# Systemprogrammierung – Probeklausur

*Die Klausur besteht aus 7 Aufgaben, die zusammen 100 Punkte ergeben.*

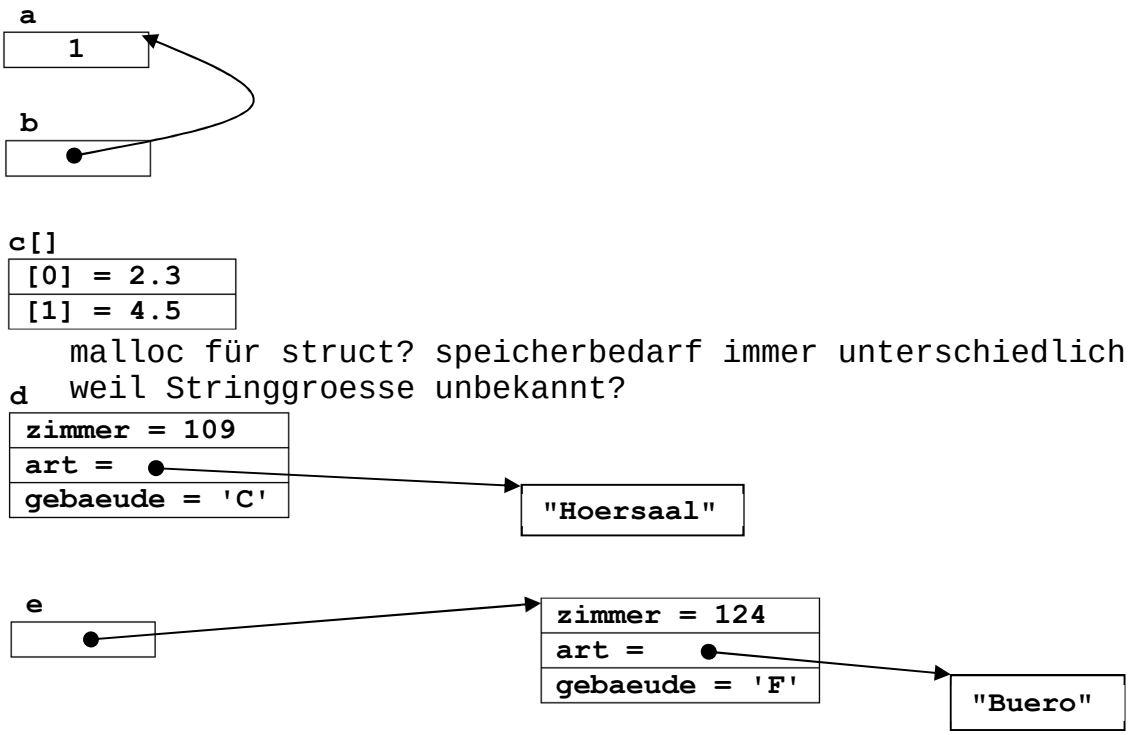
## **Aufgabe 1:** C Datentypen und Variablen *(26 Punkte)*

*In der Vorlesung haben wir Speicherbelegungen mit Rechtecken grafisch dargestellt:*

- *Wurde der Speicher per Variablendefinition reserviert, stand über dem Rechteck der Variablenname, bei Feldern zusätzlich* `[]`.
- *Bei Speicher, der nicht per Variablendefinition reserviert wurde, fehlte der Name über dem Rechteck, z.B. bei Stringliteralen und dynamisch allokiertem Heapspeicher.*
- *Bei Feldern und Strukturen wurde das Rechteck unterteilt und jeweils der Index des Feldelements bzw. der Name der Strukturkomponente angegeben*
- *Die Adresse eines Speicherbereichs haben wir als Pfeil auf den Speicherbereich gezeichnet.*

a) Erstellen Sie mit C-Definitionen und -Anweisungen die unten dargestellte Hauptspeicherbelegung:
   - deklarieren Sie zuerst eventuelle benutzerdefinierte Typen
   - definieren Sie dann die fünf Variablen ohne Initialisierung
     (leiten Sie dazu die Typen der Variablen aus den C-Literalen und den Pfeilen in der Grafik ab)
   - weisen Sie zum Schluss allen Variablen die in der Grafik gezeigten Werte zu
     (lassen Sie dabei die Fehlerbehandlung für dynamischer Speicherreservierungen weg)

