Institut für Informatik
Prof. Dr. Stefan Conrad

Sommersemester 2020
Übung 7

Hardwarenahe Programmierung

Gruppe 14 (Lars)

Bei diesem Abschlussprojekt bearbeiten Sie eine größere C-Programmieraufgabe, in der Sie Ihre gesammelten Fähigkeiten anwenden sollen. Die Aufgabe ist erstmals nicht kleinschrittig für Sie vorbereitet. Fertigen Sie zuerst einen groben Ablaufplan in Form von Kommentaren oder Funktionsaufrufen in Ihrer Codedatei an. Ersetzen Sie die Kommentare nach und nach durch funktionsfähigen Code.

Wichtig:

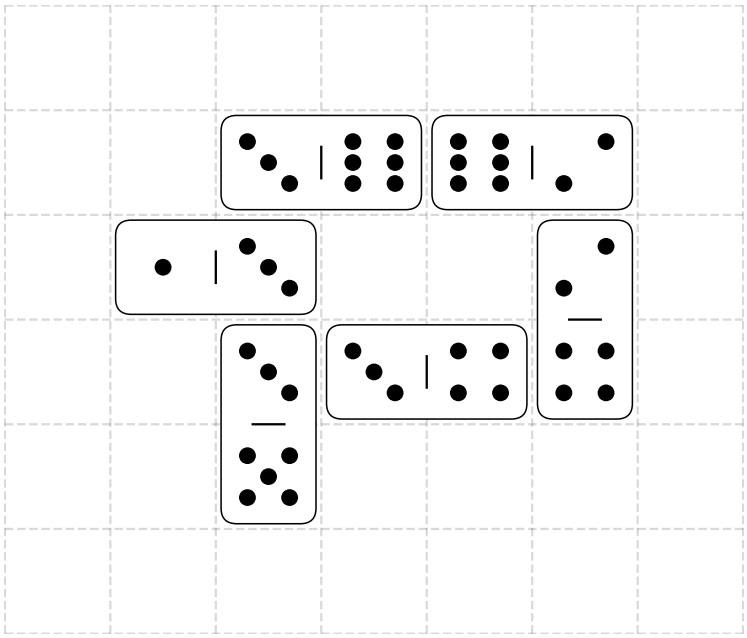
- *Die Abgabe ist diesmal nicht um 23:55 Uhr, sondern um 8:00 Uhr.*
- *Sie müssen das Abschlussprojekt Ihrem Tutor in einem Einzelgespräch erklären und dazu einige Fragen beantworten. Testen Sie* ***vorher*** *Ihre technische Ausstattung (Webcam/Mikrofon oder Smartphone) und halten Sie einen* ***Lichtbildausweis*** *bereit.*
- *Tests können nur beweisen, dass ein Programm Fehler hat. Das Gegenteil zu zeigen, also das ein Programm immer fehlerfrei arbeitet, ist extrem schwer und manchmal sogar unmöglich. Wenn die Tests also bei Ihnen zufällig durchlaufen, aber bei uns nicht, zum Beispiel, weil Variablen nicht initialisiert oder Strings nicht terminiert wurden, dann ist das Programm definitiv fehlerhaft.*

