Programmierung 2 (SS 2021)
Universität des Saarlandes
Fakultät MI
Compiler Design Lab

# Musterlösung 6
# C

Prof. Dr. Sebastian Hack
Julian Rosemann, B. Sc.
Thorsten Klößner, M. Sc.
Julia Wichlacz, M. Sc.
Maximilian Fickert, M. Sc.

Im Vorlesungskalender finden Sie Informationen über die Kapitel des Skripts, die parallel zur Vorlesung bearbeitet werden sollen bzw. dort besprochen werden. Die Übungsaufgaben dienen der Vertiefung des Wissens, das in der Vorlesung vermittelt wird und als Vorbereitung auf Minitests und Klausur.
Weitere Aufgaben zu den Themen finden Sie jeweils am Ende der Skriptkapitel.
Die Schwierigkeitsgrade sind durch Steine des 2048-Spiels gekennzeichnet, von 512 „leicht“ bis 2048 „schwer“. 4096 steht für Knobelaufgaben.

## Aufgabe 6.0: printf

Vervollständigen Sie das folgende Programm, so dass es die Reihung von Ganzzahlen `zahlen` in verschiedenen Darstellungen ausgibt.

```c
#include <stdio.h>

int main(void) {
  int zahlen[] = {-13427, -4233, -343, -12, -5, 0,
                  3, 17, 512, 2355, 29367};
  int n = /* Anzahl der Elemente von zahlen berechnen */

  for (int i=0; i < n; ++i) {
    printf("%d␣", zahlen[i]); // Diese Zeile muss jeweils angepasst werden
  }

  return 0;
}
```

Berechnen Sie zunächst mithilfe von `sizeof` die Anzahl der Elemente von `zahlen`. Geben Sie die Reihung dann in folgenden Darstellungen aus:

1. In Hexadezimaldarstellung mit vorangestelltem `0x` und bis zur Breite 8 mit Nullen aufgefüllt, also z.B. `0x00000012`. Welchen Argumenttyp verlangt der Formatspezifizierer für die hexadezimale Ausgabe?
2. Ein Tausendstel jeder Zahl, auf zwei Nachkommastellen gerundet.
3. Die Zahl als gewöhnliche Dezimalzahl, jedoch immer mit Vorzeichen (also auch +)

Jede dieser Teilaufgaben ist mit einer entsprechend gewählten Formatzeichenkette und der Bibliotheksfunktion `printf` zu erledigen.
`printf` ist eine Funktion mit variabler Argumentlänge, d.h. Aufrufe der Form `printf`($format, x_1 \dots x_n$) sind für alle $n$ zulässig. Als *format* wird eine Zeichenkette übergeben, die Formatspezifizierer enthalten kann. In der Ausgabe wird der $i$-te Formatspezifizierer durch die ensprechend formatierte Darstellung von $x_i$ ersetzt. Ein Formatspezifizierer hat die Form

`%[flags][min field width][precision][length]conversion specifier`

wobei alle in `[]` gefassten Parameter optional sind. Die Beschreibung der Parameter finden Sie beispielsweise auf der Handbuchseite zu `printf`. Diese können Sie mit dem Befehl `man 3 printf` abrufen.
Hinweis: Sie sollen lernen, den relevanten Teil zu finden und die benötigten Informationen zu extrahieren.

## Lösung

```c
#include <stdio.h>

int main(void) {
  int zahlen[] = { -13427, -4233, -343, -12, -5, 0, 3, 17, 512, 2355, 54367 };

  int n = sizeof(zahlen) / sizeof(zahlen[0]);

  for (int i = 0; i < n; ++i) {
    printf("0x%08x\t", (unsigned int)zahlen[i]);
    printf("%.2f\t", zahlen[i] / 1000.0);
    printf("%+d\n", zahlen[i]);
  }
  return 0;
}
```

## Aufgabe 6.1: scanf

Schreiben Sie ein Programm, dass den BMI berechnet. Sie müssen dazu die Körpergröße $h$ in Metern als Gleitkommazahl einlesen, sowie die Masse $m$ in Kilogramm als Ganzzahl. Geben Sie dann den BMI $b$, der sich gemäß folgender Gleichung berechnet, aus:

$$b = \frac{m}{h^2}$$

Verwenden Sie zum Einlesen der Zahlen die Bibliotheksfunktion `scanf`, die analog zu `printf` funktioniert. `scanf` verwendet ähnliche Formatspezifizierer wie `prinf`:

`%[max field width][type modifier]conversion specifier`

Beachten Sie, dass `scanf` die *Adresse des Behälters* erwartet, in den die Eingabe geschrieben werden soll. Sie können die Funktionsweise von scanf z.B. auf der Handbuchseite nachlesen, die Sie mit dem Befehl

`man scanf` bzw. die deutsche Seite mit `LANG=de_DE man scanf`

abrufen können.

## Lösung

```c
#include <stdio.h>

int main(void) {
  float h, b;
  int m;

  printf("Körpergröße (in m): ");
  scanf("%f", &h);
  printf("Körpergewicht (in kg): ");
  scanf("%d", &m);

  b = m / (h * h);

  printf("bmi = %f\n", b);

  return 0;
}
```

## Aufgabe 6.2: Simple-Calc

Konrad Klug hat seinen Taschenrechner für Ganzzahloperationen verloren. Helfen Sie ihm, indem Sie ein C-Programm schreiben, das von der Standardeingabe einen Ausdruck

```
<Ganzzahl_1> <Operator> <Ganzzahl_2>
```

einliest und in der nächsten Zeile das Ergebnis in der Form

```
<Ganzzahl_1> <Operator> <Ganzzahl_2> = <Ergebnis>
```

ausgibt. Ihr Taschenrechner sollte hierbei die Rechenoperationen +, -, *, sowie / unterstützen.

