

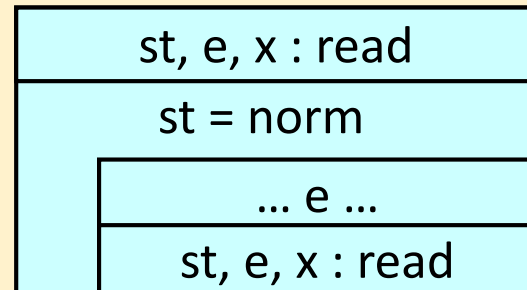
Szekvenciális inputfájl felsorolása

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 jelölhetjük.

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 ehhez a **szekvenciális inputfájl felsorolását** használjuk – kivéve az összefuttatást, mert az egyedi felsorolást kíván.

Összegzés fájlkezeléshez

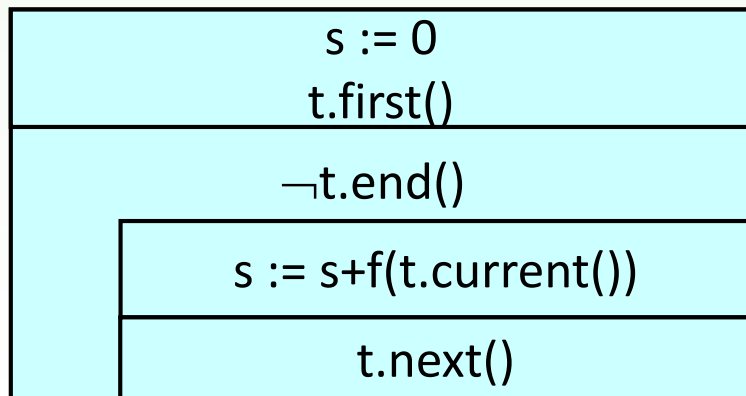
Általános összegzés

$A : t:\text{enor}(E), s:H$

$Ef : t = t_0$

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

$f : E \rightarrow H$
 $+: H \times H \rightarrow H$
 $0 \in H$



Speciális összegzés: kollatálás

$A : x:\text{infile}(E), y:\text{outfile}(F)$

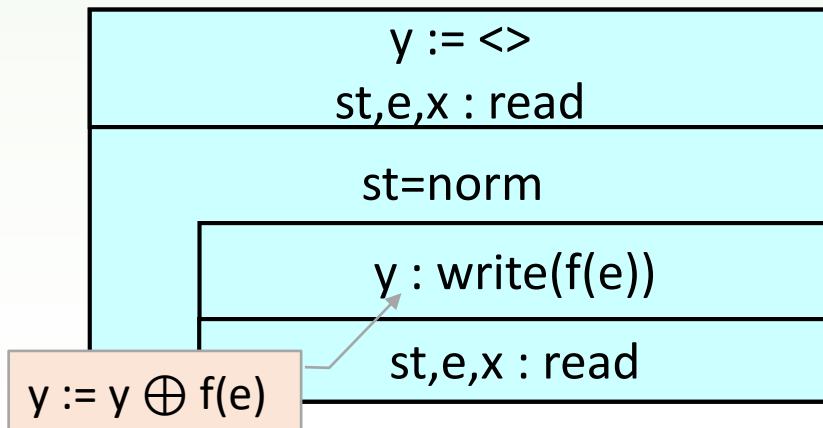
$Ef : x = x_0$

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

$f : E \rightarrow F^*$
 $\bigoplus : F^* \times F^* \rightarrow F^*$
 $\langle \rangle \in F^*$

Összegzés:

$t:\text{enor}(E) \sim x:\text{infile}(E)$
 $st, e, x : \text{read}$
 $H, +, 0 \sim F^*, \bigoplus, \langle \rangle$



1.Feladat

Alakítsunk át egy ékezeteket tartalmazó szöveget (amely egy karakteres szekvenciális inputfájl) ékezet nélkülire!

