FAKULTÄT FÜR INFORMATIK
Faculty of Informatics

Programmiersprachen
LVA 185.208, VU, 3 ECTS, 2013 S

1. Übungsaufgabe

Aufgabe:
Entwickeln Sie in einer beliebigen Programmiersprache Ihrer Wahl eine Simulation eines programmierbaren Taschenrechners entsprechend folgenden Spezifikationen und lösen Sie damit die weiter unten beschriebenen Testaufgaben. Die Testaufgaben 1 und 2 sind unbedingt durchzuführen.

Eingaben in den Taschenrechner:
Eingaben erfolgen in Postfix-Notation (zuerst kommen die Argumente, danach der Operator). Die Anzahl der benötigten Argumente hängt vom Operator ab. Argumente sind ganze Zahlen (oder Ausdrücke in Klammern, siehe unten). Zum Beispiel ist “1 2+” eine Anwendung des Operators “+” auf 1 und 2, die als Ergebnis 3 liefert. Der Ausdruck “1 2 3 4+*-” wird zu = -13 ausgewertet: Der erste Operator addiert die direkt davor stehenden Argumente 3 und 4 zu 7, der Operator “*” wird auf die nach der Addition direkt davor stehenden Argumente 2 und 7 angewandt, und schließlich “-” auf 1 und 14.

Ausdrücke in Klammern werden nicht gleich ausgewertet. Ein Operator verarbeitet geklammerte Ausdrücke als Argumente. Zum Beispiel wird “(2*)” als Argument des Operators “@” selbst als Operator gesehen, der ein Argument mit 2 multipliziert. Der Ausdruck “3(2*)@” wird zu “3 2*” bzw. 6 ausgewertet.

Der Taschenrechner verwendet (neben geklammerten Ausdrücken) nur ganze Zahlen als Daten. Zahlen im entsprechenden Wertebereich werden aber als Zeichen interpretiert und am Display angezeigt (am besten als ASCII-Zeichen, aber andere Codierungen sind ebenso erlaubt). Intern wird nicht zwischen Zahlen und Zeichen unterschieden. Beispielsweise wird die Zahl 65 am Display als “A” dargestellt. Vor einer Darstellung als “65” muss die Zahl in die ASCII-Werte 54 und 53 der beiden Ziffern umgewandelt werden.

Architektur:
Der Taschenrechner besteht aus folgenden Teilen:

Tastatur
: Über die Tastatur (= Standardeingabe) werden Programme und Daten eingegeben.

Display
: Das Display hat eine Größe von 4 Zeilen zu je 64 Spalten und kann in jedem Feld ein (ASCII-)Zeichen darstellen. Die Felder sind von 0

bis 255 durchnummeriert. Mittels eines Operators wird ein Zeichen in ein bestimmtes Feld am Display gesetzt.

Stack
: Ausdrücke in Postfix-Notation können mit Hilfe eines Stacks einfach berechnet werden: Jedes Argument wird als neuer Eintrag am Top-of-Stack abgelegt. Die Ausführung eines Operators nimmt die benötigte Anzahl von Argumenten vom Stack und legt Ergebnisse auf den Stack.

Eingabeliste
: Sie enthält die noch nicht abgearbeiteten Eingaben. Alle Eingaben werden zeichenweise in der Reihenfolge in dieser Liste abgearbeitet. Neben direkten Eingaben über die Tastatur des Taschenrechners werden auch durch den Operator “@” Eingaben in diese Liste geschrieben.

Zur Vereinfachung können für den Stack, die Eingabeliste, geklammerte Ausdrücke, Zahlen, etc. (vernünftig gewählte) Maximalgrößen festgelegt werden, bei deren Überschreitung eine Fehlermeldung ausgegeben wird. Nach einer Fehlermeldung kann die Programmausführung abgebrochen werden.

Operationen:
Die Bedeutung folgender Eingaben ist vordefiniert:

Ziffern “0” bis “9”:
: Alle direkt hintereinander stehenden Ziffern in der Eingabeliste werden zu einer Zahl zusammengesetzt, die als neuer Eintrag auf den Stack gelegt wird.

