Probeklausur

Fortgeschrittene Algorithmen und Programmierung

Wintersemester 2025/26 | HTW Berlin

Prof. Dr. Alexandra Mikityuk

📋 Prüfungsinformationen

- **Bearbeitungszeit:** 90 Minuten
- **Gesamtpunktzahl:** 100 Punkte
- **Hilfsmittel erlaubt:**
  - ✅ Vorlesungsunterlagen (gedruckt oder digital)
  - ✅ Eigene Notizen
  - ✅ Laptop, Tablet oder andere technische Geräte
- **Hilfsmittel NICHT erlaubt:**
  - ❌ Keine KI-Tools (ChatGPT, Claude, Copilot, etc.)
  - ❌ Keine Google-Suche oder andere Suchmaschinen
  - ❌ Keine Kommunikation mit anderen Personen
- **Hinweis:** Schreiben Sie leserlich und strukturiert. Unlesbare Antworten können nicht bewertet werden.
- **Code-Aufgaben:** Geben Sie vollständigen, lauffähigen Code an (inkl. `#include`, `main()`, etc.)

> ⚠️ ***Wichtig:*** *Bitte tragen Sie auf jeder Seite Ihren Namen und Ihre Matrikelnummer ein!*

Teil 1: Theorie und Verständnis (30 Punkte)

Aufgabe 1.1: Type Casting und implizite Konvertierung

10 Punkte

a) Erklären Sie den Unterschied zwischen **impliziter** und **expliziter** Typkonvertierung in C. Geben Sie je ein Beispiel. (5 Punkte)

b) Was ist das Ergebnis folgender Ausdrücke? Begründen Sie Ihre Antwort. (5 Punkte)

```
int a = 7, b = 2;
double x = 3.5;

1. a / b         = ?
2. (double)a / b = ?
3. a + x         = ? (Datentyp?)
4. (int)x * 2    = ?
```

Aufgabe 1.2: Konstanten und ihre Verwendung

8 Punkte

a) Nennen Sie **drei Vorteile** der Verwendung von `#define` Konstanten gegenüber "magischen Zahlen" im Code. (3 Punkte)

b) Schreiben Sie ein Code-Beispiel, das zeigt, wie man eine Konstante für die Mehrwertsteuer (19%) definiert und verwendet. (5 Punkte)

Aufgabe 1.3: Schleifen-Typen

12 Punkte

Vervollständigen Sie die folgende Tabelle:

Schleifentyp	Wann verwenden?	Mindestanzahl Durchläufe	Bedingung geprüft
`for`	_____	_____	_____
`while`	_____	_____	_____
`do-while`	_____	_____	_____

Zusatzfrage: Erklären Sie den Unterschied zwischen `break` und `continue`. (3 Punkte)

Teil 2: Code-Analyse und Fehlersuche (30 Punkte)

Aufgabe 2.1: Schleifen-Analyse mit break

10 Punkte

Analysieren Sie folgenden Code:

```
#include <stdio.h>

int main()
{
    int summe = 0;

    for(int i = 1; i <= 10; i++)
    {
        if(i % 3 == 0)
        {
            continue;
        }

        summe += i;

        if(summe > 15)
        {
            break;
        }
    }

    printf("Summe: %d\n", summe);
    return 0;
}
```

a) Zeigen Sie die Werte von `i` und `summe` nach jedem relevanten Schleifendurchlauf: (6 Punkte)

b) Was ist die finale Ausgabe des Programms? (2 Punkte)

c) Erklären Sie, warum bestimmte Zahlen übersprungen werden. (2 Punkte)

Aufgabe 2.2: Debugging - Finden Sie ALLE Fehler

12 Punkte

Der folgende Code soll die durchschnittliche Temperatur von 3 Tagen berechnen und prüfen, ob es warm genug zum Schwimmen ist (> 20°C). Er enthält aber mehrere Fehler:

```
#include <studio.h>

#define SCHWIMM_TEMP 20

int main()
{
    int temp1, temp2, temp3;
    double durchschnitt;

    printf("Temperaturen für 3 Tage:\n");
    scanf("%d %d %d", &temp1, &temp2, &temp3);

    durchschnitt = (temp1 + temp2 + temp3) / 3;

    printf("Durchschnitt: %.2lf°C\n", durchschnitt);

    if(durchschnitt > SCHWIMM_TEMP)
        printf("Perfekt zum Schwimmen!\n");
        printf("Badesachen einpacken!\n");
    else
        printf("Zu kalt zum Schwimmen.\n");

    return 0;
}
```

