

ELEMI ALKALMAZÁSOK FEJLESZTÉSE II. Sorozat és bejárói

Készítette: Gregorics Tibor
Steingart Ferenc

Tartalom

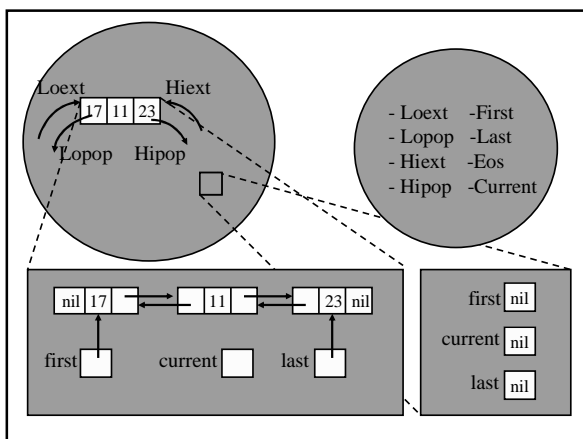
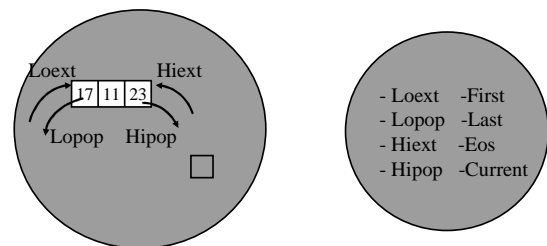


- Sorozat-típus (tároló) kétirányú láncolt listával
- Bejáró (iterátor) típus

1. Feladat

Olvassuk be a standard inputról érkező számokat, majd írjuk ki a standard outputra előbb a negatívakat, utána pedig a többit!

A feladat megoldásához készítsünk egy egész számokat tartalmazó sorozat típust. A sorozatot egy kétirányú, fejelem nélküli láncolt listával reprezentáljuk.



```
sequence.h : public
#define nil 0
class Sequence{
public:
    enum Exceptions{EMPTYSEQ};

    Sequence():
        first(nil),last(nil),current(nil){};
    ~Sequence();

    void Loext(int e);
    int Lopop();
    void Hiext(int e);
    int Hipop();

    void First() {current = first;}
    void Next() {current = current->next;}
    bool Eos() const {return current==nil;}
    int Current()const {return current->cont;}
```

sequence.h : private

```
private:
Sequence(const Sequence&);
Sequence& operator=(const Sequence&);

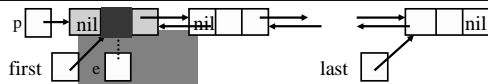
struct Node{
    int val;
    Node *next;
    Node *prev;
    Node(int e, Node *n, Node *p):
        val(e), next(n), prev(p){};
};
Node *first;
Node *last;
Node *current;
};
```

sequence.h : destruktör

```
Sequence::~Sequence()
{
    Node *p *q;
    q = first;
    while( q!=nil){
        p = q;
        q = q->next;
        delete p;
    }
}
```

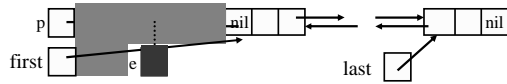
sequence.cpp : Loext

```
void Sequence::Loext(int e)
{
    Node* p = new Node(e,first,nil);
    if(first!=nil){
        first->prev = p;
    }
    first = p;
    if(last==nil){
        last = p;
    }
}
```



sequence.cpp : Lopop

```
int Sequence::Lopop()
{
    if(first==nil) throw EMPTYSEQ;
    int e = first->val;
    Node* p = first;
    first = first->next;
    delete p;
    if(first!=nil) first->prev = nil;
    else last = nil;
    return e;
}
```



sequence.cpp : Hiext

```
void Sequence::Hiext(int e)
{
    Node* p = new Node(e,nil,last);
    if(last!=nil){
        last->next = p;
    }
    last = p;
    if(first==nil){
        first = p;
    }
}
```

sequence.cpp : Hipop

```
int Sequence::Hipop()
{
    if(last==nil) throw EMPTYSEQ;
    int e = last->val;
    Node* p = last;
    last = last->prev;
    delete p;
    if(last!=nil)last->next = nil;
    else first = nil;
    return e;
}
```

fo.cpp

```
#include <iostream>
#include "sequence.h"
int main()
{
    Sequence x;
    int i;
    while(cin>>i){
        if (i>0) x.Hiext(i);
        else    x.Loext(i);
    }
    for(x.First(); !x.Eos(); x.Next()){
        cout<<x.Current()<<endl;
    }
    return 0;
}
```

Felsorolás típusú kivételek

```
Sequence x;

...

try{
    cout << x.Lopop() << endl;
}catch(Sequence::Exceptions e){
    if(e==Sequence::EMPTYSEQ){
        cout<< "törlés üres sorozatban" <<endl;
    }
}
```

2. Feladat

Olvassuk be a standard inputról érkező számokat, majd írjuk ki őket az érkezésük sorrendjében úgy, hogy megadjuk minden szám minden előfordulásánál a szám összes előfordulásának számát!

