C-ÜBUNGSAUFGABEN DATEIEN 1

1)

a) Schreiben Sie ein Programm, das die Datei "mesk.txt" anlegt und den Text:
"Programmieren ist 1 Prozent Inspiration und 99 Prozent Transpiration" abspeichert.
b) Verändern Sie das obige Programm so, daß bei einem nochmaligen Aufruf des Programms der Inhalt der (jetzt schon existierenden) Datei "mesk.txt" nicht mehr verändert wird (d.h. die Datei wird weder überschrieben noch wird an das Dateiende etwas angefügt).
c) Schreiben Sie ein Programm, das den Text aus der Datei "mesk.txt" ausliest und auf dem Bildschirm ausgibt.
d) Der Name der Datei soll nun nicht mehr fest vorgegeben sein, sondern der Anwender soll bei den vorigen Teilaufgaben den Namen der Datei über Tastatur eingeben können.

2)

Schreiben Sie ein Programm, das den kompletten Text (nicht die Datei !) in der Datei "mesk.txt" löscht (d.h. z.B. alle Zeichen (Bytes) in der Datei mit dem Zeichen 'x' überschreibt).

3)

a) Schreiben Sie das Programm "kodierung", das die Datei verschlüsselt.
Eine Verschlüsselungsmethode:
Jedes Byte wird durch den in der ASCII-Tabelle nachfolgenden ASCII-Wert kodiert.
Das Zeichen mit dem ASCII-Wert 65 (ASCII-Zeichen A) wird durch das Zeichen mit dem ASCII-Wert 66 (ASCII-Zeichen B) kodiert.
Das Zeichen mit dem ASCII-Wert 255 wird durch das Zeichen mit dem ASCII-Wert 0 kodiert.

Beispiel:
Die Datei mit dem Inhalt (Bytes)

ASCII-Werte:				
65	71	255	97	105

wird kodiert zu:

66	72	0	98	106

ASCII-Zeichen:				
A	G		a	i

B	H		b	j

b) Schreiben Sie das Programm "dekodierung", das die oben verschlüsselte Datei wieder dekodiert.
c) Erzeugen Sie mit einem Textverarbeitungsprogramm (wie z.B. Word) eine Textdatei, mit einem Compiler (wie z.B. MS VC++) eine exe-Datei, mit einem Malprogramm (wie z.B. paint) eine Bilddatei und verschlüsseln diese jeweils. Danach entschlüsseln Sie jeweils diese Dateien und versuchen Sie wieder mit dem entsprechenden Programm zu öffnen.
d) Überlegen Sie sich andere Verschlüsselungsverfahren und implementieren Sie diese.
Beispiel:
Die Reihenfolge der Bytes in einer Datei wird umgekehrt. Das letzte Byte wird zum ersten.
e) Für eine Bilddatei wäre es zu Testzwecken evtl.sinnvoll nur einen Teil der Bytes zu verschlüsseln (wie sieht das Bild dann aus ?).

4)

Schreiben Sie folgende Programme:
a) Kopieren Sie eine Datei (Quelldatei), deren Namen der Anwender über Tastatur eingibt. Der Name der kopierten Datei (Zieldatei) soll der Anwender ebenfalls eingeben können. Existiert die Zieldatei schon, soll der Anwender gefragt werden, ob diese überschrieben werden soll.
b) Eine Datei soll gelöscht werden, deren Dateiname der Anwender eingibt.
c) Eine Datei soll verschoben werden, deren Dateiname der Anwender eingibt.

5) Vergleich == und =

In C bzw. C++ wird die mathematische Gleichheit durch "= =" und die Zuweisung durch "= " dargestellt. Vermutung: Dies wurde gemacht, um dem Programmierer Schreibarbeit abzunehmen, denn die Zuweisung kommt in einem C bzw. C++ - Programm öfter vor als die Gleichheit. Bestätigen oder verwerfen Sie diese Vermutung, indem Sie verschiedenen C - Programmen untersuchen.