## Lösung

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

int main(int argc, char* argv[]) {
    int x = 0;
    int y = 0;
    char op = 0;
    char* s = calloc(64, sizeof(char));

    printf("Bitte geben Sie Ihre Rechnung ein: ");
    scanf("%d %c %d", &x, &op, &y);

    sprintf (s,"%d %c %d = ", x, op, y );
    int slen = strlen(s);
    s += slen;

    switch(op){
        case '+':
            sprintf (s, "%d", (x + y));
            break;
        case '-':
            sprintf (s, "%d", (x - y));
            break;
        case '*':
            sprintf (s, "%d", (x * y));
            break;
        case '/':
            sprintf (s, "%d", (x / y));
            break;
        default:
            sprintf(s - slen, "Ungültige Eingabe!");
            break;
    }

    s -= slen;
    printf( "%s \n", s);

    free(s);
    return 0;
}
```

## Aufgabe 6.3: Fehlersuche mit Malloc

Konrad Klug hat heute einiges über `malloc` und `free` gelernt, jedoch nicht genug Kaffee getrunken bevor er versucht hat, sein neues Wissen anzuwenden. Finden Sie in seinen Code-Schnipseln Fehler, die im Zusammenhang mit `malloc` und `free` gemacht wurden. Erklären Sie jeweils, was der Denkfehler von Konrad Klug ist.


1.

```c
// Reversiert ein Array, gibt Pointer auf reversiertes Array gleicher Größe zurück
// arrayptr: Adresse des ersten Elementes des Arrays
// arraysize: Anzahl an Elementen im zu reversierenden Array
int *arrayrev(int *arrayptr, int arraysize) {
  int *ret = malloc(arraysize);
  for(int i = 0; i < arraysize; i++){
    ret[i] = arrayptr[arraysize-i-1];
  }

  return ret;
}
```

2.

```c
// gibt zweimal die Zahl 5 aus
void broken() {
    int a = 5;
    int *b = malloc(sizeof(int));
    *b = 5;
    printf("zwei␣Zahlen:␣%d,␣%d\n", a, *b);

    free(&a);
    free(&b);
}
```

3.

```c
// gibt Quadrate der Zahlen [1, upto] aus.
void quadratics(int upto) {
    for(int i = 1; i <= upto; i++){
        int *a = malloc(sizeof(int));
        *a = i*i;
        printf("i*i␣=␣␣%d\n", *a);
    }

    free(a);
}
```

4. Bonus: Wieso funktioniert der Code in (i), der Code in (ii) jedoch nicht?

(i)

```c
int main() {
    int array[] = {10, 20, 30, 40};
    int elements = sizeof(array) / sizeof(int);
        for(int i = 0; i < elements; i++){
            printf("%d\n", array[i]);
        }
    }
```

(ii)

```c
void printArray(int array[]) {
    int elements = sizeof(array) / sizeof(int);
    for(int i = 0; i < elements; i++){
        printf("%d\n", array[i]);
    }
}

int main() {
    int array[] = {10, 20, 30, 40};
    printArray(array);
}
```

## Lösung

1. `malloc` erwartet die Größe des zu allozierenden Speichers in Bytes, nicht die Anzahl an Elementen. Richtig wäre: `int *ret = malloc(arraysize * sizeof(int));`

2. `free` erwartet einen Pointer auf ein Behälter, der mit `malloc` angefordert wurde. Der Behälter der lokalen Variablen `a` wird mit Beendigung des umschließenden Blocks (hier die Funktion) automatisch befreit und darf daher nicht explizit mit `free` freigegeben werden. Des Weiteren ist `free(&b)` fehlerhaft, da `b` bereits ein Pointer auf einen mit `malloc` allozierten Speicherbereich ist. Richtig wäre `free(b)`.

3. Es wird beim Benutzen von `free` versucht, auf die Variable `a` zu verweisen. Diese ist jedoch nur in dem Block des for-loops definiert. Dies führt zu einem Compiler-Fehler. Es wird zwar nach jedem Schleifendurchlauf automatisch der Behälter, der an Variable `a` gebunden ist befreit, der Behälter auf den die Adresse in `a` zeigt jedoch nicht. Dies führt zu einem Speicherleck.

4. Führt man den in (ii) Code aus, werden nur die ersten (je nach Prozessorarchitektur und Betriebssystem mehr oder weniger viele) Elemente ausgegeben. Das Problem wird sichtbarer, wenn man sich klarmacht, dass das `int array[]` in der Parameterliste der Funktion `printArray` lediglich syntaktischer Zucker für `int* array` ist: In `sizeof(array)` wird hier also die Größe eines `int*` zurückgegeben. Dagegen wertet im korrekten Code `sizeof(array)` zu der Anzahl an Bytes im Array aus.
   Kompiliert man den fehlerhaften Code mit gcc, liefert es auch gleich eine hilfreiche Warnung:

   > Warnung: »sizeof« auf Array-Funktionsparameter »array« gibt die Größe von »int *« zurück.

## Aufgabe 6.4: Malloc-Strings

1. Schreiben Sie eine Funktion, die einen Integer als Argument erhält und den C-String “False” zurückgibt, falls die Zahl den Wert 0 besitzt. Für jeden anderen Wert soll die Funktion “True” zurückgeben. Die Funktion soll den Rückgabewert in einer lokalen Variable speichern und diese dann zurückgeben.

2. Schreiben Sie ein Hauptprogramm, welches eine Zahl als Kommandozeilenparameter nimmt und mit Hilfe Ihres Unterprogramms aus Aufgabenteil 1 “False” zurückgibt, falls eine 0 übergeben wurde, in allen anderen Fällen “True”. Achten Sie dabei auf korrekte Speicherverwaltung.
   *Tipp: Die Argumente der Main werden immer als Array von `char*` übergeben. Man kann `atoi` verwenden, um die String-Darstellung einer Zahl in den Zahlenwert zu verwandeln.*
   *Zum Beispiel* ***int x = atoi("1234"); //x = 1234***

3. Am nächsten Tag treffen sich Rainer Wahnsinn und Konrad Klug. Konrad Klug staunt über die Fähigkeiten seines Kommilitionen. Jedoch würde Konrad Klug gerne “Richtig” und “Falsch” anstelle von “True” und “False” zurückgeben. Leider hat er sich einen Virus eingefangen, sodass er alle bereits geschriebenen Funktionen (außer der `main`-Funktion) nicht mehr abändern kann. Schreiben Sie eine Funktion `char* convert_to_german(char*)`, die einen gegebenen C-String in das deutsche Äquivalent (also “True” zu “Richtig” und “False” zu “Falsch”) übersetzt. Für den Fall, dass weder “True” noch “False” als C-String übergeben werden, soll “Undefiniert” ebenfalls als C-String zurückgegeben werden. Achtung: Denken Sie

auch hier daran den verwendeten Speicher richtig zu allozieren und danach wieder freizugeben! Es könnte sinnvoll sein die Funktion `strcmp` aus der Standard Bibliothek zu verwenden.

4. Rainer Wahnsinn hat sich einen seltenen Kryptotrojaner eingefangen, der seine `string.h` verschlüsselt hat! Helfen Sie ihm, indem Sie eine eigene Implementierung der `strcpy`-Funktion erstellen.
   *Tipp: Mit `man strcpy` können Sie die Spezifikation der `strcpy` anzeigen. In der VM sind die Manpages nicht standardmäßig installiert, führen Sie dafür den Befehl `sudo pacman -Sy man-pages` aus. Alternativ finden Sie die Manpages auch online, z.B. hier.*
   Testen Sie Ihre Implementierung, indem Sie sie in Ihrer Funktion aus Aufgabenteil 1 verwenden.

## Lösung

(1,2)

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

char*  bool_eval(int x) {
    char* ret;
    if(x == 0) {
        ret = malloc(6 * sizeof(char));
        strcpy(ret,"False");
    }
    else {
        ret = malloc(5 * sizeof(char));
        strcpy(ret,"True");
    }
    return ret;
}

int main(int argc, char* argv[]) {
    if(argc != 2) {
        fprintf(stderr,"Invalid␣number␣of␣arguments!\n");
    }
    else {
        char* test = bool_eval(atoi(argv[1]));
        printf("%s\n",test);
        free(test);
    }
}
```

