Inhalt

1 Mittlerer Wert

1.1 Lösungsidee

1. Zuerst wird überprüft, ob a der Mittlere Wert ist. Dabei wird zuerst überprüft, ob b <= a und a <= c ist oder c <= a und a <= b ist. Dabei werden folgende Fälle abgedeckt:
    - a = b = c (alle drei Variablen haben den selben Wert)
    - b < a und a < c
    - c < a und a < b
    - a = c und b ist größer/kleiner als a
    - a = b und c ist größer/kleiner als a

    Wenn einer dieser Fälle gegeben ist, dann ist die Variable a der Mittlere Wert.

2. Wenn die erste Überprüfung negativ ist, dann versuchen wir zu überprüfen, ob b der Median ist. Dabei wird überprüft, ob a <= b und b <= c ist, oder c <= b und b <= a ist. Dabei werden theoretisch folgende Fälle abgedeckt
    - a = b = c (alle drei Variablen haben den selben Wert). Dies kann aber bei diesem Schritt niemals der Fall sein, da dies bereits im Punkt 1 abgefangen wird.
    - a < b und b < c
    - c < b und b < a
    - b = c und a ist größer/kleiner als b
    - b = a und c ist größer/kleiner als b

    Wenn einer dieser Fälle gegeben ist, dann ist die Variable b der Mittlere Wert.
3. Wenn die beiden vorhergehenden Überprüfungen negativ ausfallen, dann kann der Median nur noch die Variable c sein.

1.2 Code

```
PROGRAM MedianCalc;


// Algorithm to get the Median of three integers
// It covers all combinations of integers, a sorting algorithm is not needed,
// always the median will be delivered.
FUNCTION Median (a, b, c : INTEGER): INTEGER;
BEGIN
    IF ( (b <= a) AND ( a <= c) ) OR ( (c <= a) AND (a <= b) ) THEN EXIT(a);

    IF ( (a <= b) AND (b <= c) ) OR ( (c <= b) AND (b <= a) ) THEN EXIT(b);

    EXIT(c);
END;


// Procedure to test the function median
// The input will be a, b, c and the expected value
//
// The Output Format is
// ( a, b, c ) => <expectation> - <OK / FAILED> (got <return value>)
PROCEDURE Test (a, b, c, EXPECTATION : INTEGER);
VAR
    output : INTEGER;
BEGIN
    Write('( ', a, ', ', b, ', ', c,' ) => expect ', EXPECTATION, ' - ');
    output := Median(a, b, c);
    IF output = EXPECTATION THEN Write('OK (got ', output, ')')
    ELSE Write('FAILED (got ', output, ')');
    WriteLn;
END;



BEGIN
    WriteLn('Test 1, 2, 3 in different orders:');
    Test(1, 2, 3, 2);
    Test(3, 1, 2, 2);
    Test(2, 3, 1, 2);
    Test(3, 2, 1, 2);
    Test(1, 3, 2, 2);
    Test(2, 1, 3, 2);

    WriteLn;
    WriteLn('Test -1, -2, -3 numbers:');
```

```
    Test(-1, -2, -3, -2);
    Test(-3, -1, -2, -2);
    Test(-2, -3, -1, -2);
    Test(-3, -2, -1, -2);
    Test(-1, -3, -2, -2);
    Test(-2, -1, -3, -2);

    WriteLn;
    WriteLn('Test negative and positive numbers mixed:');
    Test(0, -10, 400, 0);
    Test(-10, -5, 2, -5);
    Test(32000, -10, -10000, 0);

    WriteLn;
    WriteLn('Test equal numbers:');
    Test(0, 0, 0, 0);
    Test(-10, -10, -10, -10);
    Test(500, 500, 500, 500);

    WriteLn;
    WriteLn('Test if two numbers are equal in different order:');
    Test(1, 1, 2, 1);
    Test(1, 2, 2, 2);
    Test(2, 1, 2, 2);

    WriteLn;
    WriteLn('Test a very large and small numbers');
    Test(32767, 0, 0, 0);
    Test(-32768, 0, 0, 0);


    // These testcases will not comile, because you will get
    // an overflow and underflow, because the datatype integer
    // can only support values between -32768 and 32767
    // -32768 is the minimum, 32767 the maximum
    // Test(32768, 0, 0, 0);
    // Test(-32769, 0, 0, 0);

    // These testcases will not comile, because the read datatype is
    // a character or string

    // WriteLn;
    // WriteLn('Test letters');
    // Test('a', 0, 0, 0);
    // Test('b', 0, 0, 0);
END.
```