6)

a) Überschreiben Sie an einer bestimmten Stelle einer Datei das dort stehende Byte durch ein anderes, das der Anwender über Tastatur eingibt.
b) Fügen Sie an einer bestimmten Stelle einer Datei ein Byte ein, das der Anwender über Tastatur eingibt (alle ab dieser Stelle stehenden Bytes der Datei werden in Richtung Dateiende verschoben).

7)

Um die Datei "`c:\geheim.txt`" zu verschlüsseln, soll der Dateininhalt in umgekehrter Reihenfolge angeordnet werden.

8) Dateien-Tools

8.1)

Ermitteln Sie, ob zwei Dateien genau den gleichen Inhalt haben.

8.2)

Berechnen Sie die Grösse (Länge) einer Datei.

8.3)

Ermitteln Sie, ob die ersten n Bytes zweier Dateien gleich sind. Die Zahl n soll der Anwender eingeben.

8.4)

Fügen Sie die ersten n Bytes einer Datei an eine andere Datei an.

8.5)

Fügen Sie den Inhalt einer Datei an eine andere Datei an.

8.6)

Ermitteln Sie, an welcher Stelle in einer Datei ein bestimmtes Byte das erste Mal vorkommt.

8.7)

Ermitteln Sie, an welcher Stelle in einer Datei ein bestimmtes Byte das letzte Mal vorkommt.

8.8)
Wandeln Sie in einer Datei alle Großbuchstaben in Kleinbuchstaben um. Schauen Sie sich dazu die ASCII-Tabelle an.

8.9)
Ersetzen Sie die ersten n Bytes einer Datei durch ein anderes Zeichen.

8.10)
Überschreiben Sie ab einer bestimmten Stelle einer Datei die dort stehenden Bytes durch eine Zeichenkette, die der Anwender über Tastatur eingibt.

8.11)
Fügen Sie ab einer bestimmten Stelle einer Datei eine Zeichenkette ein, die der Anwender über Tastatur eingibt (alle ab dieser Stelle stehenden Bytes der Datei werden in Richtung Dateiende verschoben).

8.12)
Um seine Daten zu sichern, komprimiert Herr X diese (z.B. mit 7zip) zu einer exe Datei, die er dann auf einem USB Stick sichert und dann von dort auf seinen Geschaftsrechner kopiert (und wieder dekomprimiert). Leider passt diese zu große Datei nicht auf den USB Stick.
Deshalb:
Erzeugen Sie ein Programm, das eine Datei in 2 Dateien zerlegt (splittet) und auch wieder zusammenfügen kann.

8.13)
Schreiben Sie das Programm "kompress", das eine Datei komprimiert, indem es für jedes Zeichen die Anzahl der benachbarten, gleichen Zeichen abspeichert.
Beispiel:

Datei:	8	8	8	8	3	105	105	105	105	105	105

komprimierte Datei:	8	4	3	1	105	6

Bemerkung:
a) Die ganzen o.g. Aufgaben lassen sich auch auf Zeichenketten (Felder mit Datentyp char) übertragen. Man hat also noch mehr Aufgaben zum Üben !!

b) Überlegen Sie als Programmierer, wie das Programm auf Eingabefehler des Anwenders reagieren soll:
Beispiele:
Anwender gibt nicht existierende Datei an, die ersten 10 Zeichen einer 4 Byte grossen Datei sollen durch 'x' ersetzt werden, usw.

9)
Der Tennisverein TSV Ballingen lädt zu einem Turnier mit n = 16 Teilnehmern ein, die jeweils im Doppel gegeneinander antreten.
Die Teilnehmer sind in der Datei "teilnehmer.txt" gespeichert.
Geben Sie alle mögliche Doppelpaarungen an und speichern diese in der Datei "paarungen.txt".

10) Der Schwallomat

Die folgenden Wörter heißen Modalverben:
dürfen, können, mögen, müssen, sollen, wollen.
Ein versierter Schwätzer kann damit nichtssagende Redewendungen konstruieren.
Wir werden müssen wollen,
Sie werden können dürfen,
wir müssen werden wollen,
...