$A : x:\text{infile}(\text{Char}) , y:\text{outfile}(\text{Char})$

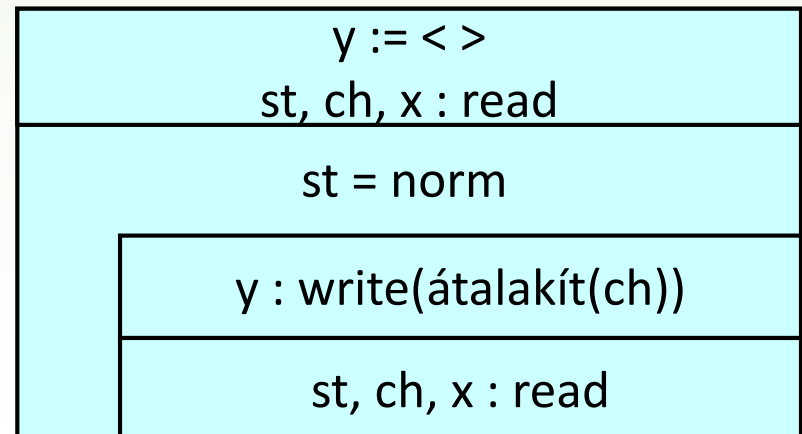
$Ef : x = x_0$

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

ahol $\text{átalakít} : \text{Char} \rightarrow \text{Char}$ és $\text{átalakít}(ch) = \dots$

Összegzés:

$t:\text{enor}(E) \sim$	$x:\text{infile}(\text{Char})$
	$st, ch, x : \text{read}$
$e \sim$	ch
$f(e) \sim$	$\langle \text{átalakít}(ch) \rangle$
$H, +, 0 \sim$	$\text{Char}^*, \bigoplus, \langle \rangle$



Szűrkedoboz tesztelés vázlata

- ❑ Az összegzés teszteléséhez vizsgálni kell
 - a felsorolót (más felsorolás 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)

az átalakítás szerint: x = <áéíöőúüű> → y = <aeioouuu>

 x = <aeioouuu> → y = <aeioouuu>

 x = <bsmnz> → y = <bsmnz>

 x = <Hazádnak rendületlenül ...>

C++

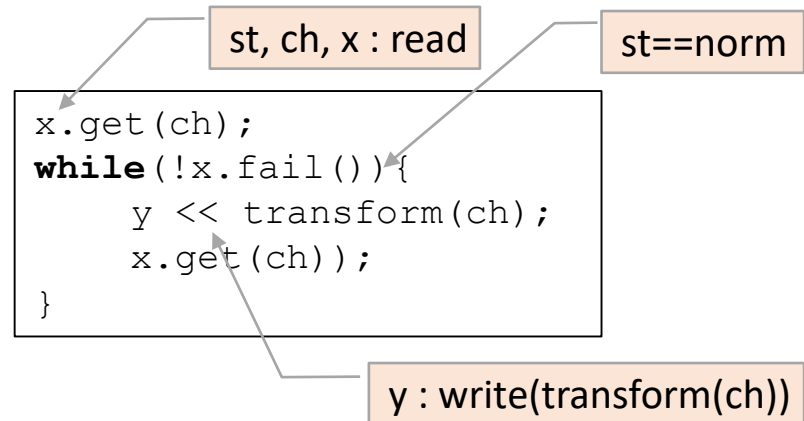
- ❑ A C++ nyelv is előre olvasási stratégiát alkalmaz a fájlolvasáshoz.
- ❑ A karakterenkénti olvasást leíró `st`, `ch`, `x : read` művelet megvalósításai:
 - `x >> ch`
 - Ez az elválasztó jeleket (white space) nem olvassa be, hanem átlépi azokat, kivéve, ha kikapcsoljuk ezt az automatizmust (`x.unsetf(ios::skipws)`).
 - `x.get(ch)`
 - Ez minden karaktert (elválasztó jeleket is) beolvas.
- ❑ A C++ nyelvben az `st==norm` vizsgálatot a `!x.eof()` helyettesíti. Sokszor azonban biztonságosabb a `!x.fail()` használata, amely nemcsak a fájl végének elérése miatti sikertelen olvasást jelzi, hanem mindenféle sikertelen olvasást (a fájl nincs helyesen kitöltve, hiányzik a legutolsó elem után a sorvége jel).

C++ program

```
int main()
{
    ifstream x( "input.txt" );
    if ( x.fail() ){ ... }
    ofstream y( "output.txt" );
    if ( y.fail() ){ ... }

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

    return 0;
}
```