1.3 Testfälle

Ausgabe vom Programm mittels Konsole. Hier wurden die Testfälle im Main ausgeführt.

```
Test 1, 2, 3 in different orders:
( 1, 2, 3 ) => expect 2 - OK (got 2)
( 3, 1, 2 ) => expect 2 - OK (got 2)
( 2, 3, 1 ) => expect 2 - OK (got 2)
( 3, 2, 1 ) => expect 2 - OK (got 2)
( 1, 3, 2 ) => expect 2 - OK (got 2)
( 2, 1, 3 ) => expect 2 - OK (got 2)

Test -1, -2, -3 numbers:
( -1, -2, -3 ) => expect -2 - OK (got -2)
( -3, -1, -2 ) => expect -2 - OK (got -2)
( -2, -3, -1 ) => expect -2 - OK (got -2)
( -3, -2, -1 ) => expect -2 - OK (got -2)
( -1, -3, -2 ) => expect -2 - OK (got -2)
( -2, -1, -3 ) => expect -2 - OK (got -2)

Test negative and positive numbers mixed:
( 0, -10, 400 ) => expect 0 - OK (got 0)
( -10, -5, 2 ) => expect -5 - OK (got -5)
( 32000, -10, -10000 ) => expect 0 - FAILED (got -10)

Test equal numbers:
( 0, 0, 0 ) => expect 0 - OK (got 0)
( -10, -10, -10 ) => expect -10 - OK (got -10)
( 500, 500, 500 ) => expect 500 - OK (got 500)

Test if two numbers are equal in different order:
( 1, 1, 2 ) => expect 1 - OK (got 1)
( 1, 2, 2 ) => expect 2 - OK (got 2)
( 2, 1, 2 ) => expect 2 - OK (got 2)

Test a very large and small numbers
( 32767, 0, 0 ) => expect 0 - OK (got 0)
( -32768, 0, 0 ) => expect 0 - OK (got 0)
```

Einige Testfälle waren aber nicht möglich zu kompilieren (siehe Code). Wenn der Wert 32767 überschritten wird, dann haben wir beim Integer einen Überlauf. Wenn der Wert -32768 unterschritten wird, haben wir beim Integer einen Unterlauf. Das ist darauf zurückzuführen, dass der Integer nur 2 Byte (= 16 Bit) groß ist und maximal 65 536 verschiedene Werte haben kann, wobei diese auf einen positiven und negativen Bereich aufgeteilt wurden. 0 ist als positive Zahl zu sehen.

Ein anderer Testfall ist, wenn wir characters (in unserem Fall ‘b‘) und strings (in unserem Fall ‘ad‘) testen. Diese haben nämlich einen ganzen anderen Datentyp. Bei einem Character wäre es eine Möglichkeit, die

Ordinalzahl zu ermitteln, dann würden wir einen Integer zum Vergleichen bekommen. Beim String wäre es noch aufwändiger, da hier die Ordinalzahl von jedem Zeichen ermittelt werden müsste und ein Algorithmus entwickelt werden, damit wir hier eine Sinnvollen Wert zurückbekommen.

2 Multiplikationstabelle

2.1 Lösungsidee

Die Lösung wird unterteilt in eine Prozedur, um die Tabelle auszugeben und den Wert einzulesen.

2.1.1 DisplayTable

1. In die Prozedur werden die Variablen COUNT_ROWS (Anzahl von Zeilen) und COUNT_COLUMNS (Anzahl von Spalten) übergeben. Diese wurden bereits davor überprüft. Die Prozedur wird nur aufgerufen, wenn die Anzahl der Spalten und die Anzahl der Zeilen im Bereich größer gleich 1 und kleiner gleich 20 sind. Dies ist auch aus der Anforderung zu entnehmen.
2. Es wird zuerst über die Anzahl Zeilen iteriert, dabei wird eine Zeile nach der anderen Ausgegeben, bis wir die Maximalanzahl erreicht haben. Dies wir durchgeführt, da wir nicht Spalte für Spalte in die Konsole reinschreiben können.
3. Bei der Iteration Zeile für Zeile wird eine weitere Iteration durchgeführt, und zwar welche Spalte für Spalte Iteriert. In dieser Iteration wird dann die Ausgabe einer einzelnen „Zelle“ durchgeführt.
   => Es wird über die Zeilen iteriert, jede Zeile iteriert die Spalten und das Ergebnis jeder Spalte wird ausgegeben. Das Produkt ergibt sich aus Index der Zeileniteration multipliziert mit dem Index der Spalteniteration.

