

☒ Gr. 1, Dr. D. Auer

Name Klemens Danner

Aufwand in h 14

☐ Gr. 2, Dr. G. Kronberger

☐ Gr. 3, Dr. S. Wagner

Punkte _____

Kurzzeichen Tutor*in / Übungsleiter*in ____ / ____

1. Feldverarbeitung mit offenen Feldparametern: Schnittmengen (8 Punkte)

Gegeben sind zwei beliebig große Felder a1 und a2, die n1 bzw. n2 ganze Zahlen in aufsteigend sortierter Reihenfolge enthalten. Gesucht ist eine Pascal-Prozedur

```
PROCEDURE Intersect(a1: ARRAY OF INTEGER; n1: INTEGER;
                   a2: ARRAY OF INTEGER; n2: INTEGER;
                   VAR a3: ARRAY OF INTEGER; VAR n3: INTEGER);
```

die das Feld a3 so befüllt, dass darin alle n3 Zahlen in aufsteigend sortierter Reihenfolge enthalten sind, die sowohl in a1 als auch in a2 vorkommen. Im Fehlerfall (Überlauf in a3, Felder a1 und a2 sind nicht aufsteigend sortiert) soll n3 auf -1 gestellt werden.

Beispiel:

a1 =	1	2	3	5	8	13	...	n1 =	6
a2 =	1	3	5	7	9	...		n2 =	5
<hr/>									
a3 =	1	3	5	...				n3 =	3

Hinweis: Implementieren Sie eine Pascal-Funktion IsSorted, die prüft, ob die Werte in einem Feld a aufsteigend sortiert sind und verwenden Sie diese Funktion in der Prozedur Intersect.

2. Funktionen zur Zeichenkettenverarbeitung (1 + 2 + 4 Punkte)

Entwickeln Sie weitere Operationen zur Zeichenkettenbearbeitung (Datentyp STRING). Verwenden Sie dafür insbesondere die bereits vorhandenen Pascal-Standardfunktionen.

- Implementieren Sie eine Funktion WithoutLastChar, die in der als Eingangsparameter s übergebenen Zeichenkette das letzte Zeichen entfernt und das Ergebnis als Funktionswert liefert.
- Implementieren Sie eine Funktion EqualsIgnoreCase, die zwei Zeichenketten a und b auf Gleichheit prüft. Dabei soll die Groß/Kleinschreibung nicht berücksichtigt werden, d.h. für die Zeichenketten a = 'Pascal' und b = 'PASCAL' muss der Algorithmus TRUE liefern.
- Implementieren Sie eine Funktion

```
FUNCTION CamelCase(words: ARRAY OF STRING; n: INTEGER): STRING;
```

die die n im String-Array words gespeicherten Wörter zu einer Zeichenkette in camelCase Schreibweise zusammensetzt und zurückliefert. Jedes Wort darf dabei nur aus Kleinbuchstaben (a..z), Großbuchstaben (A..Z) oder Ziffern (0..9) bestehen. Im Fehlerfall, z.B. wenn die übergebenen Wörter nicht korrekt sind oder die resultierende Zeichenkette zu lang ist, soll als Ergebnis die Zeichenkette ERROR zurückgeliefert werden.

Beispiele:

words:

My	CAMEL	CaSe	name2	
----	-------	------	-------	--

 → myCamelCaseName2
n: 4

words:

wrong	n#me			
-------	------	--	--	--

 → ERROR
n: 2

3. Kaffeeautomat

(7 + 2 Punkte)

Entwickeln Sie ein Steuerprogramm für einen Kaffeeautomaten. Der Automat nimmt 10- und 50-Cent-Münzen sowie 1-Euro-Münzen. Der Preis für einen Kaffee beträgt 40 Cent. Nach dem Einwurf des Geldes und dem Drücken der "Kaffee-Taste" wird ein Algorithmus gestartet. Beim Einwurf von weniger als 40 Cent soll der Algorithmus das Geld wieder auswerfen und eine entsprechende Fehlermeldung ausgeben. Ist der eingeworfene Betrag zu hoch, soll der Automat den Restbetrag zurückgeben. Er soll dabei möglichst wenige Münzen verwenden. Falls der Automat nicht über ausreichend Wechselgeld verfügt, soll er den eingeworfenen Betrag zurückgeben und eine entsprechende Fehlermeldung ausgeben. Falls dies dreimal hintereinander passiert, soll der Automat die Fehlermeldung "ERROR: Out of order!" ausgeben und auf keine weiteren Münzeinwürfe/Tastendrucke mehr reagieren.

