Objektum elvű alkalmazások fejlesztése

Kétirányú sor sablonja

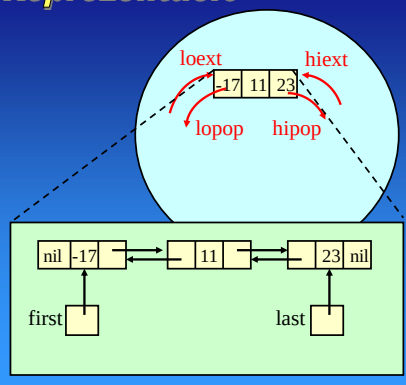
Készítette: Gregorics Tibor

Cél

Módosítsuk a kétirányú sor osztályunkat úgy, hogy ne csak egész értékeket lehessen benne tárolni, hanem tetszőleges típusú értékeket.

1. Mindig átírhatjuk az osztály definícióban az elemi típust (ez most int) arra, amire szükségünk van.
2. Elkészítjük minden típus ősosztályát, és ekkor mindenféle típusú elemet tárolhatunk ugyanabban a kétirányú sorban. A kiolvasott elemet azonban konvertálni (castolni) kell.
3. Osztály-sablont készítünk, ahol az elemi típus később megadandó paraméter.

Reprezentáció



Kétirányú sor osztály 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;
   int  _enumeratorCount;
```

Ez nem az elem típusa

biqueue.h

Kétirányú sor osztály publikus része

```
public:
   enum Exceptions{EMPTYSEQ, UNDERTRAVERSAL};

   BiQueue():_first(NULL),_last(NULL),_enumeratorCount(0){}
   BiQueue(const BiQueue&);
   BiQueue& operator=(const BiQueue&);
   ~BiQueue();

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

   ...
};
```

biqueue.h

Kétirányú sor bejáró osztálya

```
   friend class Enumerator;

   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;
   };
   Enumerator createEnumerator()
              { return Enumerator(this); }
};
```

biqueue.h

Milyen jó volna, ha lenne egy általános BiQueue

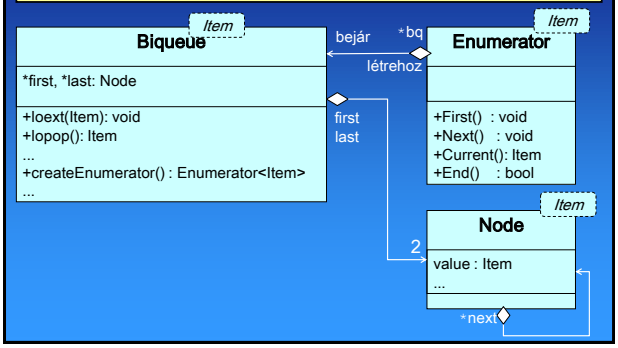
```
int main()
{
   BiQueue<int>          yi;
   BiQueue<char>         ya;
   BiQueue<string>       ys;
   BiQueue<Test*>        y1, y2;

   yi.loext(4);
   ya.loext('w');
   ys.loext("alma");
   y1.loext(new Kocka(3.0));
   …
```

main.cpp

Kétirányú sor általánosítása

A kétirányú sor típusnak lesz egy *Item* paramétere, amely a sor elemeinek típusát mutatja.



Kétirányú sor osztály privát része

```
template <typename Item>
class BiQueue{
private:

   struct Node{
      Item  _val;
      Node *_next;
      Node *_prev;
      Node(const Item &e, Node *n, Node *p)
         :_val(e),_next(n), _prev(p){}
   };

   Node *_first;
   Node *_last;

   int   _enumeratorCount;
```

BiQueue<Item> osztály-sablont definiál az Item sablon paraméterrel

A Node egy beágyazott definíció a BiQueue<Item>-ben, ezért ő egy Node<Item> sablon.

mivel a sablon helyébe kerülhet összetett típus is, ezért a bemenő paraméter const-ref legyen

biqueue.hpp

Kétirányú sor osztály publikus része

```
public:
   enum Exceptions{EMPTYSEQ, UNDERTRAVERSAL};

   BiQueue():_first(NULL),_last(NULL),_enumeratorCount(0){}
   BiQueue(const BiQueue&);
   BiQueue& operator=(const BiQueue&);
   ~BiQueue();

   void loext(const Item &e);
   Item lopop();
   void hiext(const Item &e);
   Item hipop();
```

biqueue.hpp

Kétirányú sor bejáró osztálya

```
   class Enumerator{
      public:
         Enumerator(BiQueue<Item> *s):_bq(s),_current(NULL)
                                 { ++(_bq->_enumeratorCount); }
         ~Enumerator()           { --(_bq->_enumeratorCount); }
         Item current() const {return _current->_val;}
         void first()         {_current = _bq->_first;}
         bool end()const      {return _current==NULL;}
         void next()          {_current = _current->_next;}

      private:
         BiQueue<Item> *_bq;
         Node  *_current;
   };

