

Lösungshinweise/-vorschläge zum Übungsblatt 12: Software-Entwicklung 1 (WS 2017/18)

Die Hinweise und Vorschläge in diesem Dokument sollen der Lösungsfindung dienen und erheben demnach weder Anspruch auf Vollständigkeit noch Korrektheit. Sollten Sie Fehler finden, würden wir uns freuen, wenn Sie uns diese mitteilen. (Kontaktinformationen finden Sie auf unserer Webpräsenz.)

Aufgabe 1 Mittelwert (5 Punkte)

Schreiben Sie ein C-Programm `mittelwert.c`, welches `int`-Zahlen von der Standardeingabe liest bis diese leer ist und dann den Mittelwert der eingelesenen Zahlen ausgibt.

Verwenden Sie zum Einlesen der Zahlen die Funktion `scanf` und zum Ausgeben die Funktion `printf`. Der Mittelwert soll mit 2 Nachkommastellen ausgegeben werden, also mit `printf("%.2f\n", wert);`.

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

int main(void) {
    int count = 0;
    int sum = 0;
    while (true)
    {
        int x;
        int r = scanf("%d", &x);
        if (r == EOF)
        {
            break;
        }
        count++;
        sum += x;
    }
    printf("%.2f\n", sum*1.0/count);
    return 0;
}
```

Aufgabe 2 Fibonacci-Zahlen (6 Punkte)

- a) Schreiben Sie eine Funktion `void fibArray(int *res, int n)`, welche einen Pointer `res` auf einen Speicherbereich und eine Zahl `n` nimmt. Die Funktion soll die Reihe der ersten `n` Fibonacci-Zahlen in `res` speichern. Nach Aufruf der Funktion soll gelten, dass `res[0] = 0`, `res[1] = 1` und `res[i] = res[i-1] + res[i-2]` für $1 < i < n$.

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

// requires size >= 2 && res hat Platz für size integer
void fibArray(int *res, int size)
```

```

{
    res[0] = 0;
    res[1] = 1;
    for (int i=2; i<size; i++)
    {
        res[i] = res[i-1] + res[i-2];
    }
}

```

- b) Schreiben Sie eine `main`-Funktion, welche einen `int`-Wert `n` von der Standardeingabe liest und dann mit Hilfe der Funktion `fibArray` die ersten `n` Fibonacci-Zahlen berechnet und dann ausgibt. Bei der Ausgabe soll genau eine Zahl pro Zeile ausgegeben werden.

Verwenden Sie die Funktion `malloc` um den Speicherbereich für die Funktion `fibArray` anzufordern und die Funktion `free` um den Speicher wieder freizugeben.

```

int main(void)
{
    int n;
    scanf("%d", &n);
    int *fibs = malloc(n*sizeof(int));
    fibArray(fibs, n);
    for (int i=0; i<n; i++)
    {
        printf("%d\n", fibs[i]);
    }
    free(fibs);
}

```

Aufgabe 3 Eingabe umdrehen (4 Punkte)

Schreiben Sie ein C-Programm, welches `int`-Werte von der Standardeingabe liest, bis diese zu Ende ist und dann die eingelesenen Zahlen in umgekehrter Reihenfolge ausgibt.

Verwenden Sie `malloc` um zu Beginn des Programms Speicherplatz für 8 `int`-Werte anzufordern. Speichern Sie dort die eingelesenen Zahlen und fordern Sie bei Bedarf zusätzlichen Speicher an. Geben Sie den Speicher am Ende mit `free` wieder frei.

```

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int main(void)
{
    int count = 0;
    int *buffer = malloc(8*sizeof(int));
    int size = 8;
    while (true)
    {
        int x;
        int r = scanf("%d", &x);
        if (r == EOF)
        {
            break;
        }

        if (count+1 >= size) {
            size *= 2;
            buffer = realloc(buffer, size*sizeof(int));
        }
    }
}

```

```

    }
    buffer[count] = x;
    count++;
}
for (int i=count-1; i>=0; i--)
{
    printf("%d\n", buffer[i]);
}
free(buffer);
return 0;
}

```

Aufgabe 4 Sortieren (5 Punkte)

Verwenden Sie für diese Aufgabe die Vorlage `sort.c` und passen Sie die `main`-Funktion nicht an.