   *(20 Punkte)*




b) Wieviel Speicherplatz in Anzahl Byte belegt die Variable **d** auf dem Stack?
   Geben Sie für die Architekturen ILP32 und LP64 jeweils die Mindestgröße ohne Ausrichtung sowie die tatsächliche Größe mit Ausrichtung der Datentypen an. Begründen Sie Ihre Angaben.
   *(6 Punkte)*

## **Aufgabe 2:** C Parameter *(5 Punkte)*

Erläutern Sie mit wenigen Worten den Unterschied zwischen Eingabeparametern und Ausgabeparametern. Geben Sie für beide Arten von Parametern ein Beispiel an. Das Beispiel soll jeweils aus einer einfachen Funktionsdefinition und einem Aufruf dieser Funktion bestehen.

## **Aufgabe 3:** C Übersetzungseinheiten *(31 Punkte)*

a) Erstellen Sie zwei C-Übersetzungseinheiten, die den folgenden beiden Java-Klasse **`Quadrat`** und **`Wuerfel`** entsprechen. *(17 Punkte)*

```
public final class Quadrat {
    private Quadrat() { }
    public static double flaeche(double seitenlaenge) {
        return zumquadrat(seitenlaenge);
    }
    private static double zumquadrat(final double d) {
        return d * d;
    }
}
public final class Wuerfel {
    private Wuerfel() { }
    public static double oberflaeche(double kantenlaenge) {
        return Quadrat.flaeche(kantenlaenge) * 6;
    }
    public static double volumen(double kantenlaenge) {
        return Quadrat.flaeche(kantenlaenge) * kantenlaenge;
    }
}
```

b) Die folgende Java-Klasse **`WuerfelTest`** testet die Java-Klasse **`Wuerfel`** aus a). Erstellen Sie auch hier eine entsprechende C-Übersetzungseinheit. *(11 Punkte)*

```
import java.util.Scanner;
import static java.lang.System.out;
public final class WuerfelTest {
    private WuerfelTest() { }
    public static void main(String[] args) {
        Scanner eingabe = new Scanner(args[0]);
        if (!eingabe.hasNextDouble()) {
            System.exit(1);
        }
        double k = eingabe.nextDouble();
        double f = Wuerfel.oberflaeche(k);
        double v = Wuerfel.volumen(k);
        out.printf("Kantenlaenge %f, Oberflaeche %f, Volumen %f%n",
                   k, f, v);
    }
}
```

*Hinweis: als Entsprechung zu den verwendeten Methoden der Java-Klasse **`Scanner`** gibt es bei C die Funktion **`int sscanf(const char *eingabe, const char *format, ...)`** in **`stdio.h`**, die mit dem Format **`"%lf"`** eine **`double`**-Zahl aus dem String **`eingabe`** liest. **`sscanf`** unterscheidet sich von **`scanf`** nur durch den ersten Parameter, der die Standardeingabe ersetzt.*

c) Wird das Java-Programm aus b) ohne Kommandozeilenargumente gestartet, bricht es mit einer nicht gefangenen Ausnahme **`ArrayIndexOutOfBoundsException`** ab.
Wie verhält sich das entsprechende C-Programm, wenn es wie das Java-Programm keine Fehlerbehandlung für zu wenige Kommandozeilenargumente enthält? Begründen Sie Ihre Antwort. *(3 Punkte)*

*Hinweis: überlegen Sie, was **`argc`** und **`argv`** bei einem Aufruf ohne Kommandozeilenargumente enthalten und was dann an die Funktion **`sscanf`** übergeben wird.*

## **Aufgabe 4:** C Strings *(6 Punkte)*

Schreiben Sie eine C-Anweisungsfolge, die der folgenden Java-Anweisungsfolge entspricht:

```
public static void main(String[] args) {
    String s = args[0];
    String t = args[1];
    String st = s + t;
}
```

Bei dynamischer Speicherreservierung dürfen Sie die Fehlerbehandlung weglassen. *(6 Punkte)*

