ulm university universität
uulm

Institut für Numerische Mathematik
Dr. Andreas F. Borchert und Dr. Michael C. Lehn

17. November 2016
Blatt 4

Objektorientierte Programmierung mit C++ (WS 2016/2017)

Abgabe bis zum 24. November 2016, 16:00 Uhr

Lernziele:

- Erstellung einer Klasse mit einer dynamischen Datenstruktur

Aufgabe 5: Taschenrechner mit Postfix-Notation

Die Postfix-Notation oder auch Umgekehrte Polnische Notation (UPN) ist eine Schreibweise, bei der zuerst die Operanden alle genannt werden, bevor der Operator folgt. In der Informatik ist UPN deshalb von Interesse, weil diese Schreibweise eine einfache stapelbasierte Verarbeitung ermöglicht.

Hier sind einige Beispiele, um diese Notation zu veranschaulichen:

Ausdruck in UPN	Ergebnis	Infix-Notation
3 4 −	-1	$3-4$
3 4 + 5 ∗	35	$(3+4)*5$
1 2 + 3 4 + ∗	21	$(1+2)*(3+4)$
23 2 ∗ 4 −	42	$23*2-4$
1 2 3 4 5 + − ∗ /	-0.0833	$\frac{1}{2*(3-(4+5))}$

Bei der UPN kann die Eingabe als Sequenz von durch Leerzeichen getrennten Token betrachtet werden, wobei sowohl numerische als auch symbolische Token vorkommen. Beispiele für numerische Token sind „3“ oder „4.7“, als symbolische Token werden die Grundrechenarten „$+$“, „$-$“, „$*$“ und „$/$“ unterstützt. Hierbei betrachten wir nur die binären Operatoren. Das Zeilenende wird als Ende der Sequenz interpretiert.

Die Berechnung eines Ausdrucks in der UPN-Notation ist recht einfach, da hierfür nur ein Stack für numerische Werte (etwa **double**) benötigt wird. Dann werden die Tokens einer Zeile sukzessive verarbeitet:

- Numerische Token werden in eine **double** konvertiert und auf den Stack geladen.

- Bei Operatoren werden vom Stack die benötigten Operanden heruntergenommen, das Ergebnis berechnet und wieder auf den Stack befördert. Im Rahmen dieser Aufgabe betrachten wir nur binäre Operatoren, die immer genau zwei Operanden erwarten.

Wenn die gesamte Eingabezeile verarbeitet ist, sollten alle auf dem Stack verbliebenen Werte nacheinander ausgegeben werden. Wenn es zu Fehlern kommt wie einem unbekannten Operator oder fehlenden Operatoren (*stack underflow*), dann sollte nur eine entsprechende Fehlermeldung ausgegeben werden.

Hier ist ein Ausführungsbeispiel:

```
thales$ Calc
UPN: 1 2 3 4 5 + - * /
-0.0833333
UPN: 1 2
2
1
UPN: +
stack underflow
UPN: 1 2 ?
unknown operator: ?
UPN: thales$
```

Bei der Umsetzung der Aufgabe ist darauf zu achten, dass eine Klasse *Stack* für Werte des Typs **double** entwickelt wird, die möglichst allgemeingültig ausformuliert sein soll und nicht auf entsprechenden Klassen der STL beruhen darf. Sie muss somit eine selbst entwickelte dynamische Datenstruktur für lineare Listen enthalten. Da die Klasse dann eine entsprechende Ressource gemäß dem RAII-Prinzip verwaltet, sind ein Kopier-Konstruktor, ein Verschiebe-Konstruktor (*move constructor*), ein Zuweisungs-Operator und ein Dekonstruktor zu implementieren. Hierbei empfiehlt sich die in der Vorlesung vorgestellte *Copy-and-Swap*-Vorgehensweise. Testen Sie Ihre *Stack*-Klasse mit einem Testprogramm und lassen Sie dies unter der Kontrolle von *valgrind* laufen, um zu überprüfen, ob Ihre Implementierung frei von Speicherlecks ist und keine mehrfachen Freigaben vorkommen. Das Werkzeug *valgrind* steht auf unseren Linux-Maschinen zur Verfügung – leider wird Solaris von *valgrind* nicht unterstützt, so dass es auf der Thales nicht vorhanden ist.

Die Eingabe müssen Sie lexikalisch analysieren, wobei hier Operanden (alle mit einer Ziffer beginnend) und Operatoren zu unterscheiden sind. Wenn Sie das alles direkt von der Standardeingabe einlessen, müssen Sie darauf achten, den Zeilentrenner nicht zu übersehen, da damit ein UPN-Ausdruck abgeschlossen wird.

Die zeilenweise Vorgehensweise lässt sich vereinfachen, indem zunächst mit std::getline eine Zeile in ein *std*::*string*-Objekt eingelesen wird. Mit Hilfe von std::istringstream können Sie ein *std*::*string*-Objekt für die Eingabe innerhalb einer Zeile verwenden. Wenn Sie das nächste Token von der Eingabe einlesen möchten, dann sind zunächst beliebig viele Leerzeichen zu überspringen. Wenn eine Ziffer folgt, sollte ein **double**-Wert eingelesen werden, ansonsten wiederum ein *std*::*string*. Wenn Sie ein *std*::*string*-Objekt mit Hilfe des !»!-Operators einlesen, dann werden zuerst Leerzeichen überlesen und danach das erste Leerzeichen als Trenner betrachtet. Für die Vorausschau des nächsten Zeichens in der Eingabe bietet sich die *peek*-Methode an.

Verpacken Sie Ihre Quellen in ein tar-Archiv und reichen Sie dieses wieder mit *submit* ein:

```
thales$ tar cvf upn.tar *.?pp Makefile
thales$ submit cpp 5 upn.tar
```

Viel Erfolg!