IN-G 2018 HT Aufg

Ministerium für Bildung und Kultur
SAARLAND

Schriftliche Abiturprüfung 2018

Fach:	**Informatik**
Kurstyp:	G-Kurs
Datum:	17. April 2018
Bearbeitungszeit:	3 Zeitstunden
Hilfsmittel:	nicht-programmierbarer Taschenrechner
Seitenzahl:	Die Prüfungsaufgabe umfasst mit Deckblatt und Anlagen 9 Seiten.

1. Aufgabe

1.1. Strukturierte Programmierung und Objektorientierung

Im neuen Onlinerollenspiel "World of Saarcraft" (kurz: WoS) kann ein Spieler epische Schlachten in einer fiktiven Spielwelt austragen.
Im Spiel existiert eine unzählige Anzahl von Charakteren. Diese lassen sich in von den Spielern gesteuerte Helden und von der Software gesteuerte Monster einteilen.

- Alle Charaktere haben die Attribute Leben, Rüstung, Stärke und Intelligenz, die alle Werte von 0 bis 99 annehmen können. Zu jedem Charakter gehört außerdem der Name der Rasse, zu der er gehört.
- Helden haben zusätzlich einen Namen und besitzen eine Tasche, in der sie bis zu zehn Gegenstände aufnehmen können.
- Die Tasche löst sich auf, sobald der Held stirbt und die in ihr enthaltenen Gegenstände bleiben zurück.
- Gegenstände haben einen Namen sowie ein Gewicht.
- Zu jedem Monster gibt es eine kurze textuelle Beschreibung seiner Geschichte (z.B. woher es stammt, wen es besiegt hat, …).

1.1.1. Entwerfen Sie ein UML-Diagramm, das die oben beschriebenen Klassen des Spiels WoS inklusive der vorhandenen Beziehungen darstellt.
Aus Gründen der Übersichtlichkeit soll im Diagramm auf die Angabe von Methoden verzichtet werden.

1.1.2. Implementieren Sie die Klasse Gegenstand inklusive des Konstruktors sowie aller Zugriffsmethoden.

1.1.3. Um im Spiel den Angriff eines Helden auf ein Monster zu realisieren, hat die Klasse Held eine Methode *angreifen()*.
Ein Angriff bewirkt, dass das Monster so viele Lebenspunkte verliert, wie die Stärke des Helden den Rüstungswert des Monsters übersteigt. Ist der Held schwächer als der Wert der Rüstung des Monsters passiert nichts. Die Lebenspunkte des Monsters können dabei nicht unter 0 fallen.
Implementieren Sie die Methode `angreifen(Monster monster)` in JAVA bzw. `angreifen(Monster:TMonster)` in Delphi.

1.2. Binäre Suchbäume

Aus dem Unterricht sind Ihnen die binären Suchbäume als Datenstruktur zur effektiven Suche auf sortierten Daten bekannt. Auch in „World of Saarcraft“ werden die IDs der Spieler mithilfe von binären Suchbäumen verwaltet.

1.2.1. Geben Sie die Höhe an, die ein binärer Suchbaum mit 130 Knoten mindestens hat.

1.2.2. Geben Sie den binären Suchbaum an, welcher durch das Einfügen der folgenden 10 Schlüssel (in der gegebenen Reihenfolge) in einen zunächst leeren Suchbaum entsteht und beurteilen Sie seine Effektivität als Suchbaum.

10, 30, 12, 11, 13, 25, 15, 16

1.2.3. Die Teilnehmer einer Schlacht in WoS haben alle IDs aus dem Bereich von 1 bis 1000 und sind in einem binären Suchbaum gespeichert. Es wird der Spieler mit der ID 367 gesucht.
Überprüfen Sie welche der folgenden Sequenzen die überprüfte Knotenfolge sein kann. Begründen Sie Ihre Antwort.

a) 487, 321, 128, 288, 320, 367
b) 18, 555, 388, 200, 290, 390
c) 988, 222, 300, 466, 400, 367

2. Aufgabe

Funktionsweise von Computersystemen / Automaten und Formale Sprachen

Die monatliche Rechnung für „World of Saarcraft“ kann auch mit Kreditkarte gezahlt werden. Ob die Kreditkartennummer gültig ist, wird u.a. mit Hilfe eines Prüfsummenverfahrens überprüft.

