OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG | INF | FAKULTÄT FÜR INFORMATIK

Institut für Simulation und Graphik
Dr. Christian Rössl

Magdeburg, 6. Februar 2012

Klausur zur Vorlesung "'Einführung in die Informatik"'

Name, Vorname		Studiengang	Matrikelnummer
Zusatzbätter	Unterschriften	Student/in	Aufsicht

Tabelle bitte *nicht ausfüllen*!

	Aufgabe	max. Punkte	erreicht
1	ADT	6	
2	Korrektheit (rekursiv)	5	
3	Korrektheit (imperativ)	6	
4	Komplexität: Schleifen	3	
5	Komplexität: Addition	3	
6	Registermaschine	8	
7	OOP	6	
8	Binäre Suche	10	
9	Quicksort	8	
10	Wahr oder falsch?	5	
A	*Bonus*: Quadratwurzel	(+2)	
		60 (+2)	

Bearbeitungszeit: 120 Minuten

Hinweise zur Klausurbearbeitung

- Überprüfen Sie die Klausur auf Vollständigkeit (12 nummerierte Blätter).
- Füllen Sie das Deckblatt aus!
- Beschriften Sie **alle** Blätter (Klausur, verteiltes Papier) mit Ihrem Namen.
- Legen Sie bitte alle für die Klausur benötigten Dinge, Stifte, Verpflegung und insbesondere Lichtbildausweis, auf Ihren Tisch. Schalten Sie Ihr **Mobiltelefon aus**!
- Hilfsmittel (Taschenrechner, Bücher, Skripten, Mobiltelefone, ...) sind **nicht** zugelassen!
- Täuschungsversuche führen zum Nichtbestehen der Klausur!
- Verwenden Sie für Ihre Antworten den freien Platz nach den Aufgaben und ggf. die Rückseiten der Blätter. – Melden Sie sich, wenn Sie zusätzliche leere Blätter benötigen.
- **Schreiben Sie deutlich!** Unleserliche Passagen können nicht korrigiert werden.
- Benutzung von Bleistiften sowie roter und grüner Stiftfarbe ist untersagt.
- Beschränken Sie sich auf die geforderten Angaben und halten Sie Ihre Antworten kurz und präzise. Nicht geforderte Angaben ergeben keine zusätzlichen Punkte.
- Geben Sie bei Rechenaufgaben vor jedem Schritt an, was Sie gerade berechnen. Bewertet wird der Rechenvorgang, das Endergebnis allein genügt nicht.

Viel Erfolg!

Aufgabe 1: Abstrakte Datentypen (6 Punkte)

Erweitern Sie den vorgegebenen ADT `Stack` zur Darstellung von Stapeln um die Operatoren `height` und `swap`:

- `height` liefert die *Höhe* (Anzahl der Einträge) des Stapels,
- `swap` liefert eine „Kopie“ des Stapels, wobei die beiden obersten Einträge *vertauscht* wurden!

Geben Sie dazu *Signaturen, Axiome* und gegebenenfalls *Vorbedingungen* für `height` und `swap` an!
Hinweis: Sie können, wenn nötig, die Typen `Bool` (Wahrheitswerte) und `Nat` (natürliche Zahlen inklusive Null) mit den üblichen logischen und arithmetischen Operatoren (z.B. $\neg$ und $+$) verwenden.

```
type Stack [Item]
import Bool, Nat
operators
  empty_stack : → Stack
  top         : Stack  → Item
  push        : Stack × Item → Stack
  pop         : Stack → Stack
  is_empty    : Stack → Bool
axioms (∀S : Stack), (∀i : Item)
  top(push(S,i)) = i
  pop(push(S,i)) = S
  is_empty(empty_stack) = true
  is_empty(push(S,i)) = false
preconditions
  top(S) : ¬is_empty(S)
  pop(S) : ¬is_empty(S)
```

Aufgabe 2: Korrektheit rekursiv definierter Algorithmen (5 Punkte)

Die partielle Funktion **pow** : $\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ ist rekursiv[1] definiert als

$$\begin{aligned} \texttt{pow}(x,y) \quad = \quad & \textbf{if } y = 0 \textbf{ then } 1 \\ & \textbf{else } \texttt{mult}(\texttt{pow}(x,\texttt{pred}(y)),x) \textbf{ fi} , \end{aligned}$$

mit

$$\texttt{mult}(x,y) = x \cdot y \qquad \text{und} \qquad \texttt{pred}(x) = x - 1 .$$

(a) Zeigen Sie mit Hilfe einer vollständigen Induktion, dass $\textbf{pow}(x,y) = x^y$ gilt für $y \geqslant 0$!

