Systemprogrammierung - AIN/2

Sommersemester 2024

Übungsaufgabe 5: C++ Klassen, Iteratoren, Übersetzungseinheiten

Abgabe bis 20./21.6.2024

Vorbereitung

Legen Sie ein Arbeitsverzeichnis für Aufgabe 5 an und speichern Sie darin das Archiv ↷ **aufgabe5.tar.gz**. Entpacken sie das Archiv mit dem folgenden Kommando:

```
tar xzf aufgabe5.tar.gz
```

Ihr Arbeitsverzeichnis von Aufgabe 5 sollte anschließend die vier Dateien `Makefile`, `notenspiegel.cpp`, `notenspiegel-in.txt` und `notenspiegel-out.txt` enthalten.

Programmierung

Das vorgegebene Programm `notenspiegel.cpp` erstellt einen Notenspiegel. Ergänzen Sie die fehlenden Übersetzungseinheiten wie folgt:

Erstellen Sie eine Übersetzungseinheit `benotung`, die eine gleichnamige Wertklasse mit den folgenden Komponenten enthält:

- eine private Membervariable `note` vom Typ `int`
- einen öffentlichen expliziten Konstruktor `benotung(int)` zum Initialisieren der Membervariablen
  Werfen Sie bei unzulässiger Note eine Ausnahme vom Typ `std::invalid_argument` mit `"unzulaessige Note "` und der falschen Note als Fehlertext (zulässig sind die Noten 10, 13, 17, 20, 23, 27, 30, 33, 37, 40, 50)
- zwei öffentliche konstante statische Membervariablen `beste` und `schlechteste` vom Typ `benotung` mit der besten bzw. schlechtesten Note
- eine öffentliche Memberfunktion `int int_value()`, die die im Objekt gekapselte Note liefert
- eine öffentliche Memberfunktion `bool ist_bestanden()`, die `true` liefert, wenn der Wert der gekapselten Note kleiner oder gleich `40` ist, sonst `false`
- eine befreundete Funktion `bool operator==(benotung, benotung)`, die `true` liefert, wenn beide Objekte die gleiche Noten kapseln, sonst `false`
- welche Memberfunktionen erzeugt der Compiler zusätzlich implizit in `benotung`? Implementiert er die in diesem Fall korrekt?

Erstellen Sie eine Übersetzungseinheit `fachnote`, die eine gleichnamige Entitätenklasse mit den folgenden Komponenten enthält:

- zwei öffentliche konstante Membervariablen `fach` vom Typ `std::string` und `note` vom Typ `benotung` zum Speichern eines Fachnamens mit Note
- einen öffentlichen Konstruktor `fachnote(const std::string&, const benotung&)` zum Initialisieren der Membervariablen
  Der Fachname darf nicht die Länge 0 haben. Werfen Sie eine Ausnahme vom Typ `std::invalid_argument`, wenn diese Konsistenzregel verletzt ist.
- überlegen Sie, welche impliziten Memberfunktionen Sie mit `= delete` unterdrücken müssen

Erstellen Sie eine Übersetzungseinheit `fachnoten_liste`, die eine gleichnamige Entitätenklasse enthält. Verwenden Sie das Vorlesungsbeispiel `intlist` aus Teil 5 als Vorlage und sehen Sie die folgenden Änderungen vor:

- die Liste soll `fachnote*` statt `int` speichern.
- statt eines Standardkonstruktors soll es einen expliziten Konstruktor mit einem Parameter geben. Leiten Sie den Typ aus dem Aufruf in `notenspiegel.cpp` ab. Speichern Sie die übergebene Funktionsadresse in einer zusätzlichen privaten Membervariablen.
- der Destruktor der Listenklasse ruft pro Listenelement die in der zusätzlichen Membervariablen gespeicherte Funktion auf, um den Speicher des jeweils referenzierten `fachnote`-Objekts freizugeben.

Test und Qualitätssicherung

Verwenden Sie zum Testen die gewohnten Befehle:

```
make
make cppcheck
./notenspiegel
valgrind ./notenspiegel
```

Führen Sie auch die folgenden automatisierten Tests aus:

```
valgrind ./notenspiegel < notenspiegel-in.txt
./notenspiegel < notenspiegel-in.txt > out.txt
diff -Z notenspiegel-out.txt out.txt
```

- `valgrind` darf keine Fehler und `diff` keine Unterschiede melden.
- `cppcheck` sollte möglichst keine Probleme melden.

Bessern Sie gegebenenfalls nach.

Abgabe

Führen Sie Ihr Programm mit den automatisierten Tests vor.
Zeigen Sie das ausgefüllte Teilnahmeprotokoll.

Hinweis:
Der Compiler g++ darf für Ihr Programm keine Fehler oder Warnungen mehr ausgeben.

Ihr Programm muss außerdem ordentlich formatiert sein. Bessern Sie die Formatierung gegebenenfalls mit `astyle` nach:

```
astyle -p -H --style=ansi *.[ch]*
```

Freiwillige Zusatzaufgaben (je 1 Bonuspunkt)

- Bachelorarbeiten werden an der HTWG von zwei Prüfern bewertet. Aus den Bewertungen der beiden Prüfer wird der Mittelwert gebildet. Ergänzen Sie passend dazu in Ihrer Wertklasse `benotung` einen weiteren Konstruktor `benotung(int, int)`, der das Objekt mit dem Mittelwert der beiden übergebenen Einzelnoten initialisiert. Werfen Sie bei unzulässigen Einzelnoten analog zum anderen Konstruktor eine Ausnahme vom Typ `std::invalid_argument`. Erweitern Sie `notenspiegel.cpp` um die Möglichkeit, bei einem Fach wahlweise eine oder zwei Noten eingeben zu können.
- Sorgen Sie dafür, dass die verkettete Liste der Fachnoten immer nach Noten sortiert ist, mit der besten Note am Listenanfang. Fügen Sie dazu in `fachnoten_liste::insert(fachnote*)` neue Noten nicht am Anfang der Liste ein, sondern suchen Sie mit einer Schleife die richtige Einfügestelle. Als Voraussetzung brauchen Sie für die Wertklasse `benotung` eine Ordnungsrelation in Form eines befreundeten `operator<`.
- Deklarieren Sie den Konstruktor der Klasse `fachnote` als `private` und ergänzen Sie eine Fabrikfunktion `new_instance`, die einen Zeiger vom Typ `std::unique_ptr<fachnote>` liefert (siehe das Vorlesungsbeispiel `termin` in Teil 6). Verwenden Sie dann in `notenspiegel.cpp` statt der eigenen `fachnoten_liste` eine Liste `std::forward_list<std::shared_ptr<fachnote>>`. Die Entsprechung zur Memberfunktion `insert` heißt dort `push_front`. Die Funktion `delete_fachnote` in `notenspiegel.cpp` können Sie nun weglassen, weil die Destruktoren von `std::forward_list` und `std::shared_ptr` den Speicher von Liste und Listenelementen automatisch freigeben. Prüfen Sie das mit `valgrind` nach.

Prof. Dr. H. Drachenfels
Hochschule Konstanz** - **Impressum** - **Datenschutzerklärung

Letzte Änderung: 19.3.2024