3.

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

char* bool_eval(int x) {
    char* ret;
    if(x == 0) {
        ret = malloc(6 * sizeof(char));
        strcpy(ret,"False");
    }
    else {
        ret = malloc(5 * sizeof(char));
        strcpy(ret,"True");
    }
    return ret;
}

```

```c
char* convert_to_german(char* string) {
    if(strcmp(string,"True")==0) {
        string = realloc(string, 8 * sizeof(char));
        strcpy(string,"Richtig");
        return string;
    }
    if(strcmp(string,"False")==0) {
        string = realloc(string, 7 * sizeof(char));
        strcpy(string,"Falsch");
        return string;
    }
    string = realloc(string, 12 * sizeof(char));
    strcpy(string,"Undefiniert");
}

int main(int argc, char* argv[]) {
    if(argc != 2) {
        fprintf(stderr,"Invalid␣number␣of␣arguments!\n");
    }
    else {
        char* test = bool_eval(atoi(argv[1]));
        test = convert_to_german(test);
        printf("%s\n",test);
        free(test);
    }
}
```

4.

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

char *strcpy(char *s1, const char *s2)
{
    char *s = s1;
    while ((*s++ = *s2++) != 0)
    ;
    return (s1);
}

char*  bool_eval(int x) {
    char* ret;
    if(x == 0) {
        ret = malloc(6 * sizeof(char));
        strcpy(ret,"False");
    }
    else {
        ret = malloc(5 * sizeof(char));
        strcpy(ret,"True");
    }
    return ret;
}

int main(int argc, char* argv[]) {
    if(argc != 2) {
        fprintf(stderr,"Invalid␣number␣of␣arguments!\n");
    }
    else {
        char* test = bool_eval(atoi(argv[1]));
        printf("%s\n",test);
        free(test);
```

```
33      }
34 }
```

## Aufgabe 6.5: Unterprogramme und Zeiger - Potenzieren

Gegeben sei folgendes Hauptprogramm:

```
1 #include <stdio.h>
2
3 float pot(int n, float x);
4
5 int main() {
6     float y = pot(3, 2.71);
7     printf("%.3f\n", y);
8     return 0;
9 }
```

1. Schreiben Sie eine Funktion `pot`, welche zu einer natürlichen Zahl $n$ und einer reellen Zahl $x$ die $n$-te Potenz von $x$ zurück gibt. Dabei soll `pot` den Prototypen

   ```
   float pot(int n, float x);
   ```

   implementieren.

2. No Hau hat in der letzten Vorlesung von Zeigern erfahren und möchte nun `pot` so modifizieren, dass sie keinen Rückgabewert mehr benötigt. Dazu schreibt sie sich eine Funktion `pot_shiny` mit dem Prototypen

   ```
   void pot_shiny(int n, float x, float *y);
   ```

   Machen Sie es No Hau nach und implementieren Sie `pot_shiny`. Verändern Sie anschließend das obige Hauptprogramm soweit wie nötig, um Ihre neue Funktion `pot_shiny` anstelle von `pot` zu verwenden.

3. Erweitern Sie `pot`, sodass für $n < 0$ oder $x < 0$ immer 0 zurückgegeben wird. Modifizieren Sie auch `pot_shiny`, damit das Unterprogramm unter obigen Bedingungen oder wenn der übergebene Zeiger auf `NULL` zeigt, direkt abbricht.

4. Nun wollen wir die Ergebnisse aus den vorherigen Teilaufgaben vergleichen. Schreiben Sie hierzu ein Unterprogramm gemäß dem Prototypen

   ```
   float pot_switch(int n, float x, int b);
   ```

   sodass, wenn `b == 0` die Potenz mit `pot`, andernfalls mit `pot_shiny` berechnet und zurückgegeben wird. Passen Sie das Hauptprogramm an, indem Sie lediglich das Ergebnis von

   ```
   pot_switch(3, 2.71, 0) != pot_switch(3, 2.71, 1)
   ```

   als Rückgabewert der `main`-Methode verwenden. Machen Sie sich klar, wann das Programm mit 0 terminiert.

