

Aufgabenblatt 7

letzte Aktualisierung: 25. Oktober, 15:06 Uhr

(a51bdd10e1ff2831b7e5cc71160dd1bc5d0b23d)

Ausgabe: Mittwoch, 25.10.2017

Abgabe: Montag, 30.10.2017, 09:59

Thema: Strings & Debugging

Abgabemodalitäten

1. Die Aufgaben des C-Kurses bauen aufeinander auf. Versuche daher bitte Deine Lösung noch am gleichen Tag zu bearbeiten und abzugeben.
2. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des tubIT/eecsIT mittels `gcc -std=c99 -Wall` kompilieren.
3. Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Nur wenn ein Test in Osiris angezeigt wird ist sichergestellt, dass die Abgabe erfolgt ist.
4. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben.
5. Die Abgabe erfolgt in folgendem Unterordner:
`ckurs-wise1718/Studierende/<L>/<tubIT-Login>/Abgaben/Blatt0<X>`
wobei `<L>` durch den ersten Buchstabe des TUBIT-Logins und `<X>` durch die Nummer des Aufgabenblattes zu ersetzen sind. Die Ordner werden automatisch angelegt sobald die Abgabe freigeschaltet wird.
6. Benutze für alle Abgaben soweit nicht anders angegeben das folgende Namensschema:
`ckurs_blatt0<X>_aufgabe0<Y>.c` wobei `<X>` und `<Y>` entsprechend zu ersetzen sind. Gebe für jede Unteraufgabe genau eine Quellcodedatei ab.
7. Du darfst den Abgabeordner für das Blatt nicht selbst erstellen, das machen wir jeden Morgen kurz nach 8 Uhr!
8. Du musst aber den Befehl `svn up` auf der obersten Verzeichnisebene des Repositories (also in `ckurs-wise1718`) ausführen um alle Änderungen vom Server abzuholen.
9. Im Abgaben-Ordner gelten einige restriktive Regeln. Dort ist nur das Einchecken von Dateien mit den in der Aufgabe vorgegebenen Namen erlaubt, ausserdem werden die Abgabefristen vom Server überwacht. Beachte eventuelle Fehlermeldungen beim SVN-Commit. Lade nur Dateien hoch, die Du selbst bearbeiten sollst, insbesondere also keine Vorgaben.
10. Es gibt einen Ordner 'Arbeitsverzeichnis', in dem Du Dateien für Dich ablegen kannst.
11. Die Ergebnisse der automatischen Tests kannst Du auf OSIRIS einsehen:
<https://teaching.inet.tu-berlin.de/services/osiris-wise1718/>

1. Aufgabe: Strings formatieren und zusammenfügen (unbewertet)

In Listing 1 findest Du vordefinierte Variablen, aus denen sich ein Datum ergibt. Ein Datum besteht aus einem Wochentag, einem Tag, einem Monat und einer Jahreszahl. Füge diese vier Elemente zu einem korrekten Datum zusammen. Der String muss das Format "Freitag, der 13. Mai 1927" befolgen.

Diese Aufgabe dient dazu, Dich mit der Funktion `snprintf` vertraut zu machen. Prinzipiell funktioniert `snprintf` wie `printf`. Der Unterschied besteht in den ersten zwei Argumenten der Funktion. `snprintf` bekommt zuerst einen char-Pointer (`char*`), an dessen Ziel die Zeichenkette gespeichert wird, anstatt sie wie bei `printf` auszugeben. Statt auf die Standardausgabe "schreibt" `snprintf` also in einen String. Als zweites Argument bekommt `snprintf` die maximale Länge des geschriebenen Strings, um zu vermeiden, dass zu viele Zeichen in den Speicher geschrieben werden. Die restlichen Parameter funktionieren genau wie bei `printf`. Beachte, dass `snprintf` den String mit einem Nullbyte abschließt.

Hinweis: Ihr könnt zur Vereinfachung annehmen, dass die Länge des Datums maximal 100 Zeichen betragen kann.

