# Objektum elvű alkalmazások fejlesztése

## *Kétirányú sor felsorolóval*

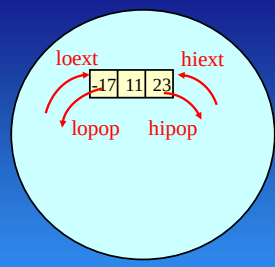
Készítette: Gregorics Tibor

# 1. Feladat

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

A feladat megoldásához készítsünk egy egész számokat tartalmazó sorozat szerkezetű típust, amelynek csak a két végéről lehet elvenni elemet, és csak a két végéhez lehet új elemet fűzni.
Az elemek sorozatát egy kétirányú, fejelem nélküli láncolt listával reprezentáljuk.

# Típus specifikáció




# Főprogram

```
#include <iostream>
#include "BiQueue.h"
using namespace std;
int main()
{
   BiQueue x;
   int e;
   while(cin >> e){
      if (e<0) x.loext(e);
      else     x.hiext(e);
   }

   for(x.first(); !x.end(); x.next()){
      cout << x.current() << endl;
   }
   return 0;
}
```

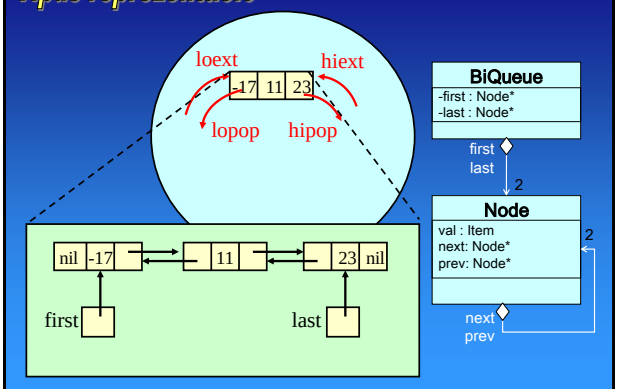
main.cpp

# Kétirányú sor típusának publikus része

```
class BiQueue{
   ...

 public:
      enum Exceptions{EMPTYQUEUE, FULLMEM};

      BiQueue();
      ~BiQueue();

      void loext(int e)  ;
      int  lopop()       ;
      void hiext(int e)  ;
      int  hipop()       ;

      void first()       ;
      void next()        ;
      bool end()    const;
      int  current()const;
};
```

biqueue.h

# Típus reprezentáció



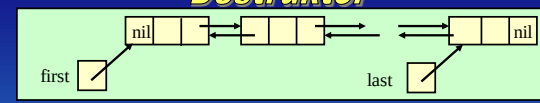
## Kétirányú sor típusának privát része

```
class BiQueue{
 private:

   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;

   BiQueue(const BiQueue&);
   BiQueue& operator=(const BiQueue&);
```

A fordító hibát jelez, ha használni akarjuk

biqueue.h

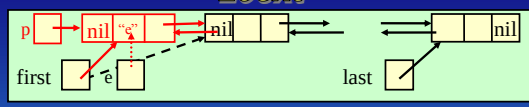
## Destruktor




```
BiQueue::~BiQueue()
{
   Node *p = _first;
   while(p!=NULL){
      Node *q = p->_next;
      delete p;
      p = q;
   }
}
```

biqueue.cpp

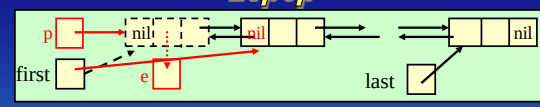
## Loext




```
void BiQueue::loext(int e)
{
   try{
      Node *p = new Node(e, _first, NULL);
      if(_first!=NULL) _first->_prev = p;
      _first = p;
      if(_last==NULL)  _last = p;
   }catch(std::bad_alloc o){
      throw FULLMEM;
   }
}
```

biqueue.cpp

## Lopop




```
int BiQueue::lopop()
{
   if(_first==NULL) throw EMPTYQUEUE;
   int  e = _first->_val;
   Node *p = _first;
   _first = _first->_next;
   delete p;
   if(_first!=NULL) _first->_prev = NULL;
   else             _last = NULL;
   return e;
}
```

