Frage **1**

Unvollständig

Erreichbare Punkte: 7,00

Hinweis:

Roter Text kennzeichnet Ihre Eingaben in der Konsole, Ihre Eingaben werden nicht tatsächlich rot sein.
Weißer Text kennzeichnet die Ausgaben Ihres Programms.

Aufgabe

Es ist eine Liste von Listen zu implementieren, die nach verschiedenen Kriterien sortiert werden kann.

Ein Eintrag in der äußeren Liste repräsentiert ein Wort. Ein Wort besteht aus Buchstaben, daher soll ein Eintrag die inneren Liste einen Buchstaben repräsentieren. Beide Listen sind einfach verkettet und verweisen jeweis auf den nächsten Eintrag in der Liste.

Die Structs der Listen sind vorgegeben:

```
typedef struct innernode {
  char letter;
  struct innernode* next;
} InnerNode;

typedef struct node {
  InnerNode* word;
  struct node* next;
} Node;
```

Lesen Sie die gesamte Angabe bevor Sie mit der Implementierung starten.

Program

Schreiben Sie eine Konsolenapplikation, die ein Menü anzeigt. Nach jedem ausgeführten Befehl soll das Menü erneut angezeigt werden, bis das Program beendet wird.

Folgende Befehle sollen unterstützt werden:

- **a** (append), ein neues Wort wird eingelesen und am Ende der Liste hinzugefügt (siehe Beispiel).
- **i** (insert), ein neues Wort, sowie ein Index am dem dieses eingefügt werden soll, werden eingelesen und das Wort wird entsprechend in die Liste eingefügt (siehe Beispiel).
- **d** (delete), ein Index wird eingelesen und dessen Eintrag aus der Liste gelöscht (siehe Beispiel).
- **s** (sort), die Liste soll alphabetisch oder nach Länge der Wörter auf- bzw. absteigend sortiert werden. Dafür sind zwei weitere Eingaben nötig (siehe Beispiel).
- **p** (print), die Liste soll in der gegebenen Reihenfolge ausgegeben werden. Jeder innere Knoten soll seinen Buchstaben ausgeben und äußere Knoten werden mit Leerzeichen getrennt (siehe Beispiel). Ist die Liste leer soll "empty list" ausgegeben werden.
- **x** (exit), das Program wird beendet und aller Speicher wieder freigegeben.

Hinweis: Sie können um sich das Einlesen der Eingabe zu vereinfachen temporär einen String verwenden und Sie dürfen davon ausgehen, dass nicht mehr als 20 Zeichen pro Wort eingegeben werden. Anschließend muss der String aber in eine Liste von `char` umgewandelt werden.

Beachten Sie, dass beim Sortieren, sowie beim Einfügen und Löschen, sich das erste Element der Liste ändern kann. Vergessen Sie also nicht die `head` Variable upzudaten.

Allgemein zu Eingaben:
Leerzeichen und Zeilenumbrüche sollen ignoriert werden. Sie können sich darauf verlassen, dass gültige Zeichen eingegeben werden, wenn Zeichen erwartet werden und Zahlen eingegeben werden, wenn Zahlen erwartet werden. Sie können weiters davon ausgehen, dass die eingegebenen Indizes im gültigen Bereich der Liste sind.

Sortieren

Um die äußere Liste zu sortieren implementieren Sie einen Sortieralgorithmus Ihrer Wahl. Beispielsweise Selection Sort, Insertion Sort oder Bubble Sort.

Verwenden Sie hierfür eine Funktion `sort` die drei Parameter entgegennimmt:

- `Node* head` den Pointer auf das erste Listenelement.
- `int (*compare)(InnerNode*, InnerNode*)` einen Funktionspointer, der zwei innere Listen vergleicht (siehe unten).

- `int descending` wenn auf 0 gesetzt soll aufsteigend, sonst absteigend sortiert werden.

Als Returnwert empfielt es sich, den Listenpointer `Node*` zurück zu geben, um diesen in der aufrufenden Funktion übernehmen zu können.

Der Funktionspointer erlaubt es uns den gleichen Code zum Sortieren zu verwenden, auch wenn wir nach unterschiedlichen Kriterien sortieren. Übergeben wird eine `compare` Funktion, die zwei Werte (in unserem Fall `InnerNode*` das erste Element einer inneren Liste) entgegen nimmt und einen Integer Wert zurückgibt:

- der Rückgabewert ist negativ, wenn das erste Element kleiner ist als das zweite.
- der Rückgabewert ist positiv, wenn das erste Element größer ist als das zweite.
- und der Rückgabewert ist 0, wenn beide Elemente gleich groß sind.

