Klausur Softwaretechnik

22. Juli 2002

Name: _____

Vorname: _____

Matrikelnummer: _____

Aufg.0	Aufg.1	Aufg.2	Aufg.3	Aufg.4	Aufg.5	Σ	Note
/1	/17	/11	/10	/16	/5	/60	

Die Klausur besteht aus 13 Seiten und ist geheftet abzugeben.
Für die Vollständigkeit nicht gehefteter Klausuren wird keine Verantwortung übernommen.

Name: _____ Matrikelnummer: _____

Aufgabe 0 (1 Punkt)

Schreiben Sie auf jedes Blatt ihren Namen und Ihre Matrikelnummer in die dafür vorgesehenen Felder.

Aufgabe 1 (17 Punkte)

Beantworten Sie folgende Fragen.
Falsche Kreuze geben negative Punkte, fehlende Kreuze, sowie fehlende oder falsche Freitext-Antworten bewirken nichts. Weniger als 0 Punkte können Sie in einem Aufgabenteil nicht erreichen.

a) Was versteht man unter dem Geheimnisprinzip (nach David Parnas) bei Modulen?

_____

_____

_____

b) Weshalb sollte eine Benutztrelation zwischen Modulen hierarchisch, d.h. frei von Zyklen sein?

_____

_____

_____

c) Nennen Sie die beiden Arten von statischen Beziehungen in einem UML Klassendiagramm:

1. _____

2. _____

d) Welche Arten von Sichtbarkeit lassen sich in UML Klassendiagrammen für Attribute und Operationen definieren?

_____

e) Was versteht man unter einem „Regressionstest“?

_____

_____

_____

f) Welche der folgenden Aussagen über UML (Unified Modelling Language) treffen zu?

Richtig/Falsch

□/□ UML ist ein Software-Entwicklungsprozess.
□/□ UML ist ein semantisches Meta-Modell.
□/□ UML ist eine visuelle Programmiersprache.
□/□ UML ist eine Entwurfssprache.
□/□ UML beschreibt Diagramme und Notationen.
□/□ UML ist als Modellierungssprache für Menschen gedacht.

g) Welche beiden Eigenschaften von Programmen sollen bei der Programmoptimierung verringert werden?

1. _____

2. _____

h) Ordnen Sie die angegebenen Optimierungsebenen in die Reihenfolge, in der sie üblicherweise betrachtet werden.

(Schritt-Nr.)

_____ Feinoptimierung
_____ Systemstruktur
_____ Problemstellung
_____ Hardware
_____ Algorithmen und Datenstrukturen
_____ Systemsoftware

i) Was ist der Zweck des Entwurfsmusters „Iterator“?

_____

_____

_____

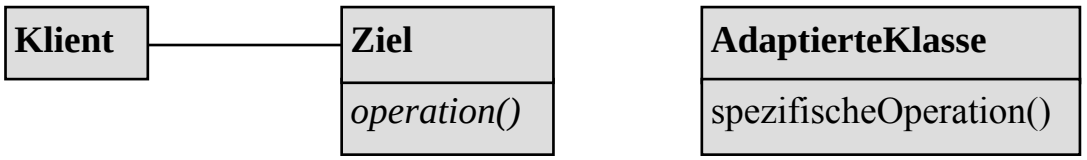
j) Nennen Sie zwei Beispiele für nichtfunktionale Produkts- und Qualitätsanforderungen, die Einfluss auf den Entwurf haben können:

1. _____

2. _____

Name: _____ Matrikelnummer: _____

k) Der Zweck des Entwurfsmusters „Adapter“ ist es, die Schnittstelle einer Klasse an eine andere von ihren Klienten erwartete Schnittstelle anzupassen.
Erweitern Sie das folgende Klassendiagramm um eine Adapter-Klasse. Zeichnen Sie alle notwendigen Relationen und Methoden ein und geben Sie auch die Implementierung der Methoden an.



l) Betrachten Sie folgende Entscheidungstabellen:

ET1: Bestellung	R1	R2	R3	R4
Artikel lieferbar	J	J	N	N
Zahlungsverhalten OK	J	N	J	N
Per Rechnung liefern	X			
Per Nachnahme liefern		X		
Artikel nachbestellen			X	X
Telefonischer Bescheid			X	
Schriftlicher Bescheid				X