Aufgabe 1 *Abschlussaufgabe – Dominos*

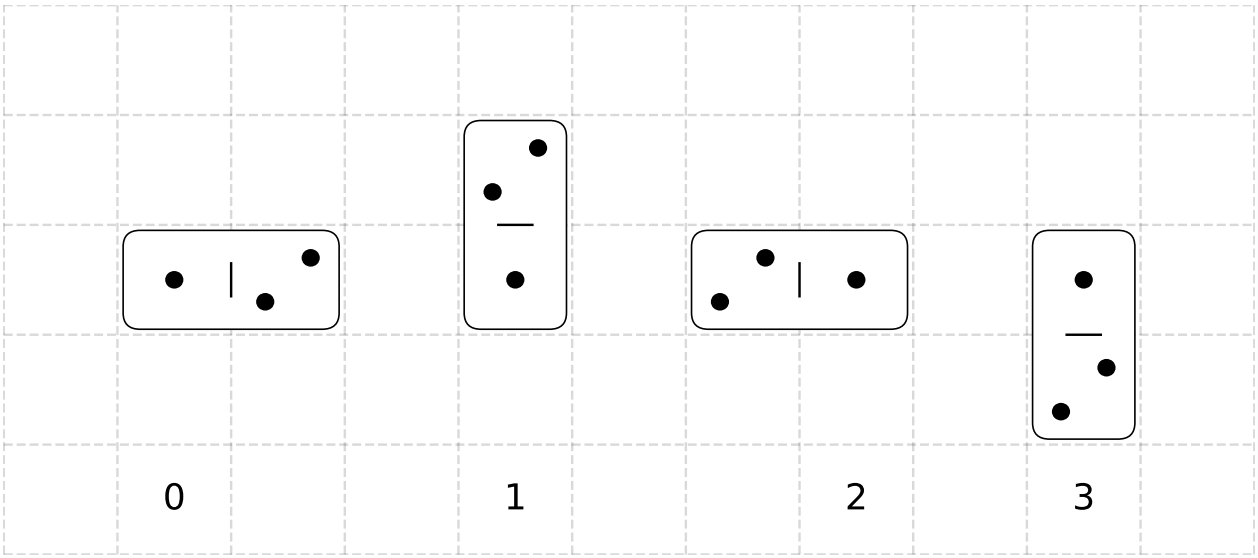
In dieser Aufgabe geht es darum, Dominosteine aneinander zu legen.

- Dominos bestehen aus zwei *Hälften*. Jedes Hälfte hat ein bis sechs *Augen*.
- Dominos dürfen nur aneinander angelegt werden, wenn die Augenzahl aller benachbarter Hälften übereinstimmt.

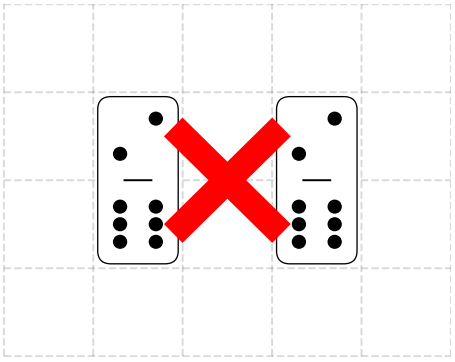
Beispiel:



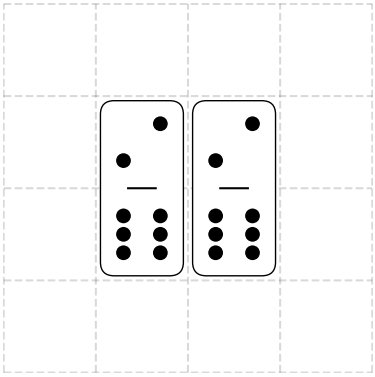
- Dominos können in vier Richtungen orientiert sein, hier durchnummeriert von 0 bis 3.



- Nach der Platzierung des ersten Dominosteins muss jeder weitere Stein direkt an mindestens einen weiteren Stein angrenzen. Freischwebende Dominosteine sind also nicht erlaubt.



- Steine dürfen auch direkt nebeneinander liegen, wenn die Augenzahlen auf beiden Hälften übereinstimmen.



