1 Aufgabe 1 (MaximumRetriever)

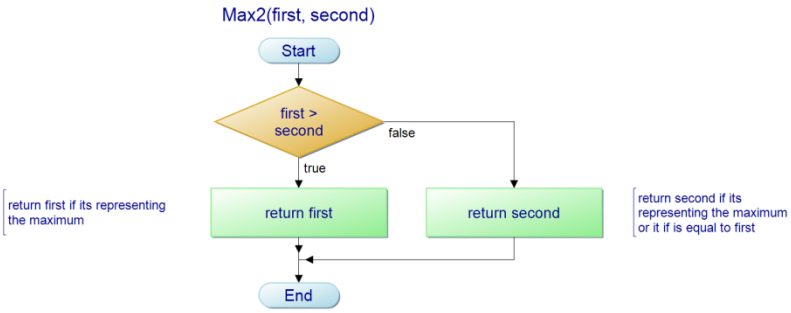
Folgend ist die Dokumentation für die Aufgabe 1 (MaximumRetriever) angeführt. Hierbei soll über drei Algorithmen jeweils einmal das Maximum zweier Zahlen (Max2) und zweimal das Maximum dreier Zahlen ermittelt werden (Max3a, Max3b), wobei die dritte Variante (Max3b) den Algorithmus der ersten Variante (Max2) wiederverwenden soll.

1.1 Lösungsidee

Folgend werden die Lösungsideen der Aufgabe 1 beschrieben. Die Lösungsidee wird in Form eines Ablaufdiagrams beschrieben.

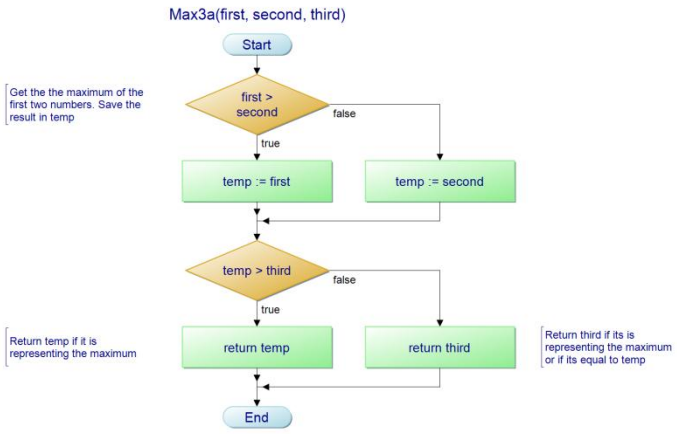
1.1.1 Max2 (Funktion zum Ermitteln des Maximums zweier Zahlen)

Folgend ist die Lösungsidee des Algorithmus zum Ermitteln des Maximums zweier Zahlen mittels einem Ablaufdiagram beschrieben. Auf eine Beschreibung mittels Prosa wird verzichtet, da dieser Algorithmus relativ trivial ist.



1.1.2 Max3a (Funktion zum Ermitteln des Maximums dreier Zahlen)

Folgend ist die Lösungsidee des Algorithmus zum Ermitteln des Maximums dreier Zahlen mittels Ablaufdiagram beschrieben. Hier wird ebenfalls auf eine Beschreibung mittels einer Prosa verzichtet.



1.1.3 Max3b (Funktion zum Ermitteln des Maximums dreier Zahlen via Max2)

Folgend ist die Lösungsidee des Algorithmus zum Ermitteln des Maximums dreier Zahlen mittel Prosa beschrieben. Diese Funktion verwendet die vorherig beschriebene Funktion Max2. Hier erfolgt die Beschreibung mittels Prosa, da die Kernfunktionalität schon vorherig im Algorithmus Max2 beschrieben wurde.

Step 1. Maximum erster beiden Zahlen ermitteln

Ermittele das Maximum von den ersten beiden Zahlen via *Max2*.

Speichere das Ergebnis es in *temp* un gehe zu Schritt 2.

Step 2. Maximum ermitteln

Ermittele das Maximum von *temp und third* via *Max2*.

Gibt das ermittelte Maximum zurück.

