3. Absztrakt adattípusok

Az adatkezelés szintjei:

1. Probléma szintje.
2. Modell szintje.
3. Absztrakt adattípus szintje.
4. Absztrakt adatszerkezet szintje.
5. Adatszerkezet szintje.
6. Gépi szint.

Absztrakt adattípus: $A = (E, M)$

1. E: értékhalmaz,
2. M: műveletek halmaza.

"Absztrakt" jelző jelentése:

i. Nem ismert az adatokat tároló adatszerkezet.

ii. Nem ismertek a műveleteket megvalósító algoritmusok, a műveletek specifikációjukkal definiáltak.

Alapvető absztrakt adattípusok

3.1. Verem

Értékhalmaz: $Verem = \{\langle a_1, \ldots, a_n \rangle : a_i \in E, i = 1, \ldots, n, n \geq 0\}$
Műveletek:

$$V : Verem, x : E$$

$\{Igaz\}$	$Letesit(V)$	$\{V = \langle\rangle\}$
$\{V = V\}$	$Megszuntet(V)$	$\{Igaz\}$
$\{V = V\}$	$Uresit(V)$	$\{V = \langle\rangle\}$
$\{V = \langle a_1, \ldots, a_n \rangle\}$	$VeremBe(V, x)$	$\{V = \langle a_1, \ldots, a_n, x \rangle\}$
$\{V = \langle a_1, \ldots, a_n \rangle \wedge n > 0\}$	$VeremBol(V, x)$	$\{V = \langle a_1, \ldots, a_{n-1} \rangle \wedge x = a_n\}$
$\{V = V\}$	$NemUres(V)$	$\{NemUres = (Pre(V) \neq \langle\rangle)\}$
$\{V = \langle a_1, \ldots, a_n \rangle \wedge n > 0\}$	$Tete je(V, x)$	$\{x = a_n \wedge V = Pre(V)\}$
$\{V = \langle a_1, \ldots, a_n \rangle \wedge n > 0\}$	$Torol(V)$	$\{V = \langle a_1, \ldots, a_{n-1} \rangle\}$

Még az egyszerű absztrakt adattípusoknak is különböző megvalósítása lehet. Mindig azt a megvalósítást kell kiválasztani, amelyik az adott feladat megoldásához a legmegfelelőbb. A megvalósítást alapvetően befolyásolja a megvalósításra használt programozási nyelv. Objektum orientált nyelvek esetén (java, C++, C#) interfész (vagy absztrakt) osztály használata biztosítja azt, hogy a megvalósítástól független algoritmust készíthessünk. Pontosabban, csak a LETESIT művelet függ a megvalósítástól. A java nyelv esetén a Verem adattípus interfész osztálya a következő:

```
public interface Verem<E>{
    public void Uresit();
    public boolean NemUres();
    public void VeremBe(E x);
    public E VeremBol();
    public E Teteje();
    public void Torol();
}
```

Ha VeremL osztály egy megvalósítása a Verem adattípusnak (interfésznek), és egész számokat tartalmazó V vermet akarunk létesíteni, akkor ezt a VeremL osztály konstruktorának hívásával végezhetjük, ahol típus-paraméterként kell megadni az Integer típust (osztályt).
Megjegyzés. Típus-paraméter csak referencia típus lehet, elemi típus (int, long, float, double, char, boolean) nem. Azonban az $int \longleftrightarrow Integer$,... típuskonverzió automatikus. Tehát pl. Verem<Integer> V típus esetén alkalmazható a int x=V.VeremBol();, V.VeremBe(x); utasítás.

```
Verem<Integer> V = new VeremL<Integer>();
```

Verem megvalósítások:
VeremT: tömb adatszerkezettel,
VeremL: lánc adatszerkezettel,
VeremK: kombinált adatszerkezettel.