## Lösung

1.

```
1 float pot(int n, float x) {
2     float y = 1;
3     while (n-- > 0)
4         y *= x;
5     return y;
6 }
```

2.

```c
#include <stdio.h>

void pot_shiny(int n, float x, float *y);

int main() {
    float y = 0.0f;
    pot_shiny(3, 2.71, &y);
    printf("%.3f\n", y);
    return 0;
}

void pot_shiny(int n, float x, float *y) {
    *y = 1;
    while (n-- > 0)
        *y *= x;
}
```

3.

```c
float pot(int n, float x) {
    if (n < 0 || x < 0)
        return 0;
    ...
}

void pot_shiny(int n, float x, float *y) {
    if (n < 0 || x < 0 || !y)
        return;
    ...
}
```

4.

```c
float pot(int n, float x);
void pot_shiny(int n, float x, float *y);
float pot_switch(int n, float x, int s);

int main() {
    return pot_switch(3, 2.71, 0) != pot_switch(3, 2.71, 1);
}

float pot_switch(int n, float x, int b) {
    if (b) {
        float y = 0;
        pot_shiny(n, x, &y);
        return y;
    }
    return pot(n, x);
}
...
```

## Aufgabe 6.6: Unterprogramme und Zeiger - Fehlerteufel

Konrad Klug ist begeistert von Zeigern und hat folgendes Programm geschrieben, das überprüft, ob eine positive Ganzzahl die Quersumme 7 hat:

```c
int mod_ten(int x) {
    return x % 10;
}

void div_ten(int *x) {
    x = *x / 10;
}

void quer(int *a, int b) {
    if (a > 0)
        b = b + mod_ten(a);
    div_ten(a);
    quer(a, b);
}

int main() {
    int a = 322;
    int b;
    quer(a, &b);
    return b = 7;
}
```

Jedoch muss Konrad feststellen, dass sich während seiner Arbeit Fehler eingeschlichen haben, nachdem er ein koffeinhaltiges Erfrischungsgetränk über seiner Tastatur verschüttet hat, infolgedessen nun einige Tasten klemmen. Helfen Sie Konrad sein Programm zu berichtigen.

## Lösung

```c
int mod_ten(int x) {
    return x % 10;
}

void div_ten(int *x) {
    *x = *x / 10;
}

void quer(int *a, int *b) {
    if (*a > 0) {
        *b = *b + mod_ten(*a);
        div_ten(a);
        quer(a, b);
    }
}

int main() {
    int a = 322;
    int b = 0;
    quer(&a, &b);
    return b != 7;
}
```

## Aufgabe 6.7: Queue

Farmer John möchte ein neues System für seine Melkmaschine etablieren, bei dem die Kühe in der Reihenfolge gemolken werden, in der sie sich anmelden: Wenn eine Kuh gemolken werden will, meldet sie sich mit ihrer Nummer bei der Maschine an. Wenn die Melkmaschine frei ist und sich keine andere ungemolkene Kuh vor ihr angemeldet hat, wird sie aufgerufen und darf zur Maschine gehen.

Konrad Klug meint, dass man das mit einer FIFO (first in first out) Queue mit den Operationen `pushBack()` und `popFront()` lösen kann. Als Datenstruktur bietet sich dafür eine Linked-List an. Ein Element der Liste enthält hierbei zusätzlich zu dem Wert einen Pointer, der auf das Folgeelement zeigt.
Dazu hat er ein kleines Codegerüst aufgebaut, in dem er Sie bittet, die Funktionen `createQueueElem()`, `pushBack()` und `popFront()` zu implementieren. Wenn Sie alles richtig machen, sollten 1, 2 und 3 in dieser Reihenfolge ausgegeben werden.

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

typedef struct Queue Queue;
typedef struct QueueElem QueueElem;
struct QueueElem{
  int value;
  QueueElem* successor;
};
struct Queue{
  QueueElem* front;
  QueueElem* back;
};


QueueElem* createQueueElem(int value){
  // TODO
}

void pushBack(Queue* queue, int value){
  // TODO
}

int popFront(Queue* queue){
  // TODO
}


int main(){
    Queue myQueue = {0,0};
    pushBack(&myQueue, 1);
    pushBack(&myQueue, 2);
    pushBack(&myQueue, 3);

    printf("%d\n", popFront(&myQueue));
    printf("%d\n", popFront(&myQueue));
    printf("%d\n", popFront(&myQueue));

    return 0;
}
```

## Lösung

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

typedef struct Queue Queue;
typedef struct QueueElem QueueElem;

struct QueueElem{
    int value;
```

```
10     QueueElem* successor;
11 };
12
13 struct Queue{
14     QueueElem* front;
15     QueueElem* back;
16 };
17
18 QueueElem* createQueueElem(int value){
19   QueueElem* element = (QueueElem*) malloc(sizeof(QueueElem));
20   element->value = value;
21   return element;
22 }
23
24 void pushBack(Queue* queue, int value){
25   QueueElem* element = createQueueElem(value);
26   if (queue->front == 0){
27     queue->front = element;
28   } else {
29     queue->back->successor = element;
30   }
31   element->successor = 0;
32   queue->back = element;
33 }
34
35 int popFront(Queue* queue){
36     assert(queue->front != NULL);
37     int returnValue = queue->front->value;
38     QueueElem* firstElem = queue->front;
39     if (firstElem->successor == 0){
40         queue->back = 0;
41     }
42     queue->front = firstElem->successor;
43     free(firstElem);
44     return returnValue;
45 }
46
47 int main(){
48     Queue myQueue = {0,0};
49     pushBack(&myQueue, 1);
50     pushBack(&myQueue, 2);
51     pushBack(&myQueue, 3);
52
53     printf("%d\n", popFront(&myQueue));
54     printf("%d\n", popFront(&myQueue));
55     printf("%d\n", popFront(&myQueue));
56
57     return 0;
58 }
```

## Aufgabe 6.8: Binäre Ordnung

Konrad Klug möchte ein C-Programm schreiben, welches mithilfe von Binärbäumen Daten aus einer Datei sortiert. Leider kommt Konrad nicht weiter und benötigt deshalb Ihre Hilfe.

