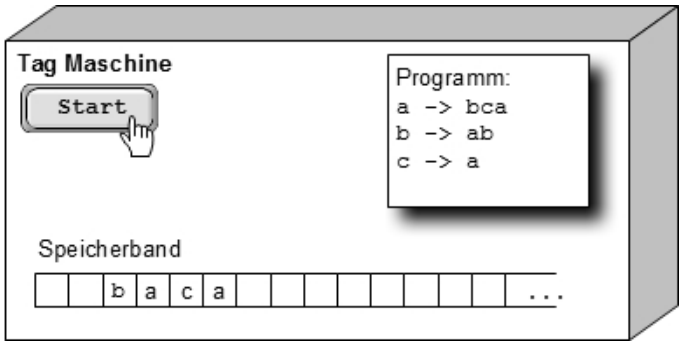
IN-G 2013 HT

Saarland
Ministerium
für Bildung und Kultur

Schriftliche Abiturprüfung 2013

Fach:	**Informatik Haupttermin**
Kurstyp:	G-Kurs
Datum:	19.04.2013
Bearbeitungszeit:	3 Zeitstunden
Hilfsmittel:	nicht-programmierbarer Taschenrechner
Seitenzahl:	Die Prüfungsaufgabe umfasst mit Deckblatt und Anlagen 7 Seiten.

1. Aufgabe



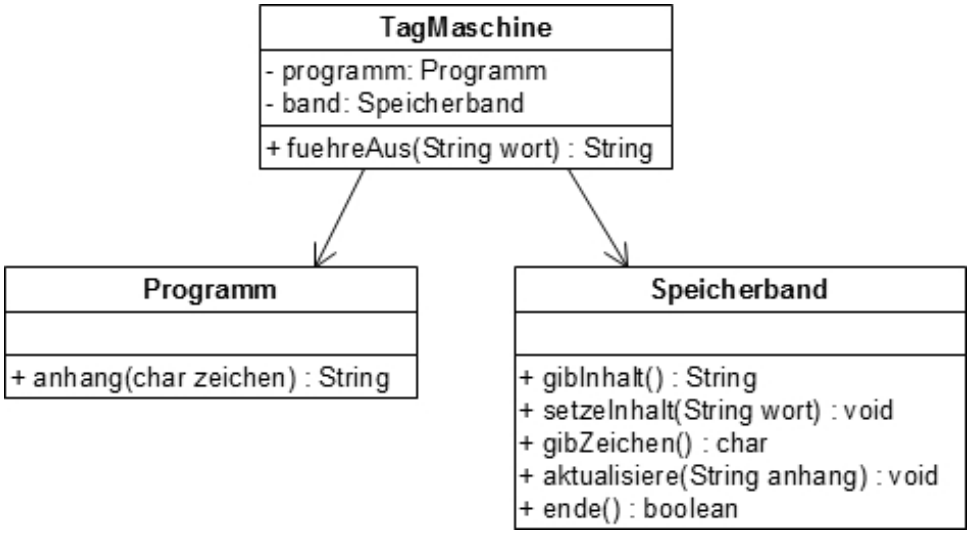
Der amerikanische Logiker *Emil Post* hat sich in den 1920er Jahren mit einer theoretischen, programmierbaren Maschine beschäftigt, die heute als *Tag-Maschine* bekannt ist (*engl.: to tag on something – etwas anhängen*). Die Maschine besitzt ein Speicherband, auf dem zu jedem Zeitpunkt genau ein Wort steht. In jedem Programmschritt wird ein Anhang an das gespeicherte Wort angehängt, der durch das *erste* Zeichen des gespeicherten Wortes und das Programm eindeutig festgelegt ist. Anschließend werden in jedem Schritt die ersten beiden Zeichen des Wortes gelöscht. Die Maschine hält, sobald das gespeicherte Wort weniger als zwei Zeichen lang ist.

Beispiel: Das Programm der oben abgebildeten Tag-Maschine führt bei der anfänglichen Speicherbandbelegung `baca` zu folgender Wortfolge auf dem Speicherband:

```
baca          (Initiale Bandbelegung. Das Zeichen b bestimmt den Anhang ab.)
  caab            (Angehängt wurde ab. Die ersten beiden Zeichen wurden gelöscht.)
    aba
      abca
        cabca
          ...
```

1.1 Ergänzen Sie das im Beispiel begonnene Beispiel um vier weitere Ausführungsschritte.

1.2 Das unten abgebildete Diagramm modelliert die Tag-Maschinen objektorientiert. *(Eine Dokumentation der drei Klassen befindet sich in Anhang 1.)*
Implementieren Sie die Methode `fuehreAus` der Klasse `TagMaschine`.



