ulm university universität
uulm

Klausurdeckblatt

Name der Prüfung:	Systemnahe Software I		
Datum und Uhrzeit:	20. Februar 2017, 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)

Nr	Max	Bewertung	
1	**20**	xxxxx	
(a)	4		xxxxx
(b)	3		xxxxx
(c)	4		xxxxx
(d)	2		xxxxx
(e)	4		xxxxx
(f)	3		xxxxx
2	**15**	xxxxx	
(a)	3		xxxxx
(b)	4		xxxxx
(c)	2		xxxxx
(d)	2		xxxxx
(e)	4		xxxxx
3	**6**	xxxxx	
4	**12**	xxxxx	
(a)	8		xxxxx
(b)	4		xxxxx

Nr	Max	Bewertung	
5	**8**	xxxxx	
(a)	4		xxxxx
(b)	4		xxxxx
6	**11**	xxxxx	
(a)	1		xxxxx
(b)	2		xxxxx
(c)	4		xxxxx
(d)	4		xxxxx
7	**14**	xxxxx	
8	**14**	xxxxx	
(a)	3		xxxxx
(b)	3		xxxxx
(c)	4		xxxxx
(d)	4		xxxxx
Summe	**100**		

- Prüfen Sie zu Beginn, ob Ihre Klausur ab der Aufgabe 1 aus 18 durchnumerierten Seiten besteht.
- Für Ihre Lösungen verwenden Sie bitte den freigelassenen Platz nach der Aufgabenstellung, die Rückseite 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 scheuen Sie sich bitte nicht, jemanden von der Aufsicht zu befragen.
- Wenn wir während der Klausur feststellen, dass eine Aufgabenstellung missverständlich ist, werden wir an der Tafel einen klärenden Hinweis für alle sichtbar hinschreiben.

Aufgabe 1 **(20 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
(h) 0777 & ~027

(a) 14 / 12 → 1
(b) 13 % 8 → 5
(c) 15.0 / 6 → 2.5
(d) 11 % 6 ∗ 12 → 60
(e) 10 ∗ 11 % 5 → 0
(f) 7 << 3 → 56
(g) 3 ^ 14 → 13
(h) 0777 & ~027 → 0750

(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

(a) $x > y$!= 0 → 0
(b) $--x$ / y? x: y → 6
(c) y /= x, y ∗= x → 6

(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!

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

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

Es dürfen keine Zeiger auf lokale Variablen zurückgegeben werden. Stattdessen ist eine neue Datenstruktur mit *malloc*() anzulegen:

```
Complex* create_complex(double real, double img) {
    Complex* c = malloc(sizeof(Complex));
    if (!c) return 0;
    c->real = real;
    c->img = img;
    return c;
}
```

(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?

4 3 2 1

(e) 4 Punkte
Schreiben Sie eine Funktion, die zwei ganze Zahlen n und m des Typs **unsigned int** erhält und das ebenfalls ganzzahlige nicht-negative Ergebnis n^m zurückliefert. Die Problematik eines Überlaufs darf ignoriert werden.

```
unsigned int potenz(unsigned int n, unsigned int m) {
int potenz(int x, int n) {
    int result = 1;
    while (m > 0) {
        result *= n;
        --m;
    }
    return result;
}
```

(f) 3 Punkte

Bitte kreuzen Sie bei den folgenden Behauptungen zu den Datentypen **float** und **double** an, ob sie zutreffen oder inkorrekt sind:

Behauptung	trifft zu	ist falsch
0.2 ist präzise darstellbar	☐	☐
0.25 ist präzise darstellbar	☐	☐
$2^{31} - 1 = 2147483647$ ist präzise als **float** darstellbar	☐	☐
$-\infty$ ist darstellbar	☐	☐
Alle ganzzahligen Werte des Typs **float** sind als Werte des Typs **long** darstellbar	☐	☐
Es gilt immer $(a + b) + c == (b + c) + a$ für alle möglichen Werte für a, b und c	☐	☐

- nein, 0.2 ist nicht präzise darstellbar
- ja, 0.25 ist präzise darstellbar
- nein, 2147483647 ist nicht präzise darstellbar
- ja, $-\infty$ ist darstellbar (siehe IEEE-754)
- nein, Gegenbeispiel: 2^{64}
- nein, Gegenbeispiel: $a = 2^{128}, b = -2^{128}, c = 1$, wobei die erste Variante 1 liefert und die zweite 0, da es im zweiten Fall zu einer Auslöschung kommt.

Aufgabe 2 **(15 Punkte)** Funktionen und Strukturen

(a) 3 Punkte
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.

