

Hardwarenahe Programmierung

Musterlösung zu den Übungsaufgaben 10 – 9. Januar 2025

Aufgabe 1: Personen-Datenbank

Wir betrachten das folgende Programm ([aufgabe-1.c](#)):

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

typedef struct
{
    char first_name[10];
    char family_name[20];
    char day, month;
    int year;
} person;

int main (void)
{
    person sls;
    sls.day = 26;
    sls.month = 7;
    sls.year = 1951;
    strcpy (sls.first_name, "Sabine");
    strcpy (sls.family_name, "Leutheusser-Schnarrenberger");
    printf ("%s_%s_wurde_am_%d.%d.%d_geboren.\n",
           sls.first_name, sls.family_name, sls.day, sls.month, sls.year);
    return 0;
}
```

Die Standard-Funktion `strcpy()` bewirkt ein Kopieren eines Strings von rechts nach links, hier also z. B. die Zuweisung der String-Konstanten "Sabine" an die String-Variable `sls.first_name[]`.

Das Programm wird für einen 32-Bit-Rechner compiliert und ausgeführt.

(Die `gcc`-Option `-m32` sorgt dafür, daß `gcc` Code für einen 32-Bit-Prozessor erzeugt.)

```
$ gcc -Wall -O -m32 aufgabe-2.c -o aufgabe-2
$ ./aufgabe-2
Sabine Leutheusser-Schnarrenberger wurde am 110.98.1701278309 geboren.
Speicherzugriffsfehler
```

- (a) Erklären Sie die Ausgabe des Programms einschließlich der Zahlenwerte. (4 Punkte)
- (b) Welche Endianness hat der verwendete Rechner? Begründen Sie Ihre Antwort. (1 Punkt)
- (c) Wie sähe die Ausgabe auf einem Rechner mit entgegengesetzter Endianness aus? (2 Punkte)
- (d) Erklären Sie den Speicherzugriffsfehler. (Es kann sein, daß sich der Fehler auf Ihrem Rechner nicht bemerkbar macht. Er ist aber trotzdem vorhanden.) (2 Punkte)

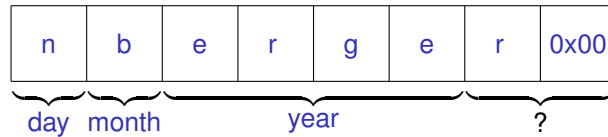
Lösung

- (a) **Erklären Sie die Ausgabe des Programms einschließlich der Zahlenwerte.**

Der String "Leutheusser-Schnarrenberger" hat 27 Zeichen und daher mehr als die in der Variablen `sls.family_name` vorgesehenen 20 Zeichen. Das "nberger" paßt nicht mehr in die String-Variablen.

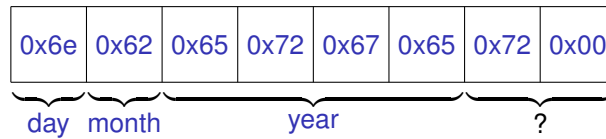
Die Zuweisung `strcpy (sls.family_name, "Leutheusser-Schnarrenberger")` überschreibt daher 8 Speicherzellen außerhalb der String-Variablen `sls.family_name` mit dem String "nberger" (7 Buchstaben zzgl. String-Ende-Symbol) – und damit insbesondere die Variablen `sls.day`, `sls.month` und `sls.year`.

Die überschriebenen Speicherzellen sehen demnach folgendermaßen aus:



(„?“ steht für zwei Speicherzellen, von denen wir nicht wissen, wofür sie genutzt werden.)

Wenn wir die ASCII-Zeichen in Hexadezimalzahlen umrechnen, entspricht dies:



Dies entspricht bereits genau den Werten **110** und **98** für die Variablen `sls.day` bzw. `sls.month`.

Für die Variable `sls.year` müssen wir ihre vier Speicherzellen unter der Berücksichtigung der Endianness des Rechners zusammenziehen. Für Big-Endian ergibt dies **0x65726765 == 1701996389**. Für Little-Endian ergibt sich der Wert **0x65677265 == 1701278309**, der auch in der Ausgabe des Programms auftaucht.

(b) Welche Endianness hat der verwendete Rechner? Begründen Sie Ihre Antwort.

Wie in (a) begründet, ergibt sich die Ausgabe von **1701278309** für das Jahr aus dem Speicherformat Little-Endian.

(c) Wie sähe die Ausgabe auf einem Rechner mit entgegengesetzter Endianness aus?

Wie in (a) begründet, ergäbe sich aus dem Speicherformat Big-Endian die Ausgabe von **1701996389** für das Jahr.

(d) Erklären Sie den Speicherzugriffsfehler. (Es kann sein, daß sich der Fehler auf Ihrem Rechner nicht bemerkbar macht. Er ist aber trotzdem vorhanden.)

Die zwei in (a) mit „?“ bezeichneten Speicherzellen wurden ebenfalls überschrieben. Dies ist in der Ausgabe zunächst nicht sichtbar, bewirkt aber später den Speicherzugriffsfehler.

(Tatsächlich handelt es sich bei den überschriebenen Speicherzellen um einen Teil der Rücksprungadresse, die `main()` verwendet, um mit `return 0` an das Betriebssystem zurückzugeben.)

Hinweis 1: Um auf einen solchen Lösungsweg zu kommen, wird empfohlen, „geheimnisvolle“ Zahlen nach hexadezimal umzurechnen und in Speicherzellen (Zweiergruppen von Hex-Ziffern) zu zerlegen. Oft erkennt man dann direkt ASCII-Zeichen: Großbuchstaben beginnen mit der Hex-Ziffer **4** oder **5**, Kleinbuchstaben mit **6** oder **7**.

Hinweis 2: Um derartige Programmierfehler in der Praxis von vornherein zu vermeiden, wird empfohlen, anstelle von `strcpy()` grundsätzlich die Funktion `strncpy()` zu verwenden. Diese erwartet einen zusätzlichen Parameter, der die maximal zulässige Länge des Strings enthält. Ohne einen derartigen expliziten Parameter kann die Funktion nicht wissen, wie lang die Variable ist, in der der String gespeichert werden soll.

Aufgabe 2: Einfügen in Strings (Ergänzung)

Diese Aufgabe ist eine Ergänzung von Aufgabe 3 der Übung 3 vom 7. November 2024 um die Teilaufgaben (e), (f) und (g). Für den „Klausur-Modus“ können Sie die Teilaufgaben (a) bis (d) als „bereits gelöst“ voraussetzen.

Wir betrachten das folgende Programm (`aufgabe-2.c`):

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

void insert_into_string (char src, char *target, int pos)
{
    int len = strlen (target);
    for (int i = pos; i < len; i++)
        target[i+1] = target[i];
    target[pos] = src;
}
```

```

int main (void)
{
    char test[100] = "Hochshule_Bochum";
    insert_into_string ('c', test, 5);
    printf ("%s\n", test);
    return 0;
}

