

## Műveletek Halmazokkal

### Feladat:

Készítsen egy halmaz típust! Alkalmazzon osztályt! A halmazt dinamikusan lefoglalt tömb segítségével ábrázolja! Implementálja a szokásos műveleteket, alkalmazzon kivételkezelést és bontsa modulokra a programját! A teszt környezet adjon lehetőséget két halmaz létrehozására, feltöltésére, a halmaz műveleteinek kipróbálására, és hívjon meg egy olyan barát-operátort is, amely kiszámítja két halmaz metszetét! Törekedjen a metszetképzés műveletigényének minimalizálására, a dokumentációban mutasson rá a saját megoldásának műveletigényére! Tegye lehetővé két halmaz típusú változó közötti értékadást!

### Specifikáció:

$$S = \text{Set}(\mathbb{R});$$

$$V = \text{Vect}(\mathbb{N}, S);$$

$$A = S \times S \times S$$

$$A \quad B \quad C$$

$$B = S \times S$$

$$A' \quad B'$$

$$Q = (A = A' \wedge B = B')$$

$$R = (Q \wedge (C = A \cap B))$$

### Megoldás:

$C := \emptyset$	
$ A  \leq  B $	
$D := A$	$D := B$
$A \neq \emptyset$	
$x \in A$	
$x \in A \wedge x \in B$	
$C := C \cup \{x\}$	$SKIP$
$A := D \setminus \{x\}$	
$\text{Ret}(C)$	

### Jelölések:

A kódolás leegyszerűsítéséért vezessük be a következő jelöléseket:

$$A \cup B = A + B \quad (\text{Unióképzés})$$

$$A \setminus B = A - B \quad (\text{Különbséghalmaz})$$

$$A \cap B = A * B \quad (\text{Metszetképzés})$$

$$\text{szim\_dif}(A, B) = (A \cup B) \setminus (A \cap B) \quad (\text{Szimmetrikus differencia})$$

**Függvények:**

$$A.\text{ures}() \Leftrightarrow A = \emptyset \quad (\text{Üreshalmaz})$$

$$A.\text{elemszam}() = |A| \quad (\text{Elemszám})$$

$$A.\text{eleme}(x) \Leftrightarrow x \in A \quad (\text{Tartalmazás eldöntése})$$

$$A.\text{get\_elem}(x) = y \mid \forall x \in \mathbb{N}_0 : \exists ! y \in A : A_x = y \quad (\text{Egy elem meghatározása})$$

