universität wien

Programmierung 1 VU 051010

Modul PR1

Uebungseinheit 7

19.11.2019

Diese Unterlagen können Sie auf Ihrem Arbeitsplatzrechner im Labor anzeigen mit dem Befehl evince /home/Xchange/PR1/ue7/uebung7.pdf &

WS 2019

universität wien

Max Mustermann
Währinger Straße 17
1080 Wien

Beispiel: Adressverwaltung

- Erstellen Sie ein struct **`City`** bestehend aus zwei Strings für **`zip_code`** und **`city_name`**.
- Definieren Sie im vorgegebenen struct **`ZIP`** (/home/Xchange/PR1/ue7/zip.h) die Methoden **`check`** und **`print`**. **`check`** soll **`true`** retournieren, wenn der **`zip_code`** des **`this-Objekts`** in der Liste **`ZIP.cities`** gefunden werden kann, **`false`** sonst. **`print`** soll den Code und die zugehörige Bezeichnung der Stadt am Bildschirm ausgeben, wie in der letzten Zeile im Beispiel oben. Kann für eine Postleitzahl kein passender Ort gefunden werden, so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- Erstellen Sie ein struct **`Address`** bestehend aus **`first_name`**, **`surname`**, **`road`**, **`street_number`** und **`zip_code`**. Die ersten vier Felder sind Strings **`zip_code`** hat den Datentyp **`ZIP`**.
- Erstellen Sie eine Methode **`print`** in **`Address`**, die die Adresse formatiert auf dem Bildschirm ausgibt (siehe Beispiel oben).

Anmerkung: Die structs müssen in unabhängigen Dateien (city.h, zip.h, zip.cpp, address.h und address.cpp) definiert und implementiert werden.

universität wien

Max Mustermann
Währinger Straße 17
1080 Wien

Beispiel: Adressverwaltung (Gruppierung)

- Erstellen Sie eine Funktion **`group`**, die eine Liste von Adressen (Vektor) als Parameter erhält und einen **`vector<vector<Address>>`** als Ergebnis liefert, in dem die Adressen nach Postleitzahl gruppiert sind (Der erste Eintrag im Ergebnisvektor beinhaltet eine Liste der Adressen mit der ersten Postleitzahl etc.; die Reihenfolge ist unerheblich).
- Sie finden ein Rahmenprogramm (address_pr1.cpp) sowie Daten (address_input_pr1.txt) zum Testen unter /home/Xchange/PR1/ue7.

Anmerkung: Zur Verwendung des Rahmenprogramms müssen die structs in unabhängigen Dateien (city.h, zip.h, zip.cpp, address.h und address.cpp) definiert und implementiert werden.

universität wien

Erweiterung: gruppierte Adressen sortieren

- Ändern Sie die Funktion **`group`** so, dass die Gruppen nach aufsteigenden Postleitzahlen und die Einträge in den Gruppen nach (Zuname, Vorname, Straße) sortiert werden.
- Verwenden Sie dazu das Insertion-Sort Verfahren: Wird ein neues Element an den (bisher schon sortierten) Vektor mit **`push_back`** angefügt, so vertauscht man dieses neue Element so lange mit dem jeweiligen Vorgänger im Vektor, bis der Vorgänger in der Sortierreihenfolge nicht mehr größer ist, oder es keinen Vorgänger mehr gibt. Das neue Element ist dann an der korrekten Stelle eingefügt und der Vektor wieder sortiert.

universität wien

Erweiterung: Adresse in Gruppierung einfügen

- Schreiben Sie eine Funktion **`insert`**, die eine nach Postleitzahlen gruppierte Adressliste **`vector<vector<Address>>`**, sowie eine Liste von Adressen **`vector<Address>`** als Parameter erhält und alle Adressen aus dem zweiten Parameter in die gruppierte Adressliste einfügt, ohne die Sortierung zu zerstören.

universität wien

Erweiterung: Adresse in Gruppierung suchen

- Schreiben Sie eine Funktion **`find`**, die eine sortierte, gruppierte Adressliste und eine Adresse als Parameter erhält und einen booleschen Wert zurückliefert. Der Returnwert soll **`true`** sein, wenn die Adresse aus dem zweiten Parameter in der Adressliste enthalten ist, **`false`** sonst.
- Verwenden Sie binäre Suche. Betrachten Sie das Element in der Mitte des zu durchsuchenden Vektors. Ist dieses das gesuchte Element, so ist die Suche zu Ende. Andernfalls sucht man in jenem Teilvektor weiter, in dem das gesuchte Element liegen muss (ist das gesuchte Element kleiner als das mittlere, dann ist das die Hälfte mit den kleineren Indizes, sonst die Hälfte mit den größeren Indizes). Die Suche endet, wenn das Element gefunden wurde, oder der zu durchsuchende Vektor nur mehr aus einem Element besteht.
- Sollte die gesuchte Adresse mehrmals enthalten sein, so ist eine Exception vom Typ runtime_error zu werfen.