Implementieren Sie einen Algorithmus mit Gedächtnis mit folgender Schnittstelle

```
PROCEDURE CoffeeButtonPressed(input: Coins; VAR change: Coins);
```

der das Drücken der "Kaffee-Taste" simuliert. Am Anfang soll der Automat über Wechselgeld in der Höhe von zehn 10-Cent-Münzen, fünf 50-Cent-Münzen und keine 1-Euro Münzen verfügen.

- a) Implementieren Sie das Gedächtnis in Form von globalen Variablen.
- b) Verpacken Sie Ihren Algorithmus mit Gedächtnis in eine Unit.

Hinweis:

Definieren Sie den Datentyp `Coins` entsprechend der Aufgabenstellung, so dass ein `Coins`-Datenobjekt die Anzahl der drei verschiedenen Münzen speichern kann.

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 Schnittmengen

1.1 Lösungsidee

Es soll eine Funktion implementiert werden, welche prüft, ob die Werte aufsteigend sortiert sind. Das wird realisiert, indem jeder Wert mit seinem Nachfolger verglichen wird.

In der Funktion Intersect wird am Anfang geprüft, ob die übergebenen Arrays sortiert sind.

Wenn dies der Fall ist, startet der eigentliche Algorithmus.

Dabei sollen die Arrays nur einmal durchlaufen werden.

In einer Schleife wird immer geprüft, ob die Einträge der beiden Arrays gleich sind. Wenn dies der Fall ist und im Zielarray genügend Platz ist, wird die Zahl in das Ausgangsarray gespeichert und beide Indizes werden erhöht.

Falls die Einträge ungleich sind, wird der Index der kleineren Zahl inkrementiert.

Der Vorgang wird so lange wiederholt, bis eines der Arrays vollständig durchlaufen wurde.

1.2 Quellcode

```
1  PROGRAM Schnittmenge;
2
3  CONST
4      max = 100;
5
6  TYPE
7      IntArray = ARRAY[1..max] OF INTEGER;
8
9  PROCEDURE ReadArray(VAR a: ARRAY OF INTEGER; n: INTEGER);
10     VAR
11         i: INTEGER;
12
13     BEGIN
14         FOR i := 0 TO (n-1) DO
15             Read(a[i]);
16         END;
17
18     PROCEDURE WriteArray(a: ARRAY OF INTEGER; n: INTEGER);
19         VAR
20             i: INTEGER;
21         BEGIN
22             FOR i := 0 TO (n-1) DO
23                 Write(a[i]:3);
24             END;
25
26     FUNCTION IsSorted(a: ARRAY OF INTEGER; n: INTEGER): BOOLEAN;
27         VAR
28             i: INTEGER;
29             sorted: BOOLEAN;
30
31         BEGIN
32             i := 0; (*OpenArray-Init*)
33             sorted := TRUE;
34             WHILE sorted AND (i <= (n-2)) DO BEGIN (*Zugriff auf i+1 & open-array*)
35                 IF a[i] > a[i+1] THEN
36                     sorted := FALSE;
37                     i := i + 1;
38                 END; (*end while*)
39                 IsSorted := sorted;
40             END;
41
42     PROCEDURE Intersect(a1: ARRAY OF INTEGER; n1: INTEGER; a2: ARRAY OF
INTEGER; n2: INTEGER; VAR a3: ARRAY OF INTEGER; VAR n3: INTEGER);
```

```

43     VAR
44         a3max, i, j: INTEGER;
45 BEGIN
46     (*Eingangskriterien prüfen*)
47     IF (NOT IsSorted(a1, n1)) OR (NOT IsSorted(a2, n2)) THEN BEGIN
48         n3 := -1;
49         Exit;
50     END;
51
52     a3max := high(a3);
53     i := 0;
54     j := 0; (*open-array-init*)
55     n3 := 0; (*init*)
56
57     WHILE (i <= (n1 - 1)) AND (j <= (n2 - 1)) DO BEGIN
58         IF (a1[i] = a2[j]) THEN BEGIN
59             IF n3 <= a3max THEN BEGIN
60                 a3[n3] := a1[i];
61                 i := i + 1;
62                 j := j + 1;
63                 n3 := n3 + 1;
64             END ELSE BEGIN
65                 n3 := -1;
66                 Exit; (*Überlauf*)
67             END
68         END ELSE
69             IF (a1[i] < a2[j]) THEN
70                 i := i + 1
71             ELSE
72                 j := j + 1;
73         END; (*WHILE*)
74     END;
75
76     VAR
77         a1, a2, a3: IntArray;
78         n1, n2, n3 : INTEGER;
79
80
81 BEGIN
82     Write('n1 > ');
83     Read(n1);
84
85     Write('Enter ', n1, ' values > ');
86     ReadArray(a1, n1);
87     WriteArray(a1, n1);
88     WriteLn;