2.1. In dieser Aufgabe wird ein vereinfachtes Verfahren betrachtet. Die Kreditkartennummern bestehen aus zwei Ziffern von 0-9 und einer Prüfziffer, die sich nach folgender Vorschrift berechnen lässt:

Man bildet die Summe aus den beiden ersten Ziffern und berechnet den Rest bei der Division durch 3.

Alle Kreditkartennummern mit korrekter Prüfziffer bilden zusammen die Sprache K der Kreditkartennummern.

2.1.1. Berechnen Sie die Prüfziffer für die Kreditkartennummer 89.

2.1.2. Schreiben Sie ein Programm für den Didaktischen Computer (DC), welches eine zweistellige Kreditkartennummer ziffernweise einliest, die zugehörige Prüfziffer berechnet und diese ausgibt.
Hinweis: Der Befehlssatz für den Didaktischen Computer befindet sich im Anhang.

2.1.3. Entwerfen Sie einen deterministischen endlichen Automaten, der genau die Sprache K akzeptiert.
Es genügt die Angabe eines Übergangsgraphen.

2.1.4. Die Sprache K der Kreditkartennummern lässt sich auch durch folgende Grammatik $G = (N, T, P, S)$ erzeugen:

$$N = \{S, A, B, C, D, E, F\}$$

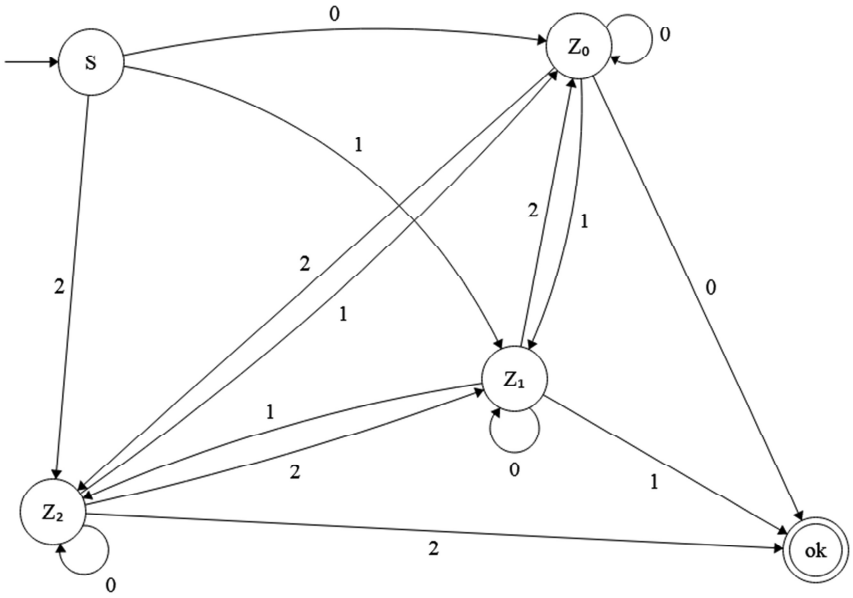
$$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$P = \left\{ \begin{array}{l} (1)\ S \rightarrow A0 \mid B1 \mid C2 \\ (2)\ A \rightarrow DD \mid EF \mid FE \\ (3)\ B \rightarrow FF \mid DE \mid ED \\ (4)\ C \rightarrow EE \mid DF \mid FD \\ (5)\ D \rightarrow 0 \mid 3 \mid 6 \mid 9 \\ (6)\ E \rightarrow 1 \mid 4 \mid 7 \\ (7)\ F \rightarrow 2 \mid 5 \mid 8 \end{array} \right\}$$

2.1.4.1. Prüfen Sie mittels Wortableitungen, ob die beiden Ziffernfolgen 641 und 830 zur Sprache K gehören.

2.1.4.2. Begründen Sie, weshalb die vorliegende Grammatik nicht regulär ist.

2.2. Der abgebildete nicht-deterministische endliche Automat akzeptiert alle Ziffernfolgen der Länge $n \geq 2$ über dem Alphabet $\Sigma = \{0,1,2\}$, bei denen die letzte Ziffer der Summe aller vorherigen Ziffern modulo 3 entspricht.