```

Die Ausgabe des Programms lautet: `Hochschhhhhhhhhhh`

- (a) Erklären Sie, wie die Ausgabe zustandekommt.
- (b) Schreiben Sie die Funktion `insert_into_string()` so um, daß sie den Buchstaben `src` an der Stelle `pos` in den String `target` einfügt.

Die Ausgabe des Programms müßte dann `Hochschule Bochum` lauten.

- (c) Was kann passieren, wenn Sie die Zeile `char test[100] = "Hochshule_Bochum";` durch `char test[] = "Hochshule_Bochum";` ersetzen? Begründen Sie Ihre Antwort.
- (d) Was kann passieren, wenn Sie die Zeile `char test[100] = "Hochshule_Bochum";` durch `char *test = "Hochshule_Bochum";` ersetzen? Begründen Sie Ihre Antwort.
- (e) Schreiben Sie eine Funktion `void insert_into_string_sorted (char src, char *target)`, die voraussetzt, daß der String `target` alphabetisch sortiert ist und den Buchstaben `src` an der alphabetisch richtigen Stelle einfügt. Diese Funktion darf die bereits vorhandene Funktion `insert_into_string()` aufrufen. (4 Punkte)

Zum Testen eignen sich die folgenden Zeilen im Hauptprogramm:

```

char test[100] = "";
insert_into_string_sorted ('c', test);
insert_into_string_sorted ('a', test);
insert_into_string_sorted ('d', test);
insert_into_string_sorted ('b', test);

```

Danach sollte `test[]` die Zeichenfolge `"abcd"` enthalten.

- (f) Wie schnell (Landau-Symbol in Abhängigkeit von der Länge n des Strings) arbeitet Ihre Funktion `void insert_into_string_sorted (char src, char *target)`? Begründen Sie Ihre Antwort. (1 Punkt)
- (g) Beschreiben Sie – in Worten oder als C-Quelltext –, wie man die Funktion `void insert_into_string_sorted (char src, char *target)` so gestalten kann, daß sie in $\mathcal{O}(\log n)$ arbeitet. (3 Punkte)

Lösung

Bemerkung: Die in dieser Aufgabe und ihrer Musterlösung vorkommenden Funktionen prüfen nicht, ob durch das Einfügen eines Zeichens der für den String reservierte Speicherplatz überläuft. Ein derartiges Verhalten wäre in einem „echten“ Programm ein **Fehler**, der katastrophale Folgen haben kann. Wenn dergleichen hier nicht berücksichtigt wird, dann nur, um in einer Klausur nicht den zeitlichen Rahmen zu sprengen.

