KV Betriebssysteme	Übung #5	SS 2002
Name:	Matr-Nr:	Gruppe:
Abgabe: 7./8.05.2002		Tutor:

Beispiel 10: Codierer/Decodierer

Unten finden Sie eine gemeinsame Schnittstelle für beliebige Codierer und Decodierer (oft auch Filter genannt). Diese übernehmen Daten und wandeln sie in ein anderes Format um (z. B. Zeilenumbrüche entfernen, Checksummen erstellen; Beispiele sind auch UUEncode, BinHex, BASE64, …). Zu dieser sollen zwei getrennte Implementationen erstellt werden: Eine Codierung von Binärdaten in Hexadezimaldarstellung und eine entsprechende Decodierung von Hexadezimaldarstellung zurück in Binärdaten.

Zusätzlich erforderlich ist ein Hauptprogramm, welches Daten aus einer Datei ausliest, diese einem (nur nach der Schnittstelle bekannten!) Codierer übergibt, und die Ergebnisse in eine andere Datei schreibt. Über ein Makefile (ebenfalls abzugeben) sollen daraus zwei Programme entstehen: Codierer (=Hauptprogramm + Codierungs-Implementation) und Decodierer (=Hauptprogramm + Decodierungs-Implementation). Das Hauptprogramm ist daher vom Codierer vollkommen unabhängig. Die tatsächliche Funktion entsteht erst durch das Linken!

Beispiel:

Eine Datei mit dem Inhalt „Test\nText“ würde codiert die folgende Datei ergeben: „546573740D0A54657874“

Implementierung:

Überlegen Sie sich was passiert, wenn bei der Hexadezimal-Decodierung insgesamt eine ungerade Anzahl von Zeichen übergeben werden: Die Abschluß-Funktion (finalize) muß hiermit umgehen können und eine Lösung anbieten. Dokumentieren sie was passiert, und warum Sie diese Lösung gewählt haben.

Sowohl Codierer wir auch Decodierer benötigen einen internen Buffer (Länge über eine Konstante definiert). Ist dieser voll, werden keine Daten mehr angenommen. Beachten Sie, daß sowohl beim Lesen wie auch beim Schreiben eine beliebige Anzahl von Bytes übernommen werden müssen. D. h. der Benutzer kann sowohl immer ein Byte schreiben (Vorsicht bei der Hex-Decodierung!) oder lesen (Vorsicht bei der Hex-Codierung!), aber auch gleich ganze Felder schicken (die eventuell nur teilweise übernommen werden falls der Buffer voll ist).

Die Parameter bei Initialisierung und Abschluß werden bei der Hex-Codierung nicht benötigt.

Beim Hauptprogramm ist darauf zu achten, daß eine ordentliche Fehlerbehandlung erfolgt, z. B. falls Dateien nicht geöffnet werden können. Zum Test soll das Hauptprogramm die Daten sowohl byteweise dem Codierer übergeben, als auch wieder byteweise von dort auslesen.

Schnittstelle (Coder.h):

```
#ifndef CODER_LIB
#define CODER_LIB


/* State: Not yet initialized. Only initializeCoder may be called! */
#define NOT_INITIALIZED 0
/* State: Currently performing coding. */
#define CODING               1
/* State: Finalized. writeToCoder amd finalize may not be called any more! */
#define FINALIZED            2


/* Define a "byte" datatype. */
typedef unsigned char byte;


/* Retrieve the name of the coder. */
const char* getCoderName();


/* Initializes the coder. May be called at any time. If the coder contains a buffer, it will be cleared. The meaning of the parameter depends on the coder. Return 0 on an error. */
int initializeCoder(void* param);


/* Writes bytes into the coder. The data is the first parameter, the number of bytes provided the second. Returns the number of bytes actually coded. This may be less or equal to len. Return 0 on errors or if there is no more room in an internal buffer. There is no way to distinguish an error from a full buffer. */
int writeToCoder(byte* input,int len);


/* Retrieves the number of bytes available for reading. */
int getAvailableCount();


/* Reads bytes from the coder. The first parameter is the buffer for the output, the second the number of bytes available in it. Returns the number of bytes actually written. This may be 0 if there is an error or no data is available. */
int getFromCoder(byte* output,int len);


/* Finalizes the coding. May be called only in state CODING. The meaning of the parameter depends on the coder. writeToCoder may not be called after this! Returns 0 on an error. */
int finalize(void* param);


/* Retrieves the current state of the coder.  See constants above! */
int getCoderState();


#endif
```