“(” und “)”:
: Die öffnende Klammer bewirkt, dass alle folgenden Eingaben bis zur entsprechenden schließenden Klammer als Einheit betrachtet werden, die (beispielsweise in Form eines Strings) als neuer Eintrag auf den Stack gelegt wird. Klammern können geschachtelt sein.

White-Space (Leerzeichen, Tabulator, Newline, Return):
: Diese Eingaben sind zur Trennung zweier Zahlen sinnvoll, haben sonst aber keine Bedeutung und werden ignoriert.

Arithmetische Operatoren und Vergleichsoperatoren (“+”, “-”, “*”, “/”, “%”, “&”, “|”, “=”, “<”, “>”):
: Diese zweistelligen Operatoren entsprechen den Grundrechnungen (mit “%” zur Berechnung des Divisionsrestes) und einfachen Vergleichen, wobei & bzw. | das logische UND bzw. ODER darstellen. Diese Operatoren nehmen die zwei obersten Elemente vom Stack und legen das Ergebnis auf den Stack. Wenn ein Argument keine Zahl sondern ein geklammerter Ausdruck ist, soll ein Fehler gemeldet werden, ebenso wenn ein Argument von & bzw. | kein Wahrheitswert (0 steht für “wahr” und 1 für “falsch”) ist. Ausgenommen hiervon ist nur “=”: Wenn “=” auf zwei gleiche geklammerte Ausdrücke oder zwei gleiche Zahlen angewandt wird, soll als Ergebnis 0 (wahr) zurückgegeben werden, sonst 1 (falsch). Bei nichtassoziativen Operationen ist auf die Reihenfolge

der Argumente zu achten: “4 2-” und “4 2/” und “2 4%” ergeben jeweils 2, und “4 2>” und “2 4<” liefern “wahr”. Ein Fehler soll gemeldet werden, wenn das zweite Argument von / oder % gleich 0 ist.

Vorzeichenumkehr (Negation) “~”:
: ist nur auf Zahlen definiert.

Kopieren “!”:
: ersetzt das oberste Element n am Stack durch eine Kopie des n-ten Elements am Stack (vom Top-of-Stack aus gezählt). Eine Fehlermeldung wird ausgegeben, wenn n keine positive Zahl ist oder der Stack nicht ausreichend viele Elemente enthält.

Löschen “#”:
: nimmt das oberste Element n vom Stack und entfernt zusätzlich das n-te Element vom Stack. Eine Fehlermeldung wird ausgegeben, wenn n keine positive Zahl ist oder der Stack nicht ausreichend viele Elemente enthält.

Anwenden “@”:
: nimmt das oberste Element vom Stack. Ist das Argument ein geklammerter Ausdruck, dann wird dieser (mit den umschließenden Klammern durch White-Space ersetzt) an vorderster Stelle in die Eingabeliste geschrieben, damit die darin enthaltenen Eingaben als nächste abgearbeitet werden. Ist das Argument eine Zahl, dann hat “@” keinerlei Auswirkung; das Argument wird wieder unverändert auf den Stack gelegt.

Auslesen “?”:
: nimmt die obersten zwei Elemente vom Stack, wobei ein Element eine nicht-negative Zahl n und das andere ein geklammerter Ausdruck sein muss; sonst erfolgt eine Fehlermeldung. Als Ergebnis wird der (ASCII-)Code des n-ten Zeichens im geklammerten Ausdruck auf den Stack gelegt. Falls der geklammerte Ausdruck weniger als n Zeichen enthält, wird -1 auf den Stack gelegt.

Erweitern “;”:
: nimmt die obersten zwei Elemente vom Stack, wobei ein Element eine als (ASCII-)Code interpretierbare Zahl und das andere ein geklammerter Ausdruck sein muss; sonst erfolgt eine Fehlermeldung. Als Ergebnis wird der an letzter Stelle (vor der schließenden Klammer) um ein Zeichen verlängerte geklammerten Ausdruck auf den Stack gelegt. Es wird also die Zahl (als Zeichen interpretiert) an den geklammerten Ausdruck angehängt.

Ausgabe “$”:
: nimmt die obersten zwei Elemente vom Stack, wobei ein Element einen (ASCII-)Code und das andere eine Nummer eines Felds am Display darstellt, und setzt das Zeichen an der entsprechenden Stelle im Display. Falls ein Element keine Zahl oder eine Zahl außerhalb des erlaubten Wertebereichs ist, kommt es zu einer Fehlermeldung.