```

```
89
90     Write('n2 > ');
91     Read(n2);
92
93     Write('Enter ', n2, ' values > ');
94     ReadArray(a2,n2);
95     WriteArray(a2,n2);
96
97     Intersect(a1, n1, a2, n2, a3, n3);
98
99     IF (n3 = -1) THEN
100         WriteLn('An Error occured: input arrays not sorted or overflow')
101     ELSE BEGIN
102         WriteLn('Those numbers are in both arrays: ');
103         WriteArray(a3, n3);
104         WriteLn;
105     END; (*IF*)
106
107 END.
108
```

Eingabe	Begründung	Ausgabe
siehe Screenshot	mehrere gleiche Zahlen	<pre> n1 > 5 Enter 5 values > 1 2 2 4 4 1 2 2 4 4 n2 > 4 Enter 4 values > 2 2 3 4 2 2 3 4 Those numbers are in both arrays: 2 2 4 </pre>
siehe Screenshot	keine gleichen Zahlen, no message is a good one; Zielarray leer, deshalb kein Output	<pre> n1 > 4 Enter 4 values > 1 3 4 5 1 3 4 5 n2 > 3 Enter 3 values > 2 6 7 2 6 7 Those numbers are in both arrays: </pre>
0	0	<pre> n1 > 0 Enter 0 values > n2 > 0 Enter 0 values > Those numbers are in both arrays: </pre>
siehe Screenshot	nicht sortiert	<pre> n1 > 4 Enter 4 values > 2 1 3 5 2 1 3 5 n2 > 3 Enter 3 values > 1 2 3 1 2 3 An Error occured: input arrays not sorted or overflow </pre>
siehe Screenshot	temporär ein Output-Array mit der Länge 5 übergeben, um einen overflow zu triggern	<pre> n1 > 6 Enter 6 values > 2 2 2 2 2 2 2 2 2 2 2 2 n2 > 6 Enter 6 values > 2 2 2 2 2 2 2 2 2 2 2 2 An Error occured: input arrays not sorted or overflow </pre>

2 Zeichenkettenverarbeitung

2.1 Lösungsidee

In der Funktion WithoutLastChar wird die Standardfunktion Copy verwendet.

In der Funktion EqualsIgnoreCase wird die Standardfunktion Uppcase verwendet, um beide strings in deren Großbuchstabenschreibweise zu vergleichen.

Die Funktion camelCase wird folgendermaßen realisiert:

Konzeptionell soll in einer Schleife in jedem Schleifendurchlauf ein neues Wort zum Ausgangsstring hinzugefügt werden.

In einem Schleifendurchlauf werden zuerst zwei Abbruchbedingungen geprüft und ggf. wird der Rückgabewert auf 'Error' gesetzt und die Funktion verlassen.

Danach wird bei jedem Wort der erste Buchstabe groß und die restlichen klein geschrieben.
(nur der allererste Buchstabe bleibt klein)

Anschließend wird das modifizierte Wort zum Ergebnisstring hinzugefügt.

