Fachbereich Informatik

Klausur Informatik 2
12. Oktober 2000

Biedl / Lemke
Schöner / Tschirley

Name: ....

Vorname: ....

Matr.-Nr.: ....

Bearbeitungszeit: 120 Minuten

- Benutzen Sie für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier gerschrieben werden, können nicht gewertet werden!**
- Schreiben Sie Ihre Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwenden Sie auch die Rückseite. **Schreiben Sie keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!**
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen können nicht gewertet werden.
- Schreiben Sie nur in **blau** oder **schwarz**. Schreiben Sie nicht mit Bleistift!
- Bei der Implementation von OPAL-Funktionen brauchen *keine* Importe mit angegeben zu werden, wenn Funktionen aus dem gegebenen Teil der Bibliotheca Opalica (oder aus der Struktur `Nat`) oder sonstigen (in den Aufgaben) gegebenen Strukturen benutzt werden!
- Die OPAL-Implementierungen müssen nicht dokumentiert werden.
- Erscheint Ihnen eine Aufgabe mehrdeutig, wenden Sie sich an die Betreuer.
- Tragen Sie jetzt (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.

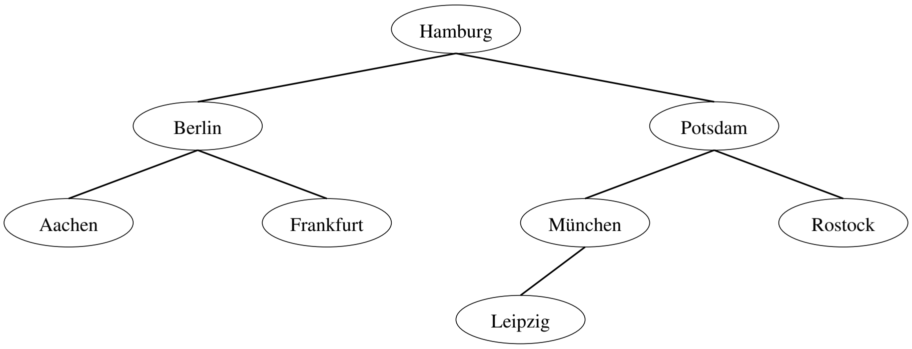
Aufgabe	Punkte	erreicht
1	10	
2	10	
3	13	
4	9	
5	4	
6	4	

1. Aufgabe (10 Punkte): Binäre Suchbäume mit Java

1.1. Definition von binären Suchbäumen (2 Punkte) Geben sie kurz die Struktur und die Eigenschaften binärer Suchbäume an.

1.2. Dynamik eines Suchbaums (2 Punkte)

1. Geben Sie eine Reihenfolge von Einfügeoperationen (inserts) an, die den unten gezeigten Suchbaum erzeugt haben könnte (Vergleichskriterium: Lexikographischer Vergleich).



2. Auf den oben gezeigten Suchbaum werden nun die folgenden Anweisungen angewendet:
insert("Lübeck"); insert("Dresden"); insert("Freiburg");
Zeichnen Sie den resultierenden Suchbaum (Vergleichskriterium: Lexikografischer Vergleich).

1.3. Aufbau eines Suchbaums (4 Punkte) Schreiben Sie in Java eine Methode, die die folgende Methodendeklaration implementiert:

```
static void searchTreeInsert(TreeNode bsb, String x);
```

Sie soll das Datum x in den binären Suchbaum, dessen Wurzelknoten bsb ist, einfügen. Verwenden Sie für diesen Algorithmus eine iterative Formulierung (keine Rekursion). Geben Sie vorher eine geeignete Deklaration des Datentyps TreeNode an (Klassendeklaration nur mit Datenelementen, ohne Konstruktoren und Methoden).

Hinweise: Sie können davon ausgehen, dass bsb nicht leer ist (ungleich null). Eine geeignete Vergleichsmethode könnte int String.compareTo(String) sein, die $-1, 0$ oder 1 zurückliefert, je nachdem ob der eigene String kleiner, gleich oder größer als der übergebene ist.