Példa verem adattípus használatára:
A postfix konverzió megvalósítása veremmel.

```
private static int Prior(char c){
    switch (c){
        case '(': return 0;
        case '+': return 1;
        case '-': return 1;
        case '*': return 2;
        case '/': return 2;
    }
    return -1;
}
public static String Postfix(String K){
    Verem<Character> V=new VeremL<Character>();
    V.VeremBe('(');

    String Postform="";
    char jel, op;

    for (int i=0; i<K.length(); i++){
        jel=K.charAt(i);
        if (Character.isLetter(jel)){
            Postform=Postform+jel;
        }else if (jel=='('){
            V.VeremBe(jel);
        }else if (jel==')'){
            op=V.VeremBol();
            while (op!='(') {
                Postform=Postform+op;
                op=V.VeremBol();
            }
        }else{  //műveleti jel
            while (Prior(V.Teteje()) > Prior(jel)){
                op=V.VeremBol();
                Postform=Postform+op;
            }
            V.VeremBe(jel);
        }
    }

    op=V.VeremBol();    //a veremben maradt műveleti jelek kiírása
```

```
    while (op!='(') {
        Postform=Postform+op;
        op=V.VeremBol();
    }
    return Postform;
}
```

3.2. Sor

Értékhalmaz: $Sor = \{\langle a_1,\ldots,a_n\rangle : a_i \in E, i = 1,\ldots,n, n \geq 0\}$
Műveletek:

$$S : Sor, x : E$$

$\{Igaz\}$	$Letesit(S)$	$\{S = \langle\rangle\}$
$\{S = S\}$	$Megszuntet(S)$	$\{Igaz\}$
$\{S = S\}$	$Uresit(S)$	$\{S = \langle\rangle\}$
$\{S = \langle a_1,\ldots,a_n\rangle\}$	$SorBa(S,x)$	$\{S = \langle a_1,\ldots,a_n,x\rangle\}$
$\{S = \langle a_1,\ldots,a_n\rangle \wedge n > 0\}$	$SorBol(S,x)$	$\{x = a_1 \wedge S = \langle a_2,\ldots,a_n\rangle\}$
$\{S = \langle a_1,\ldots,a_n\rangle\}$	$Elemszam(S)$	$\{Elemszam = n\}$
$\{S = \langle a_1,\ldots,a_n\rangle \wedge n > 0\}$	$Elso(S,x)$	$\{x = a_1 \wedge S = Pre(S)\}$
$\{S = \langle a_1,\ldots,a_n\rangle \wedge n > 0\}$	$Torol(S)$	$\{S = \langle a_2,\ldots,a_n\rangle\}$

```
public interface Sor<E>{
   public void Uresit();
   public int Elemszam();
   public void SorBa(E x);
   public E SorBol();
   public E Elso();
   public void Torol();
}
```

Sor megvalósítások:
SorT: tömb adatszerkezettel,
SorL: lánc adatszerkezettel,
SorK: kombinált adatszerkezettel.

3.3. Prioritási Sor

Értékhalmaz: $PriSor = \{\, S : S \subseteq E\,\}$, E-n értelmezett a $\leq$ lineáris rendezési reláció.
Műveletek:

$$S : PriSor, x : E$$

$\{Igaz\}$	$Letesit(S,\leq)$	$\{S = \emptyset\}$
$\{S = S\}$	$Megszuntet(S)$	$\{Igaz\}$
$\{S = S\}$	$Uresit(S)$	$\{S = \emptyset\}$
$\{S = S\}$	$SorBa(S,x)$	$\{S = Pre(S) \cup \{x\}\}$
$\{S \neq \emptyset\}$	$SorBol(S,x)$	$\{x = \min(Pre(S)) \wedge Pre(S) = S \cup \{x\}\}$
$\{S = \{a_1,\ldots,a_n\}\}$	$Elemszam(S)$	$\{Elemszam = n\}$
$\{S \neq \emptyset\}$	$Elso(S,x)$	$\{x = \min(Pre(S)) \wedge Pre(S) = S\}$
$\{S \neq \emptyset\}$	$Torol(S)$	$\{S = Pre(S) - \{\min(Pre(S))\}\}$

```
public interface PriSor<E extends Comparable<E>>
      extends Sor<E>{
}
```

Prioritási sor megvalósítások:
PriSorT: kupac-tömb adatszerkezettel,
PriSorR: kupac-tömb adatszerkezettel, a rendezési reláció konstruktor paraméter.
Példa prioritási sor alkalmazására.

