Objektumorientált Programozás I.

Algoritmizálási alapismeretek
Algoritmus végrehajtása a számítógépen
Adattípusok
Típuskonverziók

Hallgatói Tájékoztató

A jelen bemutatóban található adatok, tudnivalók és információk a számonkérendő anyag vázlatát képezik. Ismeretük szükséges, de nem elégséges feltétele a sikeres zárthelyinek, illetve vizsgának.

Sikeres zárthelyihez, illetve vizsgához a jelen bemutató tartalmán felül a kötelező irodalomként megjelölt anyag, a gyakorlatokon szóban, illetve a táblán átadott tudnivalók ismerete, valamint a gyakorlatokon megoldott példák és az otthoni feldolgozás céljából kiadott feladatok önálló megoldásának képessége is szükséges.

Objektumorientált Programozás I.

Algoritmizálási alapismeretek

Algoritmus végrehajtása a számítógépen

Adattípusok

Típuskonverziók

Az algoritmus

- **Fogalma:** Jól definiált utasítások véges sorozata
- **Algoritmus készítésének lépései:**
  - A folyamatot elemi lépésekre bontjuk
  - Figyelembe vesszük az összes felmerülő lehetőséget
  - Ügyelünk, hogy az algoritmus véges sok lépésben véget érjen

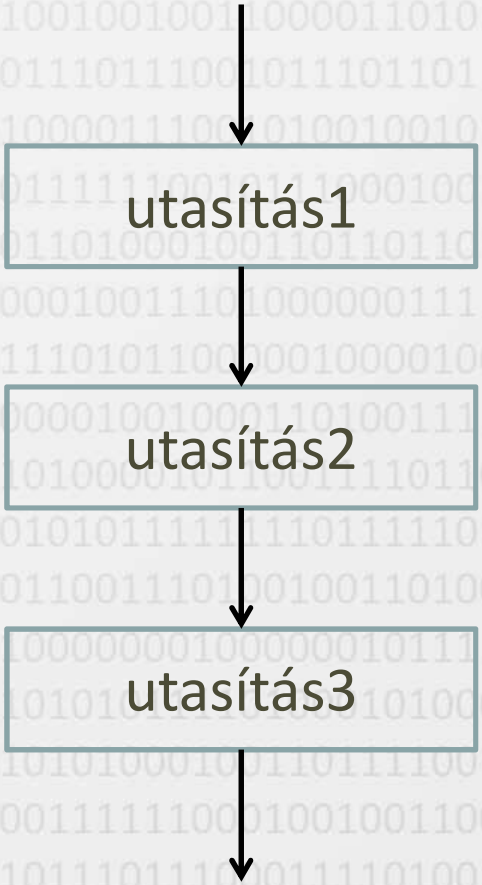
Algoritmusleíró módszerek

- **Blokkdiagram**
  Teendők és kérdések összekötése nyilakkal

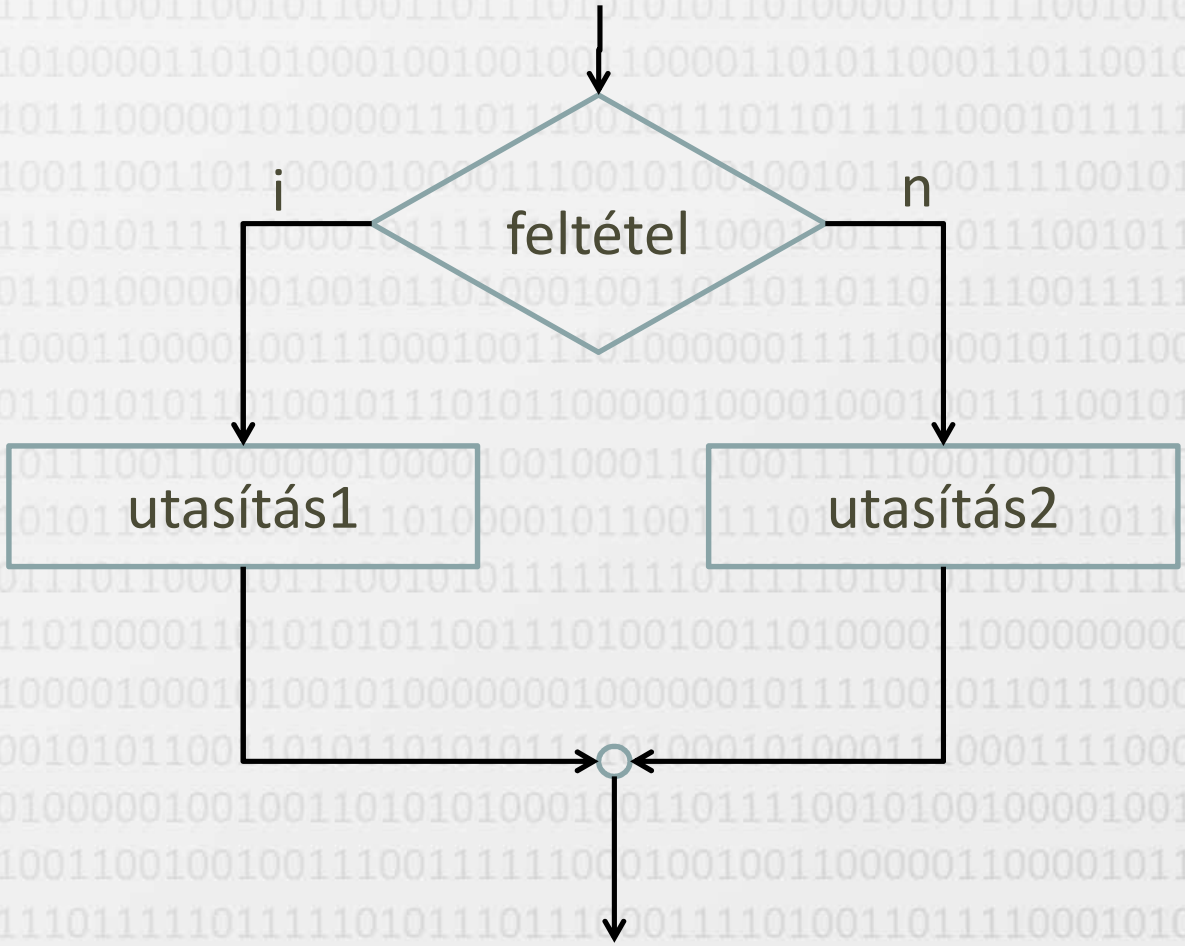
- **Struktogram**
  Teendők és kérdések strukturáltan kötött, mindig téglalap alakú képi reprezentációja

- **Szöveges leírás (pszeudokód)**
  Teendők és kérdések kötött kifejezésekkel történő szöveges leírása

Blokkdiagram
Szekvencia (utasítássorozat)