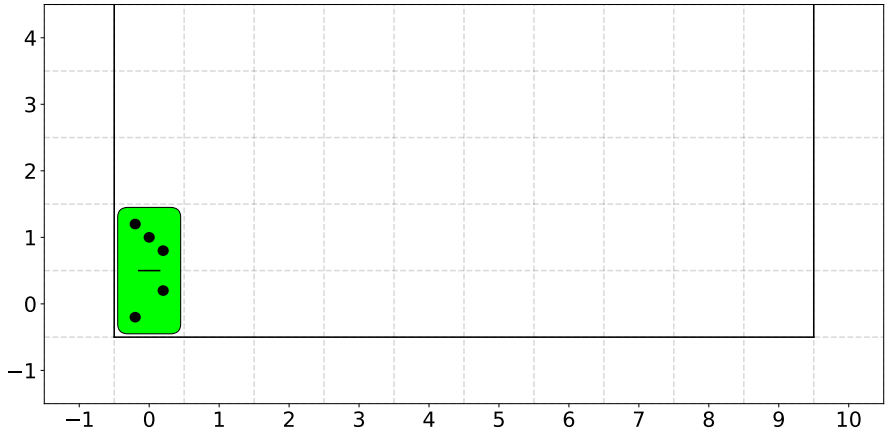
Aufgabenstellung:

```
3 2 3
1 3 1
6 1 2
3 5 0
5 4 1
5 5 2
6 2 2
4 3 1
6 6 2
4 4 3
```

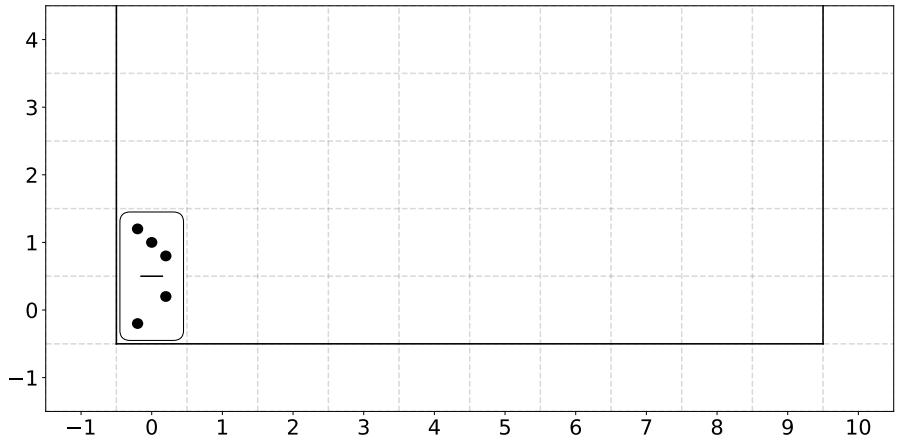
- Ihr Programm soll eine Datei entgegennehmen, in der eine Liste von Dominosteinen mit dazugehöriger Augenzahl sowie fest vorgegebener Ausrichtung enthalten sind.
- Diese Dominosteine sollen nacheinander auf einem Spielfeld mit einer maximalen Breite von 10 Kästchen platziert werden, das nach unten geschlossen und nach oben offen ist.
- Wählen Sie für jeden Stein die Position mit niedrigster y-Koordinate, also möglichst weit unten auf dem Spielfeld. Falls es mehrere solcher Positionen geben sollte, wählen sie von diesen Positionen die mit kleinster x-Koordinate, also möglichst weit links.
- Abschließend soll das Spielfeld in eine Ausgabedatei geschrieben werden.

Beispiel:

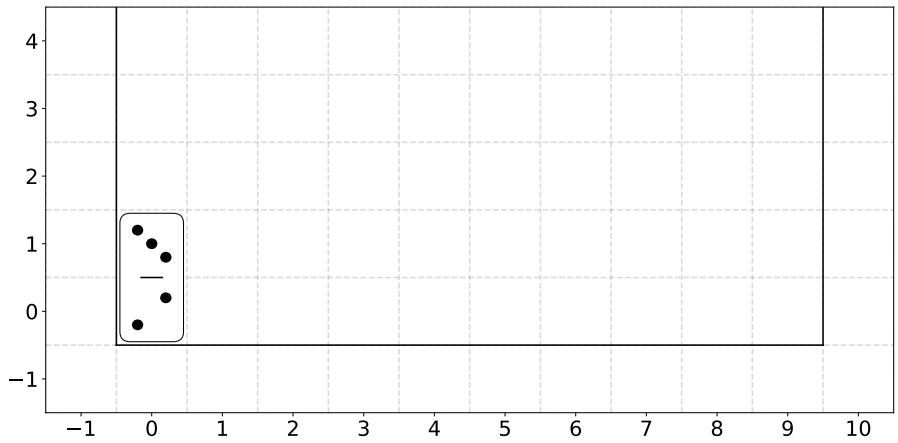
Zustand nach Einfügen von Domino mit Augen $(3, 2)$ und Richtung 3.



