Szekvenciális inputfájl felsorolása

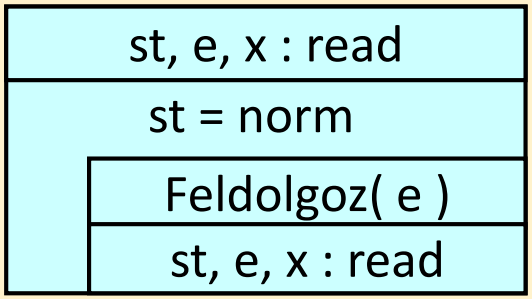
Gregorics Tibor

gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

Szekvenciális inputfájl felsorolása

- Egy x:infile(E) szekvenciális inputfájl (amely szerkezetileg egy sorozat) elemeit az st,e,x:read művelet (e:E, st:Status={abnorm, norm}) segítségével sorolhatjuk fel.
- A felsorolás műveletei:
  - first() ~ st, e, x : read
  - next() ~ st, e, x : read
  - current() ~ e
  - end() ~ st=abnorm



- A felsorolás az előre olvasási stratégiára épül: először olvasunk, majd ezután megvizsgáljuk, hogy sikerült-e az olvasás. Ha igen, a beolvasott elemet feldolgozzuk.
- A specifikációban a felsorolást az e∈x szimbólummal jelöljük.

Fájlkezelési feladatok

- A gyakorlatban sokszor találkozunk olyan feladatokkal, amikor sorozatokból sorozatokat kell előállítani. Ha ezek a sorozatok például szöveges állományokban találhatók, akkor a bemenő sorozatokat szekvenciális inputfájlként, a kimenőket szekvenciális outputfájlként érdemes kezelni.
- A leggyakoribb ilyen feladatok:
  - másolás illetve elemenkénti átalakítás (például riport készítés)
  - kiválogatás
  - szétválogatás
  - összefuttatás
- Ezekben a feladatokban az a közös, hogy mindegyiket az összegzés programozási tételére vezethetjük vissza, és a szekvenciális inputfájl felsorolását használjuk – kivéve az összefuttatást, mert az egyedi felsorolást kíván.

Összegzés fájlkezeléshez

Általános összegzés

A : t:enor(E), s:H

Ef : $t = t_0$

Uf: $s = \Sigma_{e \in t_0} f(e)$

f :E → H
+:H×H→H
0∈H

Speciális összegzés: fájlkezelés

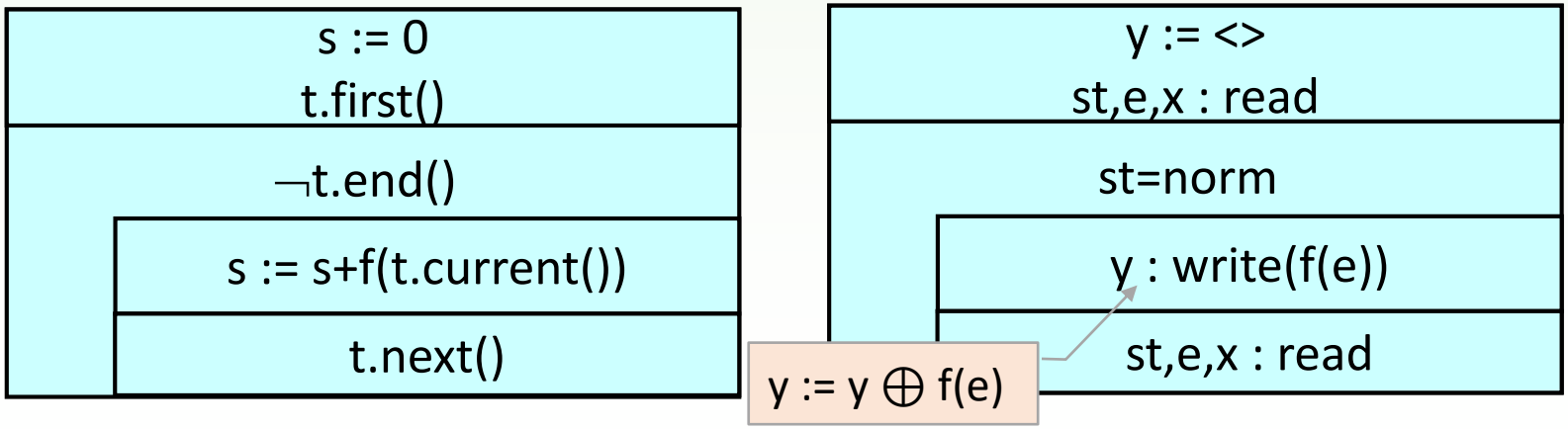
A : x:infile(E), y:outfile(F)

