



Klausurdeckblatt

Name der Prüfung: Systemnahe Software I
Datum und Uhrzeit: 17. Februar 2025, 14-16 Uhr Prüfer: Dr. Andreas F. Borchert
Bearbeitungszeit: 120 Min. Institut: Numerische Mathematik

Vom Prüfungsteilnehmer auszufüllen:

Name: _____ Studiengang: _____ Matrikelnummer: _____
Vorname: _____ Abschluss: _____

Datum, Unterschrift des Prüfungsteilnehmers

Hiermit erkläre ich, dass ich prüfungsfähig bin. Sollte ich aufgrund fehlender Anmeldung über das Hochschulportal oder über das Studiensekretariat nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

Hinweise zur Prüfung:
siehe nächstes Blatt

Erlaubte Hilfsmittel:
Bis zu fünf handgeschriebene Blätter.

Bitte dieses Feld für den Barcode freilassen!

Vom Prüfer auszufüllen:

Erreichte Punkte: _____

Note: _____

Datum, Unterschrift Prüfer (Dr. Andreas F. Borchert)

- Prüfen Sie zu Beginn, ob Ihre Klausur vollständig ist, beginnend von der Aufgabe 1 auf Seite 0(!) bis zur letzten Seite 22.
- Bitte reißen Sie die zusammengeheftete Klausur nicht auseinander und entfernen Sie auch nicht einzelne Blätter. Wenden Sie sich bitte an die Aufsicht, wenn Sie mehr Blätter für Ihre Lösungen benötigen.
- Für Ihre Lösungen verwenden Sie bitte den freigelassenen Platz nach der Aufgabenstellung, die gegenüberliegende Seite der jeweiligen Aufgabe oder die angehängte leere Seite unter Angabe der Aufgabennummer.
- Nennen Sie möglichst alle Annahmen, die Sie gegebenenfalls für die Lösung einer Aufgabe treffen!
- Sofern nichts anderes angegeben ist, können Sie bei den Programmier-Aufgaben auf die Angabe der notwendigen *#include*-Anweisungen verzichten.
- Wenn es bezüglich der Aufgabenstellung Unklarheiten gibt, dann fragen Sie bitte jemanden von der Aufsicht.
- Wenn wir während der Klausur feststellen, dass eine Aufgabenstellung missverständlich ist, werden wir an der Tafel einen klärenden Hinweis für alle sichtbar hinschreiben.
- Hinweise zu der Bewertung:
 - Punktzahlen für Teilaufgaben werden nur ganzzahlig vergeben. Wenn notwendig, wird abgerundet.
 - Bei Aufgaben, bei denen Antworten anzukreuzen sind, löscht ein falsches Kreuz ein korrektes Kreuz aus. Negative Punkte für Teilaufgaben werden jedoch nicht vergeben, schlimmstenfalls sind es nur 0 Punkte.

| Nr | Max | Bewertung | | Nr | Max | Bewertung | |
|----------|-----------|-----------|--|--------------|------------|-----------|--|
| 1 | 17 | | | 5 | 8 | | |
| (a) | 4 | | | (a) | 4 | | |
| (b) | 3 | | | (b) | 4 | | |
| (c) | 4 | | | 6 | 11 | | |
| (d) | 2 | | | (a) | 1 | | |
| (e) | 4 | | | (b) | 2 | | |
| 2 | 18 | | | (c) | 8 | | |
| (a) | 3 | | | 7 | 14 | | |
| (b) | 3 | | | 8 | 14 | | |
| (c) | 5 | | | (a) | 3 | | |
| (d) | 7 | | | (b) | 3 | | |
| 3 | 6 | | | (c) | 4 | | |
| 4 | 12 | | | (d) | 4 | | |
| (a) | 8 | | | Summe | 100 | | |
| (b) | 4 | | | | | | |

Prioritäten und Assoziativitäten Cheatsheet + korrekte Datentypen achten

Aufgabe 1

(17 Punkte) Programmier-Techniken

(a) 4 Punkte

Welchen Wert haben die folgenden arithmetischen Ausdrücke?

