Laboratory for Parallel Programming
Prof. Dr. Felix Wolf

TECHNISCHE UNIVERSITÄT DARMSTADT

Alexandru Calotoiu, calotoiu@cs.tu-darmstadt.de

Systemnahe und Parallele Programmierung (WS 18/19)

Übungsblatt 1

Die Lösungen der folgenden Aufgaben müssen bis zum 5. November 2018 um 13:30 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 müssen nicht auf dem Lichtenberg Cluster 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 zwei Kompilerfehler und einen 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* name;
6    int matriculation_number;
7    int semester;
8    char* program;
9  } student;
10
11 student*
12 create_student( char* name,
13     int matriculation_number,
14     int semester,
15     char* program )
16 {
17   student* new_student;
18   new_student->name     = name;
19   new_student.matriculation_number     = matriculation_number;
20   new_student.semester = semester;
21   new_student->program  = program;
22   return new_student;
23 }
24
25 int main()
26 {
27   student* s1 = create_student( "Max Mustermann", 424242, 1, "
       Computer Science" );
28   free( s1 );
29 }
```

Aufgabe 3

(34 Punkte) In dieser Aufgabe soll eine sortierte verlinkte Liste implementiert werden, die einen `int` als Datenelement speichert. Eine sortierte verlinkte Liste ist eine Datenstruktur, bei der jedes Element einen Zeiger auf das folgende Element speichert, und die Elemente nach einer bestimmte Regel sortiert sind, in diesem Fall aufsteigend. Der Zeiger zum nächsten Element des letzten Listenelements ist `NULL`. Ein Beispiel ist in Abbildung 1 zu sehen.

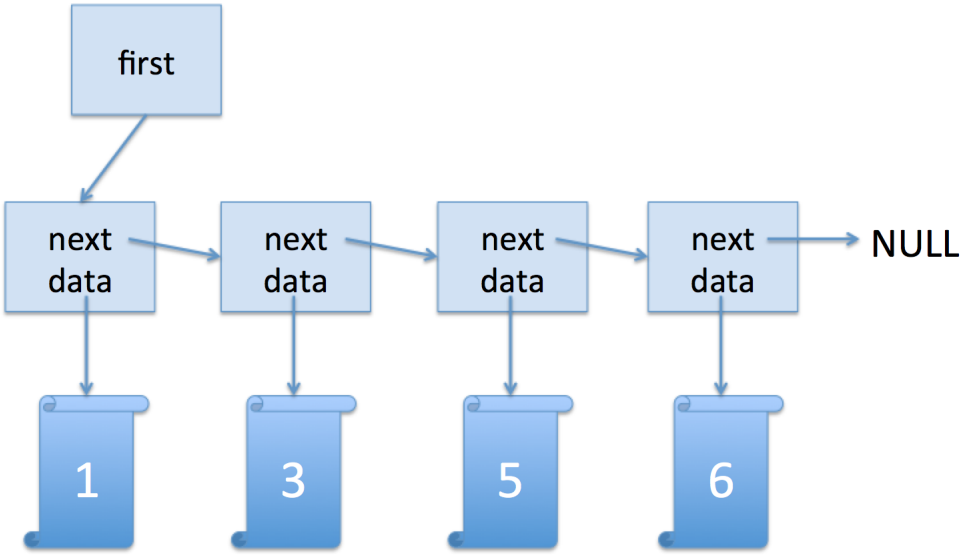

Abbildung 1: Eine sortierte verlinkte Liste.

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

- `SortedLinkedListNode`

  Diese Struktur repräsentiert ein Listenelement. Es enthält einen Zeiger auf das nächste Listenelement und einen `int`

- `SortedLinkedList`

  Diese Struktur repräsentiert die gesamte Liste und enthält einen Zeiger auf das erste Element der Liste. Ist die Liste leer, ist dieser Zeiger `NULL`.

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

- `SortedLinkedList* SortedLinkedList_create()`

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

- `void SortedLinkedList_addToList( SortedLinkedList* list, int data )`

  Diese Funktion erzeugt ein neues Listenelement, dass den Integer `data` speichert, und hängt das neue Listenelement in der Liste an, so dass die Liste aufsteigend sortiert bleibt.

- `void SortedLinkedList_delete( SortedLinkedList* list )`

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

- `SortedLinkedListNode* SortedLinkedList_getSmallest( SortedLinkedList* list )`

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