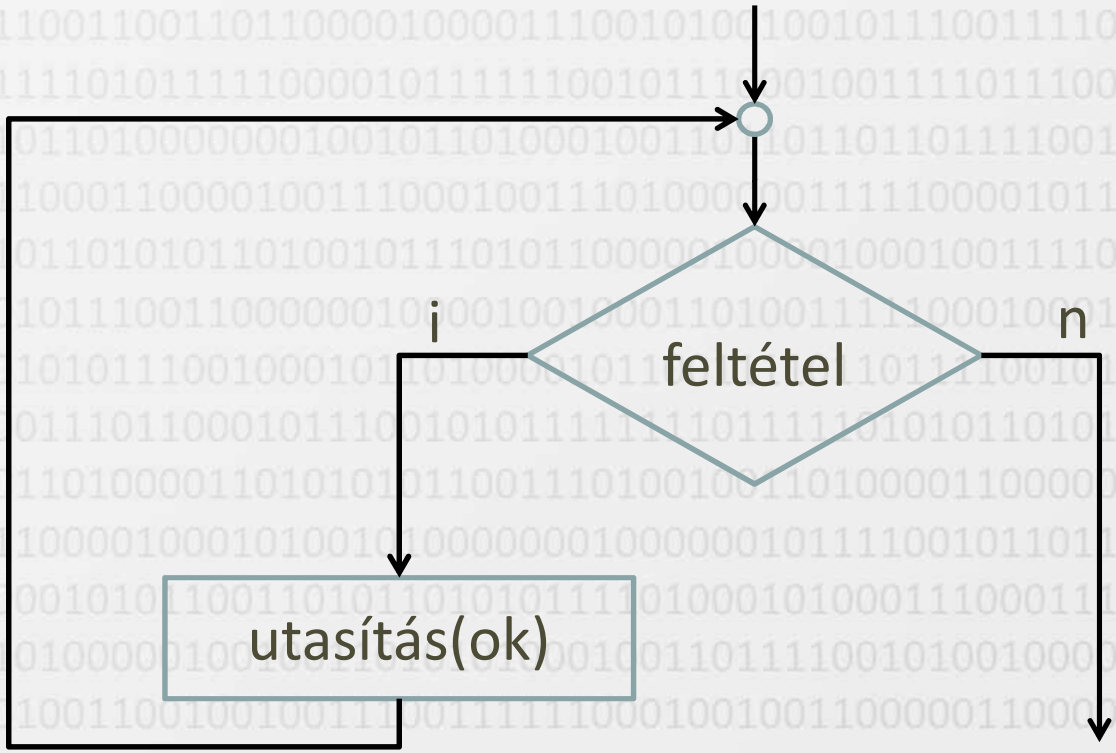


Blokkdiagram
Szelekció (elágazás)

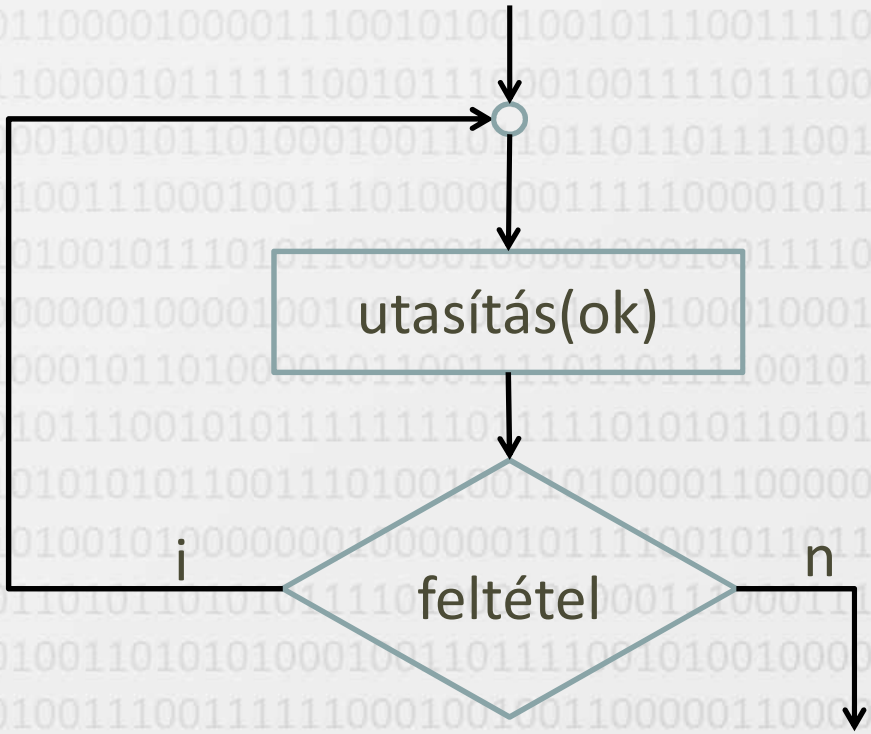


Blokk-diagram
Iteráció (elöltesztelős ciklus)

Nincs külön jelölés a ciklusra. Elágazással lehet megvalósítani.

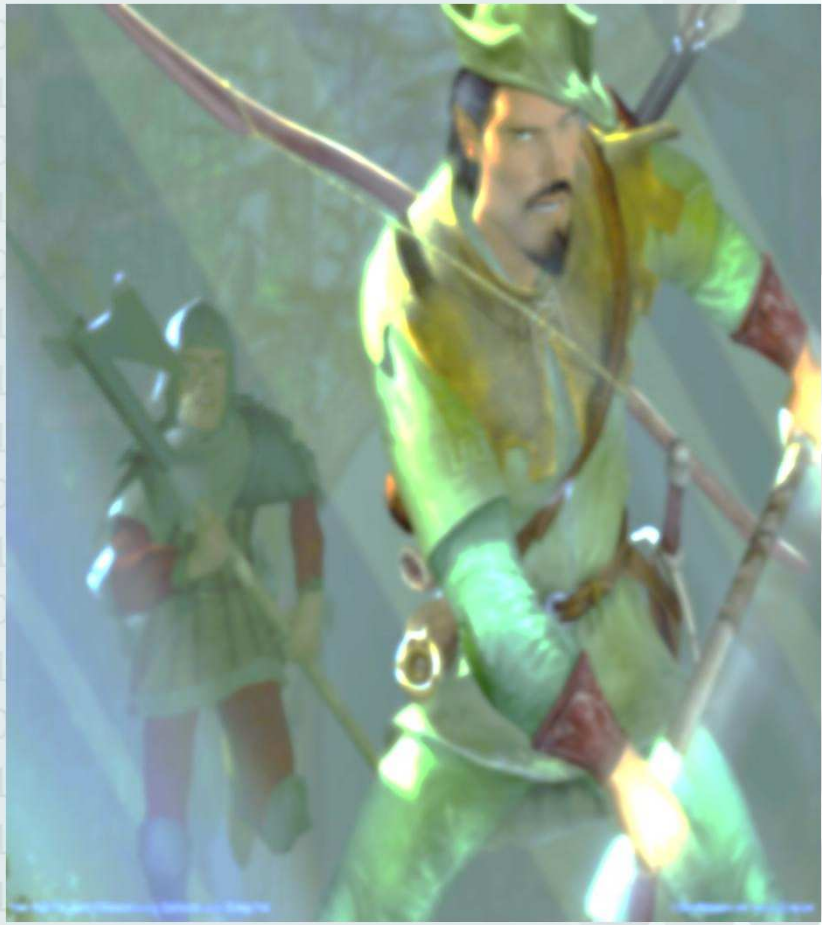


Blokkdiagram
Iteráció (hátultesztelős ciklus)

