Laboratory for Parallel Programming
Prof. Dr. Felix Wolf

TECHNISCHE UNIVERSITÄT DARMSTADT

Daniel Lorenz, lorenz@cs.tu-darmstadt.de

Systemnahe und Parallele Programmierung (WS 17/18)

Übungsblatt 1

Die Lösungen der folgenden Aufgaben müssen bis zum 13. November, 2017 um 13:00 Uhr in Moodle eingereicht werden. Die Lösungen werden benotet.
Allen Programmieraufgaben muss ein Makefile beiliegen, das das Programm baut. Bündeln sie alle benötigten Dateien (Quellcode und Makefiles) in einem tar-Archiv.
Die erstellten Programme brauchen nicht auf dem Lichtenbergcluster zu laufen. Es genügt wenn sie auf ihrem privaten Rechner ausführbar sind.

Aufgabe 1

(4 Punkte) Erstellen sie ein Programm, das den Text `Hallo Welt!` auf dem Bildschirm ausgibt.

Aufgabe 2

(12 Punkte) Das Programm error.c enthält einen Kompilerfehler und zwei Laufzeitfehler. Kompilieren und debuggen sie das Programm. Beschreiben sie die drei Fehler. Falls sie mehr als drei Fehlerbeschreibungen liefern, werden nur die ersten Drei gewertet. Korrigieren sie das Programm.

```
1  #include <stdlib.h>
2
3  typedef struct
4  {
5    char* street;
6    int number;
7    int post_code;
8    char* city;
9  } address;
10
11 address*
12 create_address( char* street,
13     int number,
14     int post_code,
15     char* city )
16 {
17   address* new_address;
18   new_address->street    = street;
19   new_address->number    = number;
20   new_address->post_code = post_code;
21   new_address->city      = city;
22   return new_address;
23 }
24
25 address*
26 duplicate_address( address orig )
27 {
28   address new_address = orig;
29   return &new_address;
30 }
```

```
31
32 int main()
33 {
34   address* a1 = create_address( "Mornewegstr.", 30, 64293, "Darmstadt
       " );
35   address* a2 = duplicate_address( &a1 );
36   free( a1 );
37   free( a2 );
38 }
```

Aufgabe 3

(34 Punkte) In dieser Aufgabe soll eine doppelt verlinkte Liste implementiert werden, die einen `char*` als Datenelement speichert. Eine doppelt verlinkte Liste ist eine Datenstruktur, bei der jedes Element einen Zeiger auf das folgende und das vorhergehende Element speichert. Der Zeiger zum nächsten Element des letzten Listenelements und der Zeiger zum vorhergehenden Element der ersten Elements sind `NULL`. Eine Beispiel ist in Abbildung 1 zu sehen.

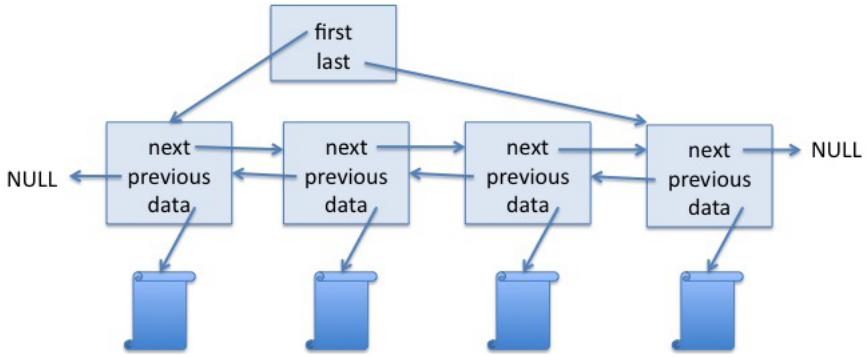

Figure 1: Eine doppelt verlinkte Liste.

Für die Implementierung der doppelt verlinkten Liste sollen zwei Datenstrukturen definiert werden:

- `LinkedListNode`

  Diese Struktur repräsentiert ein Listenelement. Es enthält einen Zeiger auf das vorhergehende und das nächste Listenelement und einen `char*` Zeiger auf die Daten.

- `LinkedList`

  Diese Struktur repräsentiert die gesamte Liste und enthält je einen Zeiger auf der erste und auf das letzte Element der Liste. Ist die Liste leer, sind beide Zeiger `NULL`.

Des Weiteren sollen Funktionen zum Zugriff und Bearbeiten der Liste erstellt werden. Die entsprechenden Funktionen sind in der Datei `list.h` deklariert.

- `LinkedList* LinkedList_create()`

  Diese Funktion gibt eine neue doppelt verlinkte Liste zurück. Sie allokiert den nötigen Speicher und initialisiert alle Felder mit `NULL`.

- `void LinkedList_append( LinkedList* list, char* data )`

  Diese Funktion erzeugt ein neues Listenelement, dass den Zeiger `data` speichert, und hängt das neue Listenelement am Ende der Liste an.

- `void LinkedList_delete( LinkedList* list )`

  Diese Funktion zerstört eine doppelt verlinkte Liste. Sie gibt auch den Speicher aller enthaltenen Listenelemente und der Datenelemente frei.

- `LinkedListNode* LinkedList_getFirst( LinkedList* list )`

  Diese Funktion gibt einen Zeiger auf das erste Element der Liste zurück. Ist die Liste leer, wird `NULL` zurückgegeben.

- `LinkedListNode* LinkedList_getLast( LinkedList* list )`

  Diese Funktion gibt einen Zeiger auf das letzte Element der Liste zurück. Ist die Liste leer, wird `NULL` zurückgegeben.