

# Hardwarenahe Programmierung

## Übungsaufgaben – 28. November 2022

Diese Übung enthält Punkteangaben wie in einer Klausur. Um zu „bestehen“, müssen Sie innerhalb von 65 Minuten unter Verwendung ausschließlich zugelassener Hilfsmittel 11 Punkte (von insgesamt 22) erreichen.

### Aufgabe 1: Einfügen in Strings (Ergänzung)

Diese Aufgabe ist eine Ergänzung von Aufgabe 3 der Übung vom 31. Oktober 2022 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-1.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)

## Aufgabe 2: Dynamisches Bit-Array

Schreiben Sie die folgenden Funktionen zur Verwaltung eines dynamischen Bit-Arrays:

- `void bit_array_init(int n)`  
Das Array initialisieren, so daß man  $n$  Bits darin speichern kann.  
Die Array-Größe  $n$  ist keine Konstante, sondern erst im laufenden Programm bekannt.  
Die Bits sollen auf den Anfangswert 0 initialisiert werden.
- `void bit_array_set(int i, int value)`  
Das Bit mit dem Index  $i$  auf den Wert  $value$  setzen.  
Der Index  $i$  darf von 0 bis  $n - 1$  gehen; der Wert  $value$  darf 1 oder 0 sein.
- `void bit_array_flip(int i)`  
Das Bit mit dem Index  $i$  auf den entgegengesetzten Wert setzen, also auf 1, wenn er vorher 0 ist, bzw. auf 0, wenn er vorher 1 ist.  
Der Index  $i$  darf von 0 bis  $n - 1$  gehen.
- `int bit_array_get(int i)`  
Den Wert des Bit mit dem Index  $i$  zurückliefern.  
Der Index  $i$  darf von 0 bis  $n - 1$  gehen.
- `void bit_array_resize(int new_n)`  
Die Größe des Arrays auf  $new\_n$  Bits ändern.  
Dabei soll der Inhalt des Arrays, soweit er in die neue Größe paßt, erhalten bleiben.  
Neu hinzukommende Bits sollen auf 0 initialisiert werden.
- `void bit_array_done(void)`  
Den vom Array belegten Speicherplatz wieder freigeben.

Bei Bedarf dürfen Sie den Funktionen zusätzliche Parameter mitgeben, beispielsweise um mehrere Arrays parallel verwalten zu können. (In der objektorientierten Programmierung wäre dies der implizite Parameter `this`, der auf die Objekt-Struktur zeigt.)

Die Bits sollen möglichst effizient gespeichert werden, z. B. jeweils 8 Bits in einer `uint8_t`-Variablen.

Die Funktionen sollen möglichst robust sein, d. h. das Programm darf auch bei unsinnigen Parameterwerten nicht abstürzen, sondern soll eine Fehlermeldung ausgeben.

Die folgenden **Hinweise** beschreiben einen möglichen Weg, die Aufgabe zu lösen. Es steht Ihnen frei, die Aufgabe auch auf andere Weise zu lösen.

- Setzen Sie zunächst voraus, daß das Array die konstante Länge 8 hat, und schreiben Sie zunächst nur die Funktionen `bit_array_set()`, `bit_array_flip()` und `bit_array_get()`.
- Verallgemeinern Sie nun auf eine konstante Länge, bei der es sich um ein Vielfaches von 8 handelt.
- Implementieren Sie nun die Überprüfung auf unsinnige Parameterwerte. Damit können Sie sich gleichzeitig von der Bedingung lösen, daß die Länge des Arrays ein Vielfaches von 8 sein muß.
- Gehen Sie nun von einem statischen zu einem dynamischen Array über, und implementieren Sie die Funktionen `bit_array_init()`, `bit_array_done()` und `bit_array_resize()`.

### 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)

```
#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;
}
```

*Viel Erfolg!*