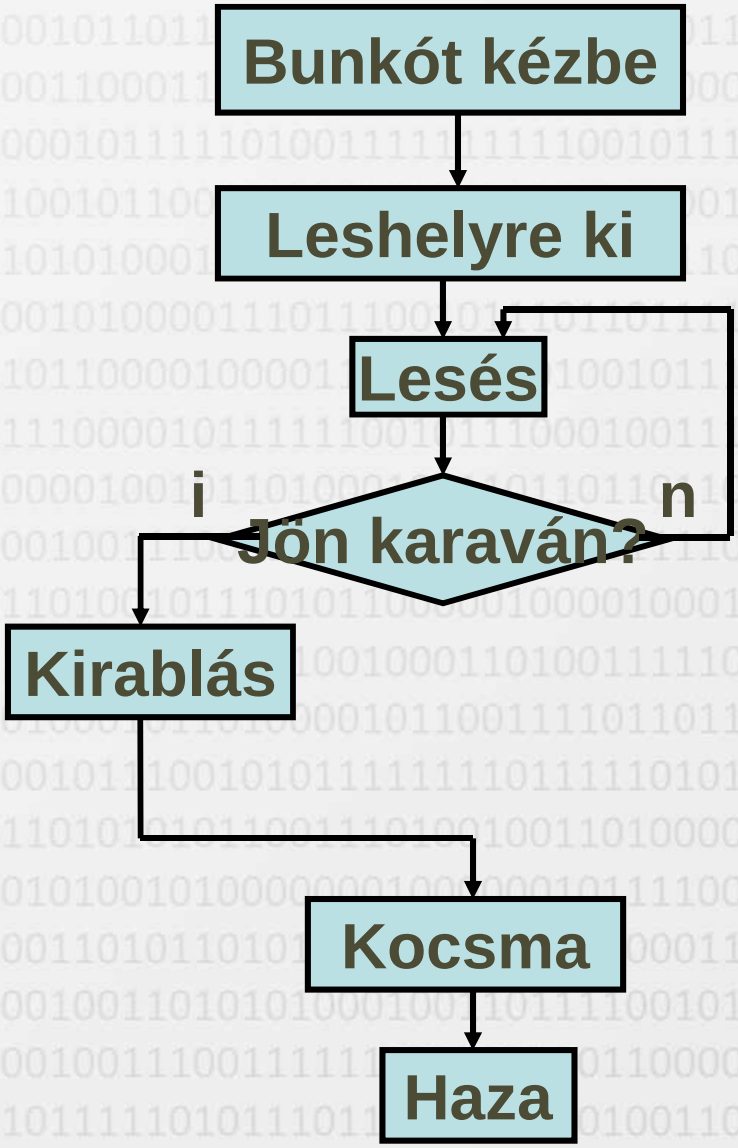


Példa – Rablóbanda

Feladat:
Egy rablóbanda az erdőben les gazdag áldozataira. A gazdasági fellendülés következtében megnőtt az erdei úton közlekedő, kincsekkel megrakott konvojok száma, ezért szükségessé vált a bandát új taggal bővíteni. Az új tagnak el kell magyarázni a rablás folyamatát.

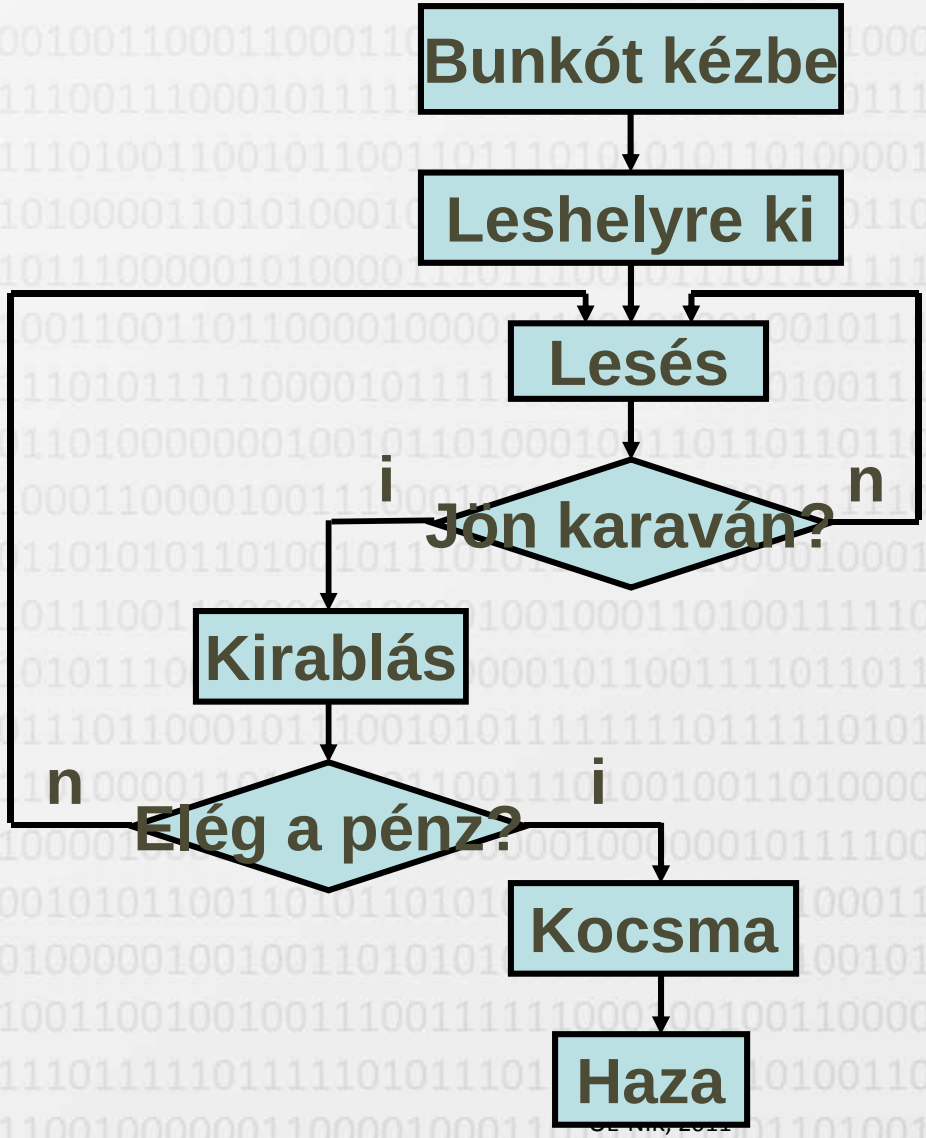


Rablók I.