ET2: Seminar	R1	R2	Else
Angemeldet?	J	J	
Abgemeldet?	N	verspätet	
Teilgenommen?	J	N	
Gebührensatz	100%	50%	0%
Rechnung schicken	X	X	

ET3: EC-Karte lesen und prüfen	R1	R2	R3	R4
EC-Karte lesbar	N	J	J	J
EC-Karte gültig_bis > heute	-	N	J	J
PIN = EC-Karte PIN	-	-	J	N
EC-Karte Fehlversuche < 3	-	-	-	J
EC-Karte Fehlversuche = Fehlversuche + 1				X
Ablehnung = Karte nicht lesbar	X			
Ablehnung = Gültigkeit abgelaufen		X		
Ablehnung = Falsche PIN				X
Karte und Pin OK			X	

Um welche Art von Entscheidungstabelle (ET) handelt es sich jeweils?

(Bedeutung der Kreuze: richtig/falsch)	ET1	ET2	ET3
Eintreffer-ET	□/□	□/□	□/□
Formal vollständige ET	□/□	□/□	□/□
Inhaltlich vollständige ET	□/□	□/□	□/□
Erweiterte ET	□/□	□/□	□/□

Aufgabe 2 (Petri-Netze) **(11 Punkte)**

In dieser Aufgabe soll ein Petri-Netz entworfen werden, welches die Anzahl der Marken in zwei Eingabestellen miteinander multipliziert und die entsprechende Zahl an Marken in einer Ausgabestelle ablegt.

Dazu ist es notwendig, eine neue Art von Kanten einzuführen.
Definition: Eine **Verbotskante** ist eine spezielle Verbindung von Stellen zu Transitionen, wodurch sich die Aktivierungsbedingung wie folgt ändert: Eine Transition ist zulässig, wenn die übliche Zulässigkeitsbedingung erfüllt ist und wenn sich in der Stelle, die durch die Verbotskante mit der Transition verbunden ist, keine Marke befindet. Somit ist ein „Test gegen Null“ möglich.
Eine Verbotskante wird wie folgt gezeichnet:

(a) (b)

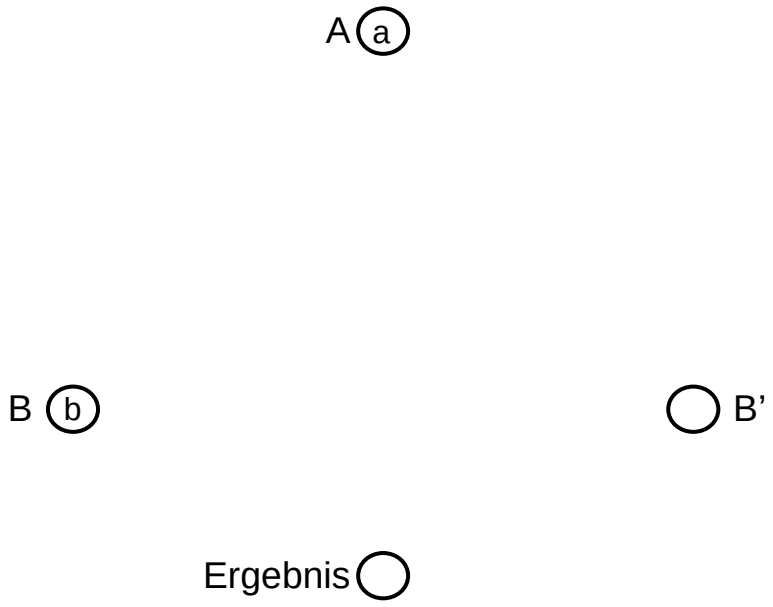
In Beispiel (a) ist die Transition aktiviert, in (b) nicht.

a) Entwerfen Sie zunächst ein Petri-Netz, welches bestimmt, ob eine gerade oder eine ungerade Zahl an Marken in der Stelle *A* liegt. Ergänzen Sie dazu die untenstehende Schablone. (Es sind keine weiteren Stellen notwendig.)
Die Marken sollen aus *A* entfernt werden, und nach Entfernung aller Marken muss eine Marke entweder in der Stelle *gerade* oder *ungerade* verbleiben.

