18. Informatik – Thematische Schwerpunkte für die schriftliche Abiturprüfung 2020

A. Fachbezogene Hinweise

Grundlage der schriftlichen Abiturprüfung in Niedersachsen sind die geltenden Einheitlichen Prüfungsanforderungen für die Abiturprüfung Informatik (EPA, 2004) sowie die Rahmenrichtlinien für das Gymnasium – gymnasiale Oberstufe, die Gesamtschule – gymnasiale Oberstufe, das Fachgymnasium[1], das Abendgymnasium, das Kolleg Informatik (RRL, 1993).

Die Rahmenrichtlinien Informatik sind so offen formuliert, dass sie Raum für die Gestaltung eines zeitgemäßen Informatikunterrichts lassen. Neue Inhalte der Informatik lassen sich unter die vorgegebenen Unterrichtsinhalte subsumieren. So findet sich in den Rahmenrichtlinien (RRL) z. B. zwar nicht der Begriff „Internet". Ein Informatikunterricht, in dem das Internet nicht an geeigneten Stellen thematisch Niederschlag findet, ist heute jedoch kaum vorstellbar.

Für die Thematischen Schwerpunkte des Zentralabiturs ergeben sich deshalb die folgenden Konsequenzen:

- Die für die Abiturprüfung verpflichtenden Kerninhalte der RRL und der Einheitlichen Prüfungsanforderungen in der Abiturprüfung Informatik (EPA) bilden die Grundlage für die Aufgabenstellungen des Zentralabiturs.
- Zeitgemäße Abituraufgaben können sich nicht auf in den RRL explizit genannte Inhalte beschränken (vgl. „Internet").
- Die vorliegenden Thematischen Schwerpunkte beschreiben den stofflichen Umfang der Aufgaben des Zentralabiturs 2020. Sie sollen die Inhalte eines zeitgemäßen Informatikunterrichts widerspiegeln, sind aber nicht so angelegt, dass dadurch die in der Qualifikationsphase zur Verfügung stehende Unterrichtszeit vollständig ausgefüllt wird.

Für den Unterricht auf erhöhtem Anforderungsniveau werden in den jeweiligen Themenbereichen Ergänzungen angegeben, die zusätzlich zu den genannten Themen zu behandeln sind.

B. Thematische Schwerpunkte

Thematischer Schwerpunkt 1: Anwendung von Hard- und Softwaresystemen sowie deren gesellschaftliche Auswirkungen

Chiffrieren und Codieren:

- Kryptografische Verfahren
  - monoalphabetische Verfahren (u. a. Caesar-Verfahren), polyalphabetische Verfahren (u. a. Vigenère-Verfahren)
  - Kryptoanalyse monoalphabetischer und polyalphabetischer Verfahren (u. a. Häufigkeitsanalyse und Kasiskitest)
  - Implementierung klassischer Verschlüsselungsverfahren
- Codierung
  - fehlererkennende und fehlerkorrigierende Codes (u. a. Paritätsbit, (7,4)-Hamming-Code)

Datenschutz und Datensicherheit:

- Erläuterung grundlegender Begriffe im Kontext der informationellen Selbstbestimmung

<u>Ergänzung für Kurse auf erhöhtem Anforderungsniveau</u>

Praktische Einsatzgebiete von Verschlüsselungsverfahren:

- Geheimhaltung, Authentifikation, Integrität

Prinzipien der Anwendung von asymmetrischen Verfahren (digitale Signatur und Zertifikat, hybride Verschlüsselung)

[1] jetzt: Berufliches Gymnasium mit gesonderten Hinweisen für das Fach Informationsverarbeitung

Thematischer Schwerpunkt 2: Werkzeuge und Methoden der Informatik

Algorithmen (auch rekursive):
- Erstellung eines Algorithmus in schriftlich verbalisierter Form und als Struktogramm
- Bearbeitung eines Algorithmus, gegeben durch ein Struktogramm oder in schriftlich verbalisierter Form
  - Analyse, u. a. mit einer Tracetabelle, durch Auswahl geeigneter Testdaten
  - Vervollständigung
  - Präzisierung
  - Korrektur
- Strukturierte Datentypen (u. a. ein- und zweidimensionale Reihungen)
- Implementierung eines Algorithmus in Java oder einer vergleichbaren Programmiersprache

Objektorientierte Modellierung:
- Klassendiagramme (Vererbung, Assoziation)
- Anwendung der Klassen (ADTs) „Schlange“, „Stapel“ und „Dynamische Reihung"

Ergänzung für Kurse auf erhöhtem Anforderungsniveau
- Abschätzen der Komplexität eines Algorithmus
- Anwendung der Klasse (ADT) „Binärbaum“

Thematischer Schwerpunkt 3: Funktionsprinzipien von Hard- und Softwaresystemen einschließlich theoretischer bzw. technischer Modellvorstellungen

Endliche Automaten mit und ohne Ausgabe:
- Analyse und Synthese von deterministischen und von nichtdeterministischen endlichen Automaten und von Mealy-Automaten
  - Entwicklung eines Zustandsgraphen für ein gegebenes Problem
  - Analyse eines gegebenen Zustandsgraphen
  - Erweiterung eines gegebenen Zustandsgraphen
  - Grenzen endlicher Automaten bei der Problemlösung

Ergänzung für Kurse auf erhöhtem Anforderungsniveau

Reguläre Sprachen:
- Analyse einer gegebenen Sprache
- Entwicklung einer Sprache für ein gegebenes Problem
- Umsetzung eines Zustandsgraphen in eine Grammatik und umgekehrt
- Grenzen regulärer Grammatiken bei der Problemlösung
- Syntaxdiagramme