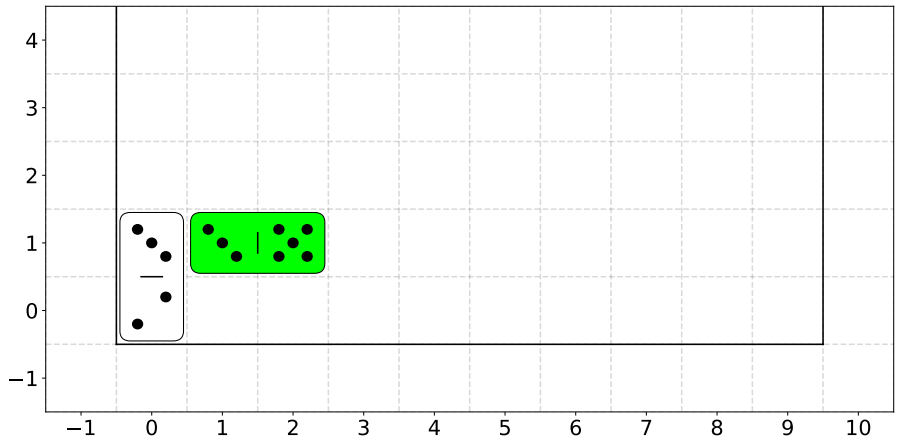
Zustand nach Einfügen von Domino mit Augen $(1, 3)$ und Richtung 1. Domino kann nicht angelegt werden.



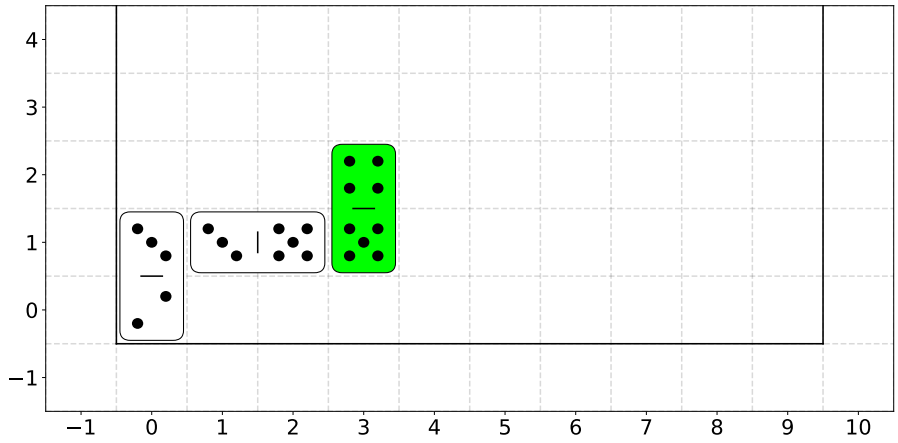
Zustand nach Einfügen von Domino mit Augen $(6, 1)$ und Richtung 2. Domino kann nicht angelegt werden.