- a) Schreiben Sie eine Funktion `bool sort2(int *x, int *y)`, welche 2 Pointer auf `int`-Werte nimmt. Wenn der von `x` referenzierte Wert größer ist, als der von `y` referenzierte Wert, dann sollen die beiden Werte vertauscht und `true` zurückgegeben werden. Ansonsten ist das Ergebnis der Funktion `false`. Verwenden Sie zur Implementierung die Funktion `swap2` aus der Vorlesung.

```

bool sort2(int *x, int *y)
{
    if (*x > *y) {
        swap2(x, y);
        return true;
    }
    return false;
}

```

- b) Schreiben Sie eine Funktion `void sort(int ar[], int size)`, welche ein Array von `int`-Werten `ar` und die Anzahl der Werte im Array nimmt. Die Funktion soll das Array aufsteigend sortieren. Verwenden Sie zum Sortieren die Funktion `sort2` um jeweils zwei Array-Einträge zu sortieren und befolgen Sie der folgenden Algorithmen-Beschreibung:

```

wiederhole so lange es Änderungen am Array gibt:
    für i aus [0,..,size-2]:
        sortiere ar[i] und ar[i+1]
    Verringere size um 1 (das letzte Element ist schon an der richtigen Stelle)

```

```

void sort(int ar[], int size)
{
    bool changes = true;
    // Idee: Wenn keine Änderungen mehr möglich muss Array sortiert sein
    while (changes)
    {
        changes = false;
        // probiere jeweils einen Eintrag und seinen Nachfolger zu sortieren
        for (int i=0; i<size-1; i++)
        {
            changes = sort2(&ar[i], &ar[i+1]) || changes;
        }
    }
}

```

Aufgabe 5 Programmverständnis (5 Punkte)

Versuchen Sie diese Aufgabe ohne Verwendung eines Computers zu lösen. In der Klausur müssen Sie eventuell ähnliche Aufgaben auch ohne Unterstützung lösen können.

- a) Was passiert beim Übersetzen und Ausführen des folgenden Programms? Falls es dabei Fehler gibt, erklären Sie diese. Falls nicht, geben Sie die Ausgabe des Programms an.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int ar[] = {200, 30, 4, 1000};
6     int *p = &ar[0];
7     int *q = &ar[3];
8     int r = 0;
9     while (p < q)
10    {
11        r = r + *p + *q;
12        p = p + 1;
13        q = q - 1;
14    }
15    printf("r = %d\n", r);
16 }
```

Die 2 pointer p und q laufen aufeinander zu. Es werden alle Zahlen im Array ar aufsummiert. Die Ausgabe ist daher 1234.

- b) Das folgende Programm enthält einen Speicherleck. Wie kann dieser behoben werden?

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <stdbool.h>
4
5 int *alloc_space(int value) {
6     int *space = malloc(10*sizeof(int));
7     for (int i=0; i<10; i++)
8     {
9         *(space + i) = value;
10    }
11    return space;
12 }
13
14 void dealloc_space(int *space) {
15     for (int i=0; i<10; i++) {
16         *(space + i) = 0;
17     }
18     space = NULL;
19 }
20
21 int main(void) {
22     while(true) {
23         int v;
24         scanf("%d", &v);
25         if (v<=0)
26         {
27             break;
28         }
29
30         int *mySpace = alloc_space(v);
31         for (int i=0; i<10; i++)
32         {
33             mySpace[i] *= i;
34         }
35         for (int i=0; i<10; i++)
36         {
37             printf("%d\n", *(mySpace + i));
38         }
39         dealloc_space(mySpace);
```

```

40 }
41 return 0;
42 }

```

Die Funktion setzt die Werte auf 0, welche sich im von space referenzierten Speicherbereich befinden und sie setzt den Parameter space auf NULL. Beides sorgt aber nicht dafür, dass der Speicherbereich freigegeben wird, der in Zeile 6 alloziert wurde. Statt dessen sollte die Funktion einen Aufruf `free(space)` enthalten.