C. Sonstige Hinweise

- Die Aufgabentexte selber enthalten keinen Code in einer konkreten Programmiersprache. Diejenigen Aufgabenteile, die die Implementierung in einer konkreten Programmiersprache erfordern, sind von den Schülerinnen und Schülern in Java oder einer vergleichbaren Programmiersprache zu bearbeiten.
- Anstelle der unterschiedlichen, sprachspezifischen Bezeichnungen „Prozedur“, „Funktion“ bzw. „Methode“ wird in den Aufgabenstellungen der Begriff „Operation“ verwendet.
- Zur Vereinfachung sind beim Mealy-Automaten bei einem Zustandsübergang statt eines einzelnen Ausgabezeichens auch Wörter zulässig, die aus dem Ausgabealphabet gebildet werden.
- Aufgaben, die am Rechner zu bearbeiten sind, werden nicht gestellt.
- Das Lehrermaterial wird ausführliche Lösungsskizzen enthalten. Die Implementierungen in einer Programmiersprache werden nur in Java vorgelegt.
- Die Anlage (Operationen der Klassen Dynamische Reihung, Stapel, Schlange und Binärbaum) ist als Hilfsmittel in der Abiturprüfung zugelassen.

Anlage

1. Operationen der Klassen Dynamische Reihung, Stapel, Schlange und Binärbaum

Die Klassen verwenden Inhaltsklassen bzw. -typen, die jeweils der aktuellen Aufgabenstellung angepasst werden. Die Klassen werden in den Aufgabenstellungen gegebenenfalls um Attribute und weitere Operationen ergänzt. Mögliche Laufzeitfehler bei der Anwendung der Operationen, z. B. Entnehmen bei einem leeren Stapel oder Zugriff auf eine nicht existierende Position einer dynamischen Reihung, müssen bei der Bearbeitung entsprechender Aufgaben explizit abgefangen werden.

Dynamische Reihung

Die Nummerierung der Elemente der dynamischen Reihung beginnt mit dem Index 1.

`DynArray()`
Eine leere dynamische Reihung wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn die Reihung kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getItem(index: Ganzzahl): Inhalt`
Der Inhalt des Elements an der Position index wird zurückgegeben.

`append(i: Inhalt)`
Der übergebene Inhalt wird als neues Element am Ende der dynamischen Reihung angefügt.

`insertAt(index: Ganzzahl, i: Inhalt)`
Der Inhalt wird als neues Element an der Position index in die dynamische Reihung eingefügt. Das Element, das sich vorher an dieser Position befunden hat, und alle weiteren werden nach hinten verschoben. Entspricht index der Länge der dynamischen Reihung +1, so wird ein neues Element am Ende der dynamischen Reihung angefügt.

`setItem(index: Ganzzahl, i: Inhalt)`
Der Inhalt des Elementes an der Position index wird durch den übergebenen Inhalt ersetzt.

`delete(index: Ganzzahl)`
Das Element an der Position index wird entfernt. Alle folgenden Elemente werden um eine Position nach vorne geschoben.

`getLength(): Ganzzahl`
Die Anzahl der Elemente der dynamischen Reihung wird zurückgegeben.

Stapel

`Stack()`
Ein leerer Stapel wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn der Stapel kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`top(): Inhalt`
Der Inhalt des obersten Elements des Stapels wird zurückgegeben, das Element aber nicht entfernt.

`push(i: Inhalt)`
Ein neues Element mit dem angegebenen Inhalt wird auf den Stapel gelegt.

`pop(): Inhalt`
Der Inhalt des obersten Elements wird zurückgegeben und das Element wird aus dem Stapel entfernt.

Schlange

`Queue()`
Eine leere Schlange wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn die Schlange kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`head(): Inhalt`
Der Inhalt des ersten Elements der Schlange wird zurückgegeben, das Element aber nicht entfernt.

`enqueue(i: Inhalt)`
Ein neues Element mit dem angegebenen Inhalt wird angelegt und am Ende an die Schlange angehängt.

`dequeue(): Inhalt`
Der Inhalt des ersten Elements wird zurückgegeben und das Element wird aus der Schlange entfernt.

Binärbaum

`BinTree()`
Ein leerer Baum wird erzeugt.

`BinTree(i: Inhalt)`
Ein Baum wird erzeugt. Die Wurzel erhält den übergebenen Inhalt als Wert.

`isEmpty(): Wahrheitswert`
Wenn der Baum kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`isLeaf(): Wahrheitswert`
Wenn der Baum keine Teilbäume besitzt, die Wurzel des Baums also ein Blatt ist, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*. Diese Abfrage ist nur für nicht leere Bäume sinnvoll.

`getItem(): Inhalt`
Die Operation gibt den Inhaltswert der Wurzel des Baumes zurück.

`setItem(i: Inhalt)`
Die Operation setzt den Inhaltswert der Wurzel des aktuellen Baumes. Ist der Baum leer, so wird zunächst die Wurzel erzeugt und der übergebene Inhalt als Wert gesetzt.

`hasLeft(): Wahrheitswert`
Wenn der Baum einen linken Teilbaum besitzt, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`hasRight(): Wahrheitswert`
Wenn der Baum einen rechten Teilbaum besitzt, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getLeft(): BinTree`
Die Operation gibt den linken Teilbaum zurück.

`getRight(): BinTree`
Die Operation gibt den rechten Teilbaum zurück.

`setLeft(b: BinTree)`
Der übergebene Baum wird als linker Teilbaum gesetzt.

`setRight(b: BinTree)`
Der übergebene Baum wird als rechter Teilbaum gesetzt.