PRG1x & ADE1x Einf. i. d. Programmierung (int. LVA) Üb. zu Element. Alg. u. Datenstrukt. **WS 13/14, Übung 10**

Abgabetermin: Sa, 18.01.2014

☐ **Gr. 1,** DI (FH) G. Horn, MSc
☐ **Gr. 2,** J.-P. Haslinger, MSc **Name** _____ **Aufwand in h** _____

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

1. Laufzeitkomplexität (7 + 4 + 1 Punkte)

Gegeben ist folgender Algorithmus, der für eine in Form einer Zeichenkette gegebene ganze Zahl ihren entsprechenden Zahlenwert ermittelt. Alle in der Zeichenkette enthaltenen Zeichen, die keine Ziffern sind (wie z. B. das Tausendertrennzeichen ".") werden dabei ignoriert.

```
CONST
    Zero = 48;
    Nine = 57;

FUNCTION GetValue(numStr: STRING): LONGINT;
VAR
    i: INTEGER;
    asciiCode: BYTE;
    result: LONGINT;
BEGIN
    result := 0;
    i := 1;
    WHILE i <= Length(numStr) DO BEGIN
        asciiCode := Ord(numStr[i]);
        IF (asciiCode >= Zero) AND (asciiCode <= Nine) THEN
            result := result * 10 + asciiCode - Zero;
        i := i + 1;
    END;
    GetValue := result;
END;
```

a) Stellen Sie unter der Verwendung der nachstehenden Tabelle eine Formel auf, welche für die Länge der Zeichenkette *numStr* und die Anzahl der darin enthaltenen Ziffern die exakte Laufzeit des Algorithmus berechnet. Berechnen Sie damit die Laufzeit für die Zeichenketten: '1234567' *(Länge = 7, Ziffern = 7)*, '1.234.567' *(Länge = 9, Ziffern = 7)*, '1xv323a+#42..83', '000001', '789', 'abc', '1' und ''.

Operation	Ausführungszeit
Wertzuweisung	1,4
Vergleich	1
Logische Verknüpfung *(AND etc.)*	1
Indizierung	0,4
Addition, Subtraktion	0,6
Multiplikation	3,6
Length(...) / Ord(...)	0 *(wird z. B. schon beim Übersetzen vom Compiler "erledigt")*
Prozeduraufruf sonst	16 + 2 * Anzahl der Parameter

b) Stellen Sie weiters unter der Annahme, dass im Schnitt auf jede dritte Ziffer ein anderes Zeichen folgt (wie z. B. in '123.456.789'), eine neue Formel auf, die nun für eine beliebige Zeichenkette nur unter der Angabe ihrer Länge die (angenäherte) Laufzeit des

Algorithmus berechnet. Erstellen Sie damit eine Tabelle, welche die Laufzeiten für alle möglichen Zeichenketten von Länge 0 bis Länge 255 darstellt.

c) Bestimmen die asymptotische Laufzeitkomplexität für den gegebenen Algorithmus in Abhängigkeit zur Länge der Zeichenkette. Sehen Sie dabei weiterhin im Schnitt jedes vierte Zeichen als ein Sonderzeichen an. Begründen Sie, wie Sie auf Ihre Lösung gekommen sind. Wie steht diese in Zusammenhang mit der in b) erstellten Tabelle?

2. Laufzeitkomplexität und Rekursion (6 + 3 + 1 + 2 Punkte)

Gegeben ist nun ein zweiter, rekursiver Algorithmus, der wiederum für eine Zeichenkette ihren entsprechenden Zahlenwert ermittelt.

```
FUNCTION GetValue2(numStr: STRING): LONGINT;
    FUNCTION GetValueInternal(pos: INTEGER): LONGINT;
    VAR
        asciiCode: BYTE;
    BEGIN
        IF pos < 1 THEN
            GetValueInternal := 0
        ELSE BEGIN
            asciiCode := Ord(numStr[pos]);
            IF (asciiCode >= Zero) AND (asciiCode <= Nine) THEN
                GetValueInternal := asciiCode - Zero + 10 *
                                        GetValueInternal(pos - 1)
            ELSE
                GetValueInternal := GetValueInternal(pos - 1);
       END;
    END;
BEGIN
    GetValue2 := GetValueInternal(Length(numStr));
END;
```

a) Stellen Sie wieder eine Formel für die Laufzeit auf Basis von Länge und Anzahl der Ziffern auf und bestimmen Sie „exakten“ Laufzeiten für die Beispiele aus 1.a).

Vorgehensweise: Bestimmen Sie zunächst getrennt für jeden der (drei) möglichen Zweige der inneren Funktion *GetValueInternal* die Laufzeit eines einzelnen Durchlaufs und überlegen Sie dann (z. B. an Hand eines Beispiels), wie oft jeder der Zweige durchlaufen wird. Durch einfaches Multiplizieren und Addieren erhalten Sie dann die Gesamtlaufzeit der ganzen Rekursion.

b) Finden Sie auch hier wieder eine Formel, welche von der in 1.b) beschriebenen Verteilung von Ziffern und Nicht-Ziffern ausgeht, und stellen Sie die gleiche Tabelle wie 1.b) auch für diesen Algorithmus auf.

c) Bestimmen Sie weiters auch für diesen Algorithmus die asymptotische Laufzeitkomplexität. Betrachten Sie dazu am besten wieder die Daten der für b) erstellten Tabelle.

d) Vergleichen Sie nun Ihre Analysen des iterativen (Beispiel 1) und den rekursiven Algorithmus (Beispiel 2). Wie ist jeweils die asymptotische Laufzeitkomplexität? Was zeigen (im Gegensatz dazu) die Grob- bzw. die Feinanalyse? Was schließen Sie daraus in Bezug auf die Güte der beiden Lösungen?