Kontrollierter Eingabemodus “"”:
: beendet den normalen Eingabemodus (in dem Eingaben über die Tastatur in die Eingabeliste kommen) und startet den kontrollierten Eingabemodus, in dem alle Eingaben über die Tastatur in

geklammerten Ausdrücken abgelegt werden. Durch Abschluss einer Zeile mittels Enter bzw. Return wird die gesamte eingegebene Zeile als geklammerter Ausdruck (nach Hinzufügen umschließender Klammern) auf den Stack gelegt. Der kontrollierte Eingabemodus ist sinnvoll, wenn Eingaben vor der Verarbeitung kontrolliert, aufbereitet oder am Display angezeigt werden sollen.

Normaler Eingabemodus “'”:

beendet den kontrollierten Eingabemodus und startet den normalen Eingabemodus.

Beispiele:

Einige Beispiele sollen die Verwendung der Operatoren verdeutlichen. Wir beschreiben einen Zustand des Taschenrechners durch den Stackinhalt (links vom Zeichen “^”, Top-of-Stack direkt neben “^”, Einträge durch Leerzeichen getrennt) und die Eingabeliste (rechts von “^”, nächstes zu verarbeitendes Zeichen direkt neben “^”). Pfeile zwischen solchen Zustandsbeschreibungen zeigen Zustandsänderungen durch Ausführung von Operationen an.

Das erste Beispiel zeigt eine bedingte Anweisung: Auf dem Stack wird 0 (wahr) oder 1 (falsch) erwartet. Abhängig davon soll der eine oder andere geklammerte Ausdruck ausgewertet werden. Wir legen zuerst den Ausdruck für den wahr-Zweig “(9)” und dann den für den falsch-Zweig “(9~)” auf den Stack und führen einen Ausdruck “(4!5#2+#@)” aus, der die eigentliche bedingte Anweisung darstellt. Die folgenden Abarbeitungsschritte zeigen, was passiert, wenn am Stack zuvor 0 gelegen ist:

```
    0 ^(9)(9~)(4!5#2+#@)@
--> 0 (9) ^(9~)(4!5#2+#@)@
--> 0 (9) (9~) ^(4!5#2+#@)@
--> 0 (9) (9~) (4!5#2+#@) ^@
--> 0 (9) (9~) ^4!5#2+#@
--> 0 (9) (9~) 4 ^!5#2+#@
--> 0 (9) (9~) 0 ^5#2+#@
--> 0 (9) (9~) 0 5 ^#2+#@
--> (9) (9~) 0 ^2+#@
--> (9) (9~) 0 2 ^+#@
--> (9) (9~) 2 ^#@
--> (9) ^@
--> ^ 9
--> 9 ^
```

