Adatszerkezetek

1. hét – Bevezetés, adatok és elemi adattípusok

Dr. Koczka Ferenc · Eszterházy Károly Egyetem · 2026

Az előadóról

- **Dr. Koczka Ferenc**, egyetemi adjunktus
- Eszterházy Károly Egyetem, Eger
- E-mail: **koczka.ferenc@uni-eszterhazy.hu**
- Konzultáció: az órák után, illetve e-mailben előre egyeztetve

> „First, solve the problem. Then, write the code.” – John Johnson

A kurzus felépítése

Hét	Téma	Hét	Téma
1	Bevezetés, elemi adattípusok	9	Dinamikus memória, láncolt lista
2	Hatékonyság, keresések	10	Strázsás, kétirányú, ciklikus lista
3	Tömbök	11	Dinamikus verem és sor, hash tábla
4	Stringek	12	Rekurzió
5	Rendezések I. (egyszerű rendezések)	13	Rendezések II. (összefésülő, gyors)
6	Halmaz	14	Fák, bináris fa, bejárások
7	Verem	15	Bináris keresőfa, kupac, összefoglalás
8	Sor	16	**Záró dolgozat és gyakorlati jegy**

Minden héten: **2 óra előadás + 2 óra gyakorlat** (Java és Go)

Követelmények és eszközök

Értékelés (16. hét)

- **Záró dolgozat** (elmélet, 90 perc)
  igaz/hamis, feleletválasztós, kódkövetés, pszeudokód
- **Gyakorlati jegy** (gépes feladat)
  adatszerkezet megvalósítása Java **vagy** Go nyelven

Eszközök

- **Java** – JDK 17 vagy újabb
- **Go** – 1.23 vagy újabb
- Szerkesztő: VS Code, IntelliJ IDEA vagy Eclipse
- A példaprogramok: `kod/java/hXX`, `kod/go/hXX`

Miért tanulunk adatszerkezeteket?

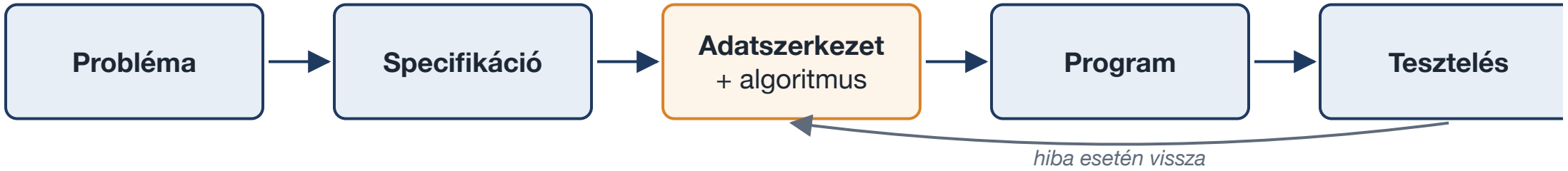
> **Algoritmusok + Adatszerkezetek = Programok**
> – Niklaus Wirth (1976)

- Ugyanaz a feladat **jó** adatszerkezettel gyors, **rossz** adatszerkezettel használhatatlanul lassú
- Minden programozási nyelv ezekre az alapokra épít (`ArrayList` , `HashMap` , Go `slice` , `map` ...)
- Az állásinterjúk kedvenc témája
- A későbbi tárgyak (adatbázisok, operációs rendszerek, mesterséges intelligencia) feltételezik

Cél: ismerjük az alapvető adatszerkezeteket, a műveleteiket, a hatékonyságukat – és **meg tudjuk valósítani** őket.

Problémamegoldás és algoritmus

A problémamegoldás lépései



- **Specifikáció:** mi a bemenet, mi a kimenet, milyen feltételek teljesülnek?
- **Adatszerkezet + algoritmus:** hogyan tároljuk az adatokat, milyen lépésekben dolgozzuk fel őket?
- Ez a kurzus a középső lépésről szól

Az algoritmus fogalma

Algoritmus: véges számú, egyértelműen meghatározott lépésből álló eljárás, amely egy feladatot megold.

