| SWO3 | Übung zu Softwareentwicklung mit klassischen Sprachen und Bibliotheken 3 | WS 2018/19, Angabe 7 |
|---|---|---|

☐ Gruppe 1 (J. Heinzelreiter)

☐ Gruppe 2 (M. Hava) Name: ____Niklas Vest_____ Aufwand [h]: ___7_____

☒ Gruppe 3 (P. Kulczycki) Übungsleiter/Tutor: _____ Punkte: _____

| Beispiel | Lösungsidee (max. 100%) | Implement. (max. 100%) | Testen (max. 100%) |
|---|---|---|---|
| 1 (30 P) | 100% | 100% | 100% |
| 2 (50 P + 20 P) | 100% | 100% | 100% |

## Beispiel 1: Autobau (`src/wmb`)

Sie werden von einem großen Münchner Autobauer beauftragt, die Verwaltung von Autobestandteilen softwareseitig zu unterstützen. Für ein Auto werden die folgenden Daten verwaltet: Typ, Farbe, Seriennummer, Produktionsdatum, Produktionsort, Getriebeart, Antriebsart, Höchstgeschwindigkeit und Gewicht. Für einen Motor werden folgende Daten benötigt: Motornummer, Treibstoffart, Leistung, Normverbrauch und Produktionsdatum. Für die Räder eines Autos werden Felgendurchmesser, Produktionsjahr, Geschwindigkeitsindex und Hersteller verwaltet.

Modellieren Sie mindestens die Klassen `auto`, `motor` und `rad` und implementieren Sie diese als C++-Klassen. „Bauen" Sie Autos mit unterschiedlichen Konfigurationen zu Testzwecken zusammen. Es muss auch eine einfache Ausgabe von Fahrzeugdaten mittels `operator<<` möglich sein. Testen Sie ihre Implementierung ausführlich, lesen Sie alle benötigten Testdaten mittels `operator>>` ein.

## Beispiel 2: ADT „Graph" (`src/adt`)

**(a)** Der von Ihnen in Übung 5 implementierte ADT für die Darstellung von ungewichteten gerichteten Graphen durch eine Adjazenzmatrix soll als Ausgangspunkt für eine objektorientierte Implementierung dienen, die dann allerdings gewichtete gerichtete Graphen repräsentieren kann. Gute Kandidaten für die zu implementierenden Klassen sind `vertex_t` und `graph_t`. Instanzen der Klasse `vertex_t` sind benannte Knoten, wobei die Namen Zeichenketten (`std::string`) sind. Die Klasse `graph_t` muss mindestens die folgende Funktionalität aufweisen:

```
handle_t       add_vertex (vertex_t vertex);   // moves 'vertex' into graph
void           add_edge   (handle_t const from, handle_t const to, int const weight);
std::ostream & print      (std::ostream & out) const;
```

Fügen Sie Ihren Klassen weitere notwendige Methoden und Datenkomponenten hinzu und testen Sie Ihre Implementierung ausführlich. Verwenden Sie dafür Streams zusammen mit `operator<<` und `operator>>`.
**Anmerkung:** Die Klasse `handle_t` identifiziert einen Vertex in einem Graphen eindeutig.

**(b)** Es gibt einen interessanten gierigen (*greedy*) Algorithmus zur Berechnung des kürzesten Wegs zwischen zwei Knoten in einem Graphen. Recherchieren Sie in der Algorithmenliteratur, suchen Sie nach dem Dijkstra-Algorithmus (*single-source shortest path*). Implementieren Sie in Ihrer Klasse `graph_t` eine Methode

```
int shortest_path (handle_t const from, handle_t const to) const;
```

die die Länge des kürzesten Wegs zwischen den beiden Knoten `from` und `to` nach dem Verfahren von Dijkstra ermittelt.