Bunkót kézbe
Leshelyre ki
Lesés
i
Jön karaván?
n
Kirablás
Kocsma
Haza

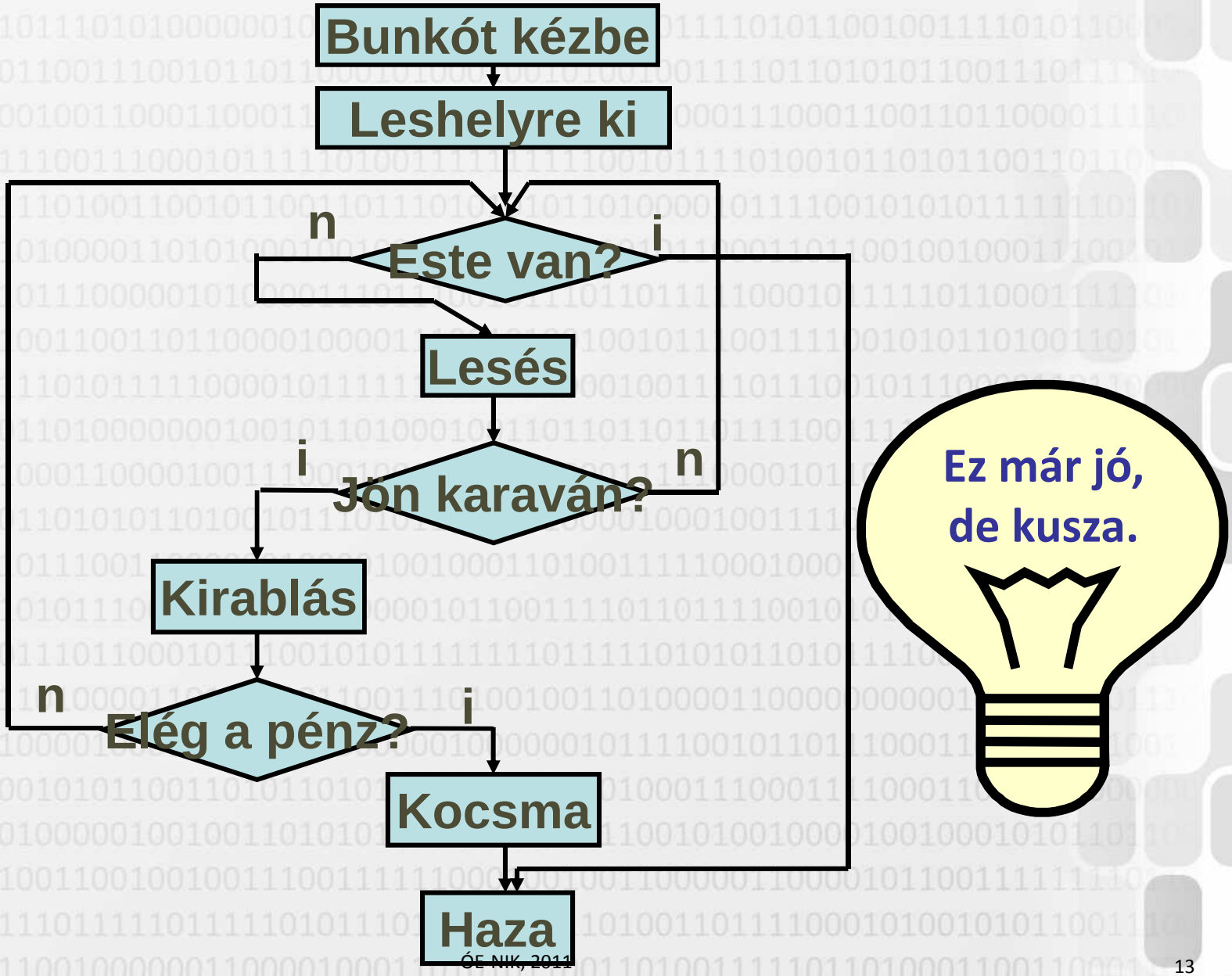

És ha nem volt
a karavánnál
pénz?

Rablók II.


Bunkót kézbe
Leshelyre ki
Lesés
Jön karaván?
i
n
Kirablás
Elég a pénz?
n
i
Kocsma
Haza


És ha nem
is jött
karaván?

Rablók III.


Bunkót kézbe
Leshelyre ki
n
Este van?
i
Lesés
i
Jön karaván?
n
Kirablás
n
Elég a pénz?
i
Kocsma
Haza
Ez már jó,
de kusza.

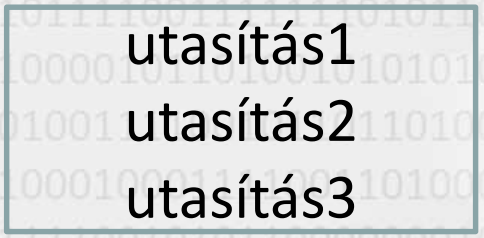
Problémák a blokkdiagrammal

- **Nyilak és vonalak kesze-kusza rendszere**
- **Teljesen ad-hoc elrendezésű, két ugyanolyan algoritmus leírása a rajzoló kénye-kedve szerint akár teljesen más elrendezésű is lehet**

➔ **Struktogram: kötött struktúra, nincsenek nyilak, csak egymásba foglalt téglalapok**

Struktogram
Szekvencia (utasítássorozat)

