

Vertiefung Software-Entwicklung in C++

Musterlösung zu den Übungsaufgaben – 15. Oktober 2018

Aufgabe 1: ROT13-Verschlüsselung

Schreiben Sie ein C-Programm, das einen Text entgegennimmt, jeden Buchstaben von A bis Z zyklisch um 13 Stellen verschiebt und den auf diese Weise „verschlüsselten“ Text wieder ausgibt. Umlaute, Ziffern, Satz- und Sonderzeichen sollen nicht verändert werden.

A	→	N	a	→	n
B	→	O	b	→	o
...			...		
M	→	Z	m	→	z
N	→	A	n	→	a
...			...		
Y	→	L	y	→	l
Z	→	M	z	→	m

Beispiel: Aus „Apfelkuchen“ wird „Ncsryxhpura“.

Lösung

Datei `loesung-1.c`:

```
#include <stdio.h>

int main (void)
{
    char buffer[100];
    scanf ("%s", buffer);
    for (int i = 0; buffer[i]; i++)
        if (buffer[i] >= 'A' && buffer[i] <= 'M')
            buffer[i] += 13;
        else if (buffer[i] >= 'M' && buffer[i] <= 'Z')
            buffer[i] -= 13;
        else if (buffer[i] >= 'a' && buffer[i] <= 'm')
            buffer[i] += 13;
        else if (buffer[i] >= 'm' && buffer[i] <= 'z')
            buffer[i] -= 13;
    printf ("%s\n", buffer);
    return 0;
}
```

Bemerkungen:

- Ein String in doppelten Anführungszeichen steht für ein Array von **chars**, also ein Array kleiner Zahlen, das durch ein Null-Symbol abgeschlossen ist.

Beispiel: `"A"` steht für das Array `{ 65, 0 }`.

Ein Buchstabe in einfachen Anführungszeichen (Apostrophen) steht für den Buchstaben selbst, also eine kleine Zahl.

Beispiel: `'A'` ist in C dasselbe wie die Zahl `65`.

Unter Verwendung dieser Schreibweise ist es insbesondere nicht nötig, die Zahlenwerte von Buchstaben in einer ASCII-Tabelle nachzuschlagen. Im Gegenteil: Dadurch daß man Buchstaben hinschreibt, ist zum einen sofort klar, was das Programm macht; zum anderen funktioniert es auch mit anderen Zeichensätzen als ASCII, sofern diese die Buchstaben in alphabetischer Reihenfolge enthalten.

- Da auf das **for** nur eine einzige Anweisung (nämlich ein langgezogenes **if**) folgt, sind hier keine geschweiften Klammern nötig. Sie wären aber auch nicht falsch.

- Die Verwendung eines Puffers fester Größe zum Einlesen eines Strings mit `scanf ("%s", buffer)` ist eigentlich ein Unding, da hier keine Prüfung auf einen möglichen Pufferüberlauf stattfindet.

Um diesem Problem zu begegnen, kann man z. B. in der Formatspezifikation für `scanf()` eine maximale Feldgröße spezifizieren. Diese bezieht sich auf die Anzahl der eingelesenen Buchstaben; der Puffer muß zusätzlich Platz für das Null-Symbol am Ende des Strings bereitstellen. In diesem Beispiel mit einem Puffer der Länge 100 wäre demnach `scanf ("%99s", buffer)` ein sinnvoller Aufruf.

Eine andere Möglichkeit ist die Verwendung anderer Funktionen zum Einlesen des Strings, z. B. `fgets (buffer, 100, stdin)`. (Bei `fgets()` steht die Größenangabe für die Größe des Puffers einschließlich Null-Symbol.)

Aufgabe 2: Programm analysieren

Wir betrachten das folgende C-Programm (Datei: [aufgabe-2.c](#)):

```
char*f="char*f=%c%s%c;main(){printf(f,34,f,34,10);}%;c";main(){printf(f,34,f,34,10);}
```

- Was bewirkt dieses Programm?
- Wofür stehen die Zahlen?
- Ergänzen Sie das Programm derart, daß seine `main()`-Funktion `int main (void)` lautet und eine `return`-Anweisung hat, wobei die in Aufgabenteil (a) festgestellte Eigenschaft erhalten bleiben soll.

Lösung

- Was bewirkt dieses Programm?**

Es gibt *seinen eigenen Quelltext* aus.

(Wichtig ist die Bezugnahme auf den eigenen Quelltext. Die Angabe

„Es gibt `char*f="char*f=%c%s%c;main(){printf(f,34,f,34,10);}%;c";main(){printf(f,34,f,34,10);}`“ aus“ genügt insbesondere nicht.)

