Szekvenciális inputfájl felsorolása

Gregorics Tibor

gt@inf.elte.hu

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

Szekvenciális fájlok

```
if x = <> then
        st := abnorm
else
        st := norm;
         e := x1;
         x := <x2 , … , x|x|>
endif
```

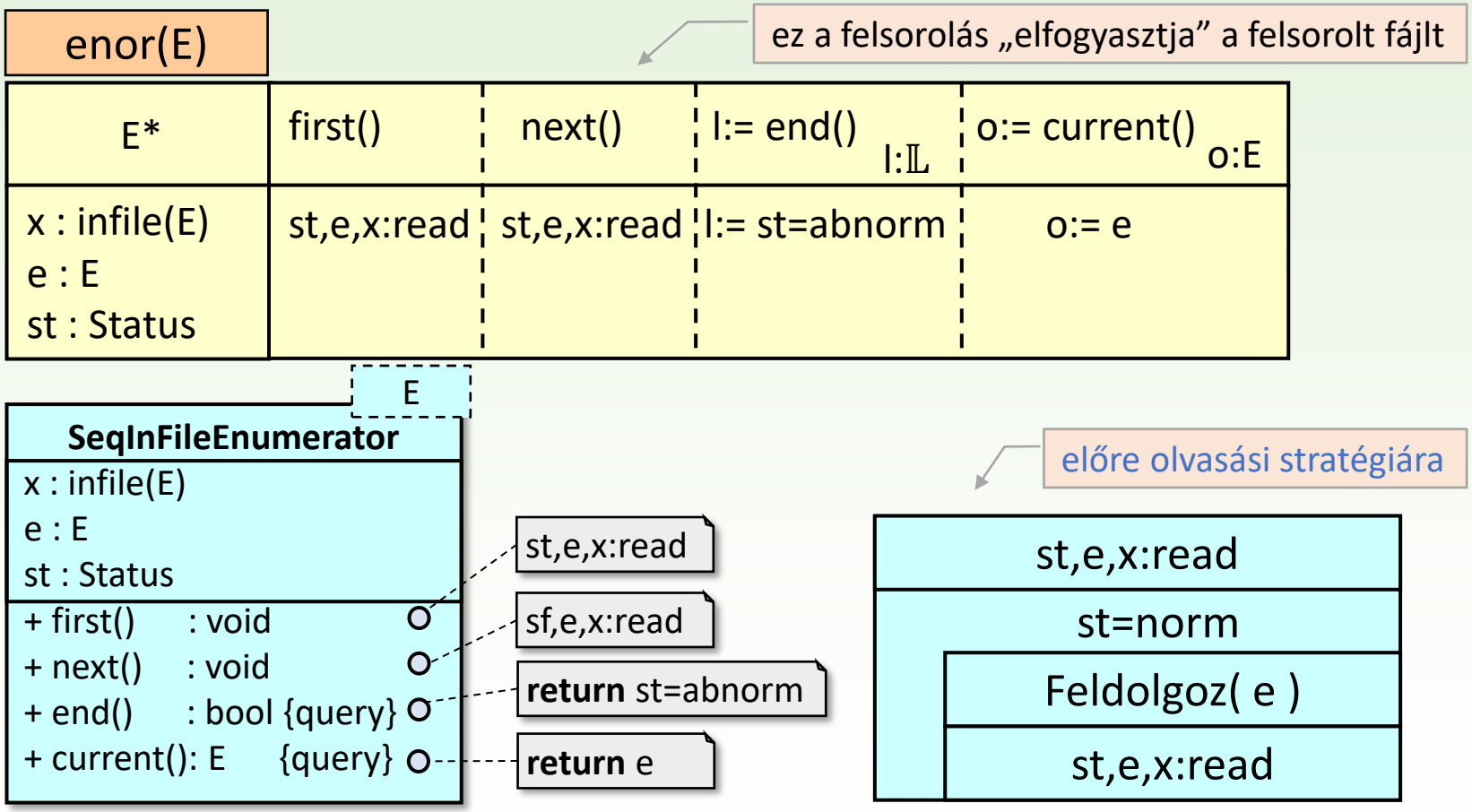
- A szekvenciális inputfájl elnevezést olyan sorozatra értjük, amelynek mindig az első elemét lehet kiolvasni.
- Az olvasás egy st, e, x := read(x) értékadás, amelyet az st, e, x :read szimbólummal rövidítünk. Ebben
  - x : **infile(E)** a szekvenciális inputfájlra hivatkozó változó
  - e : **E** a fájl elejéről kiolvasott (leszakított) elemet tartalmazó változó
  - st : **Status**={abnorm, norm} az olvasás sikerességét jelző változó

