**Fakultät IV - Elektrotechnik und Informatik**
*Einführung in die Programmierung WS 2016/17*
*Feldmann / Lichtblau / Rost / Semmler / Streibelt / Tiesel / Krüger*

# Aufgabenblatt

letzte Aktualisierung: 24. Dezember, 00:26 Uhr

(f633f57f556e4bb1e3dd81bebb1c4e0d05ae3628)

*Ausgabe:* Freitag, 22.12.2017
*Abgabe:* spätestens Mittwoch, 17.01.2018 18:00

***Thema:*** *Volltextsuche, Queues*

## Bitte beachten Sie folgende Hinweise zu diesem Übungsblatt

- Dieses Übungsblatt richtet sich an die Studierenden, die entweder gar nicht am C-Kurs teilgenommen haben oder ihre Leistung aus dem C-Kurs ersetzen möchten.
- Zur erfolgreichen Bearbeitung dieser Aufgabe ist eine Anmeldung über OSIRIS erforderlich. Eine Anleitung findet sich auf ISIS.
- Eine Anmeldung ist bis zum 05.01.2018 möglich. Eine Abmeldung ist nicht vorgesehen!
- Wenn Sie sich für die C-Kurs Kompensationsaufgabe anmelden, verfallen etwaige im C-Kurs erworbene Punkt unwiderruflich im Moment der Anmeldung!
- Da es sich bei dieser Aufgabe um ein **selbstständig zu bearbeitendes** Projekt handelt, sind hierfür keine speziellen Tutorien oder Rechnerübung vorgesehen.
- Die Abgabe der Aufgabe erfolgt in einem separaten SVN Repository, dieses müssen Sie zunächst neu auschecken:
  `svn co https://teaching.inet.tu-berlin.de/services/svn/kompensation`
  `↪ -wise1718`
- Nach der Anmeldung zur Aufgabe kann es einige Minuten dauern, bis Sie für das Repository freigeschaltet werden.
- Die Abgabe der Aufgabe erfolgt per SVN als Einzelabgabe dann im folgenden Ordner:
  `kompensation-wise1718/Studierende/<L>/<tubIT-Login>@tubit/Abgaben/Kompensation`
  wobei `<L>` durch den ersten Buchstabe des TUBIT-Logins ersetzt werden muss. Der Abgabeordner wird durch uns angelegt.
- Der Dateiname dieser Abgabe lautet `kompensation_ckurs.c`.
- Bei dieser Abgabe sind keine automatischen Tests vorgesehen, es wird lediglich geprüft, ob die Abgabe compiliert.
- Das Introprogteam kann während der vorlesungsfreien Zeit keine Unterstützung leisten.
- Bitte organisieren Sie sich im speziell für diese Aufgabe eingerichteten ISIS Forum untereinander und unterstützen Sie sich gegenseitig! Bitte posten Sie jedoch keine Lösungen - wir werden wieder auf Plagiate prüfen!



## 1. Aufgabe: Implementierung einer Volltextsuche

In dieser Aufgabe sollen Sie einen Algorithmus zur Volltextsuche implementieren. Das Ziel ist es, dass Sie für einen Eingabetext angeben können, wie oft und an welcher Textposition eine Phrase vorkommt. Die zu suchende Phrase besteht aus einem oder mehreren Wörtern, die durch Leerzeichen getrennt sind. Beachten Sie dabei, dass die einzelnen Wörter exakt übereinstimmen müssen, und nicht nur in Teilwörtern (siehe Beispiel unten).

Um die gesamten Suchphrase mit dem Eingabetext vergleichen zu können, aber nicht den kompletten Eingabetext in den Speicher einlesen zu müssen, sollen Sie eine FIFO (First-In First-Out) Queue verwenden (in dieser Verwendung auch *Look-ahead Buffer* genannt).

Die (maximale) Länge der Textqueue entspricht dabei der Länge der Suchphrase. Die Textqueue soll hierbei jeweils die letzten $n$ eingelesenen Wörter enthalten, wobei $n$ die Anzahl an Wörtern der Suchphrase ist. Die Queue soll keinesfalls den gesamten Eingabetext enthalten! Der Eingabetext wird iterativ Wort für Wort in die Textqueue eingelesen (`enqueue()`). Ist die Textqueue einmal gefüllt, wird das erste Element (Wort) entfernt (`dequeue()`) und das nächste Wort im Eingabetext hinten an die Textqueue angehängt (`enqueue()`). Damit entspricht die Länge der Textqueue maximal der Anzahl der Wörter im Suchphrase und der Eingabetext wird wortweise durchlaufen.