Implementieren Sie einen binären Suchbaum, der als Schlüssel Ganzzahlen benutzt und als Werte Zeichenketten der Länge 20 (inklusive Null-Byte) speichert. Halten Sie sich dabei an das von Konrad vorgegebene Gerüst und implementieren Sie die entsprechenden Funktionen und den Struct gemäß der Kommentare. Beachten Sie die Binärbaumeigenschaft, die zu jedem Zeitpunkt gelten soll: alle Schlüssel im linken Unterbaum sind kleiner und

alle Schlüssel im rechten Unterbaum sind größer als der Schlüssel des Vaterknotens.
Gehen Sie dabei wie folgt vor:

- Schreiben Sie zunächst ein `struct node`, das neben Schlüssel und Wert alle benötigten Informationen speichert.
- Implementieren Sie eine Funtion `createNewNode`, die im Speicher Platz reserviert und den Schlüssel und Wert des Knoten initialisiert.
- Implementieren Sie die Funktion `insertNode`. Dabei kann es hilfreich sein, sich eine zusätzliche Funktion zum Erstellen neuer Knoten anzulegen.
- Implementieren Sie eine Funktion `freeTree`, die den für die Datenstruktur allozierten Speicher wieder frei gibt.
- `printInOrder` soll alle Elemente des Baumes sortiert nach der Größe des Schlüssels ausgeben.
- Zuletzt implementieren Sie eine Funktion `getNameById`, die für einen Schlüssel den entsprechenden Wert im Binärbaum zurückgibt.

*Hinweis: An einigen Stellen bietet es sich an, Rekursion zu benutzen. Außerdem dürfen Sie die Funktionen aus `string.h` benutzen.*

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

#define BUF_SIZE 20

typedef struct node node;

/**
 * Ein Binaerbaumknoten
 */
struct node {
    // TODO
};

/**
 * Initialisiert einen neuen Knoten
 */
node* createNewNode(int n, char* name){
  // TODO
}

/**
 * Fuegt einen Knoten in den durch root markierten Baum mit Schluessel
 * n und Wert name ein.
 */
void insertNode(node** root, int n, char* name) {
    // TODO
}

/**
 * Gibt den fuer den Binaerbaum dynamisch allozierten Speicher wieder frei.
 */
void freeTree(node* root){
    //TODO
}

/**
 * Gibt alle Schluessel-Wert-Paare im Binaerbaum in der natuerlichen
```

```
 * Schluesselordnung aus.
 */
void printInOrder(node* root) {
    // TODO
}

/**
 * Findet einen Wert zu einem gegebenen Schluessel.
 */
char* getNameById(node* root, int id) {
    // TODO
}

int main (int argc, char* argv[]) {
    node* tree = NULL;
    int val;
    char name[BUF_SIZE];

    FILE* in = fopen("data.txt", "r");

    while (fscanf(in, "%d %s", &val, name) == 2) {
        insertNode(&tree, val, name);
    }

    printf("Tree:\n");
    printInOrder(tree);
    printf("\n");

    printf("%s\n", getNameById(tree, 5));
    printf("%s\n", getNameById(tree, 10));
}
```

Welches Ergebnis liefert Ihr Code für den Inhalt der folgenden, zufällig von Konrads Computer ausgewählten Datei?

```
5 Garten
10 vier
1 meinem
-6 In
17 Binaerbaeume
8 stehen
```

## Lösung

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

#define BUF_SIZE 20

typedef struct node node;

/**
 * Ein Binaerbaumknoten
 */
struct node {
    int id;
    char name[BUF_SIZE];
    node* left;
    node* right;
```

```
17 };
18
19 /**
20  * Initialisiert einen neuen Knoten.
21  */
22 node* createNewNode(int n, char* name){
23     node* new_node = malloc(sizeof(*new_node));
24     new_node->id = n;
25     strcpy(new_node->name, name);
26     return new_node;
27 }
28
29 /**
30  * Fuegt einen Knoten in den durch root markierten Baum mit Schluessel
31  * n und Wert name ein.
32  */
33 void insertNode(node** root, int n, char* name){
34     if (*root == NULL) {
35         *root = createNewNode(n, name);
36     } else if (n < (*root)->id) {
37         insertNode(&(*root)->left, n, name);
38     } else {
39         insertNode(&(*root)->right, n, name);
40     }
41 }
42
43 /**
44  * Gibt den fuer den Binaerbaum dynamisch allozierten Speicher wieder frei.
45  */
46 void freeTree(node* root){
47     if(root == NULL){
48         return;
49     } else {
50         freeTree(root->left);
51         freeTree(root->right);
52         free(root);
53     }
54 }
55
56 /**
57  * Gibt alle Schluessel-Wert-Paare im Binaerbaum in der natuerlichen
58  * Schluesselordnung aus.
59  */
60 void printInOrder(node* root){
61     if (root == NULL) {
62         return;
63     } else {
64         printInOrder(root->left);
65         printf("ID: %d name: %s \n", root->id, root->name);
66         printInOrder(root->right);
67     }
68 }
69
70 /**
71  * Findet einen Wert zu einem gegebenen Schluessel.
72  */
73 char* getNameById(node* root, int id){
74     if(root == NULL) {
75         return "No matching Name";
76     } else if (root->id == id) {
77         return root->name;
```

```
78      } else if (id < root->id) {
79          return getNameById(root->left, id);
80      } else {
81          return getNameById(root->right, id);
82      }
83  }
84
85  int main (int argc, char* argv[]){
86      node* tree = NULL;
87      int val;
88      char name[BUF_SIZE];
89
90      FILE* in = fopen("data.txt", "r");
91
92      while (fscanf(in, "%d␣%s", &val, name) == 2) {
93          insertNode(&tree, val, name);
94      }
95
96      printf("Tree:\n");
97      printInOrder(tree);
98      printf("\n");
99
100     printf("%s\n", getNameById(tree, 5));
101     printf("%s\n", getNameById(tree, 10));
102 }
```

Ausgabe:

```
Tree:
ID: -6 name: In
ID: 1 name: meinem
ID: 5 name: Garten
ID: 8 name: stehen
ID: 10 name: vier
ID: 17 name: Binaerbaeume

Garten
vier
```

## Aufgabe 6.9: Defined or Undefined?

Finden Sie für die folgenden Programme heraus, ob die Ausführung eindeutig laut C11-Standard definiert oder undefiniert ist. Gehen Sie davon aus, dass genug Speicherplatz vorhanden ist.


