

An aerial photograph of Budapest, Hungary, showing the city's architecture and the Danube River. A semi-transparent white rectangular box is centered over the image, containing the title text.

Programozási alapismeretek 9. előadás



A mai előadás foglalatja, mondásokkal



➤ „Tévedni emberi (dolog).”

- Seneca





A mai előadás foglalatja, mondásokkal



➤ „Tévedni emberi (dolog).”

- Seneca



➤ „Más szemében meglátja a szálkát, a
magáében a gerendát sem veszi észre.”

- Mt. 7,3/Lk 6,41



A mai előadás foglalatja, mondásokkal



- „Tévedni emberi (dolog).”

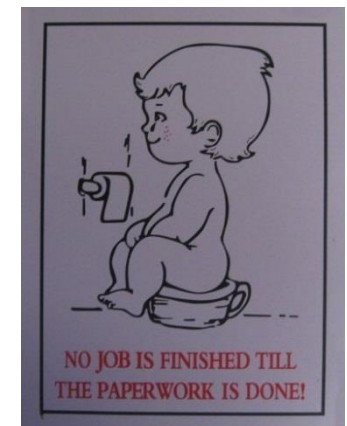
- Seneca



- „Más szemében meglátja a szálkát, a magáéban a gerendát sem veszi észre.”

- Mt. 7,3/Lk 6,41

- „Minden feladat papírmunkával zárul”





A mai előadás foglalatja, mondásokkal



- „Tévedni emberi (dolog).”

- Seneca

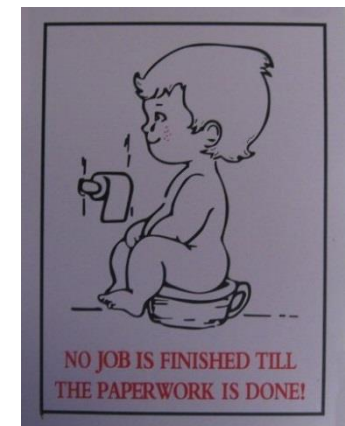


- „Más szemében meglátja a szálkát, a magáéban a gerendát sem veszi észre.”

- Mt. 7,3/Lk 6,41

- „Minden feladat papírmunkával zárul”

– így a mai előadás is.



Tartalom



➤ Tesztelés

- fogalmak + elvek + módszerek
- technika: futtatás adatfájlal – C++

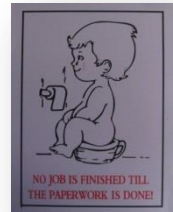
➤ Hibakeresés

➤ Hibajavítás

➤ Dokumentálás +

szövegszerkesztési minimum

➤ Programkészítési elvek

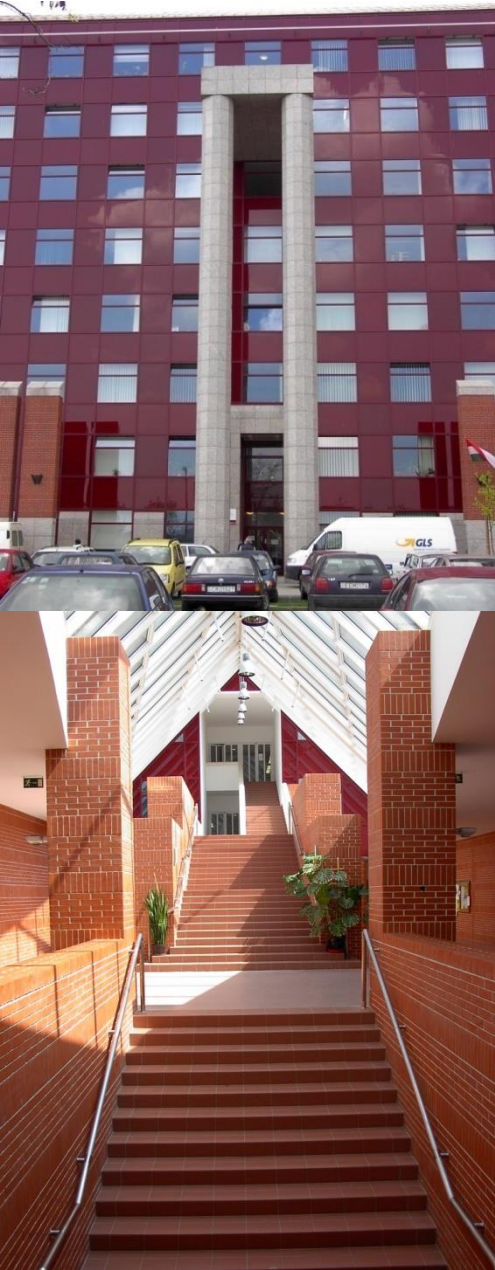


Tesztelés fogalmak

Tesztelési fogalmak:

- Teszteset = **bemenet** + **kimenet**
- Próba = teszteset-halmaz
- Jó teszteset: nagy valószínűséggel felfedetlen hibát mutat ki
- Ideális próba: minden hibát kimutat
- Megbízható próba: nagy valószínűséggel minden hibát kimutat

A „hiba” helyett jobb lenne „problémát” mondani, mivel a tesztelés nemcsak a hibakeresés, hanem a hatékonyságvizsgálat eszköze is.



Tesztelés elvek



Tesztelési elvek:

- Érvényes (megengedett) és érvénytelen (hibás) bemenetre is kell tesztelni.
- Rossz a meg nem ismételhető teszteset. (Ez ritkán a tesztelés, inkább a tesztelendő program vonása.)
- Minden teszteset által nyújtott információt maximálisan ki kell használni (a következő teszt-esetek kiválasztásánál).
- Csak más (mint a szerző) tudja jól tesztelni a programot.