C++ program

```
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 szekvenciális inputfájlból!

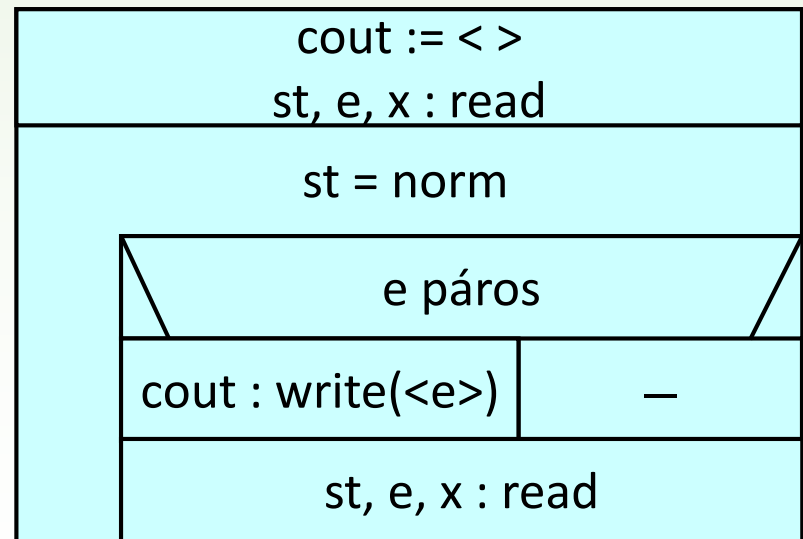
$A : x:\text{infile}(\mathbb{Z}) , \text{cout}:\text{outfile}(\mathbb{Z})$

$Ef : x = x_0$

$Uf : \text{cout} = \bigoplus_{\substack{e \in x_0 \\ e \text{ páros}}} \langle e \rangle$

Összegzés:

$t:\text{enor}(E) \sim x:\text{infile}(\mathbb{Z})$
 $f(e) \sim \text{st}, e, x : \text{read}$
 $\sim \langle e \rangle \text{ ha } e \text{ páros}$
 $H, +, 0 \sim \mathbb{Z}^*, \oplus, \langle \rangle$



Szűrkedoboz 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)

a feltétel szerint: $x = \langle -100, -55, -2, -1, 0, 1, 2, 55, 100 \rangle$
 $\rightarrow y = \langle -100, -2, 0, 2, 100 \rangle$

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;
}
```

Az elválasztójelek átlépése után az e típusának megfelelő értéket olvas.

st, ch, x : read

st==norm

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

cout : write(<e>)

3.Feladat és a specifikációja

Egy könyvtári nyilvántartásból válogassuk ki a nulla példányszámú könyveket és a 2000-nél régebbi kiadásúakat!

$A : x:\text{infile}(\text{Könyv}) , y:\text{outfile}(\text{Könyv2}) , z:\text{outfile}(\text{Könyv2})$

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

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

$Ef : x = x_0$

$Uf : y = \bigoplus_{dx \in x_0} \langle (dx.\text{azon}, dx.\text{szerző}, dx.\text{cím}) \rangle \wedge$

$dx.\text{pld}=0$

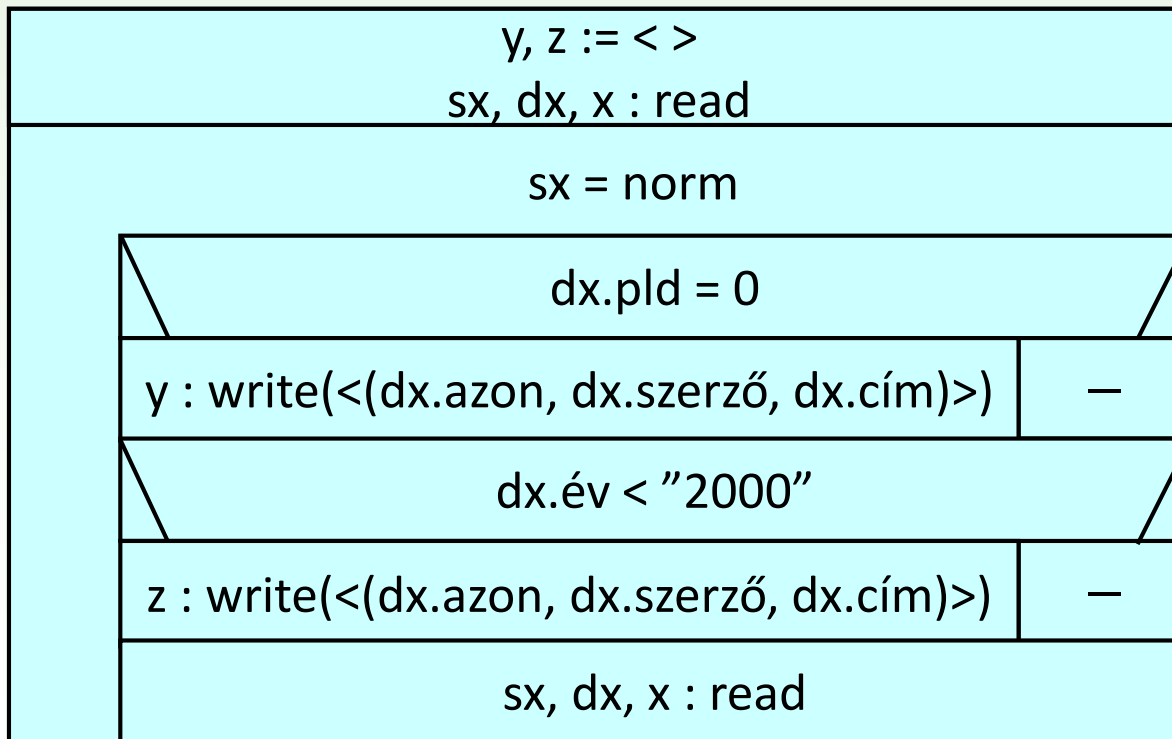
$z = \bigoplus_{dx \in x_0} \langle (dx.\text{azon}, dx.\text{szerző}, dx.\text{cím}) \rangle$

