IN-G 2015 NT

Ministerium für Bildung und Kultur
SAARLAND

Schriftliche Abiturprüfung 2015

Fach:	**Informatik**
Kurstyp:	G-Kurs
Datum:	19. Mai 2015
Bearbeitungszeit:	3 Zeitstunden
Hilfsmittel:	nicht-programmierbarer Taschenrechner
Seitenzahl:	Die Prüfungsaufgabe umfasst mit Deckblatt und Anlagen 8 Seiten.

1. Aufgabe

1.1 Rekursion

Die **Ackermannfunktion** ist eine 1926 von Wilhelm Ackermann gefundene, extrem schnell wachsende mathematische Funktion, mit deren Hilfe in der theoretischen Informatik Grenzen von Computer- und Berechnungsmodellen aufgezeigt werden können.

Die Ackermannfunktion kann wie folgt definiert werden:

$A(0, n) = n + 1,$

$A(m, 0) = A(m - 1, 1)$ für $m > 0,$

$A(m, n) = A(m - 1, A(m, n - 1))$ für $m, n > 0,$

für n, m aus den natürlichen Zahlen.

1.1.1 Berechnen Sie A(1,0) und A(1,3).

1.1.2 Implementieren Sie eine Methode `ack`, die die Ackermannfunktion nach der obigen rekursiven Definition berechnet.

1.1.3 Die Ackermannfunktion gehört zur Klasse der berechenbaren Funktionen. Nennen und erläutern Sie ein nicht berechenbares Problem.

1.2 Objektorientiertes Programmieren

Im Mathematikunterricht haben Sie die Menge R der reellen Zahlen kennengelernt. Über diese Menge hinaus gibt es noch die Menge C der komplexen Zahlen, welche in vielen wissenschaftlichen Bereichen ihre Anwendung finden.

Jede komplexe Zahl *z = (a, b)* wird durch ein Paar reeller Zahlen a und b dargestellt.

Die Zahl a heißt *Realteil* von z, die Zahl b heißt *Imaginärteil* von z.

In der Menge C gelten u.a. folgende Rechenregeln:

Die Zahlen z_1 und z_2 seien beliebige komplexe Zahlen mit $z_1 = (a_1, b_1)$ und $z_2 = (a_2, b_2)$.

- Addition: $z_1 + z_2 = (a_1 + a_2\ ,\ b_1 + b_2)$
- Multiplikation: $z_1 \cdot z_2 = (a_1 \cdot a_2 - b_1 \cdot b_2\ ,\ a_1 \cdot b_2 + b_1 \cdot a_2)$
- Betrag: $|z_1| = \sqrt{a_1^{\,2} + b_1^{\,2}}$

Beachten Sie: Ähnlich wie in der Vektorrechnung ist das Ergebnis der Addition bzw. Multiplikation zweier komplexer Zahlen wieder eine komplexe Zahl, während der Betrag einer komplexen Zahl eine reelle Zahl ist.

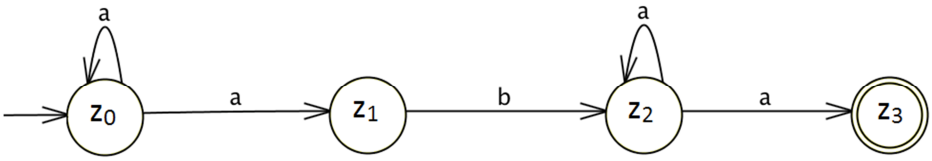
1.2.1 Entwerfen Sie ein Klassendiagramm für die Klasse `TKomplex`, welche die komplexen Zahlen zusammen mit ihrer Addition und Multiplikation darstellt. Die komplexe Zahl wird dabei durch ihren Realteil und ihren Imaginärteil repräsentiert. Ferner soll es für jede komplexe Zahl möglich sein, den Betrag sowie den Realteil oder den Imaginärteil abzufragen.

1.2.2 Implementieren Sie einen Konstruktor für die Klasse `TKomplex`, der den Realteil *a* und den Imaginärteil *b* als Parameter erhält.

1.2.3 Implementieren Sie die Methoden für die Multiplikation und den Betrag komplexer Zahlen. Sie können dabei auf die anderen Methoden der Klasse `TKomplex` zurückgreifen.

2. Aufgabe

2.1 Gegeben ist der Übergangsgraph eines endlichen Automaten:



2.1.1 Geben Sie begründet an, welche der folgenden Wörter vom Automaten akzeptiert werden.

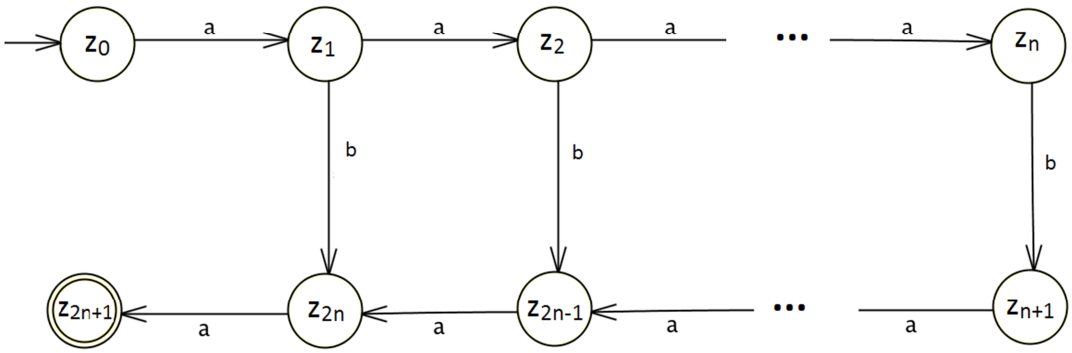
- aaab
- aabbaa
- abaa
- baaaa

2.1.2 Geben Sie einen regulären Ausdruck an, der die gleiche Sprache beschreibt.

2.1.3 Begründen Sie, dass der Automat nichtdeterministisch ist.

2.1.4 Konstruieren Sie den Übergangsgraph eines deterministischen endlichen Automaten, der die gleiche Sprache akzeptiert.