universität wien

Beispiel: struct Car Ein-/Ausgabe

- Erstellen Sie ein struct Car aus den Feldern brand (String), type (String), kW (int), color (enum) und extras (enum Liste).
- Ein Auto kann nur eine Farbe haben, aber beliebig viele Extras.
- Schreiben Sie die Methoden **`print`** und **`read`**, um den Zustand eines Autos auf dem Bildschirm auszugeben, oder von der Tastatur einzulesen.
- Beachten Sie beim Einlesen, dass jedes Extra nur einmal in der Liste der Extras auftreten darf.
- Für die Ein-/Ausgabe ist es vorteilhaft, für die enums globale Konstanten mit den passenden Bezeichnungen zu definieren, z.B.:

```
const vector<string> color_names {"red","green","blue"};
enum class Color {red, green, blue} color{Color::blue};
cout << color_names.at(static_cast<size_t>(Color::green));
cout << color_names.at(static_cast<size_t>(color));
```

universität wien

Erweiterung Auto Histogramm

- Schreiben Sie eine Funktion **`histogram`**, die einen Vektor von Autos als Parameter bekommt und einen Vektor liefert, der die Anzahlen der Autos in einer bestimmten Farbe enthält. Die Anzahl für eine Farbe soll dabei im Vektor an der, dem enum-Wert entsprechenden Indexposition gespeichert werden
- Geben Sie das Ergebnis "grafisch" aus, z.B.:

```
red:   *******
green: **
```

universität wien

Erweiterung: Auto Suche

- Schreiben Sie eine Funktion **`find`**, die eine Liste (Vektor) von Autos und Suchkriterien (Kundenwünsche) als Parameter erhält und eine Liste von passenden Autos für den Kunden / die Kundin retourniert.
- Die Suchkriterien, die angegeben werden können: gewünschte Marke und Typbezeichnung als String (leer, wenn es egal ist), gewünschte Mindestleistung als int, gewünschte Farbe (Vektor von Farboptionen, leer für egal) und gewünschte Extras (Vektor von Extras, die zumindest vorhanden sein müssen).

- Entwerfen Sie ein eigenes struct für die Suchkriterien.

universität wien

Erweiterung Auto Einschränkungen

- Unterschiedliche Hersteller bieten verschiedene Kombinationen von Extras, bzw. von Farben und Extras nicht an. Die Einschränkungen können für alle Typbezeichnungen eines Herstellers gelten, oder nur für bestimmte.
- Überlegen Sie sich eine sinnvolle Datenstruktur, in der man die Einschränkungen aller Hersteller sammeln kann.
- Schreiben Sie eine Methode **`check`**, die diese Datenstruktur als Parameter erhält und prüft, ob der Zustand des Autos den vorgegebenen Einschränkungen entspricht. Es ist **`true`** zurückzuliefen, wenn dies der Fall ist, **`false`** sonst.

universität wien

Beispiel: Geometrie

- Erstellen Sie ein struct Point zur Darstellung von Punkten im $\mathbb{R}^3$(x, y und z-Koordinate).
- Erstellen Sie die structs Line bzw. LinePD zur Darstellung von Geraden mittels zweier Punkte bzw. in Punkt Richtungsform (point direction).
- Erstellen Sie **`check`** Methoden in Line und LinePD, um zu prüfen, ob die jeweiligen Daten eine gültige Gerade darstellen.
- Erstellen Sie **`transform`** Methoden in Line und LinePD, die jeweils in die andere Darstellung umrechnen

universität wien

Erweiterung: Geometrie

- Schreiben Sie **`contains`** Methoden in Line und LinePD, die einen Punkt als Parameter erhalten und true liefern, wenn der Punkt auf der Geraden liegt, false sonst.
- Berücksichtigen Sie, dass Rechenfehler auftreten und legen Sie eine Genauigkeit von z.B. eps=10^{-9} fest.

universität wien

Erweiterung: Geometrie

- Erweitern Sie Ihre structs so, dass Punkte und Gerade in einem beliebig dimensionierten Raum ($\mathbb{R}^n$) dargestellt werden können.