2.1.2 Hauptprogramm

1. Zuerst wird n (Anzahl der Zeilen) und m (Anzahl der Spalten) der Wert 0 zugewiesen.
2. Es wird überprüft, ob n und m größer/gleich 1 und kleiner/gleich 20 sind. Wenn dieser Fall gegeben ist, wird die Prozedur DisplayTable ausgeführt.
   Wenn die Bedingung nicht zutrifft, wird überprüft, ob n nicht gleich 0 ist. Wenn dies der Fall ist, wir „Ungültige Eingabe“ ausgegeben. Der Grund warum dies implementiert werden sollte ist, da wir im Initialen Zustand für n den Wert 0 haben, welches aber unser Abbruchkriterium ist. Meine Idee zum Programm ist, es über eine REPEAT UNTIL Schleife zu realisieren. Somit wird nichts ausgegeben, wenn der Wert 0 ist und es kann die Eingabe beginnen. Zusätzlich wird der Fall abgebildet, falls die erste Bedingung (1 <= n,m <= 20) nicht zutrifft, dass „Ungültige Ausgabe“ ausgegeben wird.
3. Als nächstes wird die Anzahl der Zeilen eingelesen.
4. Es wird überprüft, ob die Anzahl der Zeilen im Bereich zwischen 1 und 20 ist. Wenn dieser Fall gegeben ist, wird die Anzahl der Spalten eingelesen
5. Es wird überprüft, ob die Anzahl der Zeilen nicht 0 ist, wenn dies der Fall ist, gehe zu Schritt 2.
6. Das Programm wird beendet.

2.2 Code

```
PROGRAM MultiplicationTable;

PROCEDURE DisplayTable (COUNT_ROWS, COUNT_COLUMNS : INTEGER);
VAR
  i, j: INTEGER;
BEGIN

  // Iterate over each row and write the result in every Column
  // In the write, the operator : 4 has been applied, that means
  // that the result is a string with 4 characters. (20 * 20 = 400)
  FOR i := 1 TO COUNT_ROWS DO BEGIN
    FOR j := 1 TO COUNT_COLUMNS DO BEGIN
      Write(i*j : 4);
    END;
    WriteLn;
  END;

  WriteLn;

END;

// n = Count of rows
// m = Count of columns
VAR
  n, m : INTEGER;
BEGIN
  n := 0;
  m := 0;

  REPEAT
    // At the inital run, the prozedure DisplayTable won't run,
    // because n and m are set to 0
    IF (1 <= n) AND (n <= 20) AND (1 <= m) AND (m <= 20) THEN DisplayTable(n, m)
    ELSE IF (n <> 0) THEN BEGIN
      WriteLn('Ungueltige Eingabe!');
      WriteLn;
    END;

    WriteLn('Anzahl der Zeilen:');
    ReadLn(n);

    IF (1 <= n) AND (n <= 20) THEN BEGIN
      WriteLn('Anzahl der Spalten:');
      ReadLn(m);
    END
```

```
  UNTIL (n = 0);

END.
```

2.3 Testfälle

```
Anzahl der Zeilen:
1
Anzahl der Spalten:
1
   1

Anzahl der Zeilen:
20
Anzahl der Spalten:
20
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38  40
   3   6   9  12  15  18  21  24  27  30  33  36  39  42  45  48  51  54  57  60
   4   8  12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72  76  80
   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90  95 100
   6  12  18  24  30  36  42  48  54  60  66  72  78  84  90  96 102 108 114 120
   7  14  21  28  35  42  49  56  63  70  77  84  91  98 105 112 119 126 133 140
   8  16  24  32  40  48  56  64  72  80  88  96 104 112 120 128 136 144 152 160
   9  18  27  36  45  54  63  72  81  90  99 108 117 126 135 144 153 162 171 180
  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170 180 190 200
  11  22  33  44  55  66  77  88  99 110 121 132 143 154 165 176 187 198 209 220
  12  24  36  48  60  72  84  96 108 120 132 144 156 168 180 192 204 216 228 240
  13  26  39  52  65  78  91 104 117 130 143 156 169 182 195 208 221 234 247 260
  14  28  42  56  70  84  98 112 126 140 154 168 182 196 210 224 238 252 266 280
  15  30  45  60  75  90 105 120 135 150 165 180 195 210 225 240 255 270 285 300
  16  32  48  64  80  96 112 128 144 160 176 192 208 224 240 256 272 288 304 320
  17  34  51  68  85 102 119 136 153 170 187 204 221 238 255 272 289 306 323 340
  18  36  54  72  90 108 126 144 162 180 198 216 234 252 270 288 306 324 342 360
  19  38  57  76  95 114 133 152 171 190 209 228 247 266 285 304 323 342 361 380
  20  40  60  80 100 120 140 160 180 200 220 240 260 280 300 320 340 360 380 400

Anzahl der Zeilen:
10
Anzahl der Spalten:
5
   1   2   3   4   5
   2   4   6   8  10
   3   6   9  12  15
   4   8  12  16  20
```

```
   5  10  15  20  25
   6  12  18  24  30
   7  14  21  28  35
   8  16  24  32  40
   9  18  27  36  45
  10  20  30  40  50

Anzahl der Zeilen:
100
Ungueltige Eingabe!

Anzahl der Zeilen:
-1
Ungueltige Eingabe!

Anzahl der Zeilen:
10
Anzahl der Spalten:
-1
Ungueltige Eingabe!

Anzahl der Zeilen:
10
Anzahl der Spalten:
0
Ungueltige Eingabe!

Anzahl der Zeilen:
10
Anzahl der Spalten:
100
Ungueltige Eingabe!

Anzahl der Zeilen:
0
```

