# 1. Belegaufgabe

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

| | | |
|---|---|---|
| Programmierung II | Sommersemester 2011 | Dozent: Horst Hansen |
| Ausgabe: 21.4.2011 | | Abgabe Gruppe 1: 19.5., Gruppe 2: 26.5 |

**Lernziele**:

Mit der Lösung dieser Aufgabe sollen sie zeigen, dass Sie in der Lage sind, unter Verwendung von generischen dynamischen Datentypen Programme mit Ein- und Ausgabe in Dateien strukturiert zu entwerfen und in C zu programmieren.

**Spezifische Ziele**:

- Strukturieren von Programmen
- Dokumentation des Programms durch seine Funktionsaufrufstruktur
- Verwenden von Zeigern
- Entwerfen und Implementieren von generischen dynamischen Datentypen
- Benutzen der Dateiein- und -ausgabefunktionen von C
- Benutzen von weiteren Funktionen aus den Standardbibliotheken von C

## **Aufgabe**: Programm zum Indizieren von Texten

Schreiben Sie ein Programm, das ein oder mehrere nach *ISO Latin 1* kodierte Textdateien indiziert!

Das Programm soll Texte aus einer Datei lesen und eine alphabetisch sortierte Liste (fast) aller Wörter im Text erstellen. Zu jedem Wort soll es sich alle Zeilennummern merken, in denen das Wort vorkommt. Die Zeilennummern sollen aufsteigend sortiert sein. Diese Wortliste soll in eine Textdatei geschrieben werden, wobei jede Zeile der Ausgabedatei mit einem Wort beginnt und danach, getrennt durch Leerzeichen, die zum Wort gehörigen Zeilennummern enthält.

In den folgenden Abschnitten werden die Anforderungen an das zu erstellende Programm aufgeführt. Alle Anforderungen, bei denen nichts anderes angegeben ist, sind *Muß-Kriterien*. Blau gedruckte Anforderungen sind *Soll-Kriterien*.

## 1 Funktionale Anforderungen

### 1.1 Anforderungen an die dynamische Liste

- Es wird eine doppelt verkettete dynamische Liste implementiert.
- Die Liste wird als generische Liste realisiert. (mittel)
- Die generische Liste übernimmt die Verantwortung für das Speichern aller Daten innerhalb der Liste. (schwer)

### 1.2 Anforderungen an die Anwendung

- Das Programm indiziert die Wörter einer oder mehrerer Eingabedatei(en) und gibt den errechneten Index in die Ausgabedatei und optional am Terminal aus.
- Es besteht die Möglichkeit, Wörter, die kürzer sind als ein gegebener Schwellwert, beim Indizieren auszulassen.
- Ein Wort ist eine zusammenhängende Folge von Zeichen, die mit einem Unterstrich oder einem Buchstaben beginnt und anschließend Buchstaben, Ziffern, Bindestriche oder Unterstriche enthält. Oder als regulärer Ausdruck: `[A-Za-z_]([A-Za-z0-9]|-|_)*` .
- Alle anderen Zeichen sind *Trennzeichen*, d.h. sie beenden ein Wort.

## 2 Anforderungen an die Benutzungsschnittstelle

### 2.1 Anforderungen an die dynamische Liste

- Die Liste soll über ihre Programmierschnittstelle (API) alle notwendigen Funktionalitäten für eine einfache Implementierung der Anwendung bereitstellen.

### 2.2 Anforderungen an die Anwendung

- Die Steuerung des Programms erfolgt ausschließlich über Kommandozeilenparameter.
- Das Programm soll mit den folgenden Kommandozeilenparametern gestartet werden:

  `<program> <options> <outputfile> <inputfile>`
  `<program>` : Programmname
  `<options>` :

  `-p` : Ausgabe der Indexliste auf der Konsole

  `-w=<n>` : Auslassen aller Wörter mit weniger als n Zeichen

  `<outputfile>` : Dateiname der Ausgabedatei mit der Indexliste
  `<inputfile>` : Dateiname der Eingabedatei mit dem zu indizierenden Text
  `<inputfile>+` : Liste von Eingabedateien mit zu indizierendem Text. (mittel)
- Ausgabe von aussagekräftigen Fehlermeldungen bei fehlerhaften Eingaben auf der Kommandozeile.
- Verhindern des Überschreibens von Ausgabedateien (leicht)
- Die Ausgabe des vom Programm erzeugten Wortindexes am Terminal und in die Ausgabedatei soll so aufgebaut sein:

  `<wort> BLANK <zeilennummer>+`
  Jedes Wort beginnt eine neue Zeile.