b) Entwerfen Sie nun einen *Kopierer* und ergänzen Sie dazu die untenstehende Schablone. (Es sind keine weiteren Stellen notwendig.)
In der Anfangsmarkierung sind die Stellen *Start*, *Fertig*, *Ziel 1* und *Ziel 2* leer. Die Stelle *Quelle* enthält die zu kopierende Anzahl Marken *n*. Das Petri-Netz soll so lange inaktiv bleiben, wie die Stelle *Start* leer ist. Sobald in *Start* eine Marke gelegt wird, soll das Netz zu schalten beginnen. Es „verschiebt“ dann alle Marken von *Quelle* nach *Ziel 1* und *Ziel 2*, so dass sich anschließend in beiden Stellen *n* Marken und in der *Quelle* 0 Marken befinden. Das Ende der Operation soll durch Entfernen der Marke in *Start* und Hinzufügen einer Marke in *Fertig* angezeigt werden.
Hinweis: Machen Sie sich klar, warum zur Lösung dieses Aufgabenteils, die oben eingeführten Verbotskanten notwendig sind.

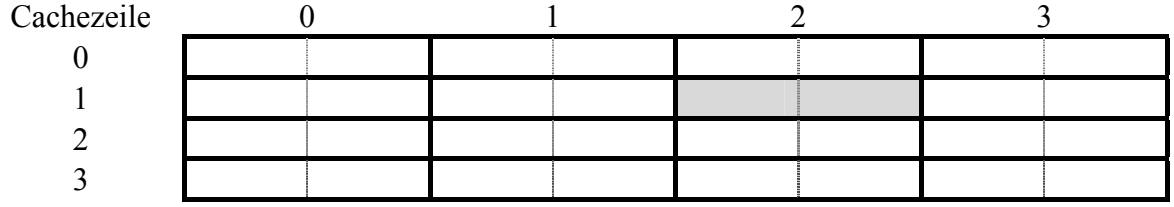
c) Entwerfen Sie nun den *Multiplizierer*, der die Anzahl der Marken in zwei Stellen *A* und *B* miteinander multipliziert und eine entsprechende Anzahl Marken der Stelle *Ergebnis* hinzufügt.
Der *Multiplizierer* entsteht durch Verknüpfung der Netze aus Aufgabenteil a) und b). Die beiden Faktoren *a* und *b* befinden sich in den Stellen *A* und *B*; die Multiplikation beginnt sobald die Marken in die Stelle *A* gelegt werden und endet, wenn diese aufgebraucht wurden. Das Produkt befindet sich in der Stelle *Ergebnis*.
Hinweise: Die Marken in der Stelle *B* werden mit Hilfe des *Kopierers* auf Aufgabenteil b) abwechselnd von *B* nach *B'* und *Ergebnis* kopiert und anschließend von *B'* nach *B* und *Ergebnis*. Bei jedem Kopiervorgang erhöht sich daher die Anzahl der Marken in *Ergebnis* um *b*, während die Summe der Marken in *B* und *B'* konstant bleibt. Das Petri-Netz aus Aufgabenteil a) wird dazu verwendet, jeweils eine Marke aus *A* zu entnehmen und abwechselnd einen Kopiervorgang in eine der beiden Richtungen anzustoßen.

Ergänzen Sie das folgende Petri-Netz, so dass es in der beschriebenen Weise funktioniert. Vergessen Sie nicht, die Anfangsmarkierung für Ihre zusätzlichen Stellen einzutragen.



Aufgabe 3 (Cache, Optimierung) **(10 Punkte)**

Gegeben sei ein direkt abgebildeter (Assoziativität=1) Cache. Er sei 32 Byte groß und habe 4 Cachezeilen. Jede Cachezeile besteht aus 8 Byte. Der Datentyp **`int`** besteht aus 2 Byte. Somit passen in jede Cachezeile immer genau 4 Datenelemente des Typs **`int`**. Die folgende Zeichnung veranschaulicht noch einmal den Aufbau des Caches.



Die einzelnen Kästchen entsprechen einem Byte; 2 können ein Element des Datentyps **`int`** aufnehmen, wie z.B. der grau markierte Bereich in Cachezeile 1 und Position 2.