Probléma: Halmaz k-adik legkisebb elemének kiválasztása.
Bement: $H = \{a_1, \ldots, a_n\}$, különböző egész szám, $1 \leq k \leq n$.
Kimenet: $a_i \in H$, amelyre $|\{x : x \in H, x \leq a_i\}| = k$,

```
import java.io.*; import java.util.Scanner;
public class Kivalaszto{
   public static int Kivalaszt(){
       Scanner stdin = new Scanner(System.in);
       System.out.println("Elemek száma?");
       int n=stdin.nextInt(); stdin.nextLine();
       System.out.println("Hanyadik?");
       int k=stdin.nextInt(); stdin.nextLine();
       if (k>n) return 0;
       int x=0;
       PriSor<Integer> S = new PriSorT<Integer>(n);
       for (int i=1; i<=n; i++){
           x=stdin.nextInt();
           S.SorBa(x);
       } stdin.close();
       for (int i=1; i<=k; i++)
           x=S.SorBol();
       return x;
   }
   public static void main (String[] args){
       int x=new Kivalaszto().Kivalaszt();
       System.out.println("A keresett szám: "+x);
   }
}
```

3.4. Lista

Értékhalmaz: $Lista = \{\langle a_1, \ldots, a_{i-1}\rangle\langle a_i, \ldots, a_n\rangle : a_j \in E, j = 1, \ldots, n, 0 \leq i-1 \leq n, n \geq 0\}$
Műveletek:

$$L, L1, L2 : Lista, x : E$$

$\{Igaz\}$	$Letesit(L)$	$\{L = \langle\rangle\langle\rangle\}$
$\{L = L\}$	$Megszuntet(L)$	$\{Igaz\}$
$\{L = L\}$	$Uresit(L)$	$\{L = \langle\rangle\langle\rangle\}$
$\{L = L\}$	$Urese(L)$	$\{Urese = (Pre(L) = \langle\rangle\langle\rangle\}$
$\{L = L\}$	$Elejen(L)$	$\{= Pre(L) = \langle\rangle\langle a_1,\ldots,a_n\rangle\}$
$\{L = L\}$	$Vegen(L)$	$\{Vegen = L = \langle a_1,\ldots,a_n\rangle\langle\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle\}$	$Elejere(L)$	$\{L = \langle\rangle\langle a_1,\ldots,a_n\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle\}$	$Vegere(L)$	$\{L = \langle a_1,\ldots,a_n\rangle\langle\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle \wedge i \leq n\}$	$Tovabb(L)$	$\{L = \langle a_1,\ldots,a_{i-1},a_i\rangle \langle a_{i+1},\ldots,a_n\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle \wedge i \leq n\}$	$Kiolvas(L,x)$	$\{x = a_i \wedge L = Pre(L)\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,a_{i+1}\ldots,a_n\rangle \wedge i \leq n\}$	$Modosit(L,x)$	$\{L = \langle a_1,\ldots,a_{i-1}\rangle \langle x,a_{i+1},\ldots,a_n\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle\}$	$Bovit(L,x)$	$\{L = \langle a_1,\ldots,a_{i-1}\rangle \langle x,a_i,\ldots,a_n\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,a_{i+1},\ldots,a_n\rangle \wedge i \leq n\}$	$Torol(L)$	$\{L = \langle a_1,\ldots,a_{i-1}\rangle \langle a_{i+1},\ldots,a_n\rangle\}$
$\{L1 = \langle\alpha_1\rangle\langle\beta_1\rangle, L2 = \langle\alpha_2\rangle\langle\beta_2\rangle\}$	$Kapcsol(L1,L2)$	$\{L1 = \langle\alpha_1\rangle\langle\alpha_2\beta_1\rangle, L2 = \langle\rangle\langle\beta_2\rangle\}$

```
public interface Lista<E>{
   public void Uresit();
   public boolean Urese();
   public boolean Elejen();
   public boolean Vegen();
   public void Elejere();
   public void Vegere();
   public void Tovabb();
   public E Kiolvas();
   public void Modosit(E x);
   public void Bovit(E x);
   public void Torol();
   public void Kapcsol(Lista<E> l2);
}
```

Lista megvalósítások:
ListaL: lánc adatszerkezettel,
ListaT: tömb adatszerkezettel.
Példa lista alkalmazására.