biqueue.cpp

## Hiext

```
void BiQueue::hiext(int e)
{
   try{
      Node *p = new Node(e, NULL, _last);
      if(_last!=NULL) _last->_next = p;
      _last = p;
      if(_first==NULL)_first = p;
   }catch(std::bad_alloc o){
      throw FULLMEM;
   }
}
```

biqueue.cpp

## Hipop

```
int BiQueue::hipop()
{
   if(_last==NULL) throw EMPTYQUEUE;
   int  e = _last->_val;
   Node *p = _last;
   _last = _last->_prev;
   delete p;
   if(_last!=NULL)_last->_next = NULL;
   else          _first = NULL;
   return e;
}
```

biqueue.cpp

## Kétirányú sor bejáró műveletei

```
class BiQueue{
      ...
public:
   BiQueue(): _first(NULL), _last(NULL){}
      ...
// bejárást biztosító tagok
   void first()         {_current = _first;}
   void next()          {_current = _current->_next;}
   bool end()     const {return _current==NULL;}
   int  current()const {return _current->_val;}

 private:
   Node      *_current;

};
```

biqQueue.h

## 2. Feladat

Olvassuk be a standard bemenetről érkező számokat, majd írjuk ki őket az érkezésük sorrendjében a standard kimenetre úgy, hogy megadjuk minden számnál az összes előfordulásának számát!

A számokkal egy sorozat típusú objektumot töltsünk fel, majd ennek elemeit dolgozzuk fel sorban egymás után: minden elemre számoljuk meg, hogy hányszor fordul elő a sorozatban.

## Főprogram

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

   for(x.first(); !x.end(); x.next()){
      int e = x.current();

      int s = 0;
      for(x.first();!x.end();x.next()){
         if (x.current()==e) ++s;
      }
      cout << e << " előfordulásainak száma: "
           << s << endl;
   }
   return 0;
}
```

?

main.cpp

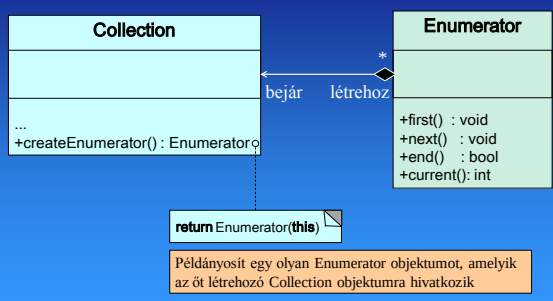
## Kétirányú sor bejáró műveletei

```
class BiQueue{
   ...

// bejárásokat biztosító tagok
   public:
      void first1()         {_current1 = _first;}
      void next1()          {_current1 = _current1->_next;}
      bool end1()     const {return _current1==NULL;}
      int  current1()const {return _current1->_val;}

      void first2()         {_current2 = _first;}
      void next2()          {_current2 = _current2->_next;}
      bool end2()     const {return _current2==NULL;}
      int  current2()const {return _current2->_val;}

   private:
      Node      *_current1;
      Node      *_current2;
};
```

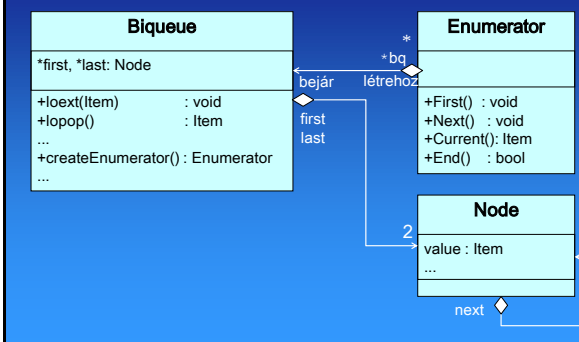
?

biqueue.h

## Gyűjtemény és felsorolói




## Kétirányú sor és felsorolása



## Kétirányú sort bejáró osztály

```
class BiQueue{
   ...

