló sorozat
$\mathbb{L}$	logikai halmaz $\mathbb{L} := \{igaz, hamis\}$
$\mathbb{Z}^n$	n darab egész számot tartalmazó tömb
$\mathbb{R}^{m \times n}$	$m \times n$ dimenziós valós számokat tartalmazó mátrix
$\forall$	univerzális kvantor (minden)
$\exists$	egzisztenciális kvantor (létezik)
$\wedge$	logikai és
$\vee$	logikai vagy
$\Longrightarrow$	implikáció (következtetés)
$\oplus$	összefűzés művelete
$\sum$	szumma (összegzés nagyoperátora)
$\prod$	produktum (szorzás nagyoperátora)
$\neg$	negálás (tagadás, ellentét vevés)
$\sim$	megfeleltetés
$\boldsymbol{A}$	állapottér
$\boldsymbol{Ef}$	előfeltétel
$\boldsymbol{Uf}$	utófeltétel
$\uparrow$	logikai igaz
$\downarrow$	logikai hamis
$\times$	Descart-szorzat

0.2 Bevezető

Sok hallgató, aki a Programtervező informatikus szakra felvételizik nem rendelkezik megfelelő előképzettséggel programozásból, ezért programozási alapismeretektől kezdve szinte szenvedve próbálja elsajátítani a programozáshoz kapcsolódó ismereteket és kezdetben a C++ programozási nyelv használatát. Többek között én is ilyen hallgató voltam a kezdetekben és hosszú évek során értem el azt a szintet, ahol most tartok. Ez a szint még messze nem a legjobb, de a kurzusokból idáig szerzett jegyeim és a pozitív visszajelzések alapján elég arra, hogy bizalommal használhatjátok ezt a kis jegyzetet, amit nektek készítettem azért, hogy a sikerhez vezető utat könnyebbé tegyem számotokra.

Először sokan azt gondolják, hogy a programozás egy megtanulható „dolog", melyet az itteni képzés során maximálisan elsajátítanak, szeretném eloszlatni ezeket a gondolatokat.
Az egyetem a programozáshoz egy erős szemléletet nyújt, melyet megfelelő programozási nyelveken keresztül próbál átadni a hallgatóknak. A programozást nem lehet csak úgy megtanulni, viszont több évi gyakorlás és tapasztalat szerzés után az egyetemen tanultak segítségével elsajátítható olyan szinten, amennyire szükségetek lesz rá.

Ezt a könyvet midenek mellett egy iránymutatónak is szánnám, hogy lásd, hogy kapcsolódnak egymáshoz az adott kurzusok. Remélem hasznosnak találod a jegyzetem.

Végezetül pedig két idézettel kezdenénk bele abba a nagy munkába, ami rád vár.

„Mondd el és elfelejtem,
mutasd meg és megjegyzem;
Engedd, hogy csináljam és megértem"

Kung-Fu-Ce

„A sikerhez és tudáshoz vezető út senki előtt sincs zárva,
akiben van bátorság és elszántság, hogy változzék,
nyitottság, hogy mások tapasztalataiból tanuljon és állhatatosság,
hogy gyakorlással elsajátítsa a sikeres cselekvés technikáját."

1. Problémamegoldástól a kódolásig

1.1 A problémamegoldás lépései egy minta segítségével

Az alábbi példánk a házépítés legyen. Az alábbi kérdéseket minden probléma esetén érdemes feltenni magunknak. ***Mi az, ami látszik? Mi az, ami ténylegesen mögötte van?***

Egy ház felépítése során az **első** dolog, amit el kell végeznünk az **igényfelmérés**.
Az igényfelmérést szempontok alapján végezzük el. Például: Mekkora a család, akiknek a házat szeretnénk építeni. Mik az elképzeléseik? Mennyi pénzt fektetnének bele összesen a házba?

Ha tisztában vagyunk a fentebb említettekkel, **második** lépésként meg kell **terveznünk** a házat. Alaprajzot kell készíteni. Meg kell állapítani az anyagigényeket. Mennyi mérnökre van szükségünk a ház elkészítéséhez?

Ezek után **harmadik** lépésként, meg kell **szervezni** a munkállatokat. Ütemtervet kell készítenünk. Vállalkozókat szereznünk.

Negyedik lépésként elkezdődhet az **építkezés**. Anyagok beszerzése. Kivitelezés.

Ötödik lépésként a **használatba vétel** következik. Szép-e a ház amit építettünk. Ki kell próbálni, hogy működnek-e a beépített kényelmi funkciói.

Végül **hatodik** lépésként **beköltözhet** a család a házunkba. Ekkor lehet, hogy egyes funkciókat nem így képzelték el, így át kell alakítanunk a lakást. Előfordulhat, hogy újabb hibák keletkeznek benne idővel. Ezt is orvosolnunk kell.

1.2 A programkészítés folyamata

1. **Specifikálás** - Miből mit szeretnénk készíteni? → *specifikáció*
2. **Tervezés** - Mivel, hogyan szeretnénk megvalósítani? → *adat- + algoritmus-leírás*
3. **Kódolás** - A gép, hogyan tudja megvalósítani? → *kód (reprezentáció + implementáció)*
4. **Tesztelés** - Hibás-e a megvalósításunk? → *hibalista (diagnózis)*
5. **Hibakeresés** - Hol hibáztunk? → *hibahely, hibaok*
6. **Hibajavítás** - Hogyan lenne jó? → *helyes program*
7. **Minőségvizsgálat, hatékonyság** - Hogyan lenne jobb? → *jó program*
8. **Dokumentálás** - Hogyan működik, használható-e? → *használható program*
9. **Használat, karbantartás** - Még mindig jó-e a program? → *évelő (időtálló) program*

1.3 Nyelvi szintek

A nyelvek közelítése:

Élőnyelv = Magyar → Specifikáció → Algoritmusleíró → Programozási ← Gépi

1.4 Az algoritmus fogalma

1.4.1 Vegyünk egy egyszerű példát: Hogyan használjuk az ital autómatát?

1. Válaszd ki az italt!
2. Dobj be egy 100 forintost
3. Nyomd meg a megfelelő gombot!
4. Várj, amíg folyik az ital
5. Vedd ki az italt!
6. Idd meg!

Az alapalgoritmus elemei:

1. Egymásutáni végrehajtás – Lásd ital autómata használatának lépései
2. Nem-determinisztikusság – Lásd bármilyen italt is választasz, akkor is működnie kell az autómatának
3. Párhuzamosság

1.4.2 Bővítsük ki az italautómatánk használatát:

1. Válaszd ki az italt!
2. Dobj be egy 100 forintost
3. Nyomd meg a megfelelő gombot!
4. **Ismételd: Nézd a poharat,amíg folyik az ital!**
5. Vedd ki az italt!
6. Idd meg!

Új algoritmus elem: **Ismétlés feltételtől függően**!

1.4.3 Ismételten bővítsük ki az italautómatánk használatát:

1. Válaszd ki az italt!
2. **Ha van 100 forintosod, dobj be egy 100 forintost, különben dobj be 5 darab 20ast!**
3. Nyomd meg a megfelelő gombot!
4. Ismételd: Nézd a poharat,amíg folyik az ital!
5. Vedd ki az italt!
6. Idd meg!

Új algoritmus elem: **Választás** két tevékenység közül, esetleg nem-determinisztikus választás.

1.4.4 Változtassuk meg a fentebbi használatot:

1. Válaszd ki az italt!
2. **Ismételd 5-ször: Dobj be 20 forintot!**
3. Nyomd meg a megfelelő gombot!
4. Ismételd: Nézd a poharat,amíg folyik az ital!
5. Vedd ki az italt!
6. Idd meg!

Új algoritmus elem: **Ismétlés adott darabszámszor**!

A fentebbi példákból együttesen pedig megmutattuk az algoritmusok összeállítási módjait!

1.4.5 **Algoritmusok alkotóelemei:**

1. Szekvencia (Egymás utáni végrehajtás)
2. Elágazás (Választás 2 vagy több feltétel közül)
3. Ciklus (Ismétlés adott darabszámtól, vagy feltételtől függően)

1.5 Specifikáció

1.5.1 A specifikáció lépései

1. **Bemenő adatok** (azonosító, értékhalmaz, mértékegység)
2. **Ismeretek a bemenetről** (előfeltétel)
3. **Eredmények** (azonosító, értékhalmaz, …)
4. **Az eredmény kiszámítási szabálya** (utófeltétel)
5. A **megoldással szembeni követelmények**
6. **Korlátozó tényezők**
7. A **használt fogalmak definíciói**

1.5.2 A specifikáció tulajdonságai

1. Egyértelmű, pontos, teljes
2. Rövid, tömör, formalizált
3. Szemléletes, érthető

1.5.3 Specifikációs eszközök

1. Szöveges leírás
2. Matematikai megadás

1.5.4 Minta feladat

Adott 3 szám, lehet-e egy derékszögű háromszögnek a három oldala?

Specifikáció:

Bemenet: x, y, z: $\mathbb{R}$ (Valós)
Kimenet: $lehet$: $\mathbb{L}$ (Logikai)
Előfeltétel: $x > 0$ *és* $y > 0$ *és* $z > 0$ $(x > 0 \ \wedge \ y > 0 \wedge z > 0)$ (A háromszög oldalai 0-nál nagyobbak)

Utófeltétel: $lehet = \ (x^2 + y^2 = z^2)$

Az utófeltétel alapján a három szám sorrendjét rögzítettük - z az átfogó!

1.6 Algoritmusleíró nyelvek

1. Szöveges leírás
   - Mondatokkal leírás
   - Mondatszerű elemek – **pszeudokódok**
2. **Rajzos leírás**
   - Folyamatábra
   - **Struktogram**

1.7 Struktogramok és pszeudokódok

1.7.1 Szekvencia

Szekvencia struktogramja:

UTASÍTÁS1
UTASÍTÁS2

Szekvencia pszeudokódja:

UTASÍTÁS1
UTASÍTÁS2

1.7.2 Kétirányú elágazás

Kétirányú elágazás struktogramja:

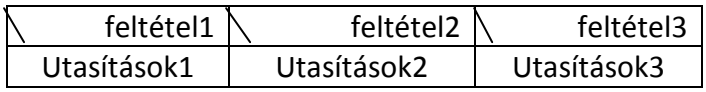
feltétel	
igaz-ág utasításai	hamis-ág utasításai

Kétirányú elágazás pszeudokódja:

Ha feltétel **akkor**
    Igaz-ág utasításai
különben
    Hamis-ág utasításai
Elágazás vége

1.7.3 Többirányú elágazás

Többirányú elágazás struktogramja:

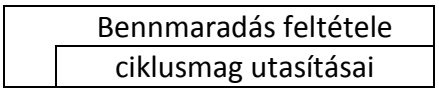


Többirányú elágazás pszeudokódja:

Elágazás
    Feltétel1 **esetén** Utasítások1
    Feltétel2 **esetén** Utasítások2
…
egyéb esetekben Utasítások
Elágazás vége

1.7.4 Ciklusok

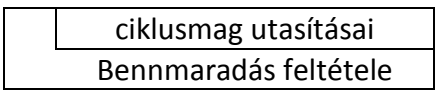
Előtesztelő ciklus struktogramja:



Előtesztelő ciklus pszeudokódja:

Ciklus amíg Feltétel
    ciklusmag utasításai
Ciklus vége

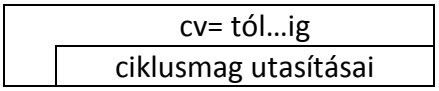
Hátultesztelő ciklus struktogramja:



Hátultesztelő ciklus pszeudokódja:

Ciklus
    ciklusmag utasításai
amíg Feltétel
Ciklus vége

Számlálós ciklus struktogramja:



Számlálós ciklus pszeudokódja:

Ciklus cv=tól…ig
    ciklusmag utasításai
Ciklus vége

1.8 Struktogram szerkesztés

1. Táblázatkezelővel, vagy szövegszerkesztővel
2. Célprogramokkal : pl. NSD
3. Weboldal: http://stukimania.hu/

1.9 Kódoláshoz használt eszköz

A félévek során a kódolásra a Code-Blocks nevű programot használjuk a leggyakrabban.
A kódoláshoz használt programozási nyelv pedig a C++ lesz.

A használt fordító program pedig a GNU GCC Compiler

Letölthető ingyenes program: http://www.codeblocks.org/

2. Alapfogalmak

2.1 Konstans és változó

Konstans: Az az *adat*, amely a műveletvégzés során *nem változtat(hat)ja meg értékét*, mindvégig ugyanabban az „állapotban” marad.

Változó: Az ilyen *adat*féleségnek lényegéhez tartozik a „változékonyság”, más szóval: vonatkozhatnak rá olyan műveletek is, amelyek új értékkel látják el. Tudományosabban fogalmazva nem egyelemű az *állapothalmaz*a.

2.2 Értékadás

Az az *utasítás*, ami révén a pillanatnyi állapotból egy másikba (a meghatározottba) kerül át a változó. (Nyilvánvaló, hogy konstans adatra nem vonatkozhat értékadás, az egy, kezdőértéket meghatározón kívül.)

2.3 Típus

Olyan „megállapodás” (absztrakt kategória), amely adatok egy lehetséges körét jelöli ki az által, hogy rögzíti azok *állapothalmaz*át és az elvégezhető *művelet*ek arzenálját.

2.4 Azonosító és kezdőérték

Azonosító: Az a *jelsorozat*, *amellyel hivatkozhatunk a tartalmára*, *amely által módosíthatjuk tartalmát*.

Kezdőérték: A *születéskor hozzárendelt érték*.
Konstansoknál nyilvánvaló; Változóknál deklarációban kap, adható, vagy futáskor szerez értéket magának.

2.5 Hozzáférési jog

Adatokat módosítani, illetve értéküket lekérdezni, használni lehet; egy adat hozzáférés szempontjából háromféle lehet:

1. lekérdezhető és módosítható;
2. lekérdezhető és nem módosítható;
3. nem lekérdezhető, de módosítható

2.6 Hatáskör és élettartam

Hatáskör: A *programszöveg azon tartománya, amelyben az adathoz hozzáférés lehetséges*.

Élettartam: A *futási időnek az az intervalluma, amelyben az adat azonosítója végig ugyanazt az objektumot jelöli*.

2.7 Értéktípus

Az adatoknak az a tulajdonsága, hogy *értékei mely* **halmaz***ból* származnak és **tevékenységek***nek* (függvények, operátorok, utasítások) *mely „készlete*, amely *létrehozza*, *felépíti*, *lerombolja* és *részekre bontja”*, alkalmazható rá.

3. Hello World!

```
#include <iostream>

using namespace std;

int main()
{
    cout << " Hello World!" << endl;

    return 0;
}
```

Az első programocskánk amit láthatunk a Code-Blocksban, miután megnyitottuk a main.cpp fileunkat.

Nézzük meg a kódot lépésről lépésre:

```
#include <iostream>
```

A fordítónak megmondjuk, hogy vegye figyelembe, hogy az input-output stream előre megírt könyvtárát is szeretnék használni kódolásunk során. Ez a könyvtár bárhol lehet a számítógépünkön, a fordító meg fogja keresni.

```
using namespace std;
```

Használjuk a standard névteret, erre azért van szükségünk, mert a konzolra kiíró utasításunkat és a sor vége jelölés ebben a névtérben található. Amennyiben nem írjuk ki, úgy az alábbi módon működhet csak a programunk:

```
#include <iostream>

int main()
{
    std::cout << " Hello World!" << std::endl;

    return 0;
}
```

```
int main(){...} – A főprogramunk
```

Amennyiben szöveget szeretnénk kiírni úgy a kiírandó szöveget " " közé tesszük.

Megjegyzés: Amennyiben kommenteket szeretnénk írni, amit a fordító nem vesz figyelembe úgy két lehetőségünk van:

1. // szöveg Ezzel az adott sort nem veszi figyelembe a fordító

/* szöveg */ Ezzel pedig azt, a részt nem veszi figyelembe, ami a /**/ között van

4. Adattípusok a C++ nyelvben

Két féle adattípust tudunk megkülönböztetni elsősorban a C++ nyelvben. Vannak beépített típusok és összetett típusok. A beépített típusokhoz nem kell a fordító számára egyéb könyvtárat vagy fejlécfájlokat linkelnük, míg összetett típusok esetén szükségünk van rá, hogy működjön.

C++ nyelvben az alábbi módon hozhatunk létre adott típusú változókat / konstansokat:
Változó:
[adattípus] [azonosító];

[adattípus] [azonosító] = [kezdőérték];

Konstans:

const [adattípus] [azonosító] = [kezdőérték];

4.1 Beépített típusok

4.1.1 Egész jellegű beépített típusok

[típus]	Jelentés
int	Általános egész típus (pozitív,negatív egészek és 0)
short	Előjeles egész típus néha kisebb értéktartománnyal
long	Előjeles egész típus néha nagyobb értéktartománnyal
unsigned int	Előjel nélküli egész (csak nemnegatív egész számok)
unsigned short	lásd unsigned int csak néha kisebb értéktartománnyal
unsigned long	lásd unsigned int csak néha nagyobb értéktartománnyal
char	Egy byte-os egész típus karakterkódok tárolására szokták használni
signed char	lásd char csak előjeles
unsigned char	lásd char csak előjel nélküli
bool	logikai típus (true vagy false), csak 0 vagy 1 lehet az értéke

Példa változóra: int v = 0; long l;
Példa konstansra: const int konstans = 42;

4.1.2 Lebegőpontos beépített típusok

[típus]	Jelentés
double	Általános valós szám típus
float	Valós szám típus kisebb értéktartománnyal/ pontossággal
long double	Valós szám típus nagyobb értéktartománnyal/ pontossággal

4.2 Összetett típus példa

[típus]	Jelentés
string	szöveg típus

Szükséges include: #include <string>

String típusú változó létrehozása:

Ha van using namespace std; , akkor string s;
Ha nincs, akkor: std::string s;

Kezdőérték adás:

std::string s = ”valami”;

Üres string: std::string s = ” ”;

5. Egyszerű algoritmusok a gyakorlatban

5.1 Szekvencia

```
#include <iostream>

using namespace std;

int main()
{
    int a = 80;
    int b = 20;
    int c;

    c = a;
    a = b;

    return 0;
}
```

A fentebbi kis programban létre hoztunk három int típusú változót a,b,c-t, melyből a-nak és b-nek kezdőértéket adtunk.

Ezek után a program annyit csinál, hogy c változónak értékéül adja az a változó értékét. Azaz c = a = 80, majd a változó értékül kapja b értékét a = b = 20.

Végeredményként tehát: a = 20, b = 20, c = 80.

5.2 Kétirányú elágazás

```
#include <iostream>

using namespace std;

int main()
{
    int a;

    cout << ”Adjon meg egy egész szamot: ” << endl;
    cin >> a;

    if (a<0)
    {
       cout << ”A beírt szám negatív! ” << endl;
    }
    else
    {
       cout << ”A beírt szám 0 vagy pozitív! ” << endl;
    }

    return 0;
}
```

A fentebbi programban létrehoztunk egy int típusú változót. Kiírjuk a konzolra, hogy Adjon meg egy egész számot. a program vár, hogy a felhasználó beírjon egy számot, majd az elágazás eldönti, hogy egy szám negatív-e vagy pozitív és ezt kiírja a felhasználónak.

5.3 Többirányú elágazás

```
#include <iostream>

using namespace std;

int main()
{
    int a;

    cout << ”Adjon meg egy egész szamot: ” << endl;
    cin >> a;

    if (a<0)
    {
       cout << ”A beírt szám negatív! ” << endl;
    }
    else if (a == 0)
    {
       cout << ”A beírt szám 0! ” << endl;
    }
    else
    {
       cout << ”A beírt szám pozitív! ” << endl;
    }

    return 0;
}
```

Az előző példánk azzal a különbséggel, hogy a 0-át külön kezeljük.

5.4 Ciklusok

5.4.1 Elöltesztelő ciklus

```
#include <iostream>

using namespace std;

int main()
{
    int a;

    while (a<=0)
    {
       cout << ”Adjon meg egy 0-nál nagyobb egész számot ” << endl;
       cin >> a;

    }

    return 0;
}
```

A fentebbi kis program amíg nem kap 0-nál nagyobb egész számot, addig piszkálja a felhasználót, hogy adjon meg neki helyes számot.

5.4.2 Hátultesztelő ciklus

```
#include <iostream>

using namespace std;

int main()
{
    int a;

    do
    {
       cout << ”Adjon meg egy 0-nál nagyobb egész számot ” << endl;
       cin >> a;

    } while (a<0);

    return 0;
}
```

A fentebbi kis program addig piszkálja a felhasználót, hogy adjon meg neki helyes számot, amíg nem kap 0-nál nagyobb egész számot,.

5.4.3 Számlálós ciklus

```
#include <iostream>

using namespace std;

int main()
{
    int s = 0;
    int a;
    cout << ”Kiszámolom a beírt 5 szám összegét: ” << endl;
    for (int i = 0; i<5; ++i)
    {
         cout << ”Adjon meg egy egész számot” << endl;
         cin >> a;
         s = s + a;
    }

    cout << ”A beírt 5 szám összege: ” << s << endl;

    return 0;
}
```

A fentebbi program 5x kéri a felhasználót, hogy írjon be egy számot, majd folyamatosan hozzáadja az s változóban eltárolt értékekhez ezeket a számokat. A végén pedig közli a felhasználóval ezen számok összegét.

6. Egyszerű tömbök

6.1 Egydimenziós tömbök

A tömb azonos típusú elemek sorozata, melyek egymás után következő memória blokkokba vannak helyezve, és minden elemére hivatkozni lehet az indexével.

Ez azt jelenti, hogy például eltárolhatunk 5 int tipusu értéket egy tömbben anélkül, hogy 5 különböző változót kéne hozzá deklarálnunk, mindegyiket külön azonositóval. Tehát a tömböket használva eltárolhatunk akár 5 különböző értéket is (a megadott tipusbol), egy külön azonositóval.

Például nevezzük Jancsi-nak a tömbünket, amiben 5 darab int típusú értéket szeretnénk eltárolni. Fontos arra is figyelnünk, hogy C++ban a tömbök számozása 0-tól kezdődik, ezért így néz ki a tömbünk elképzelve a memóriában:

	0	1	2	3	4
Jancsi	int	int	int	int	int

Itt minden blokk a tömb egy elemét tartalmazza, melyek ebben az esetben int típusúak.

Mint egy szabályos változót a tömböt is deklarálni kell, ahhoz hogy használhassunk. A tömb tipikus deklarációja ilyen:

[típus] [azonosító] [tömb mérete];

A mi esetünkben például az int jancsi[5]; egy jó deklaráció.

A tömb elemeinek száma egy konstans érték, ez nem tud változni a program folyamán.

Mikor deklarálunk egy szabályos tömböt, annak elemeit is meg tudjuk adni az alábbi módon: A tömbnek egy értékadása:

int jancsi[5] = {10, 20, 30, 40, 50};

Megjegyzés: Ha értéket adunk egy tömbnek, akkor a C++ megengedi, hogy a [ ] –ket üresen hagyjuk. Ekkor a fordító feltételezi, hogy a tömb mérete megegyezik az elemek számával.

A program bármely pontján, ahol a tömb elérhető hivatkozhatunk a tömb bármely elemének értékére a következő formulával: [azonosító][index];

Például, ha jancsi[0] –át írunk a programba az a 10-est fogja visszaadni.

6.2 Többdimenziós tömbök

Úgy lehetne leirni őket, mint „tömbök tömbje(i)”. Például egy kétdimenziós tömböt táblázatként lehet legegyszerűbben elképzelni. Más néven a kétdimenziós tömböket mátrixoknak is nevezzük.

Nevezzük a tömbünket Juliska-nak.
Például a memóriában:

		0	1	2	3
	0	int	int	int	int
Juliska	1	int	int	int	int
	2	int	int	int	int

Ekkor Juliska egy 3×4 dimenziós mátrix, vagyis egy olyan mátrix, amely 3 sorból és 4 oszlopból áll.

El képzelhetnénk úgy is hogy egy 4 elemű tömb, minden eleme egy 3 elemű inteket tartalmazó tömb.

A fentebbi tömb deklarációja így néz ki:
int juliska[3][4];

Általánosan:
[típus][azonosító][sorok száma][oszlopok száma];

Az elemeire az egydimenziós tömbökhöz hasonlóan tudunk hivatkozni.
Például: juliska[1][3]; ez a 2. sor 4. elemét fogja visszaadni

		0	1	2	3
	0	int	int	int	int
Juliska	1	int	int	int	int
	2	int	int	int	int

A többdimenziós tömbök nincsenek limitálva két dimenzióra. Annyi indexet tartalmhazhatnak, amennyit csak akarunk. De vigyázat! A memória felhasználás jelentősen megnő minden dimenzióval. Például:

char century [100][365][24][60][60];

Ez egy char típusú tömböt deklarál minden egyes másodpercre egy évszázadban, ami több mint 3 milliárd karakter. Tehát ehhez a deklarációhoz több mint 3 gigabyte memória szükséges.

7. Elemi programozási tételek egyszerű tömbökre

7.1 Sorozatszámítás / Összegzés

Specifikáció:

Bemenet: $N: Egész \quad X: Tömb[1..N: Típus_1]$
Kimenet: $S: Típus_1$
Előfeltétel: $N \geq 0$
Utófeltétel: $S = F(X[1..N])$

Ahol $F: \sum$ - N tagú összeg vagy $\prod$ - N tagú szorzat vagy $\cup$ N „halmaz” uniója vagy $\&$ - N szöveg konkatenációja.

Algoritmus:

$S \coloneqq F_0$
$i = 1..N$
$S \coloneqq f(S, X[i])$

Algoritmus N tagú öszegre:

$S \coloneqq 0$
$i = 1..N$
$S \coloneqq S + X[i]$

7.2 Eldöntés

Specifikáció:

Bemenet: $N: Egész \quad X: Tömb[1..N: Típus_1]$
Kimenet: $Van: Logikai$
Előfeltétel: $N \geq 0$
Utófeltétel: $Van = \exists i(1 \leq i \leq N): T(X[i])$

T egy tulajdonság vagy feltétel, aminek teljesülnie kell.

