Ministerium für Bildung und Kultur
SAARLAND

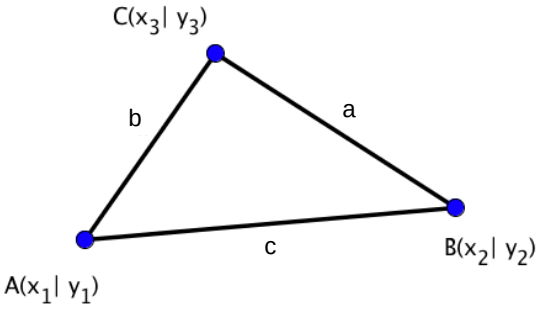
Schriftliche Abiturprüfung 2015

Fach:	**Informatik**
Kurstyp:	G-Kurs
Datum:	24. April 2015
Bearbeitungszeit:	3 Zeitstunden
Hilfsmittel:	nicht-programmierbarer Taschenrechner
Seitenzahl:	Die Prüfungsaufgabe umfasst mit Deckblatt und Anlagen 8 Seiten.

1. Aufgabe

1.1 Strukturierte Programmierung und Objektorientierung

Ein Dreieck besteht aus drei Eckpunkten (mit reellwertigen Koordinaten), die miteinander verbunden sind.



Der Abstand zwischen zwei Eckpunkten ergibt die jeweilige Seitenlänge eines Dreiecks (siehe Hinweise). Der Flächeninhalt eines Dreiecks wird durch das halbe Produkt von Grundseite und zugeordneter Höhe bestimmt. Sind die einander entsprechenden Seiten zweier Dreiecke gleich lang, so sind die Dreiecke nach dem Kongruenzsatz „SSS“ zueinander kongruent. Zur Überprüfung der Kongruenzbeziehung sollen die drei Methoden `gibGroessteSeitenlaenge()`, `gibMittlereSeitenlaenge()` und `gibKleinsteSeitenlaenge()` in der Klasse `TDreieck` zur Verfügung stehen.

1.1.1 Stellen Sie eine Klasse `TDreieck` in Form eines Klassendiagramms dar.

1.1.2 Implementieren Sie die Methode `gibSeitenlaenge` (siehe Hinweise).

1.1.3 Implementieren Sie die Methode, die überprüft, ob ein Dreieck zu einem anderen Dreieck kongruent ist. Sie können dabei auf die anderen Methoden der Klasse `TDreieck` zurückgreifen.

Hinweise:

- Der Abstand *d* zweier Punkte $P(x_1 \mid y_1)$ und $Q(x_2 \mid y_2)$ ergibt sich aus:
  $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
  Verwenden Sie die Funktion / Methode `sqrt(x: real):real` in Delphi/Pascal bzw. `double Math.sqrt(double x)` in Java zur Berechnung der Quadratwurzel einer Zahl x.

- Die Methode `gibSeitenlaenge` erhält als Parameter eine char-Variable, die zur Beschreibung einer Dreiecksseite in der Form 'a', 'b' oder 'c' dienen soll. In Abhängigkeit von dem Parameter wird die zugehörige Seitenlänge des Dreiecks berechnet und zurückgegeben.

1.2 Sortierverfahren: Insertionsort und Ripplesort

18	7	5	41	32	0

1.2.1 Sortieren Sie die oben angegebene Ausgangsanordnung nach dem Insertionsort-Verfahren.
Stellen Sie den Zustand nach jedem Durchlauf in der vorgegebenen Tabelle auf dem Blatt dar.

Hinweise:

- Nach jedem Durchlauf soll eine weitere Zahl richtig eingeordnet sein.
- Die vorgegebene Tabelle enthält unter Umständen mehr Zeilen als notwendig sind. Entwerten Sie die nicht benötigten Zeilen.
- Zwischenschritte können auf einem gesonderten Blatt notiert werden.

1.2.2 Nennen Sie die Laufzeitklasse von Insertionsort sowie ein Sortierverfahren mit Angabe der Laufzeitklasse, welches im Allgemeinen eine bessere Laufzeit als Insertionsort besitzt.

