Prof. Dr. Franz Schweiggert   Institut für Angewandte Informationsverarbeitung   16. November 2009
Michael Mattes / Wolfgang Kaifler   Blatt 6

WiMa–Praktikum I / Programmierpraktikum im Wintersemester 2009/2010

Abgabetermin: bis 24. November 2009

15 Modularisierung (4 Punkte)

Als kleinen Einstieg in die Modularisierung sollen Sie Ihre Implementationen (oder den Lösungsvorschlag) der Stringfunktionen vom letzten Blatt in ein eigenes Modul `mystr.o` auslagern. Stellen Sie hierbei eine passende Headerdatei `mystr.h` bereit, die ausschließlich Funktionsdeklarationen (keine Definitionen!) enthält. Die dazugehörigen Definitionen sollen sich in der Datei `mystr.c` befinden.

Ändern Sie die Datei `strfun.c` so ab, dass Ihre Stringfunktionen dort nur verwendet (nicht aber deklariert oder definiert) werden. Schließlich sollen Sie noch in der Lage sein, mit dem `gcc` Objektdateien erzeugen zu lassen und diese in einem weiteren Schritt zu einer ausführbaren Datei zusammenzubinden.

Hinweis 1: Das Kapitel 12 des Skripts enthält alle nötigen Informationen zum Thema Modularisierung.

Hinweis 2: Wenn Sie `size_t` verwenden wollen, ohne `string.h` einzubinden, binden Sie `stddef.h` ein.

16 Sortieralgorithmen–Bibliothek (12 Punkte)

Wenn Sie die Zusatzaufgabe weiter unten lösen, haben Sie diese Aufgabe automatisch mitgelöst.

Schreiben Sie ein kleines Modul (d.h. eine Headerdatei mit entsprechenden Funktionen und eine C-Datei mit deren Implementierung) von Sortieralgorithmen für `ints`, das `mysort.o` heißt. Stellen Sie hierbei mindestens einen Sortieralgorithmus pro Gruppenmitglied zur Verfügung.

Testen Sie die Funktionsweise aller Algorithmen anhand eines selbstgeschriebenen Testprogramms, welches nicht Teil des Moduls ist.

17 Allgemeine Sortierbibliothek (12 Zusatzpunkte)

Dies ist eine Zusatzaufgabe, die nur bei Interesse gemacht werden sollte!

Als Verfeinerung der letzten Aufgabe ist es mit nur wenigen Veränderungen möglich, beliebige Daten (z.B. `ints`, `doubles`, Strings, die Geldanlagen vom letzten Blatt, sonstige Verbünde, etc.) zu sortieren.

Möglich wird dies durch die Verwendung des allgemeinen unspezifizierten Zeigertyps `void*`, der auf beliebige Daten (insbesondere mit beliebiger Größe!) zeigen kann. Eine allgemeine Sortierfunktion benötigt daher neben dem Zeiger auf den Anfang der Liste und der Anzahl der Elemente auch die Angabe der Größe eines einzelnen Elements. Eine Beispielsignatur könnte daher wie folgt aussehen:

```
sort(void *anfang, size_t anzahl, size_t groesse);
```

Da die wenigsten Daten mit einem einfachen `<` verglichen werden können, muss die Sortierfunktion außerdem wissen, wie die Elemente verglichen werden. Dies lässt sich mit einer Vergleichsfunktion (Komparator) bewerkstelligen, die zwei Elemente übergeben bekommt und analog zu `strcmp` einen `int`-Wert als Ergebnis liefert.

Die Sortierfunktion muss den zu verwendenden Komparator natürlich kennen – er muss also (als Funktionszeiger) mit übergeben werden. Die Signatur der Sortierfunktion muss also wie folgt erweitert werden:

```
sort(void *anfang, size_t anzahl, size_t groesse,
 int (*comparator)(const void *, const void *));
```

Implementieren Sie ein eigenes allgemeines Sortiermodul (mit einem Algorithmus pro Gruppenmitglied) und testen Sie es mit mindestens drei verschiedenen Typen von Daten, z.B. ein Ganzahltyp, ein Gleitkommatyp, Strings oder die Geldanlagen vom vorigen Blatt.

Hinweis 1: Zum Testen müssen Sie für jeden Datentyp einen Komparator schreiben.

Hinweis 2: Die Funktion `qsort` aus der Standardbibliothek bietet ein ganz allgemeines QuickSort an. Sie sollen das natürlich nicht verwenden, aber in der Manpage dazu finden Sie auch Verwendungsbeispiele.

Viel Erfolg!