Algoritmus:

$i \coloneqq 1$
$i \leq N \wedge \neg T(X[i])$
$i \coloneqq i + 1$
$Van \coloneqq i \leq N$

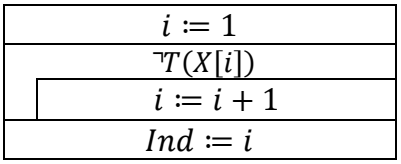
7.3 Kiválasztás

Specifikáció:

Bemenet: $N: Egész \quad X: Tömb[1..N: Típus_1]$
Kimenet: $Ind: Egész$
Előfeltétel: $N > 0 \wedge \exists i(1 \leq i \leq N): T(X[i])$
Utófeltétel: $1 \leq Ind \leq N \wedge T(X[Ind])$

T egy tulajdonság vagy feltétel, aminek teljesülnie kell.

Algoritmus:



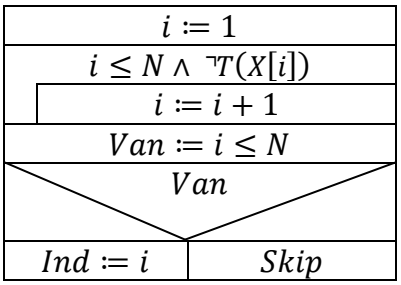
7.4 Keresés

Specifikáció:

Bemenet: $N: Egész \quad X: Tömb[1..N: Típus_1]$
Kimenet: $Van: Logikai \quad Ind: Egész$
Előfeltétel: $N \geq 0$
Utófeltétel: $Van = \exists i(1 \leq i \leq N): T(X[i]) \wedge Van \rightarrow \big(1 \leq Ind \leq N \wedge T(X[Ind])\big)$

T egy tulajdonság vagy feltétel, aminek teljesülnie kell.

Algoritmus:



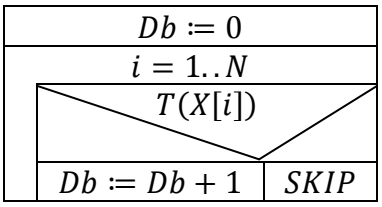
7.5 Megszámolás

Specifikáció:

Bemenet: $N: Egész \quad X: Tömb[1..N: Típus_1]$
Kimenet: $Db: Egész$
Előfeltétel: $N \geq 0$
Utófeltétel: $Db = \sum_{\substack{i=1 \\ T(X[i])}}^{N} 1$

T egy tulajdonság vagy feltétel, aminek teljesülnie kell.

Algoritmus:



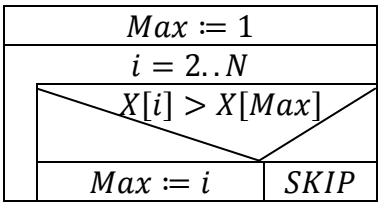
7.6 Maximum kiválasztás

Specifikáció:

Bemenet: $N: Egész \quad X: Tömb[1..N: Típus_1]$
Kimenet: $Max: Egész$
Előfeltétel: $N > 0$
Utófeltétel: $1 \leq Max \leq N \wedge \forall i(1 \leq i \leq N): X[Max] \geq X[i]$

Léteznie kell a $\geq: Típus_1 \times Típus_1 \rightarrow Logikai$ rendezési relációnak!

Algoritmus:



(Minimum kiválasztás hasonlóan, de a specifikációban a $\geq$ -ből $\leq$ -lesz továbbá az algoritmusban: $X[i] < X[Max]$. A logikus használat értelmében pedig a Max elnevezést Min-re cseréljük mindenhol!)

7.7 Kiválogatás

Specifikáció:

Bemenet: $N: Egész \quad X: Tömb[1..N: Típus_1]$
Kimenet: $Db: Egész \quad Y: Tömb[1..Db: Egész]$
Előfeltétel: $N \geq 0$
Utófeltétel: $Db = \sum_{\substack{i=1 \\ T(X[i])}}^{N} 1 \wedge \forall i(1 \leq i \leq Db): T\big(X\big[Y[i]\big]\big) \wedge Y \subseteq (1,2,\dots,N)$

Algoritmus:

$Db \coloneqq 0$	
$i = 1..N$	
$T(X[i])$	
$Db \coloneqq Db + 1$	$Skip$
$Y[Db] \coloneqq i$	

Ha az értékeket szeretnénk kiválogatni és nem az indexeket akkor:
$Y[Db] \coloneqq i$ helyett $Y[Db] \coloneqq X[i]$ -t kell vennünk. (Ekkor a specifikációt is módosítani kell!)

7.8 Szétválogatás

Specifikáció:

Bemenet: $N: Egész \quad X: Tömb[1..N: Típus_1]$
Kimenet: $Db: Egész \quad Y, Z: Tömb[1..N: Egész]$
Előfeltétel: $N \geq 0$
Utófeltétel:
$Db = \sum_{\substack{i=1 \\ T(X[i])}}^{N} 1 \wedge \forall i(1 \leq i \leq Db): T\big(X\big[Y[i]\big]\big) \wedge \forall i(1 \leq i \leq N - Db): \neg T\big(X\big[Z[i]\big]\big) \wedge$

$\wedge Y \subseteq (1,2,\dots,N) \wedge Z \subseteq (1,2,\dots,N)$

Algoritmus:

$Db \coloneqq 0 \quad DbZ \coloneqq 0$	
$i = 1..N$	
$T(X[i])$	
$Db \coloneqq Db + 1$	$DbZ \coloneqq DbZ + 1$
$Y[Db] \coloneqq i$	$Z[DbZ] \coloneqq i$

Ha az értékeket szeretnénk szétválogatni és nem az indexeket akkor:
$Y[Db] \coloneqq i\ , Z[DbZ] \coloneqq i$ helyett $Y[Db] \coloneqq X[i], Z[Db] \coloneqq X[i]$ -t kell vennünk. (Ekkor a specifikációt is módosítani kell!)

8. Minta kódok elemi programozási tételekre

8.1 Sorozatszámítás / Összegzés

Az általunk megadott N darab szám összege:

```
#include <iostream>

using namespace std;

int main()

{

      int n;
      cout << "Kerem a tomb elemszamat! : ";
      cin >> n;
      int x[n];
      for (int i = 0; i < n; i++)
      {

        cout << "Kerem a tomb " << i+1 << ". elemet! : ";
        cin >> x[i];

      }

      // Összegzés:

       int s = 0;

       for (int i = 0; i < n; i++)
       {

             s=s+x[i];

       }

        cout << "A számok összege: " << s;


      return 0;

}
```

Megjegyzés: A tömb beolvasása ennél a feladatnál nincs biztosítva, ezért feltételezzük, hogy a felhasználó helyes értéket ad meg! A későbbiekben viszont fontos, hogy teljes hibakezelést és megelőzést készítsünk!

8.2 Eldöntés

Van-e 0 a tömbünkben?

```
#include <iostream>

using namespace std;

int main()

{

    int n;
    bool van;

    cout << "Kerem a tomb elemszamat! : ";

    cin >> n;
    int x[n];

    for (int i = 0; i < n; i++)
    {
        cout << "Kerem a tomb " << i+1 << ". elemet! : ";
        cin >> x[i];
    }

    // Eldöntés:

    int i = 0;

    while (i <= n && x[i] != 0)
    {
        i++;
    }

    van = i <= n;
    cout << van;

    return 0;

}
```

Megjegyzés: A tömb beolvasása ennél a feladatnál nincs biztosítva, ezért feltételezzük, hogy a felhasználó helyes értéket ad meg! A későbbiekben viszont fontos, hogy teljes hibakezelést és megelőzést készítsünk!

8.3 Kiválasztás

A tétel működése értelmében a lentebbi kód szerint a tömb egyik megadott elemének 0-nak kell lennie!

```
#include <iostream>

using namespace std;

int main()
{
    int n;

    cout << "Kerem a tomb elemszamat! : ";
    cin >> n;
    int x[n];

    for (int i = 0; i < n; i++) {
        cout << "Kerem a tomb " << i+1 << ". elemet! : ";
        cin >> x[i];
    }

    //Kiválasztás
    int counter = 0;
    while (counter <= n && x[counter]!=0)
    {
        counter++;
    }
    cout << counter+1 << ". helyen talaltuk meg a keresett erteket.";


    return 0;
}
```

Megjegyzés: A tömb beolvasása ennél a feladatnál nincs biztosítva, ezért feltételezzük, hogy a felhasználó helyes értéket ad meg! A későbbiekben viszont fontos, hogy teljes hibakezelést és megelőzést készítsünk!

8.4 Keresés

Keressük meg egy tömb elemei között, hogy szerepel-e az 50-es szám!

```
#include <iostream>

using namespace std;

int main()
{
    int n;
    bool van;

    cout << "Kerem a tomb elemszamat! : ";
    cin >> n;
    int x[n];

    for (int i = 0; i < n; i++) {
        cout << "Kerem a tomb " << i+1 << ". elemet! : ";
        cin >> x[i];
    }

    //Keresés
    int counter = 0;
    while (counter < n && x[counter] != 50)
    {
        counter++;
    }
    van = counter < n;
    if (van)
    {
         cout << counter+1 << ". helyen talaltuk meg az 50-est!";
    }
    else
    {
        cout << "Nem talaltunk ilyen elemet!";
    }

    return 0;
}
```

Megjegyzés: A tömb beolvasása ennél a feladatnál nincs biztosítva, ezért feltételezzük, hogy a felhasználó helyes értéket ad meg! A későbbiekben viszont fontos, hogy teljes hibakezelést és megelőzést készítsünk!

8.5 Megszámolás

Hány darab 0-át találunk az adott tömbünkben?

```
#include <iostream>

using namespace std;

int main()
{
    int n;

    cout << "Kerem a tomb elemszamat! : ";
    cin >> n;
    int x[n];

    for (int i = 0; i < n; i++)
    {
        cout << "Kerem a tomb " << i+1 << ". elemet! : ";
        cin >> x[i];
    }

    //Megszámolás
    int db = 0;
    for (int i = 0; i < n; i++)
    {
        if (x[i] == 0)
        {
            db++;
        }
    }
    cout << db << " darab ilyen tulajdonsagu elemet talaltunk!";

    return 0;
}
```

Megjegyzés: A tömb beolvasása ennél a feladatnál nincs biztosítva, ezért feltételezzük, hogy a felhasználó helyes értéket ad meg! A későbbiekben viszont fontos, hogy teljes hibakezelést és megelőzést készítsünk!

8.6 Maximum és minimum kiválasztás

```
#include <iostream>

using namespace std;

int main()
{
    int n;
    do
      {
            cout << "Kerem a tomb elemszamat! : ";
            cin >> n;
      }while(n<2);
    int x[n];

    for (int i = 0; i < n; i++)
    {
        cout << "Kerem a tomb " << i+1 << ". elemet! : ";
        cin >> x[i];
    }

    //Maximum kiválasztás
    int max = 0;
    for (int i = 1; i < n; i++)
    {
        if (x[max] < x[i])
        {
            max=i;
        }
    }
    cout << x[max] << " a maximalis ertek!\n";

    //Minimum kiválasztás
    int min = 0;
    for (int i = 1; i < n; i++)
    {
        if (x[min] > x[i])
        {
            min=i;
        }
    }
    cout << x[min] << " a minimalis ertek!";

    return 0;
}
```

Megjegyzés: A tömb beolvasása ennél a feladatnál nincs biztosítva, ezért feltételezzük, hogy a felhasználó helyes értéket ad meg! A későbbiekben viszont fontos, hogy teljes hibakezelést és megelőzést készítsünk!

8.7 Kiválogatás

Válogassuk ki a nullánál nagyobb elemeket egy másik tömbbe!

```
#include <iostream>

using namespace std;

int main()
{
    int n;

    cout << "Kerem a tomb elemszamat! : ";
    cin >> n;
    int x[n];
    int y[n];

    for (int i = 0; i < n; i++)
    {
        cout << "Kerem a tomb " << i+1 << ". elemet! : ";
        cin >> x[i];
    }

    //Kiválogatás
    int db = 0;
    for (int i = 0; i < n; i++)
    {
        if (x[i] > 0)
        {
            db++;
            y[db] = x[i];
        }
    }

    for (int i = 1; i <= db; i++)
    {
        cout << "A(z) " << i << ". kivalogatott elem : " << y[i] << endl;
    }

    return 0;
}
```

Megjegyzés: A tömb beolvasása ennél a feladatnál nincs biztosítva, ezért feltételezzük, hogy a felhasználó helyes értéket ad meg! A későbbiekben viszont fontos, hogy teljes hibakezelést és megelőzést készítsünk!

8.8 Szétválogatás

Válogassuk szét a tömb elemeit úgy, hogy a nullánál nagyobb elemeket tegyük egy tömbbe, míg a másikba az összes többit!

```
#include <iostream>

using namespace std;

int main()
{
    int n;

    cout << "Kerem a tomb elemszamat! : ";
    cin >> n;
    int x[n];
    int y[n];
    int z[n];

    for (int i = 0; i < n; i++)
    {
        cout << "Kerem a tomb " << i+1 << ". elemet! : ";
        cin >> x[i];
    }

    //Szétválogatás
    int db = 0;
    int db2 = 0;
    for (int i = 0; i < n; i++)
   {
        if (x[i] > 0)
       {
            db++;
            y[db] = x[i];
       }
        else
        {
            db2++;
            z[db2] = x[i];
        }
    }

    for (int i = 1; i <= db; i++) {
        cout << "A(z) " << i << ". elem Y tombbol : " << y[i] << endl;
    }

    cout << endl;

    for (int i = 1; i <= db2; i++) {
        cout << "A(z) " << i << ". elem Z tombbol : " << z[i] << endl;
    }


    return 0;
}
```

Megjegyzés: A tömb beolvasása ennél a feladatnál nincs biztosítva, ezért feltételezzük, hogy a felhasználó helyes értéket ad meg! A későbbiekben viszont fontos, hogy teljes hibakezelést és megelőzést készítsünk!

9. Minta zárthelyi dolgozat programozási alapismeretek tárgyhoz

Feladat:
Egy vállalkozó minden hónap végén feljegyzi, hogy az adott hónapban hogyan zárta a költségvetését! Tudjuk, hogy az összkiadása egyetlen hónapban sem haladja meg a 3 millió forintot, és azt is, hogy az összbevétel még a legjobb hónapokban sem éri el a 4 millió forintot!

1. ZH: Készíts programot, amely megmondja, hogy a rögzített hónapok közül hány alkalommal volt nyereséges a vállalkozó! Adj választ arra is, hogy a rögzített adatok összesítése alapján nyereséges-e a vállalkozó!
(Specifikáció és algoritmus adott a feladathoz!)
[A KÓD MEGTALÁLHATÓ A MELLÉKLETBEN]

2. ZH: Specifikáld a fentebbi feladatot és készítsd el a program algoritmusát!

Specifikáció:

Bemenet: $N: Egész \quad KV: Tömb[1..MaxN: Egész]$ Konstans: MaxN $= 100$
Kimenet: $NyerDb: Egész \quad Nyer\_e : Logikai$
Előfeltétel: $0 < N \leq MaxN \wedge \forall i(1 \leq i \leq N): -3000000 \leq KV[i] < 4000000$
Utófeltétel: $NyerDb \in [1, N] \wedge NyerDb = \sum_{\substack{i=1 \\ KV[i]>0}}^{N} 1$

$Nyer\_e = Össz > 0, ahol \quad Össz = \sum_{i=1}^{N} KV[i]$

Algoritmus:

$NyerDb \coloneqq 0$	
$Össz \coloneqq 0$	
$i = 1..N$	
$KV[i] > 0$	
$NyerDb \coloneqq NyerDb + 1$	$Skip$
$Össz \coloneqq Össz + KV[i]$	
$Nyer\_e \coloneqq Össz > 0$	

10. Függvények

10.1 Bevezető

Hogyan **deklarálhatunk** C++ nyelven egy függvényt:
[VISSZATÉRÉSI ÉRTÉK] [FÜGGVÉNY NEVE]([PARAMÉTEREK]);

Hogyan **definiálunk** egy függvényt C++ nyelven:
[VISSZATÉRÉSI ÉRTÉK] [FÜGGVÉNY NEVE]([PARAMÉTEREK])
{
  [PARANCSOK];
}

Hogy használhatunk függvényeket a főprogramunkban? (main.cpp)

[FÜGGVÉNY DEKLARÁCIÓ]

int main()
{
  … //Ide kerülhet a függvényhívás is : [FÜGGVÉNY NEVE]([SZÜKSÉGES PARAMÉTEREK]);
}

[FÜGGVÉNY DEFINÍCIÓ]

Miért jó?
Átláthatóbb lesz a program, továbbá a későbbiekben egyszerűbb lesz csomagokra bontani a programunkat.

10.2 Eljárások a C++ nyelvben

Egyes programozási nyelvekben eljárásként szereplő függvény egy olyan függvény, melynek nincsen visszatérési értéke. (void)

```
void Kiír()
{
    std::cout << "Hello World!" << std:endl;
}
```

10.3 Függvények a C++ nyelvben

Olyan függvények, melyeknek van visszatérési értékük.

1. példa:

```
std::string Kiir()
{
    return "Hello World!";
}
```

2. példa:

```
int Szam()
{
    int m = 0;
    return m;
}
```

Arra kell figyelnünk, hogy ha int visszatérési értéket vár a függvényünk, akkor a return kulcsszó után olyan változót kell írnunk, ami megfelelő típusú.

10.4 Paraméterátadás

A C++-ban két féle paraméterátadási mód van.

1. Érték szerinti, az átadott típusból másolat készül a memóriában, az eredeti értéket nem módosítja a függvény.

```
int Osszeg(int a, int b)
{
   return a + b;
}
```

Itt a és b összegével tér vissza a függvény.

2. Cím szerinti, paraméterként az átadott típus referenciája szerepel, a függvény módosíthatja a paramétereket.

```
void Osszeg(int a, int b, int &c)
{
   c = a + b;
}
```

Itt a harmadik paraméter nem "c" értéke, hanem a memóriában elfoglalt címe.

10.5 Lokális változók

A függvények belsejében (illetve a programban lévő blokkokon belül) deklarált változókat lokális változóknak nevezzük. Ez a gyakorlatban azt jelenti, hogy a láthatóságuk és élettartalmuk a függvényen (blokkon) belülre korlátozódik. A lokális változók a függvényhívás végén automatikusan megsemmisülnek és kívülről nem hivatkozhatóak.

```
//Két változó értékének cseréje
void swap(int &a, int &b)
{
 int tmp = a; //nem dinamikusan (statikusan) lefoglalt változó
 a = b;
 b = tmp;
}
tmp = 10; //Hiba, tmp nem hivatkozható a hatókörén (a függvény blokkján) kívül
```

A dinamikus objektumokra mutató pointerek szintén megsemmisülnek a hatókörükből kikerülve, de az objektum maga nem.

```
int * createArray(int n)
{
 int * v = new int [n];
 return v; //A függvény egy n elemű tömbre mutató pointerrel tér vissza
}

int * t = createArray(10);
t[0] = 12; //Működik, most t mutat a tömbre
v[1] = 2; //Hiba, v már nem létezik
```

Ha nem gondoskodunk a blokkon belül létrehozott dinamikus objektum külső elérhetőségéről, az érvényes hivatkozás nélkül a memóriában marad, azaz memóriaszivárgás keletkezik.

```
void func()
{
 int * v = new int [10];
}

v[0] = 12;
/*Hiba, a tömbre mutató pointer már nem létezik,
 és más sem mutat rá -> memóriaszivárgás*/
```

11. Átvezetés

Program terve:

- Specifikáció
- Visszavezetés
- Algoritmus (Struktogramm)

Bevezető feladat:

(Programozási alapismeret-ről átállás Programozás-ra)

Adott két nem negatív egész szám, számítsuk ki a szorzatukat úgy, hogy a szorzás műveletét nem használjuk!

Programozási alapismereteken tanult specifikációs módszer:

Bemenet: $x, y: egész$
Kimenet: $z: egész$
Előfeltétel: $x \geq 0 \; és \; y \geq 0$
Utófeltétel: $z = x * y$

Programozáson használt specifikációs módszer:

$A :=$ állapottér (csak a bemeneti és kimeneti paramétereket tartalmazza)
$Ef :=$ előfeltétel (megjelennek benne a bemenetekre vonatkózó kezdőértékek, amennyiben vannak) – A bemenő adatokra kirótt feltételeket tartalmazzák
$Uf :=$ utófeltétel – A kimenő adatokra kirótt feltételeket tartalmazzák

Ezek alapján a fentebbi feladatot így lehetne felírni:

$A = (x: \mathbb{Z}, y: \mathbb{Z}, z: \mathbb{Z})$ (Megjelennek a matematikában használt jelölések)
$Ef = (x = x' \wedge y = y' \wedge x' \geq 0 \wedge y' \geq 0)$ (x', y'- az adott kezdőértékek)
$Uf : nézzünk \; az \; adott \; feladatra \; több \; algoritmust. (Mindegyiknek \; más \; az \; Uf - e)$

$x \geq 0 \; és \; y \geq 0$

→

$z := 0$
$i := 1 \ldots x$
$z := z + y$

$z = x * y$ ✔
$Uf = (Ef \wedge z = x * y)$

$z := 0$
$i := 1$
$i \leq x$
$z := z + y$
$i := i + 1$

$x > 0$ és $y \geq 0$

$z \coloneqq 0$	
$y > 0$	
	$z \coloneqq z + x$
	$y = y - 1$

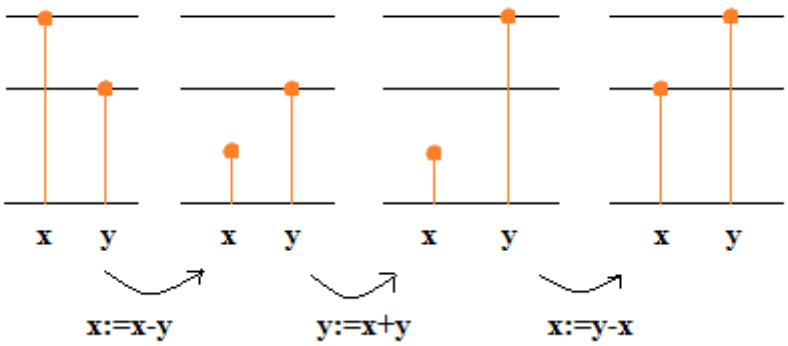
$z \neq x * y \rightarrow Uf: (z = z' * y')$

1. feladat:

Az adott algoritmus alapján specifikáld a programot és határozd meg, hogy mit csinál!

$x \coloneqq x - y$
$y \coloneqq x + y$
$x \coloneqq y - x$

Egy program sorról sorra fut le. Vegyünk két tetszőleges számot és nézzük meg, hogy mi történik.
Legyen $x = 4$ és $y = 1$
Ekkor:
$x \coloneqq x - y \rightarrow x \coloneqq 4 - 1 \rightarrow x = 3$
$y \coloneqq x + y \rightarrow y \coloneqq 3 + 1 \rightarrow y = 4$
$x \coloneqq y - x \rightarrow x \coloneqq 4 - 3 \rightarrow x = 1$
Ha megnézzük az x, y értékeit, akkor jelenleg: $x = 1, y = 4 \rightarrow$ ez egy csere program, mely felcseréli az x és y értékeit.



Specifikáció:
$A = (x: \mathbb{R}, y: \mathbb{R})$
$Ef = (x = x' \wedge y = y')$
$Uf = (x = y' \wedge y = x')$

2. feladat:
Adott egy k pozitív egész szám. Adjuk meg a hozzá legközelebb álló prímszámot!

Specifikáció:
$A = (k: \mathbb{Z}, p: \mathbb{Z})$
$Ef = (k = k' \wedge k > 0)$
$Uf = (k = k' \wedge prim(p) \wedge \forall i > 1: prim(i) \rightarrow |k - i| \geq |k - p|)$

$(10,1) \rightarrow$
$(10,*) \rightarrow$ $\Big\}$ $(10,11)$

$(9,5) \rightarrow$ $(9,7)$ $(9,11)$

$prim(p): \mathbb{Z} \rightarrow \mathbb{L}$
$prim(p): p > 1 \wedge \forall x: (2 \leq x \leq p-1): x \nmid p$
$\forall x \in [2 \ldots p-1]: \neg(x \mid p)$

Feladat:
Specifikáld: Adott egy összetett természetes szám. Adjuk meg **egy** valódi osztóját!

$A = (x: \mathbb{N}^+, y: \mathbb{N}^+)$
$Ef = \left(x = x' \wedge \neg prim(x)\right)$
$Uf = (y \mid x \wedge y \neq 1 \wedge y \neq x')$

Feladat:
Specifikáld: Adott egy összetett természetes szám. Adjuk meg **az összes** valódi osztóját!

$A_1 = \left(x: \mathbb{N}, y: 2^{\mathbb{N}}\right) \rightarrow y\ egy\ halmaz$
$A_2 = (x: \mathbb{N}, y: \mathbb{N}^*) \rightarrow y\ egy\ sorozat\ (képernyőre\ ír\ ki\ a\ program)$
$A_3 = \left(x: \mathbb{N}, y: \mathbb{N}^k\right) \rightarrow y\ egy\ tömb$

$Ef = (x = x' \wedge \exists i \in [2 \ldots x-1]: i \mid x)$

$Uf_1 = (Ef \wedge y = \{i \mid i \in [2 \ldots x-1] \wedge i \mid x\})$
$Uf_2 = \left(Ef \wedge y = \bigoplus_{\substack{i=2 \\ i|x}}^{x\ div\ 2} < i > \right)$ Megjegyzés: $\bigoplus$:= összefűzés (asszociatív, bal oldali 0-elemes művelet.
$Uf_3 = \left(Ef \wedge y[1 \ldots k] = \bigoplus_{\substack{i=2 \\ i|x}}^{x\ div\ 2} < i > \right)$

Feladat:
Adott egy $ax + b = 0$ alakú egyenlet, melyben a, b tetszőleges valós számok. Keressük az x-et.

$x = -\frac{b}{a}$

- a nem lehet 0
- ha $a = 0$ és $b = 0$ → azonosság

Specifikáció:

$A = (a{:}\,\mathbb{R}, b{:}\,\mathbb{R}, x{:}\,\mathbb{R}, l{:}\,\mathbb{L}, s{:}\,\mathbb{K}^*)$ (Megjegyzés: $\mathbb{K}^*$ karakter sorozat – string)

$Ef = (a = a' \wedge b = b')$

$$Uf = (Ef \wedge (a = 0 \wedge b = 0 \rightarrow l = \uparrow \wedge s = \text{"}azonosság\text{"}) \wedge (a = 0 \wedge b \neq 0 \rightarrow l = \downarrow \wedge s = \text{"}ellentmondás\text{"}) \wedge \left(a \neq 0 \rightarrow l = \uparrow \wedge x = -\frac{b}{a}\right))$$

Más féle utófeltétellel felírva:

$$Uf = (Ef \wedge (a = 0 \wedge b = 0 \wedge l = \uparrow \wedge s = \text{"}azonosság\text{"}) \vee (a = 0 \wedge b \neq 0 \wedge l = \downarrow \wedge s = \text{"}ellentmondás\text{"}) \vee \left(a \neq 0 \wedge l = \uparrow \wedge x = -\frac{b}{a}\right))$$

12. Intervallumos programozási tételek

12.1 Összegzés

Feladat:
Adott egy $f\colon [m..n] \to H$ függvény. A H halmaz elemein értelmezett egy asszociatív, baloldali nulla elemmel rendelkező művelet (nevezzük ezt összeadásnak és jelölje a +). Határozzuk meg a függvény intervallumon felvett értékeinek összegét!