2.2 Nach folgendem Schema lässt sich für jede vorgegebene natürliche Zahl n ein endlicher Automat konstruieren, der eine Teilmenge der Wörter der Sprache aus 2.1. akzeptiert.



Hierbei ist die Anzahl der a's vor und nach dem b gleich und auf maximal n begrenzt. Hebt man diese Begrenzung auf (unter Beibehaltung der Gleichheitsbedingung), so ergibt sich eine neue Sprache L.

2.2.1 Begründen Sie, dass diese Sprache L nicht regulär ist.

2.2.2 Geben Sie eine Grammatik für die Sprache L an.

2.3 Binäre Bäume

Mit Hilfe eines geordneten binären Baumes ("binären Suchbaumes") kann man wie folgt ein Feld sortieren: Man fügt der Reihe nach alle Feldelemente in den zunächst leeren Suchbaum ein und gibt dann die Elemente des Baumes in der Inorder-Reihenfolge wieder in das Feld aus.

2.3.1 Zeichnen Sie denjenigen binären Suchbaum, der entsteht, wenn folgendes Feld auf die oben beschriebene Art sortiert wird: 3, 7, 4, 2, 1, 8, 6, 5, 9

2.3.2 Ein Feld mit 1000 Zahlen wird mit Hilfe eines Baumes wie beschrieben sortiert. Geben Sie die minimale und maximale Höhe des dabei entstehenden Baumes an und begründen Sie ihre Antwort!

2.3.3 Schreiben Sie die Knoten des Baumes aus 2.3.1 in Preorder-Reihenfolge auf!

2.3.4 Löschen Sie nacheinander im Baum aus 2.3.1 die Knoten mit dem Inhalt 5 und dem Inhalt 7. Beschreiben Sie kurz Ihre Vorgehensweise.

2.3.5 Ein binärer Baum, der nicht geordnet ist, wurde in Postorder- und in Inorder-Reihenfolge durchlaufen:
Postorder: 9, 3, 1, 8, 6, 2, 5, 7, 4
Inorder: 3, 9, 8, 1, 4, 6, 2, 7, 5
Zeichnen Sie diesen Baum!

3. Aufgabe

3.1 Kryptographie – RSA-Verfahren

3.1.1 Berechnen Sie mit Hilfe des euklidischen Algorithmus das modulare Inverse von 29 modulo 192, also 29^{-1} mod 192!

3.1.2 Verschlüsseln Sie mit dem RSA-Verfahren die Nachricht m = 2 mit dem öffentlichen Schlüssel (e, n) = (29, 221)!

3.1.3 Zeigen Sie, dass das RSA-Verfahren mit dem öffentlichen Schlüssel (e, n) = (29, 221) relativ einfach geknackt werden kann. Geben Sie den geheimen Schlüssel (d, n) an. Erläutern Sie, wieso das RSA-Verfahren trotzdem ein sicheres Verschlüsselungsverfahren ist.

3.1.4 Eine zweite Nachricht wurde ebenfalls mit dem Schlüssel (e, n) = (29, 221) verschlüsselt. Der Geheimtext ist c = 172. Geben Sie die Rechnung an - ohne sie durchzuführen - die der Empfänger zur Entschlüsselung durchführen müsste!

3.1.5 Erläutern Sie die Erzeugung und die Verifikation einer digitalen Signatur mit Hilfe des RSA-Verfahrens.

3.2 Der didaktische Computer (DC)

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

3.2.1 Implementieren und kommentieren Sie ein DC-Programm, das eine Zahl x>0 einliest und dann die ersten x Quadratzahlen ausgibt.

Beispiel: Eingabe: 5
Ausgabe: 1, 4, 9, 16, 25

Verwenden Sie folgenden Hinweis: Mit Hilfe der ersten binomischen Formel kann man leicht zur nächsten Quadratzahl kommen. Es gilt: $(k+1)^2 = k^2 + 2k + 1$
Hierbei ist k^2 die zuvor berechnete Quadratzahl und $(k+1)^2$ die nächste.

3.2.2 Bestimmen Sie die Ausgabe des nachfolgenden DC-Assemblerprogramms bei den Eingaben 4 und 3! Welche Aufgabe erfüllt das Programm? Welche Bedingung muss hierfür an `zahl1` oder `zahl2` gestellt werden?

```
jmp         einlesen
zahl1       DEF       0
zahl2       DEF       0
erg         DEF       0
einlesen:   INM       zahl1
            INM       zahl2
ablauf:     LDA       zahl2
            JZE       ende
            LDA       erg
            ADD       zahl1
            STA       erg
            LDA       zahl2
            DEC
            STA       zahl2
            JMP       ablauf
ende:       OUT       erg
            END
```

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