2.2 Quellcode

```
1  PROGRAM Zeichenkettenverarbeitung;
2  CONST
3      max = 100;
4
5  PROCEDURE ReadStringArray(VAR a: ARRAY OF STRING; VAR n: INTEGER);
6      VAR
7          i: INTEGER;
8  BEGIN
9      Write('Anzahl der einzulesenden Woerter > ');
10     ReadLn(n);
11     WriteLn('Gib die ', n, ' Woerter ein: ');
12     FOR i := 0 TO (n-1) DO BEGIN
13         Write((i+1), '. > ');
14         ReadLn(a[i]);
15     END; (*for*)
16 END;
17
18 FUNCTION WithoutLastChar(s: STRING): STRING;
19 BEGIN
20     WithoutLastChar := copy(s, 1, (length(s)-1));
21 END;
22
23 FUNCTION EqualsIgnoreCase(a, b: STRING): BOOLEAN;
24 BEGIN
25     IF (Uppcase(a) = Uppcase(b)) THEN
26         EqualsIgnoreCase := TRUE
27     ELSE
28         EqualsIgnoreCase := FALSE;
29 END;
30
31
32 FUNCTION camelCase(words: ARRAY OF STRING; n: INTEGER): STRING;
33     VAR
34         i, j: INTEGER;
35         s: STRING;
36
37 BEGIN
38     i := 0;
39     s := '';
40
41     WHILE (i < n) DO BEGIN
42
43         IF (length(s)+length(words[i])) > high(s) THEN BEGIN
```

```

44         camelCase := 'Error';
45         Exit;
46     END; (*if*)
47
48     (*wörter prüfen*)
49     j := 1;
50     WHILE (j <= length(words[i])) DO BEGIN
51         IF NOT ((words[i][j] IN ['a'..'z', 'A'..'Z', '0'..'9'])) THEN BEGIN
52             camelCase := 'Error';
53             Exit;
54         END; (*if*)
55         inc(j);
56     END; (*while*)
57
58     IF (i = 0) THEN (*ersten Buchstaben klein schreiben*)
59         words[i] := LowerCase(copy(words[i], 1, length(words[i])))
60     ELSE
61         words[i] := UpCase(copy(words[i], 1, 1)) + LowerCase(copy(words[i], 2,
62         (length(words[i])-1)));
63
64     s := s + words[i];
65     inc(i);
66     END; (*while*)
67     camelCase := s;
68
69     VAR
70         a: ARRAY[1..max] OF STRING;
71         n: INTEGER;
72
73     BEGIN
74
75         ReadStringArray(a, n);
76
77         WriteLn;
78         WriteLn('1. Wert ohne letztem Char: ', WithoutLastChar(a[1]));
79         WriteLn('1./2. Wert gleich (Gorss/Kleinschreibung egal): ',
80         EqualsIgnoreCase(a[1], a[2]));
81         WriteLn('gesamtes Array geCamelCased > ', camelCase(a, n));
82     END.