2.2.1. Geben Sie eine reguläre Grammatik für die vom Automat akzeptierte Sprache an.

2.2.2. Entwickeln Sie mit Hilfe der Teilmengenkonstruktion einen dazu äquivalenten deterministischen Automaten.
Hinweis: Es ist nicht erforderlich, einen Übergangsgraphen zu zeichnen.

3. Aufgabe

3.1. Kryptographie

Alice und Bob kommunizieren mit Hilfe der RSA-Verschlüsselung.
Dazu wollen Sie ihre Schlüssel erstellen.

3.1.1. Alice wählt ihren öffentlichen Schlüssel (e, n) mit den Primzahlen $p = 13$ und $q = 23$. Nur einer der drei folgenden Schlüssel (e, n) ist – ungeachtet der geringen Größe von n - im Sinne einer asymmetrischen Verschlüsselung mit RSA zulässig bzw. günstig. Begründen Sie Ihre Entscheidung und bestimmen Sie den zugehörigen privaten Schlüssel (d, n).

$(33, 299)$ $(43,299)$ $(53,299)$

3.1.2. Alice entscheidet sich, den öffentlichen Schlüssel $(7,299)$ und den privaten Schlüssel $(151,299)$ zu verwenden.

3.1.2.1. Bob möchte die Nachricht $m = 2018$ an Alice senden. Begründen Sie, warum die Nachricht m mit den gewählten Schlüsseln nicht verschlüsselt werden sollte.

3.1.2.2. Verschlüsseln Sie die Nachricht $m' = 13$ und geben Sie den Geheimtext c' an.

3.1.2.3. Geben Sie die Gleichung zur Entschlüsselung der Nachricht $c' = 78$ für Alice an.

3.1.3. Erläutern Sie, wie Alice und Bob mit Hilfe der RSA-Verschlüsselung die drei wesentlichen Sicherheitsziele Vertraulichkeit, Authentizität und Integrität erreichen können.

3.2 Rekursion – Stern-Brocot-Folge

Die Stern-Brocot-Folge ist eine Folge ganzer Zahlen, die der Mathematiker Moritz Stern und der Uhrmacher Achille Brocot unabhängig voneinander entdeckten. Die Folge bildet unter anderem die Grundlage des Stern-Brocot-Baums zur Abzählung rationaler Zahlen und dient bei Uhren dazu, optimale Übersetzungsverhältnisse bei Zahnrädern zu modellieren.
Die Folge beginnt mit 0 und 1. Das Glied mit dem Index n ist gleich dem mit dem Index $\frac{n}{2}$, wenn n gerade ist. Ein Folgenglied mit ungeradem Index n berechnet man als Summe der Folgenglieder mit den Indizes $\frac{n-1}{2}$ bzw. $\frac{n-1}{2}+1$.

Also:

$$S_n = \begin{cases} 0 & \text{für n} = 0 \\ 1 & \text{für n} = 1 \\ S_{\frac{n}{2}} & \text{für n} > 0 \text{ gerade} \\ S_{\frac{n-1}{2}} + S_{\frac{n-1}{2}+1} & \text{für n} > 1 \text{ ungerade} \end{cases}$$

3.2.1 In der folgenden Tabelle sind die ersten acht Folgenglieder eingetragen. Ergänzen Sie die Werte bis zum 16. Folgenglied.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_n	0	1	1	2	1	3	2	3								

3.2.2 Programmieren Sie eine Funktion Stern_Brocot, die nach Eingabe eines ganzzahligen Indexes n größer gleich Null den Wert des entsprechenden Folgengliedes S_n berechnet.