Specifikáció:

$A = (m\colon \mathbb{Z}, n\colon \mathbb{Z}, s\colon H)$
$Ef = (m = m' \wedge n = n')$
$$Uf = \left(Ef \wedge s = \sum_{i=m}^{n} f(i) \right)$$

Algoritmus:

$s \coloneqq 0$	
$i = m \ldots n$	
	$s \coloneqq s + f(i)$

12.2 Számlálás

Feladat:
Adott egy $\beta\colon [m..n] \to \mathbb{L}$ feltétel. Határozzuk meg, hogy hányszor teljesül az intervallumon a feltétel, azaz hányszor veszi fel az igaz értéket!

Specifikáció:

$A = (m\colon \mathbb{Z}, n\colon \mathbb{Z}, c\colon \mathbb{N})$
$Ef = (m = m' \wedge n = n')$
$$Uf = \left(Ef \wedge c = \sum_{\substack{i=m \\ \beta(i)}}^{n} 1 \right)$$

Algoritmus:

$c \coloneqq 0$	
$i = m \ldots n$	
$\beta(i)$	
$c \coloneqq c + 1$	$Skip$

12.3 Maximum kiválasztás / Minimum kiválasztás

Feladat:
Adott egy $f: [m..n] \to H$ függvény. A H halmaz elemein értelmezett egy teljes rendezési reláció. Határozzuk meg melyik a függvény legnagyobb értéke és adjuk meg az egyik olyan intervallumbeli elemet, ahol a függvény ezt az értéket felveszi!

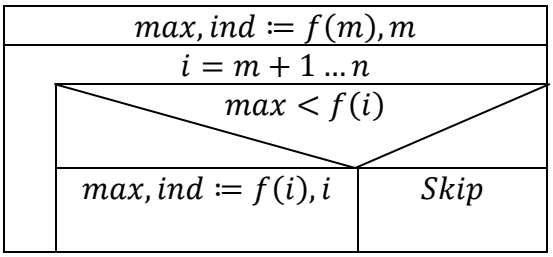
Specifikáció:

$A = (m: \mathbb{Z}, n: \mathbb{Z}, max: H, ind: \mathbb{Z})$
$Ef = (m = m' \wedge n = n' \wedge m \leq n)$
$Uf = \big(Ef \wedge ind \in [m..n] \wedge max = f(ind) \wedge \forall i \in [m..n]: max \geq f(i)\big)$

A fentebbi utófeltétel leegyszerűsítve:

$Uf = \big(Ef \wedge max, ind = MAX_{i=m}^{n} f(i)\big)$

Algoritmus:



Feladat:
Adott egy $f: [m..n] \to H$ függvény. A H halmaz elemein értelmezett egy teljes rendezési reláció. Határozzuk meg melyik a függvény legkisebb értéke és adjuk meg az egyik olyan intervallumbeli elemet, ahol a függvény ezt az értéket felveszi!

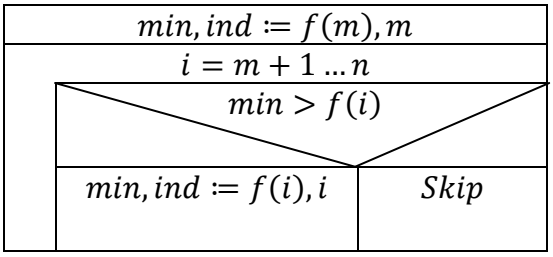
Specifikáció:

$A = (m: \mathbb{Z}, n: \mathbb{Z}, min: H, ind: \mathbb{Z})$
$Ef = (m = m' \wedge n = n' \wedge m \leq n)$
$Uf = \big(Ef \wedge ind \in [m..n] \wedge min = f(ind) \wedge \forall i \in [m..n]: min \leq f(i)\big)$

A fentebbi utófeltétel leegyszerűsítve:

$Uf = \big(Ef \wedge min, ind = MIN_{i=m}^{n} f(i)\big)$

Algoritmus:



12.4 Kiválasztás (Szekvenciális vagy Lineáris kiválasztás)

Feladat:
Adott egy $\beta: \mathbb{Z} \to \mathbb{L}$ feltétel és egy m egész szám. A feltétel az m -nél nagyobb vagy egyenlő egész számokra van értelmezve, legalább is az első olyan egész számig, ahol a feltétel igaz értéket vesz fel (teljesül). Ilyen egész szám biztosan van! Határozzuk meg az m -nél nagyobb vagy egyenlő legelső olyan egész számot, amelyre a feltétel teljesül!

Specifikáció:

$A = (m: \mathbb{Z}, i: \mathbb{Z})$
$Ef = \big(m = m' \wedge \exists k \geq m: \beta(k)\big)$
$Uf = (Ef \wedge i \geq m \wedge \beta(i) \wedge \forall k \in [m..i-1]: \neg\beta(k))$

A fentebbi utófeltétel leegyszerűsítve:

$Uf = \big(Ef \wedge i = SELECT_{i \geq m}\beta(i)\big)$

Algoritmus:

$i \coloneqq m$	
$\neg\beta(i)$	
	$i \coloneqq i + 1$

12.5 Lineáris keresés

Feladat:
Adott egy $\beta: [m..n] \to \mathbb{L}$ feltétel. Határozzuk meg az intervallum első olyan elemét, amelyre teljesül a feltétel!

Pesszimista eldöntésű Lineáris keresés**:**

Feladat**:**
Van-e olyan eleme az intervallumnak, amelyre teljesül a feltétel? – Ilyenkor mind a specifikációból, mind az algoritmusból elhagyható az ind változó, mivel nem vagyunk kíváncsiak arra, hogy hol található, csak arra, hogy Van-e?

Specifikáció:

$A = (m: \mathbb{Z}, n: \mathbb{Z}, l: \mathbb{L}, ind: \mathbb{Z})$
$Ef = (m = m' \wedge n = n')$
$Uf = \Big(Ef \wedge \big(l = \exists i \in [m..n]: \beta(i)\big) \wedge l \to ind \in [m..n] \wedge \beta(ind) \wedge \forall i \in [m..ind-1]: \neg\beta(i)\Big)$

A fentebbi utófeltétel leegyszerűsítve:

$Uf = \big(Ef \wedge l, ind = SEARCH^{n}_{i=m}\beta(i)\big)$

Algoritmus:

$l, i \coloneqq false, m$	
$\neg l \wedge i \leq n$	
	$l, ind \coloneqq \beta(i), i$
	$i \coloneqq i + 1$

<u>Optimista eldöntésű Lineáris keresés</u>:

Feladat:
Igaz-e, hogy az intervallumnak minden elemére teljesül a feltétel?

Specifikáció:

$A = (m{:}\,\mathbb{Z}, n{:}\,\mathbb{Z}, l{:}\,\mathbb{L})$
$Ef = (m = m' \wedge n = n')$
$Uf = \Big(Ef \wedge \big(l = \forall i \in [m..n]{:}\,\beta(i)\big)\Big)$

A fentebbi utófeltétel leegyszerűsítve:

$Uf = \big(Ef \wedge l = \forall SEARCH_{i=m}^{n}\beta(i)\big)$

Algoritmus:

$l, i \coloneqq true, m$	
$l \wedge i \leq n$	
	$l \coloneqq \beta(i)$
	$i \coloneqq i + 1$

12.6 Feltételes maximum/minimum keresés

Feladat:
Adott egy $f\colon [m..n] \to H$ függvény és egy $\beta\colon [m..n] \to \mathbb{L}$ feltétel. A H halmaz elemein értelmezett egy teljes rendezési reláció. Határozzuk meg, hogy melyik a függvény legnagyobb értéke azok között, amelyeket olyan intervallumbeli elemhez rendel, amelyek kielégítik a feltételt! Adjuk meg az egyik olyan intervallumbeli elemet, amelyre a feltétel teljesül és ahol a függvény ezt a maximális értéket veszi fel!

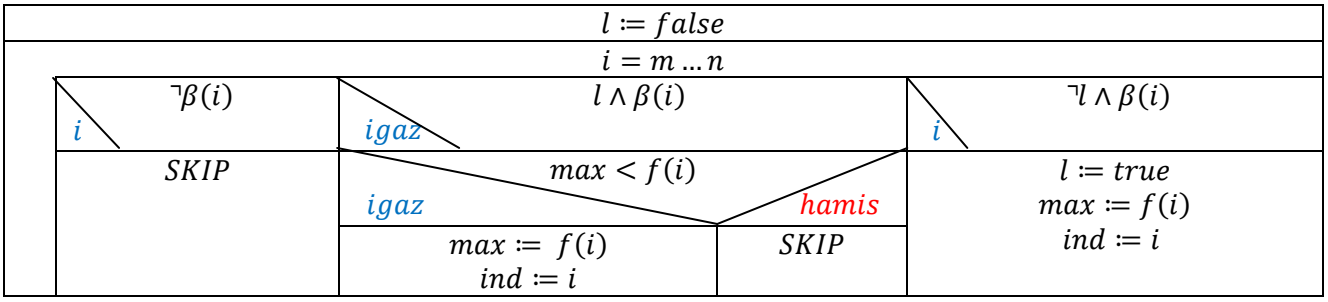
Specifikáció:

$$A = (m\colon \mathbb{Z}, n\colon \mathbb{Z}, l\colon \mathbb{L}, ind\colon \mathbb{Z}, max\colon H)$$
$$Ef = (m = m' \wedge n = n')$$
$$Uf = \Big(Ef \wedge \big(l = \exists i \in [m..n]\colon \beta(i)\big) \wedge \Big(l \to ind \in [m..n] \wedge max = f(ind) \wedge \big(\forall i \in [m..n]\colon \beta(i) \to max \geq f(i)\big)\Big)\Big)$$

A fentebbi utófeltétel leegyszerűsítve:

$$Uf = \left(Ef \wedge (l, max, ind) = MAX^{n}_{\substack{i=m \\ \beta(i)}} f(i)\right)$$

Algoritmus:



Feladat:
Adott egy $f: [m..n] \to H$ függvény és egy $\beta: [m..n] \to \mathbb{L}$ feltétel. A H halmaz elemein értelmezett egy teljes rendezési reláció. Határozzuk meg, hogy melyik a függvény legkisebb értéke azok között, amelyeket olyan intervallumbeli elemhez rendel, amelyek kielégítik a feltételt! Adjuk meg az egyik olyan intervallumbeli elemet, amelyre a feltétel teljesül és ahol a függvény ezt a minimális értéket veszi fel!

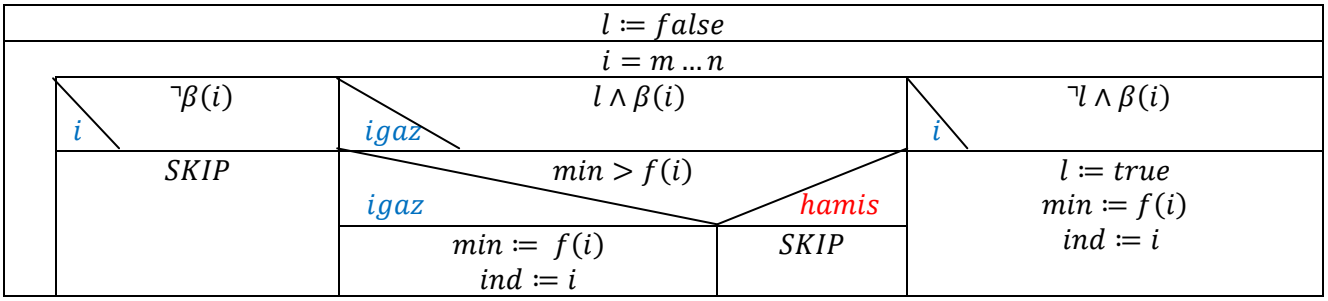
Specifikáció:

$$A = (m: \mathbb{Z}, n: \mathbb{Z}, l: \mathbb{L}, ind: \mathbb{Z}, min: H)$$
$$Ef = (m = m' \wedge n = n')$$
$$Uf = \Big(Ef \wedge \big(l = \exists i \in [m..n]: \beta(i)\big) \wedge \Big(l \to ind \in [m..n] \wedge min = f(ind) \wedge \big(\forall i \in [m..n]: \beta(i) \to min \leq f(i)\big)\Big)\Big)$$

A fentebbi utófeltétel leegyszerűsítve:

$$Uf = \left(Ef \wedge (l, min, ind) = MIN_{\substack{i=m \\ \beta(i)}}^{n} f(i)\right)$$

Algoritmus:



12.7 Logaritmikus keresés

Feladat:
Adott az egész számok egy $[m..n]$ intervalluma és egy $f: \mathbb{Z} \to H$ függvény, amelyik az $[m..n]$ intervallumon monoton növekvő. (A H halmaz elemei között értelmezett egy rendezési reláció.) Keressük meg a függvény $[m..n]$ intervallumon felvett értékei között egy adott értéket!

Specifikáció:

$A = (m: \mathbb{Z}, n: \mathbb{Z}, h: H, l: \mathbb{L}, ind: \mathbb{Z})$
$Ef = \big(m = m' \wedge n = n' \wedge h = h' \wedge \forall j \in [m..n-1]: f(j) \leq f(j+1)\big)$
$Uf = \big(Ef \wedge l = (\exists j \in [m..n]: f(j) = h) \wedge l \to (ind \in [m..n] \wedge f(ind) = h)\big)$

Algoritmus:

<table>
<tr><td colspan="3">$ah, fh, l \coloneqq m, n, false$</td></tr>
<tr><td colspan="3">$\neg l \wedge ah \leq fh$</td></tr>
<tr><td colspan="3">$ind \coloneqq \left\lfloor \frac{(ah + fh)}{2} \right\rfloor$</td></tr>
<tr><td>$f(ind) > h$</td><td>$f(ind) < h$</td><td>$f(ind) = h$</td></tr>
<tr><td>$fh \coloneqq ind - 1$</td><td>$ah \coloneqq ind + 1$</td><td>$l \coloneqq true$</td></tr>
</table>

12.8 Rekurzív függvény kiszámítása

Feladat:
Legyen az $f: \mathbb{Z} \to H$ egy k-ad rendű m bázisú rekurzív függvény ($m \in \mathbb{Z}, k \in \mathbb{Z}^+$), azaz $f(i) = h\big(i, f(i-1), \dots, f(i-k)\big)$, ahol $i \geq m$ és $h: \mathbb{Z} \times H^k \to H$
$f(m-1) = e_{m-1}, \dots, f(m-k) = e_{m-k}$,ahol $e_{m-1}, \dots, e_{m-k} \in H$
Számítsuk ki az f függvény adott n $(n \geq m)$ helyen felvett értékét!

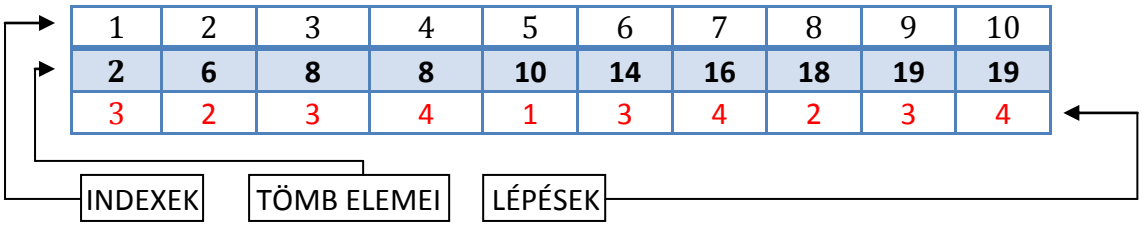
Specifikáció:

$A = (n: \mathbb{Z}, y: H)$
$Ef = (n = n' \wedge n \geq m)$
$Uf = \big(Ef \wedge y = f(n)\big)$

Algoritmus:

<table>
<tr><td>$y, y_1, \dots, y_{k-1}, i \coloneqq e_{m-1}, e_{m-2}, \dots, e_{m-k}, m$</td></tr>
<tr><td>$i \leq n$</td></tr>
<tr><td>$y, y_1, y_2, \dots, y_{k-1} \coloneqq h(i, y, y_1, y_2, \dots, y_{k-1}), y, y_1, \dots, y_{k-2}$</td></tr>
<tr><td>$i \coloneqq i + 1$</td></tr>
</table>

13. A logaritmikus keresés működése



1. Keressük a h=19-et:

$ah = 1 \;\; fh = 10 \;\; ind = \left\lfloor \frac{1+10}{2} \right\rfloor = 5$

$ah = 6 \;\; fh = 10 \;\; ind = \left\lfloor \frac{6+10}{2} \right\rfloor = 8$

$ah = 9 \;\; fh = 10 \;\; ind = \left\lfloor \frac{9+10}{2} \right\rfloor = 9 \;\; \rightarrow megtalálta$

2. Keressük a h=15-öt (Nincs ilyen elem a tömbbe most)

$ah = 1 \;\; fh = 10 \;\; ind = 5$
$ah = 6 \;\; fh = 10 \;\; ind = 8$
$ah = 6 \;\; fh = 7 \;\; ind = 6$
$ah = 7 \;\; fh = 7 \;\; ind = 7$
$ah = 7 \;\; fh = 6 \;\;$ → nem teljesül a ciklus feltétel → Nincs ilyen elem

A fenti képből adódóan látszik, hogy a Logaritmikus keresés sokszor hatékonyabb a Lineáris keresésnél.
Egy 10 elemű tömb esetén, amennyiben a keresett érték az első helyen áll, úgy a Lineáris kereséssel hamarabb megtaláljuk. Amennyiben a tömbben a 2,3,4-es helyen található, úgy azonos hatékonysággal találja meg a két tétel. Amennyiben 5,6,7,8,9,10. helyen található a keresett elem úgy a Logaritmikus keresés a hatékonyabb.

Miért hívják logaritmikus keresésnek?

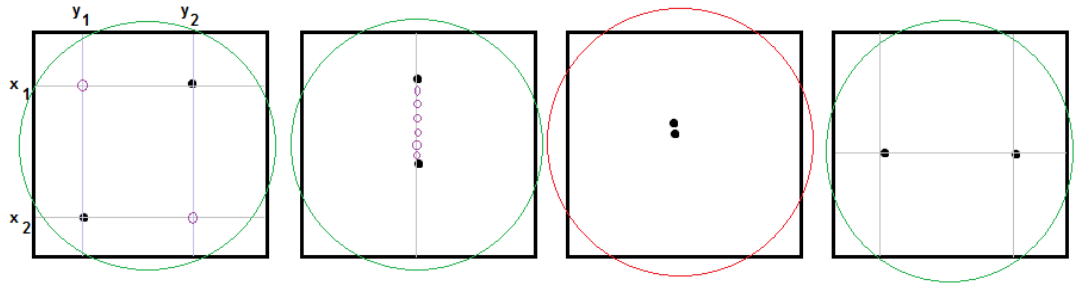
Azért, mert a maximális összehasonlítások száma: $\lceil \log_2 n \rceil$

14. Intervallumos programozási tételekre visszavezethető feladatok

14.1 Sakktábla

Adott egy sakktábla és rajta két bástya, tegyünk le úgy egy harmadik bástyát, hogy mind a kettőt üsse!

Lehetőségek:



- Nem állhatnak azonos mezőn

Specifikáció:

$$A = (x_1, y_1, x_2, y_2 \in \{1,2,\dots,8\}, z_1, z_2 \in \{1,2,\dots,8\}, l: \mathbb{L})$$

$$Ef = \left(x_1 = x_1' \wedge\ y_1 = y_1' \wedge\ x_2 = x_2' \wedge y_2 = y_2' \wedge \neg(x_1 = x_2 \wedge y_1 = y_2)\right)$$

ne legyenek azonos mezőn a bástyák

$$Uf = \left(Ef\ \wedge \left(l = (x_1 = x_2 \wedge\ |y_1 - y_2| > 1) \vee (y_1 = y_2 \wedge\ |x_1 - x_2| > 1) \vee (x_1 \neq x_2\ \wedge y_1 \neq y_2)\right) \wedge\right.$$
$$l \rightarrow \left((x_1 \neq x_2\ \wedge y_1 \neq y_2) \rightarrow (z_1 = x_2 \wedge z_2 = y_1) \vee (z_1 = x_1 \wedge z_2 = y_2)\right) \wedge$$
$$\left((x_1 = x_2 \wedge\ |y_1 - y_2| \geq 1) \rightarrow \left(z_1 = x_1 \wedge \min(y_1, y_2) < z_2 < max(y_1, y_2)\right)\right) \wedge$$
$$\left.\left((|x_1 - x_2| \geq 1 \wedge\ y_1 = y_2\ ) \rightarrow (\min(x_1, x_2) < z_1 < max(x_1, x_2) \wedge\ z_2 = y_1\ )\right)\right)$$

Magyarázat Uf-hez:

1.sor: A logikai „l” változót definiáljuk, azaz leírjuk azokat a lehetőségeket, amelyek teljesülése mellett igaz lesz a változó. Sorra: Ha a két lehelyezett bástya x helyzete megegyezik, akkor az y érzékek távolsága nagyobb legyen, mint 1. (Az adott x vonalban ne állhasson egymás mellett a két előre lehelyezett bástya) Vagy ugyan ez ne történhessen meg az y vonalában vagy (végezetül) ne lehessen egymásra tenni a két bástyát. (Ne állhasson egymáson a két bástya.

2.sor: Megadja a módszerét, hogyha a logikai „l” változó értéke igaz, akkor a programunknak mit kelljen csinálnia ahhoz, hogy ki tudja számolni a 3. bástya helyzetét. Itt ebben a sorban a fenti első ábrára ad megoldást.
3. sor: Negyedik ábrára nyújt megoldást
4. sor: Második ábrára nyújt megoldást.
Továbbá a minimum/maximum-os megoldás kiküszöböli a harmadik ábrán látott rossz esetet.

14.2 Átlaghőmérséklet

Egy nap folyamán n-szer megmértük a hőmérsékletet. Adjuk meg az átlaghőmérsékletet!

Specifikáció:

$A = (h: \mathbb{R}^n, n: \mathbb{N}, a: \mathbb{R})$
$Ef = (h = h' \wedge n' \neq 0 \wedge n = n')$
$Uf = \left(Ef \wedge a := \frac{\sum_{i=1}^{n} h[i]}{n}\right)$

Visszavezetés: összegzés tételre

$m \ldots n \sim 1 \ldots n$
$s, +, 0$
$f(i) \sim h[i]$

Magyarázat: Specifikáció alapján vezettük vissza az összegzés tételére. van egy n dimenziós tömbünk (h), aminek a dimenziója meg van határozva (ha 2, akkor 2 hőmérséklet szerepel benne, ha 3 akkor 3, ha n, akkor n…). A visszavezetés lényege, hogy az általunk használt változókat a specifikációban, ráhúzzuk az adott tételben már tanult változókra. Jelen esetben a tömböt 1-től nig definiáltuk. (Lásd Utófeltétel szumma.) Így a programozási tételünkbe az m:=1-el. Továbbá az f(i)-t felváltja a h[i], mivel jelen esetben egy tömb adott értékeit szeretnénk összegezni.Maga az s változót még nem használtuk ki, ezért hagyjuk, hogy az összeg változója ez legyen. (Látható az utófeltételből, hogy a:= s/n) továbbá a műveletünk a +, aminek a nulleleme/semleges 0. Ezért ezeken nem változtatunk. Most a megfeleltetések alapján kell megalkotnunk az algoritmust. Az Összegzés algoritmusát használjuk. A dolgunk

csak annyi, hogy a visszavezetésben használt változásokat írjuk be az algoritmusba és a végére hozzá tesszük az átlag számítást.

Algoritmus:

$s \coloneqq 0$
$i = 1 \dots n$
$s \coloneqq s + h[i]$
$a \coloneqq s/n$

14.3 Skaláris szorzat

Adjuk meg két vektor skalárszorzatát!

Specifikáció:

$A = (x: \mathbb{R}^n, y: \mathbb{R}^n, s: \mathbb{R})$
$Ef = (x = x' \wedge y = y')$
$Uf = \left(Ef \wedge s = \sum_{i=1}^{n} x[i]\, y[i] \right)$

Visszavezetés: összegzés tételre

$m \dots n \sim 1 \dots n$
$s, +, 0$
$f(i) \sim x[i]y[i]$

Algoritmus:

$s \coloneqq 0$
$i = 1 \dots n$
$s \coloneqq s + x[i]y[i]$

14.4 Kamatos kamat

Adott egy bankbetét indulóértéke. Évenként kamatos kamattal növekszik és évenként változó kamatlábbal n-évig. Mennyi lesz a betét végértéke?

