Programmiertechnik II

Sommersemester 2015 – Aufgabenblatt 5

Hasso-Plattner-Institut
Fachgebiet Computergrafische Systeme

Anmerkungen zu Aufgabenblatt 5

Der zu verwendende Programmrahmen steht als Zip-Archiv `uebung5.zip` im Moodle unter der Adresse `https://moodle.hpi3d.de/course/view.php?id=75` zum Download zur Verfügung.

Die Aufgaben sollen in den benannten und im Programmrahmen zumeist gekennzeichneten Bereichen implementiert werden. Bevor Sie mit der Lösung beginnen, lesen Sie das gesamte Aufgabenblatt und überprüfen Sie, ob der Programmrahmen auf Ihrem System fehlerfrei kompiliert, die Datensätze enthalten sind und sich alle Programme ausführen lassen. Wir wünschen viel Spaß und Erfolg bei der Bearbeitung der Aufgaben!

Aufgabe 5.1: Iteratoren (6 Punkte 2+2+2)

Iteratoren ermöglichen den Zugriff auf Inhalte von Datencontainern mit einer abstrakten Traversierungsfunktionalität. Dadurch können Zugriffs-, Such- und Auswertungsfunktionen implementiert werden, die vom konkret verwendeten Containertyp unabhängig sind. Implementieren Sie die folgenden Funktionen ohne den Subscript-Operator `[]`.

a) **Umordnung von Containerelementen:**
Gegeben sei eine Folge $Z = \{c_0, c_1, ..., c_{N-1}\}$ von Elementen beliebigen Typs. Erzeugen Sie eine neue Folge Z', bei der die Elemente abwechselnd vom Anfang bzw. vom Ende von Z stammen, d. h. Z' $= \{c_0, c_{N-1}, c_1, c_{N-2}, ...\}$. Entwickeln Sie die Funktion `front_back_pairing` generisch, so dass diese mindestens für die Containertypen `std::vector` und `std::list` funktioniert, sowie hinsichtlich des Datentyps unabhängig ist. Testen Sie ihre Implementierung zumindest für eine Zahlenfolge im Format `std::vector<int>` und eine Liste von Zeichenketten `std::list<std::string>`.

b) **Entfernung von Duplikaten:**
Gegeben sei eine Folge $Z = \{c_0, c_1, ..., c_{N-1}\}$ von Elementen beliebigen Typs, die in einem `std::vector` enthalten sind. Zu entfernen sind alle Duplikate, d. h. wenn Elemente mit einem Wert mehrfach vorkommen, dann ist nur das erste Element zu erhalten. Beispiel: Falls $c_5 = c_7 = c_{39}$, dann werden c_7 und c_{39} aus dem Container entfernt. Entwickeln Sie eine generische Funktion `remove_duplicates`, die Duplikate entfernt. STL-Standardalgorithmen dürfen verwendet werden.

c) **Erweitern einer Zahlenfolge:**
Gegeben sei eine Zahlenfolge $Z = \{c_0, c_1, ..., c_{N-1}\}$. Erzeugen Sie eine neue Folge Z', bei der zu jedem Ursprungselement jeweils links und rechts ein neues Element hinzukommt; das jeweils neue linke bzw. rechte Element enthält die numerische Differenz zum Ausgangselement, d.h. das Element c_i in der Ursprungsfolge wird ersetzt durch die Teilfolge $c_{i-1} - c_i, c_i, c_{i+1} - c_i$. Die resultierende Folge Z' ist dreifach so groß wie Z. Die Folge wird zirkulär betrachtet, d.h. $c_{-1} = c_{N-1}$ und $c_N = c_0$. Implementieren Sie eine entsprechende generische Funktion `insert_differences`, die den Inhalt des übergebenen Containers C (enthält die Folge Z) durch die neue erweiterte Folge Z' ersetzt. STL-Standardalgorithmen dürfen verwendet werden.

