PRO1UEG*x* **Übungen zu Einführung in die Programmierung, Algorithmen und Datenstrukturen** **WS 25/26, Übung 4**

Abgabetermin: Mi, 12.11.2025

☒ **Gr. 1,** Dr. D. Auer **Name** Klemens Danner **Aufwand in h** 9
☐ **Gr. 2,** Dr. G. Kronberger
☐ **Gr. 3,** Dr. S. Wagner **Punkte** _____ **Kurzzeichen Tutor*in / Übungsleiter*in** ____ / ____

1. Plateau-Problem (4 Punkte)

Gegeben ist ein Feld `a[1:n]` mit $n \geq 1$, das eine Folge von aufsteigend sortierten positiven ganzen Zahlen (Datentyp `INTEGER`) enthält. Ein Plateau ist eine Folge von gleichen Werten.

Gesucht ist eine Pascal-Funktion

```
FUNCTION Plateau(a: IntArray; n: INTEGER): INTEGER;
```

die die Länge des längsten Plateaus in `a` berechnet und dazu das Feld `a` nur einmal durchläuft.

Beispiel:

Für `a` = (1, 2, 2, 3, 4, 4, 6, 7, 7, 7, 8, 9, 9) und `n` = 13 muss die Funktion den Wert 3 liefern.

2. Das fehlende Element (6 + 6 Punkte)

Gegeben sei eine Folge ganzer Zahlen mit `n` Elementen. Die Zahlen in der Folge sind unsortiert und entstammen dem Wertebereich 1 bis `n` + 1. Bis auf eine Zahl kommen alle anderen Werte aus diesem Bereich genau einmal darin vor. Hier ein Beispiel für eine solche Folge für `n` = 4, also mit 4 Elementen: 3, 2, 4, 5. Offensichtlich fehlt hier das Element 1.

Gesucht ist eine Pascal-Funktion `MissingElement`, die das fehlende Element einer solchen Zahlenfolge liefert.

Verwenden Sie folgende Deklarationen:

```
CONST
  max = 100;

TYPE
  IntArray = ARRAY [1 .. max] OF INTEGER;

FUNCTION MissingElement(a: IntArray; n: INTEGER): INTEGER;
```

a) Implementieren Sie die Funktion, indem Sie ein Hilfsfeld `h` mit `max` Elementen vom Datentyp `BOOLEAN` verwenden. Im ersten Schritt werden alle Elemente von 1 bis n + 1 mit `FALSE` initialisiert. Im zweiten Schritt wird jedes Element in `h`, dessen Index in `a` vorkommt, auf `TRUE` gesetzt. Im dritten Schritt ist schließlich der Index jenes Elements zu ermitteln, das noch den Wert `FALSE` enthält.

b) Implementieren Sie die Funktion ohne Verwendung eines Hilfsfelds.

3. Matrizenmultiplikation (8 Punkte)

Definieren Sie einen Datentyp `Matrix` zur Repräsentation von 3x3-Matrizen mit Elementen vom Datentyp `REAL`. Implementieren Sie je eine Prozedur zur Eingabe, eine zur Ausgabe und eine weitere zur Multiplikation von Matrizen. Die Matrizenmultiplikation ist wie folgt definiert:

Das Element c_{ij} des Matrizenproduktes $C = A \cdot B$ ergibt sich als skalares Produkt $a^i \cdot b_j$ des Zeilenvektors a^i mit dem Spaltenvektor b_j:

$$\left(c_{ij}\right)_{(i=1..m,\ j=1..p)} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Voraussetzung: $A = \left(a_{i,j}\right)_{i=1..m,j=1..n}$ und $B = \left(b_{i,j}\right)_{i=1..n,j=1..p}$ (Spaltenzahl von *A* = Zeilenzahl von *B*).

Hinweise:

1. Geben Sie für alle Ihre Lösungen immer eine „Lösungsidee“ an.
2. Dokumentieren und kommentieren Sie Ihre Pascal-Programme.
3. Geben Sie immer auch Testfälle ab, an denen man erkennen kann, dass Ihr Pascal-Programm funktioniert, und dass es auch in Fehlersituationen entsprechend reagiert.