Listing 1: String zusammensetzen

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main() {
6     char wochentag[] = "Freitag";
7     int tag = 13;
8     char monat[] = "Mai";
9     int jahr = 1927;
10    char *string; // Hier soll das Datum hineingeschrieben werden!
11    const size_t kapazitaet; // Maximale Laenge des Datumstrings!
12
13    // Hier implementieren und dynamisch Speicher reservieren
14
15    printf("%s\n", string);
16    // Speicher freigeben
17
18    return 0;
19 }
```

2. Aufgabe: Census Daten (3 Punkte)

In der Datei "staedte.csv" sind die Städte eines Bundeslands, das Bundesland selbst und die Anzahl der Bewohner dieser Städte, getrennt durch Semikolons, aufgelistet.

Schreibe ein Programm, das für ein gegebenes Bundesland alle Städte findet, in denen mindestens eine vorgegebene Anzahl *n* an Menschen leben. Die Ausgabe soll mit Hilfe der Funktion `write_file` in die Datei "resultat.txt" erfolgen, und für jeden Eintrag eine Zeile enthalten, die das Format *Die Stadt Nürnberg hat 505664 Einwohner.* befolgt. Zur Vereinfachung kannst du davon ausgehen, dass die Länge dieses Strings niemals 100 Zeichen überschreitet.

In dieser Aufgabe sollen das Bundesland und die Anzahl *n* als Parameter direkt beim Aufruf des Programms übergeben werden. Einen beispielhaften Aufruf mit Parametern kannst du in Listing 2 Zeile 3 sehen. In C stehen die übergebenen Parameter als `char** argv` zur Verfügung. In `argv[1]` und `argv[2]` werden jeweils das erste und das zweite Argument übergeben. (`argv[0]` ist für den Programmnamen reserviert.)

Listing 2: Programmbeispiel

```
1 > gcc -std=c99 -Wall ckurs_blat07_aufgabe02.c input3.c \
2   -o ckurs_blat07_aufgabe02
3 > ./ckurs_blat07_aufgabe02 100 Bayern
4 > cat resultat.txt
5 Die Stadt München hat 1353186 Einwohner.
6 Die Stadt Nürnberg hat 505664 Einwohner.
7 Die Stadt Augsburg hat 264708 Einwohner.
8 Die Stadt Regensburg hat 135520 Einwohner.
9 Die Stadt Würzburg hat 133799 Einwohner.
10 Die Stadt Ingolstadt hat 125088 Einwohner.
11 Die Stadt Fürth hat 114628 Einwohner.
12 Die Stadt Erlangen hat 105629 Einwohner.
```

Wie Du in Listing 2 sehen kannst, kompilieren wir die Aufgabe mit einer Bibliothek `input3.c`. Diese Bibliothek stellt die folgenden Funktionen zur Verfügung:

- **int read_file(char *dateiname, char laender[][], char staedte[][], int
↪ bewohner [])**
Diese Funktion liest Bundesländer, Städte und Bewohnerzahlen aus der Datei `dateiname` in die Arrays `laender`, `staedte` und `bewohner`.
- **void write_file(char *result[], int len)**
Diese Funktion schreibt eine Anzahl von `len` Strings aus dem Array `result` auf einzelne Zeilen der Datei `resultat.txt`. Die Datei wird bei jedem Aufruf des Programms überschrieben.

Für das Vergleichen von Strings sowie zum Kopieren sollst Du die in der Vorlesung dargestellten Funktion benutzen (`strcmp`, `snprintf`, ...).

Um die Hausaufgabe zu vereinfachen, bitten wir Dich die vorgegebene Programmstruktur zu verwenden (siehe Listing 3). Die Abgabe muss folgenden Kriterien entsprechen:

- Das korrekte Resultat soll mittels `write_file` in die Datei `resultat.txt` ausgegeben werden.
- Die Datei `input3.c` darf nicht verändert werden.
- Zusätzlicher Speicher muss dynamisch reserviert werden.
- Dynamisch reservierter Speicher muss wieder freigegeben werden.
- Es werden keine zusätzlichen Bibliotheken zu den in der Vorgabe verwendeten benutzt.

Listing 3: Mögliche Programmstruktur

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "input3.h"
5
6 /* Die Konstanten:
7  * int MAX_LAENGE_STR - die maximale String Länge
8  * int MAX_LAENGE_ARR - die maximale Array Länge
9  * sind input3.c auf jeweils 255 und 100 definiert
10 */
11
12 int main(int argc, char **argv) {
13     if (argc < 3) {
14         printf("Aufruf: _%s_<anzahl>_<bundesland>\n", argv[0]);
15         printf("Beispiel: _%s_100_Bayern\n", argv[0]);
16         printf("Klein-/Gro ß schreibung_beachten!\n");
17         exit(1);
18     }
19     int anzahl = atoi(argv[1]);
20     char *bundesland = argv[2];
21
22     // Statisch allozierter Speicher
23     char staedte[MAX_LAENGE_ARR][MAX_LAENGE_STR];
24     char laender[MAX_LAENGE_ARR][MAX_LAENGE_STR];
25     int bewohner[MAX_LAENGE_ARR];
26
27     int len = read_file("staedte.csv", staedte, laender, bewohner);
28
29     // Hier implementieren
30
31     // Mithilfe von write_file(...) soll das Ergebnis in die "resultat.txt"
32     // geschrieben werden.
33
34     // Dynamisch allozierter Speicher muss hier freigegeben werden.
35 }
```

Checke die Abgabe im SVN ein, wie unter “Abgabemodalitäten” beschrieben.