$dx.\text{év} < "2000"$

Algoritmus

Összegzés:

$t:enor(E) \sim x:infile(Könyv), sx, dx, x : read$
 $e \sim dx$
 $f_1(e) \sim \langle dx.azon, dx.szerző, dx.cím \rangle$ ha $dx.pld = 0$
 $f_2(e) \sim \langle dx.azon, dx.szerző, dx.cím \rangle$ ha $dx.év < "2000"$
 $H, +, 0 \sim Könyv2^*, \oplus, \langle \rangle$



Szűrkedoboz 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

<u>a felsoroló</u> hossza szerint:	0, 1, 2, több olyan könyv, amelyek mind megfelelnek az összes feltételnek (másolás)
<u>a feltételek</u> szerint:	nulla és nem-nulla példányszámú, illetve 2000-nél régebbi és nem régebbi könyvek

Megvalósítás read és write függvénnnyel

```
bool read(istream &x, Book &dx, Status &sx);  
void write(ostream &x, const Book &dx);
```

```
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);  
}
```


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(istream &f, Book &dx, Status &sx){
```

```
    string line;
```

```
    getline(f,line);
```

```
    if (!f.fail() && line!="") {
```

```
        sx = norm;
```

sort olvas

```
        dx.id = atoi(line.substr( 0, 4).c_str());
```

```
        dx.author = line.substr( 5,14);
```

```
        dx.title = line.substr(21,19);
```

```
        dx.publisher = line.substr(42,14);
```

```
        dx.year = line.substr(58, 4);
```

```
        dx.nc = atoi(line.substr(63, 3).c_str());
```

```
        dx.isbn = line.substr(67,14);
```

```
    }
```

```
    else sx=abnorm;
```

```
    return norm==sx;
```

```
}
```

karakterláncot számmá alakít

rész-sztring

C stílusú karakterláncot csinál

logikai értéket is visszaad

```
void write(ofstream &f, const Book &dy){
```

```
    f << setw(4) << dy.id << ' ' << ' '
```

```
    << setw(14) << dy.author << ' ' << ' '
```

```
    << setw(19) << dy.title << endl;
```

pozícionált kiírás

#include <iomanip>

Megvalósítás osztályokkal

```
int main()
{
    try{
        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);
        }
    } catch (Stock::Errors e) {
        if (Stock::FILE_ERROR==e) cout <<
    } catch (Stock::Errors e) {
        if (Stock::FILE_ERROR==e) cout <<
    }
    return 0;
}
```

```
f.open(fname.c_str());
if(f.fail()) throw FILE_ERROR;
```

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

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

a belseje nem változott

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

a belseje nem változott

```
f.open(fname.c_str());
if(f.fail()) throw FILE_ERROR;
```

4.Feladat

Egy szöveges állomány a félévéves számonkéréseinek eredményét egy hallgató egy sor formában tartalmazza. Egy sorban 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!

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}^7)
```

Értékelés = rec(neptun : String, jegy : {0..5})

$$Ef: \mathbf{x} = \mathbf{x}_0$$
$$Uf : y = \bigoplus_{\substack{dx \in x_0 \\ \text{felt}(dx)}} \langle dx.\text{neptun}, \text{átl}(dx) \rangle$$

$$\text{felt}(\text{dx}) = \bigvee_{i=1}^7 (\text{dx.jegyek}[i] > 1) \wedge \left(\sum_{i=1}^{|\text{dx.pm}|} 1 \leq \sum_{i=1}^{|\text{dx.pm}|} 1 \right)$$