Probléma: Két rendezett lista összefűzése rendezett listává.
Bement: $L1 = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle$, $(a_k \leq a_{k+1}, 1 \leq k < n)$
$L2 = \langle b_1,\ldots,b_{j-1}\rangle\langle b_j,\ldots,b_m\rangle$, $(b_k \leq b_{k+1}, 1 \leq k < m)$
Kimenet: $L1 = \langle\rangle\langle c_1,\ldots,c_{n+m}\rangle$, $(c_i \leq c_{i+1}, 1 \leq i < n+m)$
$\{c_1,\ldots,c_{n+m}\} = \{a_1,\ldots,a_n\} \cup \{b_1,\ldots,b_m\}$
$L2 = \langle\rangle\langle\rangle$

```
public static void ListaFuz(Lista<Integer> L1, Lista<Integer> L2){
    L1.Elejere();
    L2.Elejere();
    int a=L1.Kiolvas();
```

```
    int b=L2.Kiolvas();
    while (!L1.Vegen() && !L2.Vegen()){
        if (a<=b){
            L1.Tovabb();
            if (!L1.Vegen())
                a=L1.Kiolvas();
        }else{
            L2.Tovabb();
            L1.Kapcsol(L2);
            L1.Tovabb();
            if (!L2.Vegen())
                b=L2.Kiolvas();
        }
    }//while
    if (!L2.Vegen()){
        L2.Vegere();
        L1.Kapcsol(L2);
    }
}
```

A **while** ciklus végrehajtásának egy adott pillanatában a ciklusmag elején:
$L1 = \langle c_1,\ldots,c_k\rangle\langle a_i,\ldots,a_n\rangle$, $a = a_i$ és
$L2 = \langle\rangle\langle b_j,\ldots,b_m\rangle$, $b = b_j$ és
Teljesül, hogy

$$\{c_1,\ldots,c_k\} = \{a_1,\ldots,a_{i-1}\} \cup \{b_1,\ldots,b_{j-1}\}$$
$$c_1 \le c_2 \le \ldots \le c_k$$
$$c_k \le a_i, \quad c_k \le b_j$$

Ez a feltétel ciklusinvariáns lesz, tehát az algoritmus helyes.
A LISTAFUZ algoritmus futási idejének elemzése.
Tegyük fel, hogy minden lista művelet futási ideje konstans.
Legjobb eset: $T_{lj}(n,m) = \Theta(\min(n,m))$.
legrosszabb eset: $T_{lr}(n,m) = \Theta(n+m)$.
Átlagos eset: $T_a(n,m) = \Theta(n+m)$.

$$\begin{aligned} T_a(n,m) &= \frac{1}{2}(\Theta(n) + \frac{1}{(m+1)}\sum_{j=0}^{m} j + \Theta(m) + \frac{1}{n+1}\sum_{i=0}^{n} i) \\ &= \frac{1}{2}(\Theta(n) + \frac{1}{(m+1)}\frac{m(m+1)}{2} + \Theta(m) + \frac{1}{n+1}\frac{n(n+1)}{2}) \\ &= \frac{1}{2}(\Theta(n) + \frac{m}{2} + \Theta(m) + \frac{n}{2}) = \Theta(n) + \Theta(m) + \Theta(n) + \Theta(m) \\ &= \Theta(n+m) \end{aligned}$$

A LISTAFUZ csak egész elemtípusú listákra alkalmazható. Látható azonban, hogy csak az a<=b utasítás függ az elemtípustól. Ez is csak annyiban, hogy az elemtípuson értelmezve kell legyen a <= lineáris rendezési reláció. A típus-paraméterezett metódusok lehetővé teszik, hogy ilyen esetekben egyetlen kódot írhassunk. A feltétel az, hogy az E elemtípus a Comparable<E> osztály leszármazottja legyen, tehát implementáljs a compareTo metódust.
Az ilyen esetet, amikor megköveteljük, hogy az elemtípus rendelkezzen meghatározott műveletekkel, típuskorlátozásnak nevezzük.

```
public static <E extends Comparable<E> > //típuskorlátozás
    void ListaFuz(Lista<E> L1, Lista<E> L2){
    L1.Elejere();
    L2.Elejere();
    E a=L1.Kiolvas();
```

```
    E b=L2.Kiolvas();
    while (!L1.Vegen() && !L2.Vegen()){
        if (a.compareTo(b)<=0){ // a<=b
            L1.Tovabb();
            if (!L1.Vegen())
                a=L1.Kiolvas();
        }else{
            L2.Tovabb();
            L1.Kapcsol(L2);
            L1.Tovabb();
            if (!L2.Vegen())
                b=L2.Kiolvas();
        }
    }//while
    if (!L2.Vegen()){
        L2.Vegere();
        L1.Kapcsol(L2);
    }
}
```