Aufgabe 5.2: Suchen in Datensätzen (6 Punkte 2+4)

Gegeben sind Flugverbindungsdaten, die nach dem Einlesen einer Datei (`routes.csv`) im Speicher durch eine Struktur `Route` mit den Attributen `SourceID`, `DestinationID` und `AirlineID` dargestellt werden. Eine Flugverbindung wird als Eintrag in einem Vektor vom Typ `std::vector<Route>` abgelegt. Die gegebene Einleseroutine fügt die Daten direkt in den Vektor ein und gruppiert sie nach `AirlineID`.

a) **Lineare Suche der Verbindungen zu einem Zielflughafen:**
Implementieren Sie eine Funktion, die zu einer gegebenen `DestinationID`, d.h. einem Zielflughafen, ermittelt, wie viele Verbindungen zu diesem Zielflughafen insgesamt existieren. Nehmen Sie dazu in dieser Teilaufgabe zunächst das Prinzip der lineare Suche.

Zum Test führen Sie Abfragen für alle möglichen `DestinationsID`s (1 bis 9541) aus und messen Sie die Gesamtlaufzeit dieser Suchanfragen. Ermitteln Sie als statistische Information die Anzahl der Zugriffe auf den Datenvektor.

b) **Binäre Suche der Verbindungen zu einem Zielflughafen:**
Implementieren Sie die Suchfunktion zur Ermittlung aller Verbindungen zu einem Zielflughafen (analog zu 5.1a) mit Hilfe eines sortierten Vektors. Überlegen Sie, wie Sie die Elemente so sortieren können, um dann für die Suchanfrage die Sortierreihenfolgen ausnutzen zu können. Für das Sortieren steht die Standardfunktion `std::sort` bereit.

Wie bei Teilaufgabe 5.1a, führen Sie zum Test Abfragen für alle möglichen `DestinationsID`s (1 bis 9541) aus und messen Sie die Gesamtlaufzeit dieser Suchanfragen (ohne die Sortierung). Ermitteln Sie als statistische Information die Anzahl der Zugriffe auf den Datenvektor. Wie verändern sich Laufzeit und Containerelementzugriffszahlen?

Hinweis: Nutzen Sie zur Laufzeitmessung geeignete Funktionen aus der `std::chrono`-Bibliothek.

Aufgabe 5.3: Mergesort für Vektoren (3 Punkte)

Implementieren Sie zu einem Vektor mit beliebigen Datentyp `T` (`std::vector<T>`) den Mergesort-Algorithmus. Die entsprechende Funktion erhält den Vektor per Referenz; das sortierte Ergebnis wird darin wieder gespeichert. Für den Datentyp T sei die Kleiner-Relation definiert, d.h. es existiert stets `bool operator<(const T& a, const T& b)`. Die `std::inplace_merge`-Funktion darf nicht verwendet werden.

Zusatzaufgabe: Management von Restaurant-Bestellungen (1 Klausurpunkt)

In einem Fastfood-Restaurant gibt es Speisen und Getränke. Das Bedienpersonal kommt zu den Tischen und nimmt die Aufträge entgegen. Da die geselligen Gäste gerne verweilen, gibt es meist mehrfache Bestellungen pro Tisch, zum Beispiel erst Getränke, dann Speisen und dann später nochmals Nachbestellungen. Diese Bestellung werden jeweils für einen Tisch mit einer eindeutigen Tischnummer gebündelt.

Erstellen Sie eine Implementierung, die Aufträge (*Orders*) entgegennimmt, in eine globale Auftragssammlung einfügt und dort Einzelaufträge pro Tisch zusammenfasst. Beispielsweise müssen, wenn die Gäste eines Tisches zahlen möchten, die Einzelaufträge zusammengefasst werden, um die Rechnungssumme zu ermitteln. Es gilt hier die Sonderaktion: "Jeder dritte Burger ist gratis! Jedes vierte Getränk (egal, welches) auch!"

Ein Auftrag ist in unserem Beispielrestaurant übersichtlich definiert durch die folgende Struktur:

```
struct Order {
    int table;
    int coffee;
    int coke;
    int burger;
    int salad;
    int id;
};
```

Die bestellbaren Waren umfassen Getränke (`coffee`,`coke`) und Speisen (`burger`, `salad`). Jede Bestellung bekommt eine fortlaufende globale Bestellnummer (`id`). Jede Bestellung ist immer genau einem Tisch (`table`) zugeordnet.