Segítség:
Kamatos kamat:
$r_1 = 0{,}07\ (7\%), r_2 = 0{,}065\ (6{,}5\%), r_3 = 0{,}08\ (8\%), \ldots\ (n\ darab\ érték)$
$x \cdot (1 + r_1) \cdot (1 + r_2) \cdot (1 + r_3) \cdot \ldots \cdot (1 + r_n)$

Specifikáció:

$A = (x: \mathbb{R}^+, r: \mathbb{R}^n, n: \mathbb{N}, z: \mathbb{R}^+)$
$Ef = (x = x' \wedge r = r' \wedge n = n' \wedge \forall i \in [1 \ldots n]: r[i]: 0 \leq r[i] \leq 1)$

(tizedes törtben kelljen megadni a kamatot (1-nél kisebb) ne százalékban)

$$Uf = \left(Ef \wedge z = x \cdot \prod_{i=1}^{n} (1 + r[i]) \right)$$

$$\left(Megjegyzés: Ezt\ teszi\ lehetővé\ az\ összegzés\ tétele: \prod_{i=1}^{n} (1 + r[i])\ \ ebben\ a\ példában \right)$$

1.megoldás:

Visszavezetés: Összegzés tételére

$m \ldots n \sim 1 \ldots n$
$s, +, 0 \sim s, *, 1$ → *Megjegyzés: *: szorzás művelet. Szorzás nulleleme/semleges eleme: 1*
$f(i) \sim (1 + r[i])$

Algoritmus:

$s \coloneqq 1$
$i = 1 \ldots n$
$s \coloneqq s * (1 + r[i])$
$z \coloneqq x * s$

2.megoldás:

A szorzás asszociativitása miatt $\left(x(1+r[i])\right)$
Visszavezetés: Összegzés tételére

$m \dots n \sim 1 \dots n$
$s, +, 0 \sim z, *, 1$

$f(i) \sim (1 + r[i])$

Algoritmus:

$z \coloneqq x$
$i = 1 \dots n$
$z \coloneqq z * \ (1 + r[i])$

3.probléma és megoldása:

Az x tartalmazza a betétet (végeredménynél)
Visszavezetés: Összegzés tételére

Utófeltétel módosítása:

$$Uf = \left(n = n' \wedge \ r = r' \wedge \ x = x' \cdot \prod_{i=1}^{n} (1 + r[i]) \right)$$

Visszavezetés: Összegzés tételre

$m \dots n \sim 1 \dots n$
$s, +, 0 \sim x, *, 1$

$f(i) \sim (1 + r[i])$

$x' \cdot 1$ *a szorzás asszociativitása miatt a ciklus elé kerül*

Algoritmus:

~~$x = x'$~~
$i = 1 \dots n$
$x \coloneqq x * \ (1 + r[i])$

Az első lépés az előfeltételünk miatt teljesül, így elhagyható.
A továbbiak a második megoldásból adódnak. Így az algoritmusunk:

$i = 1 \dots n$
$x \coloneqq x * \ (1 + r[i])$

14.5 Fagypont alatt

Adott n nap átlaghőmérséklete. Adott melyik napokon volt fagypont alatt a hőmérséklet. Vállogasd ki!

1.megoldás:

Specifikáció:

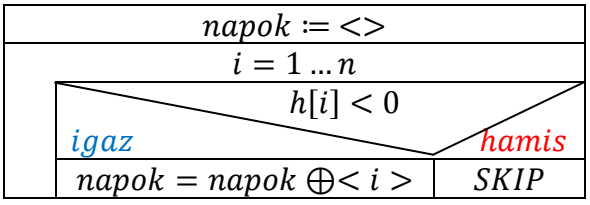
$$A = (h: \mathbb{R}^n, napok: (\mathbb{N}^+)^*)$$
$$Ef = (h = h')$$
$$Uf = \left(Ef \wedge napok = \bigoplus_{\substack{i=1 \\ h[i]<0}}^{n} < i > \right)$$

Visszavezetés: összegzés tételre

$$m \ldots n \sim 1 \ldots n$$
$$s, +, 0 \sim napok, \bigoplus, <>$$
$$f(i) \sim f(x) = \begin{cases} < i > & ha \ \ h[i] < 0 \\ <> & egyébként \end{cases}$$

Megjegyzés: $\oplus$ művelet az összefűzést szokta jelenteni. Semleges eleme/nulleleme a <> .

Algoritmus:



2.megoldás:

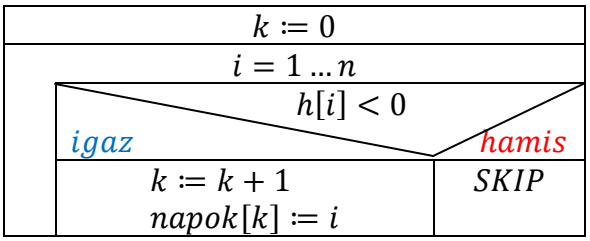
Specifikáció:

$A = (h: \mathbb{R}^n, napok: (\mathbb{N}^+)^k, k: \mathbb{N})$
$Ef = (h = h')$
$Uf = \left(Ef \wedge napok[1 \ldots k] = \bigoplus_{\substack{i=1 \\ h[i]<0}}^{n} < i >\right)$

Visszavezetés: összegzés tételre

$m \ldots n \sim 1 \ldots n$
$s, +, 0 \sim napok, \bigoplus, <>$
$f(i) \sim f(x) = \begin{cases} < i > & ha\ \ h[i] < 0 \\ <> & egyébként \end{cases}$

Algoritmus:



14.6 Páros-e?

Az alábbi feladat megoldások **hatékonyságból rosszak**, így nem ajánlatos ezeket a megoldásokat használni a való életben. Csupán azt szeretnék prezentálni, hogy ezt a feladatot is vissza lehetne vezetni az Összegzés tételre több féleképpen.!

Legyen $g: \mathbb{Z} \to \mathbb{Z}. Adott\ egy\ [a,b]\ intervallum. [a,b] - n\ van - e\ olyan\ hely,$
$ahol\ páros\ g\ értéke?\ \ (g(i)\ páros\ i \in [a,b])$

Utófeltételben szerepelhető megoldási részek (nem a teljes Uf):

(a) $l = \vee_{i=a}^{b}\ 2|g(i)$ Visszavezetésben: $s, +, 0 \sim l, \vee, \downarrow (false)$

(b) $l = 2|\left(\prod_{i=a}^{b} g(i)\right)$

(c) $l = \left(\sum_{i=a}^{b} (g(i) mod\ 2)\right) < n$

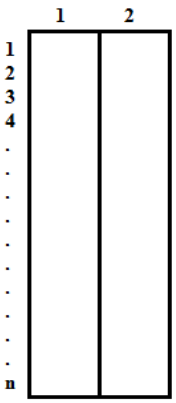
14.7 Koordináta-rendszer

Adott n pont egy koordináta-rendszerben. Hány pont esik az x és hány pont esik az y tengelyre?

Megoldás:

Specifikáció:

$$A = \left(p: \mathbb{R}^{nx2}, c_x: \mathbb{N}, c_y: \mathbb{N}\right)$$
$$Ef = (p = p')$$
$$Uf = \left(Ef \wedge c_x = \sum_{\substack{i=1 \\ p[i][2]=0}}^{n} 1 \wedge c_y = \sum_{\substack{i=1 \\ p[i][1]=0}}^{n} 1 \right)$$



Magyarázat:

1. Hogyan képzeljük el az $n \times 2$ dimenziós tömböt és hogy helyezzük el benne az adatokat:
Jelen esetünkben az 1. oszlop a pontok x koordinátáival, míg a 2. oszlop a pontok y koordinátáival van feltöltve.

2. Utófeltétel sorra: Mikor van az adott pont az x tengelyen, akkor ha az y=0. Az 1.-es pont
alapján ez a mi esetünkben akkor teljesül, ha $p[i][2] = 0$. Mikor van az adott pont az y tengelyen, akkor ha az x=0. Az 1.-es pont alapján ez a mi esetünkben akkor teljesül, ha $p[i][1] = 0$.

A számlálás tétel pedig az adott esetben azt számolja meg, hogy hányszor teljesül a feltétel. Ebből adódóan a mi algoritmusunkban két számlálás lesz. Mivel csak betű paraméterekben és a feltételben aprócska változásokban tér el, így a visszavezetést egyben végeztük el.

Visszavezetés: Számlálás tételre

$m \dots n \sim 1 \dots n$
$c \sim c_x, c_y$
$\beta(i) \sim p[i][2] = 0 \ , p[i][1] = 0$

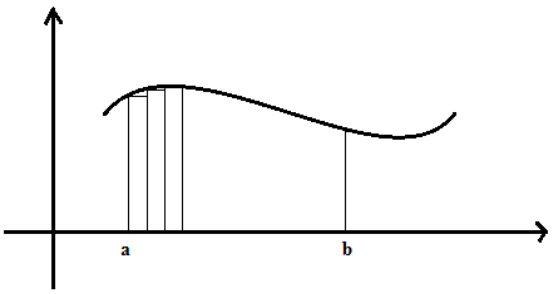
Algoritmus:

Mivel ugyan azt az intervallumot járja be a két megszámlálás és a c-k egymástól függetlenek, így a két megszámlálást egymásba lehet tenni.

$c_x \coloneqq 0, c_y \coloneqq 0$	
$i = 1 \ldots n$	
$p[i][2] = 0$	
igaz	*hamis*
$c_x = c_x + 1$	$SKIP$
$p[i][1] = 0$	
igaz	*hamis*
$c_y = c_y + 1$	$SKIP$

14.8 Integrál

Adott f $[a, b]$-n integrálható. n-részre kell bontani.
Adjuk meg az integrál közelítőértékét!



Specifikáció: n részre bontjuk

$A = (a, b: \mathbb{R}, n: \mathbb{N}^+, t: \mathbb{R}_0^+)$

$Ef = \left(a = a' \wedge b = b' \wedge n = n' \wedge a \leq b\right)$

$$Uf = \left(Ef \wedge x = \frac{b-a}{n} \wedge t = \sum_{i=0}^{n-1} (f(a+ix)x)\right)$$

Visszavezetés: Összegzés tételére

$m \ldots n \sim 0 \ldots n-1$
$s, +, 0 \sim t, +, 0$
$f(i) \sim f(a+ix)x$

Algoritmus:

$t \coloneqq 0$
$i \coloneqq 0 \ldots n-1$
$t \coloneqq t + f(a + i * x) * x$

14.9 Párosok átlaga

Adott $f: \mathbb{Z} \to \mathbb{Z}$. Add meg a páros $f(i)$ értékek átlagát egy $[1, n]$ intervallumon.

Specifikáció:

$A = (x: \mathbb{Z}^n, n: \mathbb{N}, a: \mathbb{R}, l: \mathbb{L})$

$Ef = (x = x' \wedge n = n')$

$$Uf = \left(Ef \wedge s = \sum_{\substack{i=1 \\ 2 \mid x[i]}}^{n} x[i] \wedge c = \sum_{\substack{i=1 \\ 2 \mid x[i]}}^{n} 1 \wedge l = (c \neq 0) \wedge \left(l \to a = \frac{s}{c} \right) \right)$$

Visszavezetés: Összegzés tételre és számlálás tételre (Későbbiekben lehet beágyazással is)

Összegzés:

$m \ldots n \sim 1 \ldots n$

$s, +, 0$

$$f(i) \sim \begin{cases} x[i], & ha\ \ 2 \mid x[i] \\ 0, & egyébként \end{cases}$$

Számlálás:

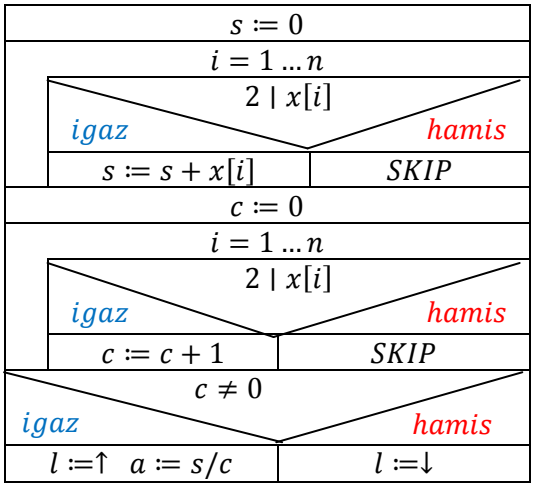
$m \ldots n \sim 1 \ldots n$

c

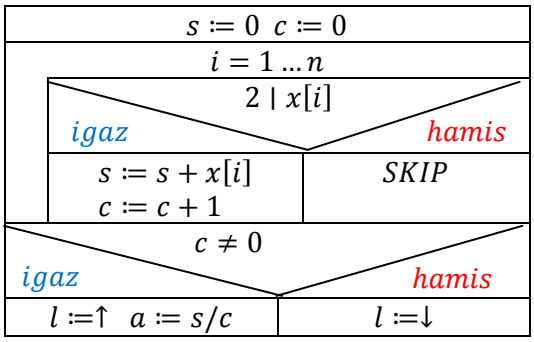
$\beta(i) \sim 2 \mid x[i]$

Algoritmus:

Jelenleg használt



Beágyazásos módszerrel:



14.10 Mire vezethető vissza?

Milyen programozási tételre vezetnéd vissza az alábbi feladatokat?

1. feladat: Adott n egymás utáni nap átlaghőmérséklete. Hányszor csökkent több mint $5°C$-kal az átlaghőmérséklet?

2. feladat: Adott egy n természetes szám és egy $k > 0$ egész. Van-e az $n \ldots n + k$ intervallumban prímszám, ha igen mi az?

3. feladat: Adott egy n egész szám, prímszám-e?

4. feladat: Adott a síkon n pont a koordinátáival, melyik esik a legtávolabb az origótól?

5. feladat: Adott n szó, adjuk meg a leghosszabb $'a'$ betűvel kezdődő szót.

6. feladat: Adott két természetes szám, adjuk meg a legnagyobb közös osztójukat.

7. feladat: Adott egy $\boldsymbol{g}: \mathbb{Z} \to \mathbb{Z}$ függvény, és egy k egész. Tudjuk, hogy $\boldsymbol{g}$ az $[a, b]$ intervallumon monoton növekszik. Felveszi-e $\boldsymbol{g}$ az $[a, b]$ intervallumon a k értéket, ha igen hol?

Megoldások:

1. feladat: Számlálás; 2. feladat: Lineáris keresés (Pesszimista); 3. feladat: Lineáris keresés (Optimista); 4. feladat: Maximum kiválasztás; 5. feladat: Feltételes maximum keresés; 6. feladat: Kiválasztás; 7. feladat: Logaritmikus keresés (vagy kevésbé hatékony Lineáris ker.)

14.11 Van-e prím?

Adott egy n természetes szám és egy $k > 0$ egész. Van-e az $n \dots n+k$ intervallumban prímszám, ha igen mi az?

Specifikáció:

$prim(i) = \uparrow, ha\ i\ prím, \downarrow egyébként$
$A = (k: \mathbb{Z}^*, n: \mathbb{Z}, l: \mathbb{L}, ind: \mathbb{Z}, p: \mathbb{N}^+)$
$\quad ugyan\ az: p: \mathbb{N}^+$
$Ef = (k = k' \wedge n = n')$

$Uf = \left(Ef\ \wedge l, ind = SEARCH_{i=n}^{n+k} prim(i) \wedge l \rightarrow (p = ind)\right)$
$\left(Uf = \left(Ef\ \wedge l, p = SEARCH_{i=n}^{n+k} prim(i)\right)\right)$

Részletes utófeltétel: $Uf = (Ef\ \wedge\ (l = \exists i \in [n \dots n+k]: prim(i) \wedge\ (l \rightarrow p \in [n \dots n+k] \wedge prim(p) \wedge\ \forall i \in [n \dots p-1]:\ \neg prim(i)\ )\ )\ )$

Visszavezetés: Lineáris keresésre (pesszimista eldöntéssel)

$m \dots n \sim n \dots n+k$
$l, ind \sim l, p$
$\beta(i) \sim prim(i)$

Algoritmus:

$l \coloneqq \downarrow\ \ i \coloneqq n$	
$\neg l\ \wedge i \leq n+k$	
	$l \coloneqq prim(i)$
	$p \coloneqq i$
	$i = i+1$

14.12 Prím vagyok?

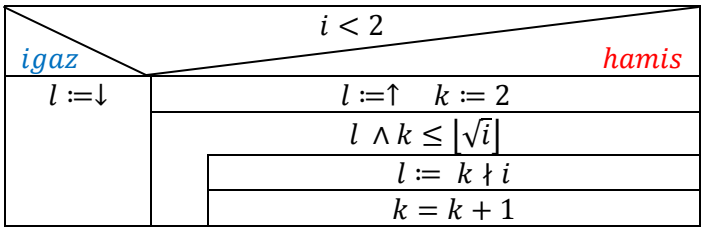
Adott egy n egész szám, prímszám-e?

Specifikáció:

$A = (i: \mathbb{Z}, l: \mathbb{L})$
$Ef = (i = i')$
$Uf = \left(Ef \wedge l = \left((i \geq 2) \wedge \left(\forall k \in [2 \ldots \lfloor\sqrt{i}\rfloor]: k \nmid i\right)\right)\right)$
Utófeltétel másik felírással:
$Uf = \left(Ef \wedge l = \left((i \geq 2) \wedge l'\right) \wedge (l' = \forall SEARCH_{k=m}^{n} k \nmid i)\right)$

Visszavezetés: Lineáris keresés tételre (optimista eldöntéssel)
$m \ldots n \sim 2 \ldots \lfloor\sqrt{i}\rfloor$
$\beta(i) \sim k \nmid i$
$i \sim k$

Algoritmus:



14.13 A legtávolabbi pont

Adott a síkon n pont a koordinátáival, melyik esik a legtávolabb az origótól?

Specifikáció:

$pont = rec(x: \mathbb{R}, y: \mathbb{R})$ - Pont record, ha $p: pont^n$, akkor az alábbi módon tudunk hivatkozni az egyes mezőkre: $p.x$ $p.y$

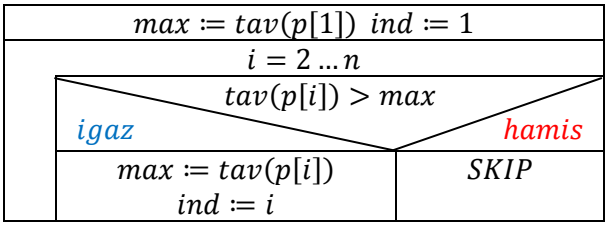
$A = (p: pont^n, n: \mathbb{N}, q: pont)$
$Ef = (n = n' \wedge p = p' \wedge n > 0)$
$tav: pont \to \mathbb{R}_0^+$
$tav(p) = \sqrt{(p.x)^2 + (p.y)^2}$, $ahol\ p \in pont$
$Uf = (Ef \wedge \underline{max, ind = MAX_{i=1}^{n} tav(p[i])} \wedge q = p[ind])$

$$ind \in [1 \ldots n] \wedge max = tav(p[ind]) \wedge \forall i \in [1 \ldots n]: max \geq tav(p[i])$$

Visszavezetés: Maximum kiválasztás (Maximum keresés)

$m \ldots n \sim 1 \ldots n$
max, ind $\qquad\qquad$ $(Ha\ > \sim <, akkor\ minimum\ keresés!)$
$f(i) \sim tav(p[i])$

Algoritmus:



14.14 A leghosszabb a betűvel kezdődő szó

Adott n szó, adjuk meg a leghosszabb $'a'$ betűvel kezdődő szót.

Specifikáció:

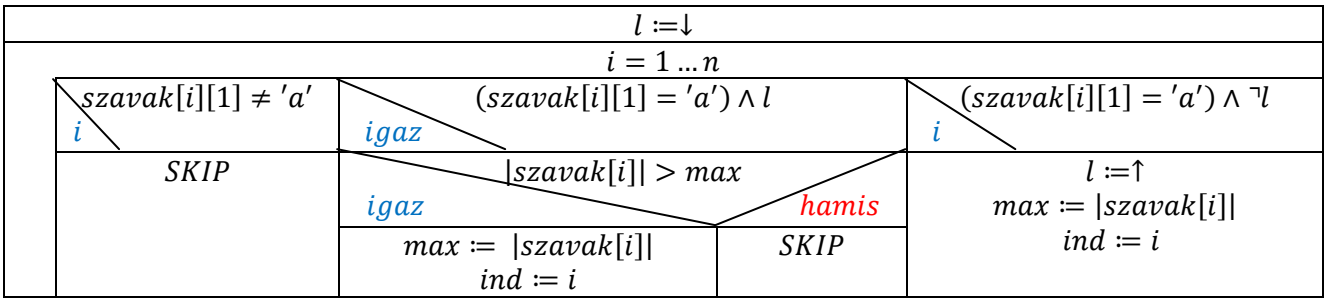
$A = (szavak: (\mathbb{K}^*)^n, n: \mathbb{N}, l: \mathbb{L}, szó: \mathbb{K}^*)$
$Ef = (n = n' \wedge szavak = szavak')$
$Uf = \left(Ef \ \wedge l, max, ind = MAX^{\,n}_{\substack{i=1 \\ szavak[i][1]='a'}} |szavak[i]| \wedge l \rightarrow szó = szavak[ind] \right)$

Feltételben: $szavak[i][1] = 'a'$, $ahol\ az\ 1.[\ \ ]\ a\ szót, míg\ a\ 2.[\ \ ]\ a\ betűt\ határozza\ meg$
Másik felírásban a feltétel lehetne: $pre(szavak[i], 1)$ (prefix)

Visszavezetés: Feltételes maximum keresés tételére

$m \ldots n \sim 1 \ldots n$
max, ind
$\beta(i) \sim szavak[i][1] =' a'$
$f(i) \sim |szavak[i]|$

Algoritmus:



14.15 Mire vezethető vissza? (2)

Milyen tételekre vezethetőek vissza az alábbi feladatok?

1.Feladat: Határozzuk meg egy egész számokat tartalmazó tömb azon legkisebb értékét, amely k-val osztva egyet ad maradékul.

2.Feladat: Adott nap átlaghőmérséklete és csapadékmennyisége. Mennyi azon napok átlaghőmérsékleteinek átlaga, amelyeken volt csapadék.

3.Feladat: Adottak az x és y vektorok, ahol y elemei az x indexei közül valók. Keressünk az x vektornak az y-ban megjelölt elemei között páros számot.

4.Feladat: Igaz-e, hogy egy tömbben elhelyezett szöveg odafelé és visszafelé olvasva is ugyanaz?

5.Feladat: Adott a síkon néhány pont a koordinátáival, és egy körlemez a középpont koordinátáival és a kör sugarával. Adjunk meg egy olyan pontot, amely a körlemezre esik.

6.Feladat: Egymást követő napokon megmértük a déli hőmérsékletet. Hányszor mértünk $0°Celsiust$ úgy, hogy közvetlenül utána fagypont alatti hőmérsékletet regisztráltunk?

7.Feladat: A Föld felsznének egy vonala mentén egyenlő távolságonként megmértük a terep tengerszint feletti magasságát (méterben), és a mért értékeket egy tömbben tároljuk. Melyik a legalacsonyabb hegycsúcs a mérési sorozatban.

8.Feladat: Egy hegyoldal hegycsúcs felé vezető ösvénye mentén egyenlő távolságonként megmértük a terep tengerszint feletti magasságát, és a mért értékeket egy vektorban tároljuk. Megfigyeltük, hogy ezek az értékek egyszer sem csökkentek az előző értékekhez képest. Igaz-e, hogy mindig növekedtek?

9. Feladat: Adott két tömb: egy x és egy b. Fektessük a b-t folyamatosan egymás után, és ezt helyezzük az x tömb mellé. Hány esetben kerül egymás mellé ugyanaz az érték?

10. Feladat: Állítsuk elő egy természetes szám összes valódi osztóját!

Megoldások:

1. Feltételes maximum keresés; 2. Összegzés + Számlálás; 3. Lineáris keresés (pesszimista);
4. Lineáris keresés (Optimista); 5. Lineáris keresés (Pesszimista); 6. Számlálás; 7. Maximum keresés;
8. Lineáris keresés (Optimista); 9. Számlálás; 10. Összegzés ($\oplus$ összefűzés művelettel)

14.16 1 a maradékom

Határozzuk meg egy egész számokat tartalmazó tömb azon legkisebb értékét, amely k-val osztva egyet ad maradékul.

Megoldás:

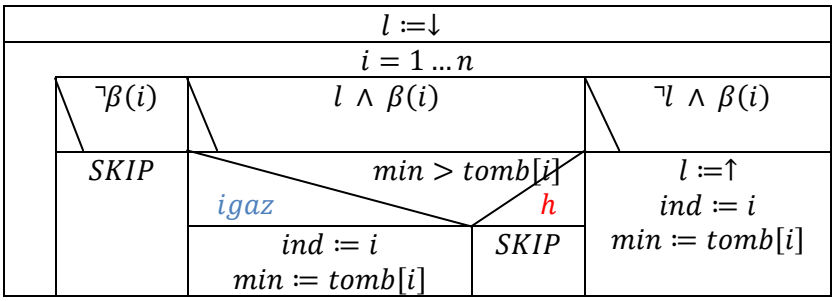
Specifikáció:

$A = (n: \mathbb{N}, l: \mathbb{L}, min: \mathbb{Z}, tomb: \mathbb{Z}^n, k: \mathbb{Z})$
$Ef = (tomb = tomb' \wedge n = n' \wedge k = k' \neq 0)$
$Uf = \left(Ef \wedge (l, min, ind) = MIN^{n}_{\substack{i=1 \\ tomb[i] \equiv 1 \, (mod\ k)}} tomb[i] \right)$

Visszavezetés: Feltételes maximum keresés tételére

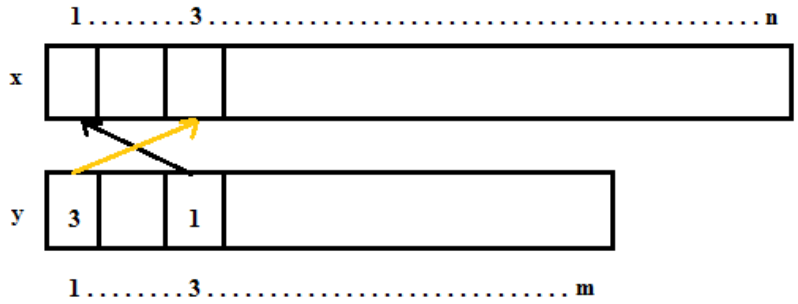
$m \ldots n \sim 1 \ldots n$
$f(i) \sim tomb[i]$
$MAX \sim MIN \quad (reláció\ megfordul)$
$\beta(i) \sim tomb[i] \equiv 1 \, (mod\ k)$

Algoritmus:



14.17 Itt is páros, ott is páros

Adottak az x és y vektorok, ahol y elemei az x indexei közül valók. Keressünk az x vektornak az y-ban megjelölt elemei között páros számot.