In diesem Fall ist ersichtlich, das beim Wiederverwenden von Max2 lediglich nur eine Zeile Code benötigt wird um den Algorithmus zu realisieren, wobei bei Max3a eine Codeverdoppelung stattfindet.

1.2 Source

Folgend ist der Source des Programms MaximumRetriever angeführt.

```
{
  This program contains three functions which are used
  to get the maximum of several numbers.

  Contained functions:
  ----------------------------------------
  1. Max2(first, second: INTEGER): INTEGER;
     Gets the maximum of two numbers.

  2. Max3a(first, second, third: INTEGER): INTEGER;
     Gets the maximum of three numbers.

  3. Max3b(first, second, third: INTEGER): INTEGER;
     Gets the maximum of three numbers by reusing the function 'Max3a'.
}
PROGRAM MaximumRetriever;

  {
    Function which gets the maximum of two numbers.
    @param first:
                the first number for the maximum retrieval
    @param second:
                the second number for the maximum retrieval
    @return
          the maximum of the two given numbers
  }
  FUNCTION Max2(first, second: INTEGER): INTEGER;
  BEGIN
    IF first > second THEN
      Max2 := first
    ELSE
      Max2 := second
  END;

  {
    Function which gets the maximum of three numbers.
    @param first:
```

```
                the first number for the maximum retrieval
    @param second:
                the second number for the maximum retrieval
    @param third:
                the third number for the maximum retrieval
    @return
          the maximum of the three given numbers
  }
  FUNCTION Max3a(first, second, third: INTEGER): INTEGER;
  VAR
    temp: INTEGER;
  BEGIN
    {
      Copy result of the first comparison to 'temp' which represents
      the maximum of the first two numbers.
      This result gets later compared to 'third' to get the maximum of 'temp' and 'third'.
    }
    IF first > second THEN
      temp := first
    ELSE
      temp := second;

    IF temp > third THEN
      Max3a := temp
    ELSE
      Max3a := third
  END;


  {
    Function which gets the maximum of three numbers, by reusing the function Max3a.
    @param first:
                the first number for the maximum retrieval
    @param second:
                the second number for the maximum retrieval
    @param third:
                the third number for the maximum retrieval
    @return
          the maximum of the three given numbers
    @see
        Max2(first, second: INTEGER): INTEGER
        Function which gets the maximum of two numbers
  }
  FUNCTION Max3b(first, second, third: INTEGER): INTEGER;
  BEGIN
    Max3b := Max2(Max2(first, second), third)
  END;

BEGIN
  {
    Tests for function Max2
  }
  WriteLn('---------------------------------------');
  WriteLn('1. Test for Max2: first: 10 / second: 13');
  WriteLn('Maximum: ', Max2(10, 13));
  WriteLn('---------------------------------------');
  WriteLn('2. Test for Max2: first: 13 / second: 10');
  WriteLn('Maximum: ', Max2(13, 10));
  WriteLn('---------------------------------------');
  WriteLn('3. Test for Max2: first: 10 / second: 10');
  WriteLn('Maximum: ', Max2(10, 10));
  WriteLn('---------------------------------------');
  WriteLn('3. Test for Max2: first: -10 / second: -20');
  WriteLn('Maximum: ', Max2(-10, -20));
  WriteLn('---------------------------------------');

  {
    Tests for function Max3a
  }
  WriteLn();
  WriteLn('---------------------------------------');
  WriteLn('1. Test for Max3a: first: 10 / second: 13 / third: 4');
  WriteLn('Maximum: ', Max3a(10, 13, 4));
  WriteLn('---------------------------------------');
  WriteLn('3. Test for Max3a: first: 4 / second: 10 / third: 13');
  WriteLn('Maximum: ', Max3a(4, 10, 13));
```

```
  WriteLn('--------------------------------------');
  WriteLn('2. Test for Max3a: first: 10 / second: 10 / third: 4');
  WriteLn('Maximum: ', Max3a(10, 10, 4));
  WriteLn('--------------------------------------');
  WriteLn('2. Test for Max3a: first: 10 / second: 10 / third: 10');
  WriteLn('Maximum: ', Max3a(10, 10, 10));
  WriteLn('--------------------------------------');
  WriteLn('2. Test for Max3a: first: -10 / second: -3 / third: -10');
  WriteLn('Maximum: ', Max3a(-10, -3, -10));
  WriteLn('--------------------------------------');

  {
    Tests for function Max3b
  }
  WriteLn();
  WriteLn('--------------------------------------');
  WriteLn('1. Test for Max3b: first: 10 / second: 13 / third: 4');
  WriteLn('Maximum: ', Max3b(10, 13, 4));
  WriteLn('--------------------------------------');
  WriteLn('3. Test for Max3b: first: 4 / second: 10 / third: 13');
  WriteLn('Maximum: ', Max3b(4, 10, 13));
  WriteLn('--------------------------------------');
  WriteLn('2. Test for Max3b: first: 10 / second: 10 / third: 4');
  WriteLn('Maximum: ', Max3b(10, 10, 4));
  WriteLn('--------------------------------------');
  WriteLn('2. Test for Max3b: first: 10 / second: 10 / third: 10');
  WriteLn('Maximum: ', Max3b(10, 10, 10));
  WriteLn('--------------------------------------');
  WriteLn('2. Test for Max3b: first: -10 / second: -3 / third: -10');
  WriteLn('Maximum: ', Max3b(-10, -3, -10));
  WriteLn('--------------------------------------');
END.
```