3.5. Kétirányú lista

Értékhalmaz: $Lista2 = \{\langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle : a_i \in E, i = 1,\ldots,n, n \geq 0\}$
Műveletek: Lista műveletei + *Vissza*

$$L : Lista2$$

$$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle \wedge i > 0\} \quad Vissza(L) \quad \{L = \langle a_1,\ldots,a_{i-2}\rangle \langle a_{i-1},a_i,a_{i+1},\ldots,a_n\rangle\}$$

```
public class Lista2<E> extends Lista<E>{
   public void Vissza();
}
```

Kétirányú lista (Lista2) megvalósítások
Lista2L: kétirányú lánc adatszerkezettel.

3.6. Tömb

Értékhalmaz: $Tomb = \{\langle a_1,\ldots,a_n\rangle : a_i \in E \cup \{\perp\}, i = 1,\ldots,n, n \geq 1\}$
Műveletek:

$$T : Tomb, x : E, i : Integer$$

$$\begin{array}{rcl} \{Igaz\} & Letesit(T,n) & \{T = \langle \perp,\ldots,\perp\rangle\} \\ \{T = \langle a_1,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n\rangle \wedge 1 \leq i \leq n\} & Kiolvas(T,i,x) & \{T = Pre(T) \wedge x = a_i\} \\ \{T = \langle a_1,\ldots,a_i,\ldots,a_n\rangle \wedge 1 \leq i \leq n\} & Modosit(T,i,x) & \{T = \langle a_1,\ldots,x,\ldots,a_n\rangle\} \end{array}$$

Megvalósítás

```
//Letesit(T,n):
E[] T=new E[n];
//Kiolvas(T,i,x):
```

```
x=T[i];
//Modosit(T,i,x):
T[i]=x;
```

A tömb típus rendelkezik iterátorral, tehát a

```
for (int i=0; i<T.length; i++) M(T[i])
```

ismétlés helyett írhatjuk az egyszerű

```
for (E x:T) M(x)
```

utasítást.

3.7. Sorozat

Értékhalmaz: $Sorozat = \{\langle a_1,\ldots,a_n\rangle : a_i \in E, i = 1,\ldots,n, n \geq 0\}$
Műveletek:

$$S : Sorozat, x : E, i : Integer$$

$$\begin{array}{rcl}
\{Igaz\} & Letesit(S) & \{S = \langle\rangle\} \\
\{S = S\} & Megszuntet(S) & \{Igaz\} \\
\{S = S\} & Uresit(S) & \{S = \langle\rangle\} \\
\{S = \langle a_1,\ldots,a_n\rangle\} & Elemszam(S) & \{Elemszam = n\rangle\} \\
\{S = \langle a_1,\ldots,a_i,\ldots,a_n\rangle \wedge 1 \leq i \leq n\} & Kiolvas(S,i,x) & \{x = a_i \wedge S = Pre(S)\} \\
\{S = \langle a_1,\ldots,a_i,\ldots,a_n\rangle \wedge 1 \leq i \leq n\} & Modosit(S,i,x) & \{S = \langle a_1,\ldots,x,\ldots,a_n\rangle\} \\
\{S = \langle a_1,\ldots,a_i,a_{i+1},\ldots,a_n\rangle \wedge 0 \leq i \leq n\} & Bovit(S,i,x) & \{S = \langle a_1,\ldots,a_i,x,a_{i+1},\ldots,a_n\rangle\} \\
\{S = \langle a_1,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n\rangle \wedge 1 \leq i \leq n\} & Torol(S,i) & \{S = \langle a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n\rangle\} \\
\{S = S\} & IterKezd(S,I) & \{\} \\
\{I = I\} & IterAd(I,x) & \{\} \\
\{I = I\} & IterVege(I) & \{\}
\end{array}$$

```
public interface Sorozat<E> extends Iterable<E>{
   public void Uresit();
   public int Elemszam();
   public E Kiolvas(int i);
   public void Modosit(int i, E x);
   public void Bovit(int i, E x);
   public void Torol(int i);
   public Iterator<E> iterator();
}
```