Ripplesort

Beim Durchlaufen des Feldes wird das erste Element nacheinander mit allen nachfolgenden Elementen des Feldes verglichen. Ist das zum Vergleich herangezogene Element kleiner als das erste, so werden sie jeweils getauscht. Sind alle Vergleiche abgeschlossen, steht das kleinste Element an der ersten Stelle. Nun wird die Prozedur mit dem zweiten Element wiederholt, dann mit dem dritten usw.

18	7	5	41	32	0

1.2.3 Sortieren Sie die oben angegebene Ausgangsanordnung nach dem Ripplesort-Verfahren unter Beachtung der obigen Hinweise.

1.2.4 Schätzen Sie begründet die Laufzeit von Ripplesort im schlechtesten Fall ab und geben Sie diese in O-Notation an. Bei der Untersuchung des Laufzeitverhaltens sollen nur die notwendigen Vergleiche von Elementen herangezogen werden.

2. Aufgabe

Thue-Morse-Folge

Die Thue-Morse-Folge wurde von verschiedenen Mathematikern unabhängig voneinander beschrieben. Die Folgenglieder sind Wörter aus Nullen und Einsen. Die Folge wird nach den folgenden Regeln gebildet:

- Das erste Folgenglied ist 0.
- Ist w ein beliebiges Folgenglied, so ist sein Nachfolger durch ww‘ gegeben, wobei w‘ die Negation von w ist, welche durch Ersetzen von 1 durch 0 und umgekehrt entsteht.

Die ersten vier Folgenglieder lauten demnach:

- $t_1 = 0$
- $t_2 = 01$
- $t_3 = 0110$
- $t_4 = 01101001$

2.1 Rekursion

2.1.1 Geben Sie die nächsten beiden Folgenglieder t_5 und t_6 an.

2.1.2 Schreiben Sie eine Delphi/Pascal-Funktion oder eine Java-Methode `thuemorseIt`, welche das n-te Folgenglied der Thue-Morse-Folge iterativ berechnet (siehe Hinweis).

2.1.3 Schreiben Sie eine Delphi/Pascal-Funktion oder eine Java-Methode `thuemorseRek`, welche das n-te Folgenglied der Thue-Morse-Folge rekursiv berechnet (siehe Hinweis).

Hinweis:

Sie dürfen die Delphi-Funktion `negiere(s:String):String` bzw. die Java-Methode `String negiere(String s)` benutzen, welche die Negation w‘ eines Wortes w berechnet.

2.2 Automaten und formale Sprachen

Die Glieder der Thue-Morse-Folge sind *kubikfrei*, d.h. kein Teil des Worts wiederholt sich mehr als zweimal direkt hintereinander.
Eine schwächere Variante dieser Eigenschaft lautet *schwach kubikfrei.* Diese Eigenschaft fordert lediglich, dass kein einzelnes Zeichen mehr als zweimal direkt hintereinander vorkommt.

Gegeben ist die Grammatik G = (N, T, P, S), welche die Sprache L der schwach kubikfreien Wörter über dem Alphabet ∑ = {0,1} erzeugt:

N = {S, A, B, C, D}

T = {0, 1}

P = { (1) S → 1A | 0B | ε ,
(2) A → 1C | 0B | ε ,
(3) B → 0D | 1A | ε ,
(4) C → 0B | ε ,
(5) D → 1A | ε }

2.2.1 Zeigen Sie durch Ableitung, dass die ersten drei Glieder der Thue-Morse-Folge zur Sprache L gehören.

2.2.2 Geben Sie die vollständige Beschreibung eines deterministischen endlichen Automaten an, welcher genau die Sprache L akzeptiert.

2.2.3 Überprüfen Sie, ob die beiden folgenden regulären Ausdrücke

$$\alpha = (0 \cdot 1)^* \mid (1 \cdot 0)^* \quad \text{und} \quad \beta = ((0 \cdot 1) \mid (1 \cdot 0))^*$$

dieselbe schwach kubikfreie Sprache beschreiben.
Falls ja, beschreiben Sie die Sprache mit Worten, falls nein, geben Sie ein Wort an, das nur in einer der beschriebenen Sprachen enthalten ist.