Schreiben Sie ein Programm, das alle möglichen Redewendungen in eine Datei schreibt. Diese muss mit dem Editor geöffnet werden können, wobei jede Redewendung in einer Zeile stehen muß.

11) Dateien-Tools

Schreiben Sie Funktionen zur Manipulation von Dateien

11.1)

strcmp(...)
vergleicht (engl: string compare) 2 Dateien lexikografisch (wie z.B. zwei Namen im Telefonbuch).

11.2)

concat(...)
Eine Dateien wird an die andere angefügt (engl: concatenate).

11.3)

contains(...)
prüft nach, ob eine Datei eine andere beinhaltet (engl: contain).

11.4)

endsWith(...)
prüft nach, ob eine Datei mit einer bestimmten Zeichenfolge endet.

11.5)

indexOf(...)
Berechnet den Index (das erstmalige Vorkommen) eines Zeichens in einer Datei.

11.6)

lastIndexOf(...)
Berechnet den Index (das letztmalige Vorkommen) eines Zeichens in einer Datei.

11.7)

length(...)
berechnet die Länge einer Datei.

11.8)

replaceFirstChar(...)
Ersetzt in einer Datei das erste Auftreten eines bestimmten Zeichens durch eine bestimmte Zeichenfolge.

11.9)
replaceAllChars(...)
Ersetzt in einer Datei jedes Auftreten eines bestimmten Zeichens durch eine bestimmte Zeichenfolge.

11.10)
replaceFirstString(...)
Ersetzt in einer Datei das erste Auftreten einer bestimmten Zeichenfolge durch eine bestimmte Zeichenfolge.

11.11)
replaceAllStrings (...)
Ersetzt in einer Datei jedes Auftreten einer bestimmten Zeichenfolge durch eine bestimmte Zeichenfolge.

11.12)
replace(...)
Ersetzt in einer Datei jedes Auftreten eines bestimmten Zeichens durch ein anderes Zeichen.

11.13)
split2(...)
teilt eine Datei in 2 Teilzeichenfolgen, wobei das erstmalige Vorkommen eines bestimmten Zeichens den ersten Dateiinhalt angibt.
Beispiel:
Dateiinhalt: "abcasdfgfdgdf"
Begrenzer: ' f '
Die zwei Dateien mit ihren Dateiinhalten sind dann:
"abcasd" und "gfdgdf"

splits2(char str[], char begrenzer[], int i)
ähnlich wie split(..), nur komplexer.
Beispiele:
str: ((a+b)*c)+(d*b))
begrenzer: +*
splits2(str, begrenzer, 1) liefert: " ((a" und "b)*c)+(d*b))"
splits2(str, begrenzer, 2) liefert: "((a+b)" und " c)+(d*b))"
splits2(str, begrenzer, 4) liefert: "((a+b)*c)+(d" und "b))"

11.14)
startsWith(...)
prüft nach, ob eine Datei mit einer bestimmten Zeichenfolge beginnt.

11.15)
substring(...)
berechnet den Dateiinhalt einer Datei, die an einem bestimmten Index beginnt.
Beispiel:
"abcdefghabc", 5 -->"fghabc"

11.16)
substringBetween(...)
berechnet den Dateiinhalt einer Datei, die an einem bestimmten Index beginnt und an einem anderen Index endet.
"abcdefghabc", 5, 7 --> "gha"

12) Aufgaben in der Klasse gerecht verteilen

In einer Schulklasse fallen immer wieder außerordentliche Arbeiten an, die von einzelnen Schülern verrichtet werden müssen. Der jeweilige Schüler wird durch Zufall ermittelt. Allerdings kommt dieser Schüler bei den nächsten zu verrichtenden Arbeiten so lange nicht mehr dran, bis alle Schüler genau einmal dran gekommen sind.
Die Schülernamen werden in die Textdatei schueler.txt eingegeben (mit einem einfachen Texteditor).
a) Implementieren Sie diesen Fall für Ihre Schulklasse
b) Machen Sie das Programm so allgemein, daß es auch den Fall zuläßt, daß Schüler - während des Schuljahres - neu hinzukommen bzw. abgehen.