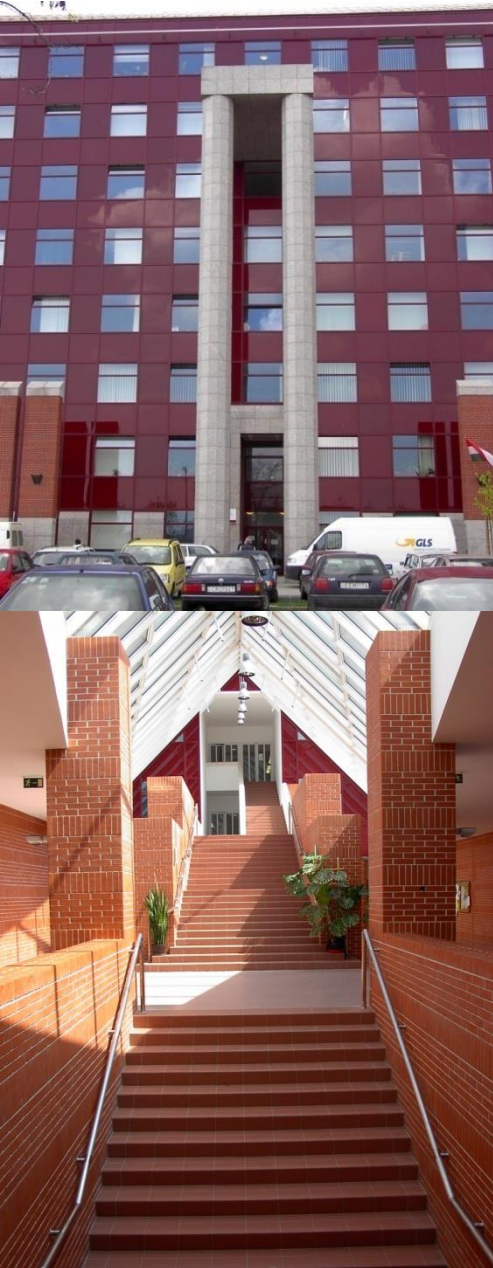
Tesztelés módszerek

Tesztelési módszerek:

- Statikus tesztelés: a programszöveget vizsgáljuk, a program futtatása nélkül.
- Dinamikus tesztelés: a programot futtatjuk különböző bemenetekkel és a kapott eredményeket vizsgáljuk.

A tesztelés eredménye:

- hibajelenséget találtunk;
- nem találtunk –még– hibát.



Statikus tesztelés

➤ KódelLENŐRZÉS:

- algoritmus ↔ kód megfeleltetés
- algoritmus + kód elmagyarázása másnak

➤ Szintaktikus ellenőrzés:

- fordítóprogram esetén automatikus
- értelmező esetén sok futtatással



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás-keresés:

- o felhasználatlan változó/érték

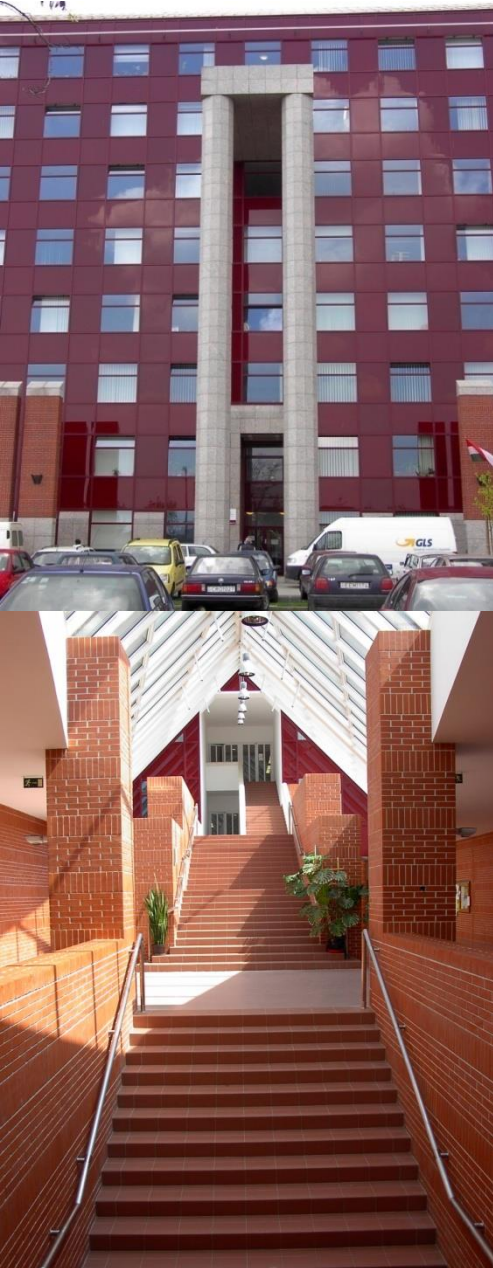
```
i=1;  
for (i=2; ...)  
{ ... }
```

- o „gyanús” változó-használat

```
i=1;  
for (int i=2; ...)  
{ ... i ... }  
... i ...
```

- o „identikus” transzformáció

```
i=1*i+0
```



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás-keresés (folytatás):

- o inicializálatlan változó

```
int n; //meggondolatlan kódolása
int k[n]; //a spec.-nak/adatleírásnak
```

- o definiálatlan (?) értékű kifejezés:

```
int n; ← globális n
...
int fv()
{
    for (int n=0; n<9; ++n) ← lokális n
    {
        ... n ... ; ← a lokális n felhasználása
    }
    return ...n...; ← a globális n-et tartalmazó formula
}
```



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondáskeresés (folytatás):

- o érték nélküli függvény

szintaktikusan **hibás** a C++ nyelvben :

```
int fv ()  
{  
    ...  
    return; ← ... error: return-statement with no value, in  
              function returning 'int' |  
}
```

... de az alábbi csak **figyelmeztetés**t vált ki:

```
int fv ()  
{  
    ...  
} ← ... warning: control reaches end of non-void function |
```



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás-keresés (folytatás):

- azonosan igaz/**hamis** feltétel

`(N>1 || N<100) / (N<1 && N>maxN)`

Talán egy **beolvasás-ellenőrzés** kódolásakor jött ki.