(a) $14 / 12$

(b) $13 \% 8$

(c) $15.0 / 6$

(d) $11 \% 6 * 12$

(e) $10 * 11 \% 5$

(f) $7 << 3$

(g) $3 ^ 14$

detail rechnen

(h) $0777 \& \sim 027$

750 ?

Lösung:

(b) 3 Punkte

Gegeben sei die folgende Deklaration:

int $x = 3, y = 6;$

Geben Sie jeweils das Endresultat der folgenden Ausdrücke an:

(a) $x > y != 0$

(b) $--x / y ? x : y$

(c) $y /= x, y *= x$

Lösung:

sequenzialisiert

Für x, y Prioritäten wichtig

$2/6 = 0$ also kommt 6 raus

(c) 4 Punkte

Die unten stehende Fassung der Funktion *create_complex* wird fehlerfrei übersetzt. Trotzdem gibt es zur Laufzeit ein undefiniertes Verhalten. Was ist falsch? Korrigieren Sie bitte die Funktion, ohne die Schnittstelle zu verändern.

```
typedef struct {
    double real;
    double img;
} Complex;
```

```
Complex* create_complex(double real, double img) {
    Complex c;
    c.real = real;
    c.img = img;
    return &c;
}
```

← hört auf zu existieren nach der Fkt (kein Speicher wird assoziiert)

Lösung:

(d) 2 Punkte

Gegeben sei folgender C-Code:

```
int zahl;
for (zahl = 1234; zahl >= 1; zahl /= 10) {
    printf("%d\n", zahl % 10);
}
printf("\n");
```

Welche Ausgabe erzeugt dieser Code?

Lösung:

(e) 4 Punkte

Schreiben Sie eine Funktion, die zwei ganze 64-Bit Zahlen n und m des Typs **unsigned long long** erhält und das ebenfalls ganzzahlige nicht-negative Ergebnis n^m zurückliefert. Die Verwendung von Gleitkommazahlen und Funktionen für Gleitkommazahlen ist nicht zulässig. Die Problematik eines Überlaufs darf ignoriert werden.

Lösung:

über Zahlendarstellung nachdenken, hier ist's sehr
einfach => Klausur wird komplizierter

Aufgabe 2**(18 Punkte)** Funktionen und Strukturen

(a) 3 Punkte

Bitte kreuzen Sie bei den folgenden Behauptungen zu den Datentypen **float** oder **double** an, ob sie zutreffen oder inkorrekt sind. Es wird dabei jeweils der Standard IEEE-754 vorausgesetzt:

| Behauptung | trifft zu | ist falsch |
|--|-------------------------------------|-------------------------------------|
| 0.2 ist präzise darstellbar | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| 0.25 ist präzise darstellbar | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| $2^{31} - 1 = 2147483647$ ist präzise als float darstellbar <i>nahtisse ist nicht lang genug</i> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| $-\infty$ ist darstellbar | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| Alle ganzzahligen Werte des Typs float sind als Werte des Typs long darstellbar | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| Es gilt immer $(a + b) + c == (b + c) + a$ für alle möglichen Werte für a, b und c | <input type="checkbox"/> | <input checked="" type="checkbox"/> |

(b) 3 Punkte

Bei unsigned würde es gehen

Stellen Sie fest, ob C bei der Parameterübergabe *call by value* oder *call by reference* unterstützt und zeigen Sie dann an einem kleinen Beispiel, mit welcher Technik auch die andere Art der Parameterübergabe erfolgen kann. Nennen Sie auch ein Beispiel aus der C-Standardbibliothek, das die alternative Parameterübergabeform verwendet.