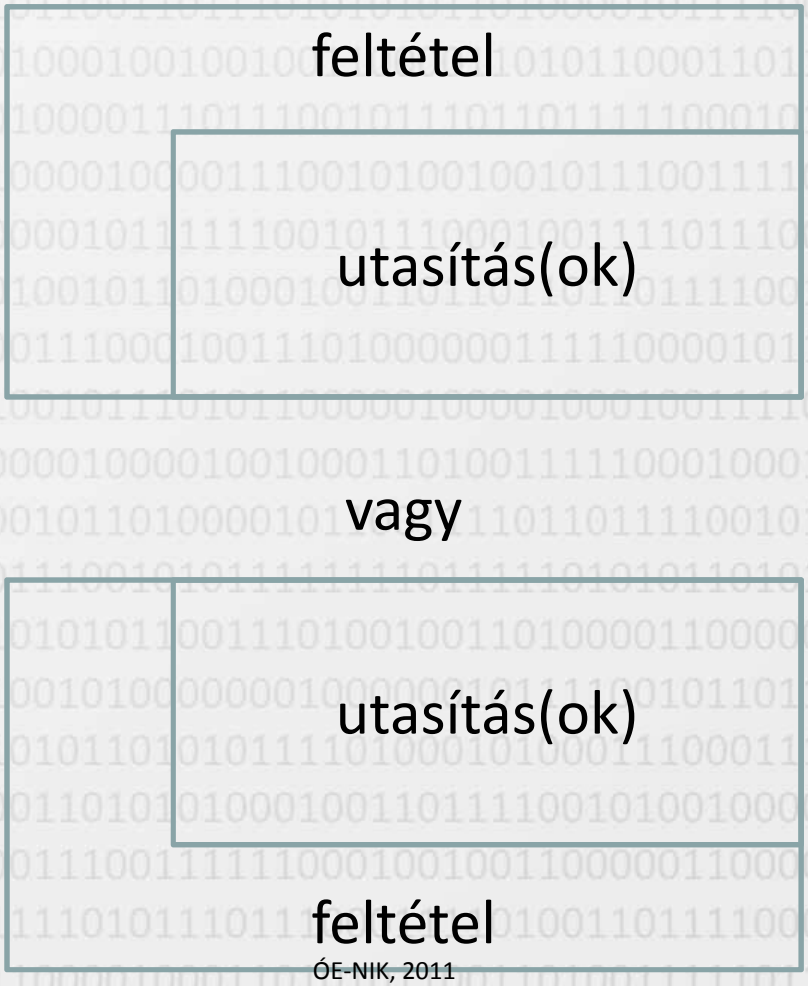
vagy



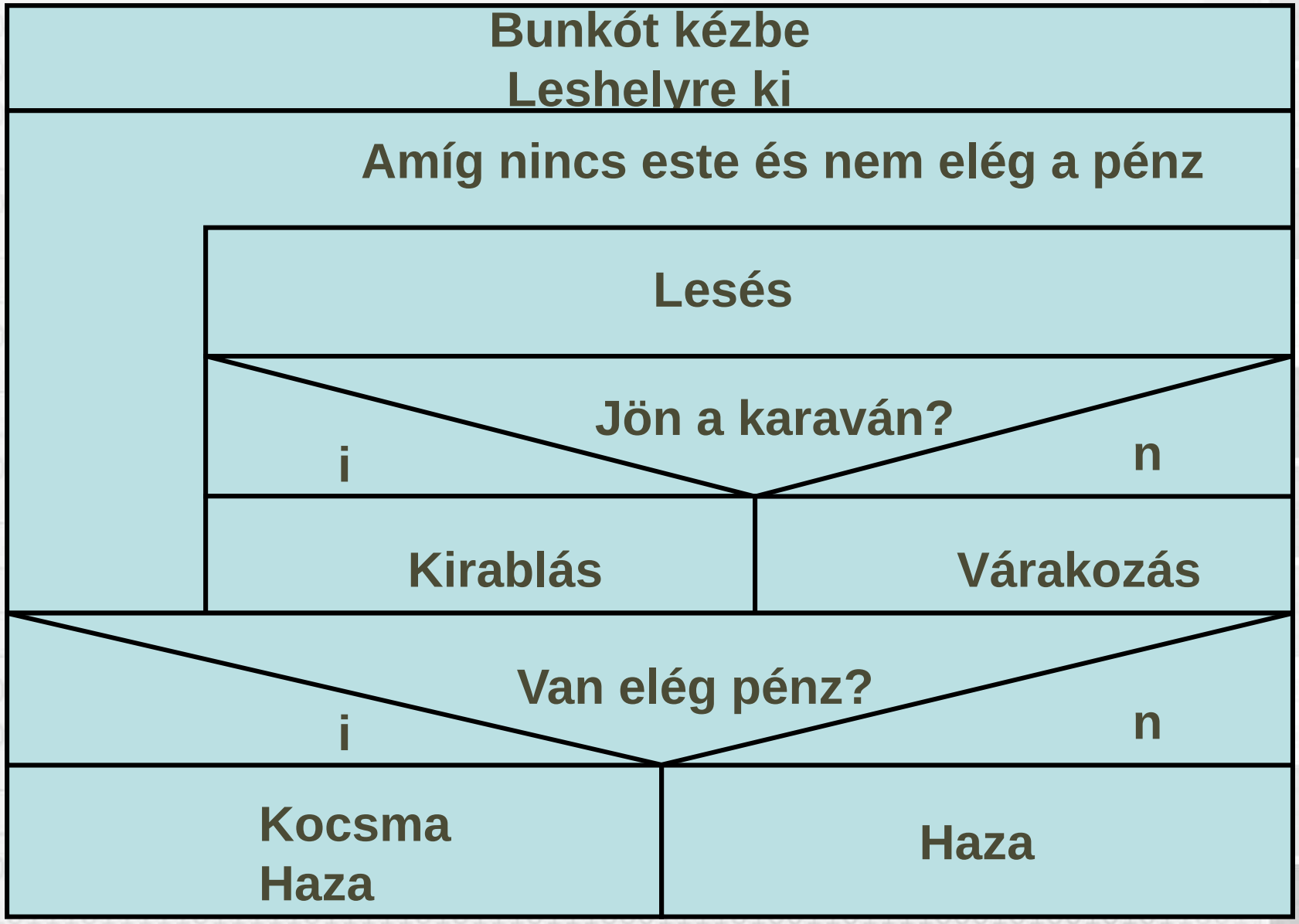
Struktogram
Szelekció (elágazás)

vagy

Struktogram
Iteráció (ciklus)



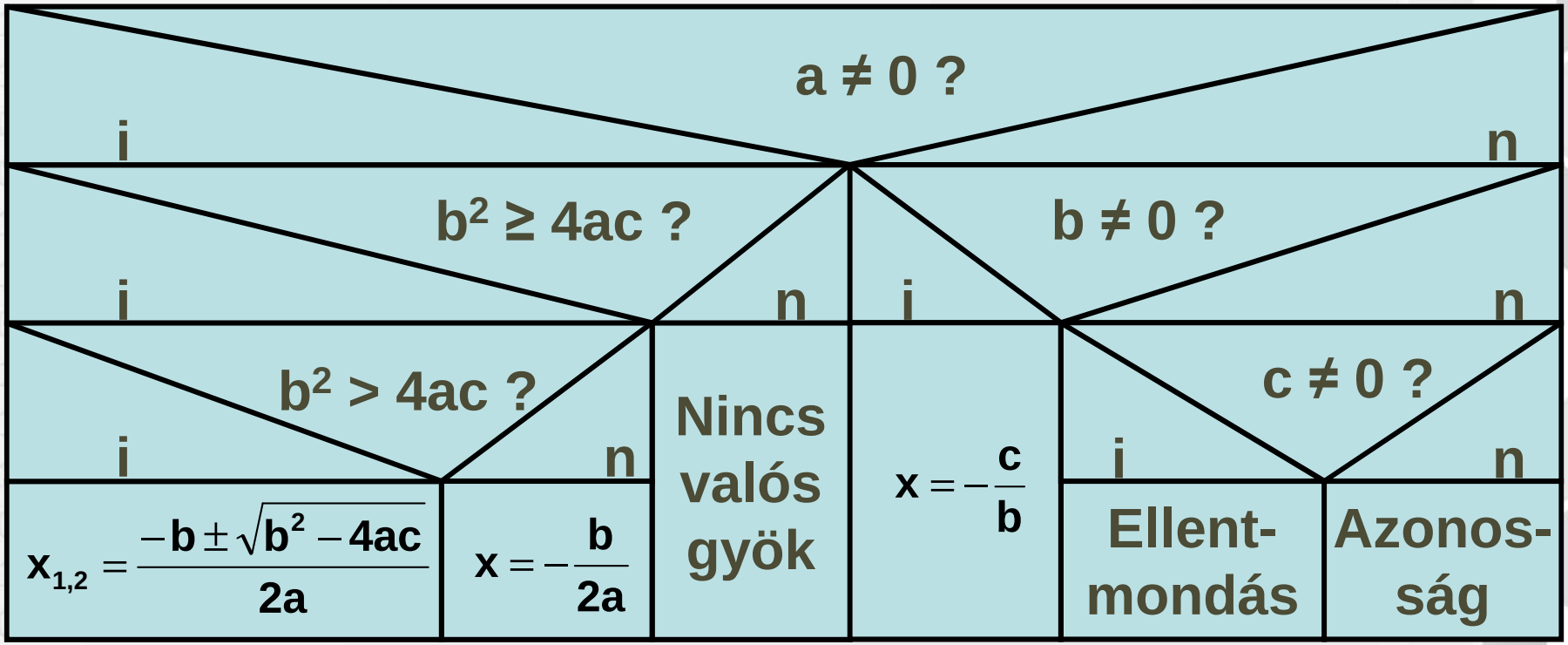
Rablók IV.