Specifikáció:

$A = (x: \mathbb{Z}^n, y: \{1,2, \dots, n\}^m, l: \mathbb{L}, p: \mathbb{Z}, ind: \mathbb{N})$
$Ef = (x = x' \wedge y = y')$
$Uf = \left(Ef \wedge (l, ind) = SEARCH_{i=1}^{m}(2 \mid x[y[i]]) \wedge l \rightarrow p = x[y[ind]]\right)$

Visszavezetés: Lineáris keresés (pesszimista) tételére

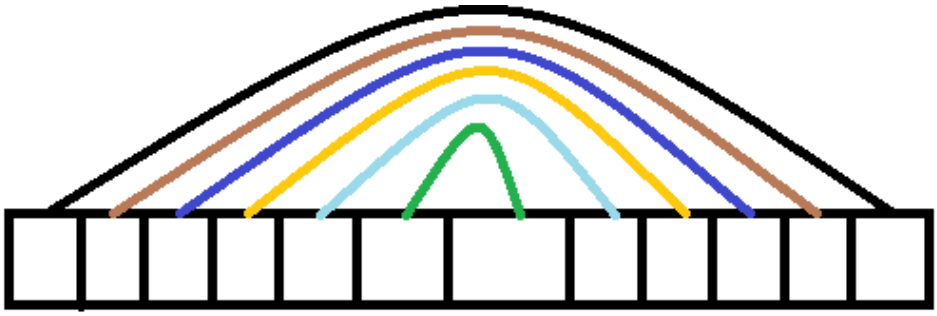
$m \dots n \sim 1 \dots m$
$\beta(i) \sim 2 \mid x[y[i]]$

Algoritmus:

$l \coloneqq \downarrow \quad i \coloneqq 1$	
$\neg l \wedge i \leq m$	
$l \coloneqq 2 \mid x[y[i]]$	
$ind \coloneqq i$	
$i \coloneqq i + 1$	
igaz l *hamis*	
$p \coloneqq x[y[ind]]$	$SKIP$

14.18 Palindrom-e?

Igaz-e, hogy egy tömbben elhelyezett szöveg odafelé és visszafelé olvasva is ugyanaz?



Specifikáció:

$A = (sz\colon \mathbb{K}^n, l\colon \mathbb{L})$
$Ef = (sz = sz')$
$Uf = \left(Ef \ \wedge l = \forall SEARCH_{i=1}^{\lfloor \frac{n}{2} \rfloor}(sz[i] = sz[n-i+1]) \right)$

Visszavezetés: Lineáris keresés (optimista) tételre

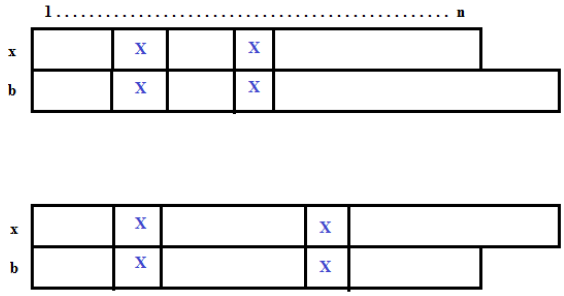
$m \ldots n \sim 1 \ldots \lfloor \frac{n}{2} \rfloor$
$\beta(i) \sim sz[i] = sz[n-i+1]$

Algoritmus:

$l \coloneqq \uparrow \ \ i \coloneqq 1$
$l \ \wedge i \leq \lfloor \frac{n}{2} \rfloor$
$l \coloneqq sz[i] = sz[n-i+1]$
$i \coloneqq i+1$

14.19 Mellettem vagy!

Adott két tömb: egy x és egy b. Fektessük a b-t folyamatosan egymás után, és ezt helyezzük az x tömb mellé. Hány esetben kerül egymás mellé ugyanaz az érték?



Specifikáció:

$A = (x: \mathbb{K}^n, b: \mathbb{K}^m, c: \mathbb{N})$
$Ef = (x = x' \wedge b = b')$

$$Uf = \left(Ef \wedge c = \sum_{x[i]=b\,[((i-1) mod\ m)+1]}^{n} 1 \right)$$

Visszavezetés: Számlálás tételére

$m \ldots n \sim 1 \ldots n$
c
$\beta(i) \sim x[i] = b\,[((i-1) mod\ m) + 1]$

Algoritmus:

$c := 0$	
$i = 1 \ldots n$	
$\boldsymbol{x[i] = b\,[((i-1) mod\ m) + 1]}$ *igaz*	*hamis*
$c := c + 1$	*SKIP*

15. Intervallumos programozási tételek megvalósítása

A lentebbi tételekben levő m,n bemeneti paraméterek, melyeket be kell olvasni az inputról.
Például:

```
#include <iostream>

using namespace std;

int main()
{

      int m,n;
      cout << ”Adja meg az intervallum also erteket: ” << endl;
      cin >> m;
      cout << ”Adja meg az intervallum felso erteket: ” << endl;
      cin >> n;

      /* IDE KERÜL A LENTEBBI TÉTELEK VALAMELYIKE! */

}
```

15.1 Összegzés

```
int s = 0;

for(int i=m; i<=n;++i)
{
      s = s + f(i);

// f(i) lehet pl egy tömb (azt is kell deklarálni!)
}
```

15.2 Számlálás

```
int c = 0;

for(int i=m; i<=n;++i)
{
      if (Betha(i)) // Betha(i) egy feltétel (Boolean értéket ad vissza)
      {
          c = c + 1; // vagy ++c;
      }
}
```

15.3 Maximum kiválasztás

```
int max = f(m); // Itt f() lehet szintén egy tömb, ekkor dekralárni kell!
int ind = m;

for(int i=m+1; i<=n;++i)
{
      if (f(i) > max)
      {
          max = f(i);
          ind = i;
      }
}
```

15.4 Kiválasztás

```
int i;

for(i=m; !Betha(i);++i); // Betha(i) egy feltétel (Boolean értéket ad vissza)
```

15.5 Lineáris keresés

```
int ind = 0;
bool l = false;

for(int i=m; !l && i<=n; ++i)
{
      l = Betha(i);
      ind = i;
}
```

15.6 Feltételes maximum keresés

```
int ind = 0;
bool l = false;
int max = 0;

for (int i=m;i<=n;++i)
  {
        if(!beta(i));
        else if(beta(i) && l)
            {
                if(max<f(i))
                {
                   ind=i;
                   max=f(i);
                }
            }
        else{
            l=true;
            max=f(i);
            ind=i;
        }
  }
```

16. Stringek

16.1 Szöveg

```
#include <iostream>

// A szöveges típus használatához kell a következõ sor:
#include <string>

using namespace std;

int main()
{
    // Karakter típus:

    char c = 'a';

    // Speciális karakterek (sor vége, tabulátor, stb.)

    cout << c << '\n' << '\t' << '\'' << '\\' << endl;

    // A karakter típus valójában egy 1 byte-os szám típus, egy karakter
    // értéke számként a karakter kódja.
    int i = c;
    cout << i << endl;

    // Szöveg típus:

    string s = "Hosszú szöveg";

    // Néhány mûvelet: karakter kiválasztása, összefûzés

    cout << s[5] << ", " << s + " több szóból" << endl;

    return 0;
}
```

16.2 Stringek

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    // A string típus egy összetett típus, a műveleteinek a többsége (de
nem
    // mind) tagfüggvényként érhető el. Minden művelethez kell a <string>
    // fejlécfájl.
    //
    // Nem tagfüggvény művelet: egy sor beolvasása

    string s;
    getline(cin, s);
```

```
    // Operátorok:
    //   Karakter kiválasztás:

    cout << s[2] << endl;

    //   Összefűzés:

    s += "," + s;

    //   Megegyezik-e két szöveg?

    string s2;
    getline(cin, s2);

    if (s == s2)
        cout << "Ugyanaz." << endl;

    //   Melyik szöveg van előbb az ábécé szerinti sorrendben?
    if (s < s2)
        cout << s << " van előbb." << endl;
    else
        cout << s2 << " van előbb." << endl;

    // Fontosabb tagfüggvények:
    //    bool string::empty();
    //    unsigned string::length();
    //    string string::substr(unsigned kezd, unsigned hossz);
    //    unsigned string::find(string s);
    //    unsigned string::rfind(string s);
    //  A kereső függvények úgy működnek, hogy ha nincs találat, akkor a
    //  "string::npos" változó értékét adják vissza.

    if (s.empty())
        cout << "Üres." << endl;

    if (s.length() > 3)
        s = s.substr(0,3);


 // A stringekkel kapcsolatban fontos tudni, hogy a "szöveg" alakú
    // konstansok (string literál a nevük) típusa nem string, hanem
karakterre
    // mutató pointer, a típus pontos neve "const char*". A C++ nyelv
    // "ősében", a C-ben ilyen típusú adatok tárolták a stringeket, és
    // kompatibilitási okokból nagy ritkán még mindig használjuk őket.

    const char* cstr = "alma";

    // Az ilyen típusú adatok automatikusan átalakulnak string típusúvá,
    // amikor arra van szükség, a másik irányú konverziót viszont egy
    // függvényhívással kell elvégezni:
    //   const char* string::c_str();

    s = cstr;
    s += "fa";
    cstr = s.c_str();

    return 0;
}
```

16.3 Stringműveletek

A lentebbi példákban s egy string típusú változó

Művelet	Hogyan használjuk?	Mit csinál/definiál?
s = ” ”	s = ” ”	Üres szöveg
>>	pl.: cin >> s	szóközig vagy sor végéig olvas a változóba a konzolról
getline()	getline(cin,s,’\n’)	olvasás a konzolról ’\n’-ig
getline()	getline(cin,s,’ x’)	olvasás a konzolról ’ x’ jelig
size()	s.size()	s karaktereinek a száma
length()	s.length()	s karaktereinek a száma
+	s + ”valami”	hozzáírás (konkatenáció)
$s[i]$	$s[i]$	s szöveg i. jele (ha i kisebb s karaktereinek számánál)
at()	s.at(i)	s szöveg i. jele objektumos jelöléssel
find()	s.find(mit)	a mit szöveg helye s-ben
substr()	s.substr(tol,ig)	s[tol..ig]
replace()	s.replace(tol,db,mivel)	tol-tól db jelig helyettesíti s-ben a karaktereket mivel
isalpha()	isalpha(s[i])	s[i] betű-e? (’a’…’z’,’A’…’Z’)
isdigit()	isdigit(s[i])	s[i] szám-e? (’0’…’9’)
isupper()	isupper(s[i])	s[i] nagy betű-e? (’A’…’Z’)
islower()	islower(s[i])	s[i] kis betű-e? (’a’…’z’)
tolower()	tolower(s[i])	s[i]-t kis betűvé alakítja
toupper()	toupper(s[i])	s[i]-t nagy betűvé alakítja

17. File műveletek

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
    // A C++ szabványos típusai között fájlok használatához is vannak
    // összetett típusok. Többféle fájlkezelési stílust is támogatnak a
    // mûveleteik, itt a szöveges adatfolyamok használatához szükséges
    // mûveleteket nézzük át.

    // Az adatfolyamok kezelésére két általános típus szolgál: az "istream"
    // és
    // az "ostream" (bemenõ és kimenõ adatfolyam). A "cin" változó például
    // istream, a "cout" változó ostream típusú. Fájlok kezelésére két
    // speciális típus áll rendelkezésre: az "ifstream" és az "ofstream".
    // A mûveleteik többsége közös az elõzõ két típussal, csak van néhány
    // fájl-specifikus mûvelete is, pl. megnyitás és lezárás.

    ifstream input;
    input.open("file.cpp");

    string s;
    getline(input, s);
    cout << "Az elsõ sor: " << s << endl;

    input.close();

    // Mûködik a >> operátor is, amivel például számokat lehet beolvasni:

    input.open("szamok.txt");

    int osszeg = 0;
    int sz;
    input >> sz;

    // A "bool istream::good()" függvény akkor ad vissza igazat, ha a
    // legutóbbi beolvasás sikeres volt
    while (input.good()) {
        osszeg += sz;
        input >> sz;
    }
    input.close();

    cout << osszeg << endl;

    // A beolvasó mûveleteknek van egy sajátossága: eredményként mindig a
    // streamet adják, amirõl az olvasás ment. Másrészt, a stream
    // változóknak
    // van egy olyan mûveletük, aminek a segítségével magát a változót is
    // lehet feltételként használni, tehát
    //
    //   while (input.good()) { ... }
    //
    // helyett írhatjuk azt, hogy
    //
```

```
    //   while (input) { ... }
    //
    // és a kettõ ugyanazt jelenti. Ennek a két lehetõségnek a
    //kombinálásával
    // nagyon elegánsan le lehet írni egy ciklust, ami elõreolvasási
    // technikát
    // használva olvassa végig egy fájl tartalmát:

    ifstream f("szamok.txt");   // ugyanaz a hatása, mint az f.open()-nek
    osszeg = 0;

    // A beolvasó utasítás eredménye az f, ami feltételként értelmezve
    // ugyanazt jelenti, mint az f.good()

    while (f >> sz)
        osszeg += sz;

    cout << osszeg << endl;

    // Végül ha fájlba szeretnénk írni szöveget:

    ofstream of("output.txt");
    of << "Az összeg: " << osszeg << endl;
    of.close();

    return 0;
}
```

18. Összetett feladatok intervallumos programozási tételekkel

18.1 Mátrix

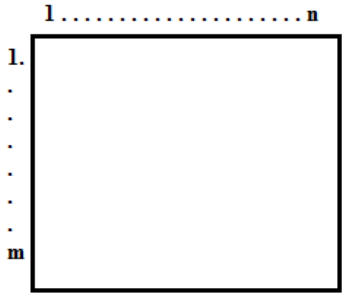
Feladat:

Adott egy $m \times n$-es egészeket tartalmazó mátrix.
(a) Határozzuk meg a legnagyobb sorösszegű sort.
(b) Van-e csupa 0 oszlop?

Megoldás:

(a) **Határozzuk meg a legnagyobb sorösszegű sort.**

- Képzeljünk el egy $m \times n$ -es mátrixot:



  Az első paraméter (m) jelöli a mátrix sorainak a számát, míg a második paraméter (n) a mátrix oszlopainak a számát jelöli.
- A feladatunk szövegéből is látszik, hogy ez már egy olyan feladat, amiben több programozási tétel is szerepel és ezek a tételek egymásba vannak ágyazva.
  Figyeljük meg a feladat szövegét. Emeljük ki belőle a kulcsszavakat: **legnagyobb, sorösszeg**. Mivel a legnagyobb esetén nem szeremel egyéb feltétel, ezért a Maximum keresés tételét kell majd alkalmaznunk, míg a sor**összeg** kulcs szó, az összegzés tételére hívja fel a figyelmet.
- Fontos kiemelni, hogy egymásba ágyazott tétel esetén a belső tételt egy függvénnyel definiáljuk a specifikációban és az algoritmusban is megjelenhet paraméter átadás formályában. Ilyenkor a függvényt külön definiáljuk a specifikáció során, míg a külső tételt tartalmazó algoritmusban felhasználjuk ezt a függvényt egy adott paraméter átadás folyamán vagy akár feltételként is, viszont ekkor oda kell figyelnünk arra, hogy ezt a paraméter átadást vagy feltételt egy külön algoritmusban tisztázni kell. (Olyan, mint ha a programunkban lenne egy belső program.) Ekkor az algoritmust úgy kell látnunk egyben, hogy a Belső tételt tartalmazó algoritmusunk benne szerepel a külső tételt tartalmazó algoritmusunk azon „dobozában”, ahol az adott paraméter átadás vagy feltétel megtalálható.

Specifikáció:

$A = (x: \mathbb{Z}^{m \times n}, max: \mathbb{Z}, ind: \mathbb{N})$
$Ef = (x = x' \wedge m \geq 1 \wedge n \geq 1)$ Megjegyzés: $m \geq 1 \wedge n \geq 1$ elhagyható, mivel $m \times n$-ben alapértelmezett
$Uf = (Ef \wedge max, ind \coloneqq MAX_{i=1}^{m} sorosszeg(i))$

Most az utófeltételben specifikáltuk a külső tételt tartalmazó algoritmusunk, tehát még specifikálnunk kell a belső tételt tartalmazó bevezetett függvényünket:

$sorosszeg: [1 \dots m] \to \mathbb{Z}$
$sorosszeg(i) \coloneqq \sum_{j=1}^{n} x[i,j]$

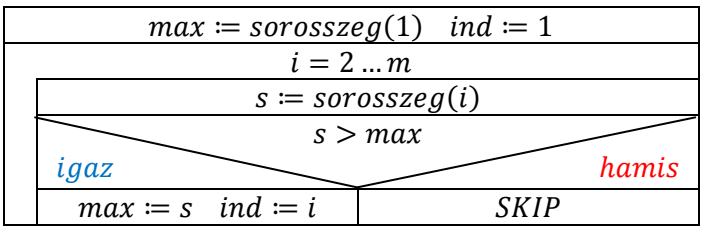
Visszavezetés: Maximum keresésbe ágyazott összegzés

Maximum keresés:
$m \dots n \sim 1 \dots m$
$max, ind, >$
$f(i) \sim sorosszeg(i)$

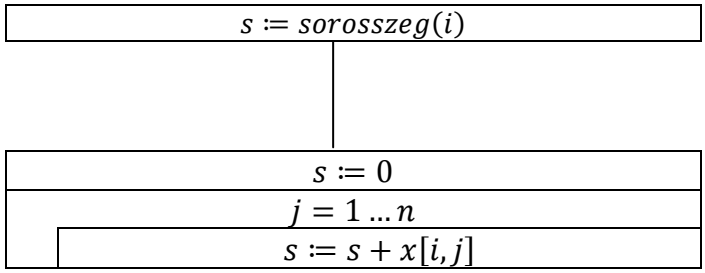
Összegzés:
$m \dots n \sim 1 \dots n$
$i \sim j$
$s, +, 0$

Algoritmus:

Külső tételt tartalmazó algoritmus:



Belső tételt tartalmazó algoritmus:



Itt jól látható, hogy a külső algoritmusban az $s \coloneqq sorosszeg(i)$ dobozka helyére kell betenni, a belső algoritmust.

($\boldsymbol{b}$) Van-e csupa 0 oszlop?

Specifikáció:

$A = (x: \mathbb{Z}^{m \times n}, l: \mathbb{L}, ind: \mathbb{N})$ Megjegyzés: itt most az ind elhagyható
$Ef = (x = x')$
$Uf = \left(Ef \ \wedge l, ind = SEARCH_{j=1}^{n} csupanulla(j)\right)$
$csupanulla: [1 \dots n] \to \mathbb{L}$
$csupanulla(j) \coloneqq \ \forall SEARCH_{i=1}^{m}(x[i,j] = 0)$

Visszavezetés: Lineáris keresésbe ágyazott optimista lineáris keresés

Lineáris keresés:
$m \dots n \sim 1 \dots n$
$i \sim j$
l, ind
$\beta(i) \sim csupanulla(j)$

Optimista lineáris keresés:
$m \dots n \sim 1 \dots m$
l
$\beta(i) \sim x[i,j] = 0$

Algoritmus:

Megjegyzés: Amennyiben kihagyjuk a külső tételből az ind változót, úgy az aloritmusunkban is ki kell törölni azokat a dobozokat, amelyek az ind paramétert tartalmazzák!

Külső tételt tartalmazó algoritmus:

$l \coloneqq \downarrow \;\; j \coloneqq 1$	
$\neg l \wedge j \leq m$	
	$l \coloneqq csupanulla(j)$
	$ind \coloneqq j$
	$j \coloneqq j + 1$

Belső tételt tartalmazó algoritmus:

$l \coloneqq csupanulla(j)$	
$l \coloneqq \uparrow \;\; i \coloneqq 1$	
$l \wedge i \leq m$	
	$l \coloneqq x[i,j] = 0$
	$i \coloneqq i + 1$

Itt jól látható, hogy a külső algoritmusban az $l \coloneqq csupanulla(j)$ dobozka helyére kell betenni, a belső algoritmust.

18.2 Növekvő számtani sorozat

Adottak téglalapok. Igaz-e, hogy a kerületük növekvő számtani sorozatot alkot?

Specifikáció:

$A = \{a\colon (\mathbb{R}^+)^n, b\colon (\mathbb{R}^+)^n, l\colon \mathbb{L}\}$
$Ef = (a = a' \wedge b = b' \wedge n \geq 2)$
$Uf = \left(Ef \wedge d = ker(2) - ker(1) \wedge l = \left(d \geq 0 \wedge \forall SEARCH_{i=2}^{n-1}(ker(i+1) = ker(i) + d)\right)\right)$
$ker(i) = 2 * (a[i] + b[i])$

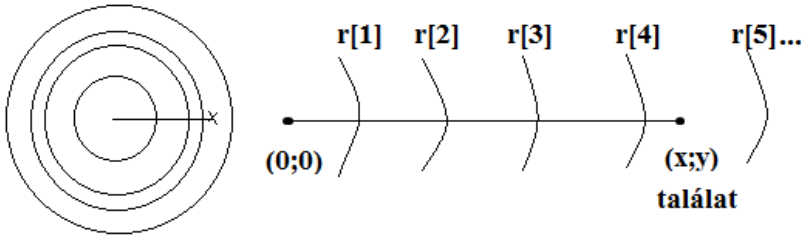
Visszavezetés: Optimista lineáris keresés tételére

$m \ldots n \sim 2 \ldots n - 1$
$\beta(i) \sim ker(i+1) = ker(i) + d$

Algoritmus:

$d \coloneqq ker(2) - ker(1)$	
$l \coloneqq d \geq 0 \;\; i \coloneqq 2$	
$l \wedge i \leq n - 1$	
	$l \coloneqq ker(i+1) = ker(i) + d$
	$i \coloneqq i + 1$

18.3 Céltábla



$A = \{r\colon (\mathbb{R}^+)^n, x\colon \mathbb{R}, y\colon \mathbb{R}, l\colon \mathbb{L}, talalat\colon \mathbb{R}^+\}$

$Ef = \{r = r' \wedge x = x' \wedge y = y' \wedge \forall i \in [1 \dots n-1]\colon r[i] < r[i+1] \wedge n \geq 1\}$

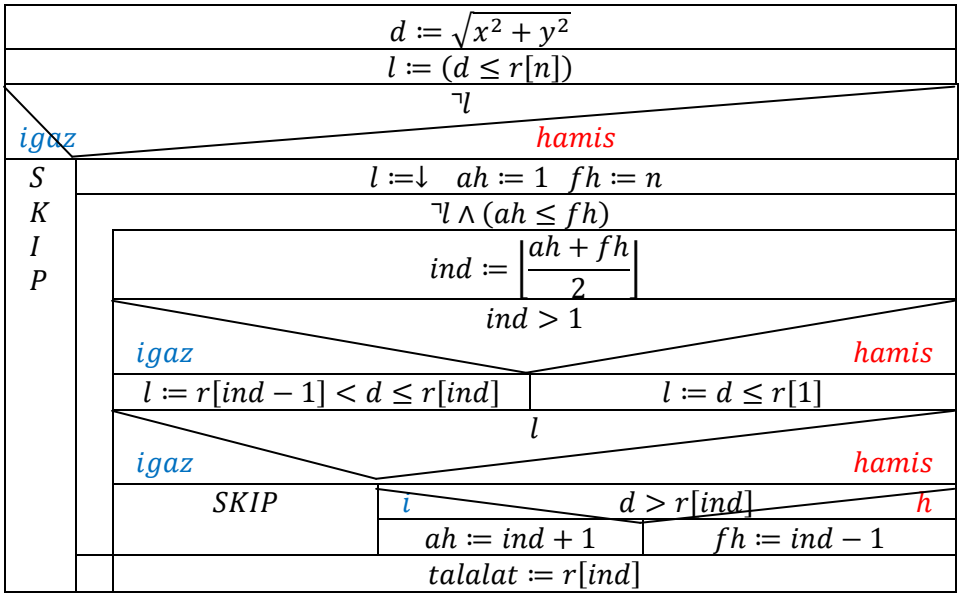
Megoldás: Kiválasztással

$$Uf = \left(Ef \wedge l = \left(\sqrt{x^2+y^2} \leq r[n]\right) \wedge l \rightarrow \left(i = SELECT_{i \geq 1}\left(\sqrt{x^2+y^2} \leq r[i]\right) \wedge talalat = r[i]\right)\right)$$

A fentebbi céltáblás feladat megoldása Logaritmikus kereséssel:

$$Uf = \left(Ef \wedge d = \sqrt{x^2+y^2} \wedge l = (d \leq r[n]) \wedge l \rightarrow (ind \in [1 \dots n] \wedge d \leq r[ind] \wedge \right.$$
$$\left. \wedge \forall i \in [1 \dots ind-1]\colon d > r[i]) \wedge talalat = r[ind]\right)$$

Kihasználjuk, hogy rendezetten vannak megadva a tömb elemei és így visszavezethetjük a logaritmikus keresés tételére a feladatunkat.



18.4 Leggyakoribb szám

Adott egy egészeket tartalmazó tömb. Melyik szám fordul elő a leggyakrabban a tömbben?

Specifikáció:

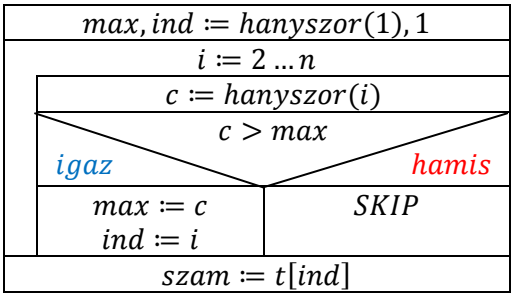
$A = (t: \mathbb{Z}^n, szam: \mathbb{Z})$
$Ef = (t = t' \wedge n \geq 1)$
$Uf = (Ef \wedge max, ind = MAX_{i=1}^{n} hanyszor(i) \wedge szam = t[ind])$

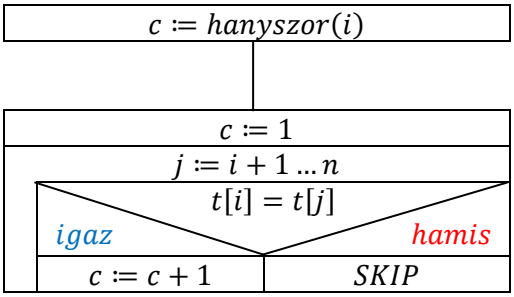
$$hanyszor(i) = 1 + \sum_{\substack{j=i+1 \\ t[i]=t[j]}}^{n} 1$$

Visszavezetés: Maximum keresésbe ágyazott számlálás

Maximum keresés: (külső tétel)
$m \dots n \sim 1 \dots n$
$f(i) \sim hanyszor(i)$

Számlálás: (belső tétel)
$m \dots n \sim i + 1 \dots n$
$i \sim j$
$c, 0 \sim c, 1$
$\beta(i) \sim t[i] = t[j]$

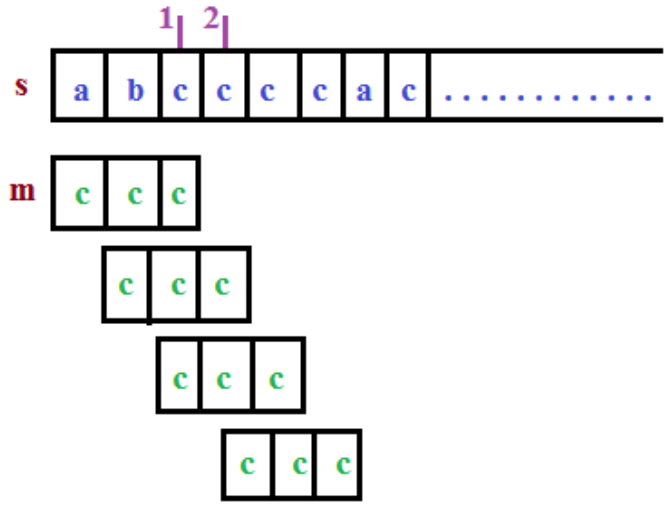




18.5 Brute-force

Adott s szöveg és m minta. Adjuk meg, mely pozíciókra illesztkedik a minta.