Zu Ihrer Referenz ist eine compare Funktion, die fürs alphabetisch sortieren verwendet werden soll, bereits vorgegeben:

```
int compare_alphabethically(InnerNode* m, InnerNode* n){
  int valueM = m?m->letter:0; // ASCII value of the letter
  int valueN = n?n->letter:0; // or 0 when node is NULL
  int diff = valueM - valueN; // indicates which is the higher value
  if(diff==0){ // on equal value
    if(m!=NULL && n!=NULL){ // if both lists have another entry
      // continue recursively
      return compare_alphabethically(m->next, n->next);
    }
  }
  return diff;
}
```

Anmerkung: Die Funktion vergleicht ASCII Werte, daher sind aufsteigend zuerst Großbuchstaben und dann Kleinbuchstaben in der Reihenfolge. Sind zwei Buchstaben gleich, werden die nächsten zwei Buchstaben verglichen, bis eines der Wörter endet.

Implementieren Sie analog dazu eine Funktion `compare_length`, die zurück gibt, welches der beiden Wörter länger ist.

Diese beiden Funktionen sollen an die Sortierfunktion übergeben werden und in dieser aufgerufen werden um die inneren Listen zu vergleichen.

Wenn zwei Elemente den selben Wert (gleiche Länge bzw. gleiche Buchstaben) haben, können diese nach dem Sortieren in beliebiger Reihenfolge nebeneinander stehen.

Beispiel Aus- und Eingaben:

Choose action: (a)ppend, (i)nsert, (d)elete, (s)ort, (p)rint, e(x)it: a

Enter Word: hello

Choose action: (a)ppend, (i)nsert, (d)elete, (s)ort, (p)rint, e(x)it: i

Enter Index: 0

Enter Word: world

Choose action: (a)ppend, (i)nsert, (d)elete, (s)ort, (p)rint, e(x)it: p

world hello
Choose action: (a)ppend, (i)nsert, (d)elete, (s)ort, (p)rint, e(x)it: s

Sort (a)lphabetically or by (l)ength? a

Sort (a)scending or (d)escending? a

Choose action: (a)ppend, (i)nsert, (d)elete, (s)ort, (p)rint, e(x)it: p

hello world
Choose action: (a)ppend, (i)nsert, (d)elete, (s)ort, (p)rint, e(x)it: d

Enter Index: 1

Choose action: (a)ppend, (i)nsert, (d)elete, (s)ort, (p)rint, e(x)it: p

hello
Choose action: (a)ppend, (i)nsert, (d)elete, (s)ort, (p)rint, e(x)it: x

Hinweis:
Achten sie auf korrekte Speicherverwaltung! Insbesondere muss jeder angeforderte Speicher (spätestens) beim Beenden wieder freigegeben werden.

Allgemein zur Formatierung:
In Coderunner wird jede einzelne Eingabe von einem Zeilenumbruch bestätigt, der in der Ausgabe aber nicht sichtbar ist. Wenn Sie außerhalb von Coderunner das Programm starten, selber Werte eingeben (siehe Beispielausgabe) und dabei Zeilenumbrüche produzieren, wirkt sich das auf die Ausgabe aus (mehr Zeilenumbrüche als beim automatischen Testen in Coderunner).

Allgemein zu Ausgaben:
Geben Sie Zeilenumbrüche immer vor einer neue Zeile aus und nicht am Ende. Wenn Sie sich daran halten, sollte die Formatierung leichter zur Übereinstimmung mit der erwarteten Ausgabe gebracht werden können. Das Programm startet daher auch mit einem Zeilenumbruch.

Antwort: (Abzugssystem: 0 %)

Antwort zurücksetzen

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct innode {
    char letter;
    struct innode* next;
} InnerNode;

typedef struct node {
    InnerNode* word;
    struct node* next;
} Node;

int compare_alphabethically(InnerNode* m, InnerNode* n){
    int valueM = m?m->letter:0; // ASCII value of the letter
    int valueN = n?n->letter:0; // or 0 when node is NULL
```

Prüfen

◄ Sammlung: Fragen zum Eigenstudium V

Direkt zu:

Peer Review Zwischentest 4 ►