1.3 Erläutern Sie, inwiefern sich der abstrakte Datentyp *Schlange* (*Queue* / *FIFO*) anbietet, um das jeweils aktuelle Wort in der Klasse `Speicherband` zu speichern.

1.4 Tag-Maschinen unterliegen im Allgemeinen dem Halteproblem. Erläutern Sie die Bedeutung dieser Aussage.

1.5 Tag-Maschinen-Programme sind – nicht nur in ihrer äußeren Erscheinung – eng verwandt mit Grammatiken.

Stellen Sie den Übergangsgraphen eines nicht deterministischen endlichen Automaten dar, der genau diejenige Sprache akzeptiert, die die folgende Grammatik produziert.

Terminale:	{a,b,c}
Nicht-Terminale:	{S, A}
Startsymbol:	S
Produktionsregeln:	S → aS | bS | cS | aA
	A → a

Entwickeln Sie mit Hilfe der Teilmengenkonstruktion den Übergangsgraphen eines deterministischen endlichen Automaten, der die selbe Sprache akzeptiert wie der von Ihnen angegebene nicht-deterministische.

2. Aufgabe

Rekursion und *DC*

Das Potenzieren zweier **natürlicher** Zahlen a und b kann auf die folgenden beiden Arten definiert werden:

i. $$a^b := \begin{cases} 1, & \text{falls } b = 0 \\ a \cdot a^{b-1}, & \text{sonst} \end{cases}$$

ii. $$a^b := \begin{cases} 1 & , \text{falls } b = 0 \\ \left(a^{b/2}\right)^2, & \text{falls } b > 0 \text{ und } b \text{ gerade} \\ a \cdot \left(a^{(b-1)/2}\right)^2, & \text{falls } b > 0 \text{ und } b \text{ ungerade} \end{cases}$$

2.1 Implementieren Sie eine Methode `potenz1` und eine Methode `potenz2`, die bei Übergabe der Parameter a und b die Potenz a^b nach Definition (i) bzw. Definition (ii) berechnen und zurückgeben.
*[**Hinweis:** Sie dürfen in dieser Aufgabe eine Methode/Funktion* `square` *benutzen, die bei Übergabe einer natürlichen Zahl deren Quadrat liefert.]*

2.2 Geben Sie alle weiteren Aufrufe der Methode `potenz2` an, die nach dem Aufruf `potenz2(2, 18)` stattfinden.

2.3 Analysieren Sie die Laufzeit der Methoden `potenz1` und `potenz2`.

2.4 Implementieren Sie eine Methode `potenz3`, die bei Übergabe der Parameter a und b die Potenz a^b iterativ berechnet und zurückgibt.

2.5 Implementieren Sie ein *DC*-Programm, das für zwei eingegebene Zahlen a und b die Potenz a^b berechnet und ausgibt. Erläutern Sie Ihren Programmcode.
*[**Hinweis:** Der Befehlssatz des DC befindet sich im Anhang. Beachten Sie, dass der übliche Befehlssatz um den Befehl* `MUL` *ergänzt ist.]*

3. Aufgabe

Kryptologie

Bei der Schlüsselerzeugung des *RSA*-Verfahrens wählt Alice die Primzahlen $p = 17$ und $q = 19$ sowie den öffentlichen Schlüssel $e = 23$.

3.1 Berechnen Sie die zugehörigen Werte n und $\varphi(n)$.

3.2 Bestimmen Sie Alices privaten Schlüssel d.

3.3 Verschlüsseln Sie die Nachricht $m = 4$ mit Alices öffentlichem Schlüssel e.

Auf *Adi Shamir* (dem 'S' in 'RSA') geht ein Verfahren zur geheimen Nachrichtenübermittlung zurück, das oftmals als *No-Key-Verfahren* bezeichnet wird. Bildhaft kann man das Verfahren wie folgt beschreiben:

i) Alice steckt ihre Nachricht in eine Kiste und verschließt die Kiste mit einem Vorhängeschloss, dessen Schlüssel sie behält. Sie schickt die Kiste an Bob.
ii) Bob verschließt die Kiste mit einem zweiten Vorhängeschloss, behält den Schlüssel und schickt die Kiste zurück an Alice.
iii) Alice entfernt ihr Schloss und schickt die Kiste an Bob.
iv) Bob entfernt sein Schloss und öffnet die Kiste.

