Dr. habil. Bernd Schürmann
Dr. Annette Bieniusa
pinc-support@cs.uni-kl.de

TU Kaiserslautern
Fachbereich Informatik

Lösungshinweise zum Übungsblatt 13: Programmieren in C (WS 2018/19)

1. Die Aufgaben auf diesem Übungsblatt sind alle freiwillig zu bearbeiten. Falls Sie bei Blatt 11 die nötige Punktzahl nicht erreicht haben, können Sie die fehlenden Punkte hier nachholen. Bei Fragen wenden Sie sich bitte direkt an `pinc-support@cs.uni-kl.de`.
2. Die Abgabe erfolgt über das Exclaim-System gemeinsam mit Ihrem Teampartner.
3. Zur Beantwortung von Fragen und Hilfestellung bei der Bearbeitung kommen Sie bitte in die wöchentliche Fragestunde, Mittwoch 15:30, in Terminalraum 32-410.
4. Bitte laden Sie einzelne Dateien hoch, wie in der Aufgabenstellung angegeben, keine Archive, keine kompilierten Dateien.
5. Bitte beachten Sie unsere Regelung bei Plagiaten:
   - Wenn Sie sekundäre Quellen, wie Bücher oder das Internet verwenden, müssen Sie immer die Quelle angeben. Das einfache Kopieren aus anderen Quellen ist für die Übungen nicht gestattet. Wenn wir in einer Übungsabgabe kopierten Code ohne Quellenangabe finden, wird die gesamte Abgabe mit 0 Punkten bewertet.
   - Sie können Übungsaufgaben gerne mit den Mitgliedern anderer Teams diskutieren. Sie sollten jedoch Ihren Code vor der Abgabefrist nicht an andere Teams weitergeben bzw. zeigen.
   - Wenn Code von anderen Teams kopiert wurde, werden die Abgaben **von beiden Teams** mit 0 Punkten bewertet.
   - Wir behalten uns vor Punkte auch nachträglich abzuziehen, wenn ein Verstoß erst später bemerkt wird.

Aufgabe 1 *Programmverständnis (5 Punkte)*

Gegeben sind die folgenden C-Strukturen zur Darstellung von einfach verketteten Listen:

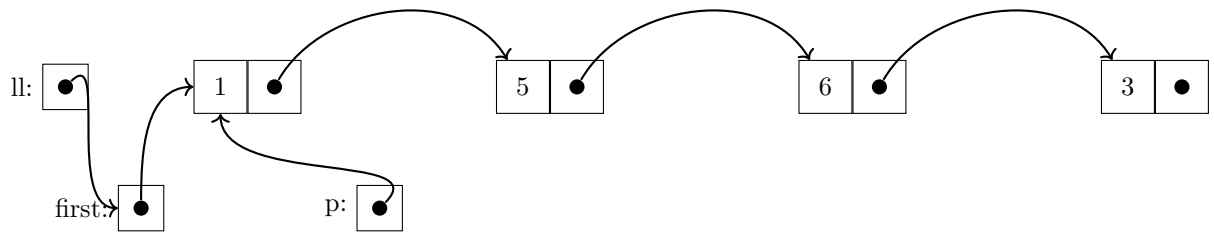
```c
typedef struct node node_t;
struct node {
  int value;
  node_t *next;
};
```

```c
typedef struct list list_t;
struct list {
  node_t *first;
};
```

Erklären Sie, was die folgende C-Funktion `f` macht. Verwenden Sie dazu die Eingabeliste mit den Elementen `1, 5, 6, 3` und zeichnen Sie den Zustand der Liste nach Ausführen der Funktion. Der Zustand vor der Ausführung der Schleife ist unten bereits vorgegeben.

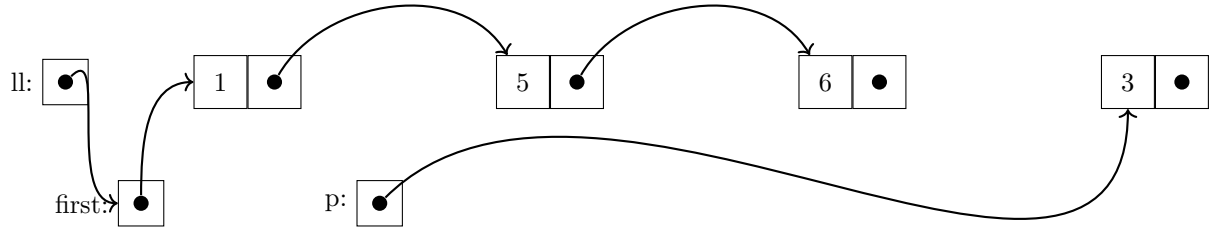
```c
node_t * f(list_t *ll) {
  node_t *p;
  node_t *tmp;

  if (ll->first == NULL) {
      return;
  }
  p = ll->first;
  if (p->next == NULL) {
      ll->first = NULL;
  } else {
    while (p->next->next != NULL) {
        p = p->next;
    }
    tmp = p->next;
    p->next = p->next->next;
    p = tmp;
  }
  return p;
}
```



Lösung 1

Die Funktion entfernt den letzten Knoten aus der Liste und gibt einen Pointer auf diesen Knoten zurück.



Texteditor (Teil 3)

Dieses Aufgabenblatt erweitert den in den vorangegangenen Übungen entwickelten Texteditor. Sie können für diese Aufgabe entweder Ihren eigenen Code verwenden oder eine entsprechende Vorlage von der Webseite der Vorlesung herunterladen.