Brute-force



Specifikáció:

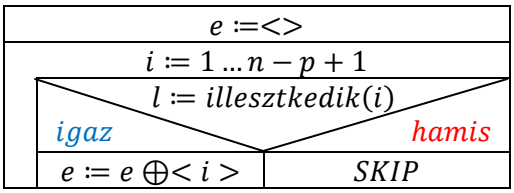
$A = (s: \mathbb{K}^n, m: \mathbb{K}^p, e: \mathbb{N}^*)$
$Ef = (s = s' \wedge m = m' \wedge p \geq 1)\ Megjegyzés: ki\ lehet\ kötni (n \geq 1) - et\ is$
$Uf = \left(Ef \wedge e = \bigoplus_{\substack{i=1 \\ illesztkedik(i)}}^{n-p+1} < i > \right)$
$illesztkedik: [1 \ldots n-p+1] \to \mathbb{L}$
$illesztkedik(i) = \forall SEARCH_{j=1}^{p} s[i+j-1] = m[j]$

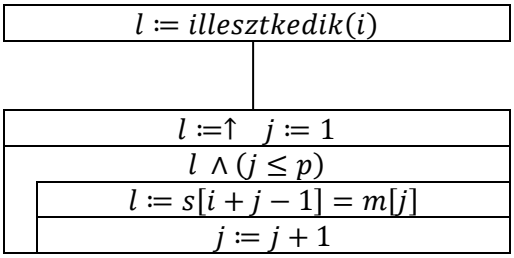
Visszavezetés: Összegzés tételébe ágyazott Optimista lineáris keresés

Összegzés:
$m \ldots n \sim 1 \ldots n-p+1$
$s, +, 0 \sim e, \oplus, <>$
$f(i) \sim \begin{cases} < i >, & ha\ illesztkedik(i) \\ <>, & egyébként \end{cases}$

Optimista lineáris keresés:
$m \ldots n \sim 1 \ldots p$
$i \sim j$
$\beta(i) \sim s[i+j-1] = m[j]$

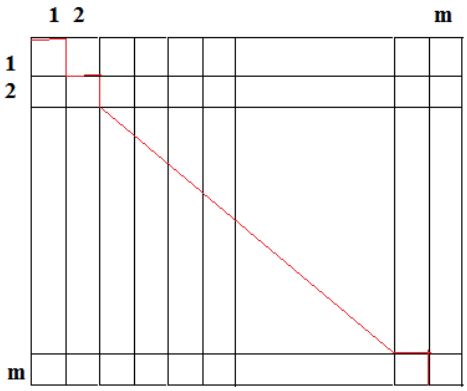
Algoritmus:





18.6 Legközelebbi pont

Adott egy koordinátarendszerben m pont. Melyik kettő van egymáshoz legközelebb?



Specifikáció:

$pont = rec(x: \mathbb{R}, y: \mathbb{R})$
$A = (p: pont^m, ind1: \mathbb{N}, ind2: \mathbb{N})$
$Ef = (p = p' \wedge m \geq 2)$
$Uf = \left(Ef \wedge min, ind1 = MIN_{i=1}^{m-1} legközelebbi(i) \wedge ind2 := legközelebbi(ind1).ind\right)$
$legközelebbi: [1 \ldots m-1] \rightarrow \mathbb{R}_0^+ \times \mathbb{N}$
$legközelebbi(i) = MIN_{j=i+1}^{m} tav(p[i], p[j])$
$tav: pont \times pont \rightarrow \mathbb{R}_0^+$
$p_1, p_2: pont$
$tav(p_1, p_2) = \sqrt{(p_1.x - p_2.x)^2 + (p_1.y - p_2.y)^2}$

Visszavezetés: Minimum keresésbe ágyazott minimum keresés

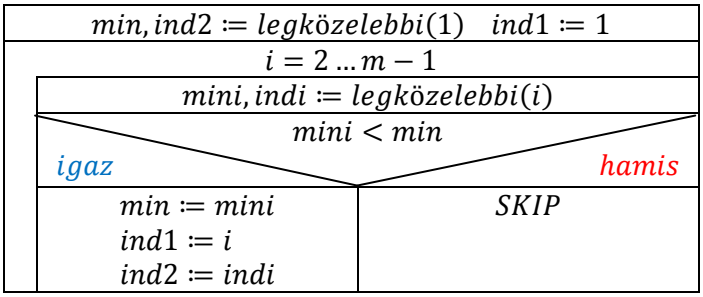
Külső minimum keresés:

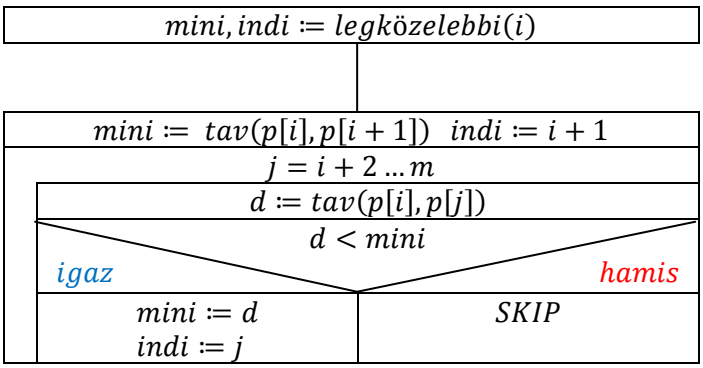
$m \ldots n \sim 1 \ldots m-1$
$f(i) \sim legközelebbi(i).min$
$> \sim <$

Belső minimum keresés:

$m \ldots n \sim i+1 \ldots m$
$i \sim j$
$> \sim <$
$f(i) \sim tav(p[i], p[j])$

Algoritmus:





18.7 Legnagyobb 3-mal osztható páros szám

Adott egy $m \times n$-es egész mátrix, keressünk egy olyan sort, amelyben a legnagyobb páros szám osztható 3-mal.

Specifikáció:

$A = (x: \mathbb{Z}^{m \times n}, l: \mathbb{L}, ind: \mathbb{N})$
$Ef = (x = x')$
$Uf = (Ef \wedge l, ind = SEARCH_{i=1}^{m}(legnagyobbparos(i).l = igaz \wedge$
$\wedge legnagyobbparos(i).\max\ mod\ 3 = 0))$

Uf-et így is fel lehet írni:
$Uf = (Ef \wedge l, ind = SEARCH_{i=1}^{m}(legnagyobbparos(i).l \wedge$
$\wedge legnagyobbparos(i).\max\ mod\ 3 = 0))$

$legnagyobbparos: [1 \dots m] \to \mathbb{L} \times \mathbb{Z} \times \mathbb{N}$
(l, max, ind)
$legnagyobbparos(i) = MAX_{x[i,j]mod\ 2=0}^{n}{}_{j=1}\ x[i,j]$

Visszavezetés: Lineáris keresésbe ágyazott Feltételes maximum keresés

Külső tétel: Lineáris keresés

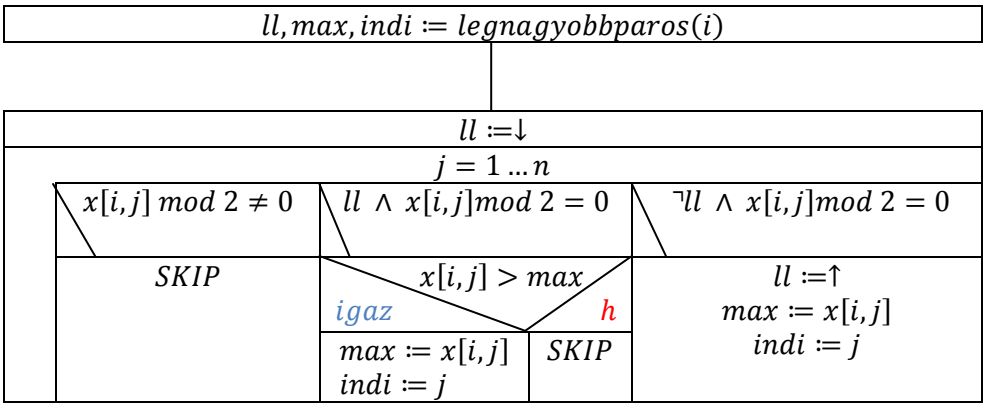
$m \dots n \sim 1 \dots m$
$\beta(i) \sim legnagyobbparos(i).l \wedge\ legnagyobbparos(i).\max\ mod\ 3 = 0$

Belső tétel: Feltételes maximum keresés

$m \dots n \sim 1 \dots n$
$i \sim j$
$\beta(i) \sim x[i,j] mod\ 2 = 0$
$f(i) \sim x[i,j]$

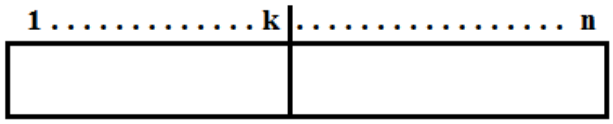
Algoritmus:

$l \coloneqq \downarrow\ \ i \coloneqq 1$	
$\neg l\ \wedge i \leq m$	
	$ll, max, indi \coloneqq legnagyobbparos(i)$
	$l \coloneqq (ll\ \wedge \max\ mod\ 3 = 0)$
	$ind \coloneqq i$
	$i \coloneqq i + 1$



18.8 Mennyire vagyok hatékony?

Adott egy n hosszú ($n \geq 2$) egészekből álló vektor és egy k ($1 \leq k < n$) index. Van-e a vektor $1 \ldots k$ indexű elemei között olyan, amelyik nagyobb az összes $k+1 \ldots n$ indexű elemnél?



Specifikáció:

$A = (t: \mathbb{Z}^n, l: \mathbb{L}, k: \mathbb{N})$
$Ef = \left(t = t' \wedge n \geq 2 \wedge k = k' \wedge (1 \leq k < n)\right)$

Kevésbé hatékony algoritmusok:

$Uf_1 = \left(Ef \wedge l = SEARCH_{i=1}^{k}\left(\forall SEARCH_{j=k+1}^{n} t[i] > t[j]\right)\right)$
- Lineáris keresésbe ágyazott optimista lineáris keresés
- Hatékonysága: $\theta(n^2)$

$Uf_2 = \left(Ef \wedge l = \forall SEARCH_{i=k+1}^{n}\left(SEARCH_{j=1}^{k} t[j] > t[i]\right)\right)$
- Optimista lineáris keresésésbe ágyazott lineáris keresés
- Hatékonysága: $\theta(n^2)$

Hatékonyabb algoritmusok:

$Uf_3 = \left(Ef \wedge max1, ind1 = MAX_{i=1}^{k} t[i] \wedge max2, ind2 = MAX_{i=k+1}^{n} t[i] \wedge l = max1 > max2\right)$
- Maximum keresés és maximum keresés (Nem egymásba ágyazott!)
- Hatékonysága: $\theta(n)$

$Uf_4 = \left(Ef \wedge max, ind = MAX_{i=n}^{1} t[i] \wedge l = ind \leq k\right)$

- Maximum keresés visszafelé
- Hatékonysága: $\theta(n)$

$Uf_5 = \left(Ef \wedge max, ind = MAX_{i=1}^{k} t[i] \wedge l = \forall SEARCH_{i=k+1}^{n}(max > t[i])\right)$

- Maximum keresés és optimista lineáris keresés (Nem egymásba ágyazott!)
- Hatékonysága: $\theta(n)$

$Uf_6 = \left(Ef \wedge max, ind = MAX_{i=k+1}^{n} t[i] \wedge l = SEARCH_{i=1}^{k}(t[i] > max)\right)$

- Maximum keresés és lineáris keresés (Nem egymásba ágyazott!)
- Hatékonysága: $\theta(n)$

19. Feladat intervallumos programozási tételekkel

```
#include <iostream>
#include <vector>
#include <fstream>
// fstream kell nekünk, hogy fájlt tudjunk kezelni

using namespace std;


// továbbfejlesztettük a beolvasás műveletet, hogy ne csak sima cin-ről
olvasson be hanem bármilyen ios beviteli cuccról
void beolvas(vector<int> &myvector, istream &_stream);

// az összegzés helyett produktum
int szorzat(const vector<int> &myvector);

// Ezt a függvényt adjuk majd át a kereséseknek felételként
// eldöntjük, hogy a szám páros-e és logikai értékkel térünk vissza (bool)
bool parose(int a);

//kiválasztás tétele, vektoron végigmegyünk és kiválasztjuk az első béta
feltételnek megfelelő paramétert
int kivalaszt(const vector<int> &_vektor, bool beta (int));

// keresés - az átadott vektoron végigmegyünk és visszatérünk vele, hogy
találtunk-e béta feltételnek megfelelő értéket, ha igen akkor index
változóba írjuk a választ
// index változót tudjuk módosítani, mivel & jel előtte van
bool kereses(const vector<int> &_vektor, bool beta (int), int &index);



int main ()
{
    //létrehozzuk a fájl beolvasására szolgáló változót
    fstream filestream;
    filestream.open ("test.txt");// megnyitjuk a fájlt

    vector<int> myvector;
    int index;

    //int myint;

    beolvas(myvector,filestream); // a módosított beolvas függvénnyel
    // beolvastatjuk a file-ból az adatokat

    //mivel a keresésnek visszatérési értéke logikai, hogy találtunk-e
    //feltételnek megfelelőt, ezért egy if-be nyugodtan berakhatjuk, mivel
    //az if logikai eredményt vár
    //a fetltétel függvényt csak egyszerűen átadjuk (zárójel nélkül, mert
    //ha zárójelet írunk akkor az a függvény meghívását jelentené)
    if ( kereses(myvector,parose,index) )
    {
        //hozzáadunk az index-hez egyet, mert a felhasználó számára nem a
        //0-tól indexelés a logikus
        cout << "elemszam: " << (index+1) << endl;
    }
```

```
    else
    {
        cout << "nincs talalat" << endl;
    }
    cout  << "sorzat: " << szorzat(myvector) << endl;

    return 0;

}


void beolvas(vector<int> &myvector, istream &_stream)
{
    myvector.clear();//kiürítjük a vektort

    int myint;
    do
    {
       _stream >> myint;// az input stream-ről (fstream vagy cin)
                        //beolvasunk egy int-et

       if(myint!=0)//csak akkor adjuk hozzá a vektorhoz, ha nem 0-t írt be
       {
            myvector.push_back (myint);
       }
    }
    while (myint != 0);
}


int szorzat(const vector<int> &myvector)
{
    int ossz;
    ossz=1;//összegzés műveletéhez képest itt 1 az alap érték, mivel 0*a =
           //0 és 1*a=a minden a-ra
    for (int i=0; i<myvector.size(); ++i)
    {
       ossz=ossz*myvector[i];
    }
    return ossz;
}


bool parose(int a)
{
    // % a maradékos osztás, és ha elosztunk egy a számot b-vel és a
    // maradék 0 akkor b osztója a-nak
    return (a%2 == 0);
    /*
    if (a%2 == 0) return true;
    else return false;

    */
}
```

```
int kivalaszt(const vector<int> &_vektor, bool beta (int))
{
    int i = 0;
    // egy ciklust csinálunk, ami addig megy amíg a feltételnek meg nem
    //felel az aktuális elem
    while (!beta(_vektor[i]))
    {
        // a magban csak léptetünk semmi más dolgunk nincs
        i++; //i=i+1;
    }
    return i;
}


bool kereses(const vector<int> &_vektor, bool beta (int), int &index)
{
    int i = 0;
    // szintén ciklussal addig megyünk amíg a feltételnek meg nem felel az
    //aktuális elem
    // csak hozzátesszük, hogy ha a ciklus végére ér akkor álljon le
    while (!beta(_vektor[i]) && i<_vektor.size())
    {
        i++;
    }


    if (i >= _vektor.size())// ha a ciklus azért állt le, mert a végére
              //értünk akkor nem találtunk feltételnek megfelelő elemet
    {
        return false;//tehát hasissal térünk vissza és index-et nem
      //módisítjuk
    }
    else // ha viszont megtaláltuk akkor
    {
        index = i; // index-ben eltároljuk hogy hol találtuk
        return true;// igazzal térünk vissza
    }
}
```

20. Hiba és kivétel kezelés

Célszerű do-while ciklust alkalmazni, abban az esetben ha olyan megoldást szeretnénk nyújtani a felhasználónak, hogy korrigálhassa a hibáját a program futási idején belül.

Ekkor a már korábban említett módon addig zargatja a programunk a felhasználót, amíg számunkra / a program számára nem egy megfelelő adatot visz be a felhasználó.

Másik lehetőségünk a Kivétel kezelés. Ezt a lentebbi módon valósíthatjuk meg a programunkban. Amennyiben ilyet használunk a programunk a hiba bekövetkeztével az adott hibára specifikus üzenet küld a felhasználónak és ezzel egyidőben a program működése leáll.

```
#include <iostream>
#include <vector>

using namespace std;

enum Hibak{NEGATIV_REKORD,NEGATIV_ERTEK};

void beolvas(vector<int> &pl);

int main()
{
    vector<int> T;
    try
    {
        beolvas(T);
    }
    catch(Hibak e)
    {
        switch(e)
        {
            case NEGATIV_REKORD: cout << "A rekordszam negativ!"; break;
            case NEGATIV_ERTEK: cout << "Az egyik ertek negativ!"; break;
        }
    }
    return 0;
}


void beolvas(vector<int> &pl)
{
    int n;
    cin >> n;
    if(n < 0)throw NEGATIV_REKORD;
    pl.resize(n);
    for(int a=0;a<pl.size();++a)
    {
        cin >> pl[a];
        if(pl[a] < 0 || pl[a] > 50)throw NEGATIV_ERTEK;
    }
}
```

Megjegyzés:

Az „enum” olyan adattípust jelöl, melynek lehetséges értékei egy konstanshalmazból kerülnek ki.
A fordító balról jobbra haladva nullával kezdve egész értékeket feleltet meg a felsorolt konstansoknak. Ha egy konstansnak egész értéket adunk, akkor a következő elemek ettől a kiindulási értéktől kapnak értéket.

21. Csomagokra bontás

Többek között a programkód átláthatósága és az újra felhasználhatóság érdekében a programjainkat csomagokra bontjuk. A félév során két féle csomag típust használunk. Az egyik csomagtípus az úgynevezett Header file-ok típusa. Ezekben a fileokban elsősorban függvény és típus deklarációkat tárolunk. Leggyakrabban használt kiterjesztése ezen fileoknak a *.h . A második típusú csomagban pedig a függvény definíciókat tároljuk. Ezen file-ok kiterjesztése pedig *.cpp .

Első lépésként térjünk el a fentebb megszokott programozási módszertől és a függvények deklarációját tároljuk egy *fuggveny.h* fileban, míg a maradék kódot hagyjuk benne a *main.cpp* ben.

Ahhoz, hogy leforduljon a megírt prógramunk figyelmeztetnünk kell a fordításért felelős egységet (linker-t) , hogy hol találja meg a deklarációkat.

Már korábban is használtunk ilyen „figyelmeztetéseket”, csak akkor a számítógépre már valahova feltelepített könyvtárakat rendeltük hozzá a programunkhoz.
Lásd #include <iostream>. Amennyiben < > között szerepel az elérni kívánt file / könyvtár , úgy a gépen bárhol lehet a könyvtár a linker megkeresi és beteszi.
Amennyiben a main.cpp függvénnyel 1 mappába tesszük a header file-ainkat, úgy egy másik módszert használunk az #include ”fuggvenyek.h” kódot. Ekkor a linker a project könyvtárban keresi meg autómatikusan a fileunkat.

Visszatérve a feladatunkhoz annyit kell tennünk, hogy a main. cpp –be felülre beírjuk az #include ”fuggvenyek.h” kódot.

A fuggvenyek.h fileunkban alábbi sorok láthatóak:

```
#ifndef FUGGVENYEK_H_INCLUDED
#define FUGGVENYEK_H_INCLUDED
.
.
.
#endif // FUGGVENYEK_H_INCLUDED
```

Ez a kód segít abban, hogy elkerüljük az összeférhetetlenséget.

Második lépésben bontsuk még tovább a programunkat.
A fuggveny.h Header fileban tároljuk a függvények deklarációját
A fuggvenyek.cpp file-ban a Függvények definícióját
A main.cpp file-ban pedig a főprogramunkat

Innentől kezdve a már megírt fuggveny.h és fuggveny.cpp –t egyszerűen fel tudjuk használni későbbi projectekben is.

22. Struktúrák

A programozás során gyakran találkozhatunk olyan esetekkel, amikor több különböző adatot egy egységként kell kezelnünk, például egy könyvnek van szerzője, címe, kiadója, stb. A C++ nyelvben a struktúra (struct) több különböző típusú adatok együttese:

```
//a Konyv struktúra definíciója
struct Konyv
{
 std::string cim;
 std::string iro;
 int ev;
};
```

A "string" típus karaktersorozatot jelöl, és a standard névtérben helyezkedik el. A struktúrák adattagjaira a pont operátor (.) segítségével hivatkozhatunk és adhatunk nekik értéket:

```
//Konyv struktúra megadása tagonként
Konyv b; //típusként kezelhetjük – Létrehozunk egy Konyv típust, b neven
b.cim = "A C++ programozási nyelv";
b.iro = "Bjarne Stroustrup";
b.ev = 2005;

//Konyv kezdőértékadása
Book c = {"A C++ programozási nyelv", "Bjarne Stroustrup", 2005};

//A c azonos tartalmú b-vel.
```

Amennyiben létrehozunk egy ilyen típust írhatunk olyan függvényeket is, melyeknek visszatérési értéke Konyv lesz.

23. Típus specifikáció

Feladat:

Racionális számok típusának leírása lét egész hányadosaként.

Absztrakt típus
Implementáció

Típus értékek	Típus műveletei
ρ I	Műveletek az implementáció szintjén

ρ - hogyan lesz az adott implementáció kivitelezve
I - invariáns tulajdonság

Absztrakt típus: Típus érték: $\mathbb{Q}$
Típus műveletei: +, -, *, /

Implementáció szintje:
$\rho: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Q}$
$\rho(x_1, x_2) = \frac{x_1}{x_2}$
$I: \mathbb{Z} \times \mathbb{Z} \to \mathbb{L}$
$I\left(\frac{x_1}{x_2}\right) = (x_2 \neq 0)$
Típus műveletei:
$a, b, c \in \mathbb{Q}$
$c = a \overset{+}{-} b$
$c = a * b$
$c = \frac{a}{b}$

Műveletek specfikálása:
$a\ legyen\ (x_1, x_2), b\ legyen\ (y_1, y_2), c\ legyen\ (z_1, z_2)$
$\overset{+}{-}$: $(z_1, z_2) = (x_1 y_2 \overset{+}{-} x_2 y_1, x_2, y_2)$
$*$: $(z_1, z_2) = (x_1 y_1, x_2 y_2)$
$/$: $(z_1, z_2) = (x_1 y_2, x_2 y_1)$

Megjegyzés:
Amennyiben például a deltoid típust szeretnénk leírni, úgy az Absztrakt típus szinten a típus értékhez Deltoid –ot írunk

24. Osztályok

24.1 Bevezető

A C++ az objektum-orientált programozás megvalósításához egyrészt kibővíti a struktúrákat, másrészt bevezeti a class típust. Mindkettő alkalmas osztály definiálására. Egy osztály (class) adattagjának háromféle elérhetősége lehet:

- **public**, nyilvános mindenki számára elérhető
- **private**, privát csak az osztályon belülről lehet elérni, illetve barát osztályok és függvények
- **protected**, védett a származtatott osztályok számára közvetlen elérhetőséget biztosít. A private tagok a leszármazottakból csak az ősosztály tagfüggvényeiből (metódusok) elérhetőek.

A kurzus során csak a public és private elérhetőséget használjuk.

```
//Egy egyszerű osztály
class EgyszeruKonyv
{
 public:
  EgyszeruKonyv (std::string param_cim){ cim = param_cim; }
 private:
  std:string cim;
};
```

24.2 Konstruktorok

Az objektumok kezdeti értékadásaiért (inicializálás) speciális tagfüggvények a konstruktorok felelnek. A konstruktor olyan tagfüggvény amelynek neve megegyezik az osztályéval és nem rendelkezik típussal.

A fordító minden olyan esetben mikor egy objektum létrejön meghívja a konstruktorát. Egy osztálynak bármennyi konstruktora lehet a szignatúrától függően. Alapértelmezés szerint minden osztály két konstruktorral rendelkezik, a paraméter nélküli **(default)** és a másoló **(copy)** konstruktorral. Ha saját konstruktort készítünk, attól fogva az alapértelmezett nem lesz elérhető. A konstruktorok egyaránt lehetnek **public**, **private** vagy **protected** elérésűek. A csak **private** konstruktorokat tartalmazó osztályt *rejtett osztály*nak nevezzük.

Példa default konstruktorra:

```
class Halmaz
{
 public:
  Halmaz (){ms=10; size=0; tomb = new int[ms];}
 private:
  …
};
```

Példa copy konstruktorra:

```
class Halmaz
{
 public:
  Halmaz (const Halmaz& o){…}
 private:
  …
};
```

24.3 Destruktor

Az objektumok által felhasznált memória mindaddig lefoglalt marad, míg fel nem szabadítjuk. Erre a célra a **C++** biztosít számunkra egy speciális tagfüggvényt, a destruktort. Hasonlóan a konstruktorhoz, a destruktornak sincs visszatérési értéke. A destruktornak nem lehetnek paraméterei. A destruktor nevét az osztály nevéből és a hullám karakterből (tilde: ~) képezzük:

```
class Halmaz
{
 public:
  Halmaz (){…} // Konstruktor
  ~Halmaz(){…} // Destruktor
 private:
  …
};
```

Ha nem definiálunk destruktort az osztályunkban, a fordító automatikusan létrehozza. A destruktor minden olyan esetben meghívódik amikor az objektum érvényessége megszűnik. Kivételt képeznek a dinamikusan (a **new** operátorral) létrehozott példányok, amelyeknek csak a megfelelő **delete** operátor hívhatja meg a destruktorát. A destruktor közvetlenül is hívható.