Tulajdonságai:

- **Véges** – véges sok lépés után befejeződik
- **Egyértelmű (determinisztikus)** – minden lépés pontosan meghatározott
- **Van bemenete** – nulla vagy több adat
- **Van kimenete** – legalább egy eredmény
- **Elvégezhető** – minden lépés ténylegesen végrehajtható
- **Általános** – a feladat egy egész osztályát oldja meg, nem csak egy esetet

Algoritmusleíró eszközök

Eszköz	Előny	Hátrány
Szöveges leírás	mindenki érti	pontatlan, hosszú
Folyamatábra	szemléletes	nagy feladatnál áttekinthetetlen
Pszeudokód	pontos, nyelvfüggetlen	meg kell tanulni a jelölést
Programkód	futtatható	nyelvfüggő részletek

Ebben a kurzusban: **pszeudokód** az előadáson és a dolgozatban, **Java és Go** a gyakorlaton.

A pszeudokód jelölései

Jelölés	Jelentés
`x ← kifejezés`	értékadás
`Ha feltétel akkor … különben … Elágazás vége`	elágazás
`Ciklus amíg feltétel … Ciklus vége`	elöltesztelő ciklus
`Ciklus i ← 1-től n-ig … Ciklus vége`	számlálós ciklus
`Függvény Név(paraméterek) … Vissza: érték … Függvény vége`	függvény
`Eljárás Név(paraméterek) … Eljárás vége`	eljárás (nincs visszatérési érték)
`A[i]`	tömb **i**-edik eleme – **az indexelés 1-től indul**
`és` , `vagy` , `nem` · `div` , `mod`	logikai műveletek · egész osztás, maradék

Java-ban és Go-ban a tömbök indexelése **0-tól** indul – ezt a kódoláskor mindig figyelembe kell venni!

Példa: legnagyobb közös osztó (Euklidesz)

Pszeudokód

```
Függvény LNKO(a, b)
  Ciklus amíg b ≠ 0
    m ← a mod b
    a ← b
    b ← m
  Ciklus vége
  Vissza: a
Függvény vége
```

a	b	a mod b
1071	462	147
462	147	21
147	21	0
21	0	

Java

```java
static int gcd(int a, int b) {
    while (b != 0) {
        int m = a % b;
        a = b;
        b = m;
    }
    return a;
}
```

Go

```go
func gcd(a, b int) int {
    for b != 0 {
        a, b = b, a%b
    }
    return a
}
```

Adat és adattípus

Adat és információ

- **Adat:** a valóság egy jellemzőjének rögzített, feldolgozható formája
  pl. `37.2`, `"Wuhan"`, `true`
- **Információ:** az adat jelentése egy adott összefüggésben
  pl. „a beteg testhőmérséklete 37,2 °C”
- A számítógép **adatokkal** dolgozik – a jelentést a program (és a programozó) adja

Az adat tulajdonságai

- **Név** (azonosító) – amivel hivatkozunk rá: `temperature`
- **Típus** – milyen értékeket vehet fel, mit lehet vele csinálni
- **Érték** – a pillanatnyi tartalma
- **Cím** – hol található a memóriában

Az adattípus

Az **adattípus** három dolgot határoz meg:

1. **Értékhalmaz** – milyen értékeket vehet fel
2. **Műveletek** – mit lehet az értékekkel csinálni
3. **Ábrázolás** – hogyan tárolja a gép (hány bit, milyen kódolás)

> **Példa – `int` (Java):**
> értékhalmaz: –2 147 483 648 … 2 147 483 647 · műveletek: `+ - * / % == < …` · ábrázolás: 32 bit, kettes komplemens

Csoportosítás

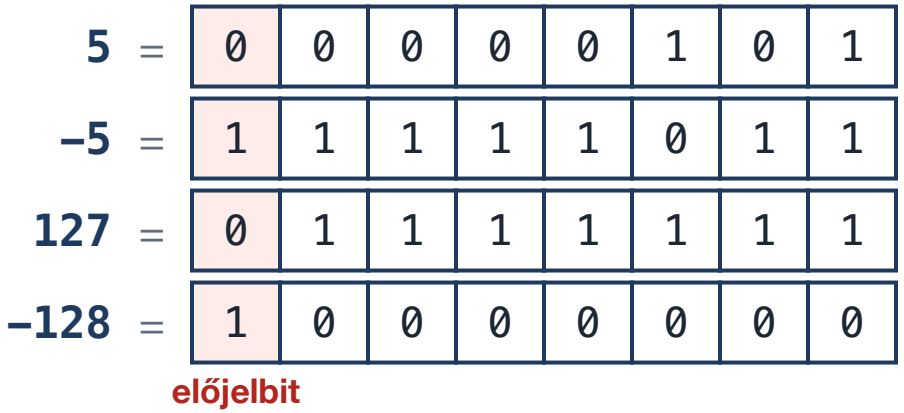
- **Elemi (egyszerű) típus:** nem bontható tovább – logikai, karakter, egész, valós, mutató
- **Összetett típus (adatszerkezet):** elemi adatokból épül fel – tömb, rekord, lista …