Betrachten Sie nun folgende Programmschleife in C:

P

```
01   int a[8];
02   int b[16];
     ...
11   for ( i=0; i<8; i++ ) {
13      a[i] = b[2*i] + a[i];
14   }
     ...
```

Die Felder **`a`** und **`b`** und alle anderen Variablen haben den Elementtyp **`int`**. Die Programmschleife addiert jedes zweite Element von Feld **`b`** zu einem Element von Feld **`a`** hinzu. Die anderen, nicht verwendeten Elemente von b werden nach der Schleife nicht mehr gebraucht.

Für die Anfangsadressen der Felder im Hauptspeicher gilt
adr(**`a[0]`**) mod 32 = 0
adr(**`b[0]`**) mod 32 = 0.

Nehmen Sie für die folgenden Aufgaben an, dass alle skalaren Variablen in Registern des Prozessors liegen und dass die Elemente der Felder **`a`** und **`b`** über den Cache in den Prozessor geladen werden.

Name: _____ Matrikelnummer: _____

a) Markieren Sie jeweils in den entsprechenden Cacheabbildungen die Cache-Positionen der Feldelemente von **a** und **b**, die bei der ersten Iteration (**i**=0) der **i**-Schleife zugegriffen werden mit einem „O“ (Kreis).

Cacheabbildung von **a**

Cachezeile	0	1	2	3
0				
1				
2				
3				

Cacheabbildung von **b**

Cachezeile	0	1	2	3
0				
1				
2				
3				

b) Markieren Sie nun den gesamten Cache-Bereich mit „X“en, der von den Feldern **a** und **b** bei einem kompletten Durchlauf der **i**-Schleife (0<=**i**<8) benutzt wird.

c) Nehmen Sie nun an, der Cache sei vor Beginn der Schleife frei von Elementen der beiden Felder. Berechnen Sie, wie häufig bei einem kompletten Durchlauf der **i**-Schleife eine neue Cache-Zeile aus dem Hauptspeicher geladen werden muss. Dabei gelte die Voraussetzung, dass bei der Addition zuerst auf das Feld **b** und dann auf das Feld **a** zugegriffen wird.

Anzahl der geladenen Cache-Zeilen: _____

d) Tragen Sie in der folgenden Tabelle die Cachebelegung nach dem Ende der Schleife ein. (Es genügt, die letzte Spalte auszufüllen.)

Cachezeile	0	1	2	3	Belegt durch Elemente aus Feld (**a**/**b**)
0					
1					
2					
3					

e) Es sollen die Cache-Konflikte zwischen den Feldern **a** und **b** aufgelöst werden. Dazu sollen die Elemente innerhalb von **b** umsortiert werden, so dass sich der von **b** benötigte Bereich im Cache nicht mehr mit dem von **a** benötigten überschneidet. Sie müssen dazu vor Zeile **11** zwei neue Zeilen (Nummer **9** und **10**) zur Vorberechnung einfügen und den Indexausdruck des Feldes **b** in Zeile **13** verändern. Nehmen Sie an, dass die Elemente an ungeraden Indizes des Feldes **b** nicht mehr benötigt werden.

Zeile 9: _____

Zeile 10: _____

Zeile 13’: _____

Anzahl der (von dem veränderten Programm) geladenen Cache-Zeilen: _____
(Nehmen Sie wiederum einen zu Beginn „leeren“ Cache an, beachten Sie aber neben der **i**-Schleife auch die neuen Anweisungen (Zeilen 9 und 10) zur Umsortierung.)

Aufgabe 4 (Datenfluss- und Kontrollflussorientierter Strukturtest) **(16 Punkte)**

Gegeben sei die Java Methode ***b*** zur Berechnung des Binominalkoeffizienten:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \prod_{i=1}^{n-k} \frac{(k+i)}{i}$$

```
public long b(long n, long k) {
     if (n < 0 | k < 0 | k > n)
          return 0;
     long p1 = 1, p2 = 1;
     long f1 = n, f2 = 1;
     while (f2 <= (n - k)) {
          p1 *= f1;
          p2 *= f2;
          f1--;
          f2++;
     }
     return p1 / p2;
}
```

a) Vervollständigen Sie die untenstehende Schablone des Kontrollflussgraphen G der Funktion **b**. Verwenden Sie dabei für die If- und While-Anweisungen leere Knoten und numerieren Sie alle Knoten in der Reihenfolge in der sie im Quelltext auftreten.