- Wofür stehen die Zahlen?**

Die 34 steht für ein Anführungszeichen und die 10 für ein Zeilenendezeichen (`\n`).

Hintergrund: Um den eigenen Quelltext ausgeben zu können, muß das Programm auch Anführungszeichen und Zeilenendezeichen ausgeben. Dies geschieht normalerweise mit vorangestelltem Backslash: `\` bzw. `\n`. Um dann aber den Backslash ausgeben zu können, müßte man diesem ebenfalls einen Backslash voranstellen: `\\`. Damit dies nicht zu einer Endlosschleife wird, verwendet der Programmierer dieses Programms den Trick mit den Zahlen, die durch `%c` als Zeichen ausgegeben werden.

- Ergänzen Sie das Programm derart, daß seine `main()`-Funktion `int main (void)` lautet und eine `return`-Anweisung hat, wobei die in Aufgabenteil (a) festgestellte Eigenschaft erhalten bleiben soll.**

Datei: [loesung-2.c](#)

```
char*f="char*f=%c%s%c;int _main(void){printf(f,34,f,34,10);return 0;}%;c";
int main(void){printf(f,34,f,34,10);return 0;}
```

Das Programm ist eine einzige, lange Zeile, die hier nur aus Platzgründen als zwei Zeilen abgedruckt wird. Auf das Semikolon am Ende der „ersten Zeile“ folgt unmittelbar – ohne Leerzeichen – das Schlüsselwort `int` am Anfang der „zweiten Zeile“.

Mit „die in Aufgabenteil (a) festgestellte Eigenschaft“ ist gemeint, daß das Programm weiterhin seinen eigenen Quelltext ausgeben soll. Die Herausforderung dieser Aufgabe besteht darin, das Programm zu modifizieren, ohne diese Eigenschaft zu verlieren.

Zusatzaufgabe für Interessierte: Ergänzen Sie das Programm so, daß es auch mit `-Wall` ohne Warnungen kompiliert werden kann.

Hinweis dazu: `#include<stdio.h>` (ohne Leerzeichen, um Platz zu sparen)

Lösung der Zusatzaufgabe: [loesung-2x.c](#)

Aufgabe 3: Kalender-Berechnung

Am 3. 1. 2009 meldete *heise online*:

Kunden des ersten mobilen Media-Players von Microsoft erlebten zum Jahresende eine böse Überraschung: Am 31. Dezember 2008 fielen weltweit alle Zune-Geräte der ersten Generation aus. Ursache war ein interner Fehler bei der Handhabung von Schaltjahren.

