Babeș-Bolyai Universität, Cluj Napoca
Fakultät für Mathematik und Informatik
Objekt-Orientierte Programmierung, 2022

Labor 3

Diskussion: Kalenderw. 14, **Abgabe:** Kalenderw.15, **Gewicht:** 6/50

Eine **Dependency-Structure-Matrix** ist eine Darstellung für Systeme: deren Elemente und Anhängigkeiten (siehe Wikipedia). Beispielsweise kann man damit modellieren welche Teile von einem Auto miteinander verbunden sind, oder in einem Programm welche Klassen welche einander verwenden.

Schritt 1: Analysiere ein beliebiges System das aus 7 bis 12 Elementen besteht und notiere die Verbindungen zwischen den Elementen als Text! (→ PDF)

Schritt 2: Stelle das System mit Hilfe einer Dependency-Structure-Matrix dar (→ PDF)

Schritt 3: Implementiere eine Klasse für DSMs und erstelle die Matrix für dein Beispiel.

Die Klasse DSM soll die Namen der Elemente als dynamisches Array und deren Verbindungen in Form eine Adjazenzmatrix speichert. Implementieren Sie die Matrix mithilfe von einem dynamischen **2-dimensionales Array**.

1. Implementieren Sie zwei **Konstruktoren**

   ```
   DSM(int elementCount)
   DSM(string[] elementNames, int elementCount)
   ```

   Und fügen sie auch den entsprechenden **Destruktor** und **Kopierkonstruktor** hinzu.

2. Implementieren Sie die **Methoden**

   a. `int size()` gibt die Anzahl der Größe der Matrix.

   b. `string getName(int index)`

   c. `void setElementName(int index, string elementName)`

   Der Zugriff außerhalb des Gültigkeitsbereichs sollte eine Ausnahme auslösen.

   d. `void addLink(string fromElement, string toElement, int weight)`

   Wiederholtes Hinzufügen einer Verbindung zwischen Elementen A und B sollte die vorherige überschreiben. Das Hinzufügen einer Verbindung mit unbekanntem Elementnamen soll dieses zur Matrix hinzufügen.

   e. `void deleteLink(string fromElement, string toElement).`

3. Implementieren Sie die folgenden Analyse-Methoden

   a. `bool hasLink(string fromElement, string toElement)`

   b. `int linkWeight(string fromElement, string toElement)`

   c. `int countToLinks(string elementName)`

   d. `int countFromLinks(string elementName)`

   e. `int countAllLinks()`

Labor 3

Diskussion: Kalenderw. 14, **Abgabe:** Kalenderw.15, **Gewicht:** 6/50

Bonus A (1 punkt): Die Gewichtseigenschaft wird als Template-Parameter implementiert.

ODER

Bonus B (1 punkt): Es gibt Methoden mit denen die Matrix in eine Datei geschrieben und aus einer Datei gelesen wird. Die Datei besteht aus maximal N^2 Zeilen des folgenden Formats (CSV):

```
fromElementName, weight, toElementName
```

Anforderungen:

1) Implementierung der Schritte 1, 2 und 3 der Aufgabe
2) Die Lösung besteht aus 5 Dateien:
   - **`L3_Nachname_Vorname_DSM.pdf`** enthält die Ergebnisse von Schritt 1 und 2
   - **`L3_Nachname_Vorname_DSM.h`** enthält die Definition der Klasse (Methodensignaturen müssen exakt mit den oben angegebenen übereinstimmen.)
   - **`L3_Nachname_Vorname_DSM.cpp`** enthält die Implementierung
   - **`L3_Nachname_Vorname_DSM_test.cpp`** enthält **`main()`** mit Unit-Tests für alle Methoden, und den Code für deine Beispiel-DSM.
   - Eine Skriptdatei compile_run.(bat oder .sh) mit den Befehlen, mit denen das Programm kompiliert wird (mit GnuCompiler: **g++ -std=c++20**) und ausgeführt wird.

Bewertungskriterien:

1) Der Programmcode vollständig sein und muss fehlerfrei kompilierbar sein. (2,5 Punkte)
2) Das Programm muss ausführbar sein und das korrekte Ergebnis liefern. (2,5 Punkte)
3) Du musst den Code erklären können. (2,5 Punkte)
4) Der Code muss gut lesbar sein. (2,5 Punkte)