G:

1	*Start* Eingabe: ***n***, ***k***

Name: _____ Matrikelnummer: _____

b) Tragen Sie in die untenstehenden Tabellen nach dem Definitions-Nutzungs-Verfahren alle Definitionen, r-Nutzungen und p-Nutzungen ein. (Markieren Sie eventuell leere Mengen mit einem Strich „-“.)

Knoten K	def(K)	r-Nutzung(K)
1	{ }	{ }
2	{ }	{ }
3	{ }	{ }
4	{ }	{ }
5	{ }	{ }
6	{ }	{ }
7	{ }	{ }

Kante K	p-Nutzung(K)
(,)	{ }
(,)	{ }
(,)	{ }
(,)	{ }
(,)	{ }
(,)	{ }
(,)	{ }

c) Geben Sie unter Verwendung von b) <u>alle</u> drn- und dpn-Mengen an.

drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }
drn(,) = { }	dpn(,) = { }

d) Bestimmen Sie zu den folgenden Pfad-Selektions-Kriterien eine minimale Menge von Testdaten (**n**, **k**), die diese erfüllen und die zugehörigen Pfade, die durchlaufen werden.
Lässt sich ein Kriterium nicht durch Testdaten erfüllen, so geben Sie eine Begründung dafür an.

„Alle Knoten“:

Testdaten: _____

Pfade: _____

„Alle Kanten“:

Testdaten: _____

Pfade: _____

„Alle Defintionen:

Testdaten: _____

Pfade: _____

e) Betrachten Sie nun die erste If-Anweisung von ***b***:

```
...
if (n < 0 | k < 0 | k > n)
     return 0;
...
```

Bestimmen Sie in diesem Teilgraphen zu den folgenden Kriterien des kontrollflussorientierten Strukturtests eine minimale Menge von Testdaten (**n**, **k**).
Lässt sich ein Kriterium nicht durch Testdaten erfüllen, so geben Sie eine Begründung dafür an.

„einfache Bedingungsüberdeckung“:

_____

_____

„mehrfache Bedingungsüberdeckung“:

_____

_____

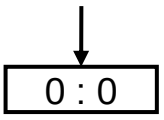
Name: _____ Matrikelnummer: _____

Aufgabe 5 (Endlicher Automat) **(5 Punkte)**

In den offiziellen Tennisregeln findet sich folgendes:

> Regel 26: Gewinn eines Spieles
>
> a) Vorteil-System
> Gewinnt ein Spieler seinen ersten Punkt, so zählt dies für ihn 15; gewinnt er seinen zweiten Punkt, so zählt dies für ihn 30; gewinnt er seinen dritten Punkt, so zählt dies für ihn 40; gewinnt er seinen vierten Punkt, so hat er ein „Spiel“ gewonnen mit folgender Ausnahme:
> Wenn beide Spieler drei Punkte gewonnen haben, wird der Spielstand „Einstand“ genannt; der nächste von einem Spieler gewonnene Punkt zählt „Vorteil“ für diesen Spieler. Gewinnt derselbe Spieler den nächsten Punkt, so gewinnt er das Spiel. Gewinnt aber der andere Spieler den nächsten Punkt, wird der Spielstand wieder „Einstand“ genannt und so weiter, bis einer der Spieler die auf „Einstand“ unmittelbar folgenden beiden Punkte gewinnt. Er hat dann das Spiel gewonnen.

Modellieren Sie ein Spiel mit Hilfe eines Mealy Automaten.
Erweitern Sie folgende Schablone. Das Eingabealphabet des Automaten ist $\{a,b\}$, wobei a für einen Punktgewinn von Spieler A und b für einen Punktgewinn von Spieler B steht. Gewinnt ein Spieler das Spiel, so soll der Automat in Zustand „Spiel A“ respektive „Spiel B“ übergehen und so das Ende des Spiels anzeigen.



Spiel A

Spiel B