- A szekvenciális outputfájl elnevezést egy olyan sorozatra használjuk, amelynek végéhez lehet új elemet hozzáírni, és kezdetben az x :=<> értékadással üresre lehet állítani.
- Az írás egy x := write(x, e) értékadás, amelyet az x : write(e) jelöl majd:
  - x : **outfile(E)** a szekvenciális outputfájlra hivatkozó változó
  - e : **E** a fájl végéhez hozzáírt elem

x := x ⊕ <e>

Szekvenciális inputfájl felsorolója

E-beli értékeket tartalmazó szekvenciális inputfájl elemeinek felsorolása



Fájlfeldolgozásos feladatok

- A gyakorlatban sokszor találkozunk olyan (ún. kollatálási) 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.
- Ezeknél a feladatoknál az inputfájlt sokszor elemenként dolgozzuk fel:
  - másolás
  - kiválogatás
  - szétválogatás
- Ezeket a feladatokat az összegzés programozási tételére vezethetjük vissza úgy, hogy a szekvenciális inputfájl felsorolását használjuk.

Összegzés fájlkezeléshez

Általános összegzés

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

Ef = ($t = t_0$)

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

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

Fájlfeldolgozás

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

Ef = ($x = x_0$)

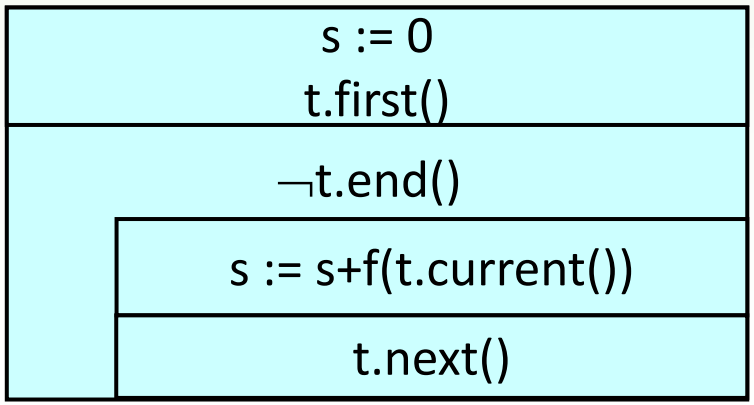
Uf = ($y = \bigoplus_{e \in x_0} f(e)$)

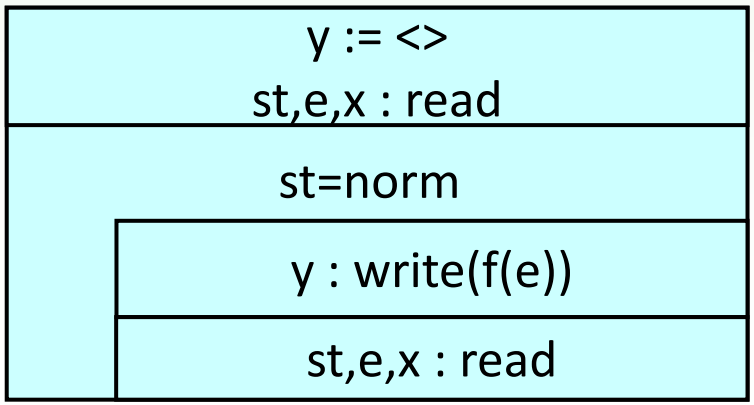
$f : E \to F^*$
$\bigoplus : F^* \times F^* \to F^*$
$<> \in F^*$ neutr. elem

Összegzés:

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

előreolvasási technika





1.Feladat

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

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

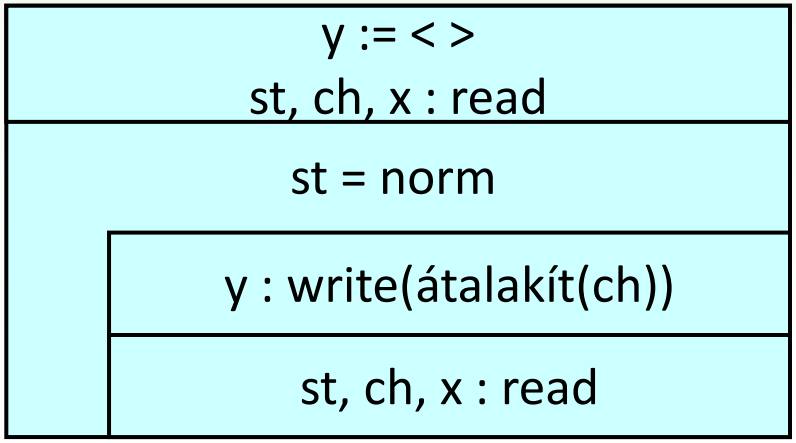
Ef = ($x = x_0$)

Uf = (y = $\bigoplus_{ch \in x_0}$ <átalakít(ch)>)

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

- Az összegzés teszteléséhez vizsgálni kell
  - a felsorolót
    - 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 az outputfájlt hossza meg fog egyezni az inputfájléval
- Ezeken kívül ellenőrizni kell a konverziót.

felsoroló szerint	hosszra:	x = <>, <a>, <ab>, <…>	→	y = <>, <a>, <ab>, <…>
	elejére:	x = <ab>, <áé>	→	y = <ab>, <ae>
	végére:	x = <ab>, <áé>	→	y = <ab>, <ae>
átalakítás szerint	ék-es mgh:	x = <ááíöőúüű>	→	y = <aeioouuu>
	ék-tlen mgh:	x = <aeioouuu>	→	y = <aeioouuu>
	msh:	x = <bsmnz>	→	y = <bsmnz>

C++ nyelvi elemek

- Szöveges állomány hátterű szekvenciális inputfájlt ifstream típusú adatfolyam objektummal reprezentáljuk.
- A C++ nyelv is az előre olvasási stratégiát támogatja a fájl olvasásánál.
- Az st, ch, x : read karakterenkénti olvasás művelet megvalósításai:
  - x >> ch - nem olvassa be az elválasztó jeleket (white space), átlépi azokat, kivéve, ha kikapcsoljuk ezt az automatizmust (x.unsetf(ios::skipws)).
  - x.get(ch) - minden karaktert (elválasztó jeleket is) beolvas.
- Az st==norm vizsgálatot a !x.eof() helyettesíti.

- Szöveges állomány hátterű szekvenciális outputfájlt ofstream típusú adatfolyam objektummal reprezentáljuk.
- Az x : write(e) karakterenkénti írás művelet megvalósításai:
  - x << ch
  - x.put(ch)