**Forráskód – fejállomány (halmaz.h):**

```
#ifndef HALMAZ_H
#define HALMAZ_H
#include <iostream>

class halmaz{
private:
    double* h;
    int elem_szam;
public:
    //ALAPERTELMEZES - Ureshalmaz
    halmaz(){
        elem_szam=0;
    }
    //KIVETELKEZELES
    enum exception{ EMPTYSET, WRONGVAL, WRONGITEM };
    //EGYSZERU FUGGVENYEK
    bool ures(){
        bool ureshalmaz;
        if(elem_szam>=1){
            ureshalmaz=false;
        }else{
            ureshalmaz=true;
        }
        return ureshalmaz;
    }
    int elemszam(){ return elem_szam; }
    //TARTALMAZAS VIZSGALATA
    bool eleme(const double x){
        bool found=false;
        if(elem_szam>=1){
            int i=0;
            while((i<elem_szam)&&(found==false)){
                if(h[i]==x){ found=true; }
                ++i;
            }
        }
        return found;
    }
    //EGY ELEM VISSZAADASA
    double elem(const int x){
        double z=0;
        bool siker=false;
        if(elem_szam>=1){
            if((x>=0)&&(x<elem_szam)){
                z=h[x];
                siker=true;
            }
        }
        if(siker==false){
```

```

        throw WRONGITEM;
    }else{
        return z;
    }
}
/*
 * TARTALOM KIIRASA
 *
 * Hasznalat:
 * a.kiir();
 *
 * Informaciok:
 * a - halmaz
 * Ez az eljárás gyönyörűen olvasható formában
 * kiírja a halmaz tartalmát a Standard Outputra.
 */
void kiir(){
    if(elem_szam>=1){
        std::cout << "{";
        for(int i=0; i<elem_szam; ++i){
            std::cout << h[i];
            if(i<elem_szam-1){
                std::cout << ", ";
            }
        }
        std::cout << "}";
    }else{
        std::cout << "ureshalmaz";
    }
}
/*
 * BESZURAS
 *
 * Hasznalat:
 * a.insert(x);
 *
 * Informaciok:
 * a - halmaz
 * x - double
 * Az eljárás beszúr egy elemet, megőrizve a halmaz
 * monoton növekvő reprezentációját, éppen ezért
 * egy kis se talán lassú és memóriaigényes, de
 * éppen a monotonitás miatt gyorsak a halmazműveleti
 * eljárások, illetve függvények.
 * Ez az eljárás ureshalmazra is működik.
 */
void insert(const double x){
    if(elem_szam>=1){
        double* tmp;
        int index1=0;
        bool inserted=false;
        bool alexists=false;
        tmp = new double[elem_szam+1];
        for(int i=0; i<elem_szam; ++i){
            if(alexists==false){
                if((x<h[i])&&(inserted==false)){
                    tmp[index1]=x;
                    ++index1;
                    inserted=true;
                }else{
                    if(x==h[i]){
                        alexists=true;
                    }
                }
            }
            tmp[index1]=h[i];
            ++index1;
        }
        //Ha x még mindig nincs bent
        if((alexists==false)&&(inserted==false)){
            tmp[index1]=x;
            ++index1;
        }
    }
}

```

```

        inserted=true;
    }
    if(inserted==true){
        delete[] h;
        h = new double[index1];
        for(int i=0; i<index1; ++i){
            h[i]=tmp[i];
        }
        elem_szam=index1;
    }
    delete[] tmp;
}
else{
    h = new double[1];
    h[0]=x;
    elem_szam=1;
}
}
/*
 * ELEM ELTAVOLITASA
 *
 * Hasznalat:
 * a.dispose(x);
 *
 * Informaciok:
 * a - halmaz
 * x - double
 * Az eljárás eltávolít egy x elemet a halmazból,
 * felteve, hogy van ilyen elem a halmazban. Amennyiben
 * nincs, akkor a halmaz tartalma nem változik meg,
 * azonban kivétel történik.
 */
void dispose(const double x){
    if(elem_szam>=1){
        double* tmp;
        int index1=0;
        tmp = new double[elem_szam];
        for(int i=0; i<elem_szam; ++i){
            if(h[i]!=x){
                tmp[index1]=h[i];
                ++index1;
            }
        }
        //Ha az eltávolítás sikerült
        if(index1==(elem_szam-1)){
            delete[] h;
            if(index1>=1){
                h = new double[index1];
                for(int i=0; i<index1; ++i){
                    h[i]=tmp[i];
                }
            }
            elem_szam=index1;
            delete[] tmp;
        }
        else{
            delete[] tmp;
            throw WRONGVAL;
        }
    }
    else{
        throw EMPTYSET;
    }
}
//HALMAZ KIURITESE
void kiurit(){
    if(elem_szam>=1){
        delete[] h;
        elem_szam=0;
    }
}
//ERTEKADAS
halmaz operator=(const halmaz &x){
    if(elem_szam>=1){ delete[] h; }
    if(x.elem_szam>=1){

```

```

        h = new double[x.elem_szam];
        for(int i=0; i<x.elem_szam; ++i){
            h[i]=x.h[i];
        }
    }
    elem_szam=x.elem_szam;
}
//BARAT-FUGGVENYEK
friend halmaz operator+(const halmaz &x, const halmaz &y);
friend halmaz operator-(const halmaz &x, const halmaz &y);
friend halmaz operator*(const halmaz &x, const halmaz &y);
friend halmaz metszet(const halmaz x, const halmaz y);
friend halmaz szim_dif(const halmaz x, const halmaz y);
};

#endif