Bei einigen Eingaben stürzt das Programm ab, dabei spielt es keine Rolle, ob die Werte für die Anzahl der Spalten oder Anzahl der Zahlen eingegeben wird, da diese den Datentyp INTEGER haben.

Wenn der Wert 32767 überschritten wird, dann haben wir beim Integer einen Überlauf. Wenn der Wert -32768 unterschritten wird, haben wir beim Integer einen Unterlauf. Das ist darauf zurückzuführen, dass der Integer nur 2 Byte (= 16 Bit) groß ist und maximal 65 536 verschiedene Werte haben kann, wobei diese auf einen positiven und negativen Bereich aufgeteilt wurden. 0 ist als positive Zahl zu sehen.

Wenn wir einen Buchstaben oder andere Zeichen dazu mischen, haben wir das Problem, dass die Eingabe kein gültiger Integer mehr ist, sondern ein String.

3 Paare befreundeter Zahlen

3.1 Lösungsidee

3.1.1 CheckIfSingleDivider

Diese Funktion überprüft, ob ein Wert geteilt durch einen Divisor einen Rest ergibt. Dies wird mit dem MOD (Modulo) Operator durchgeführt. Wenn ein Rest vorhanden ist, wird false zurückgeliefert, wenn keiner vorhanden ist, dann wird true zurückgeliefert.

3.1.2 GetDividerSum

Diese Funktion liefert die Summe der echten Teiler zurück, dabei wird wie folgt vorgegangen

1. Setze den Wert der Summe auf 0
2. Zählschleife von 1 bis VALUE – 1. Value – 1 deswegen, weil die Zahl selbst kein echter Teiler ist. In der Zählschleife wird die Funktion CheckIfSingleDivider aufgerufen. Dieser werden die Werte VALUE und der index der Zählschleife übergeben. Wenn True zurückgeliefert wird, dann wird der Index (in diesem Fall auch der Teiler der Zahl) zur Summe addiert.
3. Liefert die Summe zurück

3.1.3 CheckAndOutputSingle Number

Diese Prozedur berechnet ruft die Funktion GetDividerSum auf, um die Teilersumme zu berechnen und gibt die Teiler in der Konsole aus im Format Summe der echten Teiler von <Zahl> : <Teiler> = <Summe>

3.1.4 CheckIfNumbersAreFriends

Diese Prozedur bekommt 2 Werte übergeben, holt sich die Teilersumme ab und vergleicht, ob die beiden gleich sind. Initial wird der Wert 0 gesetzt, wenn beide Zahlen 0 sind, sind die Zahlen ebenfalls nicht befreundet.