Sorozat megvalósítások
SorozatBKF: bináris keresőfa adatszerkezettel,
SorozatAVL: AVL-kiegyensúlyozott bináris keresőfa adatszerkezettel.

3.8. Halmaz

Értékhalmaz: $Halmaz = \{\, H : H \subseteq E\}$
Műveletek:

$$H : Halmaz, x : E, I : Iterator$$

$$\begin{array}{rcl}
\{Igaz\} & Letesit(H) & \{H=\emptyset\} \\
\{H=H\} & Megszuntet(H) & \{Igaz\} \\
\{H=H\} & Uresit(H) & \{H=\emptyset\} \\
\{H=\{a_1,\ldots,a_n\}\} & Elemszam(H) & \{Elemszam=n\} \\
\{H=H\} & Eleme(H,x) & \{Eleme=x\in Pre(H)\} \\
\{H=H\} & Bovit(H,x) & \{H=Pre(H)\cup\{x\}\} \\
\{H=H\} & Torol(H,x) & \{H=Pre(H)-\{x\}\} \\
\{H=H\} & IterKezd(H,I) & \{\} \\
\{I=I\} & IterAd(I,x) & \{\} \\
\{I=I\} & IterVege(I) & \{\}
\end{array}$$

```
public interface Halmaz<E> extends Iterable<E>{
   public void Uresit();
   public int Elemszam();
   public boolean Eleme(E x);
   public void Bovit(E x);
   public boolean Torol(E x);
   public E Keres(E x);
}
```

3.9. RHalmaz

Értékhalmaz: $Halmaz = \{\ H : H \subseteq E$, E-n értelmezett a $\leq$ lineáris rendezési reláció. $\}$
Műveletek: Halmaz műveletek +

$$H : Halmaz, x : E, I : Iterator$$

$$\begin{array}{rcl}
\{H\neq\emptyset\} & Min(H) & \{=min\{x : x\in H\}\} \\
\{H\neq\emptyset\} & Max(H) & \{=max\{x : x\in H\}\} \\
\{H\neq\emptyset\} & Elozo(H,x) & \{=max\{y : y\in H-x\wedge y\leq x\}\} \\
\{H\neq\emptyset\} & Koveto(H,x) & \{=min\{y : y\in H-x\wedge x\leq y\}\}
\end{array}$$

```
public interface RHalmaz<E extends Comparable<E>>
   extends Halmaz<E>, Iterable<E>{
      public E Min();
      public E Max();
      public E Elozo(E x);
      public E Koveto(E x);
}
```

A Halmaz adattípus megvalósításai:
HalmazB: bitvektor adatszerkezettel; E=1..n
HalmazT: tömb adatszerkezettel,
HalmazL: lánc adatszerkezettel,
HalmazHL: hasítótábla (ütközésfeloldás láncolással) adatszerkezettel,
HalmazHN: hasítótábla (ütközésfeloldás nyílt címzéssel) adatszerkezettel.
Az RHalmaz adattípus megvalósításai:
RHalmazT: rendezett tömb adatszerkezettel,
RHalmazL: rendezett lánc adatszerkezettel,
RHalmazBKF: bináris keresőfa adatszerkezettel,

RHalmazAVL: AVL-kiegyensúlyozott bináris keresőfa adatszerkezettel,
RHalmazPFFa: piros-fekete fa adatszerkezettel.

```
Forall x in H Do
  M(x);
```

Diszkrét ismétléses vezérlés megvalósításai iterátorral.
Pascal megvalósítás:

```
IterKezd(H,I);
While Not IterVege(I) Do Begin
  IterAd(I,x);
  M(x);
End;
```

java megvalósítás:

```
Iterator<E> I=H.iterator(); //IterKezd(H,I)
while (I.hasNext()){        //!IterVege(I)
   x=I.next();              //IterAd(I,x)
   M(x);
}
```

vagy kiterjesztett for-ciklussal:

```
for (E x:H)
  M(x);
```

Példa Halmaz absztrakt adattípus alkalmazására.