Der Suchphrase wird ebenfalls mittels `enqueue()` in eine zweite Queue (Suchqueue) eingelesen. Anders als der Eingabetext muss dieser String nur einmal eingelesen werden und kein `dequeue()` aufgerufen werden.

Bei jedem neuem Wort des Eingabetextes, das an die Textqueue angehängt wird, wird der aktuelle Inhalt der Textqueue mit der Suchphrase in der Suchqueue verglichen. Dafür werden die beiden Queues elementweise, d.h. Wort für Wort verglichen. Bei einem Treffer (alle Elemente sind gleich) soll die Textposition (am wievielten Wort des Eingabetextes beginnt die Suchphrase) ausgegeben werden. Da eine gesuchte Phrase mehrfach im Eingabetext vorkommen kann, wird der Eingabetext wie oben beschrieben weiter eingelesen. Am Ende soll zudem ausgegeben werden, wie oft die Suchphrase insgesamt im Eingabetext vorkommt.

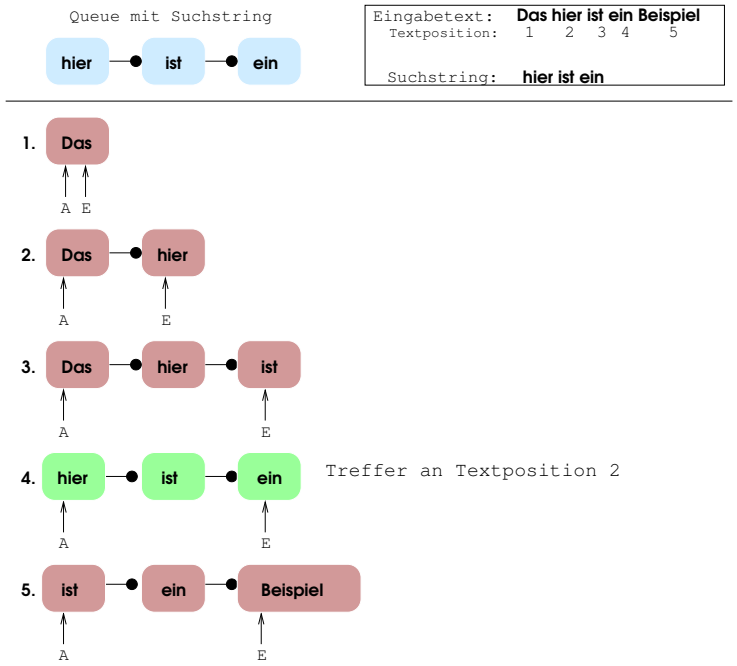


Abbildung 1: Beispiel des Progammablaufs

Abbildung 1 zeigt schematisch den Algorithmus zum Vergleich der Suchphrase mit dem Eingabetext.
Jeder Schritt entspricht einem Durchlauf. Die farbigen Boxen entsprechen Queueelementen, die Pfeile zwischen den Elementen entsprechen den `naechstes` Pointern und die Pfeile mit `A` und `E` stehen für die Pointer auf den Anfang bzw. das Ende der Queue.
Der Eingabetext mit Textpositionen sowie der Suchphrase sind in der Box angegeben. Die Schritte 1. bis 3. zeigen das initiale Auffüllen der Textqueue mit den ersten Worten des Eingabetextes. Die (maximale) Länge der Queue (in diesem Falle 3) ergibt sich aus der Länge der Suchphrase. Nach dem 3. Schritt wird jedes folgende Wort am Ende der Textqueue angefügt und dafür das erste Wort aus der Queue gelöscht. In diesem Beispiel wird das erste Element *"Das"* gelöscht und dafür am Ende *"ein"* eingefügt.
Sobald die Textqueue einmal aufgefüllt ist (Schritt 3), wird die sie in jedem Schritt, wenn ein neues Wort eingefügt wurde mit der Suchqueue verglichen. Bei jedem Treffer (Schritt 4) wird die Textposition ausgegeben.