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;
}
```

rövid változat

```
x.get(ch);
while(!x.eof()){
	y << transform(ch);
	x.get(ch);
}
```

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!

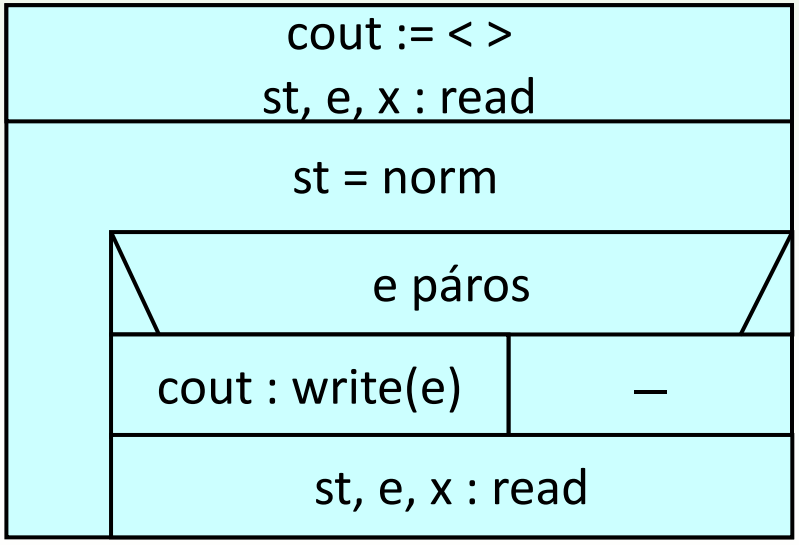
A = (x:infile($\mathbb{Z}$) , cout:outfile($\mathbb{Z}$))

Ef = ($x = x_0$)

Uf = (cout = $\bigoplus_{e \in x_0 \text{ e páros}} \langle e \rangle$)

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

- Vizsgálni kell
  - a felsorolót
    - felsorolás hossza: 0, 1, 2, több hosszú csupa páros
    - felsorolás eleje, vége: páros ill. páratlan elemek elől és hátul
  - a terheléses teszt most sem érdekes
  - a kiválogatás feltételeit

felsoroló szerint	hosszra:	x = <>, <2>, <22>, <…>	→	cout = <>, <2>, <22>, <…>
	elejére végére	x = <1>, <2>, <35>, <45>	→	cout = <>, <2>, <>, <4>
		x = <1>, <2>, <35>, <36>	→	cout = <>, <2>, <>, <6>
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;
}
```

Az elválasztójelek átlépése után az e típusának megfelelő értéket olvas.

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

Használjuk !x.eof() helyett a !x.fail()-t, mert ez 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 = ($y = \bigoplus_{\substack{dx \in x_0 \\ dx.pld=0}} \langle(dx.azon, dx.szerző, dx.cím)\rangle$

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

Algoritmus

Két összegzés közös felsorolóval:

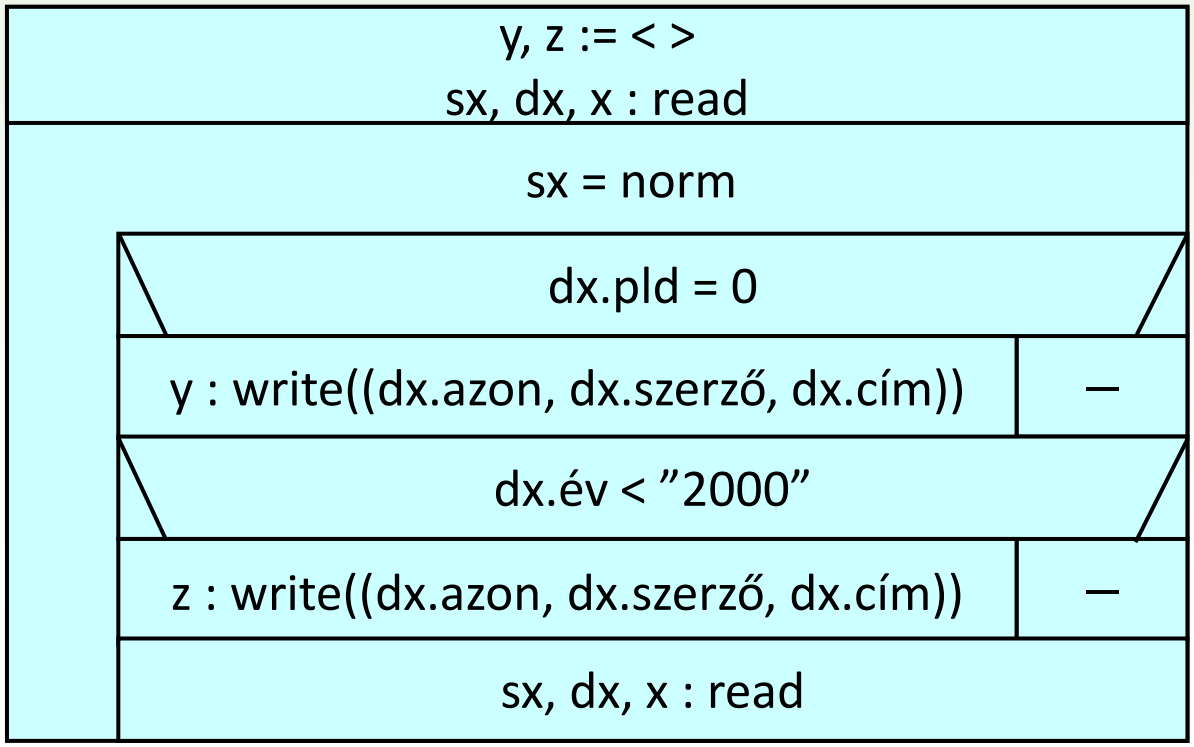
t:enor(E) ~ x:infile(Könyv), sx,dx,x : read