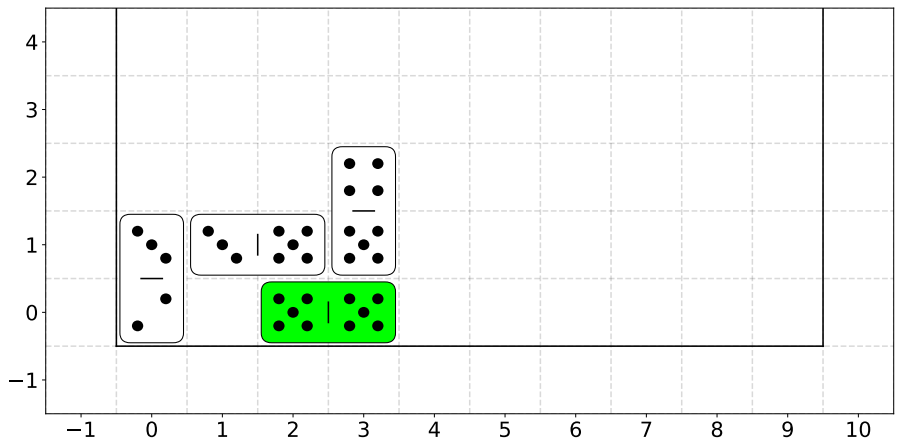
Zustand nach Einfügen von Domino mit Augen $(3, 5)$ und Richtung 0.



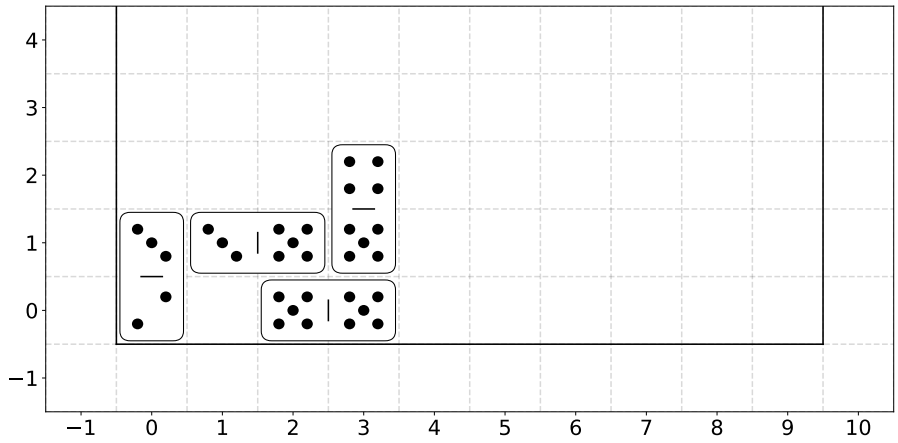
Zustand nach Einfügen von Domino mit Augen $(5, 4)$ und Richtung 1.



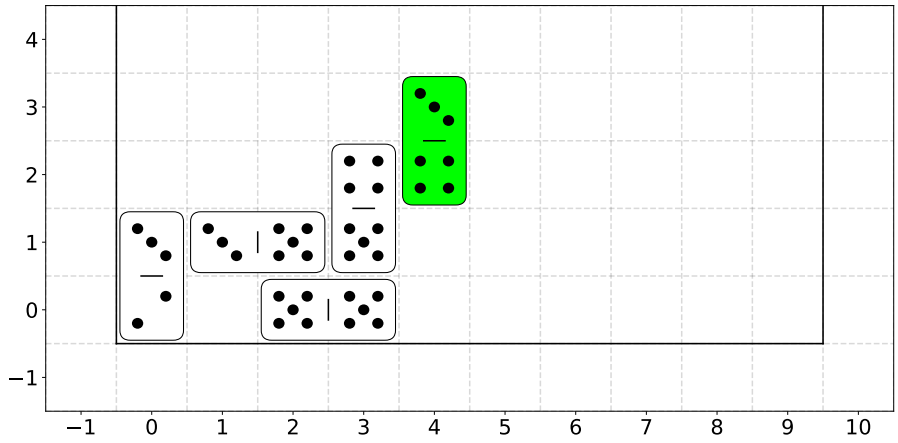
Zustand nach Einfügen von Domino mit Augen $(5, 5)$ und Richtung 2.