  Bei mehreren Eingabedateien:
  `<wort> BLANK <dateiindex>+` , wobei
  `<dateiindex> ::= <dateiname> BLANK <zeilennummer>*` ist.
  Dabei beginnen Wörter eine neue Zeile, jeder weitere Dateiname beginnt ebenfalls eine neue Zeile, wird aber einige Zeichen weit eingerückt.
- Die Ausgabe der Wortliste erfolgt stets lexikografisch sortiert.

  Achtung!
  Sortierung durch
  c-Lib prüfen!
- Die Ausgabe der Zeilennummern erfolgt in aufsteigender Reihenfolge.

## 3 Allgemeine Anforderungen

- Für die Implementierung dürfen nur die Standardbibliotheken von C verwendet werden.
- Wie üblich gilt: Die Verwendung von globalen Variablen ist verboten.
- Stellen Sie sicher, daß das Programm *vor dem Beenden ohne Fehler* allen dynamisch belegten Arbeitsspeicher wieder freigibt! Zum Überprüfen dieser Eigenschaft verwenden Sie das freie Werkzeug `valgrind`.
- Es wird ein Makefile bereitgestellt für
  - das Erzeugen des ausführbaren Programms
  - das Erzeugen der Programmdokumentation
  - das Löschen aller aus den Programmquellen erzeugten Daten
  - das Überprüfen der Speicherverwendung des Programms mittels des Werkzeugs `valgrind`

### 3.1 Anforderungen an die dynamische Liste

- Die Listenimplementierung wird als Bibliothek bereitgestellt.

### 3.2 Anforderungen an die Anwendung

- Zum Testen Ihres Programms stehen Testdateien im Verzeichnis `Testdaten` auf der Seite zur Lehrveranstaltung zur Verfügung.

**Benotung**:
Um eine sehr gute Note erreichen zu können, müssen Sie außer den *Muß-Kriterien* auch alle *Soll-Kriterien* erfüllen. Die Erfüllung einzelner *Soll-Kriterien* führt zu einer schrittweisen Verbesserung der Ausgangsnote *drei* für die Bewertung.
Lösungen, die sich nicht an die unter *Allgemeine Anforderungen* genannten Verbote halten, werden mit **null** Punkten bzw. der Note 5 (**ungenügend**) bewertet!
Beachten Sie bitte dazu auch die auf der Seite zur Lehrveranstaltung veröffentlichte Notenskala!

**Als Lösung sind abzugeben**:

- ein **Ausdruck** der Funktionsaufrufstruktur (oder *funktionalen Struktur*) Ihres Programms (Kann automatisch von `doxygen` generiert werden! `doxygen` ist freie Software. Achtung: Erst `GraphViz` und danach `doxygen` installieren!)
- ein **Ausdruck** des kommentierten Quelltextes des Programms (Um Papier zu sparen, können Sie den Quelltext vor dem Drucken mit dem Skript `pp` aufbereiten! Dieses Skript ist auf allen Rechnern unserer Labore installiert.)
- ein **Ausdruck** einer Zeichnung, die die von Ihnen tatsächlich implementierte Datenorganisation des Programms darstellt
- ein **Ausdruck** der durch Ihr Programm erstellten Indexliste für den Text in der Datei `Test.txt`
- **ohne Ausdruck**: Die durch `doxygen` erstellte Programmdokumentation des Programms mit Angaben zum Autor und der Dokumentation aller Funktionen

Die Abgabe der Lösung erfolgt durch jede Gruppe von maximal 2 Studierenden persönlich im Rahmen der entsprechenden oben genannten Übungsstunde auf einem Laborrechner an den Dozenten. Bei der Abgabe der Lösung muß jede Gruppe das unter dem Betriebssystem Linux funktionsfähige Programm vorführen und erläutern. Jeder Studierende der Gruppe muß in der Lage sein, alle Fragen zur vorgeführten Lösung zu beantworten.

**Bewertungskriterien**:
Bewertet werden neben der Vorführung mit Erläuterung (siehe oben):

- die korrekte Funktion des Programms
- die funktionale Struktur
- die Robustheit des Programms
- die Lesbarkeit des Programmtextes
- die Vollständigkeit und Verständlichkeit der Programmdokumentation
- die Form der schriftlichen Dokumente
- die Einhaltung der Programmierrichtlinien
- die Extras