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

felsoroló szerint	hosszra:	0, 1, 2, ill. több minden feltételnek megfelelő könyv	→	minden könyv megjelenik mindkét outputon
	elejére és végére:	Elől, ill. hátul olyan könyv, amely a feltételek közül egyiknek, másiknak, mindkettőnek, vagy egyiknek sem felel meg (2×4 eset).	→	Minden esetben 2-2 outputfájl keletkezik, amelynek elején, ill. végén az adott esettől függően jelenik meg az inputfájl első, ill. utolsó eleme
szétválogatás feltételei szerint	Vegyesen olyan könyvek, amelyek a feltételek közül egyiknek, másiknak, mindkettőnek, vagy egyiknek sem felelnek meg.		→	csak a megfelelő könyvek jelennek meg az egyes outputfájlokban

Saját osztályok nélkül

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

int main()
{
        ifstream x("inp.txt");
        if (x.fail() ) { cout << … ; exit(1) }
        ofstream y("out1.txt");
        if (y.fail() ) {cout << … ; exit(1) }
        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;
}
```

#include <cstdlib>

```
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 függvény

```
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;
}
```

sorokba tördelt, szigorúan pozícionált inputfájl

karakterláncot számmá alakít

rész-sztring

sort olvas

C stílusú karakterláncá alakít át egy sztringet

logikai értéket ad vissza

write függvény

```
12 J. K. Rowling    Harry Potter II.
15 A. A. Milne      Micimackó
17 Gárdonyi Géza    A láthatatlan ember
25 Fekete István    Zsellérek
```

sorokba tördelt,
szigorúan pozícionált inputfájl

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

kiírás formátumozása
#include <iomanip>

Saját 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 er ) {
        if( er==Stock::FILE_ERROR ) cout << ... ;
        exit(1);
    }catch( Result::Errors er ) {
        if( er==Result::FILE_ERROR ) cout <
        exit(1);
    }
    return 0;
}
```