```

Eingabe	Beschreibung	Ausgabe
siehe Screenshot	allgemeiner Test aller Funktionen	<pre> Anzahl der einzulesenden Woerter > 5 Gib die 5 Woerter ein: 1. > ich 2. > TeStE 3. > mEiNE 4. > funktion 5. > kORRekt 1. Wert ohne letztem Char: ic 1./2. Wert gleich (Gorss/Kleinschreibung egal): FALSE gesamtes Array geCamelCased > ichTesteMeineFunktionKorrekt </pre>
siehe Screenshot	EqualsIgnoreCase testen - funktioniert	<pre> Anzahl der einzulesenden Woerter > 2 Gib die 2 Woerter ein: 1. > serVuS 2. > SERvus 1. Wert ohne letztem Char: serVu 1./2. Wert gleich (Gorss/Kleinschreibung egal): TRUE gesamtes Array geCamelCased > servusServus </pre>
siehe Screenshot	unerlaubtes Zeichen (!)	<pre> Anzahl der einzulesenden Woerter > 4 Gib die 4 Woerter ein: 1. > Ich 2. > teste 3. > meine 4. > Funktion! 1. Wert ohne letztem Char: Ic 1./2. Wert gleich (Gorss/Kleinschreibung egal): FALSE gesamtes Array geCamelCased > Error </pre>
random 200 Zeichen String 2x eingegeben	camelCase schlägt fehl, weil 400 Zeichen nicht in einen String passen	<pre> Anzahl der einzulesenden Woerter > 2 Gib die 2 Woerter ein: 1. > ABCdef123ABCdef123ABCdef123ABCdef123XYZabghiJKL456ghiJKL456ghiJKL456ghiJKL456ghiJKL456MnOpQrstUUVW789rstUUVW789rstUUVW789rstUUVW789QrStUabcXYZ012abcXYZ012abcXYZ012abcXYZ012LmNoP 2. > ABCdef123ABCdef123ABCdef123ABCdef123ABCdef123XYZabghiJKL456ghiJKL456ghiJKL456ghiJKL456ghiJKL456MnOpQrstUUVW789rstUUVW789rstUUVW789rstUUVW789QrStUabcXYZ012abcXYZ012abcXYZ012abcXYZ012LmNoP 1. Wert ohne letztem Char: ABCdef123ABCdef123ABCdef123ABCdef123ABCdef123XYZabghiJKL456ghiJKL456ghiJKL456ghiJKL456ghiJKL456MnOpQrstUUVW789rstUUVW789rstUUVW789rstUUVW789QrStUabcXYZ012abcXYZ012abcXYZ012abcXYZ012LmNo 1./2. Wert gleich (Gorss/Kleinschreibung egal): TRUE gesamtes Array geCamelCased > Error </pre>

3) Kaffeeautomat

3.1 Lösungsidee

Der Großteil der Programmlogik wird in eine Unit verpackt.

Im Hauptprogramm wird wiederholt eine Eingabe durch den in der Unit definierten Datentyp

"Coins" entgegengenommen und damit die Prozedur CoffeeButtonPressed() aufgerufen.

Danach wird das berechnete Wechselgeld mit der Prozedur WriteCoins() ausgegeben. Je nach User-Input wird das Programm wiederholt oder beendet.

Die Prozedur CoffeeButtonPressed() ist in der Unit definiert. Nach Prüfung der Abbruchbedingungen wird mit der Funktion CalcChange das Rückgeld berechnet und zurückgegeben.

Die Funktion CalcChange berechnet das Wechselgeld aus dem Gesamtwert des eingeworfenen Betrages mittels Ganzzahldivision und Restauswertung. Dabei werden immer die größtmöglichen Münzen verwendet. Wenn kein Wechselgeld mehr verfügbar ist, wird der zusätzliche Fehlerparameter "error" auf true gesetzt.

In diesem Fall (wenn error = true) wird eine Fehlermeldung ausgegeben und ein globaler Counter wird erhöht. (Damit nach 3 errors der Automat auf out of service gestellt wird, wird am Anfang der Prozedur geprüft.) Ansonsten wird gemeldet, dass es Kaffee gibt.

In jedem Fall wird dem Ausgangsparameter change das errechnete Wechselgeld zugewiesen.

Im Hauptteil der Unit wird das vorher vorhandene Wechselgeld initialisiert.

