Dr. Annette Bieniusa
Mathias Weber, M. Sc.
Peter Zeller, M. Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Übungsblatt 13: Software-Entwicklung 1 (WS 2017/18)

Ausgabe: 29.01.18
Abgabe: 05.02.18

Aufgabe 1 String-Suche (10 Punkte)

Ziel dieser Aufgabe ist es, ein C-Programm zu schreiben, welches einen gegebenen Such-Text in einen Eingabe-Text finden kann und alle Zeilen des Eingabe-Texts ausgibt, welche den Such-Text enthalten.

a) Schreiben Sie eine Methode `bool contains(char *search, int searchSize, char *input, int inputSize )`. Die Methode nimmt einen Such-Text `search` der Länge `searchSize` und einen Eingabe-Text `input` der Länge `inputSize`. Die Funktion soll `true` zurückgeben, wenn der gesuchte Text im Eingabe-Text vorkommt und ansonsten `false`.

   Verwenden Sie keine Funktionen aus der C-Bibliothek für diese Aufgabe.

b) Schreiben Sie eine `main`-Funktion für ihr Programm, welches einen Such-Text als Programm-Parameter nimmt und die Standard-Eingabe nach Vorkommen des Such-Strings durchsucht. Die Zeilen der Eingabe, welche den Such-Text enthalten, sollen zusammen mit ihrer Zeilennummer ausgegeben werden. Dabei soll zuerst die Zeilennummer ausgegeben werden, dann ein Leerzeichen und und dann der Text der Zeile.

   Verwenden Sie die Funktion `size_t strlen(const char *s)` um die Länge von Strings zu berechnen. Für das zeilenweise Lesen der Eingabe könne Sie die Funktion `char *fgets(char *s, int n, FILE *stream )` verwenden. Diese Funktion nimmt einen Buffer `s` der Größe `n` und einen Eingabe-Strom. Wenn Sie `stdin` als Eingabe-Strom verwenden, wird von der Standardeingabe gelesen. Die Funktion liest bis eine neue Zeile startet, das Ende des Stroms erreicht ist oder die maximale Länge `n-1` erreicht ist. Rückgabe der Funktion ist `s`, falls das Einlesen erfolgreich war und `NULL` falls das Ende des Stroms erreicht wurde oder ein Fehler auftritt. Zeilenumbrüche am Ende der Zeile werden in `s` beibehalten. Der eingelesene String endet jeweils mit einem Null-Zeichen. Verwenden Sie zum Einlesen einen Buffer der Größe `1024`. Längere Zeilen müssen nicht korrekt behandelt werden, sollten aber nicht zu undefinierten Verhalten führen.

Aufgabe 2 Sortierte binäre Bäume in C (14 Punkte)

In dieser Aufgabe sollen Sie ein Wörterbuch (engl.: dictionary) implementieren, welches intern einen sortierten binären Baum verwendet. Das Wörterbuch ist eine Map mit Strings als Schlüssel und als Wert. Wir verwenden die folgenden Typdefinitionen um das Wörterbuch selbst und die Knoten des Baums darzustellen:

```
typedef struct treenode treenode_t;
struct treenode
{
    char *key;
    char *value;
    treenode_t *left;
    treenode_t *right;
};

typedef struct dictionary dictionary_t;
struct dictionary
{
    treenode_t *root;
};
```

Ein Knoten speichert Schlüssel `key` und Wert `value` als Zeiger zu einem C-String ab. Außerdem hat jeder Knoten einen Zeiger auf seinen linken und rechten Kind-Knoten. Das Wörterbuch selbst enthält nur einen Zeiger auf den Wurzelknoten.

Die Sortierung des Baums ergibt sich über das Vergleichen der Schlüssel mit `strcmp`. Für einen Knoten im Baum mit Schlüssel `k` gilt, dass alle Schlüssel `kl` im linken Teilbaum kleiner sind (`strcmp(kl,k) < 0`) und alle Schlüssel `kr` im rechten Teilbaum größer (`strcmp(kr, k) > 0`). Der Baum enthält also auch keinen Schlüssel mehrmals.