C-ÜBUNGSAUFGABEN DATEIEN 2

13)

Es soll das 120-malige Werfen eines Würfels auf die folgende Art simuliert werden:
Schreiben Sie das Programm "wsim" das **genau einmal** das Werfen eines Würfels simuliert und das Wurfergebnis (die "Augenzahl") in der Datei "wuerfeln.txt" am Ende dieser Datei abspeichert.
Beim ersten Aufruf (und Programmablauf) des Programms muß diese Datei natürlich zuerst erzeugt werden.
Am Dateianfang muß der Mittelwert aller in der Datei gespeicherten Wurfergebnisse abgespeichert werden, danach die Anzahl der Würfe und am Dateiende das Wurfergebnis selbst.

Beispiel:
Wenn beim ersten Aufruf des Programms eine 6 gewürfelt wurde, sieht die Datei nach dem Programmablauf also wie folgt aus:

6	1	6

Wenn beim zweiten Auruf des Programms eine 5 gewürfelt wurde, sieht die Datei nach dem Programmablauf also wie folgt aus:

5,5	2	6	5

Wenn beim dritten Auruf des Programms eine 1 gewürfelt wurde, sieht die Datei nach dem Programmablauf also wie folgt aus:

4	3	6	5	1

...

Beispiel für die Verwendung des Zufallsgenerators: (siehe Hilfe-Menü in VC++)

```
#include "stdafx.h"
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

void main(void){
  int erg;
  srand( (unsigned)time( NULL ) );
  erg = rand();
  printf("Zufallszahl=%d\n",erg);
}
```

14) Klassenarbeiten analysieren

In der mit einem enfachen Editor erstellten Textdatei **meineNoten.txt** haben die Daten einer Klassenarbeit die Form des folgenden Beispiels:

```
2.4.13#9:30#Deutsch#3,4#Maier#Nachtermin
31.1.12#11:15#Biologie#4,7#Müller
7.12.15#7:40#SAE#2#Maier#Nachtermin
10.11.15#13:00#Deutsch#1,9#Schulz
7.6.13#9:30#Deutsch#4,2#Maier
12.4.13#9:30#Biologie#2,5#Müller#Nachtermin
13.9.15#9:30#Deutsch#2,1#Maier#Nachtermin
13.3.13#7:40#Deutsch#4,2#Schmidt
...
```

Bei der Analyse der Noten könnten folgende Fragen auftauchen:
a) Wie groß ist der Notendurchschnitt in einem bestimmten Fach bei einem bestimmten Lehrer?
b) Wie groß ist der Notendurchschnitt in einem bestimmten Fach bei einem bestimmten Lehrer in einem bestimmten Schuljahr (1.9.Jahr_x bis 31.7.Jahr_x+1) ?
c) Wie groß ist der Notendurchschnitt in einem bestimmten Fach bei einem bestimmten Lehrer?
beim Nachtermin ? (Nachtermin ist die Klassenarbeit für Nachzügler).
d) Wie groß ist der Notendurchschnitt in einem bestimmten Fach bei einer bestimmten Uhrzeit?
(man könnte z.B. nachweisen, daß die Klassenarbeiten in der 1. Schulstunde schlechter ausfallen).
e) Erfinden Sie selbst noch weitere "Abfragen".