Lösung:

(c) 5 Punkte

Geben Sie bitte in den folgenden Fällen an, ob die Deklarationen zulässig sind, und falls ja, beschreiben Sie den Typ des deklarierten Objekts:

| Deklaration | Ok | Falsch | Beschreibung, falls ok |
|---|-------------------------------------|-------------------------------------|------------------------|
| int <i>a</i> [10]; | <input type="checkbox"/> | <input type="checkbox"/> | |
| int <i>b</i> (int); | <input type="checkbox"/> | <input type="checkbox"/> | |
| int * <i>c</i> (int); | <input type="checkbox"/> | <input type="checkbox"/> | |
| int *(* <i>d</i>)[10]; | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |
| int <i>e</i> [10](int); | <input type="checkbox"/> | <input checked="" type="checkbox"/> | |

eine Fkt. kann kein Array zurückgeben!

(d) 7 Punkte

Eine Transformationsmatrix für Punkte $(x, y) \in \mathbb{R}^2$ kann mit den Komponenten a, b, c, d, t_x, t_y dargestellt werden, die die folgende 3×3 -Matrix repräsentieren, wobei der dritte Spaltenvektor immer die gleichen konstanten Werte aufweist:

$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

Ein Kollege von Ihnen hat folgende Typdefinition und mögliche Varianten für eine Multiplikation von Transformationsmatrizen zusammengestellt:

```
typedef struct { double a, b, c, d, tx, ty; } TMatrix;
```

```
TMatrix mult1(TMatrix t1, TMatrix t2) {
    return (TMatrix) { t1.a * t2.a + t1.b * t2.c, t1.a * t2.b + t1.b * t2.d,
                      t1.c * t2.a + t1.d * t2.c, t1.c * t2.b + t1.d * t2.d,
                      t1.tx * t2.a + t1.ty * t2.c + t2.tx, t1.tx * t2.b + t1.ty * t2.d + t2.ty };
}
```

geht problematisch

```
TMatrix mult2(const TMatrix* t1, const TMatrix* t2) {
    return (TMatrix) { t1->a * t2->a + t1->b * t2->c,
                      t1->a * t2->b + t1->b * t2->d,
                      t1->c * t2->a + t1->d * t2->c, t1->c * t2->b + t1->d * t2->d,
                      t1->tx * t2->a + t1->ty * t2->c + t2->tx,
                      t1->tx * t2->b + t1->ty * t2->d + t2->ty };
}
```

geht

```
void mult3(TMatrix t1, const TMatrix t2) {
    TMatrix p; p.a = t1.a * t2.a + t1.b * t2.c; p.b = t1.a * t2.b + t1.b * t2.d;
    p.c = t1.c * t2.a + t1.d * t2.c; p.d = t1.c * t2.b + t1.d * t2.d;
    p.tx = t1.tx * t2.a + t1.ty * t2.c + t2.tx; p.ty = t1.tx * t2.b + t1.ty * t2.d + t2.ty;
    t1 = p;
}
```

gibt nicht zurück

```
void mult4(TMatrix* t1, const TMatrix* t2) {
    TMatrix p; p.a = t1->a * t2->a + t1->b * t2->c;
    p.b = t1->a * t2->b + t1->b * t2->d;
    p.c = t1->c * t2->a + t1->d * t2->c; p.d = t1->c * t2->b + t1->d * t2->d;
    p.tx = t1->tx * t2->a + t1->ty * t2->c + t2->tx;
    p.ty = t1->tx * t2->b + t1->ty * t2->d + t2->ty;
    *t1 = p;
}
```

keine Falle in der Mult.

geht, aber nicht

(weil const) A übergeben, zwisst

A nach C kopieren!