- végtelen számlálós ciklus

```
for (int i=0; i<N; ++i)
{
    ...
    --i; ← nem okoz fordítási hibaiüzenetet
}
```

Talán egy **while**-os ciklus átírása során jött ki.



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás-keresés (folytatás):

- o pontatlan ciklus-szervezés

```
for (int i=0; i<N; ++i)
{
    ...
    ++i; ← nem okoz fordítási hibaiüzenetet
}
```

Egy **while**-os ciklus következtelen átírása során jöhetett ki.



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondáskeresés (folytatás):

- o végtelen feltételes ciklus ($i < N$ feltételű ciklusban sem i , sem N nem vagy „szinkronban” változik)

```
i=1;  
while (i<=N)  
{  
    ...  
    i+=1; ← talán i+=1 akart lenni?  
}
```

- o konstans értékű, (bár) változókat tartalmazó kifejezés

```
y=sin(x)*cos(x)-sin(2*x)/2
```



Dinamikus tesztelés

Tesztelési módszerek:

- **Fekete doboz** módszerek (← *nincs kimerítő bemenet* – nem lehet minden lehetséges bemenetre kipróbálni): a teszteseteket a program **specifikációja** alapján **optimálisan** választjuk.
- **Fehér doboz** módszerek (← *nincs kimerítő út* – nem lehet minden végrehajtási sorrendre kipróbálni): a teszteseteket a **program struktúrája** alapján **optimálisan** választjuk.





Dinamikus tesztelés: fekete doboz módszerek

- **Ekvivalencia-osztályok módszere:** a bemeneteket (vagy a kimeneteket) soroljuk olyan osztályokba, amelyekre a program várhatóan egyformán működik; ezután osztályonként egy tesztesetet válasszunk!
- **Határeset elemzés módszere:** az ekvivalencia-osztályok határáról válasszunk tesztesetet!

Dinamikus tesztelés: fekete doboz módszerek

Specifikáció:

- Bemenet: $N \in \mathbb{N}$
- Kimenet: $O \in \mathbb{N}$, $\text{Van} \in \mathbb{L}$
- Előfeltétel: $N > 1$
- Utófeltétel: $\text{Van} = \exists i (2 \leq i < N) : i \mid N \text{ és } \text{Van} \rightarrow 2 \leq O < N \text{ és } O \mid N \text{ és } \forall i (2 \leq i < O) : i \nmid N$

- **Feladat:** Adja meg egy N természetes szám valódi (1-től és önmagától különböző) osztóját!

Ekvivalencia osztályok (bemenet alapján):

1. N prímszám: 3
2. N -nek egy(-féle) valódi osztója van: $25 = 5 * 5$
3. N -nek több, különböző valódi osztója is van: $77 = 7 * 11$
4. N páros
5. *N bármi, ami nem természetes szám*

Érvényes adatokra

Érvénytelen adatokra

Dinamikus tesztelés: fekete doboz módszerek

Specifikáció:

- > Bemenet: $N \in \mathbb{N}$
- > Kimenet: $O \in \mathbb{N}, \text{ Van} \in \mathbb{L}$
- > Előfeltétel: $N > 1$
- > Utófeltétel: $\text{Van} = \exists i (2 \leq i < N) : i \mid N \text{ és } \text{Van} \rightarrow 2 \leq O < N \text{ és } O \mid N \text{ és } \forall i (2 \leq i < O) : i \nmid N$

- **Feladat:** Adja meg egy N természetes szám valódi (1-től és önmagától különböző) osztóját!

Ekvivalencia osztályok (bemenet alapján):

1. N prímszám: 3
2. N -nek egy(-féle) valódi osztója van: $25 = 5 * 5$
3. N -nek több, különböző valódi osztója is van: $77 = 7 * 11$
4. N páros $\subset 2. \cup 3.$
5. *N bármi, ami nem természetes szám*

Érvényes adatokra

Érvénytelen adatokra

Dinamikus tesztelés: fehér doboz módszerek

Kipróbálási stratégiák:

- utasítás lefedés: minden utasítást legalább egyszer hajtsunk végre!
- feltétel lefedés: minden feltétel legyen legalább egyszer igaz, illetve hamis!
- részfeltétel lefedés: minden részfeltétel legyen legalább egyszer igaz, illetve hamis!

Tesztadat-generálás:

- automatikus tesztbemenet-előállítás

Ezekhez továbbiakat itt talál:
<http://people.inf.elte.hu/szlaui/PrM1felev/Pdf/PrTea7.pdf>
1.3.2. fejezetében