1.4. Angabe eines Sortierkriteriums (2 Punkte) Beschreiben Sie, wie Sie die Methode searchTreeInsert verändern müssten, damit der Suchbaum nach einem vom Benutzer zu übergebenden Sortierkriterium sortiert aufgebaut wird.

2. Aufgabe (10 Punkte): Aufwandsbestimmung

2.1. Dichte obere Schranken (5 Punkte) Beweisen oder widerlegen Sie (mit Hilfe der Definition des $\mathcal{O}$-Kalküls) jeweils, dass g eine dichte obere Schranke von f ist. Geben Sie dabei jeweils eine Konstante c und das dafür minimale n_0 an, für die die Definition zutrifft.

1. $f(n) = \ln\left(n^{0.5n}\right)$
   $g(n) = n\ln(n)$

2. $f(n) = \sqrt{9n} \cdot \frac{2n^2}{3n}$
   $g(n) = n^2$

2.2. Aufwand eines Programmfragments (5 Punkte) Bestimmen Sie die Aufwandsfunktion der folgenden Methode task im Worst-Case. Leiten sie daraus mit Hilfe des $\mathcal{O}$-Kalküls oder der im Skript oder unten angegebenen Regeln eine Aufwandsklasse ab. Nehmen Sie einen Zeitaufwand von einer Einheit für jeden Operator, jeden Funktionsaufruf und jedes return an. Achten Sie auf die Nachvollziehbarkeit Ihres Vorgehens, insbesondere müssen die gezählten Operatoren, etc. eindeutig einzelnen Zeilen der Methode zugeordnet werden können!

```
static int task(int x, int y) {
    if (y == 0)
        return (0);
    int temp = task(x,y/2);
    if ((y%2) != 0)
        return (temp+temp + x);
    else
        return (temp+temp);
}
```

Definition ($\mathcal{O}$-Kalkül):
Sind f und g zwei positiv-reellwertige Funktionen über $\mathbb{N}$, dann bedeutet $f \in \mathcal{O}(g)$, dass es eine Konstante $c \in \mathbb{R}^+$ und ein $n_0 \in \mathbb{N}$ gibt, sodass für alle $n > n_0$ gilt: $f(n) \leq c * g(n)$. g ist dann eine obere Schranke für f.

Regeln für rekurrente Aufwandsformeln:

Rekurrenzrelation		$\mathcal{O}(A)$
$A(n) = A(n-1) + bn^k$		$\mathcal{O}(n^{k+1})$
$A(n) = cA(n-1) + bn^k$	mit $c > 1$	$\mathcal{O}(c^n)$
$A(n) = cA(n/d) + bn^k$	mit $c > d^k$	$\mathcal{O}(n^{\log_d c})$
$A(n) = cA(n/d) + bn^k$	mit $c < d^k$	$\mathcal{O}(n^k)$
$A(n) = cA(n/d) + bn^k$	mit $c = d^k$	$\mathcal{O}(n^k \log n)$

3. Aufgabe (13 Punkte): Programmverifikation

3.1. Formalisieren von Aussagen (2 Punkte) Formalisieren Sie die folgenden Aussagen:

1. Jedes Element des Arrays arr ist doppelt so groß wie sein Vorgänger.
2. Der Array noprim enthält nur Zahlen ≥ 2, die keine Primzahlen sind. (**Wichtig:** Definieren Sie auch alle verwendeten Hilfsprädikate formal.)

3.2. Partielle Korrektheit (7 Punkte) Beweisen Sie mit Hilfe des Hoare-Kalküls die partielle Korrektheit der folgenden Methode. Sie modifiziert den übergebenen Array, indem sie in jedem Arrayelement die Summe aller ursprünglichen Elemente des Arrays bis zu diesem Index ablegt. Stellen Sie dazu erst eine geeignete POST-Condition auf und verifizieren Sie dann die Methode. Geben Sie die von Ihnen gefundene POST-Condition, Invariante und PRE-Condition an den vorgesehenen Stellen auf der nächsten Seite an. Verwenden Sie den Vordruck auf der nächsten Seite auch für die Verifizierung.

```
static void scan(int[] a) {
    int i = 1;
    while (i < a.length) {
        a[i] = a[i-1] + a[i];
        i = i+1;
    }
}
```

