Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2015/16
Feldmann / Semmler / Lichtblau / Streibelt / Rost

Aufgabenblatt 7

letzte Aktualisierung: 23. Oktober, 12:32 Uhr

(12f532beffaa97787b9ce840937edfde9a6dcd29)

Ausgabe: Mittwoch, 21.10.2015
Abgabe: spätestens Montag, 26.10.2015, 21:59

Thema: *Strings & Debugging*

1 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/IRB mittels `gcc -std=c99 -Wall` kompilieren.
3. Die Abgabe erfolgt ausschließlich über SVN.
4. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben.
5. Die finale Abgabe erfolgt in folgendem Unterordner:

   `ckurs-ws1516/Studierende/<L>/<tubIT-Login>@TU-BERLIN.DE/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-ws1516`) ausführen um alle Änderungen vom Server abzuholen.
9. Im Abgaben-Ordner gelten einige restriktive Regeln. Dort ist, je nach Aufgabe, nur das Einchecken von Dateien mit der Endung `.txt` und `.c` erlaubt, die nach dem Namensschema für Abgaben benannt sind. Beachte eventuelle Fehlermeldungen beim SVN-Commit. Lade nur Dateien hoch, die Du selbst bearbeiten sollst, insbesondere also keine Vorgaben.
10. Es gibt einen Ordner Workdir, 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/tubitauth/osiris/

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 `sprintf` vertraut zu machen. Prinzipiell funktioniert `sprintf` wie `printf`. Der Unterschied besteht im ersten Argument der Funktion. `sprintf` bekommt einen char pointer (`char*`), an dessen Ziel die Zeichenkette gespeichert wird, die bei `printf` ausgegeben würde. Statt auf die Standardausgabe "schreibt" `sprintf` also in einen String. Die restlichen Parameter funktionieren genau wie bei `printf`. Beachte, dass sprintf den String mit einem Nullbyte abschließt.

Listing 1: String zusammensetzen

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char wochentag[] = "Freitag";
    int tag = 13;
    char monat[] = "Mai";
    int jahr = 1927;
    char *string; // Hier soll das Datum hineingeschrieben werden!

    // Hier implementieren und dynamisch Speicher reservieren

    printf("%s\n", string);
    // Speicher freigeben

    return 0;
}
```

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

```
> gcc -std=c99 -Wall ckurs_blatt07_aufgabe02.c input3.c \
      -o ckurs_blatt07_aufgabe02
> ./ckurs_blatt07_aufgabe02 100 Bayern
```

```
> cat resultat.txt
Die Stadt München hat 1353186 Einwohner.
Die Stadt Nürnberg hat 505664 Einwohner.
Die Stadt Augsburg hat 264708 Einwohner.
Die Stadt Regensburg hat 135520 Einwohner.
Die Stadt Würzburg hat 133799 Einwohner.
Die Stadt Ingolstadt hat 125088 Einwohner.
Die Stadt Fürth hat 114628 Einwohner.
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 wir bei jedem Aufruf des Programm überschrieben.

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

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "input3.h"

/* Die Konstanten:
 *  int MAX_LAENGE_STR - die maximale String Länge
 *  int MAX_LAENGE_ARR - die maximale Array Länge
 *  sind input3.c auf jeweils 255 und 100 definiert
 */

int main(int argc, char **argv) {
    if (argc < 3) {
        printf("Aufruf: %s <anzahl> <bundesland>\n", argv[0]);
        printf("Beispiel: %s 100 Bayern\n", argv[0]);
        printf("Klein-/Gro ß schreibung beachten!\n");
        exit(1);
    }
    int anzahl = atoi(argv[1]);
    char *bundesland = argv[2];

    // Statisch allokierter Speicher
    char staedte[MAX_LAENGE_ARR][MAX_LAENGE_STR];
    char laender[MAX_LAENGE_ARR][MAX_LAENGE_STR];
    int bewohner[MAX_LAENGE_ARR];

    int len = read_file("staedte.csv", laender, staedte, bewohner);

    // Hier implementieren

    // Mithilfe von write_file(...) soll das Ergebnis in die "resultat.txt"
    // geschrieben werden.

    // Dynamisch allozierter Speicher muss hier freigegeben werden.
}
```

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. Betrachten Sie dazu zunächst den Code in Listing 4, den sie mittels

```
> gcc -std=c99 -Wall ckurs_blatt07_write_file_erklaerung.c input3.c \
      -o ckurs_blatt07_write_file_erklaerung
> ./ckurs_blatt07_write_file_erklaerung
```

kompilieren und ausführen können.

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 `sprintf` 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: Beachten Sie, dass bei der Aufgabe 2 dieses Blattes der Speicher für die einzelnen Zeilen dynamisch – mittels `malloc` – alloziert werden soll.

Listing 4: ckurs_blatt07_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     sprintf(erste_zeile, "Erste Stadt");
18     sprintf(zweite_zeile, "Zweite Stadt");
19     sprintf(dritte_zeile, "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`.