1.

```c
#include <stdio.h>
int main() {
  int a[15];
  for (int i = 0; i < 15; i++) {
    a[i] = 14 - i;
  }
  int x = 3[a];
  printf("Wert von x: %i\n", x);
  return 0;
}
```

2.

```c
#include <stdio.h>
int main() {
  int x = 0;
  int y = 42 % x;
  return 0;
}
```

3.

```c
#include <stdio.h>
int main() {
  printf("%f\n", 0);
  return 0;
}
```

4.

```c
#include <stdio.h>
int sum(int x, int y) {
  return x + y;
}
int main() {
  int i = 5;
  int res = sum(i*i, ++i);
  printf("result of sum: %i", res);
  return 0;
}
```

5.

```c
#include <stdio.h>
int main() {
  int a[10];
  for (int i = 0; i < 11; i++) {
    a[i] = a + i;
  }
  return 0;
}
```

6.

```c
#include <stdio.h>
int main() {
  char* p = NULL;
  char c = *p;
  return 0;
}
```

7.

```c
#include <stdio.h>
int foo() {
  int a = 1234;
  double b = 12.34;
}
int main() {
  int x = foo();
  return 0;
}
```

8.

```c
#include <stdio.h>
int main() {
  int x;
  x ^= x;
  return 0;
}
```

## Lösung

1. Das Programm enthält kein undefiniertes Verhalten. Die Ausgabe des Programms ist `Wert von x: 11`. Der Ausdruck `3[a]` ist nur syntaktischer Zucker für `*(3+a)` und damit gültig.
2. Das Verhalten bei einer Division bzw. Modulorechnung durch Null ist undefiniert.
3. Das Verhalten ist undefiniert. `%f` erwartet einen Wert vom Typ `double`, 0 ist allerdings vom Typ `int`.
4. Der C-Standard legt keine Auswertungsreihenfolge für die Argumente fest. Die hier entstehende Situation nennt sich “unsequenced read/write” und das Verhalten in diesem Fall ist undefiniert, da nicht fest steht, ob `i` zuerst erhöht oder ausgelesen wird.
5. Das Verhalten ist undefiniert. Die Berechnung `a + i` ist erlaubt, auch im Fall `i = 10`, da man Zeiger auf das Element nach der Reihung nutzen darf. Der Zugriff `a[i]` im Fall `i = 10` ist jedoch ungültig, da man nicht auf das Element hinter der Reihung zugreifen darf. Zudem erfordert dieser Zugriff eine Konvertierung von einem Pointer zu `int`, was implementation-defined ist und somit auch undefiniertes Verhalten sein kann.
6. Die Deferenzierung eines NULL-Zeigers führt zu undefiniertem Verhalten.
7. Hier tritt undefiniertes Verhalten auf, da in `main()` die Funktion `foo()` aufgerufen wird und auf den Rückgabewert dieser zugregriffen wird, obwohl in `foo()` allerdings das Schlüsselwort `return` fehlt.
8. Das Lesen von nicht initialisiertem Speicher resultiert in undefiniertem Verhalten (konkret ist nicht nur der Wert undefiniert, sonst könnte man meinen, dass hier immer 0 rauskommen müsste - statt einen Wert zurückzugeben dürfte das Programm aber z.B. auch mit einer Fehlermeldung abstürzen, oder weiterrechnen mit $x \neq 0$).

## Aufgabe 6.10: Argument-Parser

In dieser Aufgabe sollen Sie einen einfachen Taschenrechner implementieren. Ihr Programm soll die zu berechnenden Zahlen und die Rechenoperation als Programmparameter von der Kommandozeile einlesen. Hierfür müssen Sie einen sogenannten Argument-Parser, kurz argparser, schreiben. Verwenden Sie dabei die Funktion `getopt`, die Sie zusätzlich durch `#include <getopt.h>` inkludieren müssen.
Ihr Programm soll die Parameter `h`, `x`, `y`, `r` und `c` unterstützen:

1. `-h`: gibt eine kurze Hilfestellung zur Verwendung des Programmes aus
2. `-r <char>`: nach `-r` folgt das Zeichen der entsprechenden Rechenoperation (`+`, `-`, `*`, `/`)
3. `-x <float>`: nach `-x` folgt die erste Zahl (als `float`) der Rechenoperation
4. `-y <float>`: nach `-y` folgt die zweite Zahl der Rechenoperation
5. `-c <char*>`: nach `-c` folgt eine Zeichenkette, die erneut ausgegeben werden soll

Wird der Parameter `-h` übergeben, erwartet das Programm keine weiteren Parameter und beendet die Ausführung direkt nach dem Ausgeben der Hilfestellung. In allen anderen Fällen muss jedoch sichergestellt werden, dass die Parameter `x`, `y` und `r` immer gegeben sind, `-c` ist dagegen optional. Informationen zu `getopt` finden Sie in der Man-Page mit Hilfe des Befehls `man 3 getopt`.

(a) Implementieren Sie für Ihren Taschenrechner die Rechenoperationen `+`, `-` und `/`. Geben Sie das erechnete Ergebnis auf fünf Nachkommastellen genau aus.

(b) Erweitern Sie Ihren Taschenrechner um die Rechenoperation `*`. Was fällt Ihnen auf, wenn Sie das Programm mit dem Parameter `*` ausführen wollen?

(c) Erweitern Sie Ihren Taschenrechner um die Kommentarfunktion `-c`. Testen Sie das Programm mit beliebigen Zeichenketten. Welches Problem fällt Ihnen dabei auf?

(d) Finden Sie Lösungen für die Besonderheiten aus b) und c).

## Lösung

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

int calculate(float x, float y, char r);

int main(int argc, char *argv[]) {

    char c;

    float x, y;
    char r;

    int xflag = 0;
    int yflag = 0;
    int rflag = 0;