3.3. Totale Korrektheit (4 Punkte) Beschreiben Sie kurz die nötigen Schritte, um die totale Korrektheit einer while-Schleife zu zeigen. Führen Sie diese Schritte dann für die while-Schleife von scan durch.

```
// POST:

// PRE:

// Invariante:


static void scan(int[] a) {
    // PRE:

    int i = 1;

    while (i < a.length) {

        a[i] = a[i-1] + a[i];

        i = i+1;

    }

    // POST:
}
```

4. Aufgabe (9 Punkte): SCRAM-Assembler Programmierung

Ein *Palindrom* ist eine Zeichenkette, die sowohl von vorn als auch von hinten gelesen die gleichen Worte bildet. Bekannte Palindrome sind: „OTTO“, „ANNA“, „RELIEFPFEILER“.
Schreiben Sie ein SCRAM-Assemblerprogramm, das eine gegebene ASCII-codierte Zeichenkette (Tabelle siehe unten) dahingehend prüft, ob es sich bei der Zeichenkette um ein Palindrom handelt. Wenn dies der Fall ist, soll die Variable `is_pal` nach Abarbeitung des Programms und Anhalten des Prozessors eine `1` enthalten, andernfalls eine `0`.
Die Zeichenkette steht an der Adresse, die durch das Label `string` symbolisiert wird. Das Ende der Zeichenkette wird durch das ASCII-Zeichen `nil` angezeigt, das selbst zum String dazugehört (Null-terminierter String). Gehen Sie davon aus, daß die Zeichenkette ausschließlich Großbuchstaben sowie die terminierende Null (nil) enthält. Der String enthält niemals Null Zeichen.
Verwenden Sie zur Programmierung die Assembler-Instruktionen aus untenstehender Tabelle. Falls Sie Makros verwenden, geben Sie deren Definition an. Folgende Makros können Sie verwenden, ohne deren Definition angeben zu müssen: `INC` (*increment operand by one*), `DEC` (*decrement operand by one*) und `JNS` (*jump on negative set*).

Code	Mnemonik	Zyklen	Bedeutung der Operation
\$ 00 00	HLT	4	Halt, Ende der Programmabarbeitung.
\$ m1 op	LDA *A*	5 6 8	Laden des Akkumulators.
\$ m2 op	STA *A*	- 5 8	Speichern des Akkumulators.
\$ m3 op	ADD *A*	5 7 9	Addiere den Wert zum Akkumulator.
\$ m4 op	SUB *A*	5 7 9	Subtrahiere den Wert vom Akkumulator.
\$ m5 op	MUL *A*	5 7 9	Multipliziere Akkumulator mit dem Wert.
\$ m6 op	DIV *A*	5 7 9	Dividiere Akkumulator durch den Wert.
\$ m7 op	AND *A*	5 7 9	Bitweises UND des Akkus mit dem Wert.
\$ m8 op	OR *A*	5 7 9	Bitweises ODER des Akkus mit dem Wert
\$ m9 op	XOR *A*	5 7 9	Bitweises Exklusiv–ODER des Akkus
\$ 0A 00	NOT	5	Bitweise Negation des Akkumulators.
\$ 0B op	JMP *A*	4	Unbedingter Sprung an die Adresse *A*.
\$ 0C op	JEQ *A*	5	Bedingter Sprung an die Adresse *A*, wenn Akku den Wert Null enthält.
\$ 0D op	JNE *A*	5	Bedingter Sprung an die Adresse *A*, wenn Akku ungleich Null.
\$ 0F 00	NOP	4	Die “No Operation”.

ASCII-Codes

Die Zeichen `0x` vor einer Zahl deuten an, daß die Zahl im Hexadezimalsystem dargestellt ist. Der hexadezimale Code eines Zeichens berechnet sich aus der Summe der Spalten- und Zeilenbezeichnungen.
Bsp: `"G" = 0x40 + 0x7 = 0x47` was der dezimalen 71 entspricht.

