University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Praktikum Programmieren 3

Aufgabe #2

Elektrotechnik

Wintersemester 2021

Prof. Dr. Dirk Kutscher
Behrend Pupkes

University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Allgemeine Hinweise

In dieser Aufgabe, verwenden und erweitern wir Ihre bisherigen Arbeiten von Praktikumsaufgabe #1. Bitte erstellen Sie ein neues Verzeichnis (oder ein neues Projekt, wenn Sie VisualStudio verwenden) für diese Aufgabe 2 und kopieren Sie Ihre Dateien aus Aufgabe #1 dorthin.

Modularisierung

Verwalten Sie alle Klassen in eigenen Header- und CPP-Dateien. Verwenden Sie eine eigene Datei für die main-Funktion. Beachten: Verwendung von "Header Guards" (siehe https://www.learncpp.com/cpp-tutorial/89-class-code-and-header-files/).

Definieren Sie einen namespace **`Web`** für alle Ihre Klassen und sonstigen Funktionen (außer **`main`**). Nennen Sie Ihre Klassen entsprechend um, d.h., von **`WebRequest`** nach **`Web::Request`** usw.

Neue Klassen

- Definieren Sie Klassen für unterschiedliche Request-Typen, die alle von der Basisklasse **`Request`** erben: **`GetRequest, PutRequest, PostRequest:`**
  - Der Konstruktor von **`Request`** soll zunächst unverändert sein.
  - Die Konstruktoren von den abgeleiteten Klassen sollen nur noch einen Parameter zum Initialisieren der Member-Variablen **`requestResource`** bekommen.

- Die abgeleiteten Klassen sollen den Konstruktor von **`Request`** per Konstruktor-Delegation aufrufen und dabei den entsprechenden Request-Typ als Parameter übergeben
- Passen Sie die anderen Teile des Programms entsprechend an.

Neue Features für den Web-Server

In dieser Aufgabe sollen Sie den Web-Server etwas leistungsfähiger machen. Für die Bearbeitung von GET-Requests soll der Server mögliche Anfragen (z.B. "index.html") und passende Antworten (z.B. string-Objekt mit HTML-Inhalt) in geeigneter Datenstruktur aufbewahren und bei Anfragen die entsprechenden Antworten zurückgeben.

- Wo sollte diese Datenstruktur initialisiert werden?
- Wenn kein passendes Antwort-Objekt in der Datenstruktur gefunden wurde, dann soll der Server im lokalen Dateisystem im aktuellen Verzeichnis nach einer entsprechend benannten Datei suchen (z.B. Request nach "test.html" sollte dann eine Datei namens "test.html" öffnen und den Inhalt als Antwort liefern).
- Wenn keine passende Datei gefunden wurde, dann eine entsprechende Antwort (Response Code 404 – "Not Found") zurückgeben.
- Ändern Sie Ihr Programm so, dass es auf der Kommandozeile mit Request-Resource-Namen aufgerufen werden kann. Diese Ressourcen sollen dann alle nacheinander mit einem GET-Request abgefragt werden (z.B. "./webmain test.txt bla.txt hallo.html") – wenn Ihr ausführbares Programm "webmain" heisst. Testen Sie dies mit Dateien in Ihrem lokalen Verzeichnis (z.B. "./webmain server.hh main.cpp").
- Ändern Sie Ihre Klasse **`Web::Response`** so, dass die beim Aufruf einer Member-funktion **`getResponseText`** eine zum **`responseCode`**

passenden Text zurückgibt, z.B. "OK" für 200. Verwenden Sie dafür eine passende Datenstruktur, mit der Sie int-Werte für die Response-Codes zu den entsprechenden Response-Texten zuordnen können (d.h., ermitteln Sie den Text nicht durch if oder ähnliche Kontrollstrukturen).

- Erweitern Sie Ihr Programm so, dass zu jedem bearbeiteten Request eine Ausgabe in einer Datei namens **`server.log`** ausgegeben wird. Die Ausgaben sollen immer an die Datei angehängt werden. Schreiben Sie dafür eine Member-Funktion **`Server::log`**, die folgende Parameter bekommt:
  - **`const Request&`**
  - **`const Response&`**
- **`Server::log`** soll nur innerhalb des Klassen-Kontexts aufrufbar sein.
- Die Log-Ausgabe soll ungefähr so aussehen:

2020-11-04-17:00:55 // GET // index.html // 200 OK

Hinweise

Mit Dateien arbeiten

Beschäftigen Sie sich mit **`std::ifstream`** und **`std::ofstream`**.

Inhalt einer Datei in einen String einlesen

Sie können die folgende Funktion verwenden, um den Inhalt einer Datei in einen String einzulesen:

```
25   bool readFile(const string& filename, string& result) {
26     bool ok(false);
27
28     ifstream file(filename);
29     if(file.is_open()) {
30       result.assign(
31                 (istreambuf_iterator<char>(file)),
32                 istreambuf_iterator<char>());
33       if(result.size()) // irgendetwas wurde gelesen
34       ok=true;
35     }
36     return ok;
37   }
```

Fragen

(bitte in der Doku und im Testat beantworten)

1. Erläutern Sie den Zweck und die Funktionsweise der von Ihnen verwendeten "Header Guards" (siehe Modularisierung).
2. Was bedeutet es, wenn man in einer Klasse die Sichtbarkeit von Definitionen mit **`protected`** auszeichnet?
3. Mit welcher Datenstruktur können Sie eine assoziative Zuordnung von einem Wert zu einem anderen Wert vornehmen?
4. In der Funktion **`readFile`** (siehe oben) wurde die als **`ifstream`** geöffnete Datei nicht explizit wieder (mit **`ifstream::close`**) geschlossen. Warum ist das in diesem Fall zulässig? Wird die Datei überhaupt wieder geschlossen?