(b) Erklären Sie an diesem Beispiel kurz den Begriff *partielle Funktion*!

[1] *Hinweis*: Die nachfolgende Aufgabe 3 betrachtet eine iterative Variante.

Aufgabe 3: Korrektheit imperativ definierter Algorithmen (6 Punkte)

Gegeben ist der folgende iterative[2] Algorithmus `POW` (siehe unten).

(a) Zeigen Sie, dass `POW` partiell korrekt ist bezüglich der

- **Vorbedingung** $\{ Y \geqslant 0 \}$ und der
- **Nachbedingung** $\{ Z = X^Y \}$.

Verwenden Sie dazu die **Schleifeninvariante** $P \Leftrightarrow X^Y = Z \cdot X^W$.

(b) Welche zusätzliche Eigenschaft wäre noch zu zeigen, um die *totale Korrektheit* bezüglich Vor- und Nachbedingung zu zeigen? (Nennen Sie diese Eigenschaft! — Kein Beweis!)

```
POW:
var W,X,Y,Z : int;
input X,Y;
Z:=1;
W:=Y;
while W≠0 do
  Z:=Z*X;
  W:=W-1;
od;
output Z;
```

[2] *Hinweis*: Die vorhergehende Aufgabe 2 betrachtet eine rekursive Variante.

Aufgabe 4: Komplexität in O-Notation: Schleifen (3 Punkte)

Zu welchen Komplexitätsklassen in Abhängigkeit von `n` gehören die folgenden Anweisungsfolgen? (*Keine* Begründung nötig!)

(a)
```
for (int i=0;i<n;++i);
```

(b)
```
for (int i=100;i<n;++i);
```

(c)
```
for (int i=1;i<=n;i=i+i);
```

(d)
```
for (int i=1;i<=n;++i)
    for (int j=n;j>i;--j);
```

(e)
```
for (int i=n/2;i<n;++i);
```

(f)
```
for (int i=0;i<n;++i)
    for (int j=1;j<=n*n;++j);
```

Aufgabe 5: Komplexität in O-Notation: Schriftliches Addieren (3 Punkte)

Gegeben ist eine natürliche Zahl n. Nehmen Sie an, Sie bestimmen die Summe $n + n$ durch schriftliches Addieren mit Übertrag (im Dezimalsystem).
Beispiel: Für $n = 12345$ ergibt sich

$$\begin{array}{rr} & 12345 \\ + & 12345 \\ & 1 \\ \hline = & 24690 \end{array}$$

Geben Sie den Aufwand der schriftlichen Addition in Abhängigkeit von n an, wobei zum Addieren zweier Dezimalstellen konstanter Aufwand $O(1)$ angenommen wird. Begründen Sie Ihre Antwort!

Aufgabe 6: Registermaschine (8 Punkte)

(a) Welche Funktion berechnet das folgende Programm einer *Registermaschine* M bei einer gegebenen Startkonfiguration $b = 1, c_0 = 0, c_1 = x, c_2 = 0, \ldots$ mit $x \geqslant 0$? — Geben Sie zur Begründung die gleiche Berechnung als Java- oder Pseudocode an!

(b) Geben Sie ein Programm für eine Registermaschine an, das die folgende Funktion berechnet:

$$f(x,y) = \begin{cases} 0 & \text{für } x > y \\ 1 & \text{sonst .} \end{cases}$$

Dabei gilt für die Startkonfiguration $b = 1$ und Eingabe $c_1 = x$, $c_2 = y$ mit $x, y \geqslant 0$. Die Ausgabe erfolgt in der Endkonfiguration als $c_3 = f(x,y)$.

Registermaschine M

```
 1  CLOAD 1
 2  STORE 2
 3  LOAD 1
 4  IF c0 = 0 GOTO 12
 5  LOAD 2
 6  CMULT 3
 7  STORE 2
 8  LOAD 1
 9  CSUB 1
10  STORE 1
11  GOTO 4
12  END
```

Die **Befehle** haben folgende Bedeutung:

Befehl	Arg.	Semantik
`LOAD`	$i > 0$	$b' = b+1 \wedge c_0' = c_i \wedge c_j' = c_j$ für $j \neq 0$
`CLOAD`	$i \geqslant 0$	$b' = b+1 \wedge c_0' = i \wedge c_j' = c_j$ für $j \neq 0$
`STORE`	$i > 0$	$b' = b+1 \wedge c_i' = c_0 \wedge c_j' = c_j$ für $j \neq i$
`ADD`	$i > 0$	$b' = b+1 \wedge c_0' = c_0 + c_i$
`CADD`	$i > 0$	$b' = b+1 \wedge c_0' = c_0 + i$
`SUB`	$i > 0$	$b' = b+1 \wedge c_0' = \begin{cases} c_0 - c_i & \text{für } c_0 \geqslant c_i \\ 0 & \text{sonst} \end{cases}$
`CSUB`	$i > 0$	$b' = b+1 \wedge c_0' = \begin{cases} c_0 - i & \text{für } c_0 \geqslant i \\ 0 & \text{sonst} \end{cases}$
`MULT`	$i > 0$	$b' = b+1 \wedge c_0' = c_0 \cdot c_i$
`CMULT`	$i \geqslant 0$	$b' = b+1 \wedge c_0' = c_0 \cdot i$
`DIV`	$i > 0$	$b' = b+1 \wedge c_0' = \lfloor c_0 / c_i \rfloor$
`CDIV`	$i > 0$	$b' = b+1 \wedge c_0' = \lfloor c_0 / i \rfloor$
`GOTO`	$i > 0$	$b' = i$
`IF` $c_0 = 0$ `GOTO`	$i > 0$	$b' = \begin{cases} i & \text{für } c_0 = 0 \\ b+1 & \text{sonst} \end{cases}$
`END`		$b' = b$

Aufgabe 7: OOP: Vererbung und Polymorphie (6 Punkte)

Gegeben sind die folgenden Klassen als Java-Code.

```
public abstract class Animal {
  public abstract void move(String dest);
  public void move(int x) { System.out.println("teleport to "+x); }
}

public class Fish extends Animal {
  public void move(String dest) { System.out.println("swim "+dest); }
}

public class Bird extends Animal {
  public void move(String dest) { System.out.println("fly"+dest); }
  public void move(int x,int y) { System.out.println("fly "+x+y); }
}

public class RoadRunner extends Bird {
  public void move(String dest) { System.out.println("run "+dest); }
  public void move(int x) { System.out.println("run faster to "+x); }
}
```

(a) Wie lautet die Ausgabe des folgenden Java-Codes? Ergänzen Sie die Tabelle.

```
1 Fish shark=new Fish();
2 Bird bird=new RoadRunner();
3 Animal animal=new Bird();
4
5 animal.move("home");
6 animal=bird;
7 shark.move(0);
8 bird.move(0);
9 animal.move("away");
```

Zeile	Ausgabe nach `System.out`
5	
7	
8	
9	

(b) Erklären Sie an diesem Beispiel kurz den Unterschied zwischen *Überladen* und *Überschreiben* von Methoden! Verwenden Sie dazu insbesondere die Begriffe *Klasse* und *Signatur*!

(c) Welche der folgenden Code-Zeilen kann/können nicht übersetzt werden? Warum nicht?

```
1   Animal shark=new Fish();
2   Animal gnu=new Animal();
3   RoadRunner coyote=new RoadRunner();
4   Fish nemo=coyote;
```

Aufgabe 8: Binäre Suche (10 Punkte)

Gegeben ist ein aufsteigend sortiertes Feld von ganzen Zahlen **int** `[] a` und eine ganze Zahl **int** `x`. Sie dürfen für die gesamte Aufgabe annehmen, dass `a.length>0` und `a[0]`$\leqslant$`x`$\leqslant$`a[a.length-1]`.

Der folgende Java-Code implementiert eine *binäre Suche* von `x` in `a` als eine Funktion `find(a,x)`.

```
1  public static int find(int[] a,int x) {
2    return find(a,0,a.length-1,x);
3  }
4  static int find(int[] a,int left,int right,int x) {
5    int mid=(left+right)/2;
6
7    if (left>right || x==a[mid]) return mid;
8
9    if (x<a[mid])                return find(a,left,mid-1,x);
10   else                         return find(a,mid+1,right,x);
11 }
```

(a) Was berechnet `find(a,x)`? (Welche Bedeutung hat der Rückgabewert?)

(b) Sei $n := $ `a.length` die Größe der Eingabe. Leiten Sie einen Ausdruck für die im schlechtesten Fall nötige Anzahl von Vergleichen in Abhängigkeit von n her!

Hinweise:

- Betrachten Sie nur den Test auf Gleichheit `x==a[mid]` (Zeile 7).
- Sie dürfen annehmen, dass n eine Zweierpotenz ist!

