FEJLETT PROGRAMOZÁSI NYELVEK, 2009
6. GYAKORLAT – Operátorok túlterhelése

Cél:

- Racionális szám osztály
  - aritmetikai operátorok: `+, -, *, \`
  - inserter (<<) és extractor (>>) operátor
  - konverziós operátor: racionális -> valós
- Lista osztály beágyazott **iterátor** osztállyal
  - értékadó operátor: =
  - egyenlőségi operátorok: ==, !=
  - növelés és csökkentés> ++, -- (elő és utótagként)
  - * és -> operátorok

1. Készítsünk osztályt, amely biztosítja a **racionális számok** ábrázolását és ezek műveleteit. *Próbálja ki a megadott **main** függvényt.*

```
class rational{
    int m,n;
    //Egyszerűsíti a törtet: osztja a számlálót, illetve a nevezőt ezek
    //legnagyobb közös osztójával
    //Ezt meg kell hívni minden egyes aritmetikai műveletnél a
    //visszatérített racionális objektumra

    void simplify();
public:
    rational( int m=0, int n=1) : m(m), n( n==0 ? 1 : n){simplify();}

    //I/O  műveletek
    friend ostream& operator<<( ostream& os, const rational& r);
    friend istream& operator>>( istream& is, rational& r);

    //aritmetikai műveletek
    friend rational operator+( const rational& r1, const rational& r2 );
    friend rational operator-( const rational& r1, const rational& r2 );
    friend rational operator*( const rational& r1, const rational& r2 );
    friend rational operator/( const rational& r1, const rational& r2 );

    //relációs műveletek
    friend bool operator< ( const rational& r1, const rational& r2 );
    friend bool operator<=( const rational& r1, const rational& r2 );
    friend bool operator> ( const rational& r1, const rational& r2 );
    friend bool operator>=( const rational& r1, const rational& r2 );

    //egyenlőségi műveletek
    friend bool operator== ( const rational& r1, const rational& r2 );
    friend bool operator!=( const rational& r1, const rational& r2 );
};
#include <cstdlib>
```

FEJLETT PROGRAMOZÁSI NYELVEK, 2009
6. GYAKORLAT – Operátorok túlterhelése

```
#include<iostream>
#include "rational.h"
using namespace std;

int main(int argc, char** argv) {
    rational a(2,4), b(16,4), c;
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
    cout<<"c="<<c<<endl;
    c =  a + b;
    cout<<"c="<<c<<endl;
    cout<<"a<b : "<<(a<b)<<endl;
    cout<<"c == a : " <<(c==a) << endl;

    return (EXIT_SUCCESS);
}
```

FEJLETT PROGRAMOZÁSI NYELVEK, 2009
6. GYAKORLAT – Operátorok túlterhelése

2. Készítsük el az egyszeresen láncolt listát az alább megadott váz kitöltésével. A láncolt lista feldolgozását egy beágyazott iterátor osztály biztosítja.

A váz kitöltését úgy kell végeznünk, hogy a megadott main függvény helyesen működjön.

```
#include <iostream>
using namespace std;

template<class T>
class List{
private:
      struct Element{
         T data;
         Element *next;
         Element(T data=T(),Element *next=0):data(data), next(next){}
      };

     Element * head;
public:
      List() : head( 0 ){}
      ~List(){ //Kód }
      void insertFirst( T data ){ //Kód }
      void removeFirst( ){ //Kód }

      class iterator{
      //Aktuális pozíció kód
      public:
            iterator( Element * act=0) { //Kód }
            iterator( const iterator& it ){ //Kód }
            iterator& operator=( const iterator& it ){ //Kód }
            //Elotag
            iterator operator ++(){ //Kód }
            //Utotag
            iterator operator ++( int a ){ //Kód }
            T& operator *(){ //Kód }
            T* operator ->(){ //Kód }
            friend bool operator==(const iterator& it1,
                                   const iterator& it2){
            //Kód
            }
            friend bool operator!=( const iterator& it1,
                                   const iterator& it2 ){
            //Kód
            }
      };
      iterator begin(){ //Kód }
      iterator end(){ //Kód }
};
```

FEJLETT PROGRAMOZÁSI NYELVEK, 2009
6. GYAKORLAT – Operátorok túlterhelése

```
int main(){
	List<int> l1;
	for( int i=0; i<10; ++i )
		l1.insertFirst( i );
	List<int>::iterator it;
	for( it=l1.begin(); it != l1.end(); it++ )
		cout<<*it<<endl;
	for( it=l1.begin(); it != l1.end(); it++ )
		*it = 10;
	for( it=l1.begin(); it != l1.end(); it++ )
		cout<<*it<<endl;
	return 0;
}
```