Bunkót kézbe
Leshelyre ki
Amíg nincs este és nem elég a pénz
Lesés
Jön a karaván?
i
n
Kirablás
Várakozás
Van elég pénz?
i
n
Kocsma
Haza
Haza

Másodfokú egyenlet

Készítsen algoritmust a másodfokú egyenlet megoldására.

$$ax^2 + bx + c = 0$$



Problémák a struktogrammal

- **Nehezen módosítható, cserélhető**
- **Az elkészítése és az értelmezése olykor nehézkes**

➔ **Pszeudokód: kötött kifejezések használatával az algoritmus szöveges leírása**

Szöveges leírás I.

Szekvencia

Az utasítás végén „;”
Az utasításokat kapcsos zárójelekkel fogjuk össze:
{ ez legyen;
az legyen; }

Elágaztatás

Ha (feltétel) { ez legyen; }
egyébként { az legyen; }

Az „egyébként” ág nem kötelező

Szöveges leírás II.

Ciklus 1.

amíg (feltétel)
{ ezek ismétlődjenek; }

Ciklus 2.

tedd { ezek ismétlődjenek; }
amíg (feltétel)

Szöveges leírás III.

Szekvencia

Az utasítás végén „;”
Az utasításokat kapcsos zárójelekkel fogjuk össze:

```
{  ez legyen;
   az legyen;  }
```

Elágaztatás

```
if (feltétel) { ez legyen; }
else { az legyen; }
```

Az „else” ág nem kötelező

Szöveges leírás IV.

Ciklus 1.

```
while (feltétel)
{ ezek ismétlődjenek; }
```

Ciklus 2.

```
do { ezek ismétlődjenek; }
while (feltétel)
```

Objektumorientált Programozás I.

Algoritmizálási alapismeretek

Algoritmus végrehajtása a számítógépen

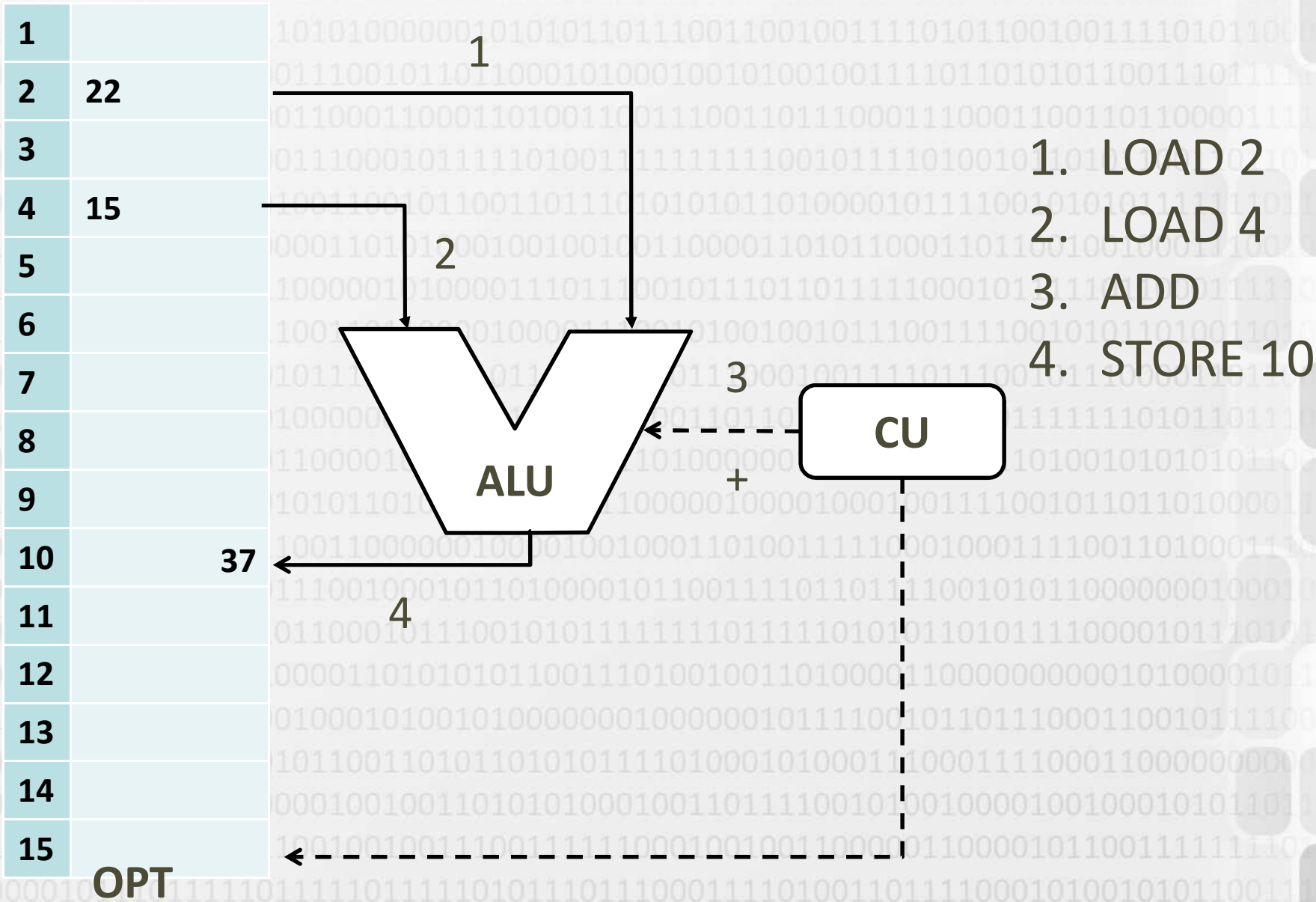
Adattípusok

Típuskonverziók

Számítógépes műveletvégzés

- **Cél: annak modellezése, hogy az egyszerű adatokat hogyan tárolja, és a műveleteket hogyan végzi el a számítógép (részletesebben: Architektúrák I.)**
- **Használt elemek:**
  - **OPT (Operatív tár: memória, cache, regisztertér is lehetne. Ez mindegy, a lényeg: byteszervezésű tár)**
  - **ALU (Arithmetical and Logical Unit: Aritmetikai és Logikai Egység; a műveletvégző egység. 2 bemeneten tud valamilyen műveletet végezni, 1 kimenet)**
  - **CU (Control Unit: Vezérlőegység)**
  - **Most NEM használjuk a „CPU” elnevezést, mert abban lenne más is (buszok, regiszterek, több fajta cache, etc.)**