Bei der Zusammenfassung der momentanen globalen Auftragsliste, sind alle Bestellungen eines Tisches zusammenzuführen. Das geschieht, indem die Anzahlen der Speisen und Getränke addiert werden. Die erste Bestellung gilt dabei als Hauptbestellung.

Beim Bezahlen muss zu einer Tischnummer ermittelt werden, was die zusammengefassten Bestellungen kosten. Dazu wird ein einfaches Kostenschema genutzt (Getränke: 2 Euro, Burger: 5 Euro, Salat: 4 Euro). Beachten Sie die Sonderaktion.

Im Programmrahmen ist eine Auftragsmenge fest kodiert, die verarbeitet werden muss. Implementieren Sie folgende Funktionen:

a) `processOrders()`
Diese Funktion überführt "incoming orders" in die globale Auftragsliste ("current orders"). Überlegen Sie, wie die Einzelaufträge sinnvoll in die globale Auftragsliste eingeführt werden sollten.

b) `mergeOrders()`
Diese Funktion "komprimiert" die vorliegenden Bestellungen in der globalen Auftragsliste ("current orders"). Alle Aufträge für einen Tisch werden zu einem einzigen Auftrag zusammengefasst.

c) `pay()`
Diese Funktion berechnet den Preis der gesamten Bestellungen pro Tisch auf Basis der oben genannten einfachen Preistabelle. Der Auftrag wird hierbei aus der globalen Auftragstabelle gelöscht. Hat ein Tisch keine Bestellung, dann wird ein Betrag von 0 Euro zurückgegeben.

Für die Implementierung sind allgemein Iteratoren und -unter anderem- auch die Funktionen `std::sort`, `std::remove_if`, `std::find_if`, `std::vector<Order>::erase` hilfreich. Dem Programmrahmen sind Testdaten für Aufträge beigelegt (`orders.txt`), gegen die automatisiert getestet wird.

Allgemeine Hinweise zur Bearbeitung und Abgabe

- Die Aufgaben sollen maximal zu zweit bearbeitet werden; Ausnahmen sind nicht vorgesehen. Je Gruppe ist nur eine Abgabe notwendig.
- Bitte reichen Sie Ihre Lösungen bis **Mittwoch, den 10. Juni um 13:00 Uhr** ein.
- Zur Prüfungszulassung muss ein Aufgabenblatt zumindest bearbeitet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte bewertet werden. Aufgaben mit Klausurpunkten werden 1:1 berücksichtigt.
- Die Aufgaben können auf allen wesentlichen Plattformen (Windows, Linux, OS X) bearbeitet werden. Lesen Sie bei Fragen zum Kompilieren und Ausführen bitte genau die den Archiven jeweils beiligende README.TXT. Bestehen weitere Fragen und Probleme, kontaktieren Sie einen der Betreuer direkt oder nutzen Sie das Forum im Moodle.
- Die Durchsicht zur Bewertung kann sowohl auf Windows, Linux oder OS X erfolgen. Es wird folglich eine **plattformunabhängige Lösung** der Aufgaben erwartet.
- Archivieren (Zip) Sie zur Abgabe Ihren bearbeiteten Programmrahmen und ergänzen Sie Ihre Matrikelnummer(n) im Bezeichner des Zip-Archivs entsprechend im folgenden Format: `uebung5_matrikNr1_matrikNr2.zip`. Beachten Sie, dass dabei nur die vollständigen Quelltexte, die CMake-Konfiguration sowie eventuelle Zusatzdaten gepackt werden (alles was im gegebenen Programmrahmen entsprechend vorhanden ist). Senden Sie uns keinesfalls Kompilate und temporäre Dateien (*.obj, *.pdb, *.ilk, *.ncb etc.). Testen Sie vor dem Verschicken, ob die Projekte aus den Zipdateien fehlerfrei kompiliert und ausgeführt werden können.
- Reichen Sie Ihr Zip-Archiv im Moodle ein:
`https://moodle.hpi3d.de/course/view.php?id=75`.