Hinweis:

Seien a und b reguläre Ausdrücke, so gilt:
- a*: Iteration von a,
- a · b: Konkatenation von a und b,
- a | b: Auswahl zwischen a und b

2.2.4 Begründen Sie, weshalb es einen regulären Ausdruck geben muss, welcher die gegebene Sprache L beschreibt.

2.2.5 Besonders bemerkenswert sind die Glieder der Thue-Morse Folge mit ungeradem n (t_1, t_3, t_5, …), da es sich hierbei um Palindrome über dem Alphabet ∑ = {0,1} handelt (siehe Hinweis). Geben Sie eine Grammatik an, welche die Sprache aller schwach kubikfreien Palindrome erzeugt und geben Sie begründet an, ob es sich dabei um eine reguläre Sprache handelt.

Hinweis:

Ein Palindrom ist eine Zeichenkette, die von vorn und von hinten gelesen dasselbe ergibt.

3. Aufgabe

3.1 Kryptographie – RSA-Verfahren

Geheimagent Willi hat von der Sicherheit des RSA-Verfahrens gehört und möchte dieses Verfahren nun zur sicheren Übermittlung von Nachrichten verwenden.

3.1.1 Beim letzten Treffen mit seinem Partner Eddie hat er diesem den öffentlichen Schlüssel (5,77) mitgeteilt, den er mit Hilfe der Primzahlen p = 7 und q = 11 erstellt hat.
Leider kann Willi die Nachrichten seines Partners nicht entschlüsseln, da er schon bei der Ermittlung des zugehörigen privaten Schlüssels scheitert.
Erklären Sie, welchen Fehler er bei seiner Wahl des öffentlichen Schlüssels begangen hat.

3.1.2 Willi versucht es nun erneut. Diesmal wählt er als Primzahlen p = 5 und q = 11 sowie e = 7. Ermitteln Sie den privaten Schlüssel (d,n).

3.1.3 Willis Partner Eddie verschlüsselt nun Zeichen für Zeichen seiner Nachricht mit dem öffentlichen Schlüssel (7,55), indem er zuerst jeden Buchstaben durch seine Positionsnummer im deutschen Alphabet ersetzt und auf die jeweils erhaltene Zahl das RSA-Verfahren anwendet. Ermitteln Sie das Ergebnis, das Eddie für das Wort BEE erhält.

3.1.4 Als der Chef des Geheimdienstes erfährt, wie Eddie und Willi Nachrichten verschlüsseln, lässt er ein großes Donnerwetter los:

„Das Verfahren, das ihr da benutzt habt, kann jeder Hobby-Kryptologe im Handumdrehen knacken, selbst wenn ihr noch so große Primzahlen verwendet!“

Nehmen Sie begründet Stellung zu dieser Aussage.

3.2 Der didaktische Computer (DC)

(Hinweis: Der Befehlssatz des DC befindet sich im Anhang.)

3.2.1 Betrachten Sie folgendes DC-Programm:

```
0   JMP  3
1   DEF  0
2   DEF  0
3   INM  1
4   LDA  1
5   JZE  12
6   LDA  2
7   SUB  1
8   JPL  11
9   LDA  1
10  STA  2
11  JMP  3
12  OUT  2
13  END
```

Bestimmen Sie die Ausgabe des Programms, wenn man nacheinander die Zahlen 7, 13, 21, 5, 0 eingibt! Notieren Sie hierzu den Inhalt der Speicherstellen 1 und 2 während des Programmablaufs jeweils vor dem Einlesen der nächsten Zahl der Folge! Welche Ausgabe liefert das Programm bei der Eingabe einer Folge von natürlichen Zahlen, die mit einer 0 endet?

3.2.2 Implementieren und kommentieren Sie ein DC-Programm, das eine positive ganze Zahl n>2 einliest und die Summe aller geraden Zahlen von 2 bis zur eingegebenen Zahl n ausgibt.

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