

PÜ32

Projektübungsblatt: Codierung von Dateien

PÜ32.1

Vom studentischen
Übungsprojekt ...

Vorbemerkungen

Zur Geschichte dieses Projekts: Anfangs gab es die [PÜ32.2 Aufgabenbeschreibung](#) und die Vorbemerkung „Zu diesem Projekt gestalten Sie Ihre Lösung mit mehr Freiheit, aber auch mehr Verantwortung als die Lösungen zu den ersten beiden Projekten.“ Die Studenten lieferten meist erschreckend dilettantische Basteleien. Darum ergänzte ich die [PÜ32.3 Hinweise zum Vorgehen und Umsetzen](#). Die Studenten ignorierten meist die Hinweise und lieferten dilettantische Basteleien. Darum ergänzte ich die [PÜ32.4 Minimalanforderungen für das erste Projektgespräch](#) und die [PÜ32.5 Minimalanforderungen für das letzte Projektgespräch](#). Die Studenten ignorierten meist die Anforderungen und Hinweise und lieferten Basteleien. Darum ergänzte ich das Dokument [KE32 Kritik eines Entwurfs zu PÜ32 Projektübungsblatt: Codierung von Dateien](#). Die Studenten ignorierten oft die Entwurfskritik, Anforderungen und Hinweise und lieferten Basteleien. Darum ergänzte ich den [BE32 Beispielentwurf zu PÜ32 Projektübungsblatt: Codierung von Dateien](#) und Musterlösungsvarianten und funktionierte das Projekt von einer Übung in Lehrmaterial um.

... zum Lehrmaterial

Lesen Sie zuerst [PÜ3 Projektübungsblatt: Gemeinsames zu PÜ31 und PÜ32](#), dann dieses Projektübungsblatt! Studieren Sie dann den inakzeptabel schwachen und deshalb kritisierten Entwurf in [KE32 Kritik eines Entwurfs zu PÜ32 Projektübungsblatt: Codierung von Dateien](#)! Studieren Sie dann den [BE32 Beispielentwurf zu PÜ32 Projektübungsblatt: Codierung von Dateien](#) und die Musterlösungsvarianten unter

http://informatik.karlheinz-hug.de/Lehrmaterial/Informatik_3/C++/Coding

Geeignete Implementationssprachen: Alle.

Aspekte: Entwurf, Schnittstellenspezifikation, Modul, Klasse, Benutzung, Vererbung, Polymorphie, dynamisches Binden, Generizität, Austauschbarkeit, Kombinierbarkeit, Entwurfsmuster, Datei, Klassenbibliothek, Codierung, Verschlüsselung, Caesar, Vernam, Datenkompression.

PÜ32.2

Aufgabenbeschreibung

Sie entwickeln nach dem **EVA-Prinzip** funktionierende Programme:

Eingabe – Verarbeitung – Ausgabe.

Jedes Programm liest von einer gegebenen Eingabedatei, bearbeitet den Inhalt und schreibt den bearbeiteten Inhalt in eine Ausgabedatei. Die Verarbeitung besteht aus Codierungen – einer Verschlüsselung (Chiffrierung) oder einer Datenkompression oder beidem. Damit aus der Ausgabe wieder die ursprüngliche Eingabe erhalten werden kann, sind auch die umgekehrten Codierungen – Entschlüsselung (Dechiffrierung) und Expansion (Dekompression) – zu entwickeln. In funktionaler Schreibweise gelten mit der Funktion V und ihrer Umkehrfunktion V^{-1} die zum Testen wichtigen Formeln