C unterstützt nur *call by value*, aber *call by reference* ist möglich mit Hilfe von Zeigern:

```
void swap(int* ap, int *bp) {
    int tmp = *ap; *ap = *bp; *bp = tmp;
}
```

Ein Beispiel in der Standardbibliothek ist *scanf*.

(b) 4 Punkte
Deklarieren Sie folgende Typen:

- Eine Funktion namens *f*, die einen Zeiger auf **int** zurückliefert,
- einen Typ namens *g*, der einen Zeiger auf eine Funktion darstellt, die ein **int** zurückliefert.
- ein Array namens *s* mit 10 Elementen des Typs Zeiger auf **char** und
- einen Zeiger namens *ap* auf ein Array mit 10 Elementen des Typs **char**.

- **typedef int*** *f*();
- **typedef int** (*g)();
- **typedef char** *s*[10];
- **typedef char** (*ap)[10];

(c) 2 Punkte
Spezifizieren Sie eine Datenstruktur *Adresse*, die ein Quadrupel repräsentiert, bestehend aus einem Vornamen, einem Nachnamen, einer Adresse und einer ganzzahligen Postleitzahl. Die ersten drei Felder sollen Zeichenketten beliebiger Länge sein.

```
struct Adresse {
    char* Vorname;
    char* Nachname;
    char* Adresse;
    unsigned int PLZ;
};
```

(d) 2 Punkte
Beschreiben Sie, was mit folgender Deklaration deklariert wird:

typedef int *int_array*[100];

int_array wird durch die Deklaration zu einem Typnamen, der für ein Array mit 100 Elementen des Typs **int** steht.

(e) 4 Punkte

Die Male-Female-Folgen von Hofstadter sind folgendermaßen definiert:

$$\begin{aligned} F(0) &= 1 \\ M(0) &= 0 \\ F(n) &= n - M(F(n-1)) \\ M(n) &= n - F(M(n-1)) \end{aligned}$$

Schreiben Sie in C die rekursiven Funktionen f und m, die $F(n)$ bzw. $M(n)$ für nichtnegative ganze Zahlen n berechnen. Vergessen Sie bitte nicht, die notwendige Vorab-Deklaration einer der beiden Prozeduren zu berücksichtigen.

```
unsigned int m(unsigned int n);

unsigned int f(unsigned int n) {
    if (n == 0) {
        return 1;
    } else {
        return n − m(f(n−1));
    }
}

unsigned int m(unsigned int n) {
    if (n == 0) {
        return 0;
    } else {
        return n − f(m(n−1));
    }
}
```

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
```

Lösungsvorschlag:

```
Anfang
Anfang A
C A
A B C D E F
B B
B C
B B
```

C B
C C
C B
A B C B A C C
B C
C C
A B C B A C C
C

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) {
    // ...
}
```

(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.

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

(b) 4 Punkte

Was muss neu übersetzt werden, wenn die jeweils angegebene Datei verändert wird?

Veränderte Datei	Neu zu übersetzen
calc.c	
tree.h	
eval.h	
eval.c	

Veränderte Datei	Neu zu übersetzen
calc.c	calc.c
tree.h	calc.c reader.c eval.c
eval.h	calc.c eval.c
eval.c	eval.c

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);
    }
}
```

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

```
 4  5  6
14 25 36
10 30 60
```

(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));
}
```

Welche Ausgabe erzeugt dieses Programm?

```
-1
0
2
-1
```

Aufgabe 6 **(11 Punkte)** Dateisystem

(a) 1 Punkte
Ist es möglich, mittels eines *fstat*-Systemaufrufes an den Namen einer Datei zu gelangen?

Nein, der Dateiname ist nicht Teil der Inode, die von *fstat* ausgelesen wird. Stattdessen ist ein Dateiname nur den Verzeichnissen zu entnehmen.

(b) 2 Punkte
Nennen Sie mindestens zwei mögliche Ursachen für das Fehlschlagen eines *stat*-Systemaufrufes.

- keine ausreichenden Leserechte
- Datei exisitiert nicht

(c) 4 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, sofern es erfolgreich ausgeführt wird.

