Bonusaufgaben zum C/C++-Praktikum

Speicherverwaltung

TECHNISCHE UNIVERSITÄT DARMSTADT

Übungsblatt 2

Hinweise zur Abgabe:
verwendet als Grundlage für die Bearbeitung die im Moodle bereitgestellte Vorlage. Beachtet auch die Hinweise auf dem ersten Bonuszettel.

Aufgabe 2.1: `[S]` Food n Stuff (5 Punkte)

In dieser Aufgabe ist im Template eine Klasse `Food` vorgegeben, mit der eine Speisekarte gespeichert werden kann. Die Aufgaben a) & b) sind dabei keine Methoden der Klasse selbst.

2.1a) Speichern (2 Punkte)

Um bei einer Änderung der Speisekarte das Programm nicht neu kompilieren zu müssen, soll diese in einer eigenen Datei gespeichert werden. Wir werden dazu das CSV-Format verwenden. CSV-Dateien werden Zeilenweise eingelesen und in jeder Zeile werden die einzelnen Felder durch einen Separator getrennt. Wir verwenden als Separator das Semikolon (`;`) und die Felder sind in der Reihenfolge: Bestellnummer, Bezeichnung, Preis (ohne Währung).

Beispiel:

```
12;Pizza;9.5
17;Kuchen;3.8
```

Implementiere die Methode

```
void speichern(const std::string &dateiname, const std::vector<Food> &speisen);
```

die einen `std::vector<Food>` sowie den Dateinamen zu der CSV-Datei erwartet.

Schaue dir hierzu die Methoden der Klasse `std::fstream` an. Mithilfe des entsprechenden Konstruktors lässt sich eine CSV-Datei öffnen und anschließend dem assoziierten fstream-Object mit dem << operator neuen Text hinzuzufügen. Beachte, dass der Stream am Ende geschlossen werden muss, um den generierten Inhalt tatsächlich in die Datei zu schreiben und diese wieder freizugeben.

2.1b) Parsen (3 Punkte)

Die zuvor erstelle CSV-Datei muss natürlich auch wieder geladen werden können. Implementiere dazu die entsprechende Funktion. Wenn du Probleme beim Parsen hast, (weil Bspw. die CSV-Datei nicht korrekt ist) werfe eine Exception vom Typ `std::runtime_error` mit einer groben Beschreibung des Fehlers.

```
void laden(const std::string &dateiname, std::vector<Food> &speisen);
```

Schaue dir hierzu am besten die Funktion `getline` an.

Aufgabe 2.2: `[S]` Hash table (8 Punkte)

In der Übung habt ihr bereits die LinkedList kennengelernt. Eine weitere weit verbreitete Datenstruktur ist eine so genannte (Hash)Map oder auch Hash Table genannt. Diese sind ähnlich wie Arrays nur, dass statt eines Indexes ein Schlüssel (engl. `key`) verwendet wird, welcher kein Integer sein muss. Hier wird als Schlüssel der Typ `std::string` verwendet.

Wie die Klasse auszusehen hat ist bereits vorgegeben, du bist nun für die Implementierung verantwortlich. Lies dir am besten zuerst alle Teilaufgaben durch, damit klar wird wofür die Struktur `MapElement` und der Typalias `MapBucket` verwendet wird und was die Aufgaben der einzelnen Methoden sind.

```
namespace cppp {
    struct Item {
        int anzahl;
        std::string bezeichnung;
    };

    class Map {
    public:
        Map(const std::size_t size);
        Map(const Map& other) = delete;
        ~Map();

        void insert(const std::string& key, const std::vector<Item>& order);
        std::vector<Item> get(const std::string& key);
        void remove(const std::string& key);

    private:
        const std::size_t size;
        std::size_t calcHash(const std::string& key);

        struct MapElement {
            std::string key;
            std::vector<Item> value;
        };
        using MapBucket = std::vector<MapElement>;

        MapBucket* arr;
    };
}
```

Anmerkung für die Interessierten: Wir verwenden hier das so genannte Separate Chaining mit dynamic arrays, Hash Tables können auch anders implementiert werden. Mehr dazu findest du unter `https://en.wikipedia.org/wiki/Hash_table#Collision_resolution`.

2.2a) Konstruktor und Destruktor (1 Punkt)

Kern der HashMap ist ein Array von fester Größe. Elemente dieses Arrays werden auch als `bucket` oder `slot` bezeichnet. Wie groß das Array sein soll wird über den Parameter `size` bestimmt, speichere diesen auch im gleichnamigen Datenfeld. Das Array ist vom Typ `MapBucket` und hat den Bezeichner `arr`.

Kümmere dich im Konstruktor bzw. Destruktor darum das Array mittels `new[]` und `delete[]` zu erstellen bzw. freizugeben.

Alternativ könnte hierfür in C++ die Klasse `std::vector` verwendet werden. Schaue dir die Funktionen `std::vector::reserve` und `std::vector::resize` unter `https://en.cppreference.com/w/cpp/container/vector` an. Worin unterscheiden sich die beiden Funktionen? Welche müsste in diesem Kontext verwendet werden um den gleichen Effekt wie die Arrayallokation zu erreichen?