- (e) **Schreiben Sie eine Funktion `void insert_into_string_sorted (char src, char *target)`, die voraussetzt, daß der String `target` alphabetisch sortiert ist und den Buchstaben `src` an der alphabetisch richtigen Stelle einfügt. Diese Funktion darf die bereits vorhandene Funktion `insert_into_string()` aufrufen.**

```

void insert_into_string_sorted (char src, char *target)
{
    int i = 0;
    while (target[i] && target[i] < src)
        i++;
    insert_into_string (src, target, i);
}

```

Die Datei [loesung-2e.c](#) enthält die o. a. Funktion sowie zusätzliche Tests.

- (f) **Wie schnell (Landau-Symbol in Abhängigkeit von der Länge n des Strings) arbeitet Ihre Funktion `void insert_into_string_sorted(char src, char *target)`? Begründen Sie Ihre Antwort.**

Die Funktion sucht im Array **mittels einer Schleife** nach der korrekten Position zum Einfügen des Zeichens und hat daher von sich aus $\mathcal{O}(n)$.

Anschließend ruft sie die Funktion `insert_into_string()` auf, die ebenfalls eine Schleife verwendet, um im Array Platz zu Einfügen zu schaffen, und daher ebenfalls $\mathcal{O}(n)$ hat.

Es bleibt daher bei $\mathcal{O}(n)$.

- (g) **Beschreiben Sie – in Worten oder als C-Quelltext –, wie man die Funktion `void insert_into_string_sorted(char src, char *target)` so gestalten kann, daß sie in $\mathcal{O}(\log n)$ arbeitet.**

In einem alphabetisch sortierten Array kann man die Suche in der Mitte beginnen und sich durch Halbieren der Intervalle an die gesuchte Position herantasten. Wegen des fortwährenden Halbierens geschieht dies in $\mathcal{O}(\log n)$. (Für eine derartige Antwort gäbe es in der Klausur die volle Punktzahl.)

Wenn wir allerdings anschließend für das eigentliche Einfügen die Funktion `insert_into_string()` verwenden, die dafür $\mathcal{O}(n)$ benötigt, kommen wir insgesamt auf $\mathcal{O}(n)$. Ein sortiertes Einfügen in ein Array ist daher in $\mathcal{O}(\log n)$ nicht möglich. (Wer dies bemerkt, kann zum einen während der Klausur nachfragen, wie denn die Aufgabenstellung genau gemeint ist, und sich zum anderen für die besondere Sorgfalt Zusatzpunkte verdienen.)

Die Datei [loesung-2g.c](#) enthält einen C-Quelltext, die den o. a. Algorithmus als Funktion implementiert. Man beachte die Behandlung des Spezialfalls, daß das einzufügende Zeichen am Ende angehängt werden muß.

Aufgabe 3: Objektorientierte Tier-Datenbank

Das unten dargestellte Programm (Datei: [aufgabe-3a.c](#)) soll Daten von Tieren verwalten.

Beim Compilieren erscheinen die folgende Fehlermeldungen:

```
$ gcc -std=c99 -Wall -O aufgabe-2a.c -o aufgabe-2a
aufgabe-2a.c: In function 'main':
aufgabe-2a.c:31: error: 'animal' has no member named 'wings'
aufgabe-2a.c:37: error: 'animal' has no member named 'legs'
```

Der Programmierer nimmt die in Rot dargestellten Ersetzungen vor (Datei: [aufgabe-3b.c](#)). Daraufhin gelingt das Compilieren, und die Ausgabe des Programms lautet:

```
$ gcc -std=c99 -Wall -O aufgabe-2b.c -o aufgabe-2b
$ ./aufgabe-2b
A duck has 2 legs.
Error in animal: cow
```