Geben Sie für jede der Funktionen in der folgenden Tabelle an, ob sie jeweils wohldefiniert und korrekt ist. Falls dies der Fall ist, sollte angegeben werden, wie die jeweilige Funktion zu verwenden ist, um zwei Transformationsmatrizen A und B zu multiplizieren, wobei das Ergebnis in C abzulegen ist, ohne A oder B zu verändern:

keine const

```
const TMatrix A = {1, 0, 0, 1, 3, 7}; /* Verschiebung um (3, 7) */
const TMatrix B = {2, 0, 0, 2, 0, 0}; /* Skalierung mit Faktor 2 */
TMatrix C;
/* Ihr einzufügender Programmtext weiter unten */
printf("C.a, C.b, C.c, C.d, C.tx, C.ty)\n",
      C.a, C.b, C.c, C.d, C.tx, C.ty);
```

Tragen Sie in der folgenden Tabelle Ihre Antworten ein. Kreuzen Sie „OK“ an und spezifizieren Sie den einzufügenden Programmtext in der rechten Zeile, falls die Funktion wohldefiniert und korrekt ist, andernfalls setzen Sie das Kreuz in der Spalte „Falsch“.

| Funktion | OK | Falsch | einzufügender Programmtext, falls ok |
|--------------|--------------------------|--------------------------|--------------------------------------|
| <i>mult1</i> | <input type="checkbox"/> | <input type="checkbox"/> | _____ |
| <i>mult2</i> | <input type="checkbox"/> | <input type="checkbox"/> | _____ |
| <i>mult3</i> | <input type="checkbox"/> | <input type="checkbox"/> | _____ |
| <i>mult4</i> | <input type="checkbox"/> | <input type="checkbox"/> | _____ |

Aufgabe 3**(6 Punkte)** Makros

In C wird der zu übersetzende Programmtext durch den sogenannten Präprozessor gefiltert. Es seien die vier Dateien *main.c*, *a.h*, *b.h* und *c.h* gegeben (siehe unten). Geben Sie die Ausgabe des Präprozessors an, wie sie etwa durch das Kommando `gcc -E main.c` erzeugt wird. (Sie können dabei die Leerzeilen und die durch den Präprozessor normalerweise erzeugten Zusatzzeilen mit Dateinamen und Zeilennummern weglassen.)

main.c

```
Anfang
#include "a.h"
A B C D E F
#include "b.h"
A B C D E F
#include "c.h"
A B C D E F
Ende
```

a.h

```
Anfang A
#ifndef A_H
#define A_H

#define Ende C
#define Anfang B

#endif
Ende A
```

b.h

```
Anfang B
#ifndef B_H
#define B_H

#ifndef Hallo
#define F C
#endif
#define E F
#include "c.h"

#endif
Ende B
```

c.h

```
Anfang C
#ifndef C_H
#define C_H

#ifdef F
#define Hallo B
#endif
#define D Anfang A
#include "b.h"

#endif
Ende C
```

Lösung bitte auf der gegenüberliegenden Seite!

Lösung:

Kommt garantiert!

Anfang

Anfang A

C A

A B C D E F

B B

B C

B B

C B

C C

C B C B A C C

...

A_H merken

Ende C

Anfang B

B_H

F C

E C

C_H

Hallo B

D B A

Aufgabe 4**(12 Punkte)** Makefile

Es sei das folgende aus fünf Dateien bestehende C-Programm gegeben. Das Bild zeigt schemenhaft den Aufbau des Programms. Die Rechtecke stellen jeweils Dateien dar, deren Name rechts oberhalb des Rechtecks steht.

calc.c

```
#include "reader.h"
#include "eval.h"

int main() {
    Node* root = read_tree();
    if (root) {
        printf("%d\n",
            evaluate(root));
    } else {
        printf("syntax_error\n");
    }
}
```

tree.h

```
#ifndef CALC_TREE_H
#define CALC_TREE_H

typedef struct node {
    int val;
    char op;
    struct node* left;
    struct node* right;
} Node;

#endif
```

reader.h

```
#ifndef CALC_READER_H
#define CALC_READER_H

#include "tree.h"
Node* read_tree();

#endif
```

reader.c

```
#include "reader.h"

Node* read_tree() {
    // ...
}
```

eval.h

```
#ifndef CALC_EVAL_H
#define CALC_EVAL_H

#include "tree.h"

int evaluate(Node* root);

#endif
```