    while ((c = getopt(argc, argv, "hx:y:r:c:")) != -1) {

        switch (c) {
            case 'h':
```

```
22                 printf("Folgende Parameter sind erlaubt:\n\
23           -r <char>: Rechenoperator, der die Rechenoperation angibt\n\
24           -x <float>: 1. Zahl für die gegebene Rechenoperation\n\
25           -y <float>: 2. Zahl für die gegebene Rechenoperation\n\
26           -c <char*>: beliebiges Kommentar\n\
27           Hierbei müssen -r, -x und -y immer gegeben sein!\n");
28                 exit(0);
29
30             case 'x':
31                 x = atof(optarg);
32                 xflag = 1;
33                 break;
34
35             case 'y':
36                 y = atof(optarg);
37                 yflag = 1;
38                 break;
39
40             case 'r':
41                 r = *optarg;
42                 rflag = 1;
43                 break;
44
45             case 'c':
46                 printf("Kommentar: %s\n", optarg);
47                 break;
48
49             case '?':
50                 if ((optopt == 'x') || (optopt == 'y') || (optopt == 'r')) {
51                     printf("Parameter -%c benötigt genau ein Argument!\
52                             Siehe -h für mehr Hilfe.\n", optopt);
53                 } else {
54                     printf("Error: Parameter -%c nicht bekannt.\n", optopt);
55                 }
56                 exit(1);
57
58             default:
59                 printf("Error: optind %i, argc %i\n", optind, argc);
60                 exit(1);
61         }
62     }
63
64         if (rflag == 0 || xflag == 0 || yflag == 0) {
65             printf("Es wurden nicht alle erforderlichen Parameter übergeben!\
66                     Siehe -h für mehr Hilfe.\n");
67             exit(1);
68         }
69
70     int ret = calculate(x, y, r);
71     exit(ret);
72
73 }
74
75
76 int calculate(float x, float y, char r) {
77
78     switch(r) {
79         case '+':
80             printf("%.5f + %.5f = %.5f\n", x, y, x+y);
81             return 0;
82         case '*':
```

```
83              printf("%.5f * %.5f = %.5f\n", x, y, x*y);
84              return 0;
85          case '-':
86              printf("%.5f - %.5f = %.5f\n", x, y, x-y);
87              return 0;
88          case '/':
89              printf("%.5f / %.5f = %.5f\n", x, y, x/y);
90              return 0;
91          default:
92              printf("Rechenoperator %c ist nicht spezifiziert!\n", r);
93              return 1;
94      }
95 }
```

Will man das obige Programm beispielsweise durch

```
./calculator -x 3 -y 4 -r *
```

aufrufen, wird '*' nicht richtig eingelesen. Dies liegt daran, dass '*' bereits ein Bash-Symbol ist und schon von der Konsole uminterpretiert wird.
Um das Problem zu lösen, kann man obiges Programm mit

```
./calculator -x 3 -y 4 -r '*'
```

oder

```
./calculator -x 3 -y 4 -r \*
```

aufrufen.
Will man als Parameter für die Kommentar-Funtkion `-c` eine Zeichenkette eingeben, die ein Leerzeichen enthält, muss diese ebenfalls in Anführungszeichen gesetzt werden:

```
./calculator -x 3 -y 4 -r + -c "Hallo Welt!"
```

Ohne diese Anführungszeichen wird vor dem Leerzeichen das Lesen beendet, sodass hier nur "Hallo" eingelesen werden würde.

## Aufgabe 6.11: Polynom

Wir betrachten den Verbund `poly_t` aus Beispiel 3.10 im Skript. Ergänzen Sie die fehlernden Unterprogramme in `poly.c` aus Abbildung 3.13 im Skript:

1. Schreiben Sie einen Konstruktor für Polynome, der den Verbund und die Reihung der Koeffzienten auf der Halde anlegt.
2. Schreiben Sie eine Routine `polynom_free`, die ein auf der Halde angelegtes Polynom (inkl. Koeffizientenreihung) freigibt.
3. Implementieren Sie die Polynomauswertung mit dem Horner-Schema für diesen Verbund.
4. Implementieren Sie einen Konstruktor, der die Koeffizienten eines Polynoms aus einer Datei liest. Die Koeffizienten liegen in der Datei durch Leerzeichen getrennt. Der Koeffeizient für das höchste Glied steht zuerst.

## Lösung

poly.h

```
1 #ifndef POLY_H
2 #define POLY_H
3
4 /* unvollständiger Verbund */
```

```
 5 typedef struct poly_t poly_t;
 6
 7 poly_t* poly_alloc(unsigned degree);
 8 void poly_free(poly_t* p);
 9 void poly_set_coeff(poly_t* p, unsigned deg, int coeff);
10 int poly_eval(poly_t const* p, int x);
11 unsigned poly_degree(poly_t const* p);
12
13 #endif /* POLY_H */
```

poly.c

```
 1 #include <stdlib.h>
 2 #include <assert.h>
 3
 4 #include "poly.h"
 5
 6
 7 struct poly_t{
 8     unsigned degree;
 9     int* coeffs;
10 };
11
12 /*
13  * Gibt den von einem Polynom belegten Speicherplatz wieder frei
14  */
15 void poly_free(poly_t* p){
16
17     assert(p != NULL);
18     free(p->coeffs);
19     free(p);
20
21     return;
22 }
23
24 /*
25  * Creates a polynom with the given degree and sets all coifficients initiallie to zero
26  */
27 poly_t* poly_alloc(unsigned degree){
28
29     poly_t* polynom = malloc(sizeof(poly_t));
30     polynom->coeffs = calloc(degree+1, sizeof(int));
31     polynom->degree = degree;
32
33     return polynom;
34 }
35
36 /*
37  * Sets the degs'th coeifficent to coeff
38  */
39 void poly_set_coeff(poly_t* p, unsigned deg, int coeff){
40
41     assert(deg <= p->degree);
42     p->coeffs[deg] = coeff;
43 }
44
45 /*
46  * Returns the degree of a given polynom
47  */
48 unsigned poly_degree(poly_t const* p){
49
```

```
50      return p->degree;
51  }
52
53  /*
54   * Helper method for poly_eval wich computes the power x^n
55   */
56  int power(int base, int exponent){
57
58      assert(exponent >= 0);
59      int res = 1;
60      for(int i=1; i<=exponent; i++){
61          res *= base;
62      }
63      return res;
64  }
65
66
67  /*
68   * Evaluates the polynom p for a given x
69   */
70  int poly_eval(poly_t const* p, int x){
71
72      assert(p != NULL);
73      int res = 0;
74      for(int i=0; i<=p->degree; i++){
75          res = res + (p->coeffs[i])*power(x,i);
76      }
77
78      return res;
79  }
```

main.c

