Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

25.05.2009

6. Hausübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Abgabetermin: Dienstag, 02.06.2009, 10:00 Uhr

1 Theorie

1.1 Korrektheit

Beweisen Sie die partielle Korrektheit folgenden Programmabschnitts

```
z[0] = 0;
k = 0;
while (n!=0) {
  z[k] = n % 8;
  n = (n - z[k]) / 8;
  k = k+1;
}
```

der eine natürliche Zahl n in Oktaldarstellung umformt, bezüglich der Spezifikation:

- Vorbedingung: $\{n = A > 0\}$
- Nachbedingung: $A = \sum_{i=0}^{k-1} z[i] \cdot 8^i$

Gehen Sie dabei gemäß dem Schema aus der Vorlesung vor. (16 Punkte)

1.2 Untere Schranken

In der Vorlesung wurde die untere Schranke $\Theta(n \log n)$ für Sortierverfahren angegeben. Wieso widerspricht dies nicht der linearen Laufzeit von *Distribution Sort*? (4 Punkte)

2 Programmierung

Dynamische Stack-Implementierung: In der Vorlesung wurde eine Implementierung des Stacks über ein Array (`ArrayStack`) vorgestellt. Hierbei wurde die Kapazität des Stacks zu Beginn festgelegt und konnte nicht zur Laufzeit verändert werden. Schreiben Sie eine Klasse `DynamicArrayStack`, die das Interface `Stack` von der Vorlesungsseite implementiert. Der Stack soll auch hierbei als Array realisiert werden, dessen Größe sich allerdings während der Benutzung dynamisch anpasst: Sind mehr als 90% des Stack belegt, so soll die Grösse jeweils um die initiale Größe erweitert werden. Die initiale Grösse soll der Klasse `DynamicArrayStack` im Konstruktor übergeben werden. (5 Punkte)

Stack zur Auswertung arithmetischer Ausdrücke: Verwenden Sie die Stack-Implementierung der letzten Teilaufgabe, um einen "Taschenrechner" zu programmieren, der einfache arithmetische Ausdrücke auswertet. In den Ausdrücken sollen lediglich ganze einstellige Zahlen (0-9) vorkommen; unterstützte Operationen sollen Multiplikation und Addition sein. Die Ausdrücke sollen in *Postfix-Notation* (auch bekannt als "Umgekehrte Polnische Notation") vorliegen. Hierbei werden zuerst die Operanden niedergeschrieben, und danach der Operator. Der Ausdruck 5 + 3 würde demnach als `5 3 +` aufgeschrieben. Diese Notation bietet den Vorteil, daß keine Klammern erforderlich sind; so ist z.B. `5 9 8 + 4 6 * * 7 + *` die Postfix-Notation für 5*(((9+8)*(4*6))+7). Zur Trennung der Elemente (Werte / Operatoren) der Eingabe soll jeweils ein Leerzeichen verwendet werden.

Schreiben Sie eine Klasse **`ArithmeticExpressionEvaluator`**, die eine Methode **`int evaluate(String expression)`** besitzt. Diese Methode soll einen als String vorliegenden arithmetischen Ausdruck in Postfix-Notation wie folgt auswerten:

- Lesen Sie einen gültigen arithmetischen Ausdruck zeichenweise ein.
- Ein Wert wird im Stack gespeichert.
- Ein Operator wird auf die beiden obersten im Stack liegenden Werte, angewendet; das Ergebnis geht wieder in den Stack.
- Am Schluss steht der Wert des Ausdrucks im Stapel und wird von dort mit der letzten Pop-Operation ausgelesen.

Der Rückgabewert soll dem Ergebnis der Auswertung entsprechen. Achten Sie bei Ihrer Implementierung auf die korrekte Behandlung von Ausnahmen, z.B. wenn ein ungültiger arithmetischer Ausdruck eingegeben wird.

Testen Sie Ihre Implementierung mittels der Klasse **`Calculator`** von der Vorlesungsseite; diese Klasse liest einige arithmetische Ausdrücke von der Datei **`expressions.txt`** und wertet sie aus. Die korrekte Ausgabe lautet:

```
Result 1: 10
Result 2: 231
Result 3: 2
Result 4: 451
Result 5: 2102
Result 6: 153
Result 7: 1641
Result 8: 362880
Result 9: 52568
Result 10: 282
```

(10 Punkte)

Viel Spaß und viel Erfolg!