Ef : $x = x_0$

Uf: $y = \bigoplus_{e \in x_0} f(e)$

f :E → F*
⊕:F*×F*→F*
<>∈F*

Összegzés:
t:enor(E) ~ x:infile(E)
st,e,x : read
H,+,0 ~ F*, ⊕, <>



1.Feladat

Alakítsunk át egy ékezeteket tartalmazó szöveget (amely egy szöveges állományban található) ékezet nélkülire (az eredmény egy másik szöveges állományba kerüljön)!

A : x:infile(Char) , y:outfile(Char)

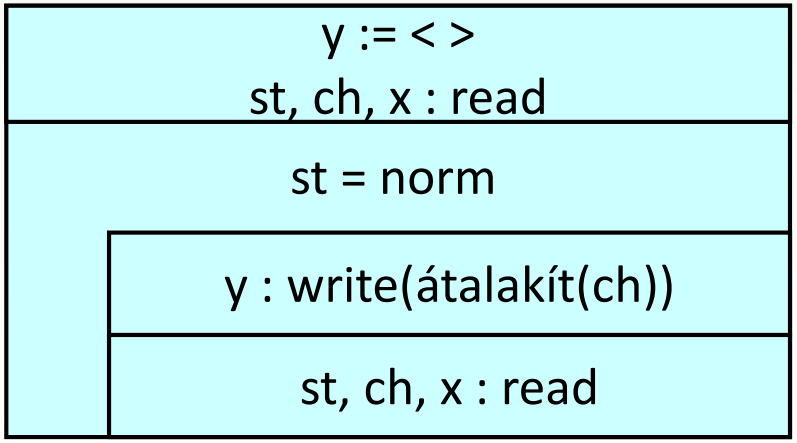
Ef : $x = x_0$

Uf: $y = \bigoplus_{ch \in x_0} \langle átalakít(ch) \rangle$

ahol átalakít : Char → Char és átalakít(ch) = …

Összegzés:

t:enor(E)	~	x:infile(Char) st,ch,x : read
e	~	ch
f(e)	~	<átalakít(ch)>
H,+,0	~	Char*, ⊕, <>



Szürkedoboz tesztelés vázlata

- Az összegzés teszteléséhez vizsgálni kell
  - a felsorolót (más felsorolós programozási tételekhez hasonlóan)
    - felsorolás hossza szerint: 0, 1, 2, illetve több elem felsorolása
    - felsorolás eleje, vége szerint: összegzésnél ez 2 eltérő elem felsorolásával már ellenőrizhető
  - a terheléses teszt most nem túl érdekes, hiszen csak az inputfájl méretével azonos outputfájlt hozhatunk létre
- Ezeken kívül ellenőrizni kell a konverziót.

a felsoroló hossza szerint: 0, 1, 2, illetve több karaktert tartalmazó input (másolás)

eleje, vége szerint:	x = <ab>	→	y = <ab>
	x = <áé>	→	y = <ae>
az átalakítás szerint:	x = <ááíöőúüű>	→	y = <aeioouuu>
	x = <aeioouuu>	→	y = <aeioouuu>
	x = <bsmnz>	→	y = <bsmnz>

C++

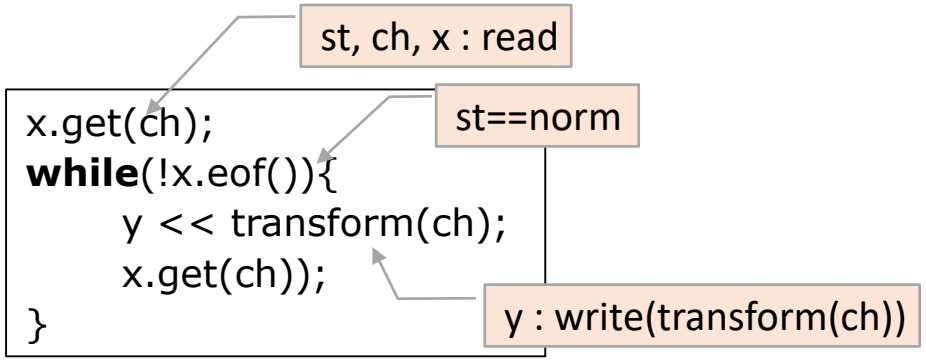
- Az szöveges állományban elhelyezett adatok inputfájlszerű olvasásához ifstream típusú adatfolyam objektumokat, szöveges állomány outputfájlszerű feltöltéséhez pedig ofstream típusú adatfolyam objektumot használunk.
- A C++ nyelv is előre olvasási stratégiát alkalmaz a fájl olvasáshoz.
- A karakterenkénti olvasást leíró st, ch, x : read művelet megvalósításai:
  - x >> ch
    - Ez az elválasztó jeleket (white space) nem olvassa be, hanem átlépi azokat, kivéve, ha kikapcsoljuk ezt az automatizmust (x.unsetf(ios::skipws)).
  - x.get(ch)
    - Ez minden karaktert (elválasztó jeleket is) beolvas.
- Az st==norm vizsgálatot a !x.eof() helyettesíti.