Fall	Ok	Fehler	Fehlschlagendes Kommando?	Ausgabe von *cat*
Test 1	☐	☐		
Test 2	☐	☐		
Test 3	☐	☐		
Test 4	☐	☐		

(d) 4 Punkte

Bitte kreuzen Sie bei den folgenden Behauptungen zu einem POSIX-Dateisystem, ob sie zutreffen oder inkorrekt sind:

Behauptung	trifft zu	ist falsch
Die Zahl der Dateinamen gehört zu einer Inode	☐	☐
Das Datum des letzten Lesezugriffs gehört zur Inode	☐	☐
Eine Datei kann mehrere Gruppen gehören	☐	☐
Eine Datei mit den Zugriffsrechten `rw-r--r--` ist für Gruppenmitglieder, die nicht der Besitzer sind, schreibbar	☐	☐

- Ja
- Ja
- Nein (Zugriffsrechte für mehrere Gruppen wären über ACLs denkbar)
- Nein

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 Zieldatei, die Startposition und die Zeichenkette werden als Kommandozeilenparameter übergeben.
Alle möglichen Fehler sind abzufangen. Im Falle eines Fehlers ist die Funktion *die()* aufzurufen, die Sie in Ihrer Lösung nicht mit implementieren müssen.
Grundsätzlich ist bei allen Einlese- und Schreiboperationen damit zu rechnen, dass weniger Bytes gelesen oder geschrieben werden als gewünscht.
Beispiel:

```
clonard$ gcc -o overwrite overwrite.c
clonard$ echo aaabbbccc >a
clonard$ overwrite a 4 'xxx'
clonard$ cat a
aaabxxxcc
clonard$
```

Hinweise: Sie können auf die Angabe der **#include**-Anweisungen verzichten. Für die Konvertierung von Zeichenketten in ganzzahlige Werte empfiehlt sich die Funktion
int *atoi*(**const char*** *s*). Sie müssen nicht überprüfen, ob der zweite Kommandozeilenparameter tatsächlich eine ganze Zahl ist, sondern nur dass die Resultate von *atoi* nicht-negativ sind.

```
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>

void die() {
    exit(1);
}

int main(int argc, char** argv) {
    --argc; ++argv; // skip command name
    if (argc != 3) die();
    char* filename = argv[0];
    off_t pos = atoi(argv[1]); if (pos < 0) die();
    char* s = argv[2];
    ssize_t len = strlen(s);
    if (len == 0) exit(0);
    int fd = open(filename, O_RDWR);
    if (fd < 0) die();
    if (lseek(fd, pos, SEEK_SET) < 0) die();
    ssize_t written = 0;
```

```
    while (written < len) {
        ssize_t nbytes = write(fd, s + written, len − written);
        if (nbytes < 0) die();
        written += nbytes;
    }
}
```

Aufgabe 8 **(14 Punkte)** Dynamische Datenstrukturen

Gegeben sei folgende Datenstruktur für eine Prioritäts-Warteschlange (*PQueue*). Alle Elemente dieser Queue wird ein Schlüssel mitgegeben, der die Reihenfolge der Abarbeitung und damit die Reihenfolge der Elemente bestimmt. Das Element mit der höchsten Priorität (kleinstem Schlüssel) sitzt ganz vorne in der Schlange und ist für die nächste Bearbeitung bestimmt.
Zu der Datenstruktur gehören die beiden Datentypen *PQueue* und *PQElement*, wobei *PQueue* die gesamte Warteschlange repräsentiert und *PQElement* ein einzelnes Mitglied:

```
typedef struct pqelement {
    int prio; // Schluessel
    int info;
    struct pqelement* 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.

```
unsigned int pq_len(PQueue* pq) {
    unsigned int len = 0;
    PQElement* p = pq->head;
    while (p) {
        ++len;
        p = p->next;
    }
    return len;
}
```

(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.

```
PQElement* pqe_create(int prio, int info) {
    PQElement* pqe = malloc(sizeof(PQElement));
    if (pqe) {
        pqe->prio = prio;
        pqe->info = info;
        pqe->next = 0;
```

```
    }
    return pqe;
}
```

(c) 4 Punkte
Schreiben Sie eine Funktion *pq_insert*, das 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->prio <= pqe->next->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**.

```
bool pq_insert(PQueue* pq, int prio, int info) {
    pqe = pqe_create(prio, info);
    if (!pqe) return false;
    PQElement** prev = &pq->head;
    PQElement* p = pq->head;
    while (p && p->prio < prio) {
        prev = &p->next;
        p = p->next;
    }
    pqe->next = p;
    *prev = pqe;
    return true;
}
```

(d) 4 Punkte
Schreiben Sie eine Funktion *pq_remove*, das 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.

```
int pq_remove(PQueue* pq) {
    PQElement* pqe = pq->head;
    if (!pqe) return 0;
    int info = pqe->info;
    pq->head = pqe->next;
    free(pqe);
    return info;
}
```