Elemi típusok Java és Go nyelven

Fajta	Java	Go	Méret
logikai	`boolean`	`bool`	–
karakter	`char` (UTF-16 kódegység)	`rune` (= `int32`, Unicode kódpont)	16 / 32 bit
bájt	`byte` (előjeles)	`byte` (= `uint8`, előjel nélküli)	8 bit
egész	`short`, `int`, `long`	`int8`, `int16`, `int32`, `int64`, `int`	16 / 32 / 64 bit
előjel nélküli egész	–	`uint8` … `uint64`, `uint`	8 … 64 bit
valós	`float`, `double`	`float32`, `float64`	32 / 64 bit
mutató / referencia	referencia (objektumokra)	`*T` mutató	64 bit

> Go-ban az `int` mérete a géptől függ (ma szinte mindig 64 bit), Java-ban az `int` mindig 32 bit.

Egész számok ábrázolása



byte: 8 bit

tartomány: −128 … 127

negálás: bitek invertálása + 1

- **n** biten **2^n** különböző érték ábrázolható
- Előjeles egész (kettes komplemens): **-2^{n-1} … $2^{n-1} - 1$**
- Java `int` (32 bit): −2 147 483 648 … 2 147 483 647

Túlcsordulás

Ha egy művelet eredménye nem fér el a típusban, az érték **„körbefordul”** – hibaüzenet nélkül!

Java

```java
int max = Integer.MAX_VALUE;
System.out.println(max + 1);
// -2147483648

byte b = 127;
b++;                // -128
```

Go

```go
var max int32 = math.MaxInt32
max++
fmt.Println(max)
// -2147483648

var b int8 = 127
b++                 // -128
```

A típust mindig úgy válasszuk meg, hogy a várható legnagyobb érték is elférjen benne (pl. `long` / `int64`).

Valós számok: lebegőpontos ábrázolás

- Alak: **előjel × mantissza × 2^kitevő** (IEEE 754 szabvány)
- `float` / `float32`: kb. 7 értékes jegy · `double` / `float64`: kb. 15–16 értékes jegy
- A legtöbb tizedes tört (pl. 0,1) **nem ábrázolható pontosan** kettes számrendszerben

```
double s = 0.1 + 0.2;
System.out.println(s);
// 0.30000000000000004
System.out.println(s == 0.3); // false
```

Megoldás: tűréssel hasonlítunk

```
final double EPS = 1e-9;
Math.abs(s - 0.3) < EPS   // true
```

Pénzügyi számításoknál egész számot (pl. fillér, fen) vagy `BigDecimal` típust használunk.

Egész osztás és típuskonverzió

- Két egész hányadosa **egész**: `7 / 2 == 3`, a maradék: `7 % 2 == 1`
- Ha legalább az egyik operandus valós, az eredmény valós: `7 / 2.0 == 3.5`

```
double f = 9 / 5 * celsius + 32;     // HIBÁS: 9 / 5 == 1
double f = 9.0 / 5 * celsius + 32;   // helyes
```

Konverzió (típuskényszerítés):

- **Bővítő** (`int` → `long`, `int` → `double`): biztonságos, Java-ban automatikus
- **Szűkítő** (`double` → `int`, `int` → `byte`): adat veszhet el, csak kifejezetten kérve `(int) 3.99 == 3` (csonkol) · `(byte) 300 == 44` (csak az alsó 8 bit marad)
- Go-ban **minden** konverziót ki kell írni: `float64(a) / float64(b)`

Karakterek: ASCII és Unicode

- **ASCII:** 7 bit, 128 karakter (angol ábécé, számjegyek, írásjelek)
  `'A' = 65`, `'a' = 97`, `'0' = 48` → `'a' - 'A' == 32`
- **Unicode:** a világ összes írásrendszere, > 150 000 karakter
  `'中' = U+4E2D = 20013` · `'é' = U+00E9 = 233`
- **UTF-8:** változó hosszúságú kódolás (1–4 bájt) – a weben és Go-ban ez az alapértelmezett

Java

```java
char c = 'A';
System.out.println((int) c);   // 65
char d = (char) (c + 1);        // 'B'
```

Go

```go
c := 'A'                 // rune
fmt.Println(c)           // 65
fmt.Printf("%c", '中') // 中
```

A stringekkel részletesen a **4. héten** foglalkozunk.