(c) Geben Sie den Java-Code für eine *nicht-rekursive* Funktion

int `ifind(`**int**`[] a,`**int** `x)`

an, die die binäre Suche wie oben aber iterativ, also mit Hilfe einer Schleife, implementiert. Es soll also gelten gelten `find(a,x)==ifind(a,x)` für alle `a`, `x`.

Aufgabe 9: Quicksort (8 Punkte)

Der *Quicksort* Algorithmus sortiert Daten rekursiv gemäß dem Teile-und-Herrsche Prinzip. Die nachfolgende Funktion `partition` implementiert einen wesentlichen Teil des Algorithmus.

```
static int partition(int[] a,int left,int right) {
  assert (left<=right);

  int p=a[right], t;                    // pivot
  int i=l-1, j=r;
  do {
    do ++i; while (a[i]<p);
    do --j; while (j>0 && a[j]>p);

    t=a[i]; a[i]=a[j]; a[j]=t;
  } while (i<j);

  a[j]=a[i]; a[i]=a[right]; a[right]=t;

  return i;                             // new index of pivot
}
```

(a) Erläutern Sie kurz die Semantik von `m=partition(a,left,right)` am Beispiel `a={2,4,1,5,3}`. (*Beschreiben Sie nur das Ergebnis – nicht die vorgegebene Implementierung!)*

(b) Implementieren Sie Quicksort in Java als Funktion

public static void `quicksort(`**int**`[] a)`.

Verwenden Sie dazu `partition()` und gegebenenfalls eine Hilfsfunktion.

(c) Geben Sie *je eine* Permutation der Folge $(1,2,3,4,5)$ an, für die Quicksort (1.) möglichst wenige (bester Fall) bzw. (2.) möglichst viele (schlechtester Fall) *Vertauschungen* benötigt!

(d) Erklären Sie kurz den Begriff *stabiles Sortierverfahren*! — Ist Quicksort *stabil*?

Aufgabe 10: Wahr oder falsch? (5 Punkte)

Kreuzen Sie jeweils (*ohne Begründung*!) an, ob eine Aussage wahr oder falsch ist.

Für jede richtige Antwort gibt es einen halben Punkt, für jede falsche Antwort wird ein halber Punkt abgezogen. (*Es gibt wenigstens 0 Punkte auf die gesamte Aufgabe.*) Beantworten Sie also lieber nur Fragen, bei denen Sie sich sicher sind!

Aussage	**wahr**	**falsch**
Mergesort eignet sich als *externes* Sortierverfahren.		
$3,5,8,14,22$ ist Teil der *Fibonacci Folge*.		
„Menge $\mathcal{M}$ ist *überabzählbar*." $\Leftrightarrow$ „$\mathcal{M}$ enthält unendlich viele Elemente."		
Die Menge der Funktionen ist *abzählbar*.		
Das Halteproblem ist *entscheidbar*.		
Nach Ausführung von `c=b; a=b++;` in Java gilt `b==c+1 && a==b` (für **int** `a,b,c`).		
In Java beschreibt `(a>b ? a : b)` das Maximum aus `a` und `b` (für **int** `a,b`).		
$\log_2 n \in O(\log n)$		
$n + \frac{1}{2}n^2 + \frac{1}{3}n^3 \in O(n^3)$		
Für eine sortierte Folge ist die Ausführungszeit einer binären Suche immer geringer als die einer sequentiellen Suche.		

Bonusaufgabe A: Ganzzahlige Quadratwurzel (2 Zusatzpunkte)

Der Algorithmus XYZ aus der Vorlesung berechnet den ganzzahligen Anteil der Quadratwurzel einer positiven ganzen Zahl.

```
XYZ:
var W,X,Y,Z : int;
input X;
Z:=0; W:=1; Y:=1
while W⩽X do
  Z:=Z+1;
  W:=W+Y+2
  Y:=Y+2
od;
output Z;
```

(a) Geben Sie den asymptotischen Aufwand zur Berechnung der Ausgabe Z in Abhängigkeit von der Eingabe X in O-Notation an! Begründen Sie Ihre Antwort!

(b) Geben Sie einen Algorithmus LXYZ an, der dasselbe Ergebnis mit logarithmischen Aufwand berechnet! (Also $(\forall \mathtt{X} \geqslant 0)\ (\mathtt{XYZ(X)} = \mathtt{LXYZ(X)})$ mit Aufwand $\mathrm{O}(\log \mathtt{X})$.) Begründen Sie Ihren Ansatz!