Universität Regensburg

PD Dr. J. Reischer 12.02.2013

Klausur "ADP" WS 2012/2013

Nachname, Vorname	
Abschluss (BA, MA, FKN etc.)	
Matrikelnummer, Semester	
Versuch (1/2/3)	

Bitte füllen Sie zuerst den Kopf des Angabenblattes aus!

Die Klausur dauert 90 Minuten.

Es sind maximal 48 Punkte zu erreichen.

Es sind keine Hilfsmittel zugelassen.

Bitte beantworten Sie alle Fragen direkt auf das Angabenblatt.
Nutzen Sie ggf. die Rückseite und kennzeichnen Sie dies entsprechend.
Eigene Schmierblätter sind nicht erlaubt.

Bei mehreren oder mehrdeutigen Lösungen wird die schlechtere Lösung gewertet. Streichen Sie daher ungültige Lösungen eindeutig durch.

Verwenden Sie nur Java, C# oder Pseudokode für Programmieraufgaben.

Viel Erfolg!

Aufgabe 1: *Iteration Analyse* (6 Punkte)

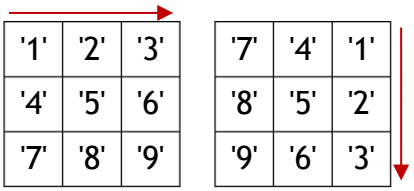
Konstruieren Sie für folgenden Algorithmus in Pseudokode ein Struktogramm.

```
// Binärsuche eines Zeichens C in einem String S
int32s BinarySearch(char C, string S)
{
  int32s XM := –1, XL := 0;
  int32s XH := S.Length–1;
  while XL ≤ XH
  {
    XM := (XH + XL) / 2;
    if C ≠ S[XM] then
      if C < S[XM]
      then XH := XM – 1;
      else XL := XM + 1;
    else return XM;
  }
  do;
  return –1;
};
```

Aufgabe 2: *Iteration Synthese* (14 Punkte)

2.1: Schreiben Sie eine Funktion **`boole`** `IsRotated(`**`char`**`[][] Matrix1,` **`char`**`[][] Matrix2)`, die die *quadratische* `Matrix2` daraufhin testet, ob die in ihr enthaltenen Zeichen den im Uhrzeigersinn rotierten Zeichen aus der *gleich großen* `Matrix1` entsprechen (d. h. ist `Matrix2` die rechts herum rotierte `Matrix1`?). Die beiden Matrizen können beliebige gleiche Ausdehnung 1×1, 2×2, 3×3, …, N×N außer 0×0 besitzen (benutzen Sie `Matrix.length()` [Java] bzw. `Matrix.Length` [C#, Pseudo-kode] zur Ermittlung der horizontalen/vertikalen Matrixgröße). Die Quadratigkeit der Matrix kann vorausgesetzt werden. (6 Punkte)

Beispiel für 3×3:

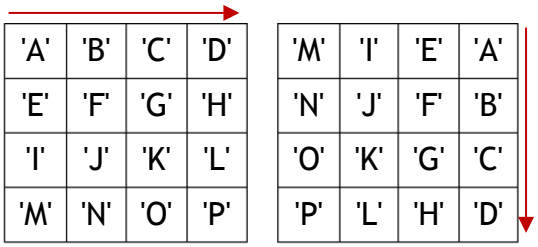


'1'	'2'	'3'
'4'	'5'	'6'
'7'	'8'	'9'

'7'	'4'	'1'
'8'	'5'	'2'
'9'	'6'	'3'

`Matrix1` links, `Matrix2` rechts

Beispiel für 4×4:



'A'	'B'	'C'	'D'
'E'	'F'	'G'	'H'
'I'	'J'	'K'	'L'
'M'	'N'	'O'	'P'

'M'	'I'	'E'	'A'
'N'	'J'	'F'	'B'
'O'	'K'	'G'	'C'
'P'	'L'	'H'	'D'

`Matrix1` links, `Matrix2` rechts

2.2: Schreiben Sie eine Funktion **`boole`** `IsSymmetric(`**`char`**`[][] Matrix)`, die die quadratische Matrix `Matrix` daraufhin testet, ob die in ihr enthaltenen Zeichen symmetrisch an der mittigen X- *und* Y-Achse zugleich gespiegelt sind, d. h. ob es sich letztlich um ein zweidimensionales Palindrom handelt (s. Beispiel unten). Die Matrizen können beliebige Ausdehnungen 1×1, 2×2, 3×3, …, N×N außer 0×0 besitzen. Die Quadratigkeit der Matrix kann vorausgesetzt werden. (8 Punkte)

Beispiel für symmetrische Matrizen:

'A'	'N'	'A'
'N'	'O'	'N'
'A'	'N'	'A'

'A'	'N'	'N'	'A'
'N'	'O'	'O'	'N'
'N'	'O'	'O'	'N'
'A'	'N'	'N'	'A'

Beispiel für asymmetrische Matrizen:

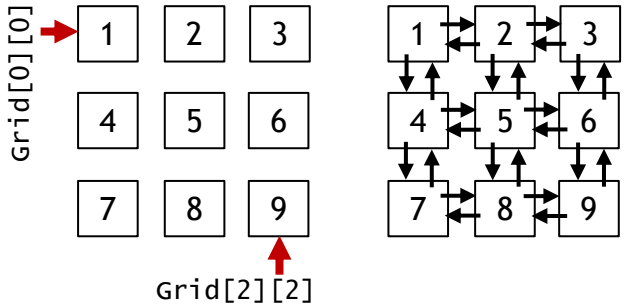
'A'	'N'	'A'
'N'	'O'	'N'
'A'	'N'	'X'

'A'	'N'	'N'	'A'
'N'	'O'	'X'	'N'
'N'	'O'	'O'	'N'
'A'	'N'	'N'	'A'

Aufgabe 3: *Iteration Synthese* (10 Punkte)

Schreiben Sie eine Funktion `GridNode[][] LinkNodes(GridNode[][] Grid)`, die ein zweidimensionales *quadratisches* Array (Matrix) `Grid` mit Elementen des Typs `GridNode` erhält (Def. s. unten) und diese mit ihren jeweiligen Nachbarn – so weit als logisch möglich – vollständig verlinkt (innere Knoten können mit maximal 4 Nachbarknoten verlinkt werden, äußere Randknoten mit weniger als 4). Das Array `Grid` mit den verlinkten Knoten soll als Funktionsergebnis zurückgegeben werden.

Beispiel-Grid: *unverlinkt* → *verlinkt*



Definition von `GridNode`:

```
class GridNode
{
  public int32s Cont = 0;
  public GridNode Up = null;
  public GridNode Down = null;
  public GridNode Left = null;
  public GridNode Right = null;
};
```

Beachten Sie, dass alle Knoten des Typs `GridNode` über das Array `Grid[][]` direkt zugreifbar sind und daher kein Einstiegsknoten notwendig ist. Alle Links (Referenzen) `Up`, `Down`, `Left` und `Right` sind mit **`null`** vorbelegt. Der Knoteninhalt `Cont` ist jeweils beispielhaft mit einer Ganzzahl belegt und muss nicht selbst gesetzt werden.

Aufgabe 4: *Rekursion Analyse* (10 Punkte)

Gegeben ist folgende rekursive Funktion in Pseudokode:

```
int32s F(int32s N)
{
  if N ≤ 0 then
    return N + 1;
  else                              // '%': Modulo-Operator
    if N % 2 = 0 then               // (hier Test auf gerade Zahl)
      return F(N – 1) + 1;          // wenn Zahl gerade
    else
      return F(N – 1) + N;          // wenn Zahl nicht gerade
};
```

4.1: Welche Werte liefert die Funktion für die Aufrufe N = 0, 1, 2, 3, 4, 5 zurück? Tragen Sie die Ergebnisse in unten stehende Tabelle ein (zwei Versuche, sonst eigene Tabelle unten konstruieren; streichen Sie einen Fehlversuch deutlich durch). (6 Punkte)

N	0	1	2	3	4	5
F(N)						
F(N)						

4.2: Welchen Komplexitätsgrad bezüglich Laufzeitverhalten O(?) weist die Funktion auf (Begründung!)? (2 Punkte)

4.3: Nennen Sie zwei verschiedene Klassifikationsdimensionen für die in obigem Kode auftretenden Arten der Rekursion. (2 Punkte)

Aufgabe 5: *Rekursion Synthese* (8 Punkte)

Schreiben Sie eine Methode `boole IsEqual(string S1, string S2)` als *rekursive Funktion* mit booleschem Rückgabewert, die zwei Strings S1 und S2 auf Gleichheit prüft (ob die Funktion `static`, `public` etc. ist, ist hier uninteressant). Rekursionslogik:

S1 und S2 sind gleich, wenn das jeweils erste Zeichen aus S1 und S2 gleich ist und der jeweilige Rest von S1 und S2 ebenfalls gleich ist.

Achten Sie auf die Sonder-/Grenzfälle bzw. Terminationsbedingungen:
Wann können S1 und S2 nicht (mehr) gleich sein?
In welcher Situation endet die Rekursion?

Sie dürfen folgende Funktionen zur Ermittlung eines Teilstrings T bzw. eines Zeichens C aus S (S1 oder S2) verwenden:

C#: `T = S.Substring(StartIndex,AnzahlZeichen);`
`C = S[ZeichenIndex];`
Java: `T = S.substring(BeginIndex,EndIndex`$_{\text{Exklusive}}$`);`★
`C = S.charAt(ZeichenIndex);`
Pseudokode: `T := S.Extract(StartIndex,AnzahlZeichen);`
`C := S[ZeichenIndex];`

(★ Exklusiver Endindex: Index des ersten Zeichens, das *nicht* mehr zu T gehören soll!)

Die Länge eines Strings in Zeichen kann mit `S.Length` (C#, Pseudokode) bzw. `S.length()` (Java) ermittelt werden.