Aufgabe 2 *Suchen und Ersetzen (5 Punkte)*

Erweitern Sie Ihren Editor um eine Funktion zum Suchen und Ersetzen von Texten :

`c/<text1>/<text2>`	Ersetzt im gesamten Dokument den Text `<text1>` durch den Text `<text2>`.

Achtung: Dazu muss die Funktion `next_command` entsprechend erweitert werden! Verwenden Sie zur Implementierung der Kommandos nach Möglichkeit Funktionen der String-Bibliothek (z.B. `strcmp`, `strcpy`, `memmove`). Das Suchen von `<text1>` soll dabei case-sensitive sein, d.h. Klein- und Großbuchstaben werden unterschieden.

Hinweis: Beachten Sie, dass sich die Zeilenlänge durch die Textersetzung ändern kann. In diesem Fall muss bei der Variante des Editors mit dynamischer Speicherverwaltung (siehe Blatt 12) der Speicherbereich für eine Zeile durch `realloc` so vergrößert oder verkleinert werden, dass der geänderte Text genau hineinpasst. Achten Sie hierbei darauf, dass das Endezeichen der Zeile `'\0'` korrekt verarbeitet wird.

Aufgabe 3 *Funktionspointer: Unterschiedliche Eingabeformate (10 Punkte)*

Neben den bisher verwendeten Editor-Kommandos sollen jetzt auch Kommandos in einer Langform erlaubt sein:

```
change "<text1>" "<text2>"
delete <row>
```

```
insert <row> "<text>"
print bzw. print <row>
quit
reset
search "<text>"
```

Nennen Sie die bisherige Funktion `next_command` in `next_command_short` um und implementieren Sie eine weitere Funktion `next_command_long`, die das hier angegebene Kommandoformat einliest.

Im Hauptprogramm soll eine neue Funktion `void read_command (<Funktionsparameter>);` aufgerufen werden, der entweder `next_command_short` oder `next_command_long` als Parameter übergeben wird, je nachdem, ob die Kurz- oder die Langform verwendet werden soll. Welche Kommandoform verwendet werden soll, wird der main-Funktion als Programmparameter übergeben.

Zur Erinnerung: Die main-Funktion hat in C eigentlich folgende Schnittstelle:

```
int main(int argc, char *argv[])
```

Der erste Parameter `argc` liefert die Anzahl der beim Programmaufruf mit übergebenen Parameter. Der zweite Parameter `argv` ist eine Liste von Texten, bei denen der erste Text (`argv[0]`) den Programmnamen enthält und alle weiteren Texte die Aufrufparameter. Beim Aufruf müssen die Parameter immer durch mindestens ein Leerzeichen getrennt sein. Rufen Sie Ihr Programm mit einem Argument 's' (→ Kurzform) bzw. 'l' (→ Langform) auf. Abhängig von diesem Wert wird der o.g. Funktion `read_command` entweder die Funktion `next_command_short` oder `next_command_long` übergeben.

Aufgabe 4 *Ein-/Ausgabe in Dateien (15 Punkte)*

Der Editor soll um zwei Kommandos `'w/<dateiname>'` und `'g/<dateiname>'` (bzw. `write_file "dateiname"` und `get_file "dateiname"`, falls Kommandos in Langform verwendet werden) erweitert werden. Diese Kommandos veranlassen den Editor, den Editor-Text in eine Datei zu schreiben bzw. von einer Datei zu lesen. Beide Funktionen sollen den gesamten Editor-Inhalt von dieser Datei einlesen bzw. in diese Datei ausgeben (ohne Zeilennummern!). Erweitern Sie dazu das Modul `in_out` um zwei Funktionen:

```
void read_from_file (char* filename);
void print_to_file (char* filename);
```

Vergessen Sie hierbei nicht, die Module `command`, `main` und `error_handling` geeignet zu erweitern.

Aufgabe 5 *Bitmaske (5 Punkte)*

Die Verwendung von Bitmasken soll exemplarisch anhand der Suchfunktion geübt werden. In vielen Fällen funktioniert eine Suche unabhängig von Klein-und Großbuchstaben. Sie können den Vergleich der Buchstaben 'A/a' bis 'Z/z' elegant unabhängig von der Groß- und Kleinschreibung durchführen, indem Sie geschickt Bits der ASCII-Zeichenkodierung maskieren.

Schreiben Sie Ihre Suchfunktion so um, dass die Buchstaben mit Hilfe von Bitmasken unabhängig von Groß- und Kleinschreibweise verglichen werden.

Projektideen für den Editor: GUI

Falls Sie Ihren Texteditor weiter ausbauen wollen, können Sie Ihn beispielsweise um eine graphische Benutzeroberfläche (Graphical User Interface, GUI) erweitern. Hierzu bietet es sich an die Bibliothek GTK (`https://www.gtk.org`) zu verwenden. GTK ermöglicht es moderne GUIs für Anwendungen für verschiedene Betriebssysteme zu entwickeln. Dazu werden verschiedene Komponenten zur Verfügung gestellt, die geeignet konfiguriert und programmiert werden müssen.

Ein Beispiel zur Verwendung dieser Bibliothek um einen Editor zu entwickeln finden Sie hier: `https://developer.gnome.org/gtk3/stable/ch01s04.html#id-1.2.3.12.5`