Probléma: Adott sorozat különböző elemének kiválasztása.
Bement: $S = \langle a_1, \ldots, a_n \rangle$ egész számok.
Kimenet: $H = \{x : x \in S\}$.

```
import java.io.*; import java.util.*;
public class HalmazPelda{
    public static void main (String[] args){
        Scanner stdin = new Scanner(System.in);
        Halmaz<Integer> H=new HalmazL<Integer>();
        int x;
        System.out.println("Elemek száma?");
        int n=stdin.nextInt(); stdin.nextLine();
        for (int i=1; i<=n; i++){
            x=stdin.nextInt();
            if (!H.Eleme(x))
                H.Bovit(x);
        }
        stdin.close();
        Iterator<Integer> it=H.iterator();  //IterKezd(H,it)
        while (it.hasNext()){               //!IterVege(it)
            x=it.next();                    //IterAd(it,x)
            System.out.print(x+" ");
        }
        System.out.println();
        for (int e:H)               //H elemeinek kiíratása for ciklussal
            System.out.print(e+" ");
   }
}
```

3.10. Függvény (parciális függvény)

Értékhalmaz:

$Fuggveny = \{\ F : F \subseteq E = K \times A, (\forall k \in K)(\forall x,y \in A)((k,x) \in F \wedge (k,y) \in F \Rightarrow x = y)\}$

Műveletek:

$$F : Fuggveny, k : K, x : A$$

$\{Igaz\}$	$Letesit(F)$	$\{F = \emptyset\}$	(1)
$\{F = F\}$	$Megszuntet(F)$	$\{Igaz\}$	(2)
$\{F = F\}$	$Uresit(F)$	$\{F = \emptyset\}$	(3)
$\{F = F\}$	$Eleme(F,k)$	$\{Eleme = (\exists x \in A)((k,x) \in F)\}$	(4)
$\{F = F\}$	$Elemszam(F)$	$\{Elemszam = \lvert F \rvert\}$	(5)
$\{(\exists a \in A)((k,a) \in F)\}$	$Kiolvas(F,k,x)$	$\{x = a \wedge F = Pre(F)\}$	(6)
$\{(\forall a \in A)((k,a) \notin F)\}$	$Kiolvas(F,k,x)$	$\{F = Pre(F) \wedge x = Pre(x)\}$	(7)
$\{(\exists a \in A)((k,a) \in F)\}$	$Modosit(F,k,x)$	$\{F = Pre(F) - \{(k,a)\} \cup \{(k,x)\}\}$	(8)
$\{(\forall a \in A)((k,a) \notin F)\}$	$Modosit(F,k,x)$	$\{F = Pre(F) \wedge x = Pre(x)\}$	(9)
$\{(\forall a \in A)((k,a) \notin F)\}$	$Bovit(F,k,x)$	$\{F = Pre(F) \cup \{(k,x)\}\}$	(10)
$\{(\exists a \in A)((k,a) \in F)\}$	$Bovit(F,k,x)$	$\{F = Pre(F) \wedge x = Pre(x)\}$	(11)
$\{(\exists a \in A)((k,a) \in F)\}$	$Torol(F,k)$	$\{F = Pre(F) - \{(k,a)\}\}$	(12)
$\{(\forall a \in A)((k,a) \notin F)\}$	$Torol(F,k)$	$\{F = Pre(F)\}$	(13)
$\{F = F\}$	$IterKezd(F,I)$	$\{\}$	(14)
$\{I = I\}$	$IterAd(I,k,x)$	$\{\}$	(15)
$\{I = I\}$	$IterVege(I)$	$\{\}$	(16)

```
public interface Fuggveny<K, A>
    extends Iterable<KulcsPar<K, A>>{
    public int Elemszam();
    public void Uresit();
    public boolean Eleme(K k);
    public void Bovit(K k, A a);
    public boolean Torol(K k);
    public A Kiolvas(K k);
    public void Modosit(K k, A a);
}

public class KulcsPar<K, A > implements Cloneable{
   public K kulcs;
   public A adat;
   public KulcsPar(K k, A a){
      kulcs=k; adat=a;
   }
   public boolean equals(Object x){
      return this.kulcs.equals( ((KulcsPar<K,A>)x).kulcs );
   }
   public int hashCode(){
      return this.kulcs.hashCode();
   }
}

Forall (k,x) in F Do
  M(k,x);
```