## **Aufgabe 5:** C++ Klassen *(17 Punkte)*

Betrachten Sie die folgende C++-Klasse für unscharfe Logik (fuzzy logic):

```
1   class fuzzy final
2   {
3   public:
4       fuzzy() = default;
5       explicit fuzzy(double);  explicit, damit nicht automatisch von
6       fuzzy(bool);             double nach fuzzy gecastet wird
7       fuzzy operator!() const;
8       friend fuzzy operator&&(const fuzzy&, const fuzzy&);
9       friend fuzzy operator||(const fuzzy&, const fuzzy&);
10  private:
11      double truth; // Wert zwischen 0.0 und 1.0
12  };
```

a) Die Klasse **fuzzy** enthält implizit weitere öffentliche Member-Funktionen.
Geben Sie diese Member-Funktionen explizit als Prototypen mit Implementierung **= default** an.
Nennen Sie außerdem den Grund, warum die mit **= default** angeforderten Implementierungen korrekt sind. *(3 Punkte)* double truth: <- primitiver typ, keine recourcen/heap verwaltung deshalb korrekt

b) Betrachten Sie das folgende Testprogramm für die Klasse **fuzzy**:

```
1   int main()
2   {
3       const fuzzy eher_ja{0.8};
4       const fuzzy eher_nein = !eher_ja;
5       fuzzy f;
6       f = eher_ja || false;
7   }
```

Geben Sie für jede Zeile an, welche der expliziten und in a) abgefragten impliziten Member- und **friend**-Funktionen der Klasse **fuzzy** dort aufgerufen werden, sofern man alle Optimierungen des Compilers außer Acht lässt. *(10 Punkte)*

*Hinweis: Bedenken Sie, dass Operatoren temporäre Objekte zurückgeben.*

c) Welchen Prototyp muss man in der Klasse **fuzzy** ergänzen, damit sich die folgende Ausgabeanweisung für einen **fuzzy**-Wert übersetzen lässt? Geben Sie den vollständigen Prototyp an. *(2 Punkte)*

```
std::cout << fuzzy{0.2} << '\n';
```

friend std::ostream& opeartor<<(std::ostream&, const fuzzy&)

d) Was sollte man in modernem C++ statt der folgenden Deklaration schreiben und warum? *(2 Punkte)*

```
fuzzy a[4];
```

std::arry<fuzzy,4> a; - fasst dasselbe wie fuzzy a[4], kennt aber seine länge

## **Aufgabe 6**: Make *(12 Punkte)*

*LaTeX ist eine Auszeichnungssprache zum Erstellen formatierter Dokumente. Mit einem beliebigen Texteditor wird eine Quelldatei mit Endung* ***.tex*** *erstellt, die den Inhalt des Dokuments mit den Auszeichnungen enthält.*

*Beispiel:*
```
% hello.tex - Our first LaTeX example!
\documentclass{article}
\begin{document}
Hello World!
\end{document}
```

*Das Linux-Kommando* ***pdflatex*** *erstellt aus der* ***.tex****-Quelle ein formatiertes* ***.pdf****-Dokument. Dabei entstehen zusätzlich Hilfsdateien mit den Endungen* ***.aux*** *und* ***.log****.*

*Beispiel:* ***pdflatex hello.tex***
*erstellt* ***hello.pdf*** *und als Nebenprodukt* ***hello.aux*** *sowie* ***hello.log***

Erstellen Sie ein **`Makefile`** mit folgendem Inhalt:

- einer Musterregel, die aus einer **`.tex`**-Datei die formatierte **`.pdf`**-Datei erstellt
- einer Regel mit Pseudoziel **`all`**, die **`hello.pdf`** erstellt
- einer Regel mit Pseudoziel **`clean`**, die alle bei **`all`** erstellten Dateien löscht

Halten Sie die in der Vorlesung besprochenen Stilregeln ein, insbesondere die Stilregeln zur Verwendung von Variablen.

## **Aufgabe 7**: POSIX *(3 Punkte)*

Die Java-Bibliothek meldet Fehlersituationen, indem sie Ausnahmen wirft. Für unterschiedliche Fehler gibt es unterschiedliche Ausnahmeklassen. In C gibt es aber keine Ausnahmebehandlung. Wie melden POSIX-Funktionen Fehlersituationen und wie muss an der Aufrufstelle damit umgegangen werden?