eval.c

```
#include "eval.h"

int evaluate(Node* root) {
    // ...
}
```

calc: calc.o reader.o eval.o
 calc.o : calc.c reader.h eval.h tree.h
 reader.o : reader.c reader.h tree.h
 eval.o : eval.c eval.h tree.h

sollte ausreichen, in der Form

Kommt ~~vor~~ auch in bisschen anderer Form

(a) 8 Punkte

Schreiben Sie ein *Makefile*, das aus den gegebenen Dateien mit Hilfe des gcc-Compilers in ein ausführbares Programm mit dem Namen *calc* erzeugt. Das *Makefile* sollte sämtliche zu erkennenden Abhängigkeiten in minimaler Weise berücksichtigen.

Entsprechend sollte bei einer Änderung einer der sechs Dateien ein anschließender Aufruf von *make* nur zur Neuübersetzung der Programmtexte führen, die zwingend neu übersetzt werden müssen. Sie dürfen dabei die üblichen voreingestellten Regeln von *make* zum Aufruf des C-Compilers und zum Zusammenbau als gegeben voraussetzen, so dass nur die Abhängigkeiten anzugeben sind.

Lösung:

auf indirekte Abhängigkeiten achten
bspw. *readers.h* inkludiert *tree.h*
auf default Regeln verlassen

(b) 4 Punkte

Angenommen, das Programm *calc* liegt in übersetzter Form vor mit allen Objektdaten. Dann ändern Sie jeweils genau eine der Quelldateien (linke Spalte der folgenden Tabelle). Für welche Quelldateien wird dann der C-Übersetzer mit der Option „-c“ bei Ihrem *Makefile* aufgerufen?

| Veränderte Datei | Neu mit „gcc -c“ zu übersetzen = nur .c Dateien |
|------------------|---|
| <i>calc.c</i> | nur <i>calc.c</i> |
| <i>tree.h</i> | alle .c |
| <i>eval.h</i> | <i>calc.c</i> <i>eval.c</i> |
| <i>eval.c</i> | |

Aufgabe 5

(8 Punkte)

(a) 4 Punkte

Folgendes Programm sei gegeben:

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

```

int* create_array(unsigned int len, int start, int incr) {
    int* ip = malloc(sizeof(int) * len);
    if (ip) {
        int val = start;
        for (int i = 0; i < len; ++i, val += incr) {
            ip[i] = val;
        }
    }
    return ip;
}

```

```

void print_array(int* ip, unsigned int len) {
    for (int i = 0; i < len; ++i) printf("%d", ip[i]);
    printf("\n");
}

```

```

void add_array(int* a, int* b, unsigned int len) {
    for (int i = 0; i < len; ++i) a[i] += b[i];
}

```

```

int main() {
    int* ip1 = create_array(6, 1, 1);
    int* ip2 = create_array(3, 10, 10);
    if (ip1 && ip2) {
        print_array(ip1 + 3, 3);
        add_array(ip1 + 3, ip2, 3); print_array(ip1 + 3, 3);
        add_array(ip2 + 1, ip2, 2); print_array(ip2, 3);
    }
}

```

das 3.
überlappt sich !

} nur das
wird sich
ändern

Welche Ausgabe erzeugt dieses Programm, wenn *malloc* jeweils erfolgreich war?**Lösung:**

(b) 4 Punkte

Folgendes Programm sei gegeben:

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

```
int obscure(char* s1, char* s2) {
    char* s1end = s1;
    while (*s1end) {
        ++s1end;
    }
    if (s2 >= s1 && s2 < s1end) {
        return s2 - s1;
    }
    return -1;
}
```

```
int main() {
    char hi1[] = "Hello!";
    char hi2[] = "Hello!";
    char* hi3 = hi1;
    char* hi4 = hi1 + 2;

    printf("%d\n", obscure(hi1, hi2));
    printf("%d\n", obscure(hi1, hi3));
    printf("%d\n", obscure(hi1, hi4));
    printf("%d\n", obscure(hi4, hi1));
}
```

Hier gibts unterschiedliche
Varianten aber hat
immer mit Strings zu
tun

-1
0
2
-1

Welche Ausgabe erzeugt dieses Programm?