```

### **Forráskód – megvalósítás (halmaz.cpp):**

```

#include "halmaz.h"

/* UNIOKEPZES
 *
 * Hasznalat:
 * z=x+y
 *
 * Informaciok:
 * x,y,z - halmaz
 * Az operator kiszámítja két halmaz unioját. Az eredményt
 * a "z" halmazban kapjuk meg, melyben az elemeket tároló
 * dinamikus helyfoglalású tömb elemszáma megegyezik a
 * halmaz elemeinek számával.
 */
halmaz operator+(const halmaz &x, const halmaz &y){
    halmaz z;
    int max_elemszam=x.elem_szam+y.elem_szam;
    if(max_elemszam>1){
        double* tmp;
        int index1=0;
        tmp = new double[max_elemszam];
        int i=0;
        int j=0;
        while((i<x.elem_szam)&&(j<y.elem_szam)){
            if(x.h[i]<y.h[j]){
                tmp[index1]=x.h[i];
                ++i;
            }else{
                tmp[index1]=y.h[j];
                if(x.h[i]==y.h[j]){
                    ++i;
                }
                ++j;
            }
            ++index1;
        }
        //Maradék elemek berakása
        while(i<x.elem_szam){
            tmp[index1]=x.h[i];
            ++index1;
            ++i;
        }
        while(j<y.elem_szam){
            tmp[index1]=y.h[j];
            ++index1;
            ++j;
        }
        //Unio elkészítése
        z.elem_szam=index1;
        z.h = new double[index1];
        for(int k=0; k<index1; ++k){

```

```

        z.h[k]=tmp[k];
    }
    delete[] tmp;
}
return z;
}
/* KULONBSEGHALMAZ
*
* Hasznalat:
* z=x-y
*
* Informaciok:
* x,y,z - halmaz
* Az operator kiszamitja ket halmaz kulonbseget. Az eredmenyt
* a "z" halmazban kapjuk meg, melyben az elemeket tarolo
* dinamikus helyfoglalasu tomb elemszama megegyezik a
* halmaz elemeinek szamaval.
*/
halmaz operator-(const halmaz &x, const halmaz &y){
    halmaz z;
    if(x.elem_szam>=1){
        double* tmp;
        int index1=0;
        tmp = new double[x.elem_szam];
        int i=0;
        int j=0;
        while((i<x.elem_szam)&&(j<y.elem_szam)){
            if(x.h[i]<y.h[j]){
                tmp[index1]=x.h[i];
                ++index1;
                ++i;
            }else{
                if(x.h[i]==y.h[j]){
                    ++i;
                }
                ++j;
            }
        }
        //Maradek elemek berakasa
        while(i<x.elem_szam){
            tmp[index1]=x.h[i];
            ++index1;
            ++i;
        }
        //Kulonbseg elkeszítése
        if(index1>=1){
            z.elem_szam=index1;
            z.h = new double[index1];
            for(int k=0; k<index1; ++k){
                z.h[k]=tmp[k];
            }
        }
        delete[] tmp;
    }
    return z;
}
/* METSZETKEPZES
*
* Hasznalat:
* z=x*y
*
* Informaciok:
* x,y,z - halmaz
* Az operator visszaadja x és y halmazok metszetet. Az
* eredmenyt a "z" halmazban kapjuk meg. Ez a gyorsabb
* es memoriakimelobb modja a metszet kiszamitasanak,
* es joval kisebb a muvetetigenye, mint fuggveny-
* valtozatanak.
*
* Figyelem:
* Az eredmenyul kapott halmaz elemeit tarolo dinamikus
* helyfoglalasu tomb folosleges es elerhetetlen

```

```

* elemeket is tartalmaz, mert a muveletigeny
* minimalizalasa miatt ezek az elemek mar nem
* lesznek kiszurve.
*/
halmaz operator*(const halmaz &x, const halmaz &y){
    halmaz z;
    int max_elemszam;
    if(x.elem_szam<y.elem_szam){
        max_elemszam=x.elem_szam;
    }else{
        max_elemszam=y.elem_szam;
    }
    if(max_elemszam>=1){
        z.h = new double[max_elemszam];
        int i=0;
        int j=0;
        while((i<x.elem_szam)&&(j<y.elem_szam)){
            if(x.h[i]<y.h[j]){
                ++i;
            }else{
                if(x.h[i]==y.h[j]){
                    z.h[z.elem_szam]=x.h[i];
                    ++z.elem_szam;
                    ++i;
                }
                ++j;
            }
        }
        return z;
    }
}

/* METSZETKEPZES
*
* Hasznalat:
* z=metszet(x,y)
*
* Informaciok:
* x,y,z - halmaz
* A fuggveny visszaadja x és y halmazok metszetet. Az
* eredmenyt a "z" halmazban kapjuk meg, melyben az elemeket
* tarolo dinamikus helyfoglalasu tomb elemszama megegyezik
* a halmaz elemeinek szamaval.
* Ez a lassabb es memoriaigenyesebb, viszont pontosabb
* modja a metszet kiszamitasanak.
*/
halmaz metszet(const halmaz x, const halmaz y){
    halmaz z;
    int max_elemszam;
    if(x.elem_szam<y.elem_szam){
        max_elemszam=x.elem_szam;
    }else{
        max_elemszam=y.elem_szam;
    }
    if(max_elemszam>=1){
        double* tmp;
        int index1=0;
        tmp = new double[max_elemszam];
        int i=0;
        int j=0;
        while((i<x.elem_szam)&&(j<y.elem_szam)){
            if(x.h[i]<y.h[j]){
                ++i;
            }else{
                if(x.h[i]==y.h[j]){
                    tmp[index1]=x.h[i];
                    ++index1;
                    ++i;
                }
                ++j;
            }
        }
    }
}

```

```

        //Metszet elkeszítése
        if(index1>=1){
            z.elem_szam=index1;
            z.h = new double[index1];
            for(int k=0; k<index1; ++k){
                z.h[k]=tmp[k];
            }
        }
        delete[] tmp;
    }
    return z;
}
/* SZIMMETRIKUS DIFFERENCIA
 *
 * Hasznalat:
 * z=szim_dif(x,y);
 *
 * Információk:
 * x,y,z - halmaz
 * A függvény visszaadja x és y halmazok szimmetrikus
 * differenciáját. Az eredményt a "z" halmazban kapjuk meg,
 * melyben az elemeket tároló dinamikus helyfoglalású tömb
 * elemszáma megegyezik a halmaz elemeinek számával.
 * Elvileg ez a függvény teljesen felesleges, mert
 * következik az uniózás és különbségképzés műveleteiből
 * de a teljesesség kedvéért definiálva lett.
 * Hogy az algoritmus egyszerűbb legyen, a metszetképzés
 * művelete is föl lett használva.
 */
halmaz szim_dif(const halmaz x, const halmaz y){
    halmaz a=x+y;
    halmaz b=x*y;
    halmaz z=a-b;
    return z;
}

```

### A halmaz kiszámításának műveletigénye:

A műveletigény a legrosszabb esetre lett kiszámítva.

- Változó-deklarációk száma: 5 db;
- Értékdadások száma:  $4n+1$  db;
- Feltétel-vizsgálatok száma:  $5n+5$  db;

A teljes műveletigény ezek összegeként adódik:  $9n+11$ , ahol  $n=|x|$ , ha  $|x|<|y|$ , illetve  $n=|y|$  egyébként.