1.3 Tests

Folgend sind die Tests der implementierten Algorithmen von Max2, Max3a und Max3b beschrieben. Ein Test für einen 'Range Overflow' oder 'Invalid Number Format' konnte hier nicht getestet werden, da keine Benutzereingaben erfolgen können und bereits zur Kompilierungszeit der Code für die Tests evaluiert wird und bereits hier sich das Programm bei einer Werteüberschreitung oder ungültigen Werteangaben nicht kompilieren lässt.

Testszenarien:

1. Einige Permutationen der Eingangswerte
2. Gleiche Werte für die Eingangswerte
3. Negative Zahlenwerte für die Eingangswerte

Permutationen der Eingangswerte:

Folgend werden die Funktionen Max2, Max3a und Max3b mit Permutationen der Eingangswerte getestet, um herauszufinden ob die Reihenfolge der Zahlenwerte für die Funktionalitäten der Algorithmen eine Rolle spielt.

1. Max2

```
-------------------------------------
1. Test for Max2: first: 10 / second: 13
Maximum: 13
-------------------------------------
2. Test for Max2: first: 13 / second: 10
Maximum: 13
-------------------------------------
```

2. Max3a

```
-------------------------------------
1. Test for Max3a: first: 10 / second: 13 / third: 4
Maximum: 13
-------------------------------------
3. Test for Max3a: first: 4 / second: 10 / third: 13
Maximum: 13
-------------------------------------
```

3. Max3b

```
-------------------------------------
1. Test for Max3b: first: 10 / second: 13 / third: 4
Maximum: 13
-------------------------------------
3. Test for Max3b: first: 4 / second: 10 / third: 13
Maximum: 13
-------------------------------------
```

Wie zu erwarten spielt die Reihenfolge der Eingangswerte für die Algorithmen keine Rolle.

Gleiche Werte der Eingangsparameter:

Folgend werden die Funktionen Max2, Max3a und Max3b mit gleichen Zahlenwerten der Eingangswerte getestet.

1. Max2

```
-------------------------------------
3. Test for Max2: first: 10 / second: 10
Maximum: 10
-------------------------------------
```

2. Max3a

```
-------------------------------------
3. Test for Max2: first: 10 / second: 10
Maximum: 10
-------------------------------------
```

3. Max3b

```
-------------------------------------
2. Test for Max3b: first: 10 / second: 10 / third: 4
Maximum: 10
-------------------------------------
2. Test for Max3b: first: 10 / second: 10 / third: 10
Maximum: 10
-------------------------------------
```

Auch hier wird das richtige Maximum ausgegeben, wobei anzumerken ist, dass das Maximum bei gleichen Werten immer über die letze geprüfte Zahl ausgegeben wird.

Negative Zahlen als Eingangswerte:

Folgend werden die Funktionen Max2, Max3a und Max3b mit negativen Eingangswerten getestet.