C++ program

```
int main()
{
	ifstream x( "input.txt" );
	if ( x.fail() ){ cout << "Wrong file name!\n"; return 1;}
	ofstream y( "output.txt" );
	if ( y.fail() ){ cout << "Wrong file name!\n"; return 1;}

	char ch;
	while(x.get(ch)){
		y << transform(ch);
	}

	return 0;
}
```



Karakterek átalakítása

```
char transform(char ch)
{
   char new_ch;
   switch (ch) {
      case 'á'   :                          new_ch = 'a'; break;
      case 'é'    :                         new_ch = 'e'; break;
      case 'í'    :                         new_ch = 'i'; break;
      case 'ó'   : case 'ö' : case 'õ' :    new_ch = 'o'; break;
      case 'ú'   : case 'ü' : case 'û' :    new_ch = 'u'; break;
      case 'Á'   :                          new_ch = 'A'; break;
      case 'É'   :                          new_ch = 'E'; break;
      case 'Í'   :                          new_ch = 'I'; break;
      case 'Ó'   : case 'Ö' : case 'Õ':     new_ch = 'O'; break;
      case 'Ú'   : case 'Ü' : case 'Û':     new_ch = 'U'; break;
      default    :                          new_ch = ch;
   }
   return new_ch;
}
```

2.Feladat

Válogassuk ki a páros számokat egy egész számokat tartalmazó szöveges állományból a konzol ablakba!

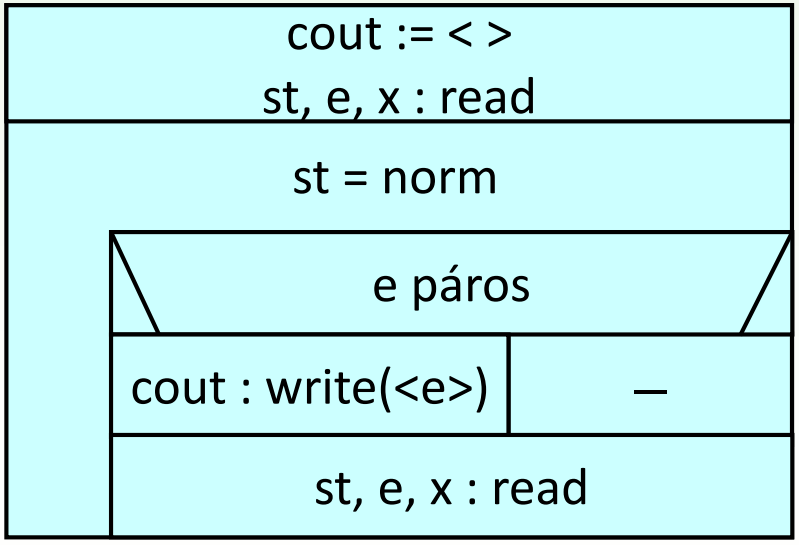
A : x:infile($\mathbb{Z}$) , cout:outfile($\mathbb{Z}$)

Ef : $x = x_0$

Uf : cout = $\bigoplus_{e \in x_0 \text{ e páros}} <e>$

Összegzés:

t:enor(E)	~	x:infile($\mathbb{Z}$) st,e,x : read
f(e)	~	<e> ha e páros
H,+,0	~	$\mathbb{Z}^*$, $\oplus$, <>