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x00	**nil**	soh	stz	etx	eot	enq	ack	bel	bs	ht	lf	vt	ff	cr	so	si
0x10	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
0x20	spc	!	”	#	$	%	&	’	(	)	*	+	,	−	.	/
0x30	0	1	2	3	4	5	6	7	8	9	;	;	<	=	>	?
0x40	@	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**
0x50	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	[	\	]	↑	←
0x60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del

Tabelle nach: N. Wirth: *Algorithmen und Datenstrukturen mit Modula-2*, 4. Aufl., B. G. Teubner, Stuttgart 1986

5. Aufgabe (4 Punkte): SCRAM-Makros

Entwerfen Sie das Makro *Shift Left* (SHL). Das Makros verschieben den Inhalt des AC um die als Operand übergebene Anzahl von Bits nach links. Verwenden Sie ausschließlich Assembleranweisungen aus der Tabelle aus Aufgabe A4. Verwenden Sie keine anderen Makros!
Achten Sie bei der Realisierung auf eine möglichst geringe Anzahl von bei der Abarbeitung durchzuführenden Assemblerbefehlen.

Hinweis: Die Shift-Operationen *shift left/shift right* lassen sich durch eine *Division/Multiplikation* der zu verschiebenden Zahl erreichen.

FB Informatik
Klausur Informatik 2
12. Oktober 2000

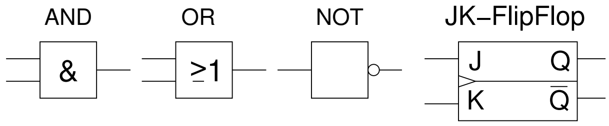
Name: ....

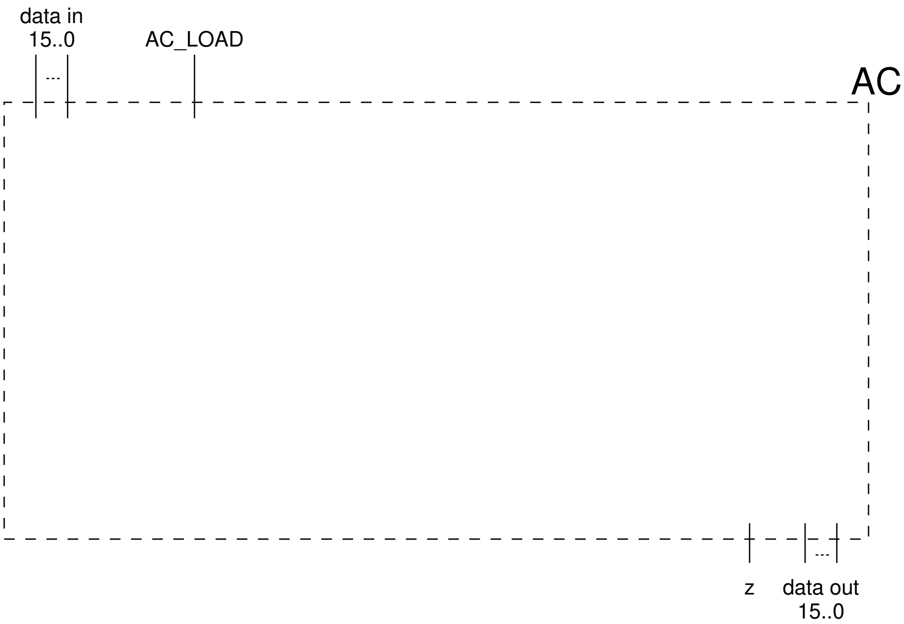
Matr.-Nr. ....

A6

6. Aufgabe (4 Punkte): Schaltwerke

Entwerfen Sie das Schaltbild, welches den Akkumulator des SCRAM laut nachstehendem SCRAM-Schaltbild realisiert. Beschränken Sie sich auf die ausführliche Darstellung der Schaltwerke für die Bits 0 und 15. Deuten Sie die dazwischenliegenden Bits nur an. Skizzieren Sie das Schaltbild in unten stehendem Diagramm.
Verwenden Sie ausschließlich die nachfolgend abgebildeten Bauteile (das UND-Gatter darf auch mehr als 2 Eingänge aufweisen):





SCRAM Schaltbild