Objektum elvű alkalmazások fejlesztése

Kétirányú sor felsorolóval

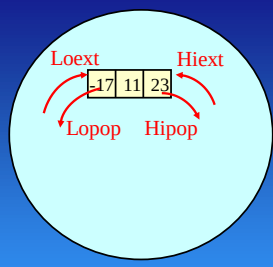
Készítette: Gregorics Tibor

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;
}
```

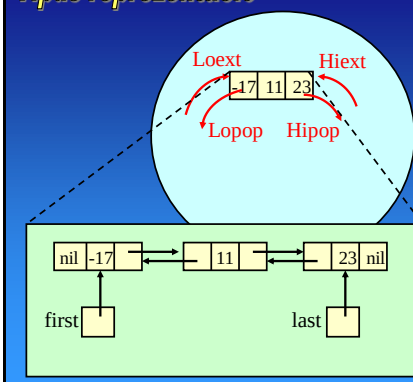
main.cpp

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();

};
```

biqueue.h

Típus reprezentáció



Kétirányú sor típusának privát része

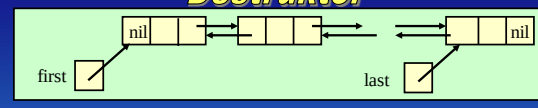
7

```
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&);
```

biqueue.h

Destruktor

8

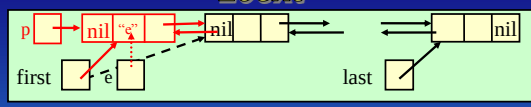



```
BiQueue::~BiQueue()
{
   Node *p = first;
   while(p!=NULL){
      Node *q = p->next;
      delete p;
      p = q;
   }
}
```

biqueue.cpp

Loext

9

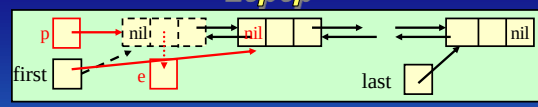



```
void BiQueue::Loext(int e)
{
   try{
      Node *p = new Node(e, first, NULL);
      if(first!=NULL) first->prev = p;
      first = p;
      if(last==NULL)  last = p;
   }catch(std::bad_alloc o){
      throw FULLMEM;
   }
}
```

biqueue.cpp

Lopop

10




```
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

11

```
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

12

```
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
   int  Current()const {return current->val;}
   void First()        {current = first;}
   bool End()const     {return current==NULL;}
   void Next()         {current = current->next;}

 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

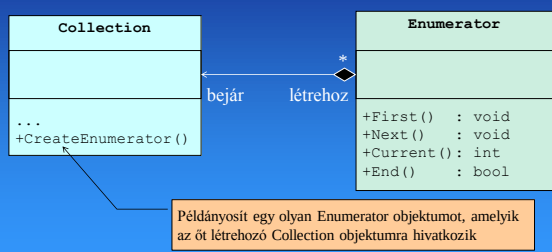
Kétirányú sor bejáró műveletei


```
class BiQueue{
   ...

// bejárásokat biztosító tagok
   public:
      int  Current1()const {return current1->val;}
      void First1()        {current1 = first;}
      bool End1()const     {return current1==NULL;}
      void Next1()         {current1 = current1->next;}

      int  Current2()const {return current2->val;}
      void First2()        {current2 = first;}
      bool End2()const     {return current2==NULL;}
      void Next2()         {current1 = current2->next;}

   private:
      Node     *current1;
      Node     *current2;
};
```

?

biqueue.h

Bejáró tervminta



Kétirányú sort bejáró osztály


```
class BiQueue{
   ...

// bejárásokat biztosító osztály
 public:
   class Enumerator{
      public:
         Enumerator(BiQueue *s): bq(s),current(NULL){}
         int  Current()const {return current->val;}
         void First()    {current = bq->first;}
         bool End()const {return current==NULL;}
         void Next()     {current = current->next;}
      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, 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

Kétirányú sor osztálya


```
public:
   class Enumerator{
      public:
         Enumerator(BiQueue *s):bq(s),current(NULL)
                                {++(bq->enumeratorCount);}
         ~Enumerator()          {--(bq->enumeratorCount);}
         int Current()const {return current->val;}
         void First()       {current = bq->first;}
         bool End()const    {return current==NULL;}
         void Next()        {current = current->next;}
      private:
         BiQueue *bq;
         Node     *current;
  };
```

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

25

```
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

Kivételkezelés

26

```
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

Inkrementáló operátor

27

```
class Enumerator{
   public:
        ...
   // it.Next helyett it++
      Enumerator operator++(int){
      Enumerator it = *this;
         current = current->next;
         return it;
      }
   // it.Next helyett ++it
      Enumerator& operator++(){
         current = current->next;
         return *this;
      }
         ...
};
```

biqueue.h

Másoló konstruktor

29

```
BiQueue::BiQueue(const BiQueue &s)
{
   enumeratorCount = s.enumeratorCount;
   if(s.first==NULL){
      first = last = NULL;
   }else{
      try{
         Node *q = new Node(s.first->val,NULL,NULL);
         first = q;
         for(Node *p=s.first->next;p!=NULL;p=p->next){
            q = new Node(p->val,NULL,q);
            q->prev->next = q;
         }
      }catch(std::bad_alloc o){
         ...
      }
   }
   last = q;
}
```

```
Node *p;
while(first!=NULL){
   p = first;
   first = first->next;
   delete p;
}
throw FULLMEM;
```

biqueue.cpp