1. Max2

```
-------------------------------------
3. Test for Max2: first: -10 / second: -20
Maximum: -10
-------------------------------------
```

2. Max3a

```
-------------------------------------
2. Test for Max3a: first: -10 / second: -3 / third: -10
Maximum: -3
-------------------------------------
```

3. Max3b

```
-------------------------------------
2. Test for Max3b: first: -10 / second: -3 / third: -10
Maximum: -3
-------------------------------------
```

Es wird auch bei negativen Eingabewerten das korrekte Maximum zurückgegeben.

2 BinomialCalculator

Folgend ist die Dokumentation der Aufgabe 2 angeführt. Hierbei soll die binomiale Formel (a + b)^2 über die Grenzen minA <= a <= maxA und 1 <= b <= 10 berechnet werden und die Ergebnisse in einer Tabelle ausgegeben werden.

2.1 Lösungsidee

Folgend ist die Lösungsidee der Aufgabe 2 beschrieben. Dieses Mal wurde auf eine Prosa verzichtet, da diese hier zu umfangreich werden würde. Es sei auf die Ablaufdiagramme und die Dokumentation im Source verwiesen. Ablaufdiagrame werden auf die drei wichtigsten Funktionen beschränkt:

1. Power
2. CalcBinomial
3. DisplayBFTable

Folgend sind die implementierten Subalgorithmen beschrieben, die für die Realisierung des Algorithmus der Aufgabe 2 verwendet werden.

Funktionen:

1. *ToString(value: INTEGER): STRING;*

   Konvertiert ein Integer zu seiner String Repräsentation.

2. *RepeatString(value: STRING; count: INTEGER): STRING;*

   Erstellt einen String mit der *count* mal Wiederholung von *value*.

   Für das Erstellen des Begrenzers zwischen Tabellenkopf und Tabelleninhalt und für die Erweiterung der Tabellenspalten Texte um das Design der Tabelle zu realisieren.

3. *EnhanceStringLength(value: STRING; width: INTEGER): STRING;*

   Füllt einen String mit Leerzeichen auf bis der String die definierte Länge aufweist.

   Verwendet *RepeatString* um Leerzeichen an den String anzufügen.

4. *GetMaximumOfTwo(first, second: INTEGER): INTEGER;*

   Ermittelt das Maximum zwischen zwei Zahlen. Wird verwendet für das Ermitteln der maximalen Länge der enthaltenen Spaltentexte.

5. *Power(value, n: INTEGER): INTEGER;*

   Potenziert *value* so oft wie von *n* vorgegeben.

6. *CalcBinomial(value, n: INTEGER): INTEGER;*

   Berechnet die binomiale Formel (a + b)^2

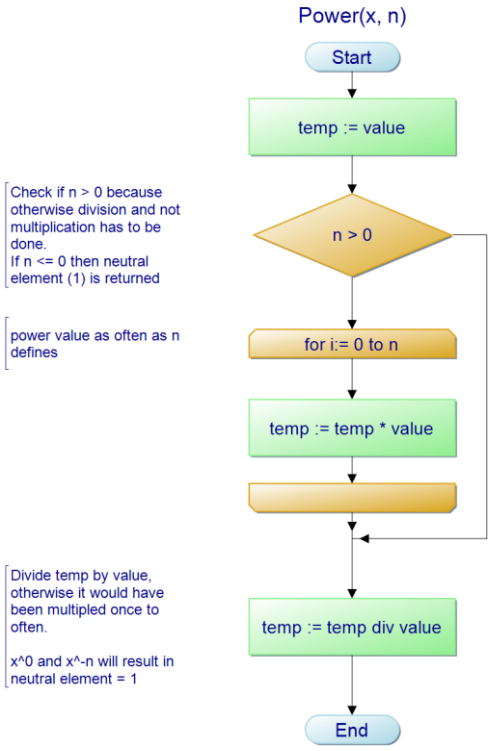
7. *DisplayBFTable (minA, maxA: INTEGER);*

   Gibt die Ergebnisse der Berechnungen in einer Tabelle aus.