Logikai típus és műveletek

a	b	a és b	a vagy b	nem a
H	H	H	H	I
H	I	H	I	I
I	H	H	I	H
I	I	I	I	H

Művelet	Java	Go
és	`&&`	`&&`
vagy	`||`	`||`
nem	`!`	`!`
kizáró vagy	`^`	`!=`

- **Rövidzár-kiértékelés:** `a && b` esetén ha `a` hamis, `b`-t ki sem értékeli `if (i < n && t[i] > 0)` – nem indexel túl a tömbön!

Literál, változó, konstans

- **Literál:** a programszövegbe közvetlenül írt érték
  `42`, `0x2A`, `0b101010`, `3.14`, `2.5e3`, `'A'`, `"hello"`, `true`
- **Változó:** névvel ellátott tárhely, értéke futás közben változhat
- **Konstans:** névvel ellátott érték, amely **nem változhat**

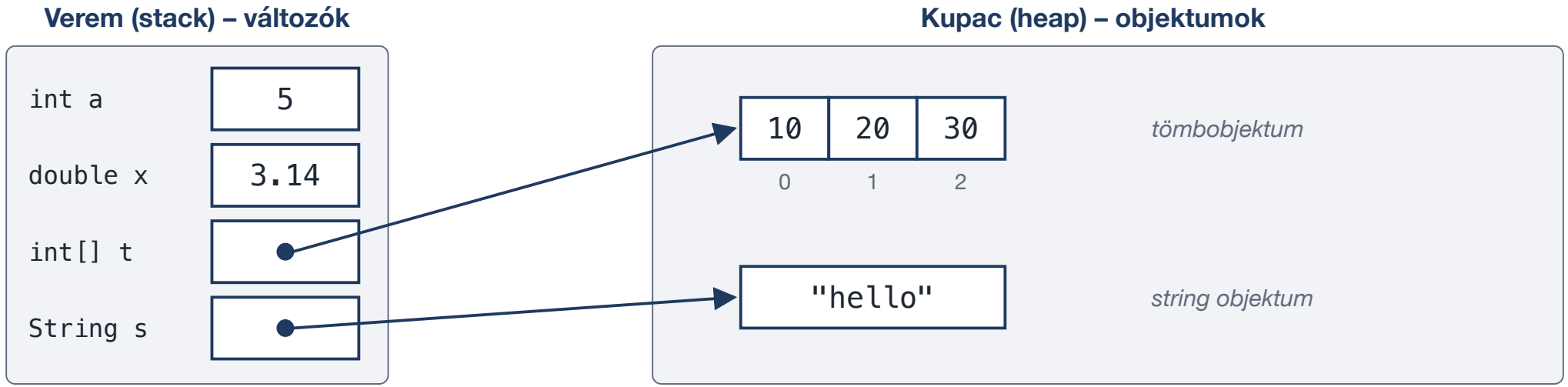
Java

```java
int count = 0;              // változó
final int DAYS = 7;         // konstans
long big = 9_000_000_000L;  // L: long
float f = 2.5f;             // f: float
```

Go

```go
var count int      // 0 a nulla érték
n := 10            // rövid deklaráció
const Days = 7     // konstans
var f float32 = 2.5
```

Mutató és referencia



primitív érték: a változóban van • referencia: a változó csak az objektum helyét tárolja

- **Mutató (pointer):** olyan adat, amelynek értéke egy **memóriacím**
- A mutató segítségével közvetve érünk el egy másik adatot
- Speciális érték: **null / nil** – „nem mutat sehova”

Referencia Java-ban, mutató Go-ban

Java

- Primitív típus (`int` , `double` …): a változó **az értéket** tárolja
- Objektum (tömb, `String` , saját osztály): a változó **referenciát** tárol
- Mutatóaritmetika nincs, a címet nem látjuk

```java
int[] a = {10, 20, 30};
int[] b = a;      // ugyanaz a tömb!
b[0] = 99;
System.out.println(a[0]);  // 99
```

Go

- Minden típus érték szerint másolódik – még a tömb (`[3]int`) is
- Mutató: `&x` a cím, `*p` a mutatott érték
- A szelet (`[]int`) és a `map` belül hivatkozást tartalmaz

```go
x := 5
p := &x        // p a címet tárolja
*p = 6
fmt.Println(x) // 6
```

A mutató/referencia teszi lehetővé a **dinamikus adatszerkezeteket** (láncolt lista, fa) – lásd 9. héttől.