// bejárásokat biztosító osztály
 public:
   class Enumerator{
      public:
         Enumerator(BiQueue *s): _bq(s), _current(NULL){}

         void first()         {_current = _bq->_first;}
         void next()          {_current = _current->_next;}
         bool end()     const {return _current==NULL;}
         int  current()const {return _current->_val;}
      private:
         BiQueue  *_bq;
         Node     *_current;
   };
   Enumerator createEnumerator()
                {return Enumerator(this);}
};
```

biqueue.h

## Főprogram

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

   BiQueue::Enumerator it1 = x.createEnumerator();
   for(it1.first(); !it1.end(); it1.next()){
      int e = it1.current();

      BiQueue::Enumerator it2 = x.createEnumerator();
      int s = 0;
      for(it2.first();!it2.end();it2.next()){
         if (it2.current()==e) ++s;
      }
      cout << e << " előfordulásainak száma: "
           << s << endl;
   }
   return 0;
}
```

main.cpp

## Elem törlése bejárás közben

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

```
BiQueue x;
 . . .
BiQueue::Enumerator it = x.createEnumerator();
it.first();
e = x.lopop();
it.next();
```

## Megoldási módok

- Teljes kizárás: A törlő művelet kivételt dob bejárás esetén.
  - bejárók számának ismerete (elég csak az aktív bejárók száma)
- Elemszintű kizárás: A törlő művelet kivételt dob ha olyan elemet törölne, amelyre bejáró hivatkozik.
  - aktív bejárók ismerete
- Törlés késleltetés: Ha a törlő művelet olyan elemet törölne, amelyre bejáró hivatkozik akkor csak megjelöljük azt, és csak akkor töröljük, ha már nem hivatkozik rá bejáró.
  - aktív bejárók ismerete
  - törlendő elemek tárolása
  - a bejáró destruktora vagy First() illetve Next() metódusa üzen, hogy próbálkozzunk újra a törléssel.

## Kétirányú sor osztálya

```
public:
  enum Exceptions{EMPTYQUEUE, FULLMEM, UNDERTRAVERSAL};
  BiQueue():first(NULL),last(NULL),_enumeratorCount(0){};
  ~BiQueue();
  void loext(int e);
  int  lopop();
  void hiext(int e);
  int  hipop();
  Enumerator createEnumerator(){return Enumerator(this);}
private:
  struct Node{ ... };

  Node *_first;
  Node *_last;
  int   _enumeratorCount;
  ...
```

biqueue.h

## Lopop művelet

```
int  BiQueue::Lopop()
{
    if(_enumeratorCount!=0) throw UNDERTRAVERSAL;
    if(_first==NULL) throw EMPTYSEQ;
    int  e = _first->_val;
    Node *p = _first;
    _first = _first->_next;
    delete p;
    if(_first!=NULL) _first->_prev = NULL;
    else             _last = NULL;
    return e;
}
```

biqueue.cpp

## Hipop művelet

```
int  BiQueue::Hipop()
{
    if(_enumeratorCount!=0) throw UNDERTRAVERSAL;
    if(_last==NULL)throw EMPTYSEQ;
    int  e = _last->_val;
    Node *p = _last;
    _last = _last->_prev;
    delete p;
    if(_last!=NULL) _last->next = NULL;
    else            _first = NULL;
    return e;
}
```

biqueue.cpp

## Kétirányú sor osztálya

```
public:
   class Enumerator{
      public:
         Enumerator(BiQueue *s):_bq(s),_current(NULL)
                                {++(_bq->_enumeratorCount);}
         ~Enumerator()          {--(_bq->_enumeratorCount);}

         void first()           {_current = _bq->_first;}
         void next()            {_current = _current->_next;}
         bool end()     const {return _current==NULL;}
         int  current()const {return _current->_val;}

      private:
         BiQueue *_bq;
         Node    *_current;
};
```

biqueue.h

## Kivételkezelés

```
BiQueue x;
    . . .
BiQueue::Enumerator it = x.createEnumerator();
it.first();

try{
   e = x.lopop();
}catch(BiQueue::Exceptions e){
   if(e==BiQueue::UNDERTRAVERSAL)
      cout << " ... " << endl;
   }
}

it.next();
```

main.cpp