2.2b) Hashing-Funktion (1 Punkt)

Implementiere die Funktion

```
std::size_t calcHash(const std::string &key);
```

Diese Hashing-Funktion wird benötigt um ein Element der HashMap anhand seines Schlüssels einem Index des Arrays `arr` zuzuordnen. Damit das funktioniert muss der Wertebereich der Hashfunktion für jeden Eingabewert zwischen 0 und size-1 liegen.

Wir verwenden eine sehr einfache Funktion, die lediglich die `char`-Werte aller Zeichen des Strings summiert und diese modulo der Arraygröße rechnet.

2.2c) Insert (2 Punkte)

Jetzt wollen wir ein Element, dessen Schlüssel wir kennen, in unserer Map speichern. Implementiere hierfür die Funktion

```
void insert(const std::string &key, const std::vector<Item> &order);
```

Um ein Element einzufügen, berechnen wir mit der Funktion `calcHash` die Position im Array `arr` an der wir dieses speichern möchten. Ist das Bucket (also der `std::vector`) leer, können wir sofort Schlüssel und Value darin speichern. Erstelle hierfür eine Instanz von `MapElement` und füge diese dem `std::vector` hinzu.

Wenn das Bucket nicht leer ist, so sind an der Position des Arrays bereits ein Werte gespeichert. Hier gilt es zwei Fälle zu Unterscheiden.

1) Der Schlüssel des einzufügenden Elements ist bereits in einem `MapElement` gespeichert. Überschreibe in diesem Fall den Wert in dem Element mit dem neuen Wert.

2) Der Schlüssel des einzufügenden Elements kommt in dem Bucket noch nicht vor. Das nennt man einen Konflikt und ist der Grund dafür, dass jedes Bucket ein `std::vector` an `MapElement`-Instanzen anstatt nur einer einzelnen Instanz ist. Der Konflikt kann so gelöst werden, indem genauso vorgegangen wird wie dies oben schon für ein leeres Bucket beschrieben wurde.

2.2d) Remove (2 Punkte)

Implementiere zum Entfernen eines Schlüssels die Funktion

```
void remove(std::string key);
```

Hier wird zunächst analog zum Einfügen vorgegangen und der Index im Array über die Hashfunktion berechnet. Dann muss der Schlüssel in dem Bucket gesucht werden. Kommt der Schlüssel vor, muss das entsprechende Element aus dem `std::vector` gelöscht werden. Andernfalls ist nichts zu tun.

2.2e) Get (2 Punkte)

Implementiere mit dem Wissen aus den beiden vorherigen Teilaufgaben die Methode

```
std::vector<Item> get(const std::string &key);
```

die den Wert zu einem Schlüssel zurückliefert.

Sollte der Schlüssel in der Map nicht existieren soll eine Exception vom Typ `std::invalid_argument` aus dem Header `<stdexcept>` geworfen werden. Übergebe der `std::invalid_argument`-Klasse als Parameter einen `std::string` mit einer aussagekräftigen Fehlermeldung.

Aufgabe 2.3: `[S]` Kreativaufgabe (optional)

Bei der Kreativaufgabe geht es darum die Arbeit der vorherigen Aufgaben sinnvoll zu nutzen (solltest du diese nicht vollständig implementiert haben kannst du entweder leere Methoden verwenden oder für Aufgabe 2 die map aus der STL verwenden). Du sollst mit diesen ein einfaches, textbasiertes Bestellsystem implementieren.

2.3a) Programmstart

Beim Aufrufen des Programmes wird als Argument der Pfad der CSV-Datei übergeben. Diese soll mit der Funktion `laden(...)` geöffnet werden.

Wird kein Pfad zu einer Datei angegeben, soll eine Fehlermeldung ausgegeben werden. Das Programm soll anschließend den Wert -1 zurückgeben um dem Betriebssystem den Fehler mitzuteilen.

2.3b) Menü

Implementiere in der `main(...)` Funktion ein einfaches, interaktives Menü. Die Speisekarte kann mit Hilfe der ersten Aufgabe implementiert werden, für die Bestellungen kann die Map verwendet werden. Dabei sollen die Kund:innen als Key fungieren und in den `cppp::Item` (vgl. Aufgabe 2.2) die Anzahl sowie die Speisen der Bestellung aufgenommen werden.

Es soll folgende Funktionalitäten im Menü geben:

- Erstellen einer neuen Bestellung
- Anzeigen einer Bestellung
- Löschen einer Bestellung
- Beenden des Programms
- Anzeigen der Speisekarte

Hinweis: beachte, dass `std::cin` nicht nur newline `\n` sondern auch Leerzeichen als Trennungszeichen behandelt. Wenn du also Namen mit einem Leerzeichen verwenden möchtest, musst du dir eine Lösung dafür überlegen.