$$V(E) = A \text{ und } V^{-1}(A) = E',$$

bei korrekter Implementation gilt $E = E'$, also

$$V^{-1}(V(E)) = E,$$

und auch

$$V(V^{-1}(A)) = A.$$

PÜ32.3

Hinweise zum Vorgehen und Umsetzen

Inkrementell vorgehen,
jederzeit testen

Datei kopieren

C++-filestream
benutzen

Funktionalen Teil gut
entwerfen

Verschlüsseln

- (1) Nach [MB Merkblatt MB.4\(9\)](#) erweitern Sie inkrementell ein stets übersetzbares und testbares Programm.
- (2) Das Projekt fokussiert zuerst auf **Dateiarbeit**: von einem Programm aus Dateien erzeugen, öffnen, lesen, schreiben, schließen. Die erste Stufe des Programms erzeugt nur eine **Kopie** der Eingabedatei, der Verarbeitungsteil bleibt fast leer. Dazu ist die Standardbibliothek der Wahlsprache nach Schnittstellen zum Dateisystem zu durchsuchen.
- (3) **Lieber neu als alt**: Für die Implementation mit C++ gibt es Dateiverarbeitung mit dem alten Datentyp *FILE* der C-Standardbibliothek und mit der neuen Klasse *fstream* (*filestream*) der C++-Standardbibliothek. Es ist kein Verbrechen, die älteren C-Dateien zu verwenden, aber wozu die Chance verschenken, die neueren C++-Dateien kennen zu lernen? Beispiele zur Dateiarbeit in C++ finden Sie zuhauf im Web, um schnell beginnen zu können. Die Qualität der Beispiele variiert stark, suchen Sie kritisch nach einer qualitätvollen Vorlage!
- (4) Nachdem Sie probiert haben, wie Sie Dateiströme programmieren, ist Entwurfsarbeit gefordert. Das Entwurfskonzept, der Programmierstil – prozedural, modular, objektorientiert – ist Ihnen bei diesem Projekt freigestellt. Nur alles in die *main*-Funktion zu stopfen ist inakzeptabel! Oder wollen Sie demonstrieren, dass Sie noch nicht einmal die ersten Lektionen von *Informatik 1* bewältigt haben?
Das **Infimum** ist ein schrittweise verfeinerter, prozeduraler Entwurf. Dann die Daten ordentlich gekapselt, die Prozeduren sinnvoll auf wenigstens zwei Übersetzungseinheiten verteilt, und schon haben Sie als **lokales Minimum** einen modularen Entwurf. Für die Implementation mit C++ gilt: Wer noch keine Include-Datei korrekt programmiert hat, der „kann kein C++“.
Höhere Entwürfe sind: Entwurf mit abstrakten Datentypen (ohne Vererbung), objektorientierter Entwurf (mit Vererbung). Wichtig bei jedem Entwurf sind **saubere Schnittstellen** zwischen den Einheiten.
- (5) Ein zweiter Fokus liegt auf **Verschlüsselungsverfahren**. Die zweite Stufe des Programms kann ein einfaches Verschlüsselungsverfahren enthalten: eine Caesar-Verschlüsselung oder die in *Informatik 1 Praktikum, Übungsblatt 5* vorgestellte angenäherte Vernam-Chiffrierung. Schlüssel werden als Kommandozeilenargumente übergeben.
- (6) **Caesar-Verschlüsselungen** verschieben byteweise nach dem Muster

chiffre [i] := clear [i] + key,

wobei automatisch modulo Byteanzahl (2^8) addiert wird (in C++ sind Bytes gleich Zeichen gleich Zahlen). Eine Schleife um die Zuweisung und der Käse(r) ist gegessen! Vergessen Sie künstlich aufgeblähte, schwache Varianten mit unnötig vielen Typanpassungen aus obskuren Webquellen! Beachten Sie, dass Caesar auch PDF-, JPG-, MPG-Dateien verschlüsseln kann, aber freilich nicht, wenn er nur Buchstaben verschiebt! Nutzen Sie die Beziehung

$$(V_{\text{Caesar}(\text{key})})^{-1} = V_{\text{Caesar}(-\text{key})}$$

um Codeduplikation zu vermeiden!

- (7) Die **Vernam-Chiffrierung** ist kaum aufwändiger zu programmieren als die leicht zu knackende Caesar-Verschlüsselung:

chiffre [i] := clear [i] **bit_xor** key [i **mod** key.size].

Der Schlüssel ist eine Reihung von Bytes. Mit zufälligem Schlüssel der Länge des Klartexts verschlüsselt Vernam perfekt sicher! Vorsicht am Ende des Schlüssels: zum Anfang zurück! Nutzen Sie die Beziehung

$$(V_{\text{Vernam}(\text{key})})^{-1} = V_{\text{Vernam}(\text{key})}$$

um Codeduplikation zu vermeiden!