Szürkedoboz tesztelés vázlata

- Vizsgálni kell
  - a felsorolót
    - felsorolás hossza szerint: 0, 1, 2, több
    - felsorolás eleje, vége szerint: 2 eltérő elem felsorolása
  - a terheléses teszt most sem érdekes
  - a kiválogatás feltételeit

a felsoroló hossza szerint: 0, 1, 2, több egész számot tartalmazó input, amely csupa páros számból áll (másolás)

eleje, vége szerint: x = <2, 2> → y = <2, 2>

a feltétel szerint: x = <-100, -55, -2, -1, 0, 1, 2, 55, 100>

→ cout = <-100, -2, 0, 2, 100 >

C++ program

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
	ifstream x;
	bool error = true;
	do{
		string fname;
		cout << "file name: "; cin >> fname;
		x.open(fname.c_str());
		if( (error=x.fail()) ) {
			cout << "Wrong file name!\n"; x.clear();
		}
	}while(error);
	cout << "Selected even numbers: ";
	int e;
	while(x >> e) {
		if(0==e%2) cout << e << " ";
	}
	return 0;
}
```

```
x >> e;
while(!x.fail()){
	if(0==e%2) cout << e;
	x >> e;
}
```

Az elválasztójelek átlépése után az e típusának megfelelő értéket olvas.

st, e, x : read

cout : write(<e>)

Ha !x.eof() helyett a !x.fail()-t használjuk, akkor az nemcsak a fájl végét, hanem egyéb olvasási hibákat is észrevesz.

3.Feladat

Egy könyvtári nyilvántartásból (szöveges állomány) válogassuk ki a nulla példányszámú könyveket és a 2000-nél régebbi kiadásúakat egy-egy új szöveges állományba!

A : x:infile(Könyv) , y:outfile(Könyv2) , z:outfile(Könyv2)

Könyv = rec(azon : $\mathbb{N}$, szerző : String, cím : String, kiadó : String, év : String, pld : $\mathbb{N}$, isbn:String)

Könyv2 = rec(azon : $\mathbb{N}$, szerző : String, cím : String)

Ef : $x = x_0$

Uf : $y = \bigoplus_{\substack{dx \in x_0 \\ dx.pld=0}} <(dx.azon, dx.szerző, dx.cím)> \wedge$

$z = \bigoplus_{\substack{dx \in x_0 \\ dx.év< \text{”2000”}}} <(dx.azon, dx.szerző, dx.cím)>$

Algoritmus

Összegzés:

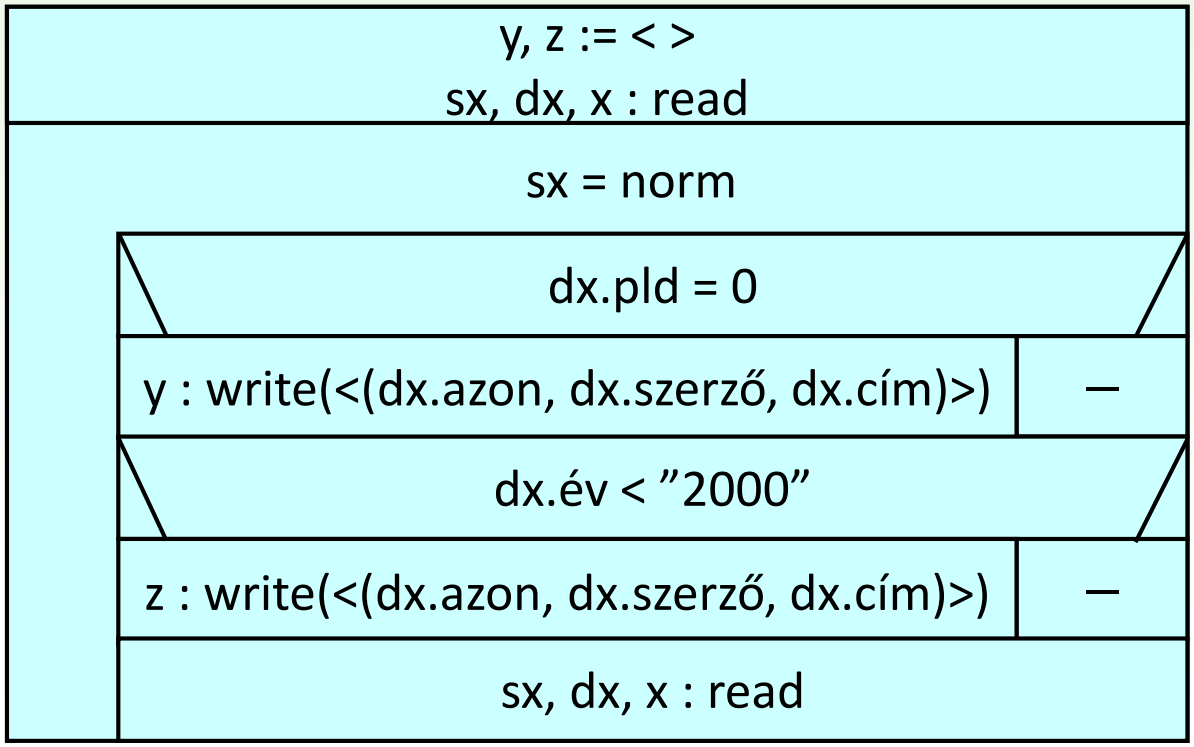
t:enor(E) ~ x:infile(Könyv), sx,dx,x : read

e ~ dx

f_1(e) ~ <(dx.azon, dx.szerző, dx.cím)> ha dx.pld = 0

f_2(e) ~ <(dx.azon, dx.szerző, dx.cím)> ha dx.év < ”2000”

H,+,0 ~ Könyv2*, ⊕, <>



Szürkedoboz tesztelés vázlata

- Vizsgálni kell
  - a felsorolót
    - felsorolás hossza szerint: 0, 1, 2, több
    - felsorolás eleje, vége szerint: 2 eltérő elem felsorolása
  - a terheléses teszt most sem érdekes
  - a szétválogatás feltételeit

a felsoroló hossza szerint:	0, 1, 2, több olyan könyv, amelyek mind megfelelnek az összes feltételnek (másolás)
eleje, vége szerint:	két könyv: első és a második is nulla példányszámú és 2000-nél régebbi
a feltételek szerint:	nulla és nem-nulla példányszámú, illetve
	2000-nél régebbi és nem régebbi könyvek

read és write függvény

sorokba tördelt,
szigorúan pozícionált inputfájl