2.1.1 Power (Potenziert den gegeben Wert)

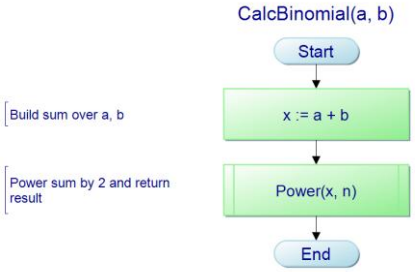
Folgend ist das Ablaufdiagram des Algorithmus der Funktion Power angeführt.

Dieser potenziert einen gegebenen Wert so wie durch n vorgegeben. Negative Potenzen können nicht behandelt werden, da in diesem Fall eine Division und keine Multiplikation von value stattfinden müsste. Bei negativen Potenzen wird das neutrale Element (1) zurückgegeben.



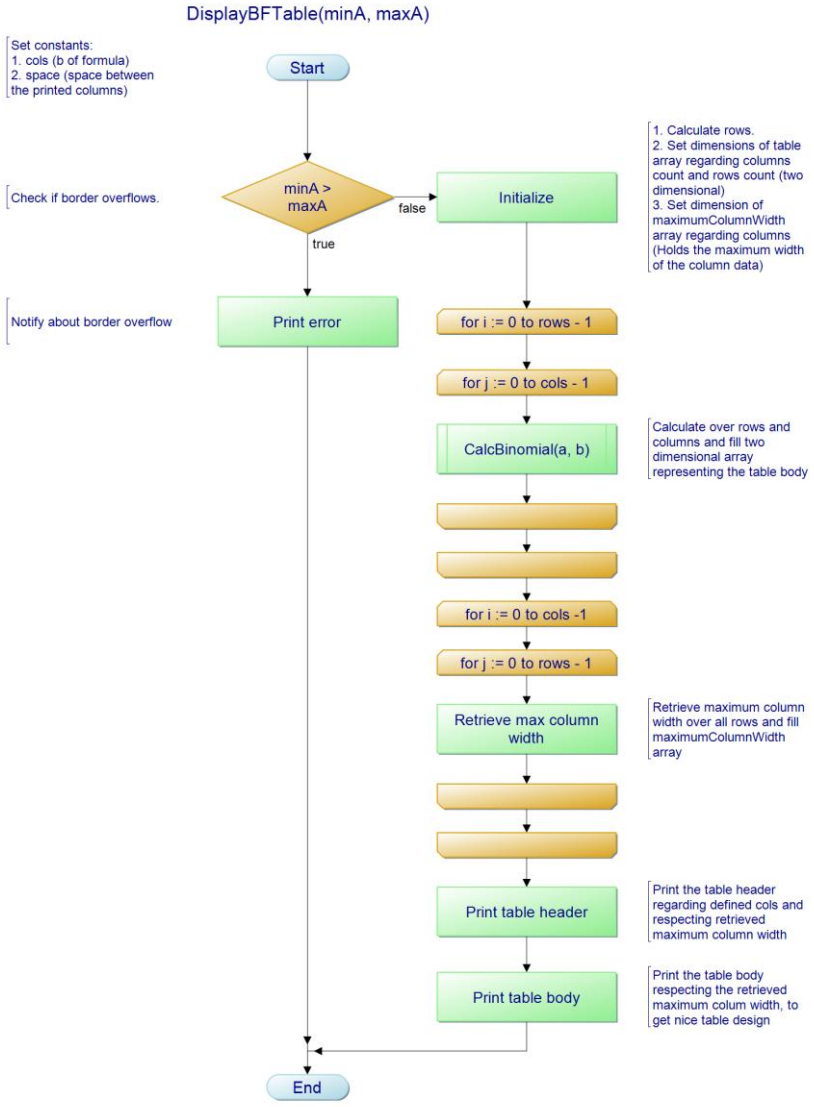
2.1.2 CalcBinomial (Berechnet (a + b)^2)

Folgend ist das Ablaufdiagram des Algorithmus der Funktion CalcBinomial angeführt, die die Formel (a + b)^2 berechnet. Dieser Algorithmus verwendet den vorherig beschriebenen Algorithmus Power.



2.1.3 DisplayBFTable (Gibt Ergebnisse aus)

Folgend ist das Ablaufdiagram des Algorithmus der Prozedur DisplayBFTable angeführt. Diese Funktion gibt die Ergebnisse in einer Tabelle in der Konsole aus. Es wurden Subalgorithmen eingeführt, um das Tabellendesign zu verbessern. Für die Beschreibung dieser Funktionen sei auf die Dokumentation im Source verwiesen.