$\equiv$

```
KulcsPar<K,A> par;
Iterator<KulcsPar<K,A>> I=F.iterator(); //IterKezd(F,I)
while (I.hasNext()){                   //!IterVege(I)
   par=I.next();                       //IterAd(I,par)
   M(par.kulcs, par.adat);
}

for (KulcsPar<K,A> par : F)
  M(par.kulcs, par.adat);
```

A Függvény adattípus megvalósításai:
FuggvenyT: tömb adatszerkezettel, az értelmezési tartomány (K) típusa Integer;
public class FuggvenyT<A> implements Fuggveny<Integer, A>.
FuggvenyH: <k,a> párok halmazaként, ekkor konstruktor paraméterként kell megadni, hogy milyen halmaz ábrázolást akarunk:

- "Lanc" lánc adatszerkezet
- "Tomb" tömb adatszerkezet
- "HasitL" hasítótábla láncolással adatszerkezet
- "HasitN" hasítótábla nyílt címzéssel adatszerkezet

FuggvenyR: az értelmezési tartományon értelmezett lin. rendezés, ekkor konstruktor paraméterként kell megadni, hogy milyen halmaz ábrázolást akarunk:

- "BKF" bináris keresőfa
- "AVLFa" AVL-keresőfa
- "PFFa" piros-fekete keresőfa

3.11. Reláció

Értékhalmaz:
$Relacio = \{\ R : R \subseteq E = K \times A\}$
Műveletek:

$$R : Relacio, k : K, a : A$$

$\{Igaz\}$	$Letesit(R)$	$\{F = \emptyset\}$
$\{F = F\}$	$Megszuntet(R)$	$\{Igaz\}$
$\{F = F\}$	$Uresit(R)$	$\{F = \emptyset\}$
$\{F = F\}$	$Eleme(R,k,a)$	$\{Eleme = ((k,a) \in R)\}$
$\{F = F\}$	$Elemszam(R)$	$\{Elemszam = \lvert R\rvert\}$
$\{(k,a) \notin R)\}$	$Bovit(R,k,a)$	$\{R = Pre(R) \cup \{(k,a)\}\}$
$\{(k,a) \in R)\}$	$Torol(R,k,a)$	$\{R = Pre(R) - \{(k,a)\}\}$
$\{R = R)\}$	$KTorol(R,k)$	$\{R = Pre(R) - \{(k,a) : (k,a) \in Pre(R)\}\}$
$\{R = R)\}$	$ATorol(R,a)$	$\{R = Pre(R) - \{(k,a) : (k,a) \in Pre(R)\}\}$
$\{R = R)\}$	$KPar(R,k)$	$\{\{a : (k,a) \in Pre(R)\}\}$
$\{R = R)\}$	$APar(R,a)$	$\{\{k : (k,a) \in Pre(R)\}\}$
$\{R = R\}$	$IterKezd(R,I)$	$\{\}$
$\{I = I\}$	$IterAd(I,k,a)$	$\{\}$
$\{I = I\}$	$IterVege(I)$	$\{\}$

```
public interface Relacio<K, A>
    extends Iterable<Par<K, A>>{
    public int Elemszam();
    public void Uresit();
    public boolean Eleme(K k, A a);
    public void Bovit(K k, A a);
    public boolean Torol(K k, A a);
    public void KTorol(K k);
    public void ATorol(A a);
    public Halmaz<A> KPar(K k);
    public Halmaz<K> APar(A a);
}
```

A Relacio adattípus megvalósításai:
RelacioL: lánc adatszerkezettel,
RelacioBKF: bináris keresőfa adatszerkezettel, mind K, mind A rendezett típus kell legyen.
RelacioAVL: AVL-fa bináris keresőfa adatszerkezettel, mind K, mind A rendezett típus kell legyen.