Übungen zu Computergrundlagen WS 2014/2015

Übungsblatt 14: Programmieren in C 3

28.01.2015

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist
  - **Freitag, 06.02.2015, 10:00**
- Schickt die Lösungen bitte per Email an Euren Tutor.

Zum Testen Eurer Programme könnt Ihr die Dateien `/group/cgl/2014/14/gpl-3.0.txt` (General Public License), `/group/cgl/2014/14/mobydick.txt` (Moby Dick) und `/group/cgl/2014/14/mobydick_sorted.txt` (Moby Dick, alle Wörter alphabetisch sortiert) verwenden.

Aufgabe 14.1: C-Programm debuggen (4 Punkte)

Das Programm `/group/cgl/2014/14/occurrence.c` enthält die Musterlösung des letzten Übungsblatts. Leider ist es fehlerhaft und enthält sieben Fehler. Nur bei Vieren davon beklagt sich der Compiler, die anderen Fehlern sind anderer Natur. Findet und korrigiert alle Fehler des Programms.

Hinweise:

- Kompiliert dazu das Programm wie folgt: `gcc -std=gnu99 -Wall -g -o occurrence occurrence.c`
- Zum Debuggen könnt Ihr den Kommandozeilendebugger `gdb` wie folgt benutzen:
  `gdb --args occurrence gpl-3.0.txt`
- Wer möchte, darf auch eines der grafischen Frontends `ddd` oder `kdbg` verwenden, oder sogar eine der integrierten Entwicklungsumgebungen (IDE) `eclipse`, `netbeans` oder `qtcreator` ausprobieren.

Aufgabe 14.2: Binärbäume (6 Punkte)

Nun soll das Programm aus der vorigen Aufgabe verbessert werden.

- **14.2.1** Erweitert das (korrigierte) Programm aus der letzten Aufgabe so, dass es zum Speichern der Worthäufigkeiten einen Binärbaum verwendet. Neue Wörter sollen in diesem Binärbaum lexikographisch (Über die Funktion `strcmp`) einsortiert werden. Schreibt dafür separate Funktionen `search`, `insert` und `output`, um die Lesbarkeit des Programms zu erhöhen (Siehe Hinweise). (5 Punkte)
- **14.2.2** Messt die Laufzeit des Programmes im Vergleich zur Version mit den dynamischen Arrays und zum Pythonskript mit allen drei Beispieltexten. Wie schneidet es im Vergleich bei den einzelnen Texten ab? Wieso? (1 Punkt)

  Hinweise:

  - Für die Laufzeitmessungen solltet Ihr beim Kompilieren des Programms die Option `-O3` (mit dem Buchstaben „O“, nicht mit der Ziffer „0“!) verwenden.

– Implementiert die Funktionen zum Umgang mit dem Baum am besten *rekursiv*, es wird dadurch wesentlich einfacher! Linker und rechter Teilbaum sind ja selber Bäume. Ob Ihr im linken oder rechten Teilbaum weitermachen müsst, könnt Ihr schon am Wurzelknoten feststellen.

– Um einen Baum zu implementieren, braucht ihr `structs`. Zusätzlich zu den Zeigern auf die benachbarten zwei Blätter beinhaltet jeder Knoten ein Wort und einen dazugehörigen Zähler für die Wort-Häufigkeit:

```
typedef struct _node{
        char *word;
        int count;
        struct _node *left, *right;
} node;
```

Wenn ihr *nicht* `typedef` verwendet, dann müsst ihr vor den Datentyp `node` immer noch ein `struct` schreiben.

– Um die Aufgabe etwas übersichtlicher zu gestalten, schreibt Euch Funktionen, die nach einem Wort suchen, eines einfügen und eine Funktion, die den Inhalt (also die Wörter und Zähler) des gesamten Baums ausgibt:

```
node *search(node *p_root, char *word);
void insert(node **p_root, char *word);
void output(node *root);
```

`node` ist hier das `struct`, das den Binärbaum repräsentiert.

– Vergesst nicht, dass die Suche nach einem Wort auch dann etwas zurück geben muss, wenn das Wort *noch nicht* eingetragen wurde. Dann muss nämlich das Wort neu hinzugefügt werden. Als Rückgabe bietet sich in diesem Fall der Nullpointer NULL an.

– Die `insert`-Funktion muss ebenfalls nach der korrekten Stelle suchen, an der das neue Wort eingetragen werden kann. Sie ähnelt daher der `search`-Funktion.