```
12 J. K. Rowling    Harry Potter II.      Animus       2000   0  963 8386 94 O
15 A. A. Milne      Micimackó             Móra         1936  10  963 11 1547 X
17 Gárdonyi Géza    A láthatatlan ember   Szépirodalmi 1973   0  SZ 1823-D-7374
25 Fekete István    Zsellérek             Nestor       1994  12  963 7523 3 4 O
```

```
bool read(ifstream &f, Book &e, Status &st){
        string line;
        getline(f,line);
        if (!f.fail() && line!="") {
                st = norm;
                e.id        = atoi(line.substr( 0, 4).c_str());
                e.author    =      line.substr( 5,14);
                e.title     =      line.substr(21,19);
                e.publisher =      line.substr(42,14);
                e.year      =      line.substr(58, 4);
                e.nc        = atoi(line.substr(63, 3).c_str());
                e.isbn      =      line.substr(67,14);
        }
        else st=abnorm;

        return norm==st;
}
```

sort olvas

karakterláncot számmá alakít

rész-sztring

C stílusú karakterláncot csinál

logikai értéket is visszaad

```
void write(ofstream &f, const Book &e){
        f << setw(4)  << e.id     << ' '
          << setw(14) << e.author << ' '
          << setw(19) << e.title  << endl;
}
```

pozícionált kiírás
#include <iomanip>

Megvalósítás függvényekkel

```
bool read(ifstream &f, Book &e, Status &st);
void write(ofstream &f, const Book &e);

int main()
{
        ifstream x("inp.txt");
        if (x.fail() ) { … }
        ofstream y("out1.txt");
        if (y.fail() ) { … }
        ofstream z("out2.txt");
        if (z.fail() ) { … }

        Book dx;
        Status sx;
        while(read(x,dx,sx)) {
                if (0==dx.nc)          write(y,dx);
                if (dx.year<"2000") write(z,dx);
        }
        return 0;
}
```

```
struct Book{
        int id;
        string author;
        string title;
        string publisher;
        string year;
        int nc;
        string isbn;
};

enum Status{abnorm, norm};
```

```
read(x,dx,sx);
while(norm==sx){
        if (0==dx.nc)          write(y,dx);
        if (dx.year<"2000") write(z,dx);
        read(x,dx,sx);
}
```

Megvalósítás osztályokkal

```
struct Book{
        int id;
        std::string author;
        std:: string title;
        std:: string publisher;
        std:: string year;
        int nc;
        std:: string isbn;
```

```
int main()
{
        Stock  x("input.txt");
        Result y("output1.txt");
        Result z("output2.txt");