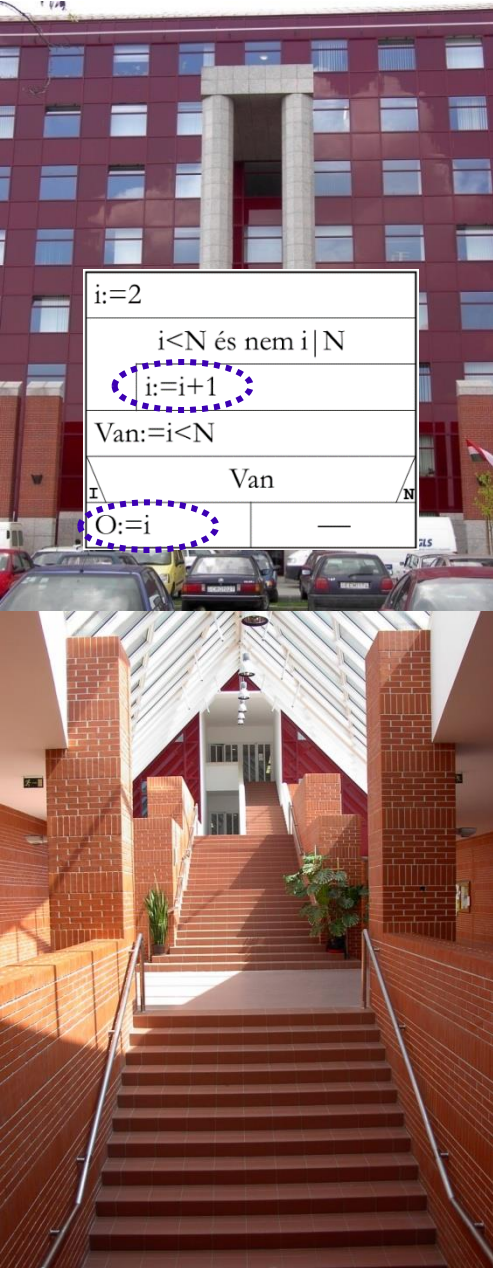


Dinamikus tesztelés: fehér doboz módszerek

➤ **Feladat:** Egy N természetes szám valódi (1-től és önmagától különböző) osztója...

Utasítás lefedés:

- $i:=i+1$ végrehajtandó: $N=3$
- $O:=i$ végrehajtandó: $(\leftarrow Van=Igaz) \ N=4$



Dinamikus tesztelés: fehér doboz módszerek



$i:=2$
$i < N$ és nem $i \mid N$
$i:=i+1$
$\text{Van}:=i < N$
Van
$O:=i$



➤ **Feladat:** Egy N természetes szám valódi (1-től és önmagától különböző) osztója...

Utasítás lefedés:

- $i:=i+1$ végrehajtandó: $N=3$
- $O:=i$ végrehajtandó: $(\leftarrow \text{Van}=\text{Igaz})$ $N=4$

Feltétel lefedés:

- Ciklusfeltétel igaz: $N=3$
- Ciklusfeltétel hamis: $N=2$ (be sem lép)
- Elágazásfeltétel igaz: $(\leftrightarrow \text{Van}=\text{Igaz})$ $N=4$
- Elágazásfeltétel hamis: $(\leftrightarrow \text{Van}=\text{Hamis})$ $N=2$

Speciális tesztelések

- Biztonsági teszt: ellenőrzések vannak?
- Hatékonysági teszt

Speciális programokhoz

- Funkcióteszt: tud minden funkciót?
- Stressz-teszt: gyorsan jönnek a feldolgozandók, ...
- Volumen-teszt: sok adat sem zavarja



Futtatás adatfájllal (C++)

Elv:

A standard input/output átirányítható fájlba. Ekkor a program **fájlt** használ az inputhoz és az outputhoz. Következmény: **szerkezetileg a konzol inputtal/outputtal megegyező kell legyen / lesz a megfelelő fájl.**

Figyelem! Ha van outputfájl, akkor a kérdés szövege is abban „jelenik meg”.

„Technika”:

A lefordított kód mögé kell paraméterként írni a megfelelő fájlok nevét.

prog.exe >>outputfájl
outputfájl**hoz** írás!

prog.exe <inputfájl >outputfájl

Nyereség:

Kényelmes és adminisztrálható tesztelés.

Futtatás adatfájllal (C++)



Demo:

1. *Készítsünk néhány bemeneti adatot tartalmazó fájlt (a konzol inputnak megfelelő szerkezetben)!*
2. *Futtassuk ezekkel az előbb elmondottak szerint:*

```
1. prog.exe <1.be >1.ki  
2. prog.exe <2.be >2.ki  
3. ....
```

3. *Ellenőrizzük a kimeneti fájlok tartalmát: olyan-
e, amilyennek vártuk!*

*Megjegyzés: tovább egyszerűsíthetjük a tesztelést, ha
egy batch állománnyal automatizáljuk a 2.-t!*

Valahogy így: próba₁, próba₂ ...



Futtatás adatfájllal (C++)

Demo:

1. *Készítsünk néhány bemeneti adatot tartalmazó fájlt (a konzol inputnak megfelelő szerkezetben)!*
2. *Futtassuk ezekkel az előbb elmondottak szerint:*

1. `prog.exe <1.be >1.ki`

2. `prog.exe <2.be >2.ki`

3.

3. *Ellenőrizzük a kimeneti fájlok tartalmát: olyan-
e, amilyennek vártuk!*

*Megjegyzés: tovább egyszerűsíthetjük a tesztelést, ha
egy batch állománnyal automatizáljuk a 2.-t!*

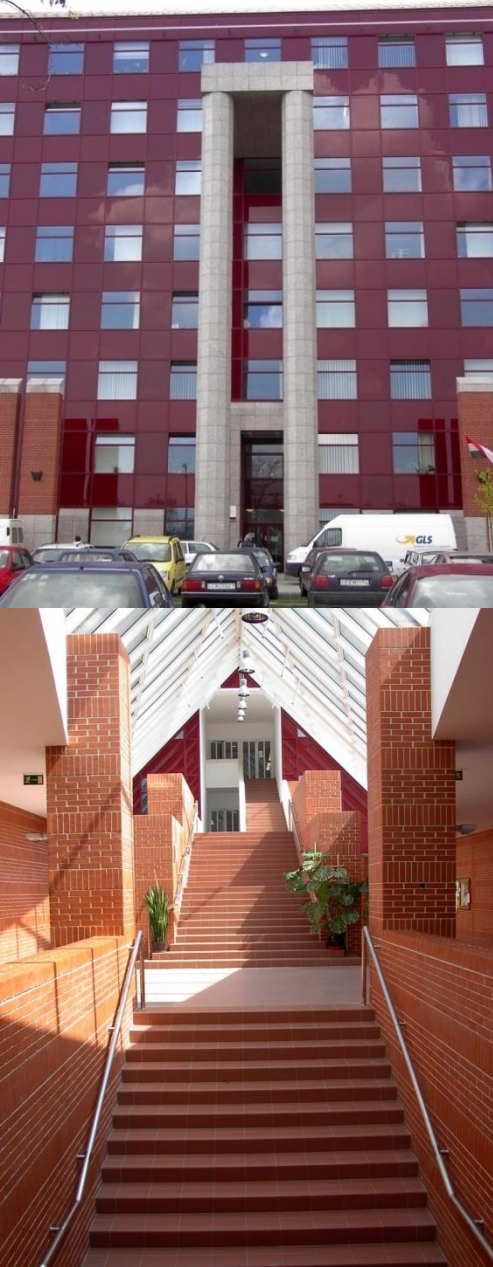
Valahogy így: próba₁, próba₂ ...