Adatszerkezetek

Mi az adatszerkezet?

Az **adatszerkezet** logikailag összetartozó adatelemek együttese, ahol adottak

1. az **elemek** (és típusuk),
2. az elemek közötti **kapcsolatok**,
3. az elvégezhető **műveletek**.

> **Példa – névsor:** elemek: nevek (string) · kapcsolat: ábécésorrend (mindenkinek van előzője és rákövetkezője) · műveletek: keresés, új név beszúrása, név törlése

Az adatszerkezeteket többféle szempont szerint csoportosítjuk ▶

Csoportosítás: az elemek típusa és száma

Az elemek típusa szerint

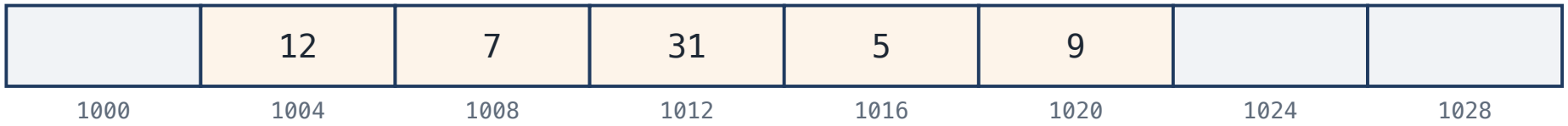
- **Homogén:** minden elem azonos típusú – tömb, string, halmaz
- **Heterogén:** az elemek típusa eltérhet – rekord (`class` / `struct`)

Az elemek száma szerint

- **Statikus:** az elemek száma a létrehozáskor rögzül, később nem változik
  Java tömb `new int[10]` , Go tömb `[10]int`
- **Dinamikus:** az elemek száma futás közben változhat
  láncolt lista, `ArrayList` , Go `slice` , fa