Laden Sie sich zur Bearbeitung der Aufgabe die Vorlage `dict.c` herunter. Diese enthält die Typen, Vorlagen für die zu implementierenden Funktionen und eine `main`-Funktion mit der sich das Wörterbuch testen lässt. Dazu können Sie dem Programm die folgenden Befehle geben, um mit einem Wörterbuch zu arbeiten:

```
put k v
get k
del k
```

Die ersten 3 Zeichen geben den Befehl an. Danach kommt getrennt durch ein Leerzeichen der Schlüssel und für `put` nach einem weiteren Leerzeichen der Wert. Bei einem Aufruf von `get` wird der aktuelle Wert im Wörterbuch ausgegeben.

a) Schreiben Sie eine Funktion `dictionary_t *dict_new()`, welche ein neues leeres Wörterbuch erstellt und einen Zeiger auf die Wörterbuch-Struktur zurückgibt.

b) Schreiben Sie eine Funktion `void dict_put(dictionary_t *dict, char *key, char *value)`, welche einen Eintrag mit Schlüssel `key` und Wert `value` im Wörterbuch speichert. Falls bereits ein Eintrag mit dem Schlüssel existiert, wird der Wert ersetzt.

   Die C-Strings `key` und `value` sind nur für den Aufruf der Funktion gültig. Erstellen Sie deshalb wenn nötig Kopien der Strings.

c) Schreiben Sie eine Funktion `char *dict_get(dictionary_t *dict, char *key)`, welche den Wert zum gegeben Schlüssel `key` zurückgibt. Falls `key` nicht im Wörterbuch gespeichert ist, soll `NULL` zurückgegeben werden.

d) Schreiben Sie eine Funktion `void dict_free(dictionary_t *dict)`, welcher sämtlichen vom Wörterbuch benutzten Speicher freigibt.

e) *Freiwillige Zusatzaufgabe*: Schreiben Sie eine Funktion `void dict_delete(dictionary_t *dict, char * key)`, welche den Eintrag mit Schlüssel `key` aus dem Wörterbuch löscht.

Aufgabe 3 Verkette Liste (15 Punkte)

Laden Sie sich die Datei `linkedlist.c` herunter, welche die Implementierung von einfach verketteten Listen aus der Vorlesung enthält. In dieser Aufgabe sollen Sie diese Liste um weitere Funktionen erweitern.

a) Schreiben Sie eine Funktion **`bool`** `list_is_sorted(linked_list_t *ll)`, welche prüft ob die in der Liste `ll` gespeicherten Werte aufsteigend sortiert sind.

b) Schreiben Sie eine Funktion **`bool`** `list_has_duplicates(linked_list_t *ll)`, welche prüft, ob in der Liste `ll` Werte existieren, die mehrmals vorkommen.

c) Schreiben Sie eine Funktion **`void`** `list_add_before(linked_list_t *ll,` **`int`** `x,` **`int`** `y)`, welche einen neuen Eintrag mit Wert `x` direkt vor dem ersten Eintrag mit Wert `y` in die Liste einfügt. Wenn `y` nicht in der Liste enthalten ist soll `x` am Ende der Liste eingefügt werden.

d) Schreiben Sie eine Funktion **`int`** `list_remove(linked_list_t *ll,` **`int`** `value)`, welche alle Vorkommen von `value` aus der Liste `ll` entfernt und zurückgibt, wie viele Elemente entfernt wurden.

   Die Liste soll von der Funktion nur einmal durchlaufen werden.

Aufgabe 4 Mergesort (9 Punkte)

Implementieren Sie den Mergesort Algorithmus (siehe Vorlesung) für das Sortieren von einfachverketteten Listen.

Laden Sie sich die Datei `mergesort.c` herunter und implementieren Sie die Funktion **`void`** `sort(linked_list_t *ll)`. Die `main`-Funktion in der Vorlage testet Ihre `sort`-Funktion mit Zahlen, die von der Standardeingabe gelesen werden.