Hibakeresés

Hibajelenségek a tesztelés során...

- hibás az eredmény,
- futási hiba keletkezett,
- nincs eredmény,
- részleges eredményt kaptunk,
- olyat is kiír, amit nem vártunk,
- túl sokat (sokszor) ír,
- nem áll le a program,
- ...



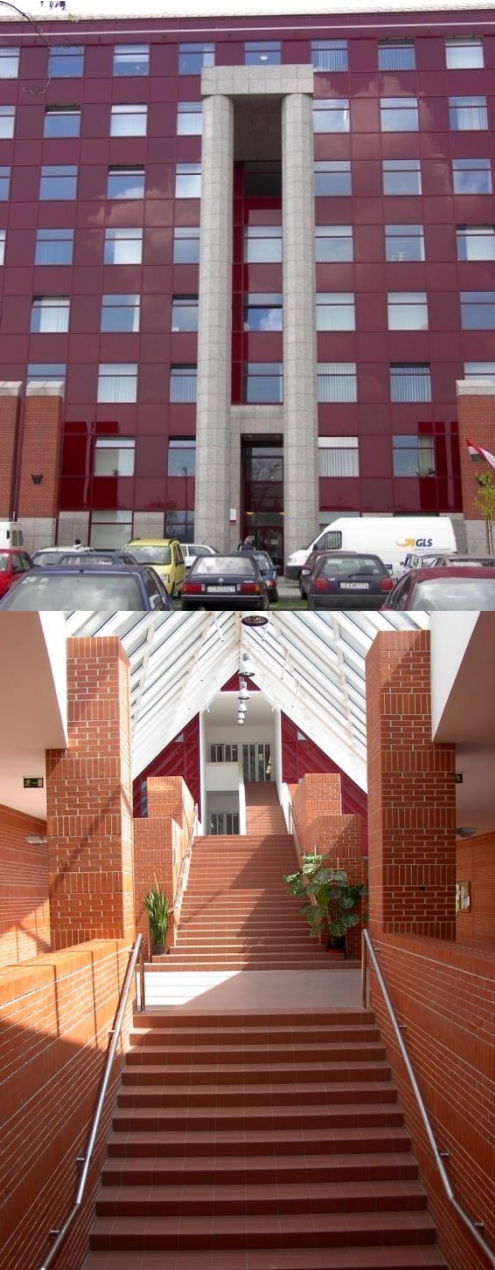
Hibakeresés

Célja:

a felfedett hibajelenség okának, helyének megtalálása.

Elvek:

- Eszközök használata előtt alapos végiggondolás.
- Egy megtalált hiba a program más részeiben is okozhat hibát.
- A hibák száma, súlyossága a program méretével nemlineárisan (annál gyorsabban!) nő.
- Egyformán fontos, hogy *miért* nem csinálja a program, amit *várunk*, illetve, hogy *miért* csinál olyat, amit nem *várunk*.
- Csak akkor javítani, ha megtaláltuk a hibát.



Hibakeresés

Hibakeresési eszközök (folytatás):

- Változó-, memória-kiírás (feltételes fordítás)
- Töréspont elhelyezése
- Lépésenkénti végrehajtás
- Adat-nyomkövetés
- Állapot-nyomkövetés (pl. paraméterekre vonatkozó előfeltételek, ciklus-invariánsok)
- Postmortem-nyomkövetés: hibától visszafelé
- Speciális ellenőrzések (pl. indexhatár: `.at(.)↔[.]`)



Hibakeresés (C++)



Hibakeresési eszközök:

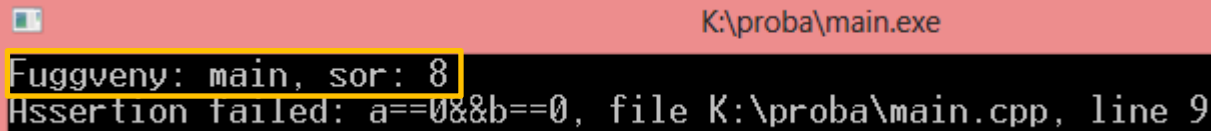
- A hiba helyének és okának kijelzése
 - automatikusan – futási hiba
 - manuálisan – standard makrókkal (`__LINE__`, `__func__`, `assert`). Pl.

```
#include <iostream>
#include <cassert> //assert-hez
using namespace std;
int main()
{
    int a,b;
    cout<<"Fuggvény: "<<__func__;//akt. függvény neve, most: main
    cout<<", sor: "<<__LINE__<<endl;//akt. sorszám, most: 8
    assert(a==0&&b==0); //egy elvárás (a==0&&b==0) ellenőrzése;
                           //nem teljesülésekor megállás, hibaüzenet
    ...
}
```


Hibakeresés (C++)



Hibakeresési eszközök:



K:\proba\main.exe
Fuggvény: main, sor: 8
Assertion failed: a==0&&b==0, file K:\proba\main.cpp, line 9

kijelzése

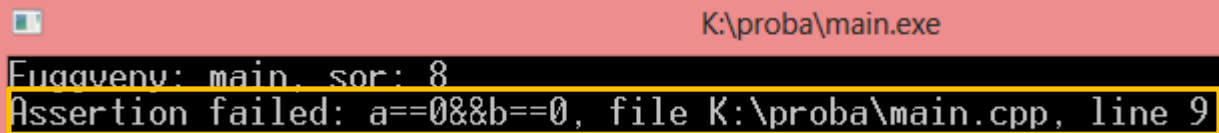
- manuálisan – standard makrókkal (`__LINE__`, `__func__`, `assert`). Pl.

```
#include <iostream>
#include <cassert> //assert-hez
using namespace std;
int main()
{
    int a,b;
    cout<<"Fuggvény: "<<__func__;//akt. függvény neve, most: main
    cout<<", sor: "<<__LINE__<<endl;//akt. sorszám, most: 8
    assert(a==0&&b==0); //egy elvárás (a==0&&b==0) ellenőrzése;
                           //nem teljesülésekor megállás, hibaüzenet
    ...
}
```

Hibakeresés (C++)

Hibakeresési eszközök:

... kijelzése



```
K:\proba\main.exe
Fuggveny: main, sor: 8
Assertion failed: a==0&&b==0, file K:\proba\main.cpp, line 9
```

- manuálisan – standard makrókkal (`__LINE__`, `__func__`, `assert`). Pl.

```
#include <iostream>
#include <cassert> //assert-hez
using namespace std;
int main()
{
    int a,b;
    cout<<"Fuggveny: "<<__func__;//akt. függvény neve, most: main
    cout<<", sor: "<<__LINE__<<endl;//akt. sorszám, most: 8
    assert(a==0&&b==0); //egy elvárás (a==0&&b==0) ellenőrzése;
                           //nem teljesülésekor megállás, hibaüzenet
    ...
}
```

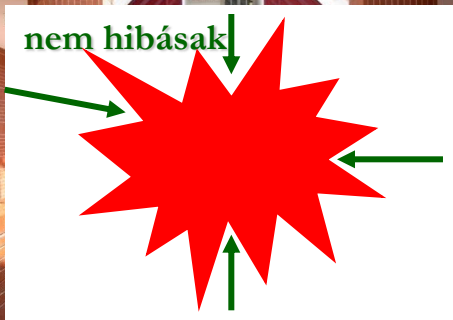
Hibakeresési módszerek

Célja:

- A **bemenetnek mely** része, amire hibásan működik a program?
- **Hol** található a **programban** a hibát okozó utasítás?

Módszerfajták:

1. Indukciós módszer (hibásak körének **bővítése**)
2. Dedukciós módszer (hibásak körének **szűkítése**)
3. Hibakeresés hibától visszafelé
4. Teszteléssel segített hibakeresés (olyan teszteset kell, amely az ismert hiba helyét fedi fel)



Hibakeresési módszerek

Példa az indukciós módszerre:



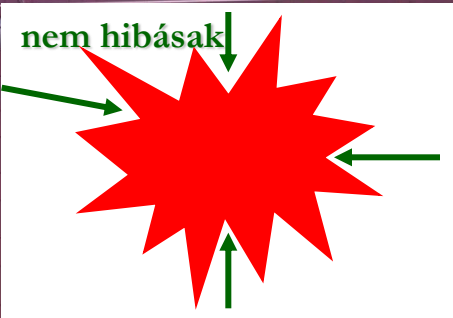
Feladat: *1 és 99 közötti N szám kiírása betűkkel*

- Tesztesetek: $N=8 \Rightarrow$ jó, $N=17 \Rightarrow$ jó, $N=30 \Rightarrow$ hibás.
- Próbáljunk a hibásakból általánosítani: tegyük fel, hogy minden 30-cal kezdődőre rossz!
- Ha beláttuk (teszteléssel), akkor próbáljuk tovább általánosítani, pl. tegyük fel, hogy minden 30 felettire rossz!
- Ha nem lehet tovább általánosítani, akkor tudjuk mit kell keresni a hibás programban.
- Ha nem ment az általánosítás, próbáljuk másképp: hibás-e minden 0-ra végződő számra!
- ...



Hibakeresési módszerek

Példa a dedukciós módszerre:



Feladat: *1 és 99 közötti N szám kiírása betűkkel*

- Tesztesetek: $N=8 \Rightarrow$ jó, $N=17 \Rightarrow$ jó, $N=30 \Rightarrow$ hibás.
- Tegyük fel, hogy minden nem jóra hibás!
- Próbáljunk a hibás esetek alapján szűkíteni: tegyük fel, hogy a 20-nál kisebbekre jó!
- Ha beláttuk (teszteléssel), akkor szűkítsünk tovább, jó-e minden 40-nél nagyobbra?
- Ha nem szűkíthető tovább, akkor megtaláltuk, mit kell keresni a hibás programunkban.
- Ha nem, szűkítsünk másképp: tegyük fel, hogy jó minden nem 0-ra végződő számra!
- ...



Hibajavítás

Célja:

a megtalált hiba kijavítása.

Elvek:

- A hibát kell javítani és nem a tüneteit.
- A hiba kijavítása a program más részében hibát okozhat (rosszul javítunk, illetve korábban elfedett más hibát).
- Javítás után a tesztelés megismételendő!
- A jó javítás valószínűsége a program méretével fordítva arányos.
- A hibajavítás a tervezési fázisba is visszanyúlhat (a módszertan célja: lehetőleg ne nyúljon vissza).



Dokumentációk

Döln szedve, ami az aktuális nagy program estén a dokumentációból elhagyható.

Fajtái:

- *Programismertető*
- Felhasználói dokumentáció
- Fejlesztői dokumentáció
- ...



Felhasználói dokumentáció

E nélkül be sem adható!

Döltén szedve, ami az aktuális nagy program estén a dokumentációból elhagyható.

Tartalma:

- **feladatszöveg** (összefoglaló és részletes is)
- futási környezet (szg.+or.+hw/sw-elmvárások)
- használat leírása (telepítés, kérdések + lehetséges válaszok,...)
- bemenő adatok, eredmények, *szolgáltatások*
- mintaalkalmazások – példafutások
- hibaüzenetek és a hibák lehetséges okai



Fejlesztői dokumentáció

E nélkül be sem adható!