3.2.1 Quellcode Unit

```
1  UNIT Kaffeeautomat_unit;
2
3  INTERFACE
4      TYPE
5          Coins = RECORD
6              cent10: INTEGER;
7              cent50: INTEGER;
8              cent100: INTEGER;
9  END;
10
11  PROCEDURE ReadCoins(VAR c: Coins);
12  PROCEDURE WriteCoins(c: Coins);
13
14  PROCEDURE CoffeeButtonPressed(input: Coins; VAR change: Coins);
15
16  IMPLEMENTATION
17
18  VAR
19      totalCoins: Coins;
20      noChangeCounter: INTEGER;
21
22  PROCEDURE ReadCoins(VAR c: Coins);
23  BEGIN
24      Write('10 cent coins > '); ReadLn(c.cent10);
25      Write('50 cent coins > '); ReadLn(c.cent50);
26      Write('100 cent coins > '); ReadLn(c.cent100);
27  END;
28
29  PROCEDURE WriteCoins(c: Coins);
30  BEGIN
31      WriteLn('10 cent coins > ', c.cent10);
32      WriteLn('50 cent coins > ', c.cent50);
33      WriteLn('100 cent coins > ', c.cent100);
34  END;
35
36  FUNCTION AddCoins(a, b: Coins): Coins;
37  BEGIN
38      AddCoins.cent10 := a.cent10 + b.cent10;
39      AddCoins.cent50 := a.cent50 + b.cent50;
40      AddCoins.cent100 := a.cent100 + b.cent100;
41  END;
42
43  FUNCTION SubtractCoins(a, b: Coins): Coins;
```

```

44 BEGIN
45     SubtractCoins.cent10 := a.cent10 - b.cent10;
46     SubtractCoins.cent50 := a.cent50 - b.cent50;
47     SubtractCoins.cent100 := a.cent100 - b.cent100;
48 END;
49
50
51
52 FUNCTION TotalValue(c: Coins): INTEGER;
53 BEGIN
54     TotalValue := c.cent10*10 + c.cent50*50 + c.cent100*100;
55 END;
56
57 FUNCTION CalcChange(c: Coins; VAR error: BOOLEAN): Coins;
58     VAR
59         change: Coins;
60         ValueOfChange: INTEGER;
61
62 BEGIN
63
64     change.cent100 := 0; (*init*)
65     change.cent50 := 0;
66     change.cent10 := 0;
67
68     totalCoins := AddCoins(totalCoins, c); (*Eingabe zu Totalcoins addieren,
zum Stand aktualisieren*)
69
70     ValueOfChange := TotalValue(c)-40; (*Kaffeepreis*)
71
72     change.cent100 := change.cent100 + (ValueOfChange DIV 100);
73     ValueOfChange := ValueOfChange MOD 100; (*Rest für die anderen Münzen*)
74     WHILE change.cent100 > totalCoins.cent100 DO BEGIN
75         dec(change.cent100);
76         change.cent50 := change.cent50 + 2;
77     END; (*while*)
78
79     change.cent50 := change.cent50 + (ValueOfChange DIV 50);
80     ValueOfChange := ValueOfChange MOD 50; (*Rest*)
81     WHILE change.cent50 > totalCoins.cent50 DO BEGIN
82         dec(change.cent50);
83         change.cent10 := change.cent10 + 5;
84     END; (*while*)
85
86     change.cent10 := change.cent10 + (ValueOfChange DIV 10);
87
88     IF change.cent10 > totalCoins.cent10 THEN BEGIN

```

```

89     error := TRUE;
90     inc(noChangeCounter);
91     totalCoins := SubtractCoins(totalCoins, c);
92     CalcChange := c;
93 END ELSE BEGIN
94     totalCoins := SubtractCoins(totalCoins, change);
95     error := FALSE;
96     noChangeCounter := 0;
97     CalcChange := change;
98 END; (*if*)
99 END;
100
101 PROCEDURE CoffeeButtonPressed(input: Coins; VAR change: Coins);
102     VAR
103         error: BOOLEAN;
104 BEGIN
105
106     IF noChangeCounter >= 3 THEN BEGIN
107         change := input;
108         WriteLn('Error - out of service (not enough coins)');
109         (*WriteLn('Rückgeld: ', change);*)
110         Exit;
111     END;
112
113     IF TotalValue(input) < 40 THEN BEGIN
114         change := input;
115         WriteLn('Zu wenig Geld, gibts keinen Kaffee. :(');
116         (*WriteLn('Rückgeld: ');*)
117         (*WriteCoins(change);*)
118         Exit;
119     END; (*if*)
120
121     change := CalcChange(input, error);
122
123     IF error THEN BEGIN (*error: kein Wechselgeld*)
124         WriteLn('Sorry, nicht genug Rueckgeld. Du bekommst dein Geld
zurueck');
125         WriteLn()
126     END ELSE
127         WriteLn('Kaffeeeeeeee!');
128 END;
129
130
131 (*Initialisation, only first time*)
132 BEGIN
133     (*Wechselgeld*)