- (a) Erklären Sie die o. a. Compiler-Fehlermeldungen. (2 Punkte)
- (b) Wieso verschwinden die Fehlermeldungen nach den o. a. Ersetzungen? (3 Punkte)
- (c) Erklären Sie die Ausgabe des Programms. (5 Punkte)
- (d) Beschreiben Sie – in Worten und/oder als C-Quelltext – einen Weg, das Programm so zu berichtigen, daß es die Eingabedaten ("A duck has 2 wings. A cow has 4 legs.") korrekt speichert und ausgibt. (4 Punkte)
- (e) Schreiben Sie das Programm so um, daß es keine expliziten Typumwandlungen mehr benötigt. Hinweis: Verwenden Sie **union**. (4 Punkte)
- (f) Schreiben Sie das Programm weiter um, so daß es die Objektinstanzen `duck` und `cow` dynamisch erzeugt. Hinweis: Verwenden Sie `malloc()` und schreiben Sie Konstruktoren. (4 Punkte)

- (g) Schreiben Sie das Programm weiter um, so daß die Ausgabe nicht mehr direkt im Hauptprogramm erfolgt, sondern stattdessen eine virtuelle Methode `print()` aufgerufen wird.

Hinweis: Verwenden Sie in den Objekten Zeiger auf Funktionen, und initialisieren Sie diese in den Konstruktoren. (4 Punkte)

```
#include <stdio.h>

#define ANIMAL 0
#define WITH_WINGS 1
#define WITH_LEGS 2

typedef struct animal
{
    int type;
    char *name;
} animal;

typedef struct with_wings
{
    int wings;
} with_wings;

typedef struct with_legs
{
    int legs;
} with_legs;

int main (void)
{
    animal *a[2];

    animal duck;
    a[0] = &duck;
    a[0]→type = WITH_WINGS;
    a[0]→name = "duck";
    a[0]→wings = 2; ← ((with_wings *) a[0])→wings = 2;

    animal cow;
    a[1] = &cow;
    a[1]→type = WITH_LEGS;
    a[1]→name = "cow";
    a[1]→legs = 4; ← ((with_legs *) a[1])→legs = 4;

    for (int i = 0; i < 2; i++)
        if (a[i]→type == WITH_LEGS)
            printf ("A_%s_has_%d_legs.\n", a[i]→name,
                    ((with_legs *) a[i])→legs);
        else if (a[i]→type == WITH_WINGS)
            printf ("A_%s_has_%d_wings.\n", a[i]→name,
                    ((with_wings *) a[i])→wings);
        else
            printf ("Error_in_animal:_%s\n", a[i]→name);

    return 0;
}
```

Lösung

- (a) Erklären Sie die o. a. Compiler-Fehlermeldungen.

`a[0]` und `a[1]` sind gemäß der Deklaration `animal *a[2]` Zeiger auf Variablen vom Typ `animal` (ein `struct`). Wenn man diesen Zeiger dereferenziert (`→`), erhält man eine `animal`-Variable. Diese enthält keine Datenfelder `wings` bzw. `legs`.

- (b) Wieso verschwinden die Fehlermeldungen nach den o. a. Ersetzungen?

Durch die *explizite Typumwandlung des Zeigers* erhalten wir einen Zeiger auf eine `with_wings`- bzw. auf eine `with_legs`-Variable. Diese enthalten die Datenfelder `wings` bzw. `legs`.

- (c) Erklären Sie die Ausgabe des Programms.

Durch die explizite Typumwandlung des Zeigers zeigt `a[0]` auf eine `with_wings`-Variable. Diese enthält nur ein einziges Datenfeld `wings`, das an genau derselben Stelle im Speicher liegt wie `a[0]→type`, also das Datenfeld `type` der `animal`-Variable, auf die der Zeiger `a[0]` zeigt. Durch die Zuweisung der Zahl 2 an `((with_wings *) a[0])→wings` überschreiben wir also `a[0]→type`, so daß das `if` in der `for`-Schleife `a[0]` als `WITH_LEGS` erkennt.

Bei der Ausgabe `A duck has 2 legs.` wird das Datenfeld `((with_legs *)a[0])→legs` als Zahl ausgegeben. Dieses Datenfeld befindet sich in denselben Speicherzellen wie `a[0]→type` und `((with_wings *) a[0])→wings` und hat daher ebenfalls den Wert 2.

Auf die gleiche Weise überschreiben wir durch die Zuweisung der Zahl 4 an `((with_legs *) a[1])→legs` das Datenfeld `a[0]→type`, so daß das `if` in der `for`-Schleife `a[1]` als unbekanntes Tier (Nr. 4) erkennt und `Error in animal: cow` ausgibt.

