Dr. Andreas Borchert
Norbert Heidenbluth
Abteilung Angewandte Informationsverarbeitung
28.04.2005
Blatt 3

# Allgemeine Informatik II für Mathematiker/Wirtschaftsmathematiker (SS 2005)

**Abgabetermin: 12. Mai 2005**

## Aufgabe 5: Ein UPN-Taschenrechner (20 Punkte)

Nun studieren Sie schon im zweiten Semester entweder Mathematik oder zumindest Fächer, die sehr mathematik-lastig sind. Und dabei wächst von Tag zu Tag Ihre Unzufriedenheit über die modernen Taschenrechner, denn um z.B. $(3 + 4) \cdot 5$ mit dem Rechner berechnen zu lassen[1], müssen Sie eine Möglichkeit der Eingabe finden, welche der "Punkt-vor-Strich"-Regel gerecht wird. Und das finden Sie nun doch irgendwie lästig.

Zufällig stolpern Sie über den Begriff der sogenannten *Umgekehrte Polnische Notation (UPN)*, die Ihnen sofort als Lösung aller Probleme ins Auge sticht (Aua!). Der obige Ausdruck in polnischer Notation sieht wie folgt aus: `3 4 + 5 *`

"Aha, interessant!", denken Sie sich, "aber was hat das nun zu bedeuten?". Nun, in der UPN folgt jeder Operator seinem/n Operanden, und der/die Operand(en) werden zunächst auf einen Stack gelegt. Folgt nun der Operator, so wird/werden sein(e) zugehöriger/en Operand(en) vom Stack geholt, ausgewertet (berechnet) und das Ergebnis seinerseits wieder auf den Stack gelegt (als Operand für den nächsten Operator). Das heißt, die Operanden und Operatoren aus der Eingabe werden von links nach rechts bearbeitet:

- Wenn ein *Operand* kommt, so wird dieser auf den Stack gelegt. Gleiches gilt für Konstanten, die ja auch Operanden sind.
- Kommt hingegen ein *Operator*, so nimmt sich dieser seine(n) benötigte(n) Operanden vom Stack, berechnet das Ergebnis der Operation und legt dies auf den Stack.

Alles klar? Nein? Dann schauen wir uns die Sache mit den Operatoren, Operanden und dem Stack mal anhand zweier Beispiele an. Mit *Token* ist im Folgenden das jeweils aktuell gelesene Symbol gemeint (also entweder ein Operator oder ein Operand).

[1] Wobei: Wenn Sie das wirklich mit dem Taschenrechner berechnen müssen, dann sollten Sie sich vielleicht lieber einen anderen Studiengang suchen...!

**Beispiel:** `3 4 + 5 *`

| Token | Inhalt des Stacks | Erklärung |
|---:|---:|---|
| 3 | 3 | |
| 4 | 4 3 | |
| + | — | 4 und 3 werden vom Stack geholt (+ ist binärer Operator) und addiert |
| | 7 | Ergebnis der Berechnung $3 + 4$ kommt auf den Stack |
| 5 | 5 7 | |
| ∗ | — | 5 und 7 werden vom Stack geholt (binäre Operation) und multipliziert |
| | 35 | Ergebnis der Berechnung $5 \cdot 7$ |

**Beispiel:** `4 5 * 5 + sqrt pi +`

| Token | Inhalt des Stacks | Erklärung |
|---:|---:|---|
| 4 | 4 | |
| 5 | 5 4 | |
| ∗ | — | 5 und 4 zur Multiplikation vom Stack holen (binäre Operation) |
| | 20 | Ergebnis der Multiplikation auf den Stack legen |
| 5 | 5 20 | |
| + | — | 5 und 20 zur Addition vom Stack holen (binäre Operation) |
| | 25 | Ergebnis der Addition auf den Stack legen |
| `sqrt` | — | `sqrt` ist unitär: nur einen Operanden vom Stack holen (25) |
| | 5 | Ergebnis der Berechnung, denn $\sqrt{25} = 5$ |
| pi | 3.14159 5 | $\pi$ ist eine Konstante, auf den Stack legen |
| + | — | 3.14159 und 5 zur Addition vom Stack holen |
| | 8.14159 | Ergebnis der Addition auf den Stack legen |

"Ja, Wahnsinn!", denken Sie, "so einen Taschenrechner muß ich haben!". Und schon greifen Sie zu Maus und Tastatur und wagen sich an folgende

### Aufgabe:

Implementieren Sie in Oberon einen Taschenrechner, der (nacheinander) beliebig viele Eingabezeilen in der Umgekehrten Polnischen Notation entgegennimmt, diese jeweils nach der vorstehend beschriebenen Methode auswertet, berechnet und am Ende jeder Eingabezeile den Inhalt des Stacks ausgibt.