2.2 Source

Folgend ist der Source des Programms BinomialCalculator angeführt.

```
{
  This program is used to calculate the binomial formula (a + b)^2.
  The result will be displayed as a table in the console, where the table will be
  displayd with a well design.
  This design is realized by calculating the maximum value width of a column and enhancing the
  converted results strings to fit this width.

  Contained util functions:
  -------------------------
  1. ToString(value: INTEGER): STRING;
     Converts a integer to a string

  2. RepeatString(value: STRING; count: INTEGER): STRING;
     Repeats the given string count itmes

  3. EnhanceStringLength(value: STRING; width: INTEGER): STRING;
     Enahnces string width to fit specified width

  4. GetMaximumOfTwo(first, second: INTEGER): INTEGER;
     Gets the maximum of two numbers

  5. Power(value, n: INTEGER): INTEGER;
     Powers the given value by n

  6. CalcBinomial(a, b: INTEGER): INTEGER;
     Calculates (a + b)^2
}
PROGRAM BinomialCalculator;

  {
    Converts a integer to a string representation.
    @param value:
              the integer to be converted to a string
    @paaram the string representation of the given integer value
  }
  FUNCTION ToString(value: INTEGER): STRING;
  VAR
    s: STRING;
  BEGIN
   Str(value, s);
   ToString := s
  END;

  {
    Repeats the given string as often as 'count' defines.
    @param value:
              the given string to be repeated
    @param count:
              defines the number of repeations
    @return the created string representing the repeated given string 'value'.
    If 'count <= 0' then a empty string is returned
  }
  FUNCTION RepeatString(value: STRING; count: INTEGER): STRING;
  VAR
    i: INTEGER;
    str: STRING;
  BEGIN
    str := '';
    IF count > 0 THEN
      FOR i := 1 TO count DO
        str := str + value;

    RepeatString := str
  END;

  {
    Builds the string representation for the column and enhances the
    given string 'value' length to the length defined by 'width' by adding spaces
    to the given string.
    @param value:
              the given string to be added into the column string
```

```
  @param width:
            the intended length of the given string
  @return the created string, if width smaller then 'value' length, then 'value' is returned
  @see RepeatString(value: STRING; count: INTEGER): STRING;
}
FUNCTION EnhanceStringLength(value: STRING; width: INTEGER): STRING;
VAR
  len: INTEGER;
  str: STRING;
BEGIN
  str := value;
  len := width - Length(str);

  IF len > 0 THEN
    str := str + RepeatString(' ', len);

  EnhanceStringLength := str;
END;

{
  Function which gets the maximum of two numbers.
  @param first:
            the first number for the maximum retrieval
  @param second:
            the second number for the maximum retrieval
  @return the maximum of the two given numbers
}
FUNCTION GetMaximumOfTwo(first, second: INTEGER): INTEGER;
BEGIN
  IF first > second THEN
    GetMaximumOfTwo := first
  ELSE
    GetMaximumOfTwo := second
END;

{
  Calculates 'value powered by n'.
  @param value:
            the value to powered
  @param n:
          the count how often the given 'value' shall be powered
  @return the given 'value powered by n', where when 'n <= 0' then the neutral
          element is returned, which is 1
}
FUNCTION Power(value, n: INTEGER): INTEGER;
VAR
  temp, i: INTEGER;
BEGIN
  {
    Calculates the 'value powered by n' where the result is divided by the
    given value to get the proper result.
  }
  temp := value;
  IF n > 0 THEN
    FOR i := 1 TO n DO
      temp := temp * value;

  Power := temp DIV value;
END;

{
  Calculates the binomial formula (a + b)^2.
  @param a:
        the first value
  @param b:
        the second value
  @return the result of the formula
  @see Power(value ,n: INTEGER): INTEGER
}