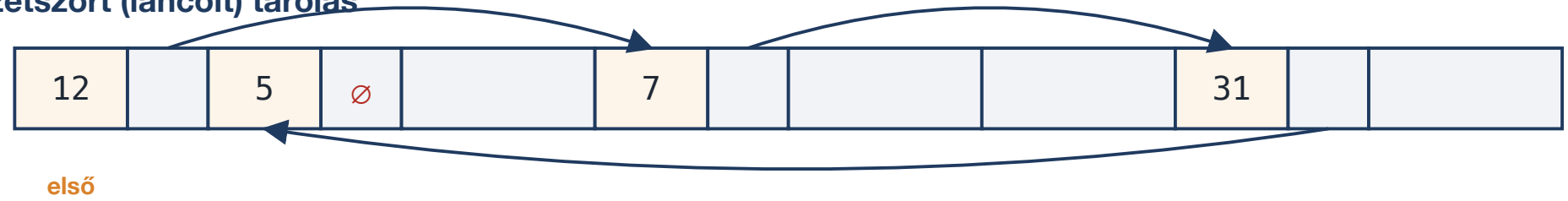
Csoportosítás: a tárolás módja

Folytonos (vektoros) tárolás



az elemek közvetlenül egymás után; a cím kiszámítható

Szétszórt (láncolt) tárolás

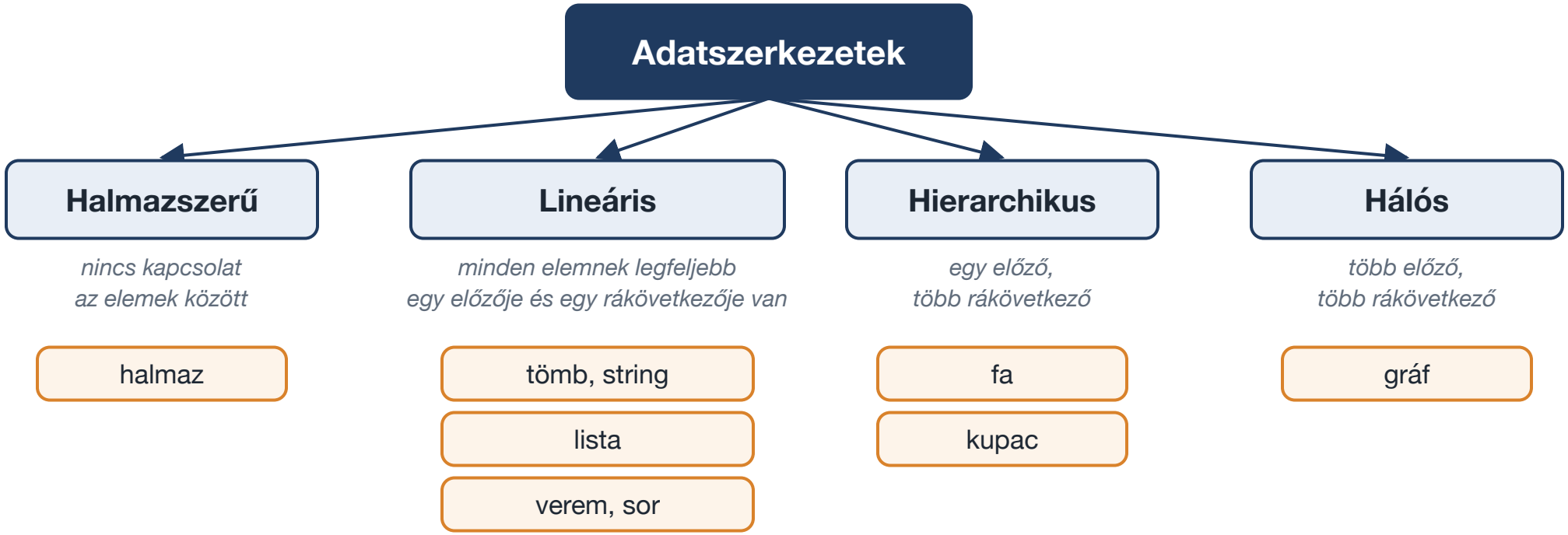


- **Folytonos:** az **i**-edik elem címe kiszámítható: `kezdőcím + (i − 1) · elemméret`
- **Szétszórt:** minden elem tárolja a következő helyét (mutató) – bővítéshez nem kell mozgatni

Csoportosítás: az elemek elérése

- **Közvetlen (direkt) elérés:** bármelyik elem azonos idő alatt elérhető, index alapján
  tömb: `t[1000]` ugyanolyan gyors, mint `t[0]`
- **Szekvenciális elérés:** egy elemhez csak az előtte lévőkön végighaladva jutunk el
  láncolt lista, (mágnesszalag)
- **Asszociatív (tartalom szerinti) elérés:** az elemet egy kulcs alapján érjük el
  hash tábla: `map["Wuhan"]`
- **Korlátozott elérés:** csak meghatározott elem érhető el
  verem (csak a legfelső), sor (csak az első)

Csoportosítás: az elemek kapcsolata



Áttekintés: a kurzus adatszerkezetei

Adatszerkezet	Elemek	Méret	Tárolás	Elérés	Kapcsolat	Hét
tömb	homogén	statikus	folytonos	közvetlen	lineáris	3
string	homogén	statikus*	folytonos	közvetlen	lineáris	4
halmaz	homogén	változó	többféle	–	nincs	6
verem, sor	homogén	dinamikus	folytonos vagy szétszórt	korlátozott	lineáris	7, 8, 11
láncolt lista	homogén	dinamikus	szétszórt	szekvenciális	lineáris	9, 10
hash tábla	homogén	dinamikus	vegyes	asszociatív	nincs	11
bináris fa	homogén	dinamikus	szétszórt (vagy tömb)	szekvenciális	hierarchikus	14, 15
rekord	heterogén	statikus	folytonos	közvetlen (mezőnév)	–	–

* Java-ban és Go-ban a string nem módosítható (immutable).

Az adatszerkezetek műveletei

Művelet	Jelentés
Létrehozás	üres szerkezet előállítása
Bővítés (beszúrás)	új elem felvétele
Törlés	elem eltávolítása
Módosítás	elem értékének cseréje
Keresés	adott tulajdonságú elem megkeresése
Bejárás (feldolgozás)	minden elem egyszeri érintése
Rendezés	elemek sorba állítása

Egy adatszerkezet akkor jó egy feladatra, ha a **leggyakoribb műveletei gyorsak**.

