3 OGP in C++

Oefening 120

```
// De gegeven methode mijn_ggd is ook nuttig buiten de Breuk-klasse.
// In Java zou je ervoor kiezen deze 'static' te maken
// (je kan immers niets buiten een klasse schrijven in Java);
// in C++ kan je een functie gerust extern schrijven.

int mijn_ggd(int a, int b){ // ... }

class Breuk{
  private:
     int teller, noemer;
     void normaliseer();

  public:
     // voor deel 1
     Breuk(int t=0, int n=1):teller(t),noemer(n) {  //initializer list
          normaliseer();
     }
     // merk op: operator= en copyconstructor moet je niet schrijven
     //     want die bestaan al (en hun standaardwerking volstaat:
     //     er zijn immers geen pointers als dataleden)

     // is geen lidfunctie;
     friend ostream& operator<<(ostream & uit, const Breuk & b);

     Breuk& operator+=(const Breuk& b);
     Breuk& operator-=(const Breuk& b);

     Breuk operator+(const Breuk& b) const;
     Breuk operator-(const Breuk& b) const;

     Breuk operator-() const;

     Breuk& operator++();
     Breuk operator++(int a);

     // voor deel 2
     Breuk operator+(int x) const;
     bool operator<(const Breuk& b) const;

     int get_teller() const { return teller; } //hier gedefineerd
     friend Breuk operator+(int x, const Breuk& b);

     // voor deel 3
     bool operator==(const Breuk& b) const;
     bool operator!=(const Breuk& b) const;
     friend istream& operator>>(istream& in, Breuk & b);
};

//lidfunctie
void Breuk::normaliseer(){
     if(noemer < 0) {
          noemer *= -1; teller *= -1;
     }
     int deler = mijn_ggd(teller,noemer);
     teller /= deler; noemer /= deler;
}
```

```
//Extern (friend van Breuk)
ostream& operator<<(ostream & uit, const Breuk & b) {
     uit << b.teller ;
     if(b.noemer != 1) uit << "/" << b.noemer;
     return uit;
}

//Lidfuncties
Breuk& Breuk::operator+=(const Breuk & b) {
     teller = b.noemer * teller + noemer * b.teller;
     noemer = noemer * b.noemer;
     normaliseer();
     return *this;
}

Breuk& Breuk::operator-=(const Breuk & b) {
     operator+=(-b);
     return *this;
}

Breuk Breuk::operator+(const Breuk & b) const {
     return Breuk( b.noemer * teller + noemer * b.teller,noemer * b.noemer);
}
//Alternatief die gebruik maakt van de operator +=
/*
Breuk Breuk::operator+(const Breuk & b) const {
     Breuk c(*this);  // Gebruikt de copyconstructor (default-versie voldoet)
         // Dat is efficienter dan Breuk c = *this (maakt eerst een Breuk aan met
         // de defaultconstructor, om daarna weer de dataleden te overschrijven).
     c += b;  //gebruikt +=
     return c;
}  */

Breuk Breuk::operator-(const Breuk & b) const {
     return Breuk( b.noemer * teller - noemer * b.teller,noemer * b.noemer);
}
//Alternatief die gebruik maakt van de operator -=
/*
Breuk Breuk::operator-(const Breuk& b) const {
     Breuk c(*this);
     c -= b;
     return c;
}  */

Breuk Breuk::operator-() const {
     return Breuk(-teller,noemer);
}

Breuk& Breuk::operator++(){
     teller += noemer;
     normaliseer();
     return *this;
}

Breuk Breuk::operator++(int a){
     Breuk temp(*this);
     teller += noemer;
     normaliseer();
     return temp;
}
```

```
/***************** DEEL 2 ********************/

Breuk Breuk::operator+(int x) const{
     Breuk c(*this);
     x *= c.noemer;
     c.teller += x;
     return c;
}

bool Breuk::operator<(const Breuk& b) const {
     return teller * b.noemer < noemer * b.teller;
}

//Extern (friend van Breuk)
Breuk operator+(int x, const Breuk& b){
     return b+x;
}

bool is_stambreuk(const Breuk & a){
     return a.get_teller()==1;
}

/***************** DEEL 3 ********************/

bool Breuk::operator==(const Breuk& b) const {
     return teller == b.teller && noemer == b.noemer;
}

bool Breuk::operator!=(const Breuk & b) const {
     return !operator==(b);
}

//Twee oplossingen voor het inlezen met "/" tussen teller en noemer
//Voor test en examen moet je enkel de getallen kunnen inlezen
//met spatie ertussen

