Feladat

Valósítsuk meg a nagyon nagyszámok típusát! Ábrázoljuk a számokat számjegyeik sorozatával és implementáljuk az összeadás és a szorzás műveleteit!

Természetes számok típusa

Típusspecifikáció

Típusérték-halmazunk $N = \mathbb{N}$, vagyis a természetes számok halmaza.

Összeadás

Két természetes szám összeadása, $z := x + y$

$$\begin{array}{rcccccc} A & = & N & \times & N & \times & N \\ & & x & & y & & z \\ A & = & N & \times & N & & \\ & & x' & & y' & & \\ Q & = & x' = x \wedge y' = y & & & & \\ R & = & Q \wedge z = x' + y' & & & & \end{array}$$

Szorzás

Két természetes szám összeszorzása, $z := xy$

$$\begin{array}{rcccccc} A & = & \mathbb{N} & \times & \mathbb{N} & \times & \mathbb{N} \\ & & x & & y & & z \\ A & = & \mathbb{N} & \times & \mathbb{N} & & \\ & & x' & & y' & & \\ Q & = & x' = x \wedge y' = y & & & & \\ R & = & Q \wedge z = x'y' & & & & \end{array}$$

Reprezentáció

A természetes számokat a 10-es számrendszerbeli számjegyeik sorozatával és a számjegyek számával reprezentáljuk. Speciális esetként a $0 \in \mathbb{N}$-t a nulla darab számjegyű számmal jelöljük.

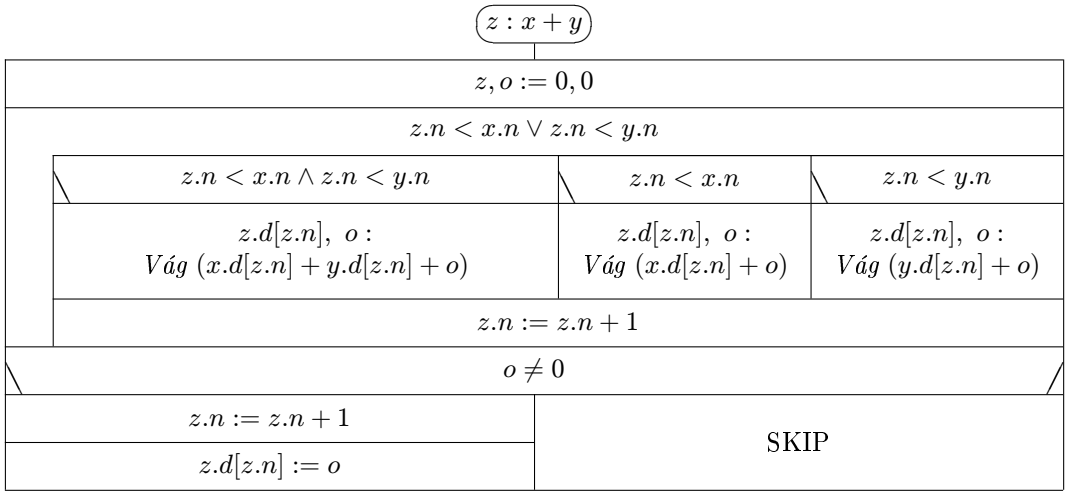
$$\begin{aligned} N &= (n : \mathbb{N},\ d : V_{\mathbb{N}}) \\ \forall x \in N : I(x) &= (x.n \leq x.d.dom) \wedge (\forall i \in \{1, \ldots, x.n\} : x.d_i \in \{0, \ldots, 9\}) \\ \forall x \in N : \rho(x) &= \sum_{i=1}^{x.n} x.d_i \cdot 10^{i-1} \end{aligned}$$

Megjegyzendő, hogy érdemes minél nagyobb számrendszert használni, amihez az adott architektúrának hardveres összeadó- ill. szorzórutinja van, mivel ílymódon a „szoftveres” lépések számát minimalizálhatjuk. Ennek ellenére az egyszerű parzolás és kiírás érdekében itt tizes számrendszert használunk.

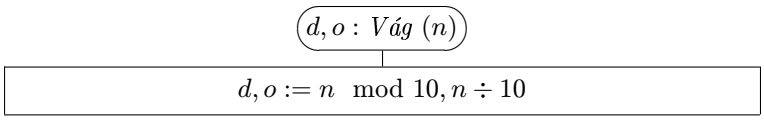
Absztrakt implementáció

Összeadás

Az összeadást a szokásos módon, helyiértékenként, az átvitelre figyelve végezzük el.