**Beachten Sie beim Bearbeiten der Aufgabe folgende Hinweise:**

- Implementieren Sie die FIFO Queue wie in der Vorlesung erläutert auf Basis einer verketteten Liste. Benutzen Sie dabei die Datenstrukturen und Funktionssignaturen aus der Vorgabe.
- Verwenden Sie dynamische Speicherallozierung für die Implementierung der verketteten Listen. Geben sie Speicher explizit frei.
- Machen Sie keine Annahmen über die Länge der zu suchenden Phrase und des Eingabetextes.
- Der Eingabetext soll zu keinem Zeitpunkt komplett in den Speicher geladen sein!
- Das Programm operiert auf Wörtern als elementare Einheit. Die Eingabezeichenketten werden jeweils an den Leerzeichen geteilt.
- Phrasen sollen nur auf vollständigen Wörtern geprüft werden, so soll das Programm z.B. die Phrase *Der kleine Hund* nicht mit *der kleine Hundebesitzer* gleichsetzen.
- Die maximale Länge der Wörter ist auf 255 Zeichen (siehe Codevorgabe) begrenzt.
- Der Eingabetext ist bereits vorverarbeitet und enthält keine Satz- und Sonderzeichen.
- Das Zählen der Textposition beginnt bei 1. D.h. das erste Wort im Eingabetext erhält den Index 1.

Der Aufruf Ihres Programms soll folgendermaßen aussehen:
`./programm <Dateiname> <Wort1> <Wort2> ...`
Beispiel: `./ckurs_kompensation_vorgabe eingabetext.txt Ich will`

Das Format der Ausgabe soll der Folgenden entsprechen:

```
1. Treffer beginnend bei Wort Nr. 10227
2. Treffer beginnend bei Wort Nr. 12958
3. Treffer beginnend bei Wort Nr. 13217
4. Treffer beginnend bei Wort Nr. 13280
5. Treffer beginnend bei Wort Nr. 14336
6. Treffer beginnend bei Wort Nr. 15090
7. Treffer beginnend bei Wort Nr. 16378
8. Treffer beginnend bei Wort Nr. 30072
===> 8 Treffer
```

Listing 1: Vorgabe `ckurs_kompensation_vorgabe.c`

```c
/* ==== KOMPENSATION ABGABE ====
 * Kompensationsaufgabe
 * Abgabe von: Erika Mustermann
 * ============================
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#define MAX_MATCH 200 // maximale Anzahl auszugebender Matches
#define MAX_STRING 255 // Maximale Wortlänge

/*** Datenstrukturen ***/
// QElement wird von Queue benutzt
struct QElement {
    // HIER Code einfügen
    // * Datenstruktur implementieren
};

typedef struct QElement QElement;

struct Queue {
    // HIER Code einfügen
    // * Datenstruktur implementieren
};

typedef struct Queue Queue;

// Queue initialisieren
Queue* init_queue() {
    // HIER Code einfügen
}

// Speicher der Qeue freigeben
void queue_freigeben(Queue *queue) {
    // HIER Code einfügen
};

// Füge string als QElement am Ende der Queue an
void enqueue(Queue *queue, char* string){
    // HIER Code einfügen
}

// Lösche das erste QElement der Queue
void dequeue (Queue *queue) {
    // HIER Code einfügen
}

// Debug Ausgabe für Queue
void queue_ausgeben(Queue *queue) {
    // HIER Code einfügen
}
```



```c

// Vergleiche Inhalte zweier Queues bis zur Länge
int match(Queue *suchphrase, Queue *queue) {
    // HIER Code einfügen
}

int main(int argc, char** argv) {
    // Parameter einlesen
    if (argc < 3)
    {
        printf("Nutzung: %s <Dateiname> <Wort1> <Wort2> ...\n", argv[0]);
        return 1;
    }

    // HIER Code einfügen
    // * Initialisiere Queues
    // * Fülle die Queue der Suchphrase
    //   Tipp: Die Elemente findest du bei argv[2] bis argv[argc]
    // * Überpüfe mit queue_ausgeben ob du es korrekt eingelesen hast.

    // HIER Code einfügen
    // * Wörter aus Datei mit enqueue in die Queue befördern bis Queue voll ist.
    // * Von da an mit enqueue und dequeue Wörter in die Queue lesen und Queues
    //   vergleichen
    // * Gebe die Positionen der gefundenen Phrase im Text an, in dem du die Wörter des
    //   Text von 1 an hochzählst. Die Position der Phrase ist identisch mit der Position
    //   des ersten Wortes der Phrase.

    // HIER Speicher der Queues freigeben
    // * Überprüfe mit valgrind ob alles freigeben wird.
}
```