<http://heise.de/-193332>,

Der Artikel verweist auf ein Quelltextfragment (Datei: [aufgabe-3.c](#)), das für einen gegebenen Wert `days` das Jahr und den Tag innerhalb des Jahres für den `days`-ten Tag nach dem 1. 1. 1980 berechnen soll:

```
year = ORIGINYEAR; /* = 1980 */
```

```
while (days > 365)
{
    if (IsLeapYear (year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

Dieses Quelltextfragment weist mehrere Code-Verdopplungen auf:

- Die Anweisung `year += 1` taucht an zwei Stellen auf.
- Es gibt zwei unabhängige Abfragen `days > 365` und `days > 366`: eine in einer `while`- und die andere in einer `if`-Bedingung.
- Die Länge eines Jahres wird nicht durch eine Funktion berechnet oder in einer Variablen gespeichert; stattdessen werden an mehreren Stellen die expliziten numerischen Konstanten 365 und 366 verwendet.

Diese Probleme führten am 31. Dezember 2008 zu einer Endlosschleife, die sich – z. B. durch eine Funktion `DaysInYear()` – leicht hätte vermeiden lassen.

Gut hingegen ist die Verwendung einer Konstanten `ORIGINYEAR` anstelle der Zahl 1980 sowie die Kapselung der Berechnung der Schaltjahr-Bedingung in einer Funktion `IsLeapYear()`.

- Erklären Sie das Zustandekommen der Endlosschleife.
- Schreiben Sie das Quelltextfragment so um, daß es die beschriebenen Probleme nicht mehr enthält.

Lösung

- Erklären Sie das Zustandekommen der Endlosschleife.**

Das Programm startet mit demjenigen Wert für `days`, der der Anzahl der Tage vom 1. 1. 1980 bis zum 31. 12. 2008 entspricht. Die `while`-Schleife läuft zunächst solange korrekt durch, bis `year` den Wert 2008 und `days` den Wert 366 hat. (Der 31. 12. des Schaltjahres 2008 ist der 366. Tag seines Jahres.)

Die Bedingung der `while`-Schleife ist damit weiterhin erfüllt; das Programm läuft weiter.

Da 2008 ein Schaltjahr ist, ist auch die Bedingung der äußeren `if`-Anweisung erfüllt.

Da `days` den Wert 366 hat und dieser nicht größer als 366 ist, ist die innere `if`-Bedingung nicht erfüllt. Somit wird innerhalb der `while`-Schleife kein weiterer Code ausgeführt, die `while`-Bedingung bleibt erfüllt, und das Programm führt eine Endlosschleife aus.

- (b) **Schreiben Sie das Quelltextfragment so um, daß es die beschriebenen Probleme nicht mehr enthält.**

Um das Programm zu testen, genügt es, das Datum auf den 31. 12. 1980 zu stellen, also `days` auf den Wert 366 zu setzen. Darüberhinaus muß man die Funktion `IsLeapYear()` bereitstellen (vgl. Aufgabe 3 vom 10. 10. 2016).

Der Quelltext `loesung-3-f1.c` ist eine lauffähige Version des Programms, die den Fehler (Endlosschleife) reproduziert.

Es liegt nahe, den Fehler in der `while`-Bedingung zu korrigieren, so daß diese Schaltjahre berücksichtigt. Der Quelltext `loesung-3-f2.c` behebt den Fehler auf diese Weise mit Hilfe von Und- (`&&`) und Oder-Verknüpfungen (`||`) in der `while`-Bedingung.

Der Quelltext `loesung-3-f3.c` vermeidet die umständliche Formulierung mit `&&` und `||` durch Verwendung des ternären Operators `?:`. Dieser stellt eine „if-Anweisung für Ausdrücke“ bereit. In diesem Fall liefert er für die rechte Seite des Vergleichs `days > den Wert 366` im Falle eines Schaltjahrs bzw. ansonsten den Wert 365.

Beide Lösungen `loesung-3-f2.c` und `loesung-3-f3.c` sind jedoch im Sinne der Aufgabenstellung **falsch**. Diese lautet: „Schreiben Sie das Quelltextfragment so um, daß es die beschriebenen Probleme nicht mehr enthält.“ Mit den beschriebenen Problemen sind die genannten drei Code-Verdopplungen gemeint, und diese befinden sich weiterhin im Quelltext. Damit ist der Fehler zwar „korrigiert“, aber das Programm ist eher noch unübersichtlicher geworden, so daß nicht klar ist, ob es nicht noch weitere Fehler enthält.

Eine richtige Lösung liefert `loesung-3-4.c`. Dieses Programm speichert den Wert der Tage im Jahr in einer Variablen `DaysInYear`. Damit erübrigen sich die `if`-Anweisungen innerhalb der `while`-Schleife, und die damit verbundenen Code-Verdopplungen verschwinden.

Etwas unschön ist hierbei die neu hinzugekommene Code-Verdopplung bei der Berechnung von `DaysInYear`. Diese ist allerdings weniger kritisch als die vorherigen, da sie nur einmal innerhalb der `while`-Schleife vorkommt und das andere Mal außerhalb derselben.

Um diese Code-Verdopplung loszuwerden, kann man das `if` durch den `?:`-Operator ersetzen und die Zuweisung innerhalb der `while`-Bedingung vornehmen – siehe `loesung-3-5.c`. Dies ist einer der seltenen Fälle, in denen ein Programm *übersichtlicher* wird, wenn eine Zuweisung innerhalb einer Bedingung stattfindet.

Alternativ kann `DaysInYear()` auch eine Funktion sein – siehe `loesung-3-6.c`. Diese Version ist wahrscheinlich die übersichtlichste, hat jedoch den Nachteil, daß die Berechnung von `DaysInYear()` zweimal statt nur einmal pro Schleifendurchlauf erfolgt, wodurch Rechenzeit verschwendet wird.

`loesung-3-7.c` und `loesung-3-8.c` beseitigen dieses Problem durch eine Zuweisung des Funktionsergebnisses an eine Variable – einmal innerhalb der `while`-Bedingung und einmal außerhalb. Der zweimalige Aufruf der Funktion `DaysInYear()` in `loesung-3-8.c` zählt nicht als Code-Verdopplung, denn der Code ist ja in einer Funktion gekapselt. (Genau dazu sind Funktionen ja da: daß man sie mehrfach aufrufen kann.)

Fazit: Wenn Sie sich beim Programmieren bei Cut-And-Paste-Aktionen erwischen, sollten Sie die Struktur Ihres Programms noch einmal überdenken.

Wahrscheinlich gibt es dann eine elegantere Lösung, deren Korrektheit man auf den ersten Blick sieht.