Az ALU sematikus működése



1	
2	22
3	
4	15
5	
6	
7	
8	
9	
10	37
11	
12	
13	
14	
15	

Problémák a közvetlen memóriacímekkel

- **Ha így működne, egy nagyobb program megírása szinte lehetetlen komplexitású lenne (a jelenlegi számítógépek címtere óriási)**
- **Fogalmunk sincs, hogy az operációs rendszer pontosan hova helyezi el a programunkat (több program fut egyszerre)**
- **Neumann-elv: „Az adatok és az utasítások a közös operatív tárban vannak” ➔ ugyanolyan bináris reprezentációban! Nehéz lenne karban tartani, hogy hol van adat, és hol van utasítás**

➔ Megoldás: a programok csak <u>változókat</u> használnak, a változók konkrét memóriacíme nem érdekes

Problémák a változókkal

- **A fenti példában a memória byteszervezésű: 1 rekesz = 1 byte, a tárolás bitekben történik**
- **Ha számot akarunk tárolni, akkor 8 bit csak a 0..255 intervallumban elég. Mi van, ha nagyobb számokat akarunk tárolni?**
- **Mi van, ha nem számot akarunk tárolni, hanem szöveget?**
- **Mi van, ha nem szöveget akarunk tárolni, hanem egy képet vagy más bináris adatot?**

➔ A változók bevezetése önmagában nem elég. Tudnunk kell, hogy a változó által kijelölt területen MENNYI adat van, és azt HOGYAN kell értelmezni ➔ típusok

Objektumorientált Programozás I.

Algoritmizálási alapismeretek
Algoritmus végrehajtása a számítógépen
Adattípusok
Típuskonverziók

Parancsok, adatok

- A számítógép minden adatot és utasítást bináris formában tárol a memóriában
- A tárolt bináris adat jelentése értelmezésfüggő
- Az adatokat változókban tároljuk
- A változó deklarációja határozza meg a tárolt adat méretét és értelmezését (típusát)

Egyszerű adattípusok

- MINDENT binárisan tárolunk, az összes adattípus mérete a byte többszöröse
- Számok
  - Egész
  - Valós (lebegőpontos)
- Karakterek, karaktersorozatok (string-ek)
- Logikai értékek

Egész (fixpontos) számok

- Két fő kérdés: tárolási méret, előjelesség → a kettőtől függ az ábrázolás értelmezési tartománya
- Relatív kicsi ábrázolási tartomány, de teljes pontosság
- Előjeles ábrázolási mód: kettes komplemens, részletesebben: Informatika Elméleti Alapjai

Bitek száma	Előjeltelen	Előjeles
8	byte	sbyte
16	ushort	short
32	uint	int
64	ulong	long

Egész (fixpontos) számok

Név	Leírás	Értéktartomány
sbyte	8 bites előjeles egész	-128 : 127
byte	8 bites előjel nélküli egész	0 : 255
short	16 bites előjeles egész	-32 768 : 32 767
ushort	16 bites előjel nélküli egész	0 : 65535
int	32 bites előjeles egész	-2 147 483 648 : 2 147 483 647
uint	32 bites előjel nélküli egész	0 : 4 294 967 295
long	64 bites előjeles egész	-9 223 372 036 854 775 808 : 9 223 372 036 854 775 807
ulong	64 bites előjel nélküli egész	0 : 18 446 744 073 709 551 615

Műveletvégzés egész változókkal

	1	
A	2	22
	3	
B	4	15
	5	
	6	
	7	
	8	
	9	
C	10	37
	11	
	12	
	13	
	14	
	15	

OPT

C# Szintaxis:

típusnév változónév = kezdőérték;

A változó fizikai helye a programozó számára többnyire ismeretlen – de igazából nem is érdekes

```
byte a=22;
byte b=15;
byte c=a+b;
```

Ez már akár C# nyelven írt kódrészlet is lehetne!

Műveletvégzés egész változókkal

	1	
A	2	22
	3	0
B	4	15
	5	0
	6	
	7	
	8	
	9	
C	10	37
	11	0
	12	
	13	
	14	
	15	

OPT

Hogyan tárolunk byte-szervezésű tárban nem egy byte-os változókat?

A konkrét tárolási mód eltérhet (MSB first / LSB first), részletesebben: IEA

```
ushort a=22;
ushort b=15;
ushort c=a+b;
```

A típusdefiníció magával vonja az adat méretét és értelmezési módját

Egész számok speciális értékei

- MinValue
  - Az ábrázolható legkisebb szám
  - byte.MinValue, int.MinValue, etc…
- MaxValue
  - Az ábrázolható legnagyobb szám
  - short.MaxValue, long.MaxValue, etc…
- Túlcsordulás (a „változó++” növeli a változó értékét)
  - byte a=255; a++; ← a változó értéke 0 lesz
  - sbyte b=-128; b--; ← a változó értéke 127 lesz

Valós (lebegőpontos) számok

$$\pm m * 2^k$$

- Normalizált szám formájában tároljuk (előjel, mantissza, karakterisztika, pontos módszer: IEA)
- Nagy számtartomány, de nem pontos
- A számábrázolás formájából adódóan nem csak abszolút értékben túl nagy, de nullához túlságosan közeli számokat sem tud ábrázolni
- A karakterisztika mérete az ábrázolható számtartomány méretét, a mantissza mérete a pontosságot határozza meg

Valós (lebegőpontos) számok

Név	Leírás	Értékes jegy	Értéktartomány
float	32 bites lebegőpontos	7	$\pm 1{,}5*10^{-45}$: $\pm 3{,}4*10^{38}$
double	64 bites lebegőpontos	15	$\pm 5{,}0*10^{-324}$: $\pm 1{,}7*10^{308}$
decimal	128 bites nagypontosságú	28	$\pm 1{,}0*10^{-28}$: $\pm 7{,}9*10^{28}$

	Méret	Előjel	Kitevő	Törtrész	Eltolás
Egyszeres IEEE-754 szabvány pontosság	32 bit	1 bit	8 bit	23 bit	127
Kétszeres pontosság	64 bit	1 bit	11 bit	52 bit	1023

Nullával való osztás

- Egész számtípus használatakor futás idejű hibát dob:

```
int a = 5;
int b = 0;
int c = a / b;
```

- Valós számtípus használatakor hibátlan:

```
float x = 5;
float y = 0;
float z = x / y;
```

- Valós számtípus esetén az eredmény lehet: végtelen (pozitív illetve negatív), illetve „Nem szám”

Valós számok speciális értékei

- 0
  - Külön +0 és -0 ábrázolható, de ezek egyenértékűek
- ±∞
  - A végtelen elfogadott, bizonyos műveletekhez használható érték
  - PozitívSzám/0 → + ∞, NegatívSzám/0 → - ∞
  - float.PositiveInfinity , double.NegativeInfinity – decimal nincs!
- Nem szám
  - 0/0, illetve ∞/∞ eredménye
  - float.NaN , double.NaN – decimal nincs!