Das nächste Beispiel zeigt anhand der Berechnung von 3 Faktorielle, wie man rekursive Routinen realisieren kann. Zur Vereinfachung kürzen wir den Ausdruck (3!3!1-2!1=()5!C@3#*) durch A ab, wobei C für den Ausdruck (4!5#2+#@) aus dem vorigen Beispiel steht. Beachten Sie, dass A und C hier nur zur Vereinfachung der Lesbarkeit dient. Im Taschenrechner sollte statt A und C jeweils der entsprechende geklammerte Ausdruck vorkommen.

```
    3 ^A3!4#3!@3#
--> 3 A ^3!4#3!@3#
--> 3 A 3 ^!4#3!@3#
--> 3 A 3 ^4#3!@3#
--> 3 A 3 4 ^#3!@3#
```

```
--> A 3 ^3!@3#
--> A 3 3 ^!@3#
--> A 3 A ^@3#
--> A 3 ^ 3!3!1-2!1=()5!C@3#* 3#
--> A 3 3 ^!3!1-2!1=()5!C@3#* 3#
--> A 3 A ^3!1-2!1=()5!C@3#* 3#
--> A 3 A 3 ^!1-2!1=()5!C@3#* 3#
--> A 3 A 3 ^1-2!1=()5!C@3#* 3#
--> A 3 A 3 1 ^-2!1=()5!C@3#* 3#
--> A 3 A 2 ^2!1=()5!C@3#* 3#
--> A 3 A 2 2 ^!1=()5!C@3#* 3#
--> A 3 A 2 2 ^1=()5!C@3#* 3#
--> A 3 A 2 2 1 ^=()5!C@3#* 3#
--> A 3 A 2 1 ^()5!C@3#* 3#
--> A 3 A 2 1 () ^5!C@3#* 3#
--> A 3 A 2 1 () 5 ^!C@3#* 3#
--> A 3 A 2 1 () A ^C@3#* 3#
--> A 3 A 2 1 () A C ^@3#* 3#
...
--> A 3 A 2 A ^@ 3#* 3#
--> A 3 A 2 ^ 3!3!1-2!1=()5!C@3#* 3#* 3#
--> A 3 A 2 3 ^!3!1-2!1=()5!C@3#* 3#* 3#
--> A 3 A 2 A ^3!1-2!1=()5!C@3#* 3#* 3#
--> A 3 A 2 A 3 ^!1-2!1=()5!C@3#* 3#* 3#
--> A 3 A 2 A 2 ^1-2!1=()5!C@3#* 3#* 3#
--> A 3 A 2 A 2 1 ^-2!1=()5!C@3#* 3#* 3#
--> A 3 A 2 A 1 ^2!1=()5!C@3#* 3#* 3#
--> A 3 A 2 A 1 2 ^!1=()5!C@3#* 3#* 3#
--> A 3 A 2 A 1 1 ^1=()5!C@3#* 3#* 3#
--> A 3 A 2 A 1 1 1 ^=()5!C@3#* 3#* 3#
--> A 3 A 2 A 1 0 ^()5!C@3#* 3#* 3#
--> A 3 A 2 A 1 0 () ^5!C@3#* 3#* 3#
--> A 3 A 2 A 1 0 () 5 ^!C@3#* 3#* 3#
--> A 3 A 2 A 1 0 () A ^C@3#* 3#* 3#
--> A 3 A 2 A 1 0 () A C ^@3#* 3#* 3#
...
--> A 3 A 2 A 1 () ^@ 3#* 3#* 3#
--> A 3 A 2 A 1 ^ 3#* 3#* 3#
--> A 3 A 2 A 1 3 ^#* 3#* 3#
--> A 3 A 2 1 ^* 3#* 3#
--> A 3 A 2 ^ 3#* 3#
--> A 3 A 2 3 ^#* 3#
--> A 3 2 ^* 3#
--> A 6 ^ 3#
--> A 6 3 ^#
--> 6 ^
```

Testaufgabe 1:
Der Taschenrechner ist so primitiv, dass (abgesehen von Fehlermeldungen in der Konsole nach einem Programmabbruch) ohne Programmierung keinerlei Ausgabe erfolgt. Schreiben Sie daher in der Sprache des Taschenrechners ein Programm, das jeweils die neuesten Eingaben in den Taschenrechner sowie die obersten Elemente am Stack im Display anzeigt. Es bleibt Ihnen überlassen, wie Sie den beschränkten Platz am Display gut nutzen.

Praktischer Hinweis: Zum Programmieren bzw. Starten des Taschenrechners sind lange und komplizierte Ausdrücke einzugeben. Am einfachsten ist dies zu erledigen, wenn die Eingabe über eine einfache Textkonsole erfolgt. Dann kann man Programme rasch durch Copy-and-Paste eingeben.

Testaufgabe 2:
Entwerfen Sie einen möglichst kurzen und effizienten Ausdruck in der Sprache des Taschenrechners der entscheidet, ob eine über die Tastatur eingelesene Zahl eine Primzahl ist oder nicht und das Ergebnis in lesbarer Form (das heißt als Text, nicht nur als 0 oder 1) am Display ausgibt. Testen Sie mit mehreren, auch größeren Zahlen (zumindest bis zu 1000).

Zusatzaufgaben für Interessierte:
Es stellt sich die Frage, ob man mit diesem Taschenrechner wirklich alles Programmieren kann. Ist es möglich, damit eine Turing-Maschine zu bauen? Die Lösung dieser Zusatzaufgaben ist nicht verpflichtend und hat keinen Einfluß auf die Beurteilung.

Anfang | HTML 4.01 | letzte Änderung: 2013-05-17 (Puntigam)