Listen Sie **alle Fehler** auf und erklären Sie, warum sie falsch sind:

Aufgabe 2.3: do-while vs. while

8 Punkte

Gegeben ist folgender Code mit `do-while`:

```
int x = 10;

do
{
    printf("%d ", x);
    x = x - 3;
} while(x > 0);
```

a) Was ist die Ausgabe? (2 Punkte)

b) Was wäre die Ausgabe, wenn man stattdessen eine `while`-Schleife verwendet? (3 Punkte)

c) Wann ist `do-while` besser als `while`? Geben Sie ein praktisches Beispiel. (3 Punkte)

Teil 3: Praktische Programmierung (40 Punkte)

Aufgabe 3.1: Eingabevalidierung mit do-while

12 Punkte

Schreiben Sie ein **vollständiges C-Programm**, das:

- Eine Zahl zwischen 1 und 100 vom Benutzer einliest
- Die Eingabe **validiert**: Bei ungültiger Eingabe (< 1 oder > 100) wird eine Fehlermeldung ausgegeben
- Die Eingabe **wiederholt**, bis eine gültige Zahl eingegeben wurde (verwenden Sie `do-while`!)
- Am Ende die gültige Zahl ausgibt

> ***Beispiel-Ablauf:***
> *Zahl (1-100): 150*
> *Ungültig! Zahl (1-100): -5*
> *Ungültig! Zahl (1-100): 42*
> *Gültige Zahl: 42*

Aufgabe 3.2: Zähler mit for-Schleife und continue

14 Punkte

Schreiben Sie ein **vollständiges C-Programm**, das:

- Alle Zahlen von 1 bis 50 durchläuft (for-Schleife)
- Zahlen, die durch 5 teilbar sind, **überspringt** (verwenden Sie `continue`!)
- Von den verbleibenden Zahlen nur die **geraden Zahlen** ausgibt
- Am Ende die **Anzahl** der ausgegebenen Zahlen anzeigt

> ***Hinweis:*** *Eine Zahl ist durch 5 teilbar, wenn* `zahl % 5 == 0`
> *Eine Zahl ist gerade, wenn* `zahl % 2 == 0`

Aufgabe 3.3: Menü-Programm mit Endlosschleife und break

14 Punkte

Schreiben Sie ein **vollständiges C-Programm** für einen einfachen Taschenrechner mit Menü:

- Verwenden Sie eine **Endlosschleife** (`while(1)`), um das Menü anzuzeigen
- Das Menü bietet folgende Optionen:
  - 1: Zwei Zahlen addieren
  - 2: Zwei Zahlen multiplizieren
  - 0: Programm beenden
- Bei Auswahl 1 oder 2: Zwei `double`-Zahlen einlesen, berechnen, Ergebnis mit 2 Nachkommastellen ausgeben
- Bei Auswahl 0: Abschiedsmeldung ausgeben und Schleife mit `break` verlassen
- Bei ungültiger Auswahl: Fehlermeldung ausgeben

> ***Beispiel-Ablauf:***
> *=== RECHNER ===*
> *1. Addition*
> *2. Multiplikation*
> *0. Beenden*
> *Wahl: 1*
> *Zahl 1: 5.5*
> *Zahl 2: 3.2*
> *Ergebnis: 8.70*
>
> *=== RECHNER ===*
> *...*
> *Wahl: 0*
> *Auf Wiedersehen!*

Gesamtpunktzahl: _____ / 100 Punkte

Viel Erfolg! 🍀

Vergessen Sie nicht, Ihre Lösungen zu überprüfen und auf Vollständigkeit zu achten.