In der Kryptologie wird dieses Bild wie folgt modelliert: Alice und Bob einigen sich zunächst auf eine Primzahl p.
Alice wählt eine geheime Zahl a ($0 < a < p$) und bestimmt eine zugehörige Zahl a' mit $a \cdot a' mod (p-1) = 1$.
Bob wählt eine geheime Zahl b ($0 < b < p$) und bestimmt eine zugehörige Zahl b' mit $b \cdot b' mod (p-1) = 1$.

i) Alice berechnet $A = m^a \; mod \; p$. Sie sendet A an Bob.
ii) …
iii) …
iv) …

3.4 Vervollständigen Sie die mathematische Beschreibung des No-Key-Verfahrens.

3.5 Zeigen Sie, weshalb Bob am Ende die Nachricht tatsächlich lesen kann.

3.6 Erläutern Sie, weshalb das Verfahren als „*No-Key*"-Verfahren bezeichnet wird.

(***Hinweis***: Falls Alice ihre Werte a und a' wie gefordert so wählt, dass $a \cdot a' mod (p-1) = 1$, so gilt $m^{a \cdot a'} mod p = m$.)

Anhang 1 (zu Aufgabe 1.2): Klassendokumentationen

Klasse `TagMaschine`

Methoden:

`fuehreAus`	Parameter:	`wort` (String)
	Beschreibung:	Die Methode schreibt zunächst `wort` auf das interne Speicherband und führt dann das interne Programm aus. Falls das Programm hält, wird der Inhalt des Speicherbandes zurückgegeben.

Klasse `Programm`

Methoden:

`anhang`	Parameter:	`zeichen` (char)
	Beschreibung:	Die Methode gibt den dem `zeichen` entsprechenden Anhang aus.

Klasse `Speicherband`

Methoden:

`gibInhalt`	Parameter:	-
	Beschreibung:	Die Methode liefert den aktuellen Bandinhalt.
`setzeInhalt`	Parameter:	`wort` (String)
	Beschreibung:	Die Methode setzt den Bandinhalt auf `wort`.
`gibZeichen`	Parameter:	-
	Beschreibung:	Die Methode liefert das vordere Zeichen des aktuellen Bandinhalts. (Falls das Speicherband leer ist, wird ein Fehler erzeugt.)
`aktualisiere`	Parameter:	`anhang` (String)
	Beschreibung:	Die Methode hängt `anhang` hinter den aktuellen Bandinhalt und löscht die ersten beiden Zeichen des Bandinhalts.
`ende`	Parameter:	-
	Beschreibung:	Die Methode liefert `true`, falls der aktuelle Bandinhalt eine Länge kleiner als 2 hat. Andernfalls liefert sie `false`.

Anhang 2 (zu Aufgabe 2.5): Befehlssatz des *DC*

Grundbefehle

Mnemo	Bedeutung
LDA	Lade den Wert der angegebenen Speicherstelle in den Akkumulator.
STA	Speichere den Inhalt des Akkumulators an der angegebenen Speicherstelle ab.
ADD	Addiere den Wert der angegebenen Speicherstelle zum Inhalt des Akkumulators.
SUB	Subtrahiere den Wert der angegebenen Speicherstelle vom Inhalt des Akkumulators.
MUL	**Multipliziere den Inhalt des Akkumulators mit dem Wert der angegebenen Speicherstelle.**
NEG	Negiere den Inhalt des Akkumulators.
INC	Erhöhe den Inhalt des Akkumulators um 1.
DEC	Erniedrige den Inhalt des Akkumulators um 1.
OUT	Gib den Wert der angegebenen Speicherstelle an die Output-Einheit.
INM	Speichere den von der Input-Einheit gelesenen Wert in der angegebenen Speicherstelle.
END	Beende die Programmausführung.
DEF	Beispiel: Mit 34 DEF 3 erhält die Speicherstelle mit der Adresse 34 den Wert 3 zugewiesen.

Sprungbefehle

Mnemo	Bedeutung
JMP	Unbedingter Sprung
JMS	Sprung, falls Akkumulatorinhalt < 0
JPL	Sprung, falls Akkumulatorinhalt > 0
JZE	Sprung, falls Akkumulatorinhalt = 0
JNM	Sprung, falls Akkumulatorinhalt ≥ 0
JNP	Sprung, falls Akkumulatorinhalt ≤ 0
JNZ	Sprung, falls Akkumulatorinhalt ≠ 0