

Übung 7

Klassen & Operatoren

A 7.1: Ein Taschenrechner mit Stack (Teil 2.)

(6)

Die Größe des *Stack*-Arrays vom Teil 1. ist nun auf 100 zu setzen.

- (a) Erweitern Sie den *Stack* so, daß er zwei verschiedene Typen von Elementen speichern kann: Integer-Zahlen und Operationen auf diesen. Gehen Sie dazu wie folgt vor:
- Beschränken Sie die zulässigen Zahlen für den *Stack* auf das Intervall **[-255, 255]** und geben Sie eine Fehlermeldung aus, wenn versucht wird, eine Zahl kleiner als **-255** oder größer als **259** (siehe nächster Punkt !) auf den *Stack* zu legen.
 - Der Elementtyp „Rechenoperation“ wird ebenfalls durch Integer-Zahlen repräsentiert. Definieren Sie dazu vier Konstanten **PLUS = 256**, **MINUS = 257**; **MAL = 258** und **DURCH = 259**. Diese Konstanten sollten auch außerhalb der Klasse sichtbar sein!
 - Erweitern Sie die Methode **gebeAus ()** so, daß sie für *Stack*-Elemente größer als **255** die entsprechenden Symbole (also z.B. '+' für **256**) ausgibt.

Erzeugen Sie ein Objekt vom Typ *Stack* und führen Sie die folgenden Operationen aus:
push(3); push(4); push(Stack::PLUS); push(5); push(Stack::MAL);
gebeAus ();

- (b) Erweitern Sie den *Stack* so, daß die Rechenoperationen auf dem *Stack* durchgeführt werden können. Erzeugen Sie ein Objekt vom Typ *Stack* und berechnen Sie mit den obigen Methoden das Ergebnis der folgenden Ausdrücke und geben Sie den *Stack*inhalt vor und nach jeder Rechenoperation aus.

(*, +, 1, 3, /, 6, 2) (+, /, 6, 3, *, 25, 10) (-, 1, +, 3, *, 6, 8)

Die Regeln sind dabei wie folgt:

- Zu Beginn müssen mindestens zwei Zahlen und eine Rechenoperation auf dem *Stack* liegen (in dieser Reihenfolge, d.h. die Rechenoperation "ganz oben")
- Eine Rechenoperation wirkt immer auf die beiden nächsten Elemente des *Stacks*, z.B. würde die Berechnung des *Stacks* **(+, 3, 4)** das Ergebnis **7** produzieren. Ist eines der beiden Elemente selber eine Rechenoperation, muß diese rekursiv berechnet werden, so ergibt z.B. **(*, 5, +, 3, 4)** das Ergebnis **5 * (3 + 4) = 35**.
- Ist eine Rechenoperation beendet, wird das Ergebnis auf den *Stack* gelegt.
- Beachten Sie, daß Zahlen im Intervall **[256, 259]** als Rechenoperation interpretiert werden, d.h. Ergebnisse in diesem Intervall müssen als Überlauf abgefangen werden !

A 7.2: Klasse "Vektor"

(12)

Deklariieren und definieren Sie eine C++ - Klasse *Vektor*, die einen 3D-Vektor und Operationen auf diesem kapselt. Gehen Sie dazu wie folgt vor:

- (a) Deklariieren und definieren Sie eine Klasse *Vektor* mit den Datenelementen **double x, y, z**, sowie dem Standard-, dem Initialisierungs- und dem Kopierkonstruktor.
- (b) Erweitern Sie die Klasse *Vektor* um die Operationen
void set(Vektor &v) und bool isEqual(Vektor &v).

Die erste Methode soll die Datenelemente des Vektors auf die des gegebenen Vektors setzen, die zweite soll TRUE zurückliefern, wenn der Vektor dem gegebenen gleicht.

(c) Erweitern Sie die Klasse `Vektor` um folgende Operationen:

- `add` Addiert den gegebenen Vektor zu diesem Vektor
- `scalMul` gibt das Skalarprodukt (double) dieses Vektor mit dem geg. Vektor zurück:

$$s = v1.x * v2.x + v1.y * v2.y + v1.z * v2.z$$
- `crossMul` bildet das Kreuzprodukt zwischen diesem und dem geg. Vektor und gibt das Ergebnis (einen Vektor) zurück.

$$c = \begin{pmatrix} v1.y * v2.z - v1.z * v2.y \\ v1.z * v2.x - v1.x * v2.z \\ v1.x * v2.y - v1.y * v2.x \end{pmatrix}$$
- `normalize` Normalisiert diesen Vektor, d.h. teilt alle Datenelemente (x, y, und z) durch die "Länge" des Vektors, $l = (x^2 + y^2 + z^2)^{1/2}$.

(d) Überladen Sie die Operatoren `=`, `+`, `*`. Die Ergebnisse der Operationen `+` und `add` als auch `*` und `crossMul` sollten übereinstimmen.

Testen Sie die Klasse mit dem folgenden Code-Fragment:

```
Vektor v1 = Vektor(17.4, 0.0, 0.0);
Vektor v2 = Vektor( 0.0, 4.8, 0.0);
Vektor v3 = v1.crossMul(v2);
Vektor v4 = v1;
Vektor v5 = v1+v2;
Vektor v6 = v1+v2*v3;

v3.normalize();
cout << v1.scalMul(v3) << endl << v2.scalMul(v3) << endl;
v3.set(v1);
if (v1.isEqual(v2))
    cout << "v1 == v2" << endl;
if (v1.isEqual(v3))
    cout << "v1 == v3" << endl;
```