Tartalma:

- **feladatszöveg**, specifikáció, *követelményanalízis*
- fejlesztői környezet (or.+fordító program, ...)
- adateleírás (feladatparaméterek reprezentálása)
- algoritmusok leírása, döntések (pl. tételekre utalás), *más alternatívák, érvek, magyarázatok*
- kód, *implementációs szabványok, ~ döntések*
- tesztesetek
- *hatékonysági mérések*
- fejlesztési lehetőségek
- **szerző**(k)

Dölni szedve, ami az aktuális nagy program estén a dokumentációból elhagyható.

Értelmesen strukturálva.

E nélkül be sem adható!

Szerző

Név: Szabó Emerencia

ETR-azonosító: SZEKAAT.ELTE

Neptun-azonosító: ESZ98A

Drótposta-cím: sze@elte.hu

Kurzus kód: IP-08PAEG/77

Gyakorlatvezető neve: Kiss-József Alfréd

Feladatsorszám: 18



Szövegszerkesztési ismeretek a dokumentációhoz

- Karakterformázás (szöveg↔program)
- Bekezdésformázás, stílusok
- Tabulátorok
- Képbeillesztés (pl. a futás bemutatásához)
- Fájlbeillesztés (pl. a kódhoz)
- Oldalformázás (pl. *előfej*, lapszámozás)
- Táblázatok (pl. struktogram, tesztek)
- Egyenletszerkesztő (pl. a specifikációhoz)
- Szervezeti diagram-készítő



Szövegszerkesztési ismeretek a dokumentációhoz



C++ kód

```
//Szerző: Szabó Emerencia
//ETR-azonosító: SZEKAAT.ELTE
//Drótposta-cím: sze@elte.hu

#include <iostream>
#include <stdlib.h>

using namespace std;

const int MaxN=100;      //tömbök maximális mérete

void beolvasas(int &n,float x[],float y[]);
void beolvasasVektor(int n,float v[]);
float skalarSzorzat(int n,const float x[],const float y[])
int maxIndex(int n,const float x[],const float y[])
//... a többi részfeladat függvényei
void kiiras(int n,const float x[],const float y[]);
void billentyureVar();
```



Szövegszerkesztési isme-



*new 2 - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

new 2

```
16 //TNyer típus definiálása:
17 struct TNyer{int bev, kia;}; //reprezentáció
18 int operator +(int _s, TNyer _x); //TNyer típus
19 //a lényegi számítás függvénye:
20 int sum(int _n, const TNyer _t[]); //_n TNyer 'összege' függvény fejeztetése
21 //Billentyűre várás:
22 void billreVar();
23
24 int main()
25 {
26     //bemenet -csak most: konstansok, így nem kell beolvasni!-:
27     const TNyer Jov[]={10,5},{5,10},{100,50}}; //jövedelem tömb értékei
28     int N=sizeof Jov / sizeof(TNyer); //aktuális elemszám
29     //kimenet -mindjárt számításal-:
30     int S=sum(N,Jov);
31     //eredménymegjelenítés:
32     cout << "Ossz jovedelem:" << S << endl;
33
34     billreVar();
35     return 0;
36 }
37
38 //TNyer típusú hozzáadás operátor definíciója:
39 int operator +(int _s, TNyer _x)
40 {
41     return _s + _x.bev - _x.kia;
42 }
```

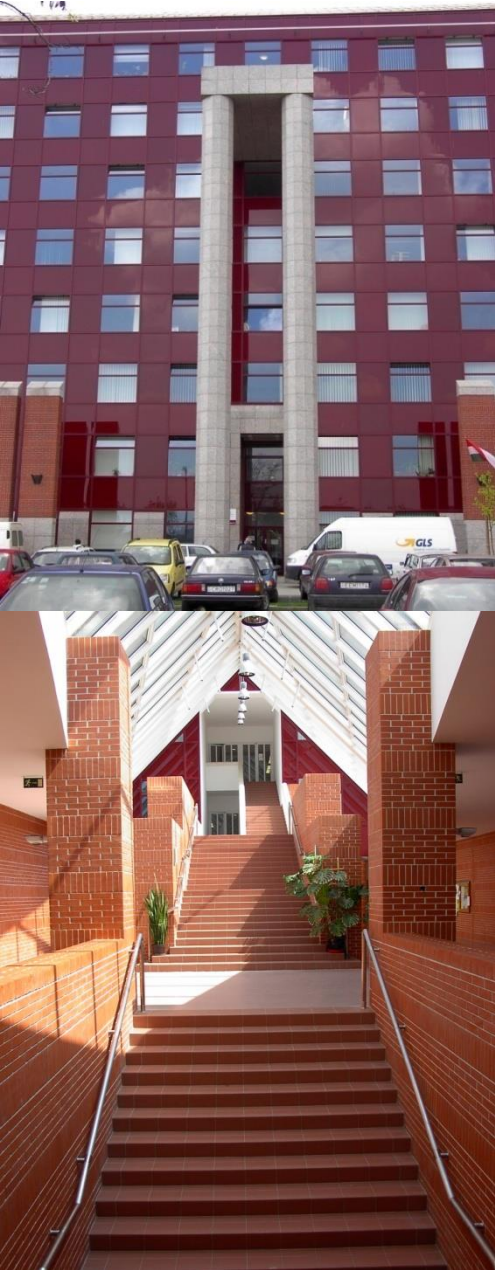
Converter
NppExport
NppFTP
Plugin Manager
Spell-Checker

Export to RTF
Export to HTML
Copy RTF to clipboard
Copy HTML to clipboard
Copy all formats to clipboard