- (d) **Beschreiben Sie – in Worten und/oder als C-Quelltext – einen Weg, das Programm so zu berichtigen, daß es die Eingabedaten ("A duck has 2 wings. A cow has 4 legs.") korrekt speichert und ausgibt.**

Damit die *Vererbung* zwischen den Objekten `animal`, `with_wings` und `with_legs` funktioniert, müssen die abgeleiteten Klassen `with_wings` und `with_legs` alle Datenfelder der Basisklasse `animal` erben. In C geschieht dies explizit; die Datenfelder müssen in den abgeleiteten Klassen neu angegeben werden (siehe [loesung-3d-1.c](#)):

```
typedef struct animal
{
    int type;
    char *name;
} animal;

typedef struct with_wings
{
    int type;
    char *name;
    int wings;
} with_wings;

typedef struct with_legs
{
    int type;
    char *name;
    int legs;
} with_legs;
```

Zusätzlich ist es notwendig, die Instanzen `duck` und `cow` der abgeleiteten Klassen `with_wings` und `with_legs` auch als solche zu deklarieren, damit für sie genügend Speicher reserviert wird:

```
animal *a[2];

with_wings duck;
a[0] = (animal *) &duck;
a[0]->type = WITH_WINGS;
a[0]->name = "duck";
((with_wings *) a[0])->wings = 2;

with_legs cow;
a[1] = (animal *) &cow;
a[1]->type = WITH_LEGS;
a[1]->name = "cow";
((with_legs *) a[1])->legs = 4;
```

Wenn man dies vergißt und sie nur als `animal` deklariert, wird auch nur Speicherplatz für (kleinere) `animal`-Variable angelegt. Dadurch kommt es zu Speicherzugriffen außerhalb der deklarierten Variablen, was letztlich zu einem Absturz führt (siehe [loesung-3d-0f.c](#)).

Für die Zuweisung eines Zeigers auf `duck` an `a[0]`, also an einen Zeiger auf `animal` wird eine weitere explizite Typumwandlung notwendig. Entsprechendes gilt für die Zuweisung eines Zeigers auf `cow` an `a[1]`.

Es ist sinnvoll, explizite Typumwandlungen so weit wie möglich zu vermeiden. Es ist einfacher und gleichzeitig sicherer, direkt in die Variablen `duck` und `cow` zu schreiben, anstatt dies über die Zeiger `a[0]` und `a[1]` zu tun (siehe [loesung-3d-2.c](#)):

```
animal *a[2];

with_wings duck;
a[0] = (animal *) &duck;
duck.type = WITH_WINGS;
duck.name = "duck";
duck.wings = 2;
```

```
with_legs cow;  
a[1] = (animal *) &cow;  
cow.type = WITH_LEGS;  
cow.name = "cow";  
cow.legs = 4;
```

- (e) **Schreiben Sie das Programm so um, daß es keine expliziten Typumwandlungen mehr benötigt.**

Hinweis: Verwenden Sie **union**.

Siehe [loesung-3e.c](#).

Diese Lösung basiert auf [loesung-3d-2.c](#), da diese bereits weniger explizite Typumwandlungen enthält als [loesung-3d-1.c](#).

Arbeitsschritte:

- Umbenennen des Basistyps **animal** in **base**, damit wir den Bezeichner **animal** für die **union** verwenden können
- Schreiben einer **union animal**, die die drei Klassen **base**, **with_wings** und **with_legs** als Datenfelder enthält
- Umschreiben der Initialisierungen: Zugriff auf Datenfelder erfolgt nun durch z. B. **a[0]→b.name**. Hierbei ist **b** der Name des **base**-Datenfelds innerhalb der **union animal**.
- Auf gleiche Weise schreiben wir die **if**-Bedingungen innerhalb der **for**-Schleife sowie die Parameter der **printf()**-Aufrufe um.

Explizite Typumwandlungen sind nun nicht mehr nötig.

Nachteil dieser Lösung: Jede Objekt-Variable belegt nun Speicherplatz für die gesamte **union animal**, anstatt nur für die benötigte Variable vom Typ **with_wings** oder **with_legs**. Dies kann zu einer Verschwendung von Speicherplatz führen, auch wenn dies in diesem Beispielprogramm tatsächlich nicht der Fall ist.

- (f) **Schreiben Sie das Programm weiter um, so daß es die Objektinstanzen **duck** und **cow** dynamisch erzeugt.**

Hinweis: Verwenden Sie **malloc()** und schreiben Sie Konstruktoren.

Siehe [loesung-3f.c](#).

- (g) **Schreiben Sie das Programm weiter um, so daß die Ausgabe nicht mehr direkt im Hauptprogramm erfolgt, sondern stattdessen eine virtuelle Methode **print()** aufgerufen wird.**

Hinweis: Verwenden Sie in den Objekten Zeiger auf Funktionen, und initialisieren Sie diese in den Konstruktoren.

Siehe [loesung-3g.c](#).