- (8) **Raffinierte Verfahren** recherchieren Sie in der Literatur und im Web. Diese werden mathematisch und algorithmisch schnell anspruchsvoll – etwas für Spezialisten. Der didaktische Zweck dieses Projekts liegt mehr darin, dass Sie eine gute modulare oder objektorientierte Struktur entwerfen und implementieren, als darin, raffinierte Chiffrieralgorithmen aus dem Web zu kopieren. Der Entwurf sollte erlauben, sowohl byteweise als auch blockweise zu verschlüsseln, und die Verfahren auszutauschen und zu kombinieren.

Komprimieren

- (9) Ein zu (5) alternativer Fokus liegt auf verlustlosen **Datenkompressionsverfahren**. Die zweite oder dritte Stufe des Programms kann ein einfaches Kompressionsverfahren enthalten – wenn Sie eins finden. Raffinierte Verfahren recherchieren Sie wie in (8), sie werden aber auch schnell Spezialistensache. Statt zu versuchen, aufwändige Codierungsverfahren zu implementieren, sollten Sie sich auf Entwurfsfragen konzentrieren: Gute Schnittstellen entwerfen, mit einfachen Algorithmen implementieren, und sei es mit identischen Abbildungen der Art

target := source.

Austauschen

- (10) **Austauschbarkeit**: Sie haben Ihr Programm so modularisiert, dass Sie schnell ein Codierungsverfahren durch ein anderes ersetzen können. Wann ist ein konkretes Verfahren wählbar – zur Übersetzungszeit, zur Ladezeit oder zur Laufzeit? Ein guter objektorientierter Entwurf ermöglicht die Wahl des Verfahrens zur Laufzeit. Dazu definieren Sie für die Codierungsarten geeignete Schnittstellenklassen und entwickeln die konkreten Codierungsverfahren als Implementationsklassen zu diesen abstrakten Klassen. Wie man das macht? Mit Polymorphie, dynamischem Binden, Fabrikfunktion, wie es der *Eiffel-Rohbau* zu [PÜ21 Projektübungsblatt: Ampelsystem/Verkehrssimulation](#) unter

http://I3/Eiffel/EiffelStudio/light_simulation

in der Klassendatei light_simulation.e zeigt.

Kombinieren

- (11) Nachdem Sie sowohl eine Verschlüsselung als auch eine Kompression entwickelt haben, steht die **Kombination** beider Codierungen an. In welcher Reihenfolge? Erst verschlüsseln, dann komprimieren? Oder umgekehrt? Oder ist die Reihenfolge irrelevant?

Abstrahieren

- (12) **Abstraktion**: Verschlüsseln kann man byte- oder blockweise, komprimieren nur blockweise. Was folgt daraus für

- ◆ den Aufwand einfacher Verfahren?
- ◆ die Abstraktion von Chiffrierung und Kompression?

Worin unterscheiden sich Chiffrierung und Kompression? Was ist ihnen gemein? Wie kann man davon zu Codierung hin abstrahieren? Zwei, ja beliebig viele hintereinander ausgeführte Codierungen ergeben wieder eine Codierung. Was folgt daraus für Ihren Entwurf? Welches objektorientierte Entwurfsmuster bietet sich zur Kombination beliebiger Codierungen an?

- | | |
|------------|--|
| Verbessern | (13) Gebrauchstauglichkeit: Eine gute Realisierung von Kompressionsverfahren erkennt beim Expandieren, mit welchem Verfahren der Text komprimiert wurde und wählt dann das passende Expansionsverfahren automatisch. Auch wenn Sie zur Kompression die Identität nach (9) einsetzen, können Sie diese Zusatzaufgabe leicht lösen und damit verschiedene Kompressionsverfahren simulieren. |
| Testen | (14) Vertrauenswürdigkeit: Es reicht nicht, das Programm ängstlich mit wenigen Zeichen oder Zeilen Eingabetext zu testen. Unterziehen Sie Ihr Programm zünftigen Stresstests! Schafft es nicht nur kleine Textdateien, sondern auch PDF-, JPG-, MPG-Dateien mit einigen 100 Kilobytes bis Megabytes Größe? |

