Dirk Praetorius,
Marcus Page,
Thomas Führer,
Josef Kemetmüller

Sommersemester 2013
04.06.2013

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 11

Aufgabe 11.1. Schreiben Sie eine Klasse `MyBigInt` zur Speicherung beliebig großer (und kleiner) Ganzzahlen $z \in \mathbb{Z}$. Diese soll die Dezimalziffern einer Zahl z als `vector<int>` speichern können. Das Vorzeichen soll in einer eigenen Member-Variablen gespeichert werden. (Der `vector` soll also für -12 und 12 die gleichen Ziffern speichern.) Schreiben Sie weiters einen Default-Konstruktor, der die Null erzeugt und einen `operator-` der zu einer Zahl z die Zahl $-z$ als Kopie zurückliefert. Orientieren Sie sich am Beispiel der Komplexen Zahlen aus der Vorlesung.
Speichern Sie den Source-Code unter `MyBigInt{.h, .cpp}` in das Verzeichnis `serie11`.
Warum muss man hier den Zuweisungsoperator nicht explizit definieren?

Aufgabe 11.2. Schreiben Sie einen Konstruktor `MyBigInt(int i)` dieser Klasse, der einen `int` übernimmt und daraus ein Objekt der `MyBigInt`-Klasse generiert. Hier müssen Sie geschickt dividieren und modulo rechnen.

Aufgabe 11.3. Schreiben Sie einen Konstruktor `MyBigInt(string s)` dieser Klasse, welche einen String übernimmt, der die Ziffern der Zahl enthält. Beachten Sie, dass die Ziffern in einem String in umgekehrter Richtung gespeichert sind. Hinweis: Falls Sie folgenden Codeausschnitt verwenden, dann können Sie auch mittels

```
MyBigInt num;
std::cin >> num;
```

Zahlen einlesen, wie bisher mit anderen Datentypen.

```
std::istream& operator>>(std::istream& input, MyBigInt& num) {
        std::string s;
        input >> s;
        num = MyBigInt(s);
        return input;
}
```

Aufgabe 11.4. Wir wollen natürlich auch eine Funktionalität zur Ausgabe von `MyBigInt`-Objekten erhalten. Der Standardweg in `C++` dafür ist das Überladen des Operators

```
operator<<
```

wie sie ihn bereits aus

```
std::cout << i;
```

kennen.

```
class MyBigInt {
   ...
   friend std::ostream& operator<<(std::ostream&, const MyBigInt&);
   ...
};
```

Das Schlüsselwort `friend` bedeutet hierbei, dass die Funktion `operator«` keine Klassenmethode darstellt, man sie also nicht per `a.operator«(b,c)` aufrufen kann - was sinnlos wäre - aber sie dennoch Zugriff auf die privaten Teile der Klasse hat. Dadurch müssen Sie nicht auf öffentliche Zugriffsfunktionen

zurückgreifen. Eine Beispielimplementierung des `operator<<` sehen Sie weiter unten. Diese Implementierung gibt immer `532` am Bildschirm aus. Ersetzen Sie die drei markierten Zeilen mit einer Schleife, welche die Einträge von `num` ausgibt.

```
std::ostream& operator<<(std::ostream& os, const MyBigInt& num) {
    os << 5; // Loeschen Sie diese Zeile
    os << 3; // Loeschen Sie diese Zeile
    os << 2; // Loeschen Sie diese Zeile
    return os;
}
```

Das `return os;` benoetigt man dabei um mehrere Ausgaben zu verketten:

```
std::cout << bignum1 << bignum2;
```

Aufgabe 11.5. Implementieren Sie die folgenden Operatoren für die Klasse `MyBigInt`:

```
bool operator == (const MyBigInt& num1, const MyBigInt& num2);
bool operator < (const MyBigInt& num1, const MyBigInt& num2);
bool operator > (const MyBigInt& num1, const MyBigInt& num2);
```

Tipp: Definieren Sie die Funktionen wie in Beispiel 11.4 als `friend` der Klasse `MyBigInt`, damit sie direkten Zugriff auf den Ziffernvektor erhalten.

Aufgabe 11.6. Implementieren Sie die Operatoren `operator+` und `operator-` für die Klasse `MyBigInt`. Verwenden Sie dazu das bekannte Verfahren aus der Schule, bei dem Sie die Ziffern einzeln addieren/-subtrahieren und gegebenenfalls einen Übertrag zur nächsten Ziffer mitnehmen. Beachten Sie, dass ein `MyBigInt` auch eine negative Zahl darstellen kann und dass das Ergebnis unterschiedliche Vorzeichen haben kann. Vermeiden Sie möglichst doppelten Code. (Sie können beispielsweise zur Addition $a + b$ einer positiven $a \geq 0$ und einer negativen Zahl $b \leq 0$ stattdessen die Subtraktion $a - (-b)$ aufrufen. Alle Fälle lassen sich auf diese Weise reduzieren auf entweder: $a + b$ mit $a \geq b \geq 0$ oder $a - b$ mit $a \geq b \geq 0$.) Testen Sie Ihren Code für alle Fälle von verschiedenen Vorzeichen und Operatoren, die hier auftreten können. ({Vorzeichen linker Operand$\pm$} $\times$ {`operator`$\pm$} $\times$ {Vorzeichen rechter Operand$\pm$} $\times$ {Vorzeichen Ergebnis$\pm$}) Tipp: Definieren Sie die beiden Funktionen wie in Beispiel 11.4 als `friend` der Klasse `MyBigInt`, damit sie direkten Zugriff auf den Ziffernvektor erhalten.

Aufgabe 11.7. Definieren Sie mit dem `operator*` eine Multiplikation für die Klasse `MyBigInt`. Wenn Sie hier mit dem Verfahren, das aus der Schule bekannt ist arbeiten, dann müssen Sie zuerst eine Multiplikation mit einer einzelnen Ziffer $\{0, \ldots, 9\}$, sowie eine Multiplikation mit 10 implementieren und können dann die Addition verwenden. Tipp: Definieren Sie die Funktionen wie in Beispiel 11.4 als `friend` der Klasse `MyBigInt`, damit sie direkten Zugriff auf den Ziffernvektor erhalten.

Aufgabe 11.8. Was ist eigentlich `666!` ?