```
1  include <stdio.h>
2  #include <stdlib.h>
3
4  #include "poly.h"
5
6  int main(int argc, char* argv[]) {
7
8      if(argc < 3){
9          fprintf(stderr, "syntax:␣%s␣x␣path", argv[0]);
10         return 1;
11     }
12
13     int x = atoi(argv[1]);
14     int y = 0;
15     int temp;
16     FILE* file = fopen(argv[2], "r");
17
18     if(file == NULL){
19
20         printf("invalid␣file␣path\n", y);
21         return 1;
22     }else{
23
24         unsigned degree = 0;
25         while((temp = fgetc(file)) != EOF){
26             if(temp != 32) {
27                 degree++;
28             }
```

```
29          }
30
31          fclose(file);
32          file = fopen(argv[2], "r");
33
34          poly_t* p = poly_alloc(degree);
35
36          int i=0;
37          degree--;
38          while ((temp = fgetc(file)) != EOF){
39              if(temp!=32){
40                  poly_set_coeff(p, degree, temp-48);
41                  degree--;
42                  i++;
43              }
44          }
45
46          fclose(file);
47          y = poly_eval(p, x);
48          poly_free(p);
49          printf("%d\n", y);
50      }
51
52      return 0;
53  }
```

## Aufgabe 6.12: Kalender

Schreiben Sie ein C Programm, welches als Grundlage für einen Kalender genutzt werden kann. Ihr Programm soll dabei:

- Einen Struct anlegen, mit dem ein Termin gespeichert werden kann. Ein Termin besteht aus einem Titel und einem Datum (aus Gründen der Übersichtlichkeit ignorieren wir hier Start- und Endzeit).
- Einen Struct anlegen, mit dem ein Datum gespeichert werden kann. Ein Datum besteht aus Tag, Monat und Jahr.
- Die Möglichkeit haben, einen Termin wahlweise von der Konsole oder aus einer Datei einzulesen (Wenn Sie wollen natürlich auch beides!). Ein Termin wird dabei immer in der Reihenfolge Tag Monat Jahr Titel eingelesen.
- Einen Termin auf der Konsole ausgeben können.
- *Challenge*: Alle Termine in einer Datenstruktur Ihrer Wahl organisieren und alle chronologisch geordnet ausgeben.
- *Challenge*: Alle in der Datenstruktur gespeicherten Termine in eine Datei schreiben und beim Starten alle in dieser Datei gespeicherten Termine einlesen.

## Lösung

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
4
5 typedef struct {
6     int year;
7     int month;
8     int day;
9 } Date;
```

```
10
11 typedef struct {
12     Date *date;
13     char *title;
14 } Event;
15
16 Date* init_Date(int day, int month, int year) {
17     Date *date = malloc(sizeof(Date));
18     date->year = year;
19     date->month = month;
20     date->day = day;
21     return date;
22 }
23
24 Event* init_Event(char *title) {
25     Event *event = malloc(sizeof(Event));
26     event->title = title;
27     return event;
28 }
29
30 void destroy_Event(Event *event) {
31     free(event->date);
32     free(event->title);
33     free(event);
34 }
35
36 Event *read_Event(FILE* fp) {
37     int day, month, year;
38     Event *event = init_Event(malloc(sizeof(char) * 256));
39     int i = scanf("␣%d␣%d␣%d␣%255[^\n]s", &day, &month, &year, event->title);
40     if (i != 4) {
41         free(event->title);
42         free(event);
43         return NULL;
44     };
45     event->date = init_Date(day, month, year);
46     return event;
47 }
48
49 //Begin Challenge 1
50
51 typedef struct list {
52     Event* event;
53     struct list* next;
54 } List;
55
56 List* read_Events() {
57     List* list = NULL;
58     int i;
59     printf("Wie␣viele␣Termine␣sollen␣es␣sein?␣");
60     if (scanf("%d", &i) != 1) {
61         return NULL;
62     }
63     List **tail = &list;
64     for (; i > 0; i--) {
65         *tail = malloc(sizeof(List));
66         (*tail)->event = read_Event(stdin);
67         (*tail)->next = NULL;
68         if ((*tail)->event == NULL) {
69             return NULL;
70         }
```

```
71          tail = &(*tail)->next;
72      }
73      return list;
74  }
75
76  int compare_Date(Date* a, Date* b) {
77      if (a->year < b->year) {
78          return -1;
79      }
80      if (a->year > b->year) {
81          return 1;
82      }
83      if (a->month < b->month) {
84          return -1;
85      }
86      if (a->month > b->month) {
87          return 1;
88      }
89      if (a->day < b->day) {
90          return -1;
91      }
92      return a->day > b->day;
93  }
94
95  //clever pointer-to-pointer magic that properly unlinks
96  Event *unlink_max(List** delete_entry) {
97      List *list = *delete_entry;
98      if (!list) return NULL;
99
100     Event* max = list->event;
101     List* last = list;
102     list = list->next;
103     while (list) {
104         if (compare_Date(list->event->date, max->date) > 0) {
105             max = list->event;
106             delete_entry = &last->next;
107         }
108         last = list;
109         list = list->next;
110     }
111     if (max != NULL) {
112         List* tofree = *delete_entry;
113         (*delete_entry) = (*delete_entry)->next;
114         free(tofree);
115     }
116     return max;
117 }
118
119 List* bubblesort(List* in) {
120     List* newlist = NULL;
121     while (in) {
122         Event* event = unlink_max(&in);
123         List* next = newlist;
124         newlist = malloc(sizeof(List));
125         newlist->next = next;
126         newlist->event = event;
127     }
128     return newlist;
129 }
130
131 void print_List(List* list) {
```

```
132     for (;list;list = list->next) {
133         Event* event = list->event;
134         printf("Der␣Termin␣lautet␣%02d.%02d.%04d␣%s\n", event->date->day,
135                 event->date->month, event->date->year, event->title);
136     }
137 }
138 
139 void destroy_List(List* list) {
140     List* next;
141     while (list) {
142         next = list->next;
143         destroy_Event(list->event);
144         free(list);
145         list = next;
146     }
147 }
148 
149 //End Challenge 1
150 
151 //Challenge 2 hat keine Musterlösung
152 
153 int main() {
154     List* list = read_Events();
155     if (!list) {
156         fprintf(stderr, "Fehler␣beim␣Einlesen!\n");
157         return 1;
158     }
159     list = bubblesort(list);
160     print_List(list);
161     destroy_List(list);
162     return 0;
163 }
```