Dinamikus tömbök esetén a konstruktorok az indexek növekvő sorrendjében hívódnak meg, a destruktorok éppen fordítva, de csak a **delete[]** operátor alkalmazásával. A statikus tömbök ugyanígy törlődnek, de automatikusan (tehát nem kell **delete**), amint kikerülnek a hatókörükből. A nem megfelelő **delete** használatával a legjobb esetben is csak a tömb első eleme semmisül meg.

24.4 Operátorok

A C++-ban nem vezethetünk be új operátorokat, de majdnem mindegyiket túlterhelhetjük. Általában meghatározott számú operandusuk lehet (egy, kettő vagy három), kivéve a függvényhívás operátor (operator()), amelynek bármennyi operandusa lehet.
Példa gyakran használt operátorokra: +,-,*,/,&,= stb.

```
class Complex
{
 public:
  …
 private:
  Complex operator+(const Complex& lhs, const Complex& rhs)
{
  return Complex(lhs) += rhs;
}

};
```

24.5 Operátorok túlterhelése

A közönséges függvényekhez hasonlóan a legtöbb operátort is túl lehet terhelni, amely a felhasználói típusok kényelmesebb, szabványosabb használatát teszi lehetővé

Például a kiíró operátor (<<) túlterhelése:

```
class Complex
{
 public:
  friend std::ostream& operator<<(std::ostream& stream, const Complex& z);
 private:
  …
{
  return Complex(lhs) += rhs;
}
};

//az ostream definíciójához nem férünk hozzá, de
//operator<<(ostream&, const complex&)-t definiálhatunk
std::ostream& operator<<(std::ostream& stream, const Complex& z)
{
  return (stream << '(' << z.re << ", " << z.im << ')');
}
```

Ezzel például a közönséges szöveg kiírása mellett egy teljesen megírt Complex class esetén egy komplex számot is rögtön ki tudunk írni:

```
//Ezután az operátort egyszerűen használhatjuk:
Complex c(1.0, 4.6);
std::cout << c; //A kimeneten megjelenik: (1.0, 4.6)
```

24.6 Csomagokra bontás

Amennyiben csomagokban dolgozunk a classokat egy header file-ban deklaráljuk, míg a classokon belül található metódusokat külön cpp fileban.

Ekkor fontos, hogy az adott cpp filehoz hozzá kell rendelnünk a Header filet, továbbá a függvény definíciókban jeleznünk kell, hogy az adott metódusok az osztályhoz tartoznak. Ezt a :: operátorral tudjuk megtenni.

Példa:

halmaz.h

```
#ifndef HALMAZ_H_INCLUDED
#define HALMAZ_H_INCLUDED
…
class Halmaz
{
 public:
  Halmaz();// Konstruktor deklarációja
  ~Halmaz();// Destruktor deklarációja
 private:
  …
};

#endif // HALMAZ_H_INCLUDED
```

halmaz.cpp

```
#include <iostream>
#include ”halmaz.h”

Halmaz::Halmaz() // Konstruktor definíciója
{
…
}
Halmaz::~Halmaz() // Destruktor definíciója
{
…
}
```

25. Felsorolós programozási tételek

25.1 Összegzés

Feladat: Adott egy E-beli elemeket felsoroló t objektum és egy $f: E \to H$ függvény. A H halmazon értelmezzük az összeadás asszociatív, baloldali nullelemes műveletét. Határozzuk meg a függvénynek a t elemeihez rendelt értékeinek összegét! (Üres felsorolás esetén az összeg értéke definíció szerint a nullelem: 0).

Specifikáció:

$A = (t: enor(e), s: H)$
$Ef = (t = t')$
$$Uf = \left(s = \sum_{e \in t'} f(e)\right)$$

Algoritmus:

$s := 0$
$t.First(\ )$
$\neg t.End(\ )$
$s := s + f(t.Current(\ ))$
$t.Next(\ )$

25.2 Számlálás

Feladat: Adott egy E-beli elemeket felsoroló t objektum és egy $\beta: E \to \mathbb{L}$ feltétel. A felsoroló objektum hány elemére teljesül a feltétel?

Specifikáció:

$A = (t: enor(e), c: \mathbb{N})$
$Ef = (t = t')$
$$Uf = \left(c = \sum_{\substack{e \in t' \\ \beta(e)}} 1\right)$$

Algoritmus:

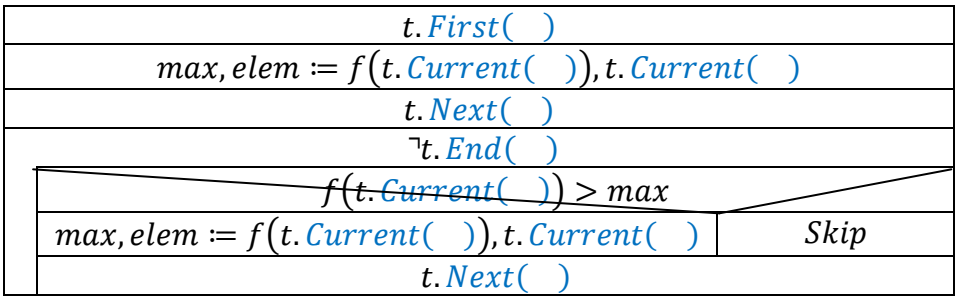
$c := 0$	
$t.First(\ )$	
$\neg t.End(\ )$	
~~$\beta(t.Current(\ ))$~~	
$c := c + 1$	$Skip$
$t.Next(\ )$	

25.3 Maximum kiválasztás

Feladat: Adott egy E-beli elemeket felsoroló t objektum és egy $f: E \to H$ függvény. A H halmazon definiáltunk egy teljes rendezési relációt. Feltesszük, hogy t nem üres. Hol veszi fel az f függvény a t elemein a maximális értékét?
Specifikáció:

$A = (t: enor(e), max: H, elem: E)$
$Ef = (t = t' \wedge |t| > 0)$
$Uf = \big((max, elem) = MAX_{e \in t'} f(e)\big)$

Algoritmus:



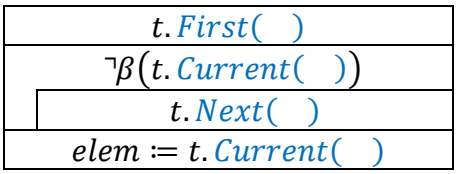
25.4 Kiválasztás

Feladat: Adott egy E-beli elemeket felsoroló t objektum és egy $\beta: E \to \mathbb{L}$ feltétel. Keressük a t bejárása során az első olyan elemi értéket, amely kielégíti a $\beta: E \to \mathbb{L}$ feltételt, ha tudjuk, hogy biztosan van ilyen.

Specifikáció:

$A = (t: enor(e), elem: E)$
$Ef = \big(t = t' \wedge \exists i \in [1..|t|]: \beta(t_i)\big)$
$Uf = \big((elem, t) = SELECT_{elem \in t'} \beta(elem)\big)$

Algoritmus:



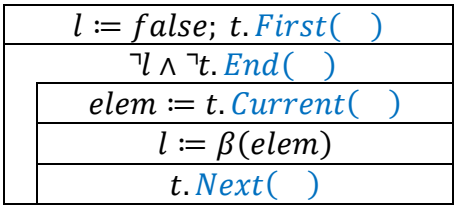
25.5 Lineáris keresés

Feladat: Adott egy E-beli elemeket felsoroló t objektum és egy $\beta: E \to \mathbb{L}$ feltétel. Keressük a t bejárása során az első olyan elemi értéket, amely kielégíti a $\beta: E \to \mathbb{L}$ feltételt.

Specifikáció:

$A = (t: enor(e), l: \mathbb{L}, elem: E)$
$Ef = (t = t')$
$Uf = \left((l, elem, t) = SEARCH_{e \in t', \beta(e)}\right)$

Algoritmus:



25.6 Feltételes maximum keresés

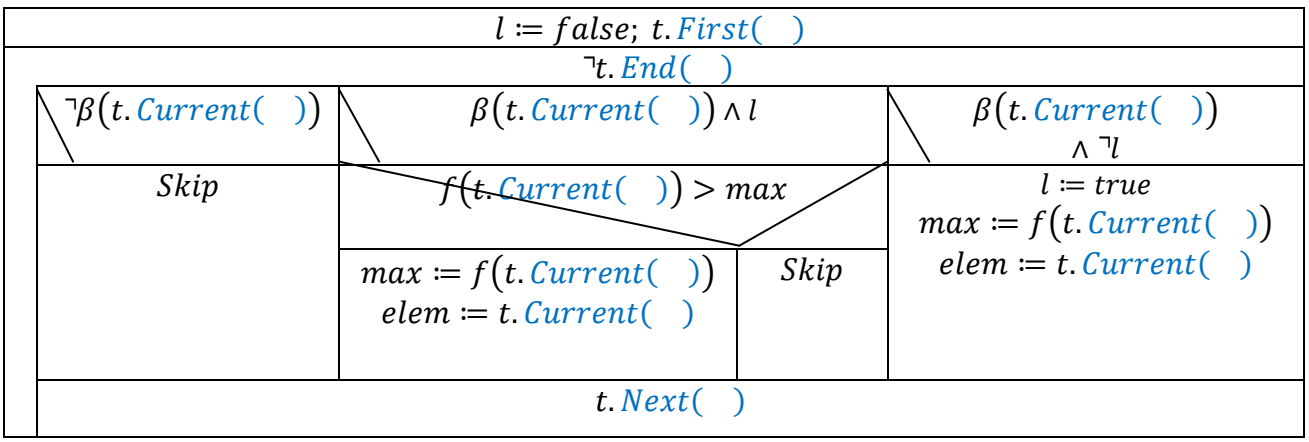
Feladat: Adott egy E-beli elemeket felsoroló t objektum, egy $\beta: E \to \mathbb{L}$ feltétel és egy $f: E \to H$ függvény. A H halmazon definiáltunk egy teljes rendezési relációt.Határozzuk meg t azon elemeihez rendelt f szerinti értékek között a legnagyobbat, amelyek kielégítik a β feltételt.

Specifikáció:

$A = (t: enor(e), l: \mathbb{L}, max: H, elem: E)$
$Ef = (t = t')$

$$Uf = \left((l, max, elem) = MAX_{\substack{e \in t' \\ \beta(e)}} f(e) \right)$$

Algoritmus:



26. Felsorolós programozási tételekre visszavezethető feladatok

26.1 Páros valódi osztók száma

Számoljuk meg egy n természetes szám páros valódi osztóinak számát

$A = (n: \mathbb{N}, t: enor(\mathbb{N}), c: \mathbb{N})$
$Ef = (n = n' \wedge t = t')$
$$Uf = \left(c = \sum_{\substack{i \in t' \\ i|n'}} 1 \right)$$

- Fontos változás az intervallumos programozási tételekhez képest, hogy az utófeltételben nem köthetjük ki, hogy az előfeltétel teljesül
- További fontos változás, ami főleg az algoritmusban fog látszani, hogy felsorolós tételek esetén nem számlálós ciklusunk van, emiatt nem szabad elfelejtenünk tovább léptetni a felsorolónkat.
- Egyedi felsorolót (ami nem nevezetes) a visszavezetésben definiálnunk kell. Egy felsoroló esetén az alábbi „funkciókat" kell megadnunk: t.First(), t.Current(),t.Next(), t.End()

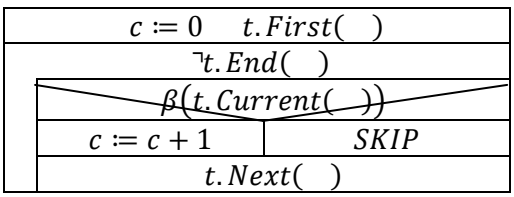
Jelen esetünkben:

t.First() $i \coloneqq 2$
t.Next() $i \coloneqq i + 2$
t.Current() i
t.End() $i > n \; div \; 2$

továbbá, mivel megszámlálás tételről van szó, így $\beta(e) \sim i|n$

Algoritmus:

- Mindig a sablon tételünkre húzzuk rá az aloritmust



$c \coloneqq 0 \quad i \coloneqq 2$	
$i \leq n \; div \; 2$	
$i \mid n$	
$c \coloneqq c + 1$	$SKIP$
$i \coloneqq i + 2$	

26.2 Halmazos

- Halmazokra vonatkozó felsorós feladat

Adott egy egész számokat tartalmazó halmaz, válogassuk ki a páros számokat egy halmazba, a páratlanokat egy vektorba.

Jelölések halmazra: $h: 2^{\mathbb{Z}}$ *vagy* $h: set(\mathbb{Z})$

Egészekből álló halmaz felsorolója:

$t: enor(\mathbb{Z})$

t.First() –
t.Next() $h - mem(h)$ (mem: member)
t.Current() - determinisztikus: $e = mem(h)$ nem determinisztikus (véletlenszerű) : $e \in h$
t.End() $h = \emptyset$

$A = \left(h: set(\mathbb{Z}), x: set(\mathbb{Z}), y: \mathbb{Z}^k\right)$
$Ef = (h = h')$
$Uf = \left(x = \cup_{\substack{e \in h' \\ 2 \mid e}} \{e\} \wedge y[1 \ldots k] = \oplus_{\substack{e \in h' \\ 2 \nmid e}} < e > \right)$

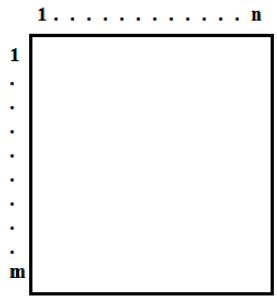
Összegzés tétele

$f(e) \sim e$
$+, 0 \sim \cup, \emptyset$
$+, 0 \sim \oplus, < >$

Algoritmus:

$x \coloneqq \emptyset \; k \coloneqq 0$	
$h \neq \emptyset$	
$e \coloneqq mem(h)$	
$2 \mid e$	
$x \coloneqq x \cup \{e\}$	$k \coloneqq k + 1$ $y[k] \coloneqq e$
$h \coloneqq h - \{e\}$	

26.3 Mátrixos sablon



A felsoroló lehet:

- sorfolytonos (1. sor elemei, 2. sor elemei, …, m. sor elemei)
- oszlopfolytonos (1. oszlop elemei, 2. oszlop elemei, …, n. oszlop elemei)

Sorfolytonos felsoroló:

$t: enor(\mathbb{N} \times \mathbb{N})$
t.First() $(i,j) = (1,1)$
t.Current() (i,j)
t.Next() $\begin{cases} (i,j) \coloneqq (i,j+1) & j < n \\ (i,j) \coloneqq (i+1,j) & j = n \end{cases}$
t.End() $i > m$

$A = (x: \mathbb{R}^{m \times n}, t: enor(\mathbb{N} \times \mathbb{N}), max: \mathbb{R}, elem: \mathbb{N} \times \mathbb{N})$
$Ef = (x = x' \wedge m > 0 \wedge n > 0)$

Nem sablon szerinti Uf és algoritmus:

$Uf = (max, elem = MAX_{e \in t, x'[e]})$

<table>
<tr><td colspan="3">$(i,j) = (1,1)$
$max, elem \coloneqq x[i,j], (i,j)$</td></tr>
<tr><td colspan="3">$j < n$</td></tr>
<tr><td colspan="2">$(i,j) \coloneqq (i,j+1)$</td><td>$(i,j) \coloneqq (i+1,j)$</td></tr>
<tr><td colspan="3">$i \leq m$</td></tr>
<tr><td></td><td colspan="2">$max < x[i,j]$</td></tr>
<tr><td></td><td>$max \coloneqq x[i,j]$
$elem \coloneqq (i,j)$</td><td>$SKIP$</td></tr>
<tr><td></td><td colspan="2">$j < n$</td></tr>
<tr><td></td><td>$(i,j) \coloneqq (i,j+1)$</td><td>$(i,j) \coloneqq (i+1,j)$</td></tr>
</table>

Sablon szerinti Uf és algoritmus:

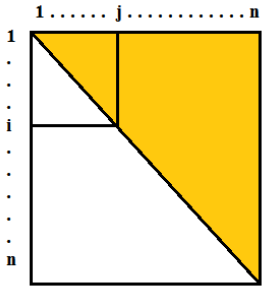
Láthatjuk, hogy ez a megoldás nem annyira hatékony, ezért módosítsuk az algoritmusunk.
Az így kapott eredmény egy sablon lesz. Ez egy olyan kivételes eset, ahol két egymásba ágyazott számlálós ciklust használunk.

Negatívuma: - 1. elemet kétszer vizsgálja meg az algoritmus
Pozitívum: Így is sokkal hatékonyabb, mint a fentebbi ☺

$$Uf = \left(x = x' \wedge max, elem = MAX_{i=1}^{m}\ {}_{j=1}^{n} x[i,j]\right)$$

$max := x[1,1]$ $elem := (1,1)$	
$i := 1 \ldots m$	
$j := 1 \ldots n$	
~~$max < x[i,j]$~~	
$x := x[i,j]$ $elem := (i,j)$	$SKIP$

26.4 Mátrixos feladat



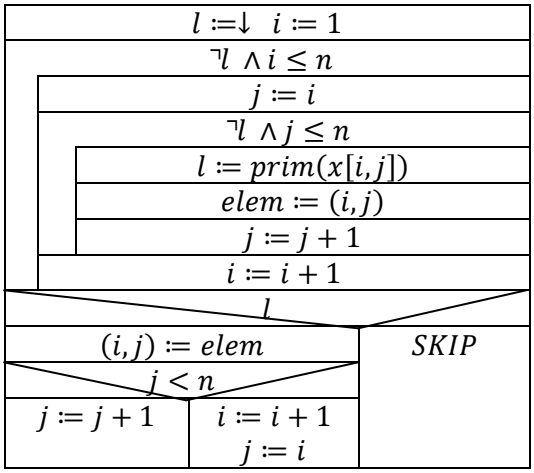
Egy négyzetes mátrix felső hárömszögében van-e prímszám?
(A primszám-ra vonatkozó függvényt nem kell most külön definiálni. Jelölje prim() !)

$$A = (x: \mathbb{Z}^{n \times n}, l: \mathbb{L}, elem: \mathbb{N} \times \mathbb{N})$$
$$Ef = (x = x')$$
$$Uf = \left(l, elem = SEARCH_{i=1}^{n}\ {}_{j=i}^{n} prim(x[i,j])\right)$$

$l :=\downarrow \ \ i := 1$		
$\neg l \wedge i \leq n$		
	$j := i$	
	$\neg l \wedge j \leq n$	
		$l := prim(x[i,j])$
		$elem := (i,j)$
		$j := j + 1$
	$i := i + 1$	

Amennyiben ki szeretnénk használni a felsorolóknak azt a tulajdonságát, hogy a még el nem használt elemekre, folytatni szeretnénk tovább a lineáris keresést, akkor az alábbi módosításokat kell végrehajtani a specifikációban és az algoritmusunkban:

$$Uf = \left(l, elem, (i,j) = SEARCH_{i=1}^{n}{}_{j=i}^{n} prim(x[i,j])\right)$$



27. Szekvenciális inputfile

A szekvenciális inputfile nevezetes felsorolója:

t.First() : sf, df, f:read
t.Next(): sf, df, f:read
t.Current(): df
t.End(): sd = abnorm

Ahol sf a státusza f-nek (norm, abnorm); df a data-ja f-nek; f – file

27.1 Nyilvántartás

Adott egy banki nyilvántartás:
- számlaszám
- egyenleg

egy szekvenciális inputfileban.

Írjuk ki egy output fileba azokat, akik tartoznak, és mennyi az összes tartozás összege?

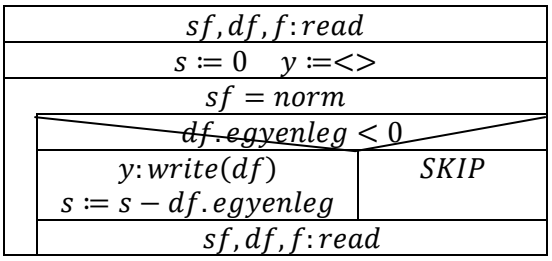
- Két összegzés

$$\text{ügyfél} = rec(számla: \mathbb{K}^*, egyenleg: \mathbb{Z})$$
$$A = \left(f: infile(\text{ügyfél}), s: \mathbb{Z}, y: outfile(\text{ügyfél})\right)$$
$$Ef = (f = f')$$
$$Uf = \left(y = \bigoplus_{\substack{df \in f' \\ df.egyenleg < 0}} < df > \wedge s = \sum_{\substack{df \in f' \\ df.egyenleg < 0}} -df.egyenleg \right)$$

Visszavezetés:

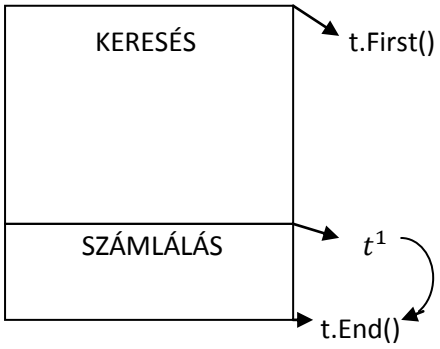
- f nevezetes felsorolóját használjuk
- $+, \emptyset \rightarrow \oplus, <>$ és $+, 0$
- $s \rightarrow y$ és s
- $f(e) \rightarrow \begin{cases} df, ha\ df.egyenleg < 0 \\ <>\ egyébként \end{cases}$ és $\begin{cases} -df.egyenleg,\ ha\ df.egyenleg < 0 \\ 0\ egyébként \end{cases}$

Algoritmus:



27.2 Speciális számlálás

Adott egy szavakat tartalmazó fájl.
Az első 'a'-betűs szó után hány 3 betűs szó van?



Specifikáció:

$A = (f: infile(\mathbb{K}^*), c: \mathbb{N})$
$Ef = (f = f')$

$$Uf = \left(l, sf^1, df^1, f^1 = SEARCH_{df \in f', df[1] =' a'} \wedge l \rightarrow c = \sum_{\substack{df \in f' \\ |df|=3}} 1 \right)$$

felsoroló állapota — most elhagyható

Visszavezetés:

Keresés:

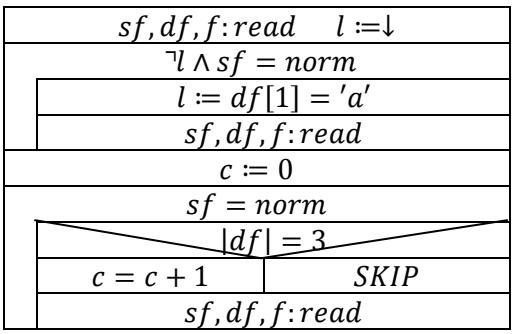
- f nevezetes felsoroló
- $\beta(e) \sim df[1] = 'a'$

Számlálás:

- f nevezetes felsoroló
- $\beta(e) \sim |df| = 3$

Algoritmus:

Keresés: - elem nem kell

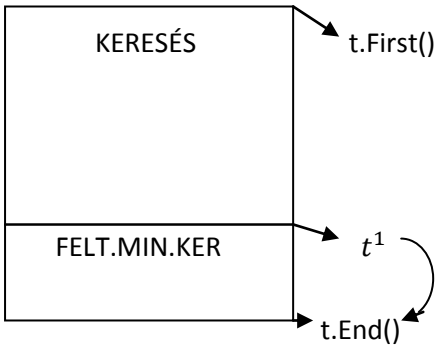


A c:=0 mellől a t.First() kimarad, mert folytatni szeretnénk a felsorolást.

27.3 Tranzakció

Banki tranzkciók:
- ügyfélkód
- időpont
- összeg

A file rendezett ügyfélkód szerint.
Adott ügyfél legkisebb betett összegét szeretnénk megtudni.



A feltételes minimum keresés nem megy végig a fileon!