3.2 Code

```
PROGRAM Friends;

FUNCTION CheckIfSingleDivider (VALUE, DIVIDER : INTEGER) : BOOLEAN;
BEGIN
  IF (VALUE MOD DIVIDER) = 0 THEN EXIT(TRUE);
  EXIT(FALSE)
END;


FUNCTION GetDividerSum (VALUE : INTEGER) : INTEGER;
VAR
  sum, i : INTEGER;
BEGIN
  sum := 0;

  FOR i := 1 TO VALUE - 1 DO BEGIN
    IF CheckIfSingleDivider(VALUE, i) THEN sum := sum + i;
```

```
  END;


  EXIT(sum);
END;


// This procedure get the divider sum of a number and outputs it in the format
// Summe der echten Teiler von <Number> : 1 + <all dividers> = <DIVIDER_SUM>
PROCEDURE CheckAndOutputSingleNumber (VALUE : INTEGER; VAR DIVIDER_SUM :
INTEGER);
VAR
  i : INTEGER;
BEGIN
  DIVIDER_SUM := GetDividerSum(VALUE);

  Write('Summe der echten Teiler von ', VALUE, ' : 1 ');

  FOR i := 2 TO (VALUE - 1) DO BEGIN
    IF CheckIfSingleDivider(VALUE, i) THEN Write('+ ', i, ' ');
  END;

  Write('= ', DIVIDER_SUM);
  WriteLn;
END;


// This procedure gets two values and check if they are friends
// Also special cases are implemented
PROCEDURE CheckIfNumbersAreFriends (VALUE_1, VALUE_2 : INTEGER);
VAR
  DIVIDER_SUM_1 : INTEGER;
  DIVIDER_SUM_2 : INTEGER;
BEGIN
  DIVIDER_SUM_1 := 0;
  DIVIDER_SUM_2 := 0;

  IF (VALUE_1 > 1) THEN CheckAndOutputSingleNumber(VALUE_1, DIVIDER_SUM_1)
  ELSE WriteLn('Der Wert ', VALUE_1, ' hat keine Echten Teiler!');
  IF (VALUE_2 > 1) THEN CheckAndOutputSingleNumber(VALUE_2, DIVIDER_SUM_2)
  ELSE WriteLn('Der Wert ', VALUE_2, ' hat keine Echten Teiler!');

  IF (DIVIDER_SUM_1 = 0) OR (DIVIDER_SUM_2 = 0) THEN WriteLn('Die Zahlen sind
nicht befreundet!')
  ELSE IF (VALUE_1 = DIVIDER_SUM_2) OR (DIVIDER_SUM_1 = VALUE_2) THEN
WriteLn('Die Zahlen sind befreundet!')
  ELSE WriteLn('Die Zahlen sind nicht befreundet');
```

```
END;


VAR
  WERT_1, WERT_2 : INTEGER;
BEGIN
  WriteLn('Erste Zahl:');
  ReadLn(WERT_1);

  WriteLn('Zweite Zahl');
  ReadLn(WERT_2);

  CheckIfNumbersAreFriends(WERT_1, WERT_2);
END
.
```

3.3 Testfälle

```
Erste Zahl:
284
Zweite Zahl
220
Summe der echten Teiler von 284 : 1 + 2 + 4 + 71 + 142 = 220
Summe der echten Teiler von 220 : 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 +
110 = 284
Die Zahlen sind befreundet!

Erste Zahl:
1184
Zweite Zahl
1210
Summe der echten Teiler von 1184 : 1 + 2 + 4 + 8 + 16 + 32 + 37 + 74 + 148 + 296
+ 592 = 1210
Summe der echten Teiler von 1210 : 1 + 2 + 5 + 10 + 11 + 22 + 55 + 110 + 121 +
242 + 605 = 1184
Die Zahlen sind befreundet!

Erste Zahl:
2620
Zweite Zahl
2924
Summe der echten Teiler von 2620 : 1 + 2 + 4 + 5 + 10 + 20 + 131 + 262 + 524 +
655 + 1310 = 2924
Summe der echten Teiler von 2924 : 1 + 2 + 4 + 17 + 34 + 43 + 68 + 86 + 172 + 731
+ 1462 = 2620
Die Zahlen sind befreundet!
```

```
Erste Zahl:
220
Zweite Zahl
-284
+ 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284
Der Wert -284 hat keine Echten Teiler!
Die Zahlen sind nicht befreundet!

Erste Zahl:
0
Zweite Zahl
1
Der Wert 0 hat keine Echten Teiler!
Der Wert 1 hat keine Echten Teiler!
Die Zahlen sind nicht befreundet!

Erste Zahl:
200
Zweite Zahl
300
Summe der echten Teiler von 200 : 1 + 2 + 4 + 5 + 8 + 10 + 20 + 25 + 40 + 50 +
100 = 265
Summe der echten Teiler von 300 : 1 + 2 + 3 + 4 + 5 + 6 + 10 + 12 + 15 + 20 + 25
+ 30 + 50 + 60 + 75 + 100 + 150 = 568
Die Zahlen sind nicht befreundet

Erste Zahl:
1
Zweite Zahl
2
Der Wert 1 hat keine Echten Teiler!
Summe der echten Teiler von 2 : 1 = 1
Die Zahlen sind nicht befreundet!
```

Es gibt auch Testfälle, wo das Programm abstürzt, etwa wenn der Integer überschritten wird oder ein Buchstabe eingelesen wird. Auch Fälle wie die Zahlen 12599 und 12598 Lösen einen Absturz aus, da hier der Integer überschritten wird.