```
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;
};
```

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

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

4.Feladat

A hallgatóknak egy kurzus számonkérései során kapott eredményeit egy szöveges állományban helyeztük el. A hallgatók adatai külön sorokban találhatók, szóközökkel vagy tabulátorjelekkel elválasztva:

- neptun-kód (6 számjegy),
- „+” és „-” -ok összefüggő (szóközökkel sem elválasztott), nem üres sztringje
- a négy darab zárthelyinek (a legelső az első géptermi) jegye,
- a beadandók összesített jegye.

Határozzuk meg azon hallgatók félévvégi átlagos jegyét (az utolsó két zh jegy duplán számít), akik kaphatnak jegyet (a „+” és „-” -ok összege nem negatív, és a zh-k – az első géptermi kivételével – nem elégtelenek)!

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

Elemzés

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 = ($x = x_0 \wedge \forall i \in [1 \,..\, |x|] : |x_i.\text{neptun}|=6 \wedge |x_i.\text{pm}|>0$)

Uf = (y = $\bigoplus_{dx \in x_0}$ <(dx.neptun, átlag(dx.jegyek))>)

nemneg(dx.pm) $\wedge$ megfelelt(dx.jegyek)

$$\text{nemneg(dx.pm)} = \sum_{\substack{i=1 \\ \text{dx.pm[i]='–'}}}^{|\text{dx.pm}|} 1 \quad \leq \quad \sum_{\substack{i=1 \\ \text{dx.pm[i]='+'}}}^{|\text{dx.pm}|} 1$$

$$\text{megfelelt(dx.jegyek)} = \forall_{i=2}^{5} (\text{dx.jegyek[i]} > 1)$$

$$\text{átlag(dx.jegyek)} = \left(\sum_{i=1}^{3} \text{dx.jegyek[i]} + 2 \cdot \sum_{i=4}^{5} \text{dx.jegyek[i]} \right) / 7$$

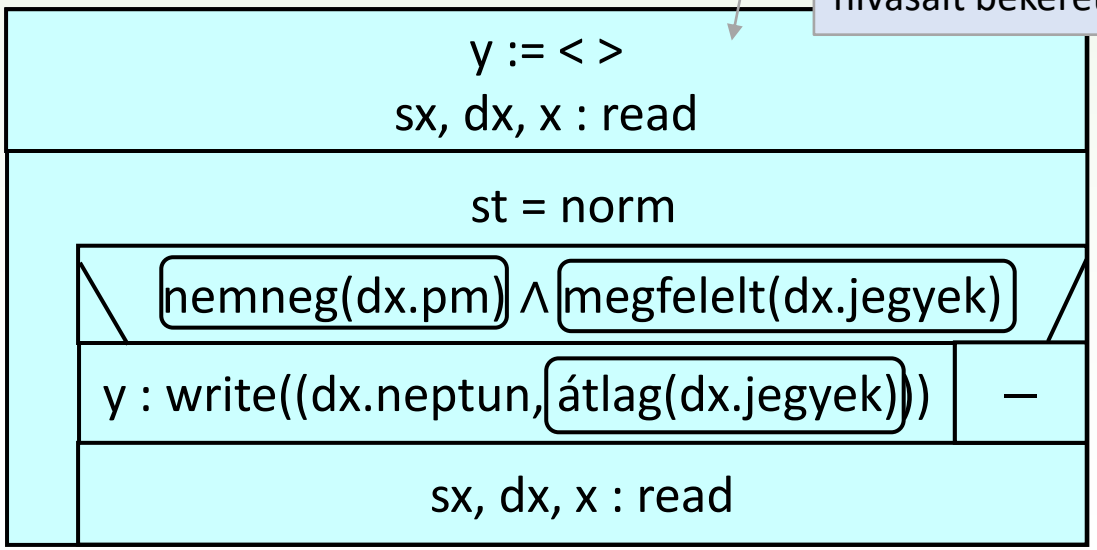
Tervezés: főprogram

