

Name: _____

Matrikelnummer: _____

Unterschrift: _____

Leibniz Universität Hannover
Fachgebiet Mensch-Computer-Interaktion

Programmieren 1
Dozent: Prof. Dr. Michael Rohs

Testklausur
20. Mai 2020

Tragen Sie bitte Ihren Namen und Ihre Matrikelnummer auf diesem Blatt ein und unterschreiben Sie es. Legen Sie bitte Ihren Studierendenausweis und Ihren Personalausweis zur Anwesenheitskontrolle auf den Tisch rechts von Ihnen. Es sind keine zusätzlichen Hilfsmittel erlaubt. Schalten Sie bitte Ihr Mobiltelefon aus und legen Sie es weg.

Dies ist eine **90**-minütige Klausur. Dieses Heft sollte aus **10** Seiten mit insgesamt vier Aufgaben bestehen. Bitte überprüfen Sie Ihre Klausur auf Vollständigkeit.

Wenn Sie **keinen Bonus** erlangt haben, müssen Sie von den vier gestellten Aufgaben **drei Aufgaben** erfolgreich lösen. **Mit Bonus** müssen Sie nur **zwei** der vier Aufgaben erfolgreich lösen.

Jede Aufgabe besteht aus einem Programmierteil (a) und einem Wissensteil (b). Eine Aufgabe gilt als gelöst, wenn 4 von 5 Punkten erreicht wurden. In (a) können maximal 5 Punkte und in (b) ein Zusatzpunkt erreicht werden. (Eine Aufgabe kann also ohne (b) mit 4-5 Punkten in (a) gelöst werden. Sie kann auch mit 3-5 Punkten in (a) plus Zusatzpunkt in (b) gelöst werden.)

Lesen Sie die Aufgabenstellungen aufmerksam durch. Schreiben Sie bitte nicht ausschweifend, sondern beantworten Sie die Fragestellung präzise. Versuchen Sie klar und deutlich zu formulieren. Sie können die Wissensfragen auf Deutsch oder Englisch beantworten. Falls Sie zusätzlichen Platz benötigen, verwenden Sie bitte die unbedruckten Bereiche.

(b) (1 Punkt) Wie viele Bytes im Speicher belegt die Zeichenkette "hello" in C?

2. Einfach verkettete Listen

- (a) (5 Punkte) Implementieren Sie die Funktion `pos_neg_list`. Der Parameter `list` ist eine Eingabeliste ganzer Zahlen. Die Parameter `pos` und `neg` sind initial leere Ausgabelisten für die positiven bzw. negativen Zahlen aus der Eingabeliste. Die Eingabeliste wird also quasi zwei Listen aufgeteilt. Nullen werden ignoriert. Die Reihenfolge in den Ausgabelisten muss die gleiche wie in der Eingabeliste sein. Die Eingabeliste darf nicht verändert werden. Die Lösung darf nur die Funktion `new_node` und `reverse` verwenden.

```
#include "base.h"

typedef struct Node {
    int value;
    struct Node* next;
} Node;

Node* new_node(int value, Node* next) {
    Node* node = xmalloc(1, sizeof(Node));
    node->value = value;
    node->next = next;
    return node;
}

Node* reverse_rec(Node* list, Node* result) {
    if (list == NULL) {
        return result;
    } else {
        return reverse_rec(list->next, new_node(list->value, result));
    }
}

Node* reverse(Node* list) {
    return reverse_rec(list, NULL);
}

void test(void) {
    Node *list, *pos, *neg;

    list = new_node(0, new_node(1, new_node(-1,
        new_node(34, new_node(-56, NULL)))));
    pos = NULL;
    neg = NULL;
    pos_neg_list(list, &pos, &neg);
    // pos: [34, 1] (Reihenfolge beliebig)
    // neg: [-56, -1] (Reihenfolge beliebig)

    list = new_node(0, NULL);
    pos = NULL;
    neg = NULL;
    pos_neg_list(list, &pos, &neg);
    // pos: []
    // neg: []
}
```

```
void pos_neg_list(Node* list, Node** pos, Node** neg) {
```

```
}
```

```
int main(void) {  
    test();  
    return 0;  
}
```

(b) (1 Punkt) Erläutern Sie kurz, wieso folgender C-Code problematisch ist.

```
1 char* a = NULL;  
2 char b = *a;  
3 printf("%c\n", b);
```

```
void test(void) {
    Tree* t = node(leaf(1), 2, leaf(3));
    int x = internal_node_sum(t); // 2

    t = leaf(100);
    x = internal_node_sum(t); // 0

    // -101
    // 2 5
    // 1 3 4 NULL
    t = node(node(leaf(1), 2, leaf(3)), -101, node(leaf(4), 5, NULL));
    x = internal_node_sum(t); // -101 + 2 + 5

    //
    //      1
    //      2      3
    //      15      NULL
    // NULL 85
    t = node(node(node(NULL, 15, leaf(85)), 2, NULL), 1, leaf(3));
    x = internal_node_sum(t); // 1 + 2 + 15
}

int main(void) {
    test();
    return 0;
}
```

(b) (1 Punkt) Geben Sie ein Anwendungsbeispiel für Funktionszeiger in C.

4. Zweidimensionales Array

- (a) (5 Punkte) Implementieren Sie die Funktion `void matrix_max(int* a, int rows, int cols, int* mval, int* mrow, int* mcol)`. Diese hat als Eingabeparameter ein 2-dimensionales `int`-Array `a` mit `rows` Zeilen und `cols` Spalten. (C speichert zweidimensionale Arrays zeilenweise.) Die Funktion setzt die Ausgabeparameter `mval`, `mrow` und `mcol` auf den Wert und die Position eines Elements im Eingabearray mit maximalem Wert. Sollte `a` leer sein, wird keiner der Ausgabeparameter gesetzt. In `matrix_max` dürfen keine Funktionen aufgerufen werden.

```
#include "base.h"
```

```
void matrix_max(int* a, int rows, int cols, int* mval, int* mrow, int* mcol) {
```

[illegible]

```
void test(void) {
    int m, r, c;
    int a[3][3] = {
        { 1, -1, 3 },
        { 2, -2, 5 },
        { 3, -3, 4 },
    };
    matrix_max((int*)a, 3, 3, &m, &r, &c);
    // m: 5, r: 1, c: 2
}
```

```
int b[2][3] = {
    { 1, -1, 3 },
    { 7, -2, 5 },
};
matrix_max((int*)b, 2, 3, &m, &r, &c);
// m: 7, r: 1, c: 0
```

```
int b[2][3] = {
```



```
        { 1, -1, 3 },
        { 7, -2, 7 },
    };
    matrix_max((int*)b, 2, 3, &m, &r, &c);
    // m: 7, r: 1, c: 0
    // oder
    // m: 7, r: 1, c: 2
}

int main(void) {
    test();
    return 0;
}
```

- (b) (1 Punkt) Erläutern Sie kurz, welches Problem mit der case-Anweisung in C verbunden ist.