1 Plateau Problem

Das Array soll mittels Zählschleife von 1 bis n durchlaufen und immer die Anzahl der aufeinander folgenden Werte in ein Hilfsarray speichern. Mit einer eigenen Funktion wird der größte Wert dieses Hilfsarrays bestimmt, welcher also die Anzahl der aufeinander folgenden Werte ist.

```
program PlateauProblem;

const
    max = 100;

Type
    IntArray = array[1..max] of integer;

function GreatestValue(a: IntArray; n: integer): integer;
var
    i, maxValue: integer;
begin
    maxValue := 0; // init
    for i := 1 to n do
        begin
            if a[i] > maxValue then
                maxValue := a[i];
        end;
        GreatestValue := maxValue;
end;


function Plateau(a: IntArray; n: integer): integer;
var
    i: integer;
    NumCounter: IntArray;

begin
    NumCounter[1] := 1; //init
    for i := 2 to n do
        if (a[i] = a[i-1]) then
            NumCounter[i] := NumCounter[i-1] + 1
        else
            NumCounter[i] := 1;

    Plateau := GreatestValue(NumCounter, n);
end;


procedure ReadIntArray(VAR a: IntArray; VAR n: integer);
var
    i: integer;
begin
    WriteLn('Enter the length of the array: ');

```

```
46      repeat //Eingabeprüfung
47          ReadLn(n);
48          if n<1 then
49              WriteLn('Try again. Enter a length >= 1')
50      until n>=1;
51
52      WriteLn('Enter ', n, ' values for your array: ');
53      for i := 1 to n do
54          Read(a[i]);
55  end;
56
57  //global var
58  var
59      testArray: IntArray;
60      n: integer;
61
62  begin
63
64      ReadIntArray(testArray, n);
65      WriteLn('Plateau-Value: ', Plateau(testArray, n));
66
67  end.
```

Tests

Beschreibung	Eingabe / Ausgabe

Beispiel	`Enter the length of the array:`
	`13`
	`Enter 13 values for your array:`
	`1 2 2 3 4 4 6 7 7 7 8 9 9`
	`Plateau-Value: 3`

Nur gleiche Werte	`Enter the length of the array:`
	`6`
	`Enter 6 values for your array:`
	`1 2 3 4 5 6`
	`Plateau-Value: 1`

nur 1 Wert	`Enter the length of the array:`
	`1`
	`Enter 1 values for your array:`
	`2`
	`Plateau-Value: 1`

negative Länge / 0	`Enter the length of the array:`
	`-1`
	`Try again. Enter a length >= 1`
	`0`
	`Try again. Enter a length >= 1`
	`1`
	`Enter 1 values for your array:`
	`3`
	`Plateau-Value: 1`

2 Das fehlende Element

2.1 Lösungsidee

Für die Funktion MissingElement1() erfolgt die Lösung laut Aufgabenstellung.

Lösungsidee MissingElement2():
Eine Schleife prüft immer, ob eine Zählvariable (startend bei 1) im Array enthalten ist oder nicht. Wenn sie enthalten ist, wird die Zählvariable um eins erhöht. Wenn sie nicht enthalten ist, so ist das offensichtlich der fehlende Wert. Dazu wird eine Hilfsfunktion inArray(i,a,n) verwendet, welche zuerst definiert wird.

2.2 Quellcode

```
1  program TheMissingElement;
2
3  const
4      max = 100;
5
6  Type
7      IntArray = array[1..max] of integer;
8
9
10 function ExistInArray(a: IntArray; len, num: integer): boolean;
11     var
12         j: integer;
13     begin
14     //durchsuchen mit while schleife für sofortigen Abbruch bei Fund
15
16         j := 1; //init
17         while (j<max) and (a[j] <> num) do //solange j hochzählen bis
   Index=Arraywert
18             begin
19                 j := j + 1;
20             end;
21
22         if j <= len then
23             ExistInArray := True
24         Else
25             ExistInArray := False;
26
27     end;
28
29
30 function MissingElement1(a: IntArray; n: integer): integer;
31     var
32         h: array[1..max] of boolean;
33         i, k: integer;
34
35     begin
36         //h init
37         for i := 1 to (n+1) do
38             h[i] := false;
39
40         //prüfen, ob Index vom Feld im Array vorkommt
41         for i := 1 to (n+1) do
42             begin
```

```
                if ExistInArray(a, n, i) then
                    h[i] := true;
            end;

        //h durchlaufen und beim ersten False aufhören, Index --> Ergebnis
        k := 0; //init

        repeat
            k := k + 1;
        until h[k] = false;

        MissingElement1 := k;

    end;


function MissingElement2(a: IntArray; n: integer): integer;
var
    i: integer;
begin

    i := 1; //init

    //wenn i nicht im Array enthalten ist, ist es der gesuchte Wert
    while ExistInArray(a,n,i) do
        begin
            i := i + 1;
        end;

    MissingElement2 := i;
end;

var
  n, i: integer;
  a: IntArray;
    input: integer;

begin
  //Länge einlesen
  Write('n: ');
  ReadLn(n);

    //Array einlesen, wenn n valid
    if n <= 0 then
        begin
            WriteLn('error: no numbers <= 0');
```

```
89              exit; //Eingangsbedingung nicht erfüllt
90          end;
91
92          WriteLn('Enter the values of your array: ');
93
94          a[1] := 0; //init, wird überschrieben
95      for i := 1 to n do
96      begin
97        ReadLn(input);
98              if (input <= 0) OR (input > n+1) OR ExistInArray(a,n,input) then
99                  begin
100                     WriteLn('error: input out of range or use a number only
    once');
101                     exit; //Eingangsbedingung nicht erfüllt
102                 end
103             else
104                 a[i] := input;
105         end;
106
107     WriteLn('missing element a): ', MissingElement1(a,n));
108     WriteLn('missing element b): ', MissingElement2(a,n));
109
110 end.
```

