Ludwig-Maximilians-Universität München
Mathematisches Institut
Gregor Kleen

Programmieren II für Studierende der Mathematik

Blatt 2 – Lösungsvorschlag

Aufgabe 2 Erstellen Sie eine Klasse `Polynom`, die Polynome über den reellen Zahlen (approximiert durch `double`) $p = \sum_{k=0}^{n} a_k x^k$ intern mithilfe einer Datenkomponente `a` vom Typ `vector<double>` speichert. Es sollen nur die Koeffizienten a_k bis einschließlich des Höchstkoeffizienten gespeichert werden. Der Höchstkoeffizient ist per Definition ungleich 0.

Um zu verhindern, dass die Invariante bzgl. des Höchstkoeffizienten verletzt wird, also das Element des internen Vektors mit höchstem Index stets ungleich 0 ist, wird `a` als `private` vereinbart.
Es wird zudem im Konstruktor, der einen `vector<double>` als Argument nimmt, und im Folgenden darauf geachtet dass alle neu gebildeten Polynome stets normalisiert werden.

Klasse

```
class Polynom {
  private:
    vector<double> a;

    explicit Polynom(int n) : a(n + 1, 0) {}
    void normalize() {
      while (a.size() > 0 && a.back() == 0)
        a.pop_back();
    }

  public:
    Konstruktor Nullpolynom
    Polynom(const vector<double>& a_) : a(a_) {
      normalize();
    }

    grad
    deriv

    Arithmetische Operatoren

    Einausgabe
};
```

Das Nullpolynom soll durch einen leereren Koeffizientenvektor dargestellt werden[1] und durch einen Konstruktor mit leerer Argumentenliste erzeugt werden können.

[1] Überzeugen Sie sich davon, dass die Invariante bzgl. des Höchstkoeffizienten hierdurch nicht verletzt wird

Konstruktor Nullpolynom

```
Polynom() : a(0) {}
```

Erstellen Sie die folgenden Methoden:

grad Berechnet den Grad des Polynoms auf das sie angewandt wird möglichst effizient und gibt ihn als `int` zurück (-1 für das Nullpolynom)

grad

```
int grad() const { return a.size() - 1; }
```

deriv Liefert für das Polynom p das abgeleitete Polynom p'

deriv

```
Polynom deriv() const {
  if (grad() <= 0)
    return Polynom{};

  Polynom p{grad() - 1};
  for (unsigned int i = 1; i < a.size(); i++)
    p.a[i - 1] = i * a[i];

  return p;
}
```

Überladen Sie die arithmetischen Operatoren für Addition, Multiplikation und Subtraktion von Objekten der Klasse `Polynom` in mathematisch sinnvoller Weise. Zur Implementierung der Multiplikation können Sie folgende Beziehung verwenden:

$$\left(\sum_{i=0}^{m} a_i x^i\right) \cdot \left(\sum_{j=0}^{n} b_j x^j\right) = \sum_{k=0}^{m+n} c_k x^k \quad \text{mit} \quad c_k = \sum_{i=\max(0,k-n)}^{\min(k,m)} a_i b_{k-i}$$

Hinweis. Zur Berechnung von min und max stehen in der Standarbibliothek (`<algorithm>`) bereits gleichnamige Implementierungen für beliebige Typen zur Verfügung.

Arithmetische Operatoren

```
friend Polynom operator+(Polynom p, Polynom q) {
  int m = p.grad(), n = q.grad();

  Polynom r{max(m, n)};
  for (int i = 0; i < max(m, n) + 1; i++) {
    if (i < m + 1)
      r.a[i] += p.a[i];
    if (i < n + 1)
      r.a[i] += q.a[i];
  }

  r.normalize();
```

```
  return r;
}

Polynom operator-() const {
  Polynom r{*this};

  for (unsigned int i = 0; i < r.a.size(); i++)
    r.a[i] = -r.a[i];

  return r;
}
friend Polynom operator-(Polynom p, Polynom q) { return p + (-q); }

friend Polynom operator*(Polynom p, Polynom q) {
  int m = p.grad(), n = q.grad();

  if (m < 0 || n < 0)
    return Polynom{};

  Polynom r{m + n};
  for (int k = 0; k <= m + n; k++)
    for (int i = max(0, k - n); i <= min(k, m); i++)
      r.a[k] += p.a[i] * q.a[k - i];

  return r;
}
```

Polynome sollen durch Einlesen von n a_0 a_1 a_2 ... a_n von der Standardeingabe erstellt werden können. Zudem sollen Polynome in der Form a_0 + a_1*x + a_2*x^2 + ... + a_n*x^n ausgegeben werden können. Überladen Sie die Shift-Operatoren entsprechend.

Einausgabe

```
friend istream& operator>>(istream& stream, Polynom &p) {
  int n;
  stream >> n;

  p = Polynom{max(-1, n)};
  for (unsigned int i = 0; i < p.a.size(); i++)
    stream >> p.a[i];
  p.normalize();

  return stream;
}
friend ostream& operator<<(ostream& stream, const Polynom& p) {
  bool at_begin = true;
  for (unsigned int i = 0; i < p.a.size(); i++) {
    if (p.a[i] == 0)
      continue;

    if (!at_begin)
      stream << " + ";
    else
      at_begin = false;

    if (p.a[i] != 1 || i == 0)
```

```
      stream << p.a[i];
    if (p.a[i] != 1 && i != 0)
      stream << "*";
    if (i != 0)
      stream << "x";
    if (i > 1)
      stream << "^" << i;
  }

  if (at_begin)
    stream << "0";

  return stream;
}
```

Schreiben Sie ein Hauptprogramm in dem Sie zwei Polynome p und q von der Standardeingabe einlesen. Berechnen Sie dann $(pq)'$ und $p'q + pq'$ und geben Sie die beiden Resultate aus. Geben Sie zudem jeweils den Grad der Polynome p, q, $(pq)'$ und $(pq)' - p'q - pq'$ aus.

Hauptprogramm

```
int main() {
  Polynom p, q;

  cout << "p q: ";
  cin >> p >> q;

  // cout << "p   = " << p << endl;
  // cout << "-p  = " << -p << endl;
  // cout << "q   = " << q << endl;
  // cout << "p+q = " << p+q << endl;
  // cout << "p-q = " << p-q << endl;
  // cout << "p*q = " << p*q << endl;

  cout << "(pq)' = " << (p * q).deriv() << endl;
  cout << "p'q + pq' = " << (p.deriv() * q + p * q.deriv()) << endl;

  cout << "grad(p) = " << p.grad() << endl;
  cout << "grad(q) = " << q.grad() << endl;
  cout << "grad((pq)') = " << (p * q).deriv().grad() << endl;
  cout << "grad((pq)' - (p'q + pq')) = " << ((p * q).deriv() - (p.deriv() * q + p *
  ↪  q.deriv())).grad() << endl;

  return 0;
}
```

polynom.cpp

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

using namespace std;
```

Klasse

Hauptprogramm

Führen Sie Ihr Programm aus für $p = 1 - 2x + x^2$ und $q = x + 2x^2 + 3x^3 + 4x^4 + 5x^5$.

```
p q: 2 1 -2 2
5 0 1 2 3 4 5
(pq)' = 1 + 3*x^2 + 8*x^3 + 15*x^4 + -12*x^5 + 70*x^6
p'q + pq' = 1 + 3*x^2 + 8*x^3 + 15*x^4 + -12*x^5 + 70*x^6
grad(p) = 2
grad(q) = 5
grad((pq)') = 6
grad((pq)' - (p'q + pq')) = -1
```