18. Informatik – Fachbezogene Hinweise und Thematische Schwerpunkte für die schriftliche Abiturprüfung 2024

A. Fachbezogene Hinweise

Grundlage für die schriftliche Abiturprüfung 2024 in Niedersachsen sind die Einheitlichen Prüfungsanforderungen in der Abiturprüfung Informatik (EPA vom 01.12.1989 i.d.F. vom 05.02.2004), konkretisiert durch das Kerncurriculum Informatik für das Gymnasium – gymnasiale Oberstufe, die Gesamtschule – gymnasiale Oberstufe und das Kolleg (KC, 2017).

Entsprechend der Ausführungen des Kerncurriculums zu einheitlichen Darstellungsformen und Funktionsumfängen (KC, 2017, Abschnitt 3.5) sind die durch die „Ergänzenden Hinweise zum Kerncurriculum Informatik für die gymnasiale Oberstufe am Gymnasium und an der Gesamtschule sowie für das Kolleg“ mit Stand vom Juni 2021 vorgenommenen Festlegungen für die schriftliche Abiturprüfung 2024 verbindlich zu berücksichtigen.

B. Hinweise zu den Prüfungsaufgaben

Sämtliche im Kerncurriculum (KC, 2017) genannten inhaltlichen und prozessorientierten Kompetenzen sind für die schriftliche Abiturprüfung verbindlich.[1] Dies gilt insbesondere auch für die Kompetenzen, die in den Lernfeldern für die Einführungsphase ausgewiesen sind.

Für das erhöhte und für das grundlegende Anforderungsniveau gilt: Jede **Prüfungsaufgabe** besteht aus drei Aufgaben. Den Prüflingen werden jeweils Aufgaben aus zwei Blöcken (1 und 2) zur Auswahl vorgelegt. Aus Block 1 ist genau eine Aufgabe zur Bearbeitung auszuwählen. Aus Block 2 sind genau zwei der drei Aufgaben auszuwählen. Andere Kombinationen sind nicht zulässig.
Jede Aufgabe aus Block 1 umfasst 50% der insgesamt zu erreichenden Bewertungseinheiten (BE). Jede der drei Aufgaben im Block 2 umfasst 25% der insgesamt zu erreichenden BE.

Block 1 (50% der BE)
Aufgabe 1A
Aufgabe 1B

Eine von zwei Aufgaben ist zu wählen.

Block 2 (50% der BE)
Aufgabe 2A
Aufgabe 2B
Aufgabe 2C

Zwei von drei Aufgaben sind zu wählen.

C. Sonstige Hinweise

- Die Aufgabenstellungen enthalten keinen Code in einer konkreten Programmiersprache.
- Aufgaben, die die Implementierung in einer konkreten Programmiersprache erfordern, sind von den Schülerinnen und Schülern in Java oder einer anderen objektorientierten Sprache mit imperativem Kern zu bearbeiten.
- Es werden keine Aufgaben gestellt, für die der Einsatz eines Rechners erforderlich ist.

Hilfsmittel

- Die in Anlage 1 dokumentierten Ausführungen zu Operationen für Zeichenketten, Operationen für dynamische Reihungen, Stapel, Schlangen und Binärbäume sowie zum Sprachumfang von Datenbankabfragen sind in ausgedruckter Form als Hilfsmittel im Fach Informatik zulässig.
- Die Verwendung eines Taschenrechners oder einer Formelsammlung ist in der schriftlichen Abiturprüfung im Fach Informatik nicht zulässig

[1] Das Kerncurriculum (KC, 2017) ermöglicht Freiraum für individuelle Schwerpunktsetzungen. Darauf ist in diesem Durchgang ggf. zu verzichten, um auf die verbindlichen Vorgaben des Kerncurriculums zu fokussieren.

Anlage 1

Hilfsmittel für Prüflinge zur Verwendung in der schriftlichen Abiturprüfung im Fach Informatik

1 Operationen für Zeichenketten

Für die Arbeit mit Zeichenketten ist der Umfang der zu verwendenden Operationen auf die folgenden eingeschränkt:

- Bestimmen der Länge einer Zeichenkette
- Auslesen eines Zeichens an einer bestimmten Position
- Ersetzen eines Zeichens an einer bestimmten Position
- Verbinden von zwei Zeichenketten zu einer
- Prüfen des Inhalts von zwei Zeichenketten auf Gleichheit
- Lexikographisches Vergleichen von zwei Zeichenketten

Die Verwendung von darüber hinausgehenden Zeichenkettenoperationen ist nicht zulässig.