   Enumerator createEnumerator(){return Enumerator(this);}
};
```

Az Enumerator egy beágyazott definíciója a BiQueue<Item>-ben, ő egy Enumerator<Item> sablon

biqueue.hpp

Lopop művelet

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

BiQueue<Item> osztály-sablonhoz tartozó metódus-sablon

Vajon a sablon paramétert helyettesítő típusra értelmezve lesz-e a másoló konstruktor? Vajon értelmezve lesz-e az Item-re rá minden itt a kódban használt művelet?

biqueue.hpp

Hipop művelet

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

biqueue.hpp

Loext

```
template <typename Item>
void BiQueue<Item>::loext(const Item &e)
{
   Node *p = new Node(e, _first, NULL);
   if(_first!=NULL) _first->_prev = p;
   _first = p;
   if(_last==NULL) _last = p;
}
```

Hiext

```
template <typename Item>
void BiQueue<Item>::hiext(const Item &e)
{
   Node *p = new Node(e, NULL, _last);
   if(_last!=NULL) _last->_next = p;
   _last = p;
   if(_first==NULL) _first = p;
}
```

biqueue.hpp

Copy konstruktor

```
template <typename Item>
BiQueue<Item>::BiQueue(const BiQueue &s)
{
   _enumeratorCount = s._enumeratorCount;
   if(s._first==NULL){
      _first = _last = NULL;
   }else{
      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;
      }
      _last = q;
   }
}
```

biqueue.hpp

Értékadás operátor

```
template <typename Item>
BiQueue<Item>& BiQueue<Item>::operator=(const BiQueue &s)
{
    if (this==&s) return *this;

    // destruktor

    // copy konstruktor

    return *this;
}
```

biqueue.hpp

Destruktor

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

biqueue.hpp

Csomag diagram

Egy sablon önmagában nem fordítható,
ezért minden elemét egy közös fejállományba tesszük.

biqueue.h

```
class Biqueue{
  private:
    …
  public:
    …
    class Enumerator{
    …
    };
};
```

biqueue.hpp

```
template <typename Item>
class Biqueue{
  private:
    …
  public:
    …
    class Enumerator{
    …
    };
};
template <typename Item>
Biqueue::Biqueue(){ … }
template <typename Item>
void loext(const Item &e){ … }
template <typename Item>
Item Biqueue::lopop(){ … }
…
```

biqueue.cpp

```
Biqueue::Biqueue(){ … }
void loext(int e){ … }
int Biqueue::lopop(){ … }
…
```

Feladat

Olvassuk be a standard bemenetről érkező sztringeket, majd írjuk ki őket az érkezésük sorrendjében a standard kimenetre úgy, hogy megadjuk minden sztring minden előfordulásánál annak összes előfordulásának számát!

Főprogram

```
#include <iostream>
#include "biqueue.hpp"

using namespace std;

int main()
{
   BiQueue<string> x;

   string str;
   cin >> str;
   while(str != "quit"){
      x.hiext(str);
      cin >> str;
   }
```

osztály példányosítás a sablon alapján fordítási időben

objektum létrehozás futási időben

main.cpp

Főprogram

```
   BiQueue<string>::Enumerator it1 = x.createEnumerator();
   for(it1.first(); !it1.end(); it1.next()){
      string s = it1.current();

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

A `BiQueue` sablon példányosítása a beágyazott `Enumerator` sablont is példányosította: ez itt az `Enumerator<string>` osztály

main.cpp

Kivételkezelés

```
BiQueue<int> x;
BiQueue<string> y;

try{
   int i    = x.lopop();
   string s = y.lopop();

}catch(BiQueue<int>::Exceptions e){
   if(e == BiQueue<int>::UNDERTRAVERSAL)
      cout << " ... " << endl;
   }

}catch(BiQueue<string>::Exceptions e){
   if(e == BiQueue<string>::UNDERTRAVERSAL)
      cout << " ... " << endl;
   }
}
```

Sablon-paraméterek fajtái

- Típust meghatározó paraméter
  - **template** <**typename** Item>
    class *T* { … };
- Típust meghatározó paraméter
  - **template** <**typename** BiQueue<Item> >
    class *T* { … };
- Értéket meghatározó paraméter
  - **template** <**int** size>
    class *T* { … };

Idegen tollak

```
template <int N>
struct Factorial
{
   enum { value = N*Factorial<N-1>::value };
};

template<>
struct Factorial<1>
{
   enum { value = 1};
};

int main()
{
   int r = Factorial<5>::value;
}
```