Informatik I für D-ITET Musterlösung 12 WS 04/05

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

Aufgabe 1

Programm: Integer.C

```
#include <iostream>

#include <deque>
#include <cassert>

namespace ifet {

  const unsigned int base = 10000;

  class Integer {

    typedef std::deque<unsigned int>     Vec;
    typedef Vec::iterator                It;
    typedef Vec::const_iterator          Cit;
    typedef Vec::const_reverse_iterator  Crit;

  public:

    Integer();
    // POST: initialisiert zu 0

    Integer(int i);
    // POST: initialisiert zu i

    Integer(unsigned int i);
    // POST: initialisiert zu i

    friend bool operator==(const Integer& x, const Integer& y);
    // POST: true <=> x == y.

    friend bool operator<(const Integer& x, const Integer& y);
    // POST: true <=> x < y.

    bool abs_less(const Integer& x) const;
    // POST: true <=> |*this| < |x|.

    Integer operator-() const;
    // POST: Rueckgabewert ist -x.

    Integer& operator+=(const Integer& x);
    // POST: x zu *this addiert

    Integer& operator-=(const Integer& x);
    // POST: x von *this subtrahiert

    Integer& operator*=(const Integer& x);
    // POST: *this mit x multipliziert

    Integer& operator/=(const Integer& x);
```

```
  // PRE: x != 0
  // POST: *this durch x dividiert

  Integer& operator%=(const Integer& x);
  // PRE: x > 0
  // POST: *this == Rest bei Division durch x

  friend std::ostream& operator<<(std::ostream& o, const Integer& x);

private:

  void init(unsigned int i);
  // POST: Haengt Ziffern (zur Basis base) von i an v_ an.

  Integer& add(const Integer& x);
  // POST: |x| zu |*this| addiert

  Integer& subtract_pos(const Integer& x);
  // PRE: |x| <= |*this|
  // POST: |x| von |*this| subtrahiert

  Integer& mult(unsigned int x);
  // PRE: x < base
  // POST: *this mit x multipliziert

  Integer& div(const Integer& x, Integer& r);
  // POST: *this durch x dividiert:
  //   r == Resultat der Division, *this == Rest der Division

  // repraesentiert durch ein Vorzeichen
  bool s_; // false <=> negative
  // und einen Vektor
  Vec v_;
  // von Ziffern zur Basis base.
  /*
  //                              v_.size()-1
  //                                 ---
  // Die dargestellte Zahl ist       \
  //                                 /    v_[i] * base^i
  //                                 ---
  //                                 i=0
  */
};

bool operator!=(const Integer& x, const Integer& y);
bool operator<=(const Integer& x, const Integer& y);
bool operator>(const Integer& x, const Integer& y);
bool operator>=(const Integer& x, const Integer& y);

Integer operator+(const Integer& x, const Integer& y);
Integer operator-(const Integer& x, const Integer& y);
Integer operator*(const Integer& x, const Integer& y);
Integer operator/(const Integer& x, const Integer& y);
Integer operator%(const Integer& x, const Integer& y);

//
// Public Memberfunktionen der Klasse Integer
//
```

```
  Integer::Integer() : s_(true)
  {
    v_.push_back(0);
  }

  void Integer::init(unsigned int i)
  {
    if (i == 0)
      v_.push_back(0);
    else
      while (i > 0) {
v_.push_back(i % base);
i /= base;
      }
  }

  Integer::Integer(int i) : s_(i >= 0)
  {
    if (i < 0) i = -i;
    init(i);
  }

  Integer::Integer(unsigned int i) : s_(true)
  {
    init(i);
  }

  bool Integer::abs_less(const Integer& x) const
  {
    if (v_.size() < x.v_.size()) return true;
    if (v_.size() > x.v_.size()) return false;
    Crit i = v_.rbegin();
    Crit j = x.v_.rbegin();
    for (; i != v_.rend(); ++i, ++j) {
      if (*i < *j) return true;
      if (*i > *j) return false;
    }
    return false;
  }

  Integer Integer::operator-() const
  {
    Integer x = *this;
    x.s_ = !s_;
    return x;
  }

  Integer& Integer::operator+=(const Integer& x)
  {
    if (s_ == x.s_) return add(x);
    if (!abs_less(x)) return subtract_pos(x);
    Integer z = x;
    z.subtract_pos(*this);
    std::swap(z, *this);
    return *this;
  }
```

```
Integer& Integer::operator-=(const Integer& x)
{
  if (s_ != x.s_) return add(x);
  if (!abs_less(x)) return subtract_pos(x);
  Integer z = x;
  z.subtract_pos(*this);
  z.s_ = !z.s_;
  std::swap(z, *this);
  return *this;
}

Integer& Integer::operator*=(const Integer& x)
{
  Integer r = 0;
  r.s_ = (s_ && x.s_ || !s_ && !x.s_);
  for (Cit i = x.v_.begin(); i != x.v_.end(); ++i) {
    Integer z = *this;
    z.mult(*i);
    r.add(z);
    v_.push_front(0);
  }
  std::swap(r, *this);
  return *this;
}

Integer& Integer::operator/=(const Integer& x)
{
  assert(x != 0);
  Integer r;
  div(x, r);
  std::swap(*this, r);
  return *this;
}