//Extern (friend van Breuk)
//Gebruikt de functie  int atoi(char *)
istream& operator>>(istream &in,  Breuk & b) {
    string lijn;
    getline(in,lijn);
    int teller,noemer;
    int p = lijn.find("/");
    if(p!=string::npos){
        teller = atoi(lijn.substr(0,p).c_str());
        noemer = atoi(lijn.substr(p+1).c_str());
        b = Breuk(teller,noemer);
    }
    else{
        int getal = atoi(lijn.c_str());
        b = Breuk(getal);
    }
    return in;
}

//Gebruikt stringstream (te vergelijken met Scanner)
/*
istream& operator>>(istream& in, Breuk & b) {
     string getalbeeld;
     in >> getalbeeld;
     stringstream ss; ss << getalbeeld;
     int positie = getalbeeld.find("/");
```

```
     if(positie != string::npos) {
          int t; char c; int n;
          ss >> t; ss >> c; ss >> n;
          if(c == '/' && !ss.fail())
              b = Breuk(t,n);
          else
              b = Breuk();
     }
     else {  // misschien is er geen breukstreep, omdat je
             // enkel een geheel getal (dus met noemer = 1) opgaf
          int t; ss >> t;
          string overschot; ss >> overschot;
          if(overschot == "")
              b = Breuk(t);
          else
              b = Breuk();
     }
     return in;
}*/
```

Oefening 121

```
#include <iostream>
#include <vector>

using namespace std;

template <typename T>
class Doos;

template <typename T>
class Schijf {

    public:
        Schijf();
        Schijf(const Schijf<T>& );
        Schijf<T>& operator=(const Schijf<T>&);
        ~Schijf<T>();

    private:
        Doos<T> *a;
};

template <typename T>
class Doos {

    public:
        Doos();
        Doos(const Doos<T>&doos);
        Doos<T>& operator=(const Doos<T>&);
        ~Doos();

   private:
        vector<T> b;
        Schijf<T> **d;
        Doos<T> *c;
};
```

```
/****************************** SCHIJF ****************************/

template<typename T>
Schijf<T> :: Schijf():a(0){}  //initializer list

template<typename T>
Schijf<T> :: Schijf(const Schijf<T>& schijf){
     if(schijf.a != 0){
         a=new Doos<T>(*(schijf.a));
     }
     else
         a=0;
}

template<typename T>
Schijf<T> :: ~Schijf<T>(){
     delete a;
}

template <typename T>
Schijf<T>& Schijf<T> :: operator=(const Schijf<T>& schijf){
     if (this!=&schijf) {
          delete a;
          a=0;
          if (schijf.a!=0){
               a=new Doos<T>(*(schijf.a));
          }
     }
     return *this;
}

/******************************** DOOS *************************/

template<typename T>
Doos<T> :: Doos():b(4),c(0){ //initializer list
     d=new Schijf<T>*[3];
     for(int i=0;i<3;i++)
         d[i]=0;
}

template<typename T>
Doos<T> :: Doos(const Doos<T>& doos){
     b = doos.b;  //vector kopieren
     if(doos.c != 0){
         c = new Doos<T>(doos.c);
     }
     else
         c=0;
     d = new Schijf<T>*[3]; //3 elementen in de array
    // de elementen van de array d moeten nu elk een nieuwe schijf toegewezen
       krijgen, als
    // de parameter 'doos' daar ook een schijf heeft.
     for(int i=0; i<3; i++){
          if(doos.d[i] != 0) {
              d[i] = new Schijf<T>(*doos.d[i]);
          }
          else
              d[i] = 0;
     }
}
```

```
template<typename T>
Doos<T>& Doos<T> :: operator=(const Doos<T>&doos){
     if (this!=&doos) {
          b = doos.b;  //vector kopieren
          delete c;
          if(doos.c!=0){
               c = new Doos<T>(*(doos.c));
          }
          // de elementen van de array d moeten nu elk een nieuwe schijf toegewezen
              krijgen, als
          // de parameter 'doos' daar ook een schijf heeft.
          for(int i=0; i<3; i++){
               delete d[i];
               if (doos.d[i]!=0) {
                   d[i]=new Schijf<T>(*(doos.d[i]));
               }
               else
                   d[i]=0;
          }
     }
     return *this;
}

template<typename T>
Doos<T> :: ~Doos(){
     delete c;
     for(int i=0;i<3;i++) {
         delete(d[i]);
     }
     delete []d;
}
```