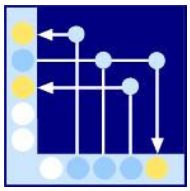
Fakultät Elektronik und Informatik
Studiengang Informatik



Programmieren in C++

Vorlesung im Wintersemester 2017/2018
Prof. Dr. habil. Christian Heinlein

3. Übungsblatt (9. November 2017)

Aufgabe 7: Konstruktoren, Destruktoren und Zuweisungen

Teilaufgabe 7.a)

„Verpacken“ Sie den Typ `bigint` aus Aufgabe 2 in eine Klasse/Struktur mit einem oder mehreren Konstruktoren, Destruktor, Kopierkonstruktor und -zuweisung sowie Verschiebekonstruktor und -zuweisung!

Definieren Sie zugehörige globale Funktionen für Addition und Multiplikation!

Teilaufgabe 7.b)

Ergänzen Sie die Funktionen der Klasse um sinnvolle Ausgabeanweisungen, damit Sie ihre größtenteils implizit erfolgenden Aufrufe verfolgen können!

Schreiben Sie ein Testprogramm, in dem Sie Objekte der Klasse als globale und lokale Variablen verwenden, per Wert oder per Referenz an Funktionen übergeben, als Resultat von Funktionen zurückliefern, dynamisch mittels `new` und `delete` erzeugen und löschen usw.!

Was ändert sich, wenn Verschiebekonstruktor und -zuweisung entfernt werden?

✎

```
// Lösung von Aufgabe 2 einbinden.
// Damit die dort definierte Funktion main nicht mit
// der hier definierten kollidiert, wird sie umbenannt.
#define main bigint_main
#include "bigint.cxx"
#undef main

// Dynamische Kopie der langen Zahl x liefern.
bigint bi_dup (bigint x) {
    int n = bi_size(x);
    bigint z = bi_new(n);
    for (int i = 1; i <= n; i++) z[i] = x[i];
    return z;
}
```

```
// op, obj und val ausgeben, sofern Makro TRACE definiert ist.
void trace (string op, void* obj, bigint val) {
    #ifdef TRACE
    cout << op << " " << obj << " ";
    bi_print(val);
    #endif
}

// Klasse für lange vorzeichenlose ganze Zahlen.
struct BigInt {
    // Zeiger auf ein dynamisches Feld von unsigned-char-Werten.
    bigint bi;

    // Normaler Konstruktor.
    BigInt (unsigned char x) : bi(bi_make(x)) {
        trace("construct value", this, bi);
    }

    // Konstruktor für Addition und Multiplikation.
    explicit BigInt (bigint bi) : bi(bi) {
        trace("construct result", this, bi);
    }

    // Kopierkonstruktor.
    BigInt (const BigInt& that) : bi(bi_dup(that.bi)) {
        trace("copy construct", this, bi);
    }

    // Kopierzuweisung.
    BigInt& operator= (const BigInt& that) {
        trace("copy assign", this, that.bi);
        bigint old = bi;
        bi = bi_dup(that.bi);
        bi_del(old);
        return *this;
    }

    #ifdef MOVE

    // Verschiebekonstruktor.
    BigInt (BigInt&& that) : bi(that.bi) {
        trace("move construct", this, bi);
        that.bi = bi_make(0);
    }

    // Verschiebezuweisung.
    BigInt& operator= (BigInt&& that) {
        trace("move assign", this, that.bi);
        bigint tmp = bi; bi = that.bi; that.bi = tmp;
        return *this;
    }

    #endif
```

```
    // Destruktor.
    ~BigInt () {
        trace("destruct", this, bi);
        bi_del(bi);
    }
};

// Wenn Makro REF definiert ist, ist der im folgenden verwendete
// Parametertyp Param ein Alias für const BigInt&, andernfalls
// ein Alias für BigInt.
#ifdef REF
using Param = const BigInt&;
#else
using Param = BigInt;
#endif

// Lange Zahlen x und y addieren.
BigInt add (Param x, Param y) {
    return BigInt(bi_add(x.bi, y.bi));
}

// Lange Zahlen x und y multiplizieren.
BigInt mul (Param x, Param y) {
    return BigInt(bi_mul(x.bi, y.bi));
}

// Lange Zahl x um 1 erhöhen.
BigInt inc (Param x) {
    BigInt z = add(x, 1);
    return z;
}

// Testprogramm.
int main () {
    BigInt x = 1;
    for (int i = 2; i <= 5; i++) {
        x = mul(x, i);
    }

    x = add(mul(x, 2), 3);
    BigInt y = inc(x);
}
```