Specifikáció:

$tranz = rec(\text{ükód}: \mathbb{K}^*, idő: \mathbb{K}^*, \text{összeg}: \mathbb{Z})$
$A = (f: infile(tranz), kód: \mathbb{K}^*, l: \mathbb{L}, min: \mathbb{Z}^+)$
$Ef = (f = f' \wedge kód = kód' \wedge f\ rendezett\ \text{ükód}\ szerint)$
$Uf = (ll, sf^1, df^1, f^1 = SEARCH_{df \in f}, df.\text{ükód} = kód' \quad \wedge$

első olyan rec, ahol az ügyfél kódja megjelenik

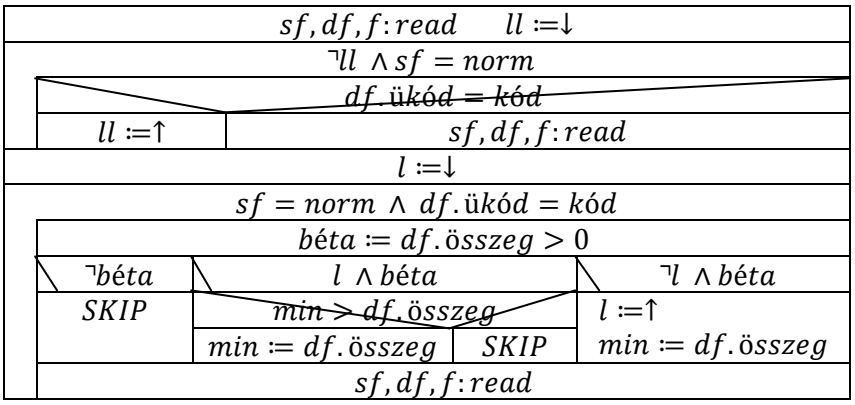
$$ll \rightarrow l, min = MIN^{\substack{df.\text{ükód} = df'.\text{ükód} \\ df \in f' \\ df.\text{összeg} > 0}} df.\text{összeg} \ \wedge \ \neg ll \rightarrow l =\downarrow \Big)$$

Visszavezetés:

Keresés:
- f nevezetes felsorolója
$\beta(e) \sim df.\text{ükód} = kód$

Felt.min.ker:
- f nevezetes felsorolója; $t.End(\quad) \sim t.End(\quad) \vee df.\text{ükód} \neq kód$ módosítással
$\beta(e) \sim df.\text{összeg} > 0$
$f(e) \sim df.\text{összeg}$
$\max \sim min$

Algoritmus:



28. Szekvenciális I/O megvalósítása

nyilvki.h

```
#ifndef NYILVKI_H_INCLUDED
#define NYILVKI_H_INCLUDED

#include "cica.h"
#include <fstream>


class Nyilvki
{
    std::ofstream f;

public:
    Nyilvki(std::string fnev);
    void vrajt(Cica &df);
    ~Nyilvki();
};

#endif // NYILVKI_H_INCLUDED
```

nyilvki.cpp

```
#include "nyilvki.h"

#include <string>

using namespace std;

Nyilvki::Nyilvki(std::string fnev)
{
    f.open(fnev.c_str());
}


void Nyilvki::vrajt(Cica &df)
{
    f << df;
}

Nyilvki::~Nyilvki()
{
    f.close();
}
```

nyilv.h

```
#ifndef NYILV_H_INCLUDED
#define NYILV_H_INCLUDED
#include "cica.h"
#include <fstream>

enum Status{NORM, ABNORM};

class Nyilv
{
    std::ifstream f;

public:
    Nyilv(std::string fnev);
    void reed(Status &sf, Cica &df);
    ~Nyilv();
};
#endif // NYILV_H_INCLUDED
```

nyilv.cpp

```
#include "nyilv.h"
#include <string>

using namespace std;

Nyilv::Nyilv(std::string fnev)
{
    f.open(fnev.c_str());
}


void Nyilv::reed(Status &sf, Cica &df)
{
    f >> df.nev;
    string faja;
    f >> faja;
    if(faja=="SZIAMI")
    {
        df.faj=Cica::SZIAMI;
    }
    else if(faja=="HAZI")
    {
        df.faj=Cica::HAZI;
    }
    else if(faja=="PERZSA")
    {
        df.faj=Cica::PERZSA;
    }
    f >> df.kor;
    f >> df.suly;
    if(f.eof())
    {
        sf=ABNORM;
    }
    else
    {
        sf=NORM;
    }
}

Nyilv::~Nyilv()
{
    f.close();
}
```

main.cpp

```
#include <iostream>
#include <fstream>
#include "nyilv.h"
#include "nyilvki.h"

using namespace std;



int main()
{
    Nyilv macskak("input.txt");
    Nyilvki macski("output.txt");
    Nyilvki macski2("output2.txt");
    Status sf=NORM;
    Cica df;

    while(sf!=ABNORM)
    {
        macskak.reed(sf, df);
        if( df.pontoz() >3)
        {
            macski.vrajt(df);
        }
        else
        {
            macski2.vrajt(df);
        }

    }
    return 0;
}
```

cica.h

```
#ifndef CICA_H_INCLUDED
#define CICA_H_INCLUDED

#include <iostream>


struct Cica
{
    enum Faj{SZIAMI, PERZSA, HAZI};

    std::string nev;
    Faj faj;
    int kor;
    double suly;
    int pontoz();
};

std::ostream& operator<<(std::ostream&, const Cica&);

#endif // CICA_H_INCLUDED
```

cica.cpp

```
#include "cica.h"

using namespace std;

int Cica::pontoz()
{
    return (100-kor)*suly*(faj==HAZI ? 0.6 : 1.1)/100 ;
}

ostream& operator<<(ostream& o, const Cica& c)
{
    o << c.nev << ' ';

    if(c.faj==Cica::SZIAMI)
    {
         o << "SZIAMI";
    }
    else if(c.faj==Cica::HAZI)
    {
         o << "HAZI";
    }
    else if(c.faj==Cica::PERZSA)
    {
         o << "PERZSA";
    }
    o << ' ' << c.kor << ' ' << c.suly << '\n';
}
```

TESZT

input.txt

Miajuu SZIAMI 20 10
Ciccmiricc HAZI 1 2
Tesztmacska PERZSA 5 110
Szemi HAZI 8 11
Megegy PERZSA 73 10

output.txt

Miajuu SZIAMI 20 10
Tesztmacska PERZSA 5 110
Szemi HAZI 8 11

output2.txt

Ciccmiricc HAZI 1 2
Megegy PERZSA 73 10

29. Egyedi felsorolók

29.1 Lakások

Adottak lakások adatai:

- kerület
- azonosító
- ára
- terület
- tulajdonos

Továbbá tudjuk, hogy a file rendezett kerület, azon belül lakás azonosító szerint.
Melyik kerületben legdrágább az átlagos négyzetméter ár?

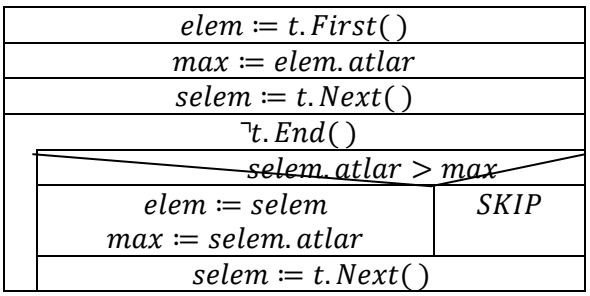
$lakás = rec(ker: \mathbb{N}, az: \mathbb{K}^*, ar: \mathbb{N}, ter: \mathbb{N}, tul: \mathbb{K}^*)$
$A = (f: infile(lakás), ker: \mathbb{N}, max: \mathbb{N})$
Mivel ezen az állapottéren nem tudjuk megoldani a feladatot, ezért áttérünk egy új állapottére, amire rá tudjuk húzni az egyik programozási tétel sablonunkat. (Állapottér átalakításos feladat)

$A = (t: enor(keratl), ker: \mathbb{N}, max: \mathbb{N})$
$keratl = rec(ker: \mathbb{N}, atlar: \mathbb{N})$
$Ef = (t = t' \wedge |t| > 0)$
$Uf = (max, elem = MAX_{e \in t'} e.atlar \wedge ker = elem.ker)$

Írjuk fel a feladat megoldásához szükséges Maximum keresés felsorolókra vonatkozó algoritmusát, a sablon alapján.

Most bevezetünk egy selem: keratl típusú segédváltozót.

Miután felírtuk az algoritmusunk, külön kell specifikálni a t.First(),t.Next(),t.End() és t.Current() metódusokat.



A t.First(),t.Next(),t.End() és t.Current() metódusok gyakran belső tételek. Mivel sok esetben nehezebb őket specifikálni, ezért először az algoritmusukat rajzoljuk fel, majd csak utána specifikáljuk.

t.First( ):

Megjegyzés: Maximum keresés tétele miatt f nem üres, ezért sf-et nem fogjuk ellenőrizni.
Továbbá legyen : elem – keratl típusú változó.

$sf, df, f: read$
$s \coloneqq 0 \;\; c \coloneqq 0 \;\; elem.ker = df.ker$
$sf = norm \;\wedge df.ker = elem.ker$
$s \coloneqq s + \frac{df.ar}{df.ter}$ $c \coloneqq c + 1$
$sf, df, f: read$
$elem.atlar \coloneqq \frac{s}{c}$

Specifikáció:
$A_{t.First()} = (f: infile(lakás), elem: keratl)$
$Ef = (f = f' \wedge |f| > 0 \wedge f \nearrow ker \uparrow az)$ Megjegyzés: $\nearrow ker \uparrow az$ monoton növekvő ker szerint, ezen belül szig. mon nő azonosító szerint.

$$Uf = \left(sf^1, df^1, f^1: read \wedge elem.ker = df^1.ker \wedge sf, df, f, s = \sum_{df \in f^1}^{df.ker = elem.ker} \left(\frac{df.ar}{df.ter}\right) \wedge \right.$$
$$\left. \wedge\, c = \sum_{df \in f^1}^{df.ker = elem.ker} 1 \;\wedge elem.atlar = \frac{s}{c} \right)$$

t.Next( ):

$sf = norm$	
$t.First()$ algo. első sora nélkül!	$l \coloneqq \downarrow$
$l \coloneqq \uparrow$	

Specifikáció:

$A_{t.Next()} = (f: infile(lakás), elem: keratl, l: \mathbb{L})$
$Ef = (f = f' \wedge f \nearrow ker \uparrow az)$
$Uf = (l = (sf' = norm) \wedge l \rightarrow elem.ker = df'.ker \wedge\; sf^1, df^1, f^1: read \wedge$
$$\wedge\; sf, df, f, s = \sum_{df \in f^1}^{df.ker = elem.ker} \left(\frac{df.ar}{df.ter}\right) \wedge c = \sum_{df \in f^1}^{df.ker = elem.ker} 1 \;\wedge elem.atlar = \frac{s}{c} \Bigg)$$

t.End( ): $l = \downarrow$

29.2 W

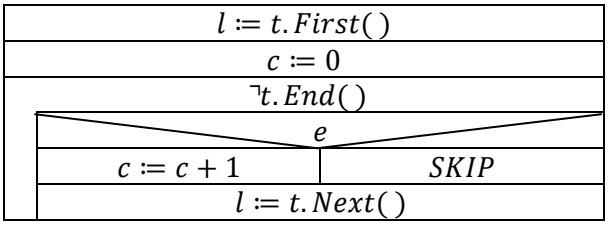
Karakterekből álló szekvenciális inputfileban hány 'w' –betűt tartalmazó szó van?

Specifikáció:

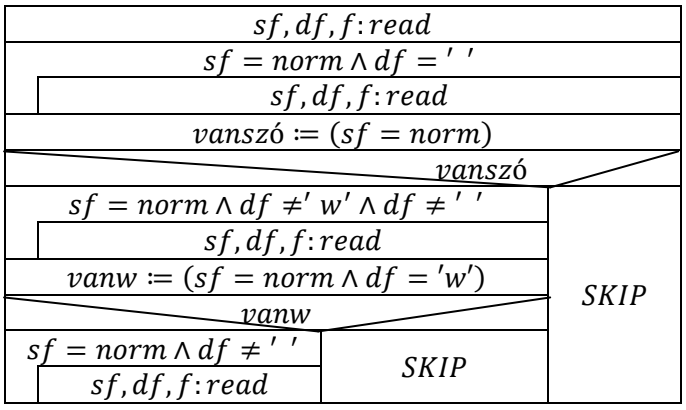
$A = \left(f\colon infile(\mathbb{K})\right)\ ?\ a\ feladat\ ezen\ az\ állapot\ téren\ nem\ megoldható$
Új állapot tér:

$A = (t\colon enor(\mathbb{L}), c\colon \mathbb{N})$
$Ef = (t = t')$
$$Uf = \left(c = \sum_{\substack{e \in t' \\ e}} 1\right)$$

Algoritmus:



t.First() = t.Next( ) Algoritmusa és Specifikációja



Specifikáció:

$A_{t.First(\,),t.Next(\,)} = (f\colon infile(\mathbb{K}), vanszó\colon \mathbb{L}, vanw\colon \mathbb{L})$
$Ef = (f = f')$
$Uf = \left(sf^1, df^1, f^1 = SELECT_{df \in f'} sf = abnorm \vee df \neq' '\wedge vanszó = (sf' = norm)\ \wedge\right.$
$\wedge vanszó \rightarrow vanw, sf^2, df^2, f^2 = SEARCH^{df=' '}_{df \in f^1} df = 'w'\ \wedge$

$\left.\wedge vanw \rightarrow sf, df, f = SELECT_{df \in f^2} sf = abnorm \vee df \neq' '\right)$

29.3 Programozó verseny

Programozási verseny:

Adott a fileban:
- csapat azonosító
- feladat sorszám
- beküldési idő (hány perc telt el a kezdettől)

Csapat azonosító szerint rendezett a file.

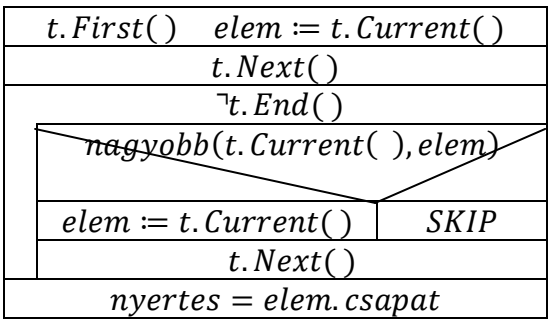
Ki nyerte a versenyt?
Nyerés:
- Az a csapat, aki a legtöbb feladatot megoldotta
- Ha két csapat ugyan annyi feladatot oldott meg, akkor a nyertes aki hamarabb oldotta meg

Specifikáció:

Eredeti állapot tér:
$verseny = rec(csapat: \mathbb{K}^*, feladat: \mathbb{N}^+, idő: \mathbb{N}^+)$
$A = (f: infile(verseny), nyertes: \mathbb{K}^*)$
Ezen az állapot téren nem tudjuk megoldani a feladatot, így átalakítjuk:

$eredmény = rec(csapat: \mathbb{K}^*, db: \mathbb{N}^+, fidő: \mathbb{N}^+)$
$A = (t: enor(eredmény), nyertes: \mathbb{K}^*)$
$Ef = (t = t' \wedge |t| > 0)$
$Uf = (nyertes' = MAX_{e \in t'} e \ \wedge nyertes = nyertes'.csapat)$

Maximum keresés



elem:eredmény típusú változó.

t. First( ): 1. csapat feldolgozása

$sf, df, f: read$	
$e.csapat \coloneqq df.csapat$ $e.db \coloneqq 1$ $e.fido \coloneqq df.ido$	
$sf, df, f: read$	
$sf = norm \wedge df.csapat = e.csapat$	
$e.db = e.db + 1$	
~~$df.ido \geq e.fido$~~	
$e.fido \coloneqq df.ido$	$SKIP$
$sf, df, f: read$	

e: eredmény típusú

$A_{t.First()} = (f: infile(verseny), e: eredmény, sf: status, df: verseny)$
sf,df a felsoroló folytatása
$Ef = (f = f' \wedge |f'| > 0 \wedge f \nearrow csapat)$
$Uf = (sf^1, df^1, f^1: read \wedge e.csapat = df^1.csapat \wedge$

$$e.db, sf^2, df^2, f^2 = \sum_{df \in f'}^{df.csapat=e.csapat} 1 \wedge$$

$$\wedge\ e.fido, sf^2, df^2, f^2 = MAX_{df \in f'}^{df.csapat=e.csapat} df.ido)$$

t. Next( ):

~~$sf = abnorm$~~	
$vége \coloneqq \uparrow$	$e.csapat \coloneqq df.csapat$ $e.db \coloneqq 1$ $e.fido \coloneqq df.ido$
	. . lásd $t.First()$

Kevés idő miatt elfogadható ez a felírási mód a ZH-n

$A_{t.next()} = (f: infile(verseny), df: verseny, sf: status, e: eredmény)$
$Ef = (f = f' \wedge df = df' \wedge sf = sf' \wedge f \nearrow csapat)$
$Uf = (vége = (sf' = abnorm) \wedge \neg vége \rightarrow e.csapat = df'.csapat \wedge \ldots t.First())$
Itt is lehet ezt a felírást használni!

t. End( ):$(vége = \uparrow)$
t. Current( ):elem egy példánya

Ami még kell: a nagyobb függvény elkészítése!!!

30. Összefuttató felsorolók

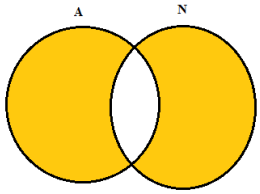
- Adott két rendezett sorozat, elő kell állítani a kimenetet

Az alábbi minta feladatokon keresztül példákat láthatunk a metszet, a szimmetrikus differenciál és a különbség halmazműveletekre vonatkozó feladatokra egy-egy egyszerűbb példát, amilyen a tavalyi zárthelyi dolgozatban is találkozhattunk, továbbá egy nehezebb feladatot (record-os) az unió műveletre.

30.1 Szimmetrikus differencia

Bemenet: két rendezett sorozat
- első file tartalmazza az angolul tudó hallgatók neveit
- második file tartalmazza a németül tudó hallgatók neveit

Kik azok a hallgatók, akik pontosan egy nyelvet beszélnek?
(Szimmetrikus differencia)



Specifikáció:

$A = \left(a: infile(\mathbb{K}^*), n: infile(\mathbb{K}^*), z: outfile(\mathbb{K}^*)\right)$
$Ef = (a = a' \wedge n = n' \wedge a \uparrow \wedge n \uparrow)$
$Uf = \left(z = \bigoplus_{e \in a' \cup n'} f(e)\right)$

$$f(e) = \begin{cases} < e > & e \in a \wedge e \notin n \\ < e > & e \notin a \wedge e \in n \\ < \ > & e \in a \wedge e \in n \end{cases}$$

$Megjegyzés: f(e)$
$-$ ö$sszefuttató\ felsoroló\ (a\ mi\ eset$ünk$ben\ most\ a\ szimmetrikus\ differenciára)$

<table>
<tr><td colspan="4">$z :=<>$ $sa, da, a: read$ $sn, dn, n: read$</td></tr>
<tr><td colspan="4">$sa = norm$ ∨ $sn = norm$</td></tr>
<tr><td></td><td>$(sn = abnorm)$ ∨
$(sa = norm \wedge da < dn)$</td><td>$sa = norm \wedge sn = norm$ ∧
$da = dn$</td><td>$(sa = abnorm)$ ∨
$(sn = norm \wedge da > dn)$</td></tr>
<tr><td></td><td>$z: write(da)$
$sa, da, a: read$</td><td>$sa, da, a: read$
$sn, dn, n: read$</td><td>$z: write(dn)$
$sn, dn, n: read$</td></tr>
</table>

Magyarázatok a pirossal kiemelt részekről:

∨: Általában vagyot használunk, mert mind a két fileunkat teljesen fel szeretnénk dolgozni. Abban az esetben, ha metszetet vagy különbséget kérdeznek a feladatban, akkor jobb az ∧-t használni, mivel ezekben az esetekben ügyesebb és hatékonyabb nem feldolgozni a két fileunkat.

$da < dn$:
$(sn = abnorm)$ ∨
$(sa = norm \wedge da < dn)$
Abban az esetben, ha elfogyott a németesek nevét tartalmazó fileunk vagy van még az angol neveket tartalmazó file-ban valami adatunk és névsor szerint előrébb található az adatban foglalt név, a németeseket tartalmazó file-unkhoz képest, akkor lépünk be ebbe az ágba.
Itt da-t dolgozzuk fel és csak az a fileból olvasunk.

$da = dn$:
$sa = norm \wedge sn = norm$ ∧
$da = dn$

Mind két fileból feldolgozunk, ezért szükséges, hogy mind 2 file tartalmazzon adatot és ha olyan állapotba érünk, hogy az angolos és németes fileban is megtaláljuk ugyan annak a tanulónak a nevét, akkor lépünk be ebbe az elágazásba.
Ekkor a feladat miatt (szimmetrikus differencia) a metszet nem számít, így nem iratjuk ki a hallgatót, csak mind két fileból beolvassuk a következő nevet.

da > dn:

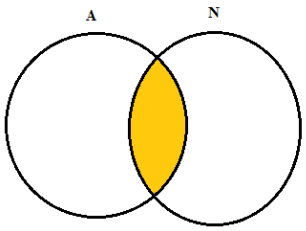
(sa = abnorm) ∨
(sn=norm ∧ da > dn)

Abban az esetben, ha elfogyott az angolosok nevét tartalmazó fileunk vagy van még a német neveket tartalmazó file-ban valami adatunk és névsor szerint előrébb található az adatban foglalt név, az angolosok nevét tartalmazó file-unkhoz képest, akkor lépünk be ebbe az ágba.
Itt dn-t dolgozzuk fel és csak az n fileból olvasunk.

30.2 Metszet

Bemenet: két rendezett sorozat
- első file tartalmazza az angolul tudó hallgatók neveit
- második file tartalmazza a németül tudó hallgatók neveit

Kik azok, akik két nyelven beszélnek?
(Metszet)



Specifikáció:

$A = \left(a: infile(\mathbb{K}^*), n: infile(\mathbb{K}^*), z: outfile(\mathbb{K}^*)\right)$

$Ef = (a = a' \wedge n = n' \wedge a \uparrow \wedge n \uparrow)$

$Uf = \left(z = \bigoplus_{e \in a' \cup n'} f(e)\right)$

$$f(e) = \begin{cases} < > & e \in a \wedge e \notin n \\ < > & e \notin a \wedge e \in n \\ < e > & e \in a \wedge e \in n \end{cases}$$

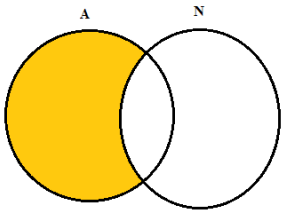
Algoritmus:

<table>
<tr><td colspan="4">$z := <> \quad sa, da, a: read \quad sn, dn, n: read$</td></tr>
<tr><td colspan="4">$sa = norm \ \wedge \ sn = norm$</td></tr>
<tr><td></td><td>$da < dn$</td><td>$da = dn$</td><td>$da > dn$</td></tr>
<tr><td></td><td>$sa, da, a: read$</td><td>$z: write(da)$
$sa, da, a: read$
$sn, dn, n: read$</td><td>$sn, dn, n: read$</td></tr>
</table>

30.3 Különbség

Bemenet: két rendezett sorozat
- első file tartalmazza az angolul tudó hallgatók neveit
- második file tartalmazza a németül tudó hallgatók neveit

Kik azok, akik csak angolul beszélnek?
(Különbség)



Specifikáció:

$A = \left(a: infile(\mathbb{K}^*), n: infile(\mathbb{K}^*), z: outfile(\mathbb{K}^*)\right)$
$Ef = (a = a' \wedge n = n' \wedge a \uparrow \wedge n \uparrow)$
$Uf = \left(z = \oplus_{e \in a' \cup n'} f(e)\right)$

$$f(e) = \begin{cases} < e > & e \in a \wedge e \notin n \\ < > & e \notin a \wedge e \in n \\ < > & e \in a \wedge e \in n \end{cases}$$

Algoritmus:

$z :=<>$ $sa, da, a: read$ $sn, dn, n: read$		
$sa = norm$		
$(sn = abnorm) \vee (da < dn)$	$sn = norm \wedge da = dn$	$(sn = norm \wedge da > dn)$
$z: write(da)$ $sa, da, a: read$	$sa, da, a: read$ $sn, dn, n: read$	$sn, dn, n: read$

30.4 Összetettebb uniós

Adott egy zh-kat és egy pót zh-kat tartalmazó file.

Az adott fileban: - EHA kód és pont szerepel
EHA kód szerint szigorúan monoton növekvő sorrendben

Mi lett az adott diák eredménye, ha a jobbik pontszám számít?
(Unió)

Specifikáció:

$tanuló = rec(Eha: \mathbb{K}^*, pont: \mathbb{N})$

$A = \big(zh: infile(tanuló), pót: infile(tanuló), eredmény: outfile(tanuló)\big)$

$Ef = (zh = zh' \wedge pót = pót' \wedge zh \uparrow Eha \wedge pót \uparrow Eha)$

$Uf = \Big(eredmény = \bigoplus_{e.Eha \in Eha(zh') \cup Eha(pót')} f(e)\Big)$

$$f(e) = \begin{cases} < e > & e.Eha \in Eha(zh') \wedge e.Eha \notin Eha(pót') \\ < e > & e.Eha \notin Eha(zh') \wedge e.Eha \in Eha(pót') \\ g(e) & e.Eha \in Eha(zh') \wedge e.Eha \in Eha(pót') \end{cases}$$

$g(e) - t$ két féle képpen lehet felírni. Szabadon választható az egyik.

1. :

$g(e) = e(zh')^{pont \leftarrow max(e(zh').pont, e(pót').pont)}$

A felírás: $e(zh')$ - vegyük a hallgatót a zh'-ből

$pont \leftarrow max(e(zh').pont, e(pót').pont)$ Módosítsuk a pont rekordot, a zh' és pót' közül a maximálisra.

2.:

$g(e) = \langle e(zh').Eha; max(e(zh').pont, e(pót').pont)\rangle$

Algoritmus:

<table>
<tr><td colspan="3">$eredmény :=<>$ $szh, dzh, zh: read$ $spot, dpot, pot: read$</td></tr>
<tr><td colspan="3">$szh = norm \vee spot = norm$</td></tr>
<tr><td>$(spot = abnorm) \vee (szh = norm \wedge dzh.eha < dpot.eha)$</td><td>$szh = norm \wedge spot = norm \wedge dzh.eha = dpot.eha$</td><td>$(szh = abnorm) \vee (spot = norm \wedge dzh.eha > dpot.eha)$</td></tr>
<tr><td>$eredmény: write(dzh)$
$szh, dzh, zh: read$</td><td>$dzh.pont :=$
$max(dzh.pont, dpot.pont)$
$eredmény: write(dzh)$
$szh, dzh, zh: read$
$spot, dpot, pot: read$</td><td>$eredméy: write(dpot)$
$spot, dpot, pot: read$</td></tr>
</table>

31. Kiegészítés

Oktatók weboldala:

Gregorics Tibor: Programozás - http://people.inf.elte.hu/gt/prog/prog.html
Gregorics Tibor: Objektum elvű alkalmazások - http://people.inf.elte.hu/gt/oaf/oaf.html
Hudoba Péter: Programozás - http://hudi89.web.elte.hu/

Saját weboldal:

http://people.inf.elte.hu/naksabi/

Egyéb:

Lövei László weboldala: http://digitus.itk.ppke.hu/~lovei/

Wikipedia

Gregorics Tibor - Programozás – Tervezés c. könyve
Gregorics Tibor - Programozás – Megvalósítás c. könyve

Minta programok:

Folyamatosan bővülnek a weboldalamon:
http://people.inf.elte.hu/naksabi/

32. Felhasznált irodalom (Források)

Programozási alapismeretek:

Zsakó László előadásai
Szlávi Péter gyakorlatai

Szabó Richárd jegyzete

Saját minta zárthelyi megoldás

http://progalap.elte.hu/

Programozás:

Gregorics Tibor – Programozás előadásai
Gregorics Tibor - Programozás – Tervezés c. könyve
Gregorics Tibor - Programozás – Megvalósítás c. könyve
Gregorics Tibor weboldala – programozási tételek
http://people.inf.elte.hu/gt/prog/prog.html

Veszprémi Anna táblás gyakorlatai

Hudoba Péter géptermi gyakorlatai
Hudoba Péter weboldala - http://hudi89.web.elte.hu/

Bjarne Stroustrup – A C++ programozási nyelv

Lövei László oktatáshoz használt minta programjai
Lövei László weboldala - http://digitus.itk.ppke.hu/~lovei/

Wikipedia - http://hu.wikipedia.org/wiki/C%2B%2B

Saját beadandó feladatok