2.3 Tests

Eingabe	Beschreibung	Ausgabe
n: 4 3,2,4,5	Aufgabenstellung	n: 4 Enter the values of your array: 3 2 4 5 missing element a): 1 missing element b): 1
n: 7 4,2,7,3,1,6,5	anderes n + letzte zahl fehlt	n: 7 Enter the values of your array: 4 2 7 3 1 6 5 missing element a): 8 missing element b): 8
n: 1 2	kleinster gültiger input	n: 1 Enter the values of your array: 2 missing element a): 1 missing element b): 1
n: 0	n out of range	n: 0 error: no numbers <= 0
n: 4 3, 2, 4, -1	a[i] out of range	n: 4 Enter the values of your array: 3 2 4 -1 error: input out of range or use a number only once
n: 4 3, 2, 2, 4	Wert doppelt eingegeben	n: 4 Enter the values of your array: 3 2 2 error: input out of range or use a number only once

3 Matrizenmultiplikation

3.1 Lösungsidee

Die Realisation erfolgt mittels mehrdimensionaler Felder. Die Eingabe wird mit zwei verschachtelten Zählschleifen gemacht, die Ausgabe analog dazu. Bei der Multiplikation wird wieder durch zwei verschachtelte Zählschleifen zur aktuellen Stelle c[i, j] "navigiert" und dort die Formel zur Berechnung durch Aufsummieren angewendet.

3.2 Quellcode

```
program Matrizenmultiplikation;

type
    Matrix = array[1..3,1..3] of integer;

procedure MatrixInput(VAR mat: Matrix); //Zeilen und Spalten einer 3x3
Matrix einlesen
var
    i, j: integer;

begin
    for i := 1 to 3 do //Zeilen
    begin
          for j := 1 to 3 do //Spalten
              Read(mat[i,j]);
      ReadLn;
    end;
end;

procedure MatrixOutput(mat: Matrix);
var
    i, j: integer;

begin
    for i := 1 to 3 do //Zeilen
        begin
            for j := 1 to 3 do //Spalten
                Write(mat[i,j]:4);

            WriteLn; //Zeilenumbruch bei neuer Zeile
        end;
end;

procedure MatrixMulti(a, b: Matrix; VAR c: Matrix);
var
    i, j, k: integer;
begin
    for i := 1 to 3 do
        for j := 1 to 3 do
            begin //Stelle c[i,j] von Ergebnismatrix
                c[i,j] := 0; //init
                //Formel zur Berechnung von c[i,j] anwenden

```

```
                for k := 1 to 3 do
                    begin
                        c[i,j] := c[i,j] + (a[i,k]*b[k,j]);
                    end;
            end;
end;

//global variables
var
    a, b, c: Matrix;
    MultiWantedChar: char;

begin
  WriteLn('Enter two 3x3 matrices');
    MatrixInput(a);
    WriteLn;

    MatrixOutput(a);
    WriteLn;

    MatrixInput(b);
    WriteLn;

    MatrixOutput(b);
    WriteLn;

    //Abfragen ob Multiplikation gewünscht
    WriteLn('Do you want to multiply the entered matrices? (y/n)');
    Read(MultiWantedChar);

    if MultiWantedChar = 'y' then
        begin //Multiplikation
            MatrixMulti(a,b,c);
            MatrixOutput(c);
        end;
end.
```

3.3 Tests

Eingabe	Beschreibung	Ausgabe
123 456 789 987 654 321	random Matrizen	Enter two 3x3 matrices 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 9 8 7 6 5 4 3 2 1 9 8 7 6 5 4 3 2 1 Do you want to multiply the entered matrices? (y/n) y 30 24 18 84 69 54 138 114 90
123 456 789 100 010 001	Einheitsmatrix	Enter two 3x3 matrices 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 1 Do you want to multiply the entered matrices? (y/n) y 1 2 3 4 5 6 7 8 9
-1 2 -3 4 -5 6 1 2 3 4 -5 6 -1 2 -3 3 2 1	negative Zahlen	Enter two 3x3 matrices -1 2 -3 4 -5 6 1 2 3 -1 2 -3 4 -5 6 1 2 3 4 -5 6 -1 2 -3 3 2 1 4 -5 6 -1 2 -3 3 2 1 Do you want to multiply the entered matrices? (y/n) y -15 3 -15 39 -18 45 11 5 3

Eingabe	Beschreibung	Ausgabe
0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9	0	Enter two 3x3 matrices 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 Do you want to multiply the entered matrices? (y/n) y 0 0 0 0 0 0 0 0 0