y := $\bigoplus_{dx \in x_0}$ <(dx.neptun, átlag(dx.jegyek))>
nemneg(dx.pm) ∧ megfelelt(dx.jegyek)

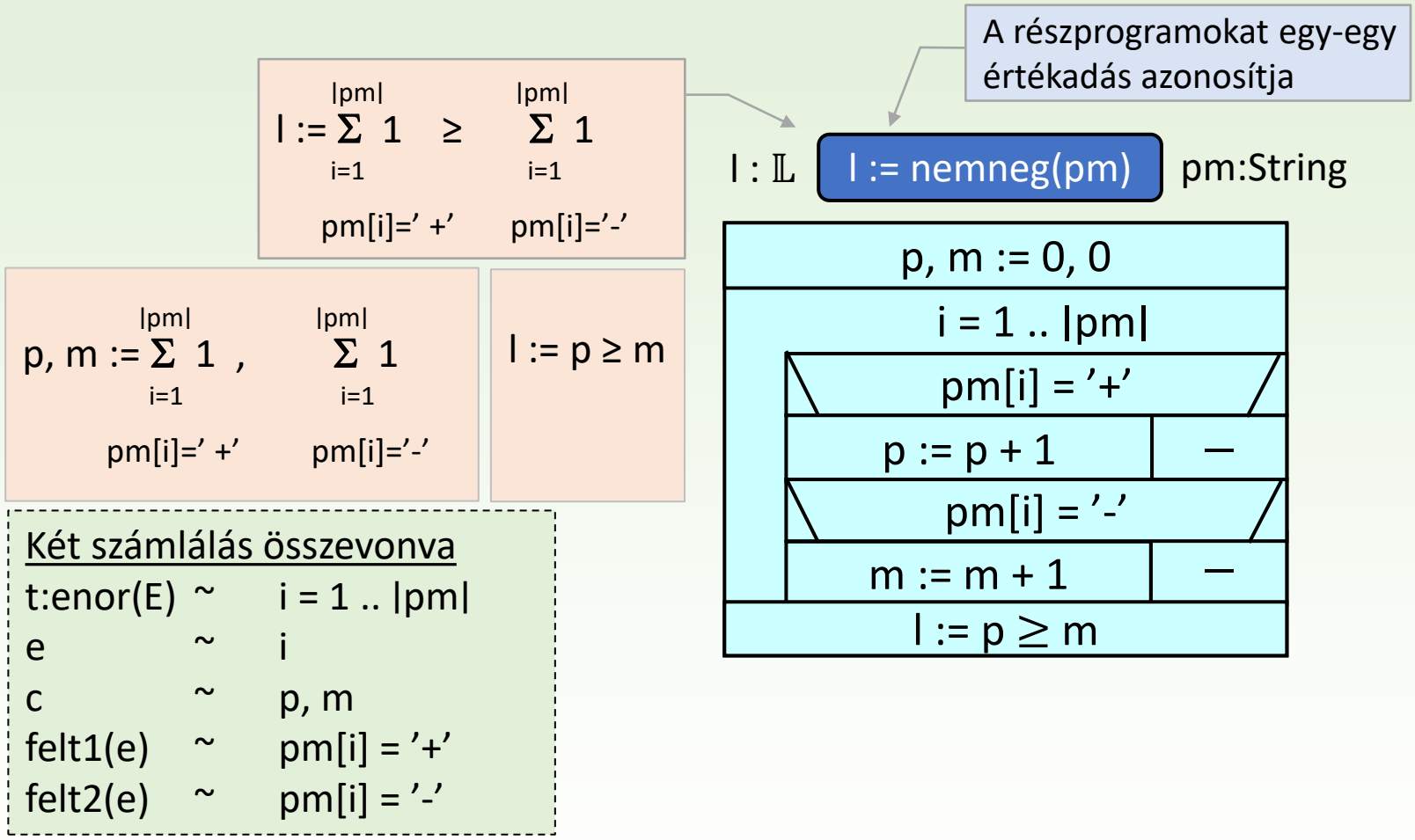
Összegzés:

t:enor(E)	~	x:infile(Hallgató)
		sx,dx,x : read
e	~	dx
f(e)	~	ha nemneg(dx.pm) ∧ megfelelt(dx.jegyek)
		akkor <(dx.neptun, átlag(dx.jegyek))>
H,+,0	~	Értékelés*, ⊕, <>

a részfeladatotokat megoldó részprogramok függvényszerű hívásait bekeretezzük



Alprogramok



Alprogramok

$$l := (\forall_{i=2}^{5}\ jegyek[i]>1)$$

$l : \mathbb{L}$ **l := megfelelt(jegyek)** jegyek:$\{0..5\}^5$

l, i := true, 2
$l \wedge i \leq 5$
l := jegyek[i] > 1
i := i + 1

Opt. lineáris keresés:

t:enor(E)	~	i = 2 .. 5
e	~	i
felt(e)	~	jegyek[i]>1

$$a := (\Sigma_{i=1}^{3}\ jegy[i] + 2 \cdot \Sigma_{i=4}^{5}\ jegy[i]) / 7$$

$a : \mathbb{R}$ **a := átlag(jegyek)** jegyek:$\{0..5\}^5$

s := 0
i = 1 .. 3
s := s + jegyek[i]
i = 4 .. 5
s := s + 2 · jegyek[i]
a := s / 7

Két összegzés egymás után:

t:enor(E)	~	i = 1 .. 3, i = 4 .. 5
e	~	i
f(e)	~	jegyek[i], 2 · jegyek[i]
H,+,0	~	$\mathbb{N}$, +, 0

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, ha az első és utolsó kaphat jegyet
terhelés:	nem kell
a cond() és f() vizsgálata:	lásd alább

Plusz-mínuszok számlálása:

a felsoroló hossza szerint:	1, 2, több kizárólag '+'-t elérő hallgató
a felsorolás eleje/vége:	az első és utolsó karakter 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 (fix 4, a legelső jegy nem számít))
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 (fix 5)
a felsorolás eleje/vége:	csak az elején és csak az végén 2-es, a többi 5-ös
terhelés:	nem kell

C++ program