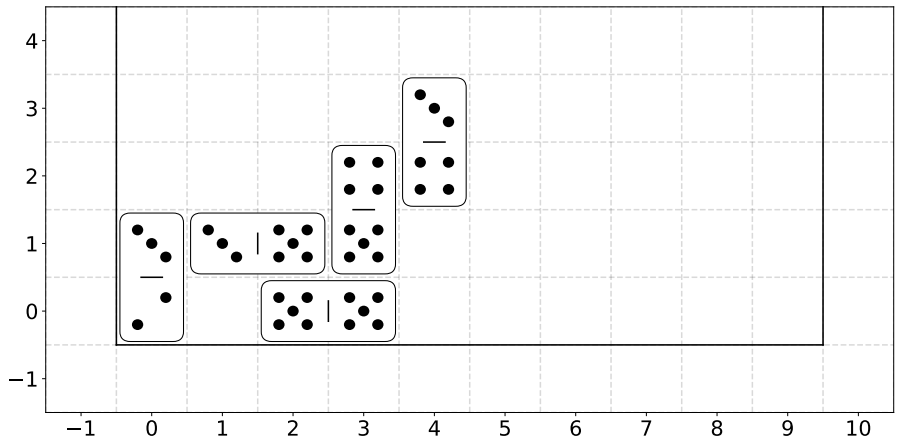
Zustand nach Einfügen von Domino mit Augen $(6, 2)$ und Richtung 2. Domino kann nicht angelegt werden.



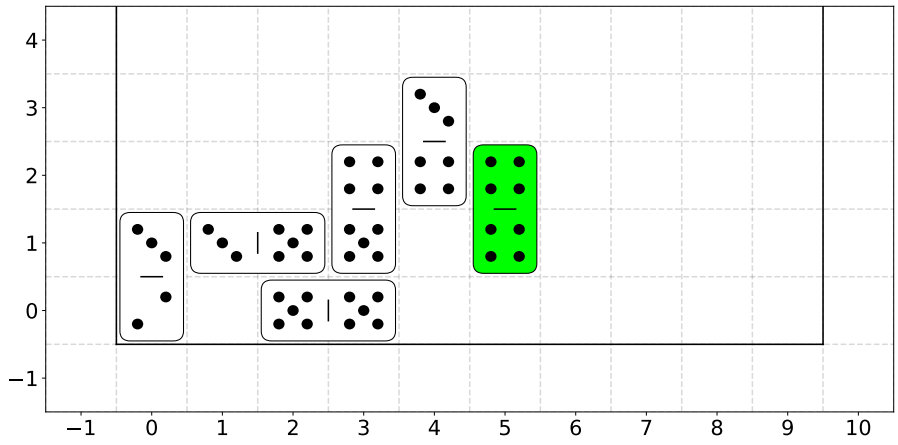
Zustand nach Einfügen von Domino mit Augen $(4, 3)$ und Richtung 1.



Zustand nach Einfügen von Domino mit Augen $(6, 6)$ und Richtung 2. Domino kann nicht angelegt werden.



Zustand nach Einfügen von Domino mit Augen $(4, 4)$ und Richtung 3.



Erstellen des Programms: Nennen Sie Ihr Programm `dominos`. Wir haben Ihnen ein Makefile zur Verfügung gestellt, mit dem Sie Ihr Programm kompilieren, ausführen und testen sollen.

```
3 2 3
1 3 1
6 1 2
3 5 0
5 4 1
5 5 2
6 2 2
4 3 1
6 6 2
4 4 3
```

Prüfen des Programms: Prüfen Sie mit `make diff` die Ausgabe Ihres Programms und prüfen Sie mit `make valgrind` die Speicherverwaltung. Mit `make run` können Sie beides überprüfen. Wir haben diese targets im Makefile bereits für Sie erstellt.

