Auffrischungen zu Allgemeine Informatik
29. April 2005

Andreas F. Borchert
borchert@mathematik.uni-ulm.de

28. April 2005

Aufgabe 4

Ein Anagramm ist ein Wort oder eine Phrase, die genau aus den Buchstaben eines anderen Worts oder Phrase zusammengesetzt ist. Hier sind einige Beispiele:

- "EU-Osterweiterung" und "Euro tut wegreisen"
- "Geisterfahrer" und "irres Gefaehrt"
- "Clint Eastwood" und "Old West Action"

(Quellen: `http://www.anagramme.de/` und `http://www.wordsmith.org/anagram/`).

Ziel dieser Aufgabe ist die Entwicklung eines Programms, das Ihnen dem Finden neuer Anagramme behilflich ist. Es ist aber hilfreich, in einzelnen Etappen vorzugehen:

Schritt 1

Bei einem Anagramm werden Klein- und Großbuchstaben nicht unterschieden. Ferner sind Leerzeichen und andere Nicht-Buchstaben nicht zu berücksichtigen. Entsprechend ist es sinnvoll, eine Prozedur **Normalize** zu haben, die eine Zeichenkette dahingehend bearbeitet, indem sie

- Sämtliche Kleinbuchstaben in Großbuchstaben verwandelt (hier hilft die Standardfunktion **CAP**) und
- alle Nicht-Buchstaben entfernt.

Dazu sollten Sie sofort ein kleine Testprozedur schreiben, die mit **Read.Line** Zeilen aus der Eingabe liest, sie an **Normalize** übergibt und das Resultat anschließend ausgibt. Beispiel:

```
dublin$ Anagrams
Dies ist ein Test von Normalize!
DIESISTEINTESTVONNORMALIZE
Hurra, es klappt :-)
HURRAESKLAPPT
dublin$
```

Schritt 2

Wenn wir zwei Kandidaten miteinander vergleichen wollen, ob sie zusammen ein Anagramm ergeben, müssen wir untersuchen, wie häufig welcher Buchstabe in jedem Kandidaten vorkommen. Entsprechend wird als nächstes eine Prozedur **GetLetterFrequencies** benötigt, die die Buchstaben einer Zeichenkette zählt und als Häufigkeits-Array zurückgibt. Dabei darf die Prozedur (dank **Normalize**) davon ausgehen, daß nur Großbuchstaben in der Zeichenkette enthalten sind.

Wenn Sie dafür eine weitere Testprozedur spendieren, könnte dies so aussehen:

```
dublin$ Anagrams
Dies ist ein Test von GetLetterFrequencies!
C  1
D  1
E  9
F  1
G  1
I  4
L  1
N  3
O  1
```

```
Q  1
R  2
S  4
T  6
U  1
V  1
dublin$
```

Hinweise: Da es insgesamt 26 verschiedene Buchstaben im lateinischen Alphabet gibt, sollten sie das Häufigkeits-Array entsprechend von 0 bis 25 indizieren, wobei 0 den Buchstaben "A" reprasentiert 1 den Buchstaben "B" bis hin zu 25, das für "Z" steht. Entsprechend benötigen Sie eine Konvertierung in beide Richtungen:

- Vom Großbuchstaben zum zugehörigen Index:
  `index := ORD(letter) - ORD("A");`
- Vom Index zum zugehörigen Großbuchstaben:
  `letter := CHR(ORD("A") + index);`

Es empfiehlt sich, die Ausgabe der Häufigkeiten in eine Prozedur **PrintLetterFrequencies** zu verpacken, da sie diese im nächsten Schritt benötigen werden.

Schritt 3

Zwei Zeichenketten bilden jetzt ein Anagram, wenn nach der Anwendung von **Normalize** und **GetLetterFrequencies** für beide Zeichenketten die Häufigkeits-Arrays sich gleichen. Bei der Konstruktion eines Anagramms, ist es aber häufig hilfreich zu erfahren, welche Buchstaben noch übrig bleiben bzw. ob bereits eine Verletzung der Anagram-Eigenschaft vorliegt, weil von einem Buchstaben bereits zuviel vergeben worden sind. Entsprechend ist die Differenz zweier Häufigkeits-Arrays interessant, die wiederum als Häufigkeits-Array repräsentiert wird.

Schreiben Sie also eine Prozedur **SubtractLetterFrequencies**, die zwei Häufigkeits-Arrays erhält und ein Häufigkeits-Array zurückliefert, in der die Differenzen zu finden sind. Damit Sie rasch herausfinden, ob ein Anagram gefunden worden ist, sollte diese Prozedur einen **BOOLEAN**-Rückgabewert liefern, der genau dann **TRUE** ist, falls die Differenz überall 0 beträgt.

So könnte dann ein Dialog aussehen:

```
dublin$
dublin$ Anagrams
Phrase #1: Hallo!
Phrase #2: Aloha!
This is not yet an anagram:
A -1
L  1
Phrase #1: Listen
Phrase #2: Silent
Congratulations, you found an anagram!
Phrase #1: dublin$
```

Schritt 4

Wenn an einem Anagram gebastelt wird, ist es hilfreich, ein Wort herauszunehmen und mit dem Rest weiterzuarbeiten. Entsprechend wäre eine Prozedur **GenLetters** sinnvoll, die ein Häufigkeits-Array in eine Zeichenkette verwandelt, die, wenn sie durch **GetLetterFrequencies** bearbeitet werden würde, das gleiche Häufigkeits-Array wieder liefern würde.

So könnte dann der Dialog aussehen, wenn bei fehlender Anagram-Eigenschaft die verbliebenen Zeichen als Zeichenkette ausgegeben werden würden:

```
dublin$ Anagrams
Phrase #1: Allgemeine Informatik
Phrase #2: Erfolg
This is not yet an anagram:
A  2
E  2
I  3
K  1
L  1
M  2
N  2
T  1
Unused letters: AAEEIIIKLMMNNT
Phrase #1: AAEEIIIKLMMNNT
Phrase #2: mit
This is not yet an anagram:
A  2
```

```
E  2
I  2
K  1
L  1
M  1
N  2
Unused letters: AAEEIIKLMNN
Phrase #1: AAEEIIKLMNN
Phrase #2: Eile
This is not yet an anagram:
A  2
I  1
K  1
M  1
N  2
Unused letters: AAIKMNN
Phrase #1: AAIKMNN
Phrase #2: kam
This is not yet an anagram:
A  1
I  1
N  2
Unused letters: AINN
Phrase #1: Allgemeine Informatik
Phrase #2: kam mit Erfolg in Eile an
Congratulations, you found an anagram!
Phrase #1: dublin$
```