C-ÜBUNGSAUFGABEN DATEIEN 3

15) Experiment: Maschinenbefehle (Maschinencode) in exe-Dateien

In einer exe-Datei befinden sich die eigentlichen Befehle (Maschinenbefehle) eines Programms.
Jeder Maschinenbefehl besteht aus einer Folge von Bytes. Wenn man ein Byte eines Maschinenbefehles ändert, "stürzt" das Programm ab.
Dies kann man in einem Experiment selbst ausprobieren:
Erstellen Sie mit der Visual C++ Entwicklunsumgebung ein beliebiges Programm.
Mit dem Debugger springt man dann zur 1. Anweisung dieses Programms. Diese Anweisung wird nicht im Maschinencode angezeigt (sondern der Quellcode von C). Damit man die Maschinenbefehle ansehen kann, macht man folgendes:
Ansicht ---> Debug-Fenster ---> Disassemblierung
Man sieht jetzt zwar die Adressen und die Assembler-Befehle, aber noch nicht die Maschinenbefehle (als Folge von Bytews). Dies macht man jetzt wie folgt:
Rechte Maustaste im Disassemblierungsfenster ---> Bei "Code Bytes" einen Haken machen.
(Alternative Möglichkeit:
Extras ---> Optionen ---> Reiter Debug ---> Bei "Code Bytes" einen Haken machen.)

Jetzt kann man sich einige Bytes dieser nun folgenden Maschinenbefehle dieses Programms merken und diese in der exe-Datei suchen.
Wie sucht man diese Bytes in der exe-Datei ?
Man benötigt dazu einen sogenannten Hex-Editor, in dem man die einzelnen Bytes der Datei anschauen kann. Falls man keinen Hex-Editor besitzt, wird die Entwicklungsimgebung von MSVC++ dazu zweckentfremdet:
Datei ---> öffnen... ---> Bei öffnen als nicht Auto sondern binär auswählen ---> Dann die exe Datei auswählen, die geöffnet werden soll.
Jetzt werden die Bytes der Datei angezeigt. Nun kan man man nach den Bytes suchen:
Bearbeiten ---> suchen

Jetzt kann man sein Experiment zu Ende bringen, indem man ein Byte eines Maschinenbefehls ändert und dann die exe-Datei startet.

Programmieraufgabe (Veränderung dieses Maschinencodes):
Eine exe-Datei enthält Maschinencode.
Um den Maschinencode einer exe-Datei zu überschreiben, ist es sinnvoll nicht an den Dateianfang zu schreiben, da sich dort der Header (Info über das Programm und die Datei) befindet (eher in die "Mitte" der Datei). Schreiben Sie ein Programm, das an einer bestimmten Stelle der Datei eine Zeichenkette schreibt. Sichern Sie den zu überscheibenden Teil der Datei in einer anderen Datei, so dass man die Datei wieder reparieren kann.

16) Wunderbare Zahlen oder wie ein "Universum" expandiert

A1) Es sollen die "wunderbaren Zahlen" durch Expansion entwickelt werden:
Aus den bestehenden Zahlen werden immer wieder nach einer bestimmten Vorschrift neue gebildet und an das Dateiende angefügt (aber nur wenn sie nicht darin schon enthalten sind).
Vorschrift:
Nimm jeweils 2 verschiedene Zahlen, bilde deren Summe und füge sie an das Dateiende an, aber nur, wenn diese nicht schon in der Datei entahlten sind.
Man beginnt z.B. mit
{3 ; 5}
1. Expansion: {3 ; 5 ; 8}
2. Expansion: {3 ; 5 ; 8 ; 11 ; 13}
3. Expansion: {3 ; 5 ; 8 ; 11 ; 13 ; 14; 16 ; 18 ; 19 ; 21 ; 24}
...
Wenn man "unendlich" oft expandiert, bekommt man die Menge der wunderbaren Zahlen.

A2) Variationen der vorigen Aufgabe
V1) Es können andere Anfangszahlen (und eine andere Anzahl) verwendet werden.
V2) Es können andere Vorschriften benutzt werden, wie z.B:
(x , y) --> x + y , falls $x \neq y$
(x , y) --> | x- y | , falls $x \neq y$
(x , y) --> x + y , falls $x \neq y$ und nicht x und y beide gerade Zahlen

Man könnte sich folgende Fragen überlegen:
F1) Gibt es eine Zahl z, aber der alle darauffolgenden Zahlen z+1, z+2, z+3, ...(ohne Lücke) in der Menge der wunderbaren Zahlen vorkommen ?
F2) Kann man die Anfangszahlen so vorbelegen (und die Vorschrift so gestalten) , daß dies gelingt ?
F3) usw.