Eingabe: Die Eingabe erfolgt über eine Text-Datei mit drei Spalten und jeweils einer Zeile pro Dominostein. In jeder Zeile steht zuerst die Augenzahl der ersten Dominostein-Hälfte, dann ein Leerzeichen, dann die Augenzahl der zweiten Hälfte, wieder ein Leerzeichen, die Ausrichtung und schließlich ein Zeilenumbruch (`\n`).

Ausgabe: Die Ausgabe des Spielfelds erfolgt ebenfalls über eine Text-Datei. Hierbei soll für jede Hälfte eines platzierten Dominosteins die zugehörige Augenzahl an der dazugehörigen Spielfeldposition ausgegeben werden.
Die Ausgabe muss jeden Dominostein vollständig beinhalten, darf aber nicht größer sein. Leere Felder werden mit einem Leerzeichen angezeigt. Nach jeder nicht-leeren Zeile folgt ein Zeilenumbruch, auch nach der letzten Zeile.
Auf der rechten Seite sehen Sie die Textausgabe für das vorherige Beispiel.

```
   3
  444
3355 4
2 55
```

Das Programm soll im Fehlerfall eine kurze, aussagekräftige Meldung ausgeben und folgende **Rückgabewerte** haben:

- 0, wenn alles ordnungsgemäß funktioniert.
- 1, wenn nicht genau zwei zusätzliche Parameter übergeben werden.
- 2, wenn die Ein- oder Ausgabedatei nicht geöffnet werden konnte oder es Fehler beim Lesen oder Schreiben von Dateien gab.
- 3, wenn die Eingabedatei nicht dem vorgegebenen Format entspricht.

Aufruf des Programms: Das Programm bekommt beim Aufruf die Namen der Eingabe- und Ausgabedatei übergeben. Dabei steht die Eingabedatei immer vor der Ausgabedatei:

```
./dominos eingabe.txt ausgabe.txt
```

Wichtige Anforderungen an das Programm:

- Der Speicher ihrer Datenstrukturen muss dynamisch verwaltet werden.
- Insbesondere darf dieser also nicht statisch alloziert werden, d.h. es darf keine feste Obergrenze geben, ab der Ihr Programm plötzlich nicht mehr funktioniert.
- Die Datenstrukturen müssen dynamisch wachsen und sich der im Programmablauf steigenden Spielfeldgröße anpassen.
- Vor Beendigung des Programms muss der gesamte Speicher wieder freigegeben werden und geöffnete Dateien müssen geschlossen werden.
- Valgrind darf weder Speicherzugriffsverletzungen noch Speicherlecks oder sonstige Fehler melden.
- Alle Funktionen sowie Datenstrukturen müssen in einer separaten Headerdatei deklariert werden.
- Ihr Algorithmus soll mit den von Ihnen allozierten Speicherbereichen arbeiten und Dateizugriffe nur für das Einlesen in und Schreiben aus dem Speicher verwenden.
- Ihr Programm darf nie mehr als eine Datei gleichzeitig geöffnet haben.
- Die Eingabe- und Ausgabe-Dateien dürfen nur einmal geöffnet und geschlossen werden.
- Das Programm muss alle Tests fehlerfrei bestehen, d.h. `make run` darf keine Fehler liefern.
- Testen Sie Ihr Programm selbst mit `valgrind`, zum Beispiel um eventuelle Sonderfälle zu überprüfen.
- Ein Testdurchlauf für alle Testdateien darf nicht länger als fünf Minuten dauern. Selbst eine mittelmäßige Implementierung benötigt allerdings nur Sekundenbruchteile für `make diff` und einige Sekunden für `make valgrind`.
- Achten Sie auf die Einhaltung der Coderichtlinien im Lernmodul. Insbesondere müssen Sie Ihren Code in Funktionen gliedern und globale Variablen dürfen nicht verwendet werden.
- Zum Bestehen müssen Sie Ihren Code in einem Einzelgespräch erklären und Fragen beantworten. Vereinbaren Sie dazu mit Ihrem Tutor einen Termin.

Viel Erfolg!