        Book dx;
        Status sx;
        while(x.read(dx,sx)){
                if (0==dx.nc)          y.write(dx);
                if (dx.year<"2000") z.write(dx);
        }
        return 0;
}
```

```
enum Status{abnorm, norm};
class Stock{
public:
        Stock(std::string fname);
        bool read(Book &dx, Status &sx);
private:
        std::ifstream f;
};
```

a belseje nem változott

```
f.open(fname);
if(f.fail()) {
        cout << "Wrong filename\n";
        exit(1);
}
```

#include <cstdlib>

```
class Result{
public:
        Result(std::string fname);
        void write(const Book &dx);
private:
        std::ofstream f;
};
```

a belseje nem változott

4.Feladat

Egy szöveges állományban egy kurzus félévéves számonkéréseinek eredményét helyeztük el. Minden sor egy hallgató adatait tartalmazza. Ebben szóközökkel vagy tabulátorjellel elválasztva az alábbi sorrendben találjuk az adatokat :

- neptun-kód (6 számjegy),
- „+” és „-” -ok összefüggő (szóközökkel sem elválasztott) nem üres sztring
- 1 beadandó és a 4 zárthelyi eredménye (mindegyik 0 .. 5)

Határozzuk meg azon hallgatók félévvégi összesített jegyét, akik kaphatnak jegyet (azaz a „+” és „-” -ok összege nem negatív, és egyik jegyük sem elégtelen)!

```
AA11XX   ++++-++++      5 5 5 5 5
CC33ZZ   ++++--++--     2 1 0 5 5
BB22YY   --+---+++-     2 2 3 3 5
```

Megoldási terv

A : x : infile(Hallgató) , y : outfile(Értékelés)

Hallgató = rec(neptun : String, pm : String, jegyek : $\{0..5\}^5$)

Értékelés = rec(neptun : String, jegy : $\mathbb{R}$)

Ef : $x = x_0$

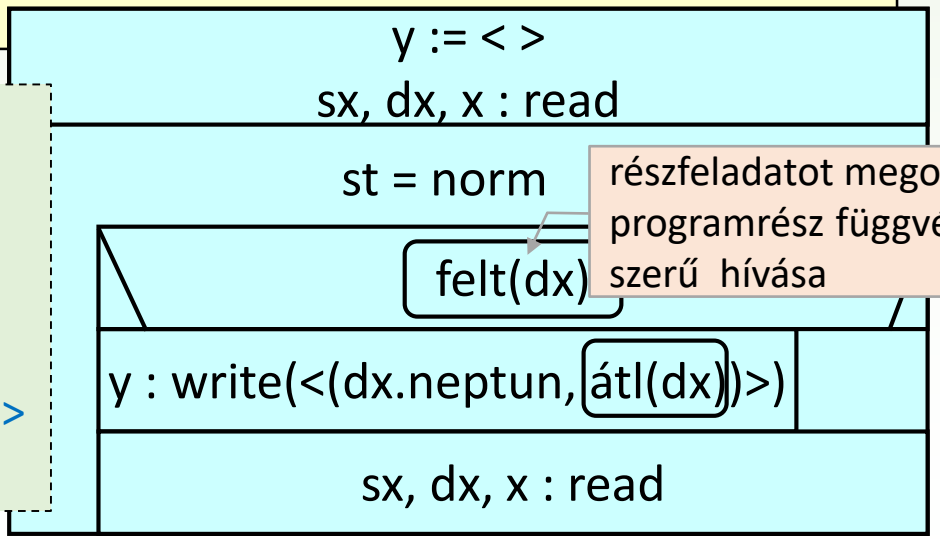
Uf : $y = \bigoplus_{\text{felt}(dx)}^{dx \in x_0} <(dx.\text{neptun}, \text{átl}(dx))>$

$$\text{felt}(dx) = \forall_{i=1}^{5} (dx.\text{jegyek}[i] > 1) \wedge \left(\sum_{i=1}^{|dx.pm|}{}_{dx.pm[i]='-'} 1 \leq \sum_{i=1}^{|dx.pm|}{}_{dx.pm[i]='+'} 1 \right)$$

$$\text{átl}(dx) = \left(\sum_{i=1}^{5} dx.\text{jegyek}[i] \right) / 5$$

Összegzés:

t:enor(E)	~	x:infile(Hallgató)
		sx,dx,x : read
e	~	dx
f(e)	~	ha felt(dx) akkor
		<(dx.neptun, átl(dx))>
H,+,0	~	Értékelés*, ⊕, <>



Alprogramok

$l := (\forall_{i=1}^{5}\ dx.eredm[i]>1)$

Opt. lineáris keresés:

t:enor(E)	~	i = 1 .. 5
e	~	i
felt(e)	~	dx.eredm[i]>1

$p, m := \sum_{i=1}^{|dx.pm|} 1_{dx.pm[i]=' +'}\ ,\ \sum_{i=1}^{|dx.pm|} 1_{dx.pm[i]=' -'}$

Két számlálás egyben:

t:enor(E)	~	i = 1 .. |dx.pm|
e	~	i
felt1(e)	~	dx.pm[i] = '+'
felt2(e)	~	dx.pm[i] = '-'

$s := (\sum_{i=1}^{5} dx.eredm[i]) / 5$

Összegzés:

t:enor(E)	~	i = 1 .. 5
e	~	i
f(e)	~	dx.eredm[i]
H,+,0	~	$\mathbb{N}$, +, 0

l := felt(dx) $l : \mathbb{L}$