Valós számok speciális értékei

- Epsilon
  - A legkisebb ábrázolható pozitív szám
  - float.Epsilon , double.Epsilon – decimal nincs!
- Kezdőérték megadása
  - Kódban tizedesPONT használandó: double pi=3.14;
  - Minden így megadott érték típusa double!
- Jelzőkarakterek kezdőérték megadásánál
  - float pi=3.14f;
  - decimal pi=3.14M

Karakterek

- Egy karakter tárolása ugyanúgy binárisan történik → kell lennie egy szabálynak, hogy melyik kód melyik karakternek felel meg
- ASCII: kezdetben 7 bites. 0-31: vezérlő karakterek; 32-127: angol ABC kis- és nagybetűi, számok, írásjelek
- 8 bites ASCII: 128-255: rajzoló karakterek, speciális karakterek (ä, ç), nyugat-európának megfelel
  - Hiányzó karakterek: ő, Ő, ű, Ű (csak ô, û)
  - Nincs elég hely: japán, kínai, szír, etc…
  - Alternatíva: kódlapok (cp437, cp850/852, cp1250)
  - Kódlapok szabványosítása (ISO8859-1, -2, -15)
  - Probléma: készítés kódlapja ←→ feldolgozás kódlapja

Karakterek

- Alternatíva: felejtsük el az 1 byte = 1 karakter szabályt
  - Probléma: akkor hogyan állapítjuk meg egy karakterlánc hosszát? Eddig egyszerű volt, de ezután… ➔ mindent újra kell írni... ☹
- UNICODE kódolás, UTF-8, UTF-16, UTF-32 kódlapok
  - UTF-8: Az angol ABC betűinek kódolása ugyanaz, a többi karakternek egyedi kódja van, 2-4 byte / karakter
  - UTF-16: 2 vagy 4 byte / karakter
  - AZ UTF-16 A C# NYELV ÉS A .NET KERETRENDSZER BELSŐ KÓDOLÁSA *(a file-ok kódolása UTF-8)*
  - Minden karakteres típus, minden szövegkezelő függvény ez alapján működik ➔ ☺

Karakterek, karakterláncok

- Karakter: char (megadás: aposztróffal)
  - char karakter='ű';
- Karakterlánc: string (megadás: idézőjellel)
  - string karakterlanc="Árvíztűrő Tükörfúrógép";
- Speciális karakterek is megadhatóak (@ jellel kikapcsolható):

Jelölés	Karakter
\0	Null karakter
\a	Sípszó
\b	Visszatörlés
\f	Lapdobás
\n	Soremelés
\r	Kocsi vissza
\t	Vízszintes tabulátor

Jelölés	Karakter
\v	Függőleges tabulátor
\x....	Hexadecimális kód
\u....	Unicode karakter
\U....	Unicode karakter
\'	Aposztróf
\"	Idézőjel
\\	Backslash

Logikai típus

Név	Leírás	Értéktartomány
bool	**Logikai adattípus**	true **vagy** false **(igaz vagy hamis)**

- **Teljesítmény-okokból általában nem 1 biten ábrázoljuk, részletesebben lásd IEA**
- **Logikai műveletek:**

A	B	$A \wedge B$	$A \vee B$	$A \oplus B$	$\neg(A)$
H	H	H	H	H	I
H	I	H	I	I	I
I	H	H	I	I	H
I	I	I	I	H	H

Változók deklarálása és használata

```
int j = -10;
int x = 10, y = 20;
double pi = 3.14159;
const int száz = 100;
char d = 'x';
char UnicodePélda = '\u0170'; // "Ű" karakter
string jegy = "jeles";
string ElérésiÚt = "C:\\Program Files\\";
string ElérésiÚt2 = @"C:\Program Files\";
string vers = @"Hová merült el
  szép szemed világa";
bool igaz = true;
```

Fontos szabály: azonos névvel egy változót nem lehet kétszer deklarálni!

A közvetlenül beírt értékek más neve: *literál*

Speciális literálok

- Egész literál:
  - Típusuk: int, uint, long, vagy ulong (ebben a sorrendben) attól függően, hogy melyik típusban fér el a megadott érték
  - Az egész literál típusa is módosítható a literál mögé írt betűkkel:
    - U : uint, vagy ulong (pl.: 255U)
    - L : long vagy ulong (pl.: -356L)
    - UL : ulong (pl.: 222UL)
  - Megadható hexadecimálisan: 0xFF
- Valós literál, tudományos megadás: 1.23456E-2

Objektumorientált Programozás I.

Algoritmizálási alapismeretek
Algoritmus végrehajtása a számítógépen
Adattípusok
Típuskonverziók

Típuskonverziók

- A számtípusok közötti konverzió miként történik attól függ, hogy történik –e értékvesztés a konverzió során
- Egyszerű értékadás használható, amennyiben biztos, hogy nincs értékvesztés:

```
byte a=5;      long c=5;      float f=3.2f;
int b=a;       float d=c;     double g=f;
```

- Amennyiben értékvesztés történhet, akkor mindenképp jelezni kell a konverziót, ez az ún. típuskényszerítés, „kasztolás” (typecasting):

```
int a=999;          double d=3.14;   int i1=-1;
byte b=(byte)a;     int c=(int)d;    uint i2=(uint)i1;
```

Típuskonverziók

- A stringgé történő konverzió a C# nyelven MINDEN változónál ugyanúgy történik:

```
byte b=250;                    float f=3.14f;
string s1=b.ToString();        string s2=f.ToString();
```

- Stringből számmá tudunk konvertálni:

```
string s="123";                string s2="123,456";
byte b=byte.Parse(s);          float f=float.Parse(s2);
```

- Typecasting esetén (ebben a félévben számok között):
  célváltozó = (céltípus)forrásváltozó;
- Stringgé konvertálásnál:
  célváltozó = forrásváltozó.ToString();
- Stringből konvertálásnál:
  célváltozó=céltípus.Parse(stringváltozó);

Objektumorientált Programozás I.

- ✓Algoritmizálási alapismeretek
- ✓Algoritmus végrehajtása a számítógépen
- ✓Adattípusok
- ✓Típuskonverziók

Irodalom, feladatok

- **Kotsis-Légrádi-Nagy-Szénási: Többnyelvű programozástechnika, PANEM, Budapest, 2007**
- **Faraz Rasheed: C# School, Synchron Data, 2006 http://www.programmersheaven.com/2/CSharpBook**
- **Reiter István: C# jegyzet, DevPortal, 2010, http://devportal.hu/content/CSharpjegyzet.aspx**