Dr. Annette Bieniusa
Mathias Weber, M. Sc.
Peter Zeller, M. Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Lösungshinweise/-vorschläge zum Übungsblatt 11: Software-Entwicklung 1 (WS 2016/17)

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. (Kontaktdaten finden Sie auf unserer Webpräsenz.)

Aufgabe 1 Mittelwert (5 Punkte)

Lernziel: Ein- und Ausgabe in C

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`.

```c
#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("%d\n", sum/count);
    return 0;
}
```

Aufgabe 2 Fibonacci-Zahlen (6 Punkte)

Lernziel: Ein- und Ausgabe in C, manuelle Speicherverwaltung, Arrays in C

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.

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("fib %d = %d\n", i, fibs[i]);
    }
    free(fibs);
}
```

Aufgabe 3 Eingabe umdrehen (4 Punkte)

Lernziel: Ein- und Ausgabe in C, manuelle Speicherverwaltung

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 benutzen Sie die Funktion `realloc`, um den Speicherplatz wenn nötig zu vergrößern. Geben Sie den Speicher am Ende mit `free` wieder frei.

```c
#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)

Lernziel: Pointer und Arrays in C

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 String-Suche (Sinnvoll bearbeiten)

a) Schreiben Sie eine Methode `bool contains(char const *search, int searchSize, char const *input, int inputSize)`. Die Methode nimmt einen Such-Text `search` der Länge `searchSize` und einen Eingabe-Text `input` der Länge `inputSize`. Die Funktion soll `true` zurückgeben, wenn der gesuchte Text im Eingabe-Text vorkommt und ansonsten `false`.

Verwenden Sie keine Funktionen aus der C-Bibliothek für diese Aufgabe.

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

bool contains(char const *search, int searchSize, char const *input, int inputSize)
{
    for (int i=0; i<inputSize-searchSize; i++) {
        bool found = true;
        for (int j=0; j<searchSize; j++)
        {
            if (input[i+j] != search[j])
            {
                found = false;
                break;
            }
        }
        if (found)
        {
            return true;
        }
    }
    return false;

}
```

b) Schreiben Sie eine `main`-Funktion für ihr Programm, welches einen Such-Text als Programm-Parameter nimmt und die Standard-Eingabe nach Vorkommen des Such-Strings durchsucht. Die Zeilen der Eingabe, welche den Such-Text enthalten, sollen zusammen mit ihrer Zeilennummer ausgegeben werden.

```
int main(int argc, char **argv)
{
    if (argc != 2)
    {
        printf("1 program parameter expected but %d given.\n", argc);
        return 1;
    }
    char const *searchString = argv[1];
    int const searchStringLen = strlen(searchString);
    int bufferSize = 1024;
    char *buffer = malloc(bufferSize);
    int lineNr = 1;
    while (true)
    {
        char *s = fgets(buffer, bufferSize, stdin);
        if (s == NULL)
        {
            break;
        }
        if (contains(searchString, searchStringLen, s, strlen(s))) {
            printf("%d %s", lineNr, buffer);
        }
        lineNr++;
    }
    free(buffer);
}
```