Задача 4. (15 точки) Класът DLList, представя списък с две връзки, съдържащ числа с плаваща запетая. Дефиницията му е следната:

```
struct node {
  double data;
  node* next;
  node* prev;
};

class DLList {
  private:
    node* start;         // указател към началото на списъка
    node* end;           // указател към края на списъка
    node* forwardIter;   // итератор за обхождане напред
    node* backwardIter;  // итератор за обхождане назад
  public:
    // функции от голямата четворка
    DLList();
    DLList(DLList const&);
    DLList& operator=(DLList const&);
    ~DLList();
    // вмъква числото x пред елемента, сочен от p
    void insertBefore(node* p, double x);
    // вмъква числото x след елемента, сочен от p
    void insertAfter(node* p, double x);
    // изтрива елемента, сочен от p от списъка
    // и го записва в x
    void deleteElem(node* p, double& x);
    // установява forwardIter да сочи към началото на списъка
    // или към елемента, сочен от p, ако за p е зададена
    // ненулева стойност
    void startForward(node* p = NULL);
    // установява backwardIter да сочи към края на списъка
    // или към елемента, сочен от p, ако за p е зададена
    // ненулева стойност
    void startBackward(node* p=NULL);
    // премества forwardIter напред и връща старата му стойност
    node* nextForward();
    // премества backwardIter назад и връща старата му стойност
    node* nextBackward();
};
```

а) (3 т.) Да се дефинират член-функциите от голямата четворка;

б) (3 т.) Да се дефинират член-функциите insertBefore и insertAfter;

в) (2 т.) Да се дефинира член-функцията deleteElem;

г) (3 т.) Да се дефинира функцията bool mirror(DLList& dl1, DLList& dl2), която проверява дали списъкът dl1 е огледален образ на dl2, т.е. дали dl1 се състои от елементите на dl2, разглеждани в обратен ред;

д) (4 т.) Да се дефинира функцията node* split(DLList& dl, double x), която с едновременно обхождане от началото и от края нарежда елементите на списъка dl в две области така, че в началото на списъка да се окажат само числа, по-малки от x, а в края на списъка — числа, по-големи от x. За целта може да се използва следният алгоритъм: списъкът dl се обхожда от двата края до момента, в който бъдат достигнати два елемента a (от началото) и b (от края), такива че a > x > b. При достигане на такива елементи те да бъдат разменени и обхождането да продължи със същата стратегия. Като резултат да бъде върната границата между двете области, т.е. указател към клетката, съдържаща последния по ред елемент в dl, който е по-малък от **x**.

Задача 5. (10 точки) Какъв е резултатът от изпълнението на програмата:

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

class A {
private: int n; double d;
public:
      A(int a = 0): n(a), d(a) { d = 1.5; dump(); }
      A(const A& p) { n = p.n + 1; d = p.d + 1.5; dump(); }
      A& operator=(const A& p){
            if(this!=&p){
                  n = p.n;
                  d = p.d;
                  dump();
            }
            return *this;
      }
      void dump() { cout << "A(n): " << n << ", A(d): " << d << endl;}
};

class B {
private: int n; double d;
public:
      B(double b = 2.5) { n = 2; d = b; dump(); }
      B(const B& p) { n = p.n; d = p.d; dump(); }
      B& operator=(const B& p){
            if(this!=&p){
                  n = p.n + 2;
                  d = p.d + 2.5;
                  dump();
            }
            return *this;
      }
      void dump() { cout << "B(n): " << n << ", B(d): " << d << endl;}

};

class C : protected B, A {
private: int n, m;
public:
      C(int x = 3, int y = 2, int z = 1): A(x), B(y) { n = z; m = x + y; dump();}
      C& operator=(const C& p) {
            if (this!=&p) {
                  A::operator =(p);
                  B::operator =(p);
                  n = p.n;
                  m = p.m;
                  dump();
            }
            return *this;
      }
      void dump() { cout << "C(n): " << n << ", C(m): " << m << endl;}
};

void main() {
      A b(3);
      C c1, c2(1, 2, 3), c3(4, 5);
      C c4(c1);
      c4 = c2;
}
```