7. Übung zur Computeralgebra I

Prof. Dr. Plesken **(SS 2006)**

Aufgabe 1. (Hilbert-Reihe)
Sei $n \in \mathbb{N}$ und K ein Körper. Zeige, daß $K[x_1, \ldots, x_n]$ mit der üblichen Graduierung die Hilbert-Reihe

$$\frac{1}{(1-t)^n} = \sum_{i \geq 0} \binom{n+i-1}{i} t^i$$

besitzt.

Aufgabe 2. (Hilbert-Polynom)
Sei $n \in \mathbb{N}$, K ein Körper, $R = K[x_1, \ldots, x_n]$ und M ein endlich erzeugter graduierter R-Modul. Zeige: Es gibt ein Polynom $HP_M \in \mathbb{Q}[t]$, so daß $\dim_K M_i = HP_M(i)$ gilt für alle hinreichend großen $i \in \mathbb{Z}$. HP_M heißt das *Hilbert-Polynom* von M.

Aufgabe 3. (Polynomklasse in C++)
Erweitere die C++-Bibliothek GiNaC (`www.ginac.de`) um eine Klasse für multivariate Polynome, die es ermöglicht, führende Monome und führende Koeffizienten solcher Polynome bzgl. der grad-invers-lexikographischen Monomordnung $<_{\text{degrevlex}}$ und der rein-lexikographischen Monomordnung $<_{\text{plex}}$ zu bestimmen. Teste diese Klasse an ein paar Beispielen.

Hinweis: Die beiden folgenden Seiten erleichtern den Einstieg.

Die Monomordnungen $<_{\text{degrevlex}}$ und $<_{\text{plex}}$ sind wie folgt definiert: Für zwei Monome $m_1 = x_1^{a_1} \cdot \ldots \cdot x_n^{a_n}$ und $m_2 = x_1^{b_1} \cdot \ldots \cdot x_n^{b_n}$ ist

$$\begin{aligned} m_1 <_{\text{degrevlex}} m_2 \quad :\Longleftrightarrow \quad & \text{Grad } m_1 < \text{Grad } m_2 \quad \text{oder} \\ & \text{Grad } m_1 = \text{Grad } m_2 \quad \text{und} \quad m_1 \neq m_2 \quad \text{und} \\ & a_i > b_i \quad \text{für} \quad i = \max\{j \mid a_j \neq b_j\} \end{aligned}$$

bzw.

$$m_1 <_{\text{plex}} m_2 \quad :\Longleftrightarrow \quad m_1 \neq m_2 \quad \text{und} \quad a_i < b_i \quad \text{für} \quad i = \min\{j \mid a_j \neq b_j\}.$$

Abgabe: Mittwoch, 24.05.2006, in der Übung.

```
// ----------------------------------------------------------------------
// poly.h
// ----------------------------------------------------------------------
#ifndef __POLY_H__
#define __POLY_H__

#include <ginac/ginac.h>
#include <iostream>

namespace GiNaC {

using namespace std;

class poly : public ex {
private:
    lst indeterminates;

public:
    poly(const lst &) throw();
    poly(const poly &);
    poly(const ex &, const lst &);
    poly(const std::string &, const lst &);

    poly & operator=(const poly &);

    poly leadmon_degrevlex() const;
    numeric leadcoeff_degrevlex() const;

    poly leadmon_plex() const;
    numeric leadcoeff_plex() const;

    friend istream& operator>> (istream &, poly &);
};

inline poly::poly(const lst &indets) throw() : ex() {
    for (lst::const_iterator i = indets.begin(); i != indets.end(); ++i)
        indeterminates.append(*i);
}

inline poly::poly(const poly &p) : ex(p) {
    for (lst::const_iterator i = p.indeterminates.begin();
        i != p.indeterminates.end(); ++i)
        indeterminates.append(*i);
}

inline poly::poly(const ex &p, const lst &indets) : ex(p) {
    for (lst::const_iterator i = indets.begin(); i != indets.end(); ++i)
        indeterminates.append(*i);
}

inline poly::poly(const std::string &s, const lst &indets): ex(s, indets) {
    for (lst::const_iterator i = indets.begin(); i != indets.end(); ++i)
        indeterminates.append(*i);
}
```

```
inline poly & poly::operator=(const poly &other) {
    ex::operator=(other);
    indeterminates.remove_all();
    for (lst::const_iterator i = other.indeterminates.begin();
        i != other.indeterminates.end(); ++i)
        indeterminates.append(*i);
}

} // namespace GiNaC

#endif // ndef __POLY_H__


// ----------------------------------------------------------------------
// poly.cpp
// ----------------------------------------------------------------------
#include <algorithm>
#include "./poly.h"

namespace GiNaC {

struct get_degree : public unary_function<ex, int>
{
    ex p;
    get_degree(const ex &q) : p(q) {}
    int operator() (const ex &x) { return p.degree(x); }
};

list<int> exponents(const ex &p, const lst &indets)
{
    list<int> e;
    e.insert(e.end(), indets.nops(), 0);
    transform(indets.begin(), indets.end(), e.begin(), get_degree(p));
    return e;
}

// ... ... ...

istream& operator>> (istream &i, poly &p)
{
    string s;
    getline(i, s);
    try {
        poly q(s, p.indeterminates);
        p = q;
        return i;
    } catch (exception &x) {
        cerr << x.what() << endl;
        exit(1);
    }
}

} // namespace GiNaC
```