```
bool nonnegative(const string &pm );
bool sufficient(const vector<int> &marks );
double  average(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 (nonnegative(dx.pm) && sufficient(dx.marks)) {
                        Grade dy(dx.neptun, average(dx.marks));
                        y.write(dy);
                  }
            }
      }catch( InpFile::Errors er ) {
            if( er==InpFile::FILE_ERROR ) cout << … ; exit(1);
      }catch( OutFile::Errors er ) {
            if( er==OutFile::FILE_ERROR ) cout << … ; exit(1);
      }
      return 0;
}
```

Segédfüggvények

```
bool nonnegative(const string &pm ) {
	int p, m; p = m = 0;
	for(unsigned int i = 0; i<pm.size(); ++i){
		if(pm[i]=='+') ++p;
		if(pm[i]=='-') ++m;
	}
	return m<=p;
}
```

```
bool sufficient(const vector<int> &marks) {
	bool l = true;
	for(unsigned int i=1; l && i<marks.size(); ++i){
		l=marks[i]>1;
	}
	return l;
}
```

```
double average(const vector<int> &marks) {
	int s = 0;
	unsigned int i = 0;
	for( ; i<= 2; ++i){
		s += marks[i];
	}
	for( ; i<= marks.size(); ++i){
		s += 2*marks[i];
	}
	return double(s) / 7.0;
}
```

Hátrány:
Az average() és a sufficient() is végig nézi a jegyeket.

Hátrány:
Feltételezi, hogy van 5 darab jegy

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;
};
```

```
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 Grade {
        std::string neptun;
        double mark;
        Grade(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 Grade &dy) {
                f.setf(std::ios::fixed);
                f.precision(2);
                f << dy.neptun << std::setw(7) << dy.mark << std::endl;
        }
private:
    std::ofstream f;
};
```

konstruktor a struktúrához

Módosítás

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
```

változó tagból álló, változó hosszú nevek

```
int main(){
    try{
        InpFile x("input.txt");
        OutFile y("output.txt");
        Student dx;
        Status sx;
        while(x.read(dx,sx)) {
            if (dx.has) {
                Grade 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 csak a megoldáshoz szükséges adatokból áll. Ezeket a szöveges állomány egyes sorai alapján a beolvasásnál kell kiszámolni.

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;
            }

            e.has = nonnegative(str) && sufficient_average(in, e.result);
      } else st=abnorm;
      return norm==st;
}
```

cél az e kitöltése az aktuális sor 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

tartalmazza az osztályzatokat

A nonnegative() és az average() az InpFile osztály privát metódusai

Segédfüggvények

```
bool InpFile::sufficient_average(istringstream &in, double &avr)
{
	int mark;
	in >> mark; if(in.fail()) return false;
	int s = mark;
	int i = 1;
	while( in >> mark ) {
		if ( mark<=1 ) return false;
		++i;
		s+= (i<=3) ? mark : 2*mark;
	}
	if(5!=i) return false;
	avr = double(s) / 7.0;
	return true;
}
```

Összevontuk az average() és sufficient() függvényeket

```
bool InpFile::nonnegative(const string &pm)
{
	int p, m;
	p = m = 0;
	for(unsigned int i = 0; i<pm.size(); ++i){
		if(pm[i]=='+') ++p;
		if(pm[i]=='-') ++m;
	}
	return m<=p;
}
```

Olvasás szöveges állományokból

f : infile(E)	**st, data, f : read**	**st = norm**
E ≡ char // karakterek elválasztás nélkül	f.get(data);	!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 line;
	getline(f, line);
	istringstream is(line);
	is >> …
		!f.fail()