Integer& Integer::operator%=(const Integer& x)
{
  assert(x > 0);
  Integer r;
  return div(x, r);
}

//
// Private Memberfunktionen der Klasse Integer
//

Integer& Integer::add(const Integer& x)
{
  while (v_.size() < x.v_.size()) v_.push_back(0);
  bool carry = false;
  It i = v_.begin();
  for (Cit j = x.v_.begin(); j != x.v_.end(); ++j, ++i) {
    *i += *j;
    if (carry) ++*i;
    carry = (*i >= base);
    if (carry) *i -= base;
  }
  if (!carry) return *this;
  for (; i != v_.end(); ++i)
```

```
      if (++*i == base) *i = 0; else return *this;
    v_.push_back(1);
    return *this;
  }

  Integer& Integer::subtract_pos(const Integer& x)
  {
    assert(!abs_less(x));
    It i = v_.begin();
    bool carry = false;
    for (Cit j = x.v_.begin(); j != x.v_.end(); ++j, ++i) {
      assert(i != v_.end());
      *i += base - *j;
      if (carry) --*i;
      carry = (*i < base);
      if (!carry) *i -= base;
    }
    if (carry) {
      assert(i != v_.end());
      while (*i == 0) {
*(i++) = base - 1;
assert(i != v_.end());
      }
      --*i;
    }
    while (v_.back() == 0)
      if (v_.size() == 1) {
s_ = true;
return *this;
      } else
v_.pop_back();
    return *this;
  }

  Integer& Integer::mult(unsigned int x)
  {
    if (x == 0) return *this = 0;
    assert(x < base);
    unsigned int carry = 0;
    for (It i = v_.begin(); i != v_.end(); ++i) {
      carry = *i * x + carry;
      *i = carry % base;
      carry /= base;
    }
    if (carry > 0) v_.push_back(carry);
    return *this;
  }

  Integer& Integer::div(const Integer& x, Integer& r)
  {
    if (abs_less(x)) {
      r = 0;
      return *this = x;
    }
    // Initialisiere Rest und berechne Vorzeichen
    r.s_ = (s_ && x.s_ || !s_ && !x.s_);
    r.v_.clear();
    s_ = true;
```

```
    // Align divisor
    Integer d = x;
    unsigned int sh = 1;
    while (d.v_.size() < v_.size()) {
      d.v_.push_front(0);
      ++sh;
    }
    if (abs_less(d)) {
      d.v_.pop_front();
      --sh;
    }

    do {
      // Binaersuche nach Multiplikator in [low,upp)
      unsigned int low = 0;
      unsigned int upp = base;
      do {
Integer dt = d;
unsigned int t = (low + upp) / 2;
dt.mult(t);
if (abs_less(dt)) upp = t; else low = t;
      } while (upp - low > 1);

      // subtract
      r.v_.push_front(low);
      Integer dt = d;
      dt.mult(low);
      subtract_pos(dt);

      // shift
      d.v_.pop_front();
    } while (--sh > 0);

    return *this;
  }

  //
  // Binaere Operatoren
  //

  bool operator==(const Integer& x, const Integer& y)
  {
    return x.s_ == y.s_ && x.v_ == y.v_;
  }

  bool operator!=(const Integer& x, const Integer& y)
  {
    return !(x == y);
  }

  bool operator<(const Integer& x, const Integer& y)
  {
    return x.s_ < y.s_ || x.s_ == y.s_ && x.abs_less(y);
  }

  bool operator<=(const Integer& x, const Integer& y)
  {
```

```
  return !(y < x);
}

bool operator>(const Integer& x, const Integer& y)
{
  return y < x;
}

bool operator>=(const Integer& x, const Integer& y)
{
  return !(x < y);
}

Integer operator+(const Integer& x, const Integer& y)
{
  Integer z = x;
  z += y;
  return z;
}

Integer operator-(const Integer& x, const Integer& y)
{
  Integer z = x;
  z -= y;
  return z;
}

Integer operator*(const Integer& x, const Integer& y)
{
  Integer z = x;
  z *= y;
  return z;
}

Integer operator/(const Integer& x, const Integer& y)
{
  Integer z = x;
  z /= y;
  return z;
}

Integer operator%(const Integer& x, const Integer& y)
{
  Integer z = x;
  z %= y;
  return z;
}

std::ostream& operator<<(std::ostream& o, const Integer& x)
{
  if (!x.s_) o << "-";
  for (Integer::Crit i = x.v_.rbegin(); i != x.v_.rend(); ++i)
    o << *i;
  return o;
}


Integer sqrt(const Integer& n)
```

```
  {
    // PRE: n >= 0.
    // POST: Gibt Approximation der Quadratwurzel von n zurueck.

    assert(n >= 0);

    // Binaersuche in [low,upp)
    Integer upp = 1;
    while (!n.abs_less(upp * upp)) upp *= 2;
    Integer low = upp / 2;
    while (upp - low > 1) {
      assert(low * low <= n);
      assert(upp * upp > n);
      Integer m = (low + upp) / 2;
      if (n.abs_less(m * m)) upp = m; else low = m;
    }
    return low;
  }

} // namespace ifet

int main()
{
  ifet::Integer x = 4294967295u;
  ifet::Integer y = 4294967295u;
  std::cout << "x = " << x << std::endl;
  std::cout << "y = " << y << std::endl;
  std::cout << "x + y = " << x + y << std::endl;
  std::cout << "x - y = " << x - y << std::endl;
  std::cout << "y - x = " << y - x << std::endl;
  std::cout << "x * y = " << x * y << std::endl;
  std::cout << "x / 3 = " << x / 3 << std::endl;
  std::cout << "3 / x = " << 3 / x << std::endl;
  return 0;
}
```