Lösung:

Aufgabe 6**(11 Punkte)** Dateisystem

(a) 1 Punkte

Ist es möglich, mittels eines *fstat*-Systemaufrufes an den Namen einer Datei zu gelangen?**Lösung:**

(b) 2 Punkte

Nennen Sie mindestens zwei mögliche Ursachen für das Fehlschlagen eines *stat*-Systemaufrufes.**Lösung:**

(c) 8 Punkte

Betrachten Sie die vier folgenden Befehlssequenzen. Gehen Sie davon aus, dass jede dieser Sequenzen in einem Verzeichnis ausgeführt wird, das zu Beginn leer ist und auf das Sie Lese-, Schreib- und Ausführungsrechte besitzen.

```
# Test 1
echo Hallo >a
ln a b
echo Huhu >>a
rm a
cat b
```

```
# Test 2
echo Hallo >a
ln -s a b
echo Huhu >>a
rm a
cat b
```

```
# Test 3
echo foo >a
echo bar >b
ln a b
cat b
```

```
# Test 4
echo foo >a
ln -s a b
mv a c
cat b
```

Bitte geben Sie für jeden der vier Fälle an, ob es zu Fehlermeldungen kommt und falls ja, durch welches Kommando. Zudem ist die Ausgabe des jeweils abschließenden *cat*-Kommandos anzugeben (unabhängig davon, ob es zuvor Fehler gab oder nicht).

Kommt in der Art drin

selbst wenn es einen
Fehler gab

| Fall | Ok | Fehler | Fehlschlagendes Kommando? | Ausgabe von <i>cat</i> |
|--------|--------------------------|--------------------------|---------------------------|------------------------|
| Test 1 | <input type="checkbox"/> | <input type="checkbox"/> | | |
| Test 2 | <input type="checkbox"/> | <input type="checkbox"/> | | |
| Test 3 | <input type="checkbox"/> | <input type="checkbox"/> | | |
| Test 4 | <input type="checkbox"/> | <input type="checkbox"/> | | |

Aufgabe 7**(14 Punkte)** Ein- und Ausgabe

Schreiben Sie ein C-Programm namens *overwrite*, das einen gegebenen Dateibereich in einer Datei mit einer Zeichenkette überschreibt. Die hierzu erforderlichen Parameter wie die Zielfdatei, die Startposition und die Zeichenkette werden als Kommandozeilenparameter übergeben. Beachten Sie, dass für die Startposition ganzzahlige 64-Bit-Werte zu unterstützen sind.

Alle möglichen Fehler sind abzufangen. Im Falle eines Fehlers ist die Funktion *die()* aufzurufen, die Sie in Ihrer Lösung nicht mit implementieren müssen.

Sie dürfen die *stdio* hierbei nicht verwenden. Stattdessen sind nur Systemaufrufe zulässig. Grundsätzlich ist bei allen Einlese- und Schreiboperationen damit zu rechnen, dass weniger Bytes gelesen oder geschrieben werden als gewünscht. In diesem Falle darf *die()* nicht aufgerufen werden. Die Ein- und Ausgabe muss effizient erfolgen, d.h. es sollten nicht unnötig viele Systemaufrufe verwendet werden. Insbesondere ist ein zeichenweises Lesen oder Schreiben nicht zulässig.

