2. Belegaufgabe

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Programmierung II	Sommersemester 2011	Dozent: Horst Hansen
Ausgabe: 9.6.2011		Abgabe Gruppe 1: 14.7., Gruppe 2: 7.7.

Lernziele:
Mit der Lösung dieser Aufgabe sollen sie zeigen, dass Sie in der Lage sind, unter der Verwendung der objektorientierten Sprachelemente der Programmiersprache C++ generische dynamische Datentypen zu entwerfen und zu implementieren. Die Funktionsfähigkeit der implementierten Klassen soll durch eine einfache Anwendung zur Indizierung von Texten belegt werden.

Spezifische Ziele:

- Objektorientiertes Strukturieren von Programmen
- Dokumentation des Programms durch seine Klassenstruktur
- Objektorientierte Implementierung in C++
- Implementieren von generischen dynamischen Datentypen
- Verwenden von Funktionsparametern
- Benutzen der Dateiein- und -ausgabefunktionen von C++
- Aufteilung von Programmen in Header- und Quelldateien
- Verwendung von make in Softwareentwicklungsprojekten

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 und Freigeben 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 **deutsche** Buchstaben, Ziffern, Bindestriche oder Unterstriche enthält.
  Oder als regulärer Ausdruck: `([A-Za-z_] | Umlaute) ([A-Za-z0-9]|-|_| Umlaute)*` .
  `Umlaute` sind die deutschen Umlaute und ß.
- Alle anderen Zeichen sind *Trennzeichen*, d.h. sie beenden ein Wort.
- Die lexikografische Sortierung der Wörter behandelt die Umlaute so: ä wird als ae betrachtet, usw., ß als ss!

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
- 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.
- Die Ausgabe der Zeilennummern erfolgt in aufsteigender Reihenfolge.

3 Allgemeine Anforderungen

- Für die Implementierung dürfen nur Standardbibliotheken von C und C++ — allerdings keine Bibliotheken der STL — verwendet werden.
- Wie üblich gilt: Die Verwendung von globalen Variablen ist verboten.
- Zur Implementierung neuer Typen werden ausschließlich *Klassen* verwendet.
- Der Ausgabeoperator darf als globaler Operator überladen werden.
- 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 der Bibliothek
  - 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 (bestehend aus mehreren Klassen) bereitgestellt.

3.2 Anforderungen an die Anwendung

- Außer der Funktion `main` gibt es keine globalen Funktionen in der Implementierung.
- Callbackfunktionen werden — sofern es sich nicht vermeiden läßt — als Klassenmethoden, aber nicht als globale Funktionen implementiert.
- 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** des Klassendiagramms mit den öffentlichen Methoden Ihrer Klassen
- ein **Ausdruck** einer Skizze, die den Aufbau der von ihnen *tatsächlich implementierten Liste* mit allen Teilen (und Typen) zeigt
- 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** 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 objektorientierte 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