3.2.3 Die Folgenglieder lassen sich unter Weglassung von S_0 in einer Stufentabelle notieren und zwar so, dass bei jeder Zweierpotenz eine neue Tabellenzeile beginnt.

Zeile										Summe
k=0	$n = 2^0$	1								1
k=1	$n = 2^1$	1	2							3
k=2	$n = 2^2$	1	3	2	3					9
k=3	$n = 2^3$									
…										

Es stellt sich heraus, dass die Summe der Zahlen in der k-ten Zeile genau 3^k ergibt.

3.2.3.1 Überprüfen Sie diesen Sachverhalt für die vierte Zeile.

3.2.3.2 Programmieren Sie eine Funktion `boolean istDreierpotenz(int k)` bzw. `istDreierpotenz(k:integer):boolean`, die diesen Sachverhalt für die k-te Zeile verifiziert.

Bei der Programmierung können die Hilfsfunktion *power* bzw. *Math.pow* sowie die Funktion aus 3.2.2. benutzt werden.

Delphi: `power(basis,exponent:integer):integer;`

Java: `int Math.pow(int basis,int exponent);`

Anhang (zu Aufgabe 3.2): Befehlssatz des *DC*

a) Grundbefehle

Mnemo	Bedeutung
LDA	LOAD INTO ACCUMULATOR - Lade den Wert der angegebenen Speicherstelle in den Akkumulator.
STA	STORE ACCUMULATOR TO MEMORY - Speichere den Inhalt des Akkumulators an der angegebenen Speicherstelle ab.
ADD	ADD TO ACCUMULATOR - Addiere den Wert der angegebenen Speicherstelle zum Inhalt des Akkumulators.
SUB	SUBTRACT FROM ACCUMULATOR - Subtrahiere den Wert der angegebenen Speicherstelle vom Inhalt des Akkumulators.
NEG	NEGATE ACCUMULATOR - Negiere den Inhalt des Akkumulators.
INC	INCREMENT ACCUMULATOR - Erhöhe den Inhalt des Akkumulators um 1.
DEC	DECREMENT ACCUMULATOR - Erniedrige den Inhalt des Akkumulators um 1.
OUT	OUTPUT MEMORY - Gib den Wert der angegebenen Speicherstelle an die Output-Einheit. Die auszugebende Zahl erscheint in einer Zeile oberhalb des Eingabe-Fensters.
INM	INPUT TO MEMORY - Speichere die von der Input-Einheit gelesene Zahl an der angegebenen Adresse ab. Das Programm hält bei diesem Befehl an und wartet auf die Eingabe einer Zahl.
END	ENDE - Programm beenden.
DEF	DEFINE word - Beispiel: Mit `34 DEF 3` erhält die Speicherstelle mit der Adresse 34 den Wert 3 zugewiesen. Dies ist keine vom Mikroprozessor ausführbare Anweisung, sondern dient nur der Wertbelegung von Speicherstellen beim Einlesen eines DC-Programmes oder im Direktmodus.

b) Sprungbefehle

Mnemo	Bedeutung
JMP	JUMP - Unbedingter Sprung. Springe zur angegebenen Speicherstelle und fahre mit dem dort stehenden Befehl fort.
JMS	JUMP IF MINUS - Springe zur angegebenen Speicherstelle und fahre mit dem dort stehenden Befehl fort, wenn der Inhalt der Inhalt des Akkumulators negativ ist. Wenn nicht, dann fahre mit dem nächsten Befehl fort. Sprung, falls Akkumulatorinhalt < 0
JPL	JUMP IF PLUS - Sprung, falls Akkumulatorinhalt > 0
JZE	JUMP IF ZERO - Sprung, falls Akkumulatorinhalt = 0
JNM	JUMP IF NOT MINUS - Sprung, falls Akkumulatorinhalt ≥ 0
JNP	JUMP IF NOT PLUS - Sprung, falls Akkumulatorinhalt ≤ 0
JNZ	JUMP IF NOT ZERO - Sprung, falls Akkumulatorinhalt ≠ 0