### Forráskód – főprogram (hteszt.cpp):

```

/*
 * ELSŐ SZÁMÚ BEADANDO
 * (Javított változat)
 * Készítette: Ábrahám Róbert
 * EHA-kód: ABROAAI.ELTE
 * Datum: 2008/10/14.
 */

#include <iostream>
#include <string>
#include "halmaz.h"
#include "halmaz.cpp"

using namespace std;

```



```
//Legyen meg egyszerubb a kiiras!
void show_result(halmaz x, const string text);

int main()
{
    halmaz* t;
    halmaz a,b,c,d;
    int n,m;
    double k,l;
    cout << "MUVELETEK HALMAZOKKAL" << endl;
    cout << endl;
    cout << "Halmazok szama: "; cin >> n;
    if(n<2)
    {
        cout << "Minimum 2 halmazt kell megadni!" << endl;
        exit(1);
    }
    t = new halmaz[n];
    for(int i=0; i<n; ++i)
    {
        cout << endl;
        cout << i+1 << ". halmaz elemszama: "; cin >> m;
        if(m<0)
        {
            cout << "Nem adhat meg negativ erteket!" << endl;
            delete[] t;
            exit(2);
        }
        if(m>=1)
        {
            for(int j=0; j<m; ++j)
            {
                cout << j+1 << ". elem: "; cin >> k;
                t[i].insert(k);
            }
        }
        cout << i+1 << ". halmaz tartalma:" << endl;
        t[i].kiir(); cout << endl;
        //Unio es metszetszamitas
        a=a+t[i];
        if(i==0){
            b=t[i];
        }else{
            b=b*t[i];
        }
        //Kulonbseg es szimmetrikus differencia
        if(i>=1)
        {
            c=t[i-1]-t[i];
            cout << endl;
            cout << "Kulonbseghalmaz (";
            cout << i << "-" << i+1;
            cout << "):" << endl;
            c.kiir(); cout << endl;
            d=szim_dif(t[i-1],t[i]);
            show_result(d,"Szimmetrikus differencia:");
        }
    }
    //Eredmenyek
    show_result(a,"A halmazok unioja:");
    show_result(b,"A halmazok metszete:");
    //Elem eltavolitasa
    if(b.ures()==false)
    {
        cout << "Ezt vegyuk ki belole: "; cin >> l;
        try
        {
            b.dispose(l);
            show_result(b,"Az igy kapott halmaz tartalma:");
            cout << "A halmaz elemszama: " << b.elemszam();
            cout << endl;
        }
    }
}
```

```
        catch(halmaz::exception hiba)
        {
            if(hiba==halmaz::WRONGVAL)
            {
                cout << l << " nem eleme a halmaznak!" << endl;
            }
        }
        cout << endl;
        cout << "Tartalmazas ellenorzes: "; cin >> l;
        if(b.eleme(l)==true)
        {
            cout << l << " eleme a halmaznak." << endl;
        }
        else
        {
            cout << l << " nem eleme a halmaznak." << endl;
        }
        delete[] t;
        return 0;
    }

void show_result(halmaz x, const string text)
{
    cout << endl;
    cout << text << endl;
    x.kiir(); cout << endl;
}
```

**Tesztelési terv:**

- Halmazok mennyiségének megadása (negatív, nulla, illetve pozitív egész értékekkel);
- Halmazok elemszámának megadása (negatív, nulla, illetve pozitív egész értékekkel);
- Halmazok feltöltése (valós értékekkel);
- Eredmények áttekintése;
- Elem eltávolításának kipróbálása (valós értékekkel);
- Tartalmazás ellenőrzése (valós értékekkel);

**Fejlesztési lehetőségek:**

- Az uniózás, különbségképzés és szimmetrikus differencia operátorokra, illetve függvényre lehetne gyorsabb és kevesebb memóriát igénylő megoldásokat adni;
- A halmaz elemeinek reprezentálására szolgáló, valós típusú, dinamikus helyfoglalású tömböt ki lehetne terjeszteni további (pl. bool, integer, character vagy string) típusokra is;
- A halmaz kiterjesztett típusaira típusvezérlő metódusokat lehetne definiálni (lehessen a halmaz típusát egyetlen rövid utasítással megadni, és ez vonatkozzon az összes – már definiált – metódusra is);
- Kihasználva a C++ adta lehetőségeket, a kiterjesztett halmaz típusait kombinálni is lehetne (egy halmaz-típuson belül többféle típus legyen használatban egyszerre).