ELEMI ALKALMAZÁSOK FEJLESZTÉSE II.

Sorozat és bejárói

Készítette: Gregorics Tibor
Steingart Ferenc

Tartalom

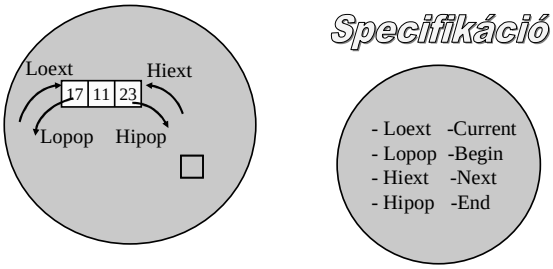
- Sorozat-típus (tároló) kétirányú láncolt listával
- Bejáró (iterátor) típus

1. Feladat

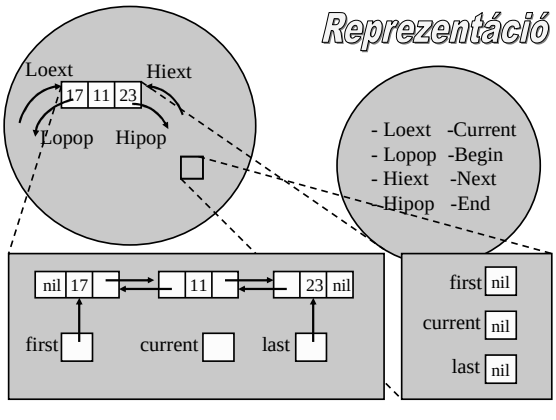
Olvassuk be a standard inputról érkező számokat, majd írjuk ki a standard outputra előbb a negatívokat, utána pedig a többit!

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

Specifikáció



Reprezentáció



Sorozat-típus publikus része

```
#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();

    int  Current() const {return current->cont;}
    void Begin()   {current = first;}
    bool End()     const {return current==nil;}
    Sequence operator++(int)
         {current = current->next;return *this;}
    Sequence& operator++()
         {current = current->next;return *this;}
```

sequence.h

Sorozat-típus privát része

```
  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;
};
```

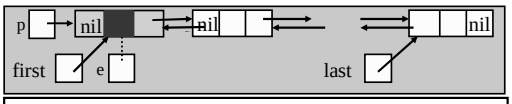
sequence.h

Destruktor

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

sequence.h

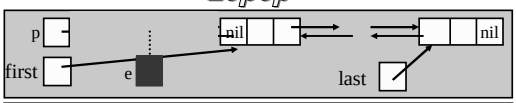
Loext




```
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;
}
```

sequence.cpp

Főprogram

```
#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.Begin(); !x.End(); x++){
       cout<<x.Current()<<endl;
    }
  return 0;
}
```

fo.cpp

Felsorolás típusú kivételek

```
Sequence x;

...

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

fo.cpp

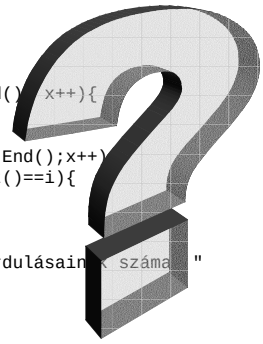
2. Feladat

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 megszámoljuk annak előfordulásait

Főprogram

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

    for(x.Begin(); !x.End(); x++){
        i = x.Current();
        int s = 0;
        for(x.Begin();!x.End();x++)
            if (x.Current()==i){
                s++;
            }
        }
    cout << i << " előfordulásainak száma: "
         << s << endl;
    }

  return 0;
```



fo.cpp

Sorozat osztály

```
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();

    int  Current() const {return current->cont;}
    void Begin()   {current = first;}
    bool End()     const {return current==nil;}
    Sequence operator++(int)
          {current = current->next;return *this;}
    Sequence& operator++()
          {current = current->next;return *this;}
```

sequence.h

Sorozat osztály

```
  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

Sorozat osztály

```
  public:
    friend class Iterator;
    class Iterator{
        friend class Sequence;
      public:
        Iterator(Sequence& s):
                    seq(&s),current(nil){};
        int  Current()const {return current->cont;}
        void Begin()  {current = seq->first;}
        bool End()    const {return current==nil;}
        Iterator operator++(int)
           {current = current->next;return *this;}
        Iterator& operator++()
           {current = current->next;return *this;}
      private:
        Sequence *seq;
        Node* current;
    };
```

sequence.h

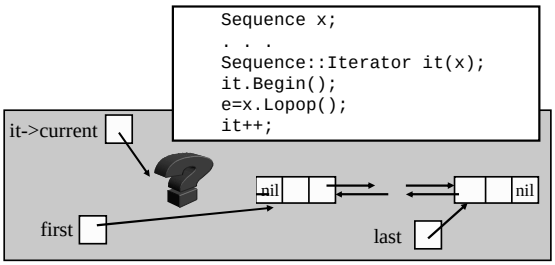
Főprogram

```
int main()
{
    Sequence x;

    ... // Beolvasás

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

fo.cpp

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.

Sorozat osztály

```
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

Sorozat osztály

```
  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

Sorozat osztály

```
public:
  friend class Iterator;
  class Iterator{
      friend class Sequence;
    public:
      Iterator(Sequence& s):
                  seq(&s),current(nil){}
                  {seq->iteratorCount++;}
      ~Iterator() {seq->iteratorCount--;}
      int  Current()const {return current->cont;}
      void Begin()  {current = seq->first;}
      bool End()    const {return current==nil;}
      Iterator operator++(int)
        {current = current->next;return *this;}
      Iterator& operator++()
        {current = current->next;return *this;}
    private:
      Sequence *seq;
      Node* current;
```

sequence.h

Sorozat osztály

```
int  Sequence::Lopop()
{
    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

Sorozat osztály

```
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;
}
```

sequence.cpp

Ami eddig kimaradt

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.cpp

Értékadás

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

    // destruktor

    // copy konstruktor

    return *this;
}
```

sequence.cpp