2 Operationen für dynamische Reihungen, Stapel, Schlangen und Binärbäume

Dynamische Reihung

Die Nummerierung der Elemente der dynamischen Reihung beginnt mit dem Index 0.

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

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

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

`append(inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird am Ende der dynamischen Reihung angefügt.

`insertAt(index: Ganzzahl, inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird an der Position `index` in die dynamische Reihung eingefügt. Das Element, das sich vorher an dieser Position befunden hat, und alle nachfolgenden werden nach hinten verschoben. Entspricht der Wert von `index` der Länge der dynamischen Reihung, so wird ein neues Element am Ende der dynamischen Reihung angefügt.

`setItem(index: Ganzzahl, inhalt: Inhaltstyp)`
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(): Inhaltstyp`
Der Inhalt des obersten Elements des Stapels wird zurückgegeben, das Element aber nicht entnommen.

`push(inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird oben auf den Stapel gelegt.

`pop(): Inhaltstyp`
Das oberste Element des Stapels wird entnommen. Der Inhalt dieses Elements wird zurückgegeben.

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(): Inhaltstyp`
Der Inhalt des vordersten Elements der Schlange wird zurückgegeben, das Element aber nicht entnommen.

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

`dequeue(): Inhaltstyp`
Das vorderste Element wird entnommen. Der Inhalt dieses Elements wird zurückgegeben.

Binärbaum

`BinTree()`
Ein Baum wird erzeugt. Der Baum besitzt keine Teilbäume. Die Wurzel besitzt keinen Inhaltswert.

`BinTree(inhalt: Inhaltstyp)`
Ein Baum wird erzeugt. Der Baum besitzt keine Teilbäume. Die Wurzel erhält den übergebenen Inhalt als Wert.

`hasItem(): Wahrheitswert`
Wenn die Wurzel des Baums einen Inhaltswert besitzt, wird der Wert `wahr` zurückgegeben, sonst der Wert `falsch`.

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

`setItem(inhalt: Inhaltstyp)`
Die Wurzel des Baums erhält den übergebenen Inhalt als Wert.

`deleteItem()`
Die Operation löscht den Inhaltswert der Wurzel des Baums.

`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`.

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

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

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

`deleteLeft()`
Die Operation löscht den linken Teilbaum.

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

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

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

`deleteRight()`
Die Operation löscht den rechten Teilbaum.

3 Datenbankabfragen

SELECT-Anweisung:

```
SELECT [DISTINCT | ALL] * | spalte1 [AS alias1], spalte2 [AS alias2],
       …, spalten [AS aliasn]
FROM tabelle1, tabelle2, …, tabellem
[WHERE bedingung1 (AND | OR) bedingung2 … (AND | OR) bedingungk]
[GROUP BY spalte1, spalte2, … , spaltep
[HAVING gruppenBedingung1 (AND | OR) gruppenBedingung2 … (AND | OR)
        gruppenBedingungs] ]
[ORDER BY spalte1 [ASC | DESC], spalte2 [ASC | DESC], …,
          spaltet [ASC | DESC] ]
[LIMIT anzahl]
```

Angaben in eckigen Klammern sind optional. Spalten können Attribute, Aggregatfunktionen oder Berechnungen sein. Bei `GROUP BY` und `ORDER BY` ist auch die Angabe eines Alias möglich.

Operatoren für Berechnungen: +, −, *, /

Operatoren für Vergleiche in Bedingungen: =, != (ungleich), >, <, >=, <=, `NOT`, `LIKE` (mit den Platzhaltern _ und %) , `BETWEEN`, `IN`, `IS NULL`

Aggregatfunktionen: `AVG`, `COUNT`, `MAX`, `MIN`, `SUM`

4 Notation von Bitfolgen nach dem (7,4)-Hamming-Code

Die Daten- und Prüfbits im (7,4)-Hamming-Code werden in der Reihenfolge $p_0\ p_1\ d_0\ p_2\ d_1\ d_2\ d_3$ notiert.

Die Kontrollgruppen sind entsprechend der folgenden Abbildung zusammengesetzt:

	Datenbits			
Prüfbit	d_0	d_1	d_2	d_3
p_0	x	x	-	x
p_1	x	-	x	x
p_2	-	x	x	x