Erklärung zur Verwendung der write_file Funktion

Da es Fragen zur Verwendung der Funktion write_file gab, möchten wir diese hier kurz beschreiben. Betrachte dazu zunächst den Code in Listing 4, den Du mittels

```
1 > gcc -std=c99 -Wall ckurs_blat07_write_file_erklaerung.c \
2   input3.c -o ckurs_blat07_write_file_erklaerung
3 > ./ckurs_blat07_write_file_erklaerung
```

kompilieren und ausführen kannst.

Es wird zunächst das Array string_array vom Typ char* alloziert. Somit kann an jeder Position des Arrays string_array ein Pointer auf einen "String" abgelegt werden. Anschließend wird statisch Speicher als char Array für die einzelnen Zeilen, die geschrieben werden sollen, alloziert (Zeilen 12-14). Im Folgenden wird dann die Funktion snprintf der string.h Bibliothek benutzt, um den allozierten Speicher mit Werten zu befüllen (Zeilen 17-19). Letztlich werden im string_array noch die Pointer auf die einzelnen Zeilen-Strings abgelegt und dann mittels der Funktion write_file die Datei resultat.txt geschrieben (Zeilen 22-24). Abbildungen 1 bis Abbildung 3 verdeutlicht die Speicherbelegung zu den verschiedenen Zeitpunkten der Programmausführung exemplarisch.

Hinweis: Beachte, dass bei der Aufgabe 2 dieses Blattes der Speicher für die einzelnen Zeilen dynamisch – mittels malloc – alloziert werden soll.

Listing 4: ckurs_blat07_write_file_erklaerung.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "input3.h"
5
6 int main(int argc, char **argv) {
7     // lege Speicher an, um die Pointer zu den einzelnen Zeilen zu speichern
8     char* string_array[5];
9
10    // reserviere den Speicher für den Inhalt der Zeilen statisch
11    // dies sollte in Aufgabe 2 dynamisch mittels malloc geschehen
12    char erste_zeile[50];
13    char zweite_zeile[50];
14    char dritte_zeile[50];
15
16    // beschreibe den Speicher mit entsprechenden Informationen
17    snprintf(erste_zeile, 50, "Erste_Stadt");
18    snprintf(zweite_zeile, 50, "Zweite_Stadt");
19    snprintf(dritte_zeile, 50, "Dritte_Stadt");
20
21    // lege die Pointer im Array ab
22    string_array[0] = erste_zeile;
23    string_array[1] = zweite_zeile;
24    string_array[2] = dritte_zeile;
25
26    // schreibe die Datei mit den entsprechenden 3 Zeilen
27    write_file(string_array, 3);
28 }
```

	Adresse im Speicher	Inhalt des Speichers	
	:	:	
string_array	0x603030	0x00...00	= string_array[0]
&string_array[1]	0x603038	0x00...00	= string_array[1]
&string_array[2]	0x603040	0x00...00	= string_array[2]
&string_array[3]	0x603048	0x00...00	= string_array[3]
&string_array[4]	0x603050	0x00...00	= string_array[4]
	:	:	

Abbildung 1: Zustand des Arrays result_array in Zeile 8: die char Pointer zeigen auf NULL.

	Adresse im Speicher	Inhalt des Speichers	
	:	:	
string_array	0x603030	0x00...00	= string_array[0]
&string_array[1]	0x603038	0x00...00	= string_array[1]
&string_array[2]	0x603040	0x00...00	= string_array[2]
	:	:	
erste_zeile	0x60faa0	"Erste Stadt\0"	
	:	:	
zweite_zeile	0x60fba0	"Zweite Stadt\0"	
	:	:	
dritte_zeile	0x60fca0	"Dritte Stadt\0"	
	:	:	

Abbildung 2: Zustand des Speichers nach dem Allozieren und Schreiben der (String-)Arrays erste_zeile, zweite_zeile und dritte_zeile in Zeile 20.

	Adresse im Speicher	Inhalt des Speichers	
	:	:	
string_array =	0x603030	0x60faa0	= erste_zeile
&string_array[1] =	0x603038	0x60fba0	= zweite_zeile
&string_array[2] =	0x603040	0x60fca0	= dritte_zeile
&string_array[3] =	0x603048	0x00...00	
&string_array[4] =	0x603050	0x00...00	
	:	:	
erste_zeile =	0x60faa0	"Erste Stadt\0"	
	:	:	
zweite_zeile =	0x60fba0	"Zweite Stadt\0"	
	:	:	
dritte_zeile =	0x60fca0	"Dritte Stadt\0"	
	:	:	

Abbildung 3: Zustand des Speichers nach der Ausführung der Zeilen 22-24: das Array string_array zeigt nun auf die eigentlichen Strings erste_zeile, zweite_zeile und dritte_zeile.