```
l, i := true, 1
l ∧ i ≤ 5
    l := dx.eredm[i] > 1
    i := i + 1
p, m := 0, 0
i = 1 .. |dx.pm|
    dx.pm[i] = '+'
        p := p + 1
    —
    dx.pm[i] = '-'
        m := m + 1
    —
l := l ∧ p ≥ m
```

a := átl(dx.eredm) $a : \mathbb{R}$

```
s := 0
i = 1 .. 5
    s := s + dx.eredm[i]
a := s / 5
```

Szürkedoboz tesztelés vázlata

Külső feltételes összegzés:

a felsoroló hossza szerint:	0, 1, 2, több olyan hallgató, akik kaphatnak jegyet
a felsorolás eleje/vége:	a fentiekkel letudva
terhelés:	nem kell
a cond() és f() vizsgálata:	lásd alább

Plusz-mínuszok számlálása:

a felsoroló hossza szerint:	0, 1, 2, több csak ’+’
a felsorolás eleje/vége:	2 hosszú felsorolások, felváltva ’+’ vagy ’–’ (4 eset)
eredmény szerint:	eddigieken túl: 0, 1, több ’–’ és mellette ’+’-ok

Nincs elégtelen eldöntése (optimista linker) :

a felsoroló hossza szerint:	nem kell (garantáltan 5)
a felsorolás eleje/vége:	csak az eleje 1, csak a vége 1
eredmény szerint:	csupa 1, van 1, mind legalább 2

Osztályzatok összegzése:

a felsoroló hossza szerint:	nem kell (garantáltan 5)
a felsorolás eleje/vége:	elején és végén különböző osztályzatokkal
terhelés:	nem kell

C++ program

```
bool cond(const vector<int> &marks, const string &pm );
double  avr(const vector<int> &marks);

int  main(){
      try{
            InpFile x("input.txt");
            OutFile y("output.txt");
            Student dx;
            Status sx;
            while(x.read(dx,sx)) {
                  if (cond(dx.marks, dx.pm)) {
                        Evaluation dy(dx.neptun, avr(dx.marks));
                        y.write(dy);
                  }
            }
      }catch( InpFile::Errors er ) {
            if( er==InpFile::FILE_ERROR ) cout << … ;
      }catch( OutFile::Errors er ) {
            if( er==OutFile::FILE_ERROR ) cout << … ;
      }
      return 0;
}
```

Az InpFile illetve OutFile típusú objektumok által jelzett hibák (dobott kivételek) lekezelése

Segédfüggvények

```
bool cond(const vector<int> &marks, const string &pm ) {
	bool l = true;
	for(unsigned int i=0; l && i<marks.size(); ++i){
		l=marks[i]>1;
	}
	int p, m; p = m = 0;
	for(unsigned int i = 0; i<pm.size(); ++i){
		if(pm[i]=='+') ++p;
		if(pm[i]=='-') ++m;
	}
	return l && m<=p;
}
```

```
double avr(const vector<int> &marks) {
	int s = 0;
	for(unsigned int i = 0; i< marks.size(); ++i){
		s += marks[i];
	}
	return (0== marks.size() ? 0.0 : double(s) / marks.size());
}
```

Szekvenciális inputfájl

```
struct Student {
        std::string neptun;
        std::string pm;
        std::vector<int> marks;
};
enum Status {abnorm, norm};

class InpFile{
public:
        enum Errors{FILE_ERROR};
    InpFile(std::string fname){
                f.open(fname.c_str());
                if(f.fail()) throw FILE_ERROR;
        }
    bool read( Student &e, Status &st);
private:
    std::ifstream f;
};
```

hibaesetek (kivételek) felsorolása

hiba jelzése (kivétel dobása)
Csak jelzi a hibát, de nem kezeli le.