Beispiel:

```
theon$ gcc -o overwrite overwrite.c
theon$ echo aaabbbccc >a
theon$ overwrite a 4 'xxx'
theon$ cat a
aaabxxxcc
theon$ overwrite a 4294967296 'text'
theon$ ls -lh a
-rw-rw-r-- 1 borchert sai 4.0G Feb  2 14:29 a
theon$ tail -4b a; echo
text
theon$
```

man muss read oder write aufrufen
umgehen

Buffer muss angelegt werden

Hinweis: Es steht Ihnen frei, wie Sie das zweite Argument konvertieren. Hilfreich ist ggf. folgende Standardfunktion:

```
long long strtoll(const char* s, char** endptr, int base);
```

Hierbei zeigt *s* auf die zu konvertierende Zeichenkette, *endptr* zeigt auf einen Endzeiger, der auf die Position hinter *s* gesetzt wird, die nicht mehr für die Konvertierung berücksichtigt wurde, und *base* ist die zu verwendende Basis, wobei bei 0 alle in C üblichen Darstellungen (dezimal, oktal oder hexadezimal) akzeptiert werden. Das konvertierte Resultat wird zurückgeliefert, der Endzeiger sollte im Erfolgsfall nach dem Aufruf auf das terminierende Nullbyte in *s* verweisen.

Lösung:

sollte nicht viel Schreibarbeit sein
korrekte Datentypen! size_t off_t ...

Einfache Dynamische Datenstruktur garantiert

20

Aufgabe 8

(14 Punkte) Dynamische Datenstrukturen

Gegeben sei folgende einfach linear verkettete Datenstruktur für eine Prioritäts-Warteschlange (*PQueue*). Allen Elemente dieser Queue wird ein Schlüssel mitgegeben, der die Reihenfolge der Abarbeitung und damit die Reihenfolge der Elemente bestimmt. Die lineare Liste wird sortiert unterhalten mit dem Element mit der höchsten Priorität (kleinstem Schlüsselwert) zu Beginn. Zu der Datenstruktur gehören die beiden Datentypen *PQueue* und *PQElement*, wobei *PQueue* die gesamte Warteschlange repräsentiert und *PQElement* ein einzelnes Mitglied:

```
typedef struct pquelement {  
    int prio; // Schlüssel  
    int info;  
    struct pquelement* next;  
} PQElement;
```

```
typedef struct {  
    PQElement* head;  
} PQueue;
```

(a) 3 Punkte

Schreiben Sie eine Funktion *pq_len*, das als Parameter einen Zeiger auf eine Warteschlange erhält und die Länge einer Warteschlange als nicht-negative ganze Zahl zurückliefert.

Lösung:

(b) 3 Punkte

Schreiben Sie eine Funktion *pqe_create*, das zwei Parameter des Typs **int** erhält, die die Priorität und die Information repräsentieren, und ein neues Warteschlangenelement erzeugt und einen Zeiger darauf zurückliefert. Falls nicht genügend Speicher belegt werden kann, ist der Nullzeiger zurückzugeben.

Lösung:

(c) 4 Punkte

Schreiben Sie eine Funktion *pq_insert*, die als Parameter eine Warteschlange und zwei ganze Zahlen erhält, die die Priorität und die Information repräsentieren. Die Funktion soll dann ein entsprechendes Warteschlangenelement erzeugen (siehe *pqe_create* aus der vorherigen Teilaufgabe) und in die Warteschlange entsprechend seiner Priorität einfügen, d.h. der Zeiger *head* in der *PQueue* verweist auf das Element mit dem kleinsten *prio*-Wert und für jedes Element *pqe* gilt, dass $pqe \rightarrow prio \leq pqe \rightarrow next \rightarrow prio$, falls *pqe* → *next* nicht Null ist. Wenn es bereits Elemente mit der gleichen Priorität geben sollte, dann ist das neue Element dahinter einzufügen. Die Funktion liefert **true** zurück, wenn alles geklappt hat und ansonsten **false**.

Lösung:

(d) 4 Punkte

Schreiben Sie eine Funktion *pq_remove*, die als Parameter einen Zeiger auf eine Warteschlange erhält, das Element mit der höchsten Priorität (= niedrigster Schlüssel) aus der Warteschlange entnimmt und die Information als ganze Zahl zurückliefert. Hierbei ist der nicht mehr benötigte Speicherplatz freizugeben. Falls kein Element mehr in der Warteschlange war, ist 0 zurückzugeben.

Lösung:

