

Műveletek Halmazokkal II.

Feladat:

Készítsen egy halmaz típust! Alkalmazzon osztályt! A halmazt rendezetlen láncolt listával ábrázolja! Implementálja a szokásos műveleteket, egészítse ki az osztályt a kényelmes és biztonságos használat érdekében megfelelő metódusokkal (beolvasó operátor \gg , kiíró operátor \ll), alkalmazzon kivételkezelést, készítsen teszt környezetet, és bontsa modulokra a programját! A teszt környezet hívjon meg egy olyan barát-operátort is, amely kiszámítja két halmaz metszetét! A metszetképzés műveletigénye: $O(m \cdot n)$, ahol m és n a két halmaz elemszáma.

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:
    //FELEPITES
    int elem_szam;
    struct Node{
        double value;
        Node *next;
        Node(int i=0, Node *p=NULL) : value(i), next(p){}
    };
    Node *first;
public:
    //ALAPERTEKEK
    halmaz(){
        elem_szam=0;
        first = new Node();
        first->next=NULL;
    }
    //KIVETELEK
    enum exception{ EMPTYSET, WRONGVAL, WRONGITEM };
    //FUGGVENYEK es ELJARASOK
    bool eleme(const double e);
    void insert(const double e);
    void dispose(const double e);
    double elem(const int x);
    int elemszam();
    bool ures();
    void kiurit();
    void kiir();
    //OPERATOROK
    void operator<<(const double e);
    double operator>>(const int n);
    halmaz operator=(const halmaz x);
    halmaz operator*(const halmaz &x);
    halmaz operator+(const halmaz &x);
    halmaz operator-(const halmaz &x);
    //BARAT-FUGGVENYEK
    friend halmaz szim_dif(halmaz x, halmaz y);
};

#endif
```

Forráskód – megvalósítás (halmaz.cpp):

```
#include "halmaz.h"

//ERTEKADAS-OPERATOR
halmaz::operator=(const halmaz x){
    Node *p=x.first->next;
    Node *q=first;
    Node *r;
    elem_szam=0;
    while(p!=NULL){
        r = new Node();
        r->next=NULL;
        r->value=p->value;
        q->next=r;
        q=q->next;
        ++elem_szam;
        p=p->next;
    }
    return *this;
}

/*
 * Tartalmazas eldontese
 *
 * Hasznalat:
 * l=a.eleme(x);
 *
 * Informaciok:
 * l - bool;
 * a - halmaz;
 * x - double;
 * A fuggveny eldonti adott "x" ertekrol, hogy eleme-e az
 * "a" halmaznak.
 */
bool halmaz::eleme(const double e){
    bool found=false;
    Node *p=first->next;
    while((p!=NULL)&&(found==false)){
        if(p->value==e){ found=true; }
        p=p->next;
    }
    return found;
}

/*
 * Elem beszurasa
 *
 * Hasznalat:
 * a.insert(x);
 *
 * Informaciok:
 * a - halmaz;
 * x - double;
 * Az eljaras beszurja az "x" elemet az "a" halmazba, amennyiben
 * a halmaz meg nem tartalmazza ezt az elemet.
 * Eredetileg a tartalmazas fuggvenyre epult ez az eljaras,
 * azonban a hatekonysag novelese erdekeben most mar az
 * ellenorzest is ez az eljaras vegzi el a mar definialt
 * "eleme(x)" fuggveny helyett.
 */
void halmaz::insert(const double e){
    bool found=false;
    Node *p=first;
    while((p->next!=NULL)&&(found==false)){
        p=p->next;
        if(p->value==e){
            found=true;
        }
    }
    if(found==false){
```