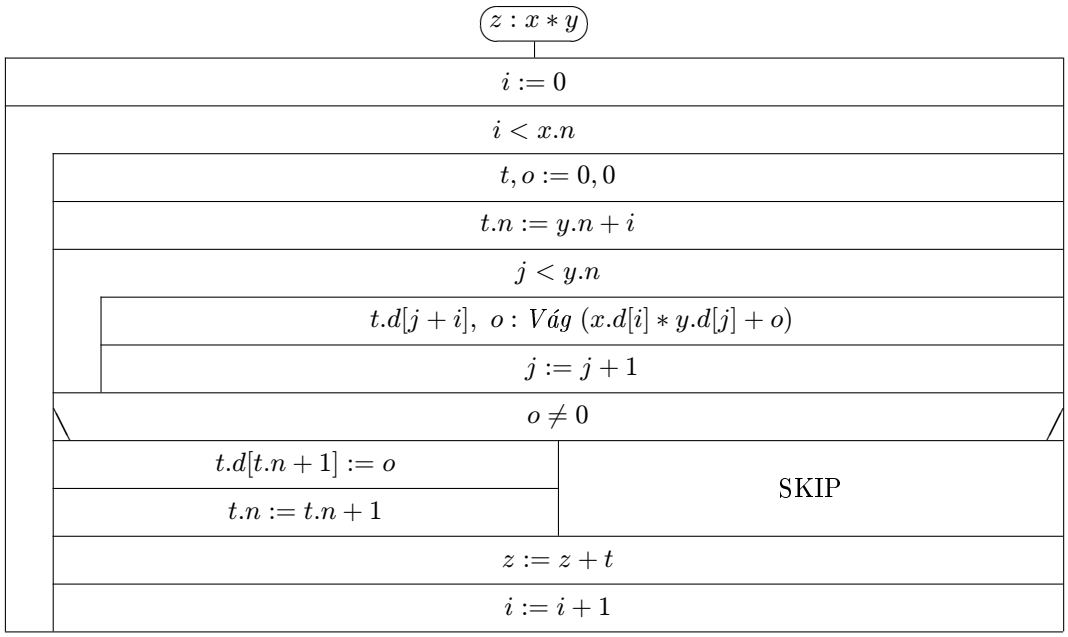


Egy természetes számból *(számjegy, átvitel)* párt az alábbi eljárással képzünk ($\div$ jelöli az egészosztást):



Szorzás

A szorzást szintén úgy végezzük, ahogy papíron szokás: az egyik tényező helyiértékei alapján számjegyenként elvégezzük a szorzást, majd összeadjuk az így kapott tagokat.



Fekete doboz-tesztelés

1. A 0, mint elfajult érték vizsgálata: $0 + 0 = 0$, $a + 0 = a$, $0 * 0 = 0$, $a * 0 = 0$
2. $1 * a = a$

C++ implementáció

A számjegyeket `unsigned char`-ok dinamikusan kezelt tömbjeként ábrázoljuk. A C++ memóriakezelési modelljéből következő eljárások:

Konstruktor, destruktor

A default konstruktor a 0 szám reprezentációját hozza létre:

```
Bignum::Bignum():
    n(0),
    d(0)
{
}
```

Egy másik, belső használatra szánt (privát) konstruktorban pedig helyet foglalunk n számjegynek, de továbbra is a 0-t reprezentáljuk. Ez hasznos lesz pl. összeadáskor az eredmény előzetes lefoglalására, mert a számjegyek számára egyszerűen tudunk felső becslést adni.

```
Bignum::Bignum(size_t n_):
    n(0),
    d(new digit_t[n_])
{
}

Bignum::~Bignum()
{
    delete[] d;
}
```

Értékadás, másolás

Másoló konstruktort és értékadó operátort is készítünk osztályunkhoz, természetesen mindkét művelet egyszerűen a számjegyek átmásolását jelenti.

```
Bignum::Bignum(const Bignum &that):
    n(that.n),
    d(new digit_t[n])
{
    for(size_t i = 0; i != n; ++i)
        d[i] = that.d[n];
}

Bignum& Bignum::operator=(const Bignum &that)
{
    if(this == &that) return *this;

    delete[] d;

    n = that.n;
    d = new digit_t[n];
    for(size_t i = 0; i != n; ++i)
        d[i] = that.d[i];

    return *this;
}
```

