4. Übung

Allgemeine Hinweise

- Verwenden Sie für die Kompilierung g++ mit den Parametern:
  -std=c++11 -Wall -Wextra
- Achten Sie darauf, dass Sie keine Compiler-Warnungen erhalten
- Halten Sie sich an die Best Practices bzgl. Clean Code
- Es ist nicht erlaubt, externe Libraries (außer Standardlibraries) zu verwenden
- Legen Sie für jede der drei Aufgaben einen Unterordner an: u4_1, u4_2 und u4_3
- Erstellen Sie ein Makefile für die Kompilierung
  - Das Output-File soll "program" heißen
- Entfernen Sie vor dem Hochladen die ausführbaren Kompilate, sowie die Object-Files (*.o)
- Erstellen Sie ein komprimiertes Archiv (.zip oder .tar.gz), das die drei Unterordner enthält und laden Sie dies rechtzeitig auf Moodle
- Spätester Abgabezeitpunkt: **Mi., 26.06.2024, 23:59**

Weitere Hinweise

Die Aufgaben bauen teilweise aufeinander auf.
Es ist sinnvoll, sich zu Beginn alle drei Aufgaben anzusehen und Gemeinsamkeiten festzustellen (z. B. Datei-Einlesen, ggfs. Datenstrukturen etc.).
So können Teile des Programms bspw. in eine zweite CPP-Datei ausgelagert und bei den folgenden Aufgaben wiederverwendet werden.

Aufgabe #1 – Domino-Steine

Erstellen Sie ein Programm, das Anzahl an Domino-Steinen in der längsten Kette ausgibt.

Ein Domino-Stein besteht aus zwei Zahlen (1-6), eine Kette ist nur möglich, wenn die erste Zahl des nächsten Stein gleich der zweiten Zahl es vorhergehenden Steines ist.
Beispiel: [1 3] [3 4] [4 2] ist eine gültige Kette mit der Länge 3.

Die Domino-Steine werden aus einer Datei eingelesen, deren Pfad als Kommandozeilen-Argument beim Programmaufruf an das Programm übergeben wird.

Jede Zeile der Datei steht für einen Stein, dessen Ziffern durch ein Leerzeichen getrennt sind. Beispiel-Inhalt:

```
1 3
3 4
4 2
```

In einer Datei können auch mehrere Ketten vorhanden sein.
Die Steine dürfen dabei nicht gedreht werden
(z. B. [1 3] [3 4] wäre eine Kette, aber [1 3] [4 3] nicht).

Als Ausgabe ist nur die Anzahl der Steine in der längsten Kette nötig.
Tipp: Für Aufgabe #3 können Sie bestimmte Funktionen wiederverwenden, überlegen Sie, diese in ein externes cpp-File auszulagern, um diese leichter wiederverwenden zu können.

Beispiele

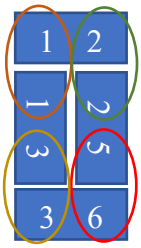
Dateiinhalt	1 2 2 4 1 5 6 5 5 2 2 3 3 4 1 4	5 2 3 1	3 5 1 4 4 2 3 1 2 2 2 4 4 6
Lösung/Ausgabe	4	1	3

Aufgabe #2 – Domino-Steine #2

Erstellen Sie ein Programm, das feststellt, ob eine definierte Anordnung von Domino-Steinen möglich ist.

Das Programm soll wie bei Aufgabe #1 beliebige Domino-Steine aus einer Datei lesen, deren Pfad als Kommandozeilen-Argument beim Programmaufruf an das Programm übergeben wird.

Es soll berechnet werden, ob folgende Anordnung von Domino-Steinen möglich ist:



Dabei müssen angrenzende Steine gleiche Zahlenwerte haben

Allerdings soll berechnet werden, ob durch Vertauschen und Drehen der Domino-Steine die angegebene Form gebildet werden kann.

Sie können davon ausgehen, dass sich exakt 4 Steine in der Datei befinden.

Beispiele

Dateiinhalt	1 2 4 1 4 3 3 2	2 2 3 1 1 2 1 3
Lösung/Ausgabe	JA (kann gebildet werden)	Nein (kann nicht gebildet werden)
Notiz (nicht in der Ausgabe enthalten)		

Aufgabe #3 – Domino-Steine #3

Erstellen Sie ein Programm, das die längstmögliche Kette an Domino-Steinen berechnet.

Das Programm soll wie bei Aufgabe #1 beliebige Domino-Steine aus einer Datei lesen, deren Pfad als Kommandozeilen-Argument beim Programmaufruf an das Programm übergeben wird.

Allerdings soll berechnet werden, wie durch Vertauschen und Drehen der Domino-Steine die längstmögliche Kette gebildet werden kann.

Verwenden Sie hierfür einen Algorithmus Ihrer Wahl (z.B. Brute-Force, Greedy, etc.).
Sie können davon ausgehen, dass sich nicht mehr als 15 Steine der Datei befinden.

Geben Sie das Ergebnis folgendermaßen aus:
[A1 A2] [B1 B2] [C1 C2] … wobei A1, A2, B1 … für die Zahlen an den Steinen stehen.
Beachten Sie, dass für jedes Beispiel mehrere Lösungen richtig sein können, es reicht, wenn Sie eine der möglichen Lösungen ausgeben.

Beispiele

Dateiinhalt	1 2 3 2 1 4 5 6 4 6	5 2 3 1 2 1 4 1
Lösung/Ausgabe	[5 6][6 4][4 1][1 2][2 3]	[5 2][2 1][1 3]