```

        Node *q;
        q = new Node();
        q->next=NULL;
        q->value=e;
        p->next=q;
        ++elem_szam;
    }
}
/*
 * Elem eltávolítása
 *
 * Használat:
 * a.dispose(x);
 *
 * Információk:
 * a - halmaz;
 * x - double;
 * Az eljárás eltávolítja az "x" elemet az "a" halmazból,
 * feltéve, hogy van ilyen elem. Ha nincs, akkor a halmaz
 * tartalma változatlan marad.
 */
void halmaz::dispose(const double e){
    if(elem_szam>=1){
        bool found=false;
        Node *p=first;
        Node *q=first;
        while((p->next!=NULL)&&(found==false)){
            p=p->next;
            if(p->value==e){
                q->next=p->next;
                delete p;
                --elem_szam;
                found=true;
            }
            q=q->next;
        }
        if(found==false){
            throw WRONGVAL;
        }
    }else{
        throw EMPTYSET;
    }
}
/*
 * Egy elem visszaadása
 *
 * Használat:
 * z=a.elem(x);
 *
 * Információk:
 * a - halmaz;
 * x - integer;
 * z - double;
 * A függvény visszaadja az "a" halmaz x. elemét, feltéve
 * hogy a megadott sorszámu elem létezik.
 */
double halmaz::elem(const int x){
    double z=0;
    bool siker=false;
    if(elem_szam>=1){
        if((x>=0)&&(x<elem_szam)){
            int i=0;
            Node *p=first;
            while((p->next!=NULL)&&(siker==false)){
                p=p->next;
                if(i==x){
                    z=p->value;
                    siker=true;
                }
                ++i;
            }
        }
    }
}

```

```
        }
        if(siker==false){
            throw WRONGITEM;
        }else{
            return z;
        }
    }
}
/*
 * Kiiratas
 * (Standard Output)
 *
 * Hasznalat:
 * a.kiir();
 *
 * Informaciok:
 * a - halmaz;
 * Az eljárás gyönyörűen olvasható formában kiírja
 * az "a" halmaz elemeit a Standard Outputra.
 */
void halmaz::kiir(){
    if(elem_szam>=1){
        std::cout << "{";
        Node *p=first;
        while(p->next!=NULL){
            p=p->next;
            std::cout << p->value;
            if(p->next!=NULL){
                std::cout << ", ";
            }
        }
        std::cout << "}" << std::endl;
    }else{
        std::cout << "Ureshalmaz" << std::endl;
    }
}
/*
 * Elemszam visszaadása
 *
 * Hasznalat:
 * x=a.elemszam();
 *
 * Informaciok:
 * a - halmaz;
 * x - integer;
 * A függvény visszaadja az "a" halmaz elemszamat.
 */
int halmaz::elemszam(){
    return elem_szam;
}
/*
 * Uresseg eldöntése
 *
 * Hasznalat:
 * l=a.ures();
 *
 * Informaciok:
 * a - halmaz;
 * l - bool;
 * A függvény megadja, hogy az "a" halmaz ures-e.
 */
bool halmaz::ures(){
    if(elem_szam>=1){
        return false;
    }else{
        return true;
    }
}
/*
 * Halmaz kiurítése
 *
 * Hasznalat:
 * a.kiurit();
 */
```

```

*
* Informaciok:
* a - halmaz;
* Az eljárás kiurítja az "a" halmazt.
*/
void halmaz::kiurit(){
    Node *p=first->next;
    first->next=NULL;
    Node *q=NULL;
    while(p!=NULL){
        q=p;
        p=p->next;
        delete q;
    }
    elem_szam=0;
}
/*
* Elem beszúrása
*
* Használat:
* a<<x;
*
* Informaciok:
* a - halmaz;
* x - double;
* Az eljárás beszúrja az "x" elemet az "a" halmazba, amennyiben
* a halmaz még nem tartalmazza ezt az elemet.
* Ez az eljárás ugyanazt csinálja, mint az a.insert(x) eljárás,
* csak itt operator-felüldefiniálással lett megvalósítva
* a művelet - a feladatkiírásnak megfelelően.
*/
void halmaz::operator<<(const double e){
    bool found=false;
    Node *p=first;
    while((p->next!=NULL)&&(found==false)){
        p=p->next;
        if(p->value==e){
            found=true;
        }
    }
    if(found==false){
        Node *q;
        q = new Node();
        q->next=NULL;
        q->value=e;
        p->next=q;
        ++elem_szam;
    }
}
/*
* Egy elem visszaadása
*
* Használat:
* z=a>>n;
*
* Informaciok:
* a - halmaz;
* n - integer;
* z - double;
* A függvény visszaadja az "a" halmaz n. elemét, feltéve
* hogy a megadott sorszámu elem létezik.
* A függvény ugyanazt csinálja, mint a z=a.elem(n) függvény
* de a feladatkiírásnak megfelelően operatorral is
* meg lett valósítva.
*/
double halmaz::operator>>(const int n){
    double z=0;
    bool siker=false;
    if(elem_szam>=1){
        if((n>=0)&&(n<elem_szam)){
            int i=0;
            Node *p=first;

```