```

```
134     totalCoins.cent10 := 10;  
135     totalCoins.cent50 := 5;  
136     totalCoins.cent100 := 0;  
137  
138     noChangeCounter := 0;  
139 END.
```

3.2.2 Quellcode Hauptprogramm

```
1  PROGRAM Kaffeeautomat_usingUnit;
2
3  USES
4      Kaffeeautomat_unit;
5
6  VAR
7      input: Coins;
8      change: Coins;
9      userInteraction: STRING;
10     moreCoffee: BOOLEAN;
11
12 BEGIN
13
14     moreCoffee := TRUE; (*init*)
15
16     WHILE moreCoffee DO BEGIN
17
18         WriteLn('Enter your input: ');
19         ReadCoins(input);
20
21         CoffeeButtonPressed(input, change);
22         WriteLn;
23
24         WriteLn('Here is your change: ');
25         WriteCoins(change);
26
27         WriteLn;
28
29         WriteLn('Do you want another coffee? (y/n)');
30         ReadLn(userInteraction);
31
32         IF (Length(userInteraction) > 0) AND ((userInteraction[1] = 'y') OR
33         (userInteraction[1] = 'Y')) THEN
34             moreCoffee := TRUE
35         ELSE
36             moreCoffee := FALSE;
37
38     END; (*while*)
39 END.
```

3.3 Tests

Eingabe (Münzen)	Beschreibung	Ausgabe
10c: 3 50c: 0 1 €: 0	nicht genug Geld	<pre>Enter your input: 10 cent coins > 3 50 cent coins > 0 100 cent coins > 0 Zu wenig Geld, gibts keinen Kaffee. :(Here is your change: 10 cent coins > 3 50 cent coins > 0 100 cent coins > 0 Do you want another coffee? (y/n)</pre>
10c: 3 50c: 1 1 €: 0	genug Geld, Wechselgeld	<pre>Enter your input: 10 cent coins > 3 50 cent coins > 1 100 cent coins > 0 Kaffeeeeeeee! Here is your change: 10 cent coins > 4 50 cent coins > 0 100 cent coins > 0</pre>
10c: 10 50c: 0 1 €: 0	genug Geld, viel Kleingeld (Automat soll große Münzen zurückgeben)	<pre>Enter your input: 10 cent coins > 10 50 cent coins > 0 100 cent coins > 0 Kaffeeeeeeee! Here is your change: 10 cent coins > 1 50 cent coins > 1 100 cent coins > 0</pre>

Eingabe (Münzen)	Beschreibung	Ausgabe
10c: 0 50c: 0 1 €: 1	6 Mal hintereinander eingeben, damit das Rückgeld leer ist (will immer 1 x 50c und 1 x 10c zurückgeben)	<pre> Do you want another coffee? (y/n) y Enter your input: 10 cent coins > 0 50 cent coins > 0 100 cent coins > 1 Kaffeeeeeeee! Here is your change: 10 cent coins > 1 50 cent coins > 1 100 cent coins > 0 Do you want another coffee? (y/n) y Enter your input: 10 cent coins > 0 50 cent coins > 0 100 cent coins > 1 Sorry, nicht genug Rueckgeld. Du bekommst dein Geld zurueck </pre>
10c: 50c: 1 €:	noch 2x versuchen, um den state "out of service" zu triggern	<pre> Do you want another coffee? (y/n) y Enter your input: 10 cent coins > 0 50 cent coins > 0 100 cent coins > 1 Sorry, nicht genug Rueckgeld. Du bekommst dein Geld zurueck Here is your change: 10 cent coins > 0 50 cent coins > 0 100 cent coins > 1 Do you want another coffee? (y/n) y Enter your input: 10 cent coins > 0 50 cent coins > 0 100 cent coins > 1 Sorry, nicht genug Rueckgeld. Du bekommst dein Geld zurueck Here is your change: 10 cent coins > 0 50 cent coins > 0 100 cent coins > 1 Do you want another coffee? (y/n) y Enter your input: 10 cent coins > 0 50 cent coins > 0 100 cent coins > 1 Error - out of service (not enough coins) Here is your change: 10 cent coins > 0 50 cent coins > 0 100 cent coins > 1 </pre>