PÜ32.4

Minimalanforderungen für das erste Projektgespräch

Das in *MB Merkblatt, MB.6 Minimalanforderungen für das erste Gespräch eines Projekts* Verlangte ist hier konkretisiert und Grundlage für das Testat:

- (1) Liste offener Fragen zum Problem
- (2) Entwurfsskizzen:
 - (a) Physisches Modell des Problembereichs
 - (b) Entwurf des Kommandos der Kommandozeilenanwendung mit Syntax und Semantik der Argumente. Beispielansatz:


```
>coder Eingabedatei Ausgabedatei
```

 Durch weitere Argumente lässt er sich leicht ausbauen, etwa so:


```
>coder Eingabedatei Ausgabedatei /copy
>coder Eingabedatei Ausgabedatei /lencrypt /key=12
>coder Eingabedatei Ausgabedatei /decrypt=vernam /key=Schlüsseldatei
>coder Eingabedatei Ausgabedatei /compress
>coder Eingabedatei Ausgabedatei /decompress
```
 - (c) Entscheidungsgründe für Programmierstil: prozedural, modular, objekt-orientiert
 - (d) Entsprechend (c) ein Prozedur-, Modul- oder Klassendiagramm
 - (e) Nachweis, dass keine der Schwächen des inakzeptabel schwachen, kritisierten Entwurfs in *KE32 Kritik eines Entwurfs zu PÜ32 Projektübungsblatt: Codierung von Dateien* kopiert oder reproduziert ist
- (3) Liste benötigter Sprachkonstrukte
- (4) Vorgehensplan

PÜ32.5

Minimalanforderungen für das letzte Projektgespräch

Das in *MB Merkblatt, MB.7 Minimalanforderungen für das letzte Gespräch eines Projekts* Verlangte ist hier konkretisiert und Grundlage für das Testat:

- (1) Korrektheit
- (2) Benutzbarkeit

Dazu gehören Rückmeldungen an den Benutzer, ob das Kommando erfolgreich ausgeführt oder welcher Fehler erkannt wurde.
- (3) Testbarkeit

Obligatorische Kommandozeilenargumente: Quelldatei, Zieldatei, Operation, Schlüssel. Inakzeptabel, da nicht vernünftig testbar: Zieldatei = Quelldatei, da Überschreiben der Quelle benutzerfeindlicher Unsinn ist.

- (4) Zerlegung in mindestens zwei Übersetzungseinheiten, d.h. drei Quelldateien: *main.cpp*, *other.hpp*, *other.cpp*
Inakzeptabel: Alles in einer Datei, alles in *main*: Bisschen Zerlegung muss sein!
- (5) Trennung von E/A und Funktion
main.cpp enthält Konsol-E/A-Teile, *other.hpp/cpp* enthalten funktionale Teile ohne Konsol-E/A. Inakzeptabel, da nicht wiederverwendbar in GUI-Kontext: *cout* oder *cerr* << "Fehler" in *other.cpp*
- (6) Datenabstraktion
- (7) Nebeneffektfreie Schnittstellen
- (8) Ausnahmefreie Schnittstellen
- (9) Dokumentationssprache Englisch
- (10) Kopfkomentare in Quelldateien
- (11) Sinnvolle Namen
- (12) Dokumentierte Schnittstellen in Schnittstellendateien (*header file*, *x.hpp*)
- (13) Minimaler Funktionsumfang:
 - (a) Zum Ver-/Entschlüsseln Caesar mit beliebigem Schlüssel (***char*** *key*), der nicht nur Buchstaben, sondern alle ***char*** verschiebt
 - (b) Beliebige Dateien sind verschlüsselbar, nicht nur Textdateien
 - (c) Test: Das Programm kann eine Text- *und* eine JPG- oder PDF-Datei von mindestens 500 KB Größe in akzeptabler Zeit verschlüsseln und wieder korrekt entschlüsseln, sodass die Quelle wieder da ist.

PÜ32.6**Testate**

Datum,
Unterschrift des
Betreuers

Name des Studenten:

Abgenommene Leistungen

Offene Aufgaben