Ekvivalencia

Az ekvivalenciát is számjegyenként vizsgáljuk: két természetes szám egyenlő, ha számjegyeik egyenlőek.

```
bool operator== (const Bignum &x, const Bignum &y)
{
    if (&x == &y) return true;

    if (x.n != y.n) return false;
    for (size_t i = 0; i != x.n; ++i)
        if (x.d[i] != y.d[i]) return false;

    return true;
}
```

Összeadás, szorzás

A C++ implementáció egy része triviálisan adódik a fenti absztrakt programokból:

```
Bignum::digit_t Bignum::truncate_digit (digit_t a_plus_b, digit_t &overflow)
{
    overflow = a_plus_b / 10;
    return a_plus_b % 10;
}

Bignum operator+ (const Bignum &x, const Bignum &y)
{
    Bignum z (std::max(x.n, y.n) + 1);

    Bignum::digit_t overflow = 0;

    size_t i = 0;
    for (; i < x.n || i < y.n; ++i)
    {
        if (i < x.n && i < y.n)
            z.d[i] = Bignum::truncate_digit (x.d[i] + y.d[i] + overflow, overflow);
        else if (i < x.n)
            z.d[i] = Bignum::truncate_digit (x.d[i] + overflow, overflow);
        else if (i < y.n)
            z.d[i] = Bignum::truncate_digit (y.d[i] + overflow, overflow);
    }

    if (overflow)
        z.d[i++] = overflow;

    z.n = i;

    return z;
}

Bignum operator* (const Bignum &x, const Bignum &y)
{
    Bignum z;

    for (size_t i = 0; i != x.n; ++i)
    {
        Bignum tmp (y.n + i + 1);
        tmp.n = y.n + i;
```

```
        Bignum::digit_t overflow = 0;

        for (size_t j = 0; j != y.n; ++j)
            tmp.d[j + i] = Bignum::truncate_digit (x.d[i] * y.d[j] + overflow, overflow);

        if (overflow)
            tmp.d[tmp.n++] = overflow;

        z += tmp;
    }

    return z;
}
```

Hozzáadás, beszorzás

A `+=` és a `*=` operátorokat a kényelmesebb használat érdekében definiáljuk, természetesen a fent definiált `+` és `*` operátorok segítségével:

```
Bignum Bignum::operator+= (const Bignum &y)
{
    *this = *this + y;
    return *this;
}

Bignum Bignum::operator*= (const Bignum &y)
{
    *this = *this * y;
    return *this;
}
```

Sztringből/sztringgé alakítás

Az egyszerű kezelhetőség kedvéért lehetővé tesszük sztringek számmá alakítását, és viszont. Ha ezt nem tennénk, akkor nemcsak felhasználói inputból, de programatikusan is bajosan tudnánk konstansokat előállítani: szükség lenne a `++` rákövetkező-operátorra, és így az egyetlen, 0 konstans elegendő lenne.

```
Bignum::Bignum (const std::string &s):
    n(0),
    d(new digit_t[s.size()])
{
    int k;
    for (k = 0; k != s.size () && s[k] == '0'; ++k);

    n = s.size () - k;
    for (unsigned int i = 0; i != s.size () - k; ++i)
        d[n - (i + 1)] = digit_from_char (s[k + i]);
}

Bignum::digit_t Bignum::digit_from_char (char c)
{
    if (!('0' <= c && c <= '9')) throw std::bad_cast();
    return c - '0';
}
```

```
std::string Bignum::str () const
{
    if (n == 0)
        return "0";

    std::string s = "";
    s.reserve (n);

    for (unsigned int i = n; i; --i)
    {
        s += '0' + d[i - 1];
    }

    return s;
}
```

Beolvasás/kiírás

A fenti átalakításokat felhasználva már könnyedén olvashatunk és írhatunk stream-ekbe:

```
std::ostream& operator<< (std::ostream &ostr, const Bignum &bignum)
{
    ostr << bignum.str ();
    return ostr;
}

std::istream& operator>> (std::istream &istr, Bignum &bignum)
{
    std::string str;
    istr >> str;
    if (istr)
        try
        {
            bignum = Bignum (str);
        }
        catch (std::bad_cast &e)
        {
            istr.setstate (std::istream::failbit);
        }
    return istr;
}
```

Tesztelési terv

1. (l. absztrakt implementációs rész)
2. Extrém nagy számok bevitele és számolás velük
3. Sztringek átalakítása számmá
4. Hibás sztringek átalakítása