### Jede Menge Hinweise:

- Zur Implementierung einiger Rechenfunktionen (z.B. sin, cos) können Sie die entsprechenden Funktionen im Modul `Math` verwenden. Siehe dazu auch *man Math*.
- Aus diesem Grund ist es ratsam, Zahlen im Datentyp `REAL` abzulegen und *nicht* als `INTEGER`.

- Gegen Ende des letzten Semesters haben wir die Prozedurtypen kennengelernt. Diese könnten Ihnen (spätestens in Etappe 4, s.u.) nützlich werden.

- Wie immer sollte Ihr Programm robust gegen falsche Eingaben sein, hier insbesondere gegen ungültige Operatoren und zu wenig (auf dem Stack verbleibende) Operanden.

## Empfohlenes Vorgehen

Um diese Aufgabe zu bearbeiten, empfehlen wir die Aufteilung des Programms in die folgenden "Etappen":

**Etappe 1: Einlesen (5 Punkte)** Bevor Sie sich an die Implementierung der eigentlichen Rechnerfunktion machen, ist es nötig, das korrekte Einlesen der Zeile zu gewährleisten. Zu Unterscheiden sind hier Operanden von Operatoren, so daß diese entsprechend "gekennzeichnet" werden müssen. (Tip: Record!)

Weitere Tips hierzu erhalten Sie in den Übungen!

Für die Grammatik-Fans unter Ihnen kann man die Regeln für eine gültige Eingabe auch gut in Form einer Grammatik angeben:

$$
\begin{array}{rcl}
\langle\text{UPNExpressions}\rangle & \longrightarrow & \epsilon \mid \langle\text{UPNExpression}\rangle\langle\text{UPNExpressions}\rangle \\
\langle\text{UPNExpression}\rangle & \longrightarrow & \langle\text{Tokenlist}\rangle\text{LineTerminator} \\
\langle\text{TokenList}\rangle & \longrightarrow & \epsilon \mid \langle\text{Token}\rangle\langle\text{TokenList}\rangle \\
\langle\text{Token}\rangle & \longrightarrow & \langle\text{Operator}\rangle \mid \langle\text{Operand}\rangle \\
\langle\text{Operand}\rangle & \longrightarrow & \text{(* Gleitkommazahl, die von Read.Real akzeptiert wird *)} \\
\langle\text{Operator}\rangle & \longrightarrow & \text{“+”} \mid \text{“-”} \mid \text{“*”} \mid \text{“/”usw.}
\end{array}
$$

**Etappe 2: Der Stack (5 Punkte)** Das Herzstück des Taschenrechners wird die Datenstruktur des sogenannten *Stacks* sein, welcher im Skript zur Vorlesung wie folgt definiert wird:

> "Ein Stapel (oder Stack) ist eine lineare Liste, die sich dadurch auszeichnet, daß nur an einem der beiden Enden der Liste Elemente hinzugefügt und wieder weggenommen werden."

Deshalb sollten Sie zu Beginn des Programms diese Datenstruktur sowie die zugehörigen Prozeduren (`Push()`, `Pop()` und ggf. `InitStack()`) implementieren.

**Etappe 3: Binäre Operationen (5 Punkte)** Jetzt haben Sie das Rüstzeug zusammen, um Ihrem Programm das Rechnen beizubringen. Dazu genügt zunächst die Implementierung der binären Operationen $+$, $-$, $\cdot$ und $/$. (Diese Operationen heißen deshalb *binär*, weil immer genau zwei Operanden für diese Operationen gebraucht werden. *Unitäre* Operationen brauchen hingegen genau einen Operanden!)

Wie weiter oben bereits beschrieben, sollte Ihr Programm die Operanden und den Operator vom Stack holen, die Berechnung durchführen und das Ergebnis dann auf den Stack legen.

**Etappe 4: (5 Punkte)** Festhalten — Jetzt geht es nochmals richtig rund! Denn nun soll Ihr Programm so "umgestrickt" werden, daß außer den binären Operationen (aus Etappe 3) nun auch unitäre Operationen (z.B. sin, cos, $\sqrt{\ldots}$ usw.) und Konstanten ($\pi$, $e$) unterstützt werden. Somit wird zum Beispiel auch die Rechnung `2 e ln *` möglich, die nichts anderes bedeutet als: $2 \cdot \ln e (= 2)$.

Hierzu empfiehlt es sich, auch alle Operatoren in Form einer linearen Liste abzulegen. Ein Beispiel dafür, wie man eine lineare Liste durchsuchen kann, erhalten Sie in den Übungen!

An dieser Stelle kommen auch die in den Hinweisen bereits erwähnten Prozedurtypen ins Spiel, denn mit ihrer Hilfe ist es möglich, die eigentliche Rechenfunktionalität unabhängig von der konkret zu verwendenden Prozedur zu halten: Sie holen sich die zu verwendende Rechenprozedur aus der Operatoren-Liste heraus und reichen sie an die Stelle in Ihrem Programm weiter, an der die Berechnung durchgeführt werden soll.

**Viel Erfolg!**