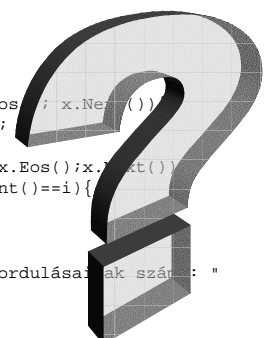
A feladat megoldásához egyidejűleg két bejáró kell: az egyikkel végig megyünk az elemeken, a másikkal minden elemre megszámloljuk annak előfordulásait

fo.cpp

```
main()
{
    Sequence x;
    ... // Beolvasás

    for(x.First(); !x.Eos(); x.Next()){
        i = x.Current();
        int s = 0;
        for(x.First(); !x.Eos(); x.Next()){
            if (x.Current()==i){
                s++;
            }
        }
        cout << i << " előfordulásai száma: "
            << s << endl;
    }

    return 0;
}
```



sequence.h: public

```
class Sequence{
public:
    enum Exceptions{EMPTYSEQ};

    Sequence():
        first(nil),last(nil),current(nil){};
    ~Sequence();

    void Loext(int e);
    int  Lopop();
    void Hiext(int e);
    int  Hipop();

    void First() {current = first;}
    void Next()  {current = current->next;}
    bool Eos()   const {return current==nil;}
    int  Current()const {return current->cont;}
```

sequence.h: private

```
private:
    Sequence(const Sequence&);
    Sequence& operator=(const Sequence&);

    struct Node{
        int val;
        Node *next;
        Node *prev;
        Node(int e, Node *n, Node *p):
            val(e), next(n), prev(p){};
    };
    Node *first;
    Node *last;
    Node *current;
```

sequence.h : public

```
public:
    friend class Iterator;
    class Iterator{
    friend class Sequence;
    public:
        Iterator(Sequence& s):
            seq(&s),current(nil){};
        void First() {current = seq->first;}
        void Next() {current = current->next;}
        bool Eos() const {return current==nil;}
        int Current()const {return current->cont;}

    private:
        Sequence *seq;
        Node* current;
    };
};
```

fo.cpp

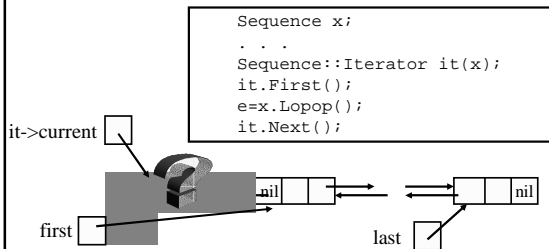
```
int main()
{
    Sequence x;

    ... // Beolvasás

    Sequence::Iterator it1(x);
    Sequence::Iterator it2(x);
    for(it1.First(); !it1.Eos(); it1.Next()){
        i = it1.Current();
        int s = 0;
        for(it2.First();!it2.Eos();it2.Next()){
            if (it2.Current()==i) s++;
        }
        cout << i << " előfordulásainak száma: "
             << s << endl;
    }
    return 0;
}
```

Elem törlése bejárás közben

- Ha kitöröljük azt a listaelemet, amelyre egy bejáró hivatkozik, akkor a bejárás elromlik.



Megoldási módok

- Teljes kizárás: A törlő művelet kivételt dob bejárás esetén.
- Elemszintű kizárás: A törlő művelet kivételt dob ha olyan elemre vonatkozik, amelyre bejáró hivatkozik.
- Törlés késleltetés: A törlendő elem csak akkor törlődik, ha már nem hivatkozik rá bejáró.

sequence.h : public

```
class Sequence{
public:
    enum Exceptions{EMPTYSEQ, UNDERTRAVERSAL};
    Sequence():
        first(nil),last(nil),current(nil),
        iteratorCount(0){};
    ~Sequence();

    void Loext(int e);
    int Lopop();
    void Hiext(int e);
    int Hipop();
};
```

sequence.h : private

```
private:
    Sequence(const Sequence&);
    Sequence& operator=(const Sequence&);
    struct Node{
        int val;
        Node *next;
        Node *prev;
        Node(int e, Node *n, Node *p):
            val(e), next(n), prev(p){};
    };
    Node *first;
    Node *last;
    int iteratorCount;
};
```

```
sequence.h: public
friend class Iterator;
class Iterator{
    friend class Sequence;
public:
    Iterator(Sequence& s):
        seq(&s), current(nil){}
        {seq->iteratorCount++;}
    ~Iterator() {seq->iteratorCount--;}

    void First() {current = seq->first;}
    void Next() {current = current->next;}
    bool Eos() const{return current==nil;}
    int Current()const{return current->cont;}

private:
    Sequence *seq;
    Node* current;
};
};
```

```
sequence.cpp : Lopot
int Sequence::Lopot()
{
    if(iteratorCount!=0) throw UNDERTRAVERSAL;
    if(first==nil) throw EMPTYSEQ;
    int e = first->val;
    Node* p = first;
    first = first->next;
    delete p;
    if(first!=nil) first->prev = nil;
    else last = nil;
    return e;
}
```

```
sequence.cpp : Hipop
int Sequence::Hipop()
{
    if(iteratorCount!=0) throw UNDERTRAVERSAL;
    if(last==nil)throw EMPTYSEQ;
    int e = last->val;
    Node* p = last;
    last = last->prev;
    delete p;
    if(last!=nil) last->next = nil;
    else first = nil;
    return e;
}
```

Ami még kimaradt

```
sequence.h: copy konstruktor
Sequence::Sequence(const Sequence& s)
{
    current = nil;
    if (s.first==nil){
        first = nil;
        last = nil;
    }
    else {
        Node* p = s.first;
        Node* q = new Node(p->val,nil,nil);
        first = q;
        while (p->next!=nil){
            p->next;
            q->next = new Node(p->val,nil,q);
            q = q->next;
        }
        last = q;
    }
}
```

```
sequence.h : értékadás
Sequence& Sequence::operator=(const Sequence& s)
{
    if (this==&s) return *this;

    // destruktorkor
    // copy konstruktor

    return *this;
}
```