```

        while( (p->next!=NULL)&&(siker==false)){
            p=p->next;
            if(i==n){
                z=p->value;
                siker=true;
            }
            ++i;
        }
    }
}
if(siker==false){
    throw WRONGITEM;
}else{
    return z;
}
}
/* METSZETKEPZES
 *
 * Hasznalat:
 * z=x*y
 *
 * Informaciok:
 * x,y,z - halmaz
 * Az operator visszaadja x és y halmazok metszetet. Az
 * eredményt a "z" halmazban kapjuk meg.
 */
halmaz halmaz::operator*(const halmaz &x){
    halmaz z;
    Node *p=first->next;
    Node *q;
    Node *r;
    Node *s=z.first;
    bool found;
    while(p!=NULL){
        found=false;
        q=x.first->next;
        while((q!=NULL)&&(found==false)){
            if(q->value==p->value){
                found=true;
            }
            q=q->next;
        }
        if(found==true){
            r = new Node();
            r->next=NULL;
            r->value=p->value;
            s->next=r;
            s=s->next;
            ++z.elem_szam;
        }
        p=p->next;
    }
    return z;
}
/* UNIOKEPZES
 *
 * Hasznalat:
 * z=x+y
 *
 * Informaciok:
 * x,y,z - halmaz
 * Az operator kiszámítja két halmaz unioját. Az eredményt
 * a "z" halmazban kapjuk meg.
 */
halmaz halmaz::operator+(const halmaz &x){
    halmaz z;
    double e;
    Node *p=first->next;
    Node *q;
    Node *r=z.first;
    Node *s;
    bool found;

```

```

        while(p!=NULL){
            q = new Node();
            q->next=NULL;
            q->value=p->value;
            r->next=q;
            r=r->next;
            ++z.elem_szam;
            p=p->next;
        }
        p=x.first->next;
        while(p!=NULL){
            s=z.first->next;
            found=false;
            while((s!=NULL)&&(found==false)){
                if(s->value==p->value){
                    found=true;
                }
                s=s->next;
            }
            if(found==false){
                q = new Node();
                q->next=NULL;
                q->value=p->value;
                r->next=q;
                ++z.elem_szam;
                r=r->next;
            }
            p=p->next;
        }
        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.
 */
halmaz::operator-(const halmaz &x){
    halmaz z;
    Node *p=first->next;
    Node *q;
    Node *r;
    Node *s=z.first;
    bool found;
    while(p!=NULL){
        q=x.first->next;
        bool found=false;
        while((q!=NULL)&&(found==false)){
            if(p->value==q->value){
                found=true;
            }
            q=q->next;
        }
        if(found==false){
            r = new Node();
            r->next=NULL;
            r->value=p->value;
            s->next=r;
            s=s->next;
            ++z.elem_szam;
        }
        p=p->next;
    }
    return z;
}
}
/* SZIMMETRIKUS DIFFERENCIA
 *
 * Hasznalat:

```



```

* z=szim_dif(x,y);
*
* Informaciok:
* x,y,z - halmaz
* A függvény visszaadja x és y halmazok szimmetrikus
* differenciáját. Az eredményt a "z" halmazban kapjuk meg.
*/
halmaz szim_dif(halmaz x, halmaz y){
    halmaz a=x+y;
    halmaz b=x*y;
    halmaz z=a-b;
    return z;
}

```

Forráskód – főprogram (hteszt.cpp):

```

/*
* MASODIK SZAMU BEADANDO
* Készítette: Abraham Robert
* EHA-kod: ABROAAI.ELTE
* Datum: 2008/11/6.
*/

#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]<<k;
            }
        }
        cout << i+1 << ". halmaz tartalma:" << endl;
        t[i].kiir(); cout << endl;
        //Unio es metszetszamitas
        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 eltávolítása
    if(b.ures()==false)
    {
        cout << "Ezt vegyuk ki belole: "; cin >> l;
        try
        {
            b.dispose(l);
            show_result(b,"Az ily 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);