$\text{dx.pm}[i] = '-'$
 $\text{dx.pm}[i] = '+'$

$$\text{atl}(\text{dx}) = (\sum_{i=1}^7 \text{dx.jegyek}[i]) / 7$$

Összegzés:

$$t:enor(E) \sim x:infile(Hallgató)$$

sx,dx,x : read

$$e \sim dx$$

f(e) ~ ha felt(dx) akkor
 $\langle dx.neptun, \text{átl}(dx) \rangle$

$$H_{+,0} \sim \text{Értékelés}^*, \oplus, \langle \rangle$$
$$y := \langle \rangle$$

sx, dx, x : read

```
st = norm
```

felt(dx)

```
y : write(<(dx.neptun, [atl(dx)])>)
```

sx, dx, x : read

Alprogramok

Opt. lineáris keresés:

$t:enor(E) \sim i = 1 \dots 7$
 $e \sim i$
 $felt(e) \sim dx.eredm[i] > 1$

$$l := (\bigvee_{i=1}^7 dx.eredm[i] > 1)$$

Két számlálás egyben:

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

$$p, m := \sum_{i=1}^{|dx.pm|} 1, \sum_{i=1}^{|dx.pm|} 1$$

$dx.pm[i] = '+' \quad dx.pm[i] = '-'$

Összegzés:

$t:enor(E) \sim i = 1 \dots 7$
 $e \sim i$
 $f(e) \sim dx.eredm[i]$
 $H, +, 0 \sim \mathbb{R}, +, 0$

$$s := \sum_{i=1}^7 dx.eredm[i] / 7$$

$l := felt(dx)$

$l, i := true, 1$

$l \wedge i \leq 7$

$l := dx.eredm[i] > 1$

$i := i + 1$

$p, m := 0, 0$

$i = 1 \dots |dx.pm|$

$dx.pm[i] = '+'$

$p := p + 1$

—

$dx.pm[i] = '-'$

$m := m + 1$

—

$l := l \wedge p \geq m$

$a := \text{átl}(dx.eredm)$

$s := 0$

$i = 1 \dots 7$

$s := s + dx.eredm[i]$

$a := s / 7$

Szűrkedoboz 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 7)
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 7)
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;
}
```

C++ függvények

```
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) {
    double s = 0.0;
    for(unsigned int i = 0; i< marks.size(); ++i){
        s += marks[i];
    }
    return (0== marks.size() ? 0 : 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 &dx, Status &sx);  
private:  
    std::ifstream f;  
};
```

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

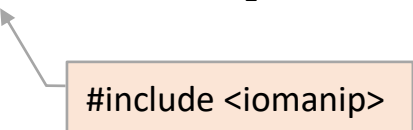
egy sor adatainak olvasásához
#include <sstream>

vector törlése

egy új elemet fűz
a vector végéhez

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;  
};
```



#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 &dx, Status &sx)
```

```
{
```

```
    string line, str;
```

```
    getline(f, line);
```

```
    if (!f.fail() && line!="") {
```

```
        sx=norm;
```

```
        istringstream in(line);
```

```
        in >> dx.name;
```

```
        in >> dx.neptun;
```

```
        in >> str;
```

```
        while( !( '+'== str[0] || '-'== str[0] ) ) {
```

```
            dx.name += " " + dx.neptun;
```

```
            dx.neptun = str;
```

```
            in >> str;
```

```
        }
```

```
        vector<int> marks;
```

```
        int mark;
```

```
        while( in >> mark ) marks.push_back(mark);
```

```
        dx.has = cond(marks, str);
```

```
        dx.result = avr(marks);
```

```
    } else sx=abnorm;
```

```
    return norm==sx;
```

```
}
```

dx 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 tekintjük egyelőre neptun kódnak

Inp osztály privát metódusai

Olvasás szöveges állományokból

x : infile(E)	st, data, x : read	st = abnorm
E ≡ char // karakterek elválasztás nélkül	x.get(data); x >> data; //x.unsetf(ios::skipws)	x.eof()
E ≡ <elemi típus> // elválasztó jelekkel szeparált elemi // típusú érték	x >> data;	x.fail()
E ≡ struct(s1 : <elemi típus>, s2 : <elemi típus>, sn : <elemi típus> ⁿ , ...) // fix számú, elválasztó jelekkel szeparált // elemi típusú értékekkel kitölthető // struktúra	x >> data.s1 >> data.s2; for(int i=0; i<n; ++i) { x >> data.sn[i]; }	x.fail()
E ≡ sor // sorokba szervezett, soronként eltérő számú adat esetén	string data; getline(x, data); istringstream is(data); is >> ...	x.fail()