Notepad++

Programkészítési elvek

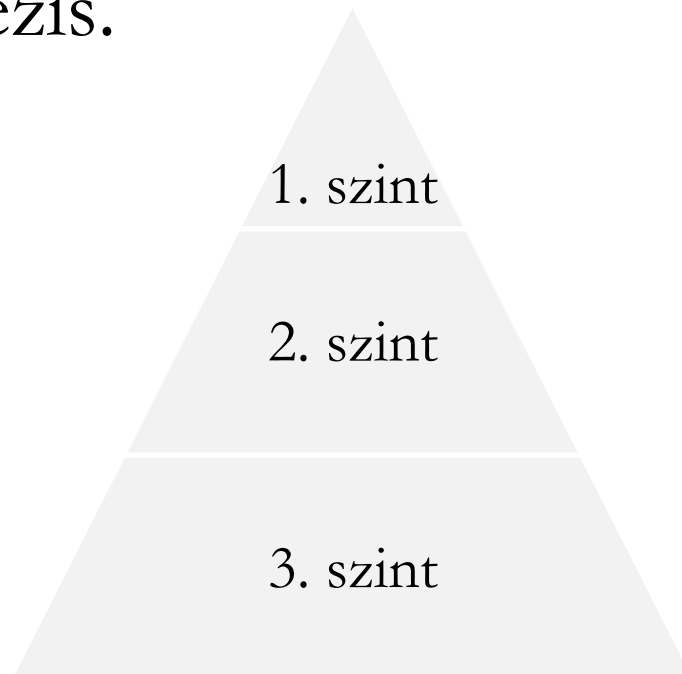
- **Stratégiai elv:** a problémamegoldás logikája – a lépésenkénti finomítás.
- **Taktikai elvek:** az algoritmuskészítés gondolati elvei a felülről lefelé kifejtéshez.
- **Technológiai elvek:** algoritmus és kód módszertani kívánalmak.
- **Technikai elvek:** kódolási technika.
- **Esztétikai, ergonómiai elvek:** emberközelség.



Stratégiai elv: lépésenkénti finomítás

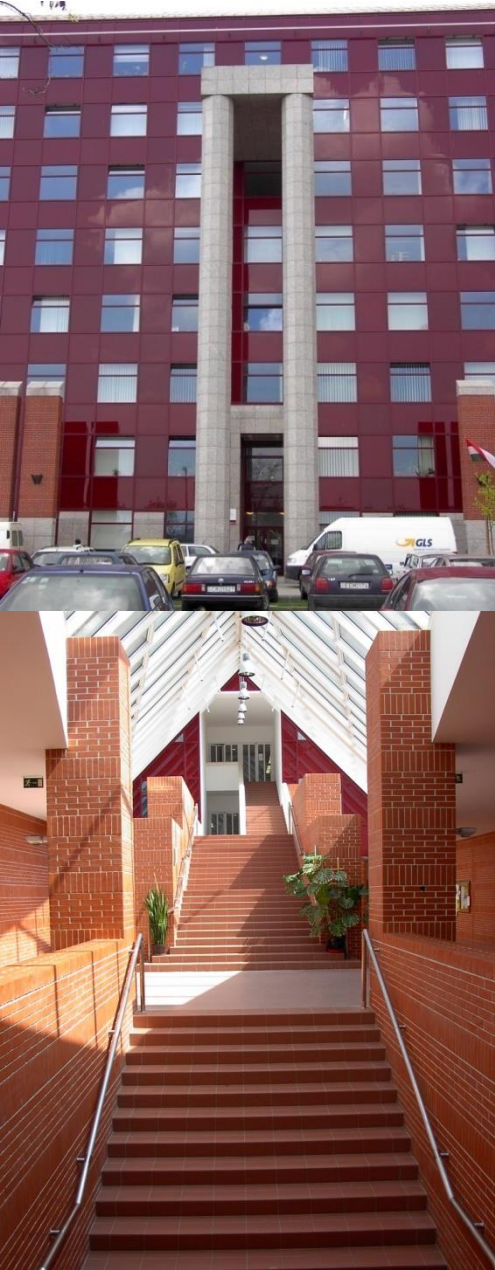
- Felülről–lefelé (top–down) = probléma–dekomponálás, –analizálás.
- Alulról–felfelé (bottom–up) = probléma–szintézis.

**Nem
alternatívák!**



Taktikai elvek

- Párhuzamos finomítás
- Döntések elhalasztása
- Döntések nyilvántartása
- Vissza az ősökhöz
- Nyílt rendszer felépítés (általánosítás)
- Párhuzamos ágak függetlensége
- Szintenkénti teljes kifejtés
- Adatok elszigetelése (pl. alprogramokba helyezéssel + paraméterezéssel + lokális adatok deklarációjával)



Technológiai elvek az algoritmus készítéshez

Struktogram esetén ezek nyilvánvalóan teljesülnek

- Struktúrák zárójelezése
- Bekezdéses struktúrák
- Értelmes utasítás-csoportosítás
- Kevés algoritmusleíró szabály definiálása, de azok szigorú betartása (pl. tétel → algoritmus)
- Beszédes azonosítók, kifejező névkonvenciók (pl. Hungarian Notation)



Technikai elvek a kódoláshoz



- Barátságosság (pl. kérdések, címek)
- Biztonságosság (pl. I/O-ellenőrzések)
- Kevés kódolási szabály definiálása, de azok következetes betartása (**algoritmus és kód koherenciája**; továbbá pl. amígos ciklusokhoz, I/O-hoz)
- Jól olvashatóság (vö.: Code::Blocks-ban a kódolási stílusok)





Esztétikai/ergonómiai elvek



- Lapokra tagolás, kiemelés, elkülönítés
- Menütechnika
- Ikontechnika, választás egérrel
- Következetesség (beolvasás, kiírás, ...)
- Hibafigyelés, hibajelzés, javíthatóság
- Súlyzó, tájékoztató
- Ablakkezelés
- Értelmezési tartomány kijelzése
- Naplózás



Programozási alapismeretek

9. előadás vége