- c) Was ist die Ausgabe des folgenden Programms? Begründen Sie Ihre Antwort, indem Sie den Zustand der Variablen `number`, `pointer` und `p` jeweils nach Ausführen von Zeile 8, 9, 10 und 11 beschreiben oder grafisch darstellen.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(void)
5 {
6     int number[] = {1, 2, 3, 4};
7     int *pointer[] = {&number[0], &number[1], &number[2], &number[3]};
8     int **p = pointer;
9     p = p + 1;
10    *p = *p + 1;
11    **p = **p + 1;
12
13    for (int i = 0; i < 4; i++)
14    {
15        printf("number[%d] = %d\n", i, number[i]);
16    }
17 }

```

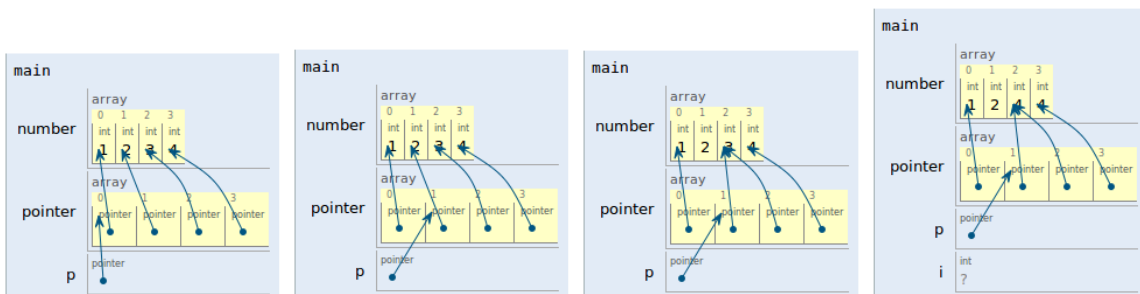
Die Ausgabe ist:

```

number[0] = 1
number[1] = 2
number[2] = 4
number[3] = 4

```

Die Zustände der Variablen bei der Ausführung lassen sich mit einem Debugger oder dem Online-Tool Python Tutor⁴ überprüfen. Die folgenden Screenshots zeigen die geforderten Zustände im Python Tutor



⁴<http://pythontutor.com/visualize.html>

Aufgabe 6 Verschachtelte Arrays (5 Punkte)

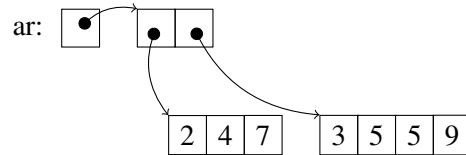
Verwenden Sie für diese Aufgabe die Vorlage `flatten.c` und passen Sie die `main`-Funktion nicht an.

Schreiben Sie eine Funktion `int flatten(int **ar, int n, int **res)`, welche einen Zeiger `ar` und eine Größe `n` nimmt. Dieser zeigt auf einen Speicherbereich mit `n` Zeigern auf die inneren `int`-Arrays. Diese inneren Arrays sind ähnlich wie Arrays in Java strukturiert: Sie beinhalten eine Größe, welche wir hier im ersten Array-Eintrag speichern. Nach der Größe folgen die eigentlichen Array-Einträge.

Die Funktion `flatten` soll alle Zahlen, ohne die Längen-Information, der Reihe nach in einen zusammenhängenden Speicherbereich übertragen. Die Anzahl der übertragenen Zahlen soll als Rückgabewert der Funktion zurückgegeben werden und der Parameter `res` soll nach dem Aufruf auf den Speicherbereich mit den Zahlen zeigen. Sie können davon ausgehen, dass der Parameter `res` zu Beginn `NULL` ist oder auf einen gültigen Speicherbereich zeigt (das heißt, dass `res` als Parameter von `free` oder `realloc` verwendet werden darf).

Beispiel:

```
int main(void)
{
    int a1[] = {2, 4, 7};
    int b2[] = {3, 5, 5, 9};
    int *ar[] = {a1, b2};
    int *res = NULL;
    int size = flatten(ar, 2, &res);
    printf("size = %d\n", size);
    for (int i=0; i<size; i++) {
        printf("res[%d] = %d\n", i, res[i]);
    }
    free(res);
}
```



Erwartete Ausgabe:

```
size = 5
res[0] = 4
res[1] = 7
res[2] = 5
res[3] = 5
res[4] = 9
```

```
int flatten(int **ar, int n, int** res) {
    int size = 0;
    for (int i=0; i<n; i++) {
        size += *ar[i];
    }
    int *r = realloc(*res, size*sizeof(int));
    if (r == NULL) {
        printf("Could not allocate memory\n");
        exit(1);
    }
    int k = 0;
    for (int i=0; i<n; i++) {
        for (int j=1; j<=*ar[i]; j++) {
            r[k] = ar[i][j];
            k++;
        }
    }
    *res = r;
    return size;
}
```