Absztrakt adattípus (ADT)

- **ADT:** az adatszerkezetet a **műveletei** határozzák meg – **mit** tud, nem **hogyan**
- A megvalósítás (ábrázolás) elrejthető és lecserélhető

Java – interface

```java
interface Stack {
    void push(int x);
    int pop();
    boolean isEmpty();
}
```

Go – interface

```go
type Stack interface {
    Push(x int)
    Pop() int
    IsEmpty() bool
}
```

Ugyanaz a verem megvalósítható **tömbbel** (7. hét) és **láncolt listával** (11. hét) – a felhasználó kódja nem változik.

Összefoglalás

- Programok = **algoritmusok + adatszerkezetek**
- Az algoritmus véges, egyértelmű lépéssorozat; leírása: **pszeudokód** (1-től indexelünk)
- Az **adattípus** = értékhalmaz + műveletek + ábrázolás
- Az ábrázolásnak korlátai vannak: **túlcsordulás**, **lebegőpontos pontatlanság**, **egész osztás**
- A **mutató/referencia** egy memóriacímet tárol
- Az adatszerkezetek csoportosítása: homogén/heterogén · statikus/dinamikus · folytonos/szétszórt · közvetlen/szekvenciális elérés · lineáris/hierarchikus/hálós

Következő hét: Hogyan mérjük egy algoritmus hatékonyságát? Lineáris és bináris keresés.

Ellenőrző kérdések

Igaz vagy hamis?

1. A tömb heterogén adatszerkezet.
2. A láncolt lista elemei szekvenciálisan érhetők el.
3. Java-ban `Integer.MAX_VALUE + 1` futási hibát okoz.
4. A `0.1 + 0.2 == 0.3` kifejezés értéke Java-ban hamis.
5. Egy mutató értéke egy memóriacím.
6. A statikus adatszerkezet elemeinek értéke nem változhat.

Gyakorlat

✎ A környezet beállítása

Java

```
java -version          # 17 vagy újabb
cd kod/java/h01
java Types.java        # fordítás + futtatás
```

vagy klasszikusan:

```
javac Types.java
java Types
```

Go

```
go version             # 1.23 vagy újabb
cd kod/go
go run ./h01/types
```

Formázás és ellenőrzés:

```
gofmt -w .
go vet ./...
```

✎ Bemutató programok

Program	Java	Go	Mit mutat?
Elemi típusok	`Types.java`	`h01/types`	tartományok, literálok, karakterkód, konstansok
Csapdák	`Pitfalls.java`	`h01/pitfalls`	túlcsordulás, egész osztás, 0.1 + 0.2, konverzió
Érték és referencia	`ValueRef.java`	`h01/valueref`	paraméterátadás, aliasing, null/nil, csere
Euklidesz	`Gcd.java`	`h01/gcd`	pszeudokód → kód, lépések kiírása

> Futtasd le mindegyiket, és **előtte** tippeld meg, mit fog kiírni! A meglepő sorokat beszéljük meg.

✎ Feladatok

1. **Bitek száma:** írj függvényt, amely megadja, hány bit kell egy **n** ≥ 0 egész tárolásához! (`255 → 8`, `256 → 9`)
2. **Kettes számrendszer:** írd ki egy pozitív egész bináris alakját ismételt 2-vel osztással! (`50 → 110010`)
3. **Nagybetű:** alakíts egy kisbetűt nagybetűvé a karakterkód segítségével (könyvtári függvény nélkül)!
4. **Valós összehasonlítás:** írj `almostEqual(x, y)` függvényt tűréssel!
5. **Igazságtábla:** írd ki az `és`, `vagy`, `nem`, `kizáró vagy` igazságtábláját!
6. **Túlcsordulás-figyelés:** döntsd el **összeadás előtt**, hogy `x + y` túlcsordulna-e (`int` / `int32`)!

Megoldások: `Exercises01.java`, `h01/exercises`

✎ Házi feladat

1. Mi lesz `(byte) 200` értéke Java-ban, és `int8(200)` Go-ban? Számold ki kézzel, majd ellenőrizd!
2. Hány különböző értéket tud tárolni egy 16 bites előjel nélküli egész? Mi a legnagyobb értéke?
3. Írj programot, amely kiírja a `"Hello, 世界"` szöveg minden karakterét és annak Unicode kódját!
4. Keress a mindennapi életből példát **verem**, **sor** és **fa** szerkezetre!