```
bool InpFile::read(Student &e, Status &st)
{
        string line;
        getline(f, line);
        if (!f.fail() && line!="")
                st=norm;
                istringstream in(line);
                in >> e.neptun;
                in >> e.pm;
                e.marks.clear();
                int mark;
                while( in >> mark )
                        e.marks.push_back(mark);
        } else st=abnorm;
        return norm==st;
}
```

egy sor adatainak olvasásához
#include <sstream>

vector törlése

Szekvenciális outputfájl

```
struct Evaluation {
        std::string neptun;
        double mark;
        Evaluation(std::string str, double j) : neptun(str), mark(j) {}
};

class OutFile{
public:
        enum Errors{FILE_ERROR};
        OutFile(std::string fname){
                f.open(fname.c_str());
                if(f.fail()) throw FILE_ERROR;
        }
        void write(const Evaluation &dy) {
                f.setf(std::ios::fixed);
                f.precision(2);
                f << dy.neptun << std::setw(7) << dy.mark << std::endl;
        }
private:
    std::ofstream f;
};
```

hibaesetek (kivételek) felsorolása

hiba jelzése (kivétel dobása)
Csak jelzi a hibát, de nem kezeli le.

#include <iomanip>

Feladat és a program módosítása

Egy szöveges állományban a sorok a hallgatók nevével kezdődnek, amely tetszőleges számú, de legalább egy tagból áll (közöttük elválasztó jelek).

Gipsz Jakab Elemér	AA11XX	++++++++++ 5 5 5 5 5
Szer Elek	CC33ZZ	+++++-++++ 2 1 0 5 1
Jose Fernando Llano del Colona	BB22YY	---+++---- 2 4 4 0 0

```
int  main(){
      try{
            InpFile x("input.txt");
            OutFile y("output.txt");
            Student dx;
            Status sx;
            while(x.read(dx,sx)) {
                  if (dx.has) {
                        Evaluation dy(dx.neptun, dx.result);
                        y.write(dy);
                  }
            }
      }
      …
}
```

```
struct Student {
      std::string name;
      std::string neptun;
      bool has;
      double result;
};
```

A feldolgozandó szekvenciális inputfájl egy-egy eleme nem a szöveges állomány megfelelő sorának másolata: csak a megoldáshoz szükséges azon adatokból áll, amelyeket az egyes sorokból lehet kiszámítani.

Változó számú adat olvasása

```
bool InpFile::read(Student &e, Status &st)
{
      string line, str;
      getline(f, line);
      if (!f.fail() && line!="") {
            st=norm;
            istringstream in(line);
            in >> e.name >> e.neptun;
            in >> str;
            while( !('+'== str[0] || '-'== str[0]) ){
                  e.name += " " + e.neptun;
                  e.neptun = str;
                  in >> str;
            }
            vector<int> marks;
            int mark;
            while( in >> mark ) marks.push_back(mark);
            e.has = cond(marks, str);
            e.result = avr(marks);
      } else st=abnorm;
      return norm==st;
}
```

e kitöltése az aktuális sor (line) alapján

ha str nem + vagy – jellel kezdődik, akkor az még a név része vagy legfeljebb a neptun kód

amit eddig neptun kódnak hittünk, az még a név része

str-t tekintsük egyelőre neptun kódnak

InpFile osztály privát metódusai

Olvasás szöveges állományokból

f : infile(E)	st, data, f : read	st = norm
E ≡ char // karakterek elválasztás nélkül	f.get(data); f >> data; //f.unsetf(ios::skipws)	!f.eof()

E ≡ <elemi típus>
// elválasztó jelekkel szeparált elemi
// típusú érték
	f >> data;	!f.fail()

E ≡ rec(s1 : <elemi típus>,
s2 : <elemi típus>,
sn : <elemi típus>^{n},
…
)
// fix számú, elválasztó jelekkel szeparált
// elemi típusú értékekkel kitölthető
// struktúra
	f >> data.s1 >> data.s2;
	for(int i=0; i<n; ++i) {
	f >>data.sn[i];
	}
		!f.fail()

E ≡ sor
// sorokba szervezett, soronként eltérő
számú adat esetén
	string data;
	getline(f, data);
	istringstream is(data);
	is >> …
		!f.fail()