FUNCTION CalcBinomial(a, b: INTEGER): INTEGER;
BEGIN
  CalcBinomial := Power(a + b, 2);
END;

{
  Prints the result of the calculated formula in a table.
```

```
    The value 'b' in the formula will have values from '1 to 10', see CONST section of this
procedure.
    @param minA:
            the bottom border of the value 'a' in the formula
    @param maxA:
            the top border of the value 'a' in the formula

    @see CalcBinomial(a, b: INTEGER): INTEGER;
    @see GetMaximumOfTwo(first, second: INTEGER): INTEGER;
    @see EnhanceStringLength(value: STRING; width: INTEGER): STRING;
    @see RepeatString(value: STRING; count: INTEGER): STRING;
    @see ToString(value: INTEGER): STRING;
  }
  PROCEDURE DisplayBFTable(minA, maxA: INTEGER);
  VAR
    {
      Two dimensional array which holds the calculated values.
    }
    table: ARRAY OF ARRAY OF INTEGER;
    {
      Array which holds the calculated maximum column width,
      which is needed to create the column strings width a proper width,
      so that the printed table will have a well design
    }
    maxColumWidth: ARRAY OF INTEGER;
    {
      i, j: iteration variables;
      rows: the calculated rows of the table
      maxWidth: Used for determining the maximum width of the column values
    }
    i, j, rows, maxWidth: INTEGER;
    {
      Used for building the printed lines of the table
    }
    line: STRING;
  CONST
    {
      Constant for the columns, representing 'b' in the formula (a + b)^2
    }
    cols = 10;
    {
      Constant for the space between the columns
    }
    space = 3;
  BEGIN
    WriteLn();
    WriteLn('Calculating (a + b)^2 from : minA: ', minA, ' to maxA: ', maxA, ' / b: 1-',
cols);
    {
      Check range of actual parameters and avoid border overflow.
    }
    IF minA > maxA THEN
      WriteLn('MinA is not supposed to overflow maxA: || minA: ', minA, ' / maxA: ', maxA)
    ELSE BEGIN
      {
       Calculate rows and set array dimensions.
      }
      rows := maxA - minA + 1;
      setLength(table, rows, cols);
      setLength(maxColumWidth, cols);

      {
        Calculate table values, and fill two dimensional
        array which represents the table
      }
      FOR i := 0 TO rows - 1 DO
        FOR j := 0 TO cols - 1 DO
          table[i, j] := CalcBinomial((minA + i), ( j + 1));

      {
        Calculate the maximum width of the to display column, by comparision of the column
        contained value width of the string representation of the hold value.
      }
      FOR i := 0 TO cols - 1 DO BEGIN
        maxWidth := 0;
        FOR j := 0 TO rows - 1 DO
```

```
        maxWidth := GetMaximumOfTwo(maxWidth, Length(ToString(table[j, i])) + space);
      maxColumWidth[i] := maxWidth
    END;

    {
      Prints the table header and sets the column width to a proper width
      by adding spaces to the string representation of the value to fit intended
      column width, calculated before
    }
    line := EnhanceStringLength('a / b', 5) + ' | ';
    FOR i := 0 TO cols - 1 DO
      line := line + EnhanceStringLength(ToString(i + 1), maxColumWidth[i]);
    WriteLn(RepeatString('-', Length(line)));
    WriteLn(line);
    WriteLn(RepeatString('-', Length(line)));

    {
      Print the table content, where also the string representation of the value
      will be set to a width to fit the before calculated column width.
    }
    FOR i := 0 TO rows -1 DO BEGIN
      line := EnhanceStringLength(ToString(minA + i), 5) + ' | ';
      FOR j := 0 TO cols - 1 DO
        line := line + EnhanceStringLength(ToString(table[i, j]), maxColumWidth[j]);
      WriteLn(line)
    END;
    WriteLn(RepeatString('-', Length(line)))
  END;
END;


VAR
  result: INTEGER;
BEGIN
  {
    Tests for DisplayBFTable
  }
  DisplayBFTable(8, 8);
  DisplayBFTable(8, 13);
  DisplayBFTable(15, 20);
  DisplayBFTable(8, 7);
  DisplayBFTable(21, 30);
END.
```

2.3 Tests

Folgend sind die Test angeführt, die den Algorithmus des Programms binomialCalculator testen.
Auf die Tests der Hilfsfunktionen wurde hierbei verzichtet, da sich Fehler in diesen Funktionen in den unten angeführten Tests niederschlagen würden.

Verschieden Grenzen:

Folgend sind die Tests für verschiedene Grenzen (minA, maxA) angeführt, die zeigen das sich der Algorithmus und die verwendeten Funktion wie erwartet verhalten.

```
Calculating (a + b)^2 from : minA: 8 to maxA: 8 / b: 1-10
-------------------------------------------------------------------
a / b | 1    2     3     4     5     6     7     8     9     10
-------------------------------------------------------------------
8     | 81   100   121   144   169   196   225   256   289   324
-------------------------------------------------------------------

Calculating (a + b)^2 from : minA: 8 to maxA: 13 / b: 1-10
-------------------------------------------------------------------
a / b | 1     2     3     4     5     6     7     8     9     10
-------------------------------------------------------------------
8     | 81    100   121   144   169   196   225   256   289   324
9     | 100   121   144   169   196   225   256   289   324   361
10    | 121   144   169   196   225   256   289   324   361   400
11    | 144   169   196   225   256   289   324   361   400   441
12    | 169   196   225   256   289   324   361   400   441   484
13    | 196   225   256   289   324   361   400   441   484   529
-------------------------------------------------------------------

Calculating (a + b)^2 from : minA: 15 to maxA: 20 / b: 1-10
-------------------------------------------------------------------
a / b | 1     2     3     4     5     6     7     8     9     10
-------------------------------------------------------------------
15    | 256   289   324   361   400   441   484   529   576   625
16    | 289   324   361   400   441   484   529   576   625   676
17    | 324   361   400   441   484   529   576   625   676   729
18    | 361   400   441   484   529   576   625   676   729   784
19    | 400   441   484   529   576   625   676   729   784   841
20    | 441   484   529   576   625   676   729   784   841   900
-------------------------------------------------------------------
```

Grenzbereichsüberschreitung:

Folgend sind die Tests angeführt, die das Verhalten testen wenn eine Grenzbereichsüberschreitung auftritt. Hierbei wird auf die Konsole eine dementsprechende Benachrichtigung ausgegeben.

```
Calculating (a + b)^2 from : minA: 8 to maxA: 7 / b: 1-10
MinA is not supposed to overflow maxA: || minA: 8 / maxA: 7
```

Wertebereichsüberschreitung:

Folgend ist der Test angeführt, der das Verhalten bei einer Grenzbereichsüberschreitung testet. Da die Tests im Source enthalten sind und hierbei keine Werte über die Konsole eingegeben werden können, die den Wertebereich überschreiten könnten (Kompilierungsfehler), kann diese Art von Fehler nur zur Laufzeit des Programms auftreten, wenn eines der berechneten Ergebnisse den Wertebereich überschreitet. Es wurde hierbei INTEGER und nicht LONGINT verwendet, wobei ein LONGINT eine Berechnung über einen größeren Wertebereich erlauben würde.

```
Calculating (a + b)^2 from : minA: 21 to maxA: 30 / b: 1-10
Runtime error 201 at $00401603
  $00401603
  $00404B5A
```

3 Aufgabe 3 (TimeConverter)

Folgend ist die Dokumentation der Aufgabe 3 angeführt. Hierbei sollen die Algorithmen eine Sekundendarstellung in eine Zeitdarstellung und umgekehrt konvertieren.

3600 -> 1:0:0 -> 3600

3.1 Lösungsidee

Folgend ist die Lösungsidee der beiden Algorithmen TimeSpanToSeconds und SecondstoTimeSpan angeführt, welche über Ablaufdiagrame ausgedrückt wird.

Enthaltene Funktionen und Prozeduren:

1. *ToString(value: INTEGER): STRING;*

   Konvertiert ein Integer zu seiner String Repräsentation

2. *ValidateRange(value, bottom, top: INTEGER): BOOLEAN;*

   Prüft ob der gegeben Wert innerhalb der definierten Grenzen liegt..

3. *GetErrorMessage(code: INTEGER): STRING;*

   Generiert die entsprechende Error Message abhängig vom gegebenen Error code.

4. *TestTimeSpanToSeconds(hh, mm, ss: INTEGER);*

   Testet die TimeSpanToSeconds Funktion.

