Informatik I für D-ITET Musterlösung 9 WS 04/05

URL: http://www.ti.inf.ethz.ch/ew/courses/inf1-ITET/

Aufgabe 1

Programm: polynom.C

```
// Programm: polynom.C
// Einfache Operationen auf Polynomen.

#include <iostream>
#include <vector>
#include <cassert>

typedef int                               Nt;
typedef std::vector<Nt>                   Polynom;
typedef Polynom::iterator                 It;
typedef Polynom::reverse_iterator         Rit;
typedef Polynom::const_iterator           Cit;
typedef Polynom::const_reverse_iterator   Crit;

Nt eval(const Polynom& p, Nt x)
// PRE: !p.empty().
// POST: Rueckgabewert ist Wert des Polynoms an der Stelle x.
{
  assert(!p.empty());
  // Horner Schema
  Nt r = 0;
  for (Crit i = p.rbegin(); i != p.rend(); ++i)
    r = r * x + *i;
  return r;
}

void add(Polynom& s, const Polynom& a, const Polynom& b)
// PRE: s.empty(), !a.empty(), !b.empty().
// POST: s <- Summe von a und b.
{
  assert(s.empty() && !a.empty() && !b.empty());
  if (a.size() > b.size()) return add(s, b, a);
  // Hier ist a.size() <= b.size().
  Cit ai = a.begin();
  Cit bi = b.begin();
  while (ai != a.end()) {
    s.push_back(*ai + *bi);
    ++ai;
    ++bi;
  }
  // Haenge restliche Koeffizienten von b an.
  while (bi != b.end()) s.push_back(*(bi++));
}

void scalar_multiply(Polynom& p, Nt a)
// PRE: !p.empty().
// POST: p <- Produkt von p und a.
{
  assert(!p.empty());
```

```
  for (It i = p.begin(); i != p.end(); ++i)
    *i *= a;
}

void x_multiply(Polynom& p)
// PRE: !p.empty().
// POST: p <- Produkt von p und x (x ist die Variable von p).
{
  assert(!p.empty());
  // bewege alle Koeffizienten um eine Stelle nach rechts
  p.push_back(0);
  Rit i = p.rbegin();
  Rit prev = i;
  while (++i != p.rend()) {
    *prev = *i;
    prev = i;
  }
  // erster Koeffizient (von x^0) ist Null.
  *prev = 0;
}

void multiply(Polynom& p, const Polynom& a, const Polynom& b)
// PRE: p.empty(), !a.empty(), !b.empty().
// POST: p <- Produkt von a und b.
{
  assert(p.empty() && !a.empty() && !b.empty());
  // Schulmethode: shift and add.
  Polynom ashift = a;
  p.push_back(0);     // Initialisiere p(x) = 0.
  for (Cit i = b.begin(); i != b.end(); ++i) {
    Polynom amult = ashift;
    scalar_multiply(amult, *i);
    Polynom pnew;
    add(pnew, amult, p);
    p = pnew;
    x_multiply(ashift);
  }
}

void output(const Polynom& p)
// PRE: !p.empty().
// POST: Eine Beschreibung von p wurde nach std::cout geschrieben.
{
  assert(!p.empty());
  for (unsigned int i = p.size() - 1; i > 0; --i)
    std::cout << p[i] << "x^" << i << " + ";
  std::cout << p[0];
}

int main()
{
  Polynom p;  // p(x) = x^2 + 2x - 1
  p.push_back(-1);
  p.push_back(2);
  p.push_back(1);
  std::cout << "p(x) = ";
  output(p);
  std::cout << "." << std::endl;
```

```
  std::cout << "p(1) = " << eval(p, 1) << "." << std::endl;

  Polynom q;  // q(x) = 2x^3 - x + 3
  q.push_back(3);
  q.push_back(-1);
  q.push_back(0);
  q.push_back(2);
  std::cout << "q(x) = ";
  output(q);
  std::cout << "." << std::endl;
  std::cout << "q(2) = " << eval(q, 2) << "." << std::endl;

  Polynom s;
  add(s, p, q);
  std::cout << "(p+q)(x) = ";
  output(s);
  std::cout << "." << std::endl;

  Polynom pr;
  multiply(pr, p, q);
  std::cout << "(p*q)(x) = ";
  output(pr);
  std::cout << "." << std::endl;

  return 0;
}
```

Aufgabe 2

Programm: shuffle.C

```
// Programm: shuffle.C
// Erzeugung zufaelliger Permutationen.

#include <iostream>
#include <vector>
#include <IFET/math.h>

typedef std::vector<int>     Vec;
typedef Vec::iterator        It;
typedef Vec::const_iterator  Cit;

namespace ifet {

  It min_element(It b, It e)
  // PRE: [b,e) ist gueltiger range.
  // POST: Rueckgabewert ist der erste Iterator i in [b,e),
  //   fuer den gilt: fuer alle j in [b,e) ist *i <= *j.
  {
    if (b == e) return e;
    It m = b++;
    for (; b != e; ++b)
      if (*b < *m) m = b;
    return m;
  }
```

```
  void min_sort(It b, It e)
  // PRE: [b,e) ist gueltiger range.
  // POST: [b,e) ist aufsteigend sortiert, d.h., fuer alle
  //   Paare i,j aus [b,e) mit j aus [i,e) gilt *i <= *j.
  {
    for (; b != e; ++b)
      std::iter_swap(b, ifet::min_element(b, e));
  }

  void random_shuffle(It b, It e)
  // POST: [b,e) ist zufaellig permutiert. (d.h. die (Multi-)Menge
  // der Elemente ist unveraendert, aber die Reihenfolge ist
  // zufaellig gleichverteilt unter allen moeglichen Reihenfolgen.)
  {
    double rand = ifet::random(0);
    for (; b != e; ++b) {
      std::iter_swap(b, b + int((e-b) * rand));
      rand = ifet::random(rand);
    }
  }

  void output(const Vec& v)
  // PRE: !v.empty().
  // POST: Eine Beschreibung von v wurde nach std::cout geschrieben.
  {
    std::cout << "v = (";
    for (Cit i = v.begin(); i != v.end(); ++i)
      std::cout << *i << " ";
    std::cout << ")" << std::endl;
  }

} // namespace ifet

int main()
{
  Vec v;
  for (int i = 0; i < 10; ++i)
    v.push_back(i);
  ifet::output(v); // v == (0,1,2,3,4,5,6,7,8,9)

  // Permutieren von v
  ifet::random_shuffle(v.begin(), v.end());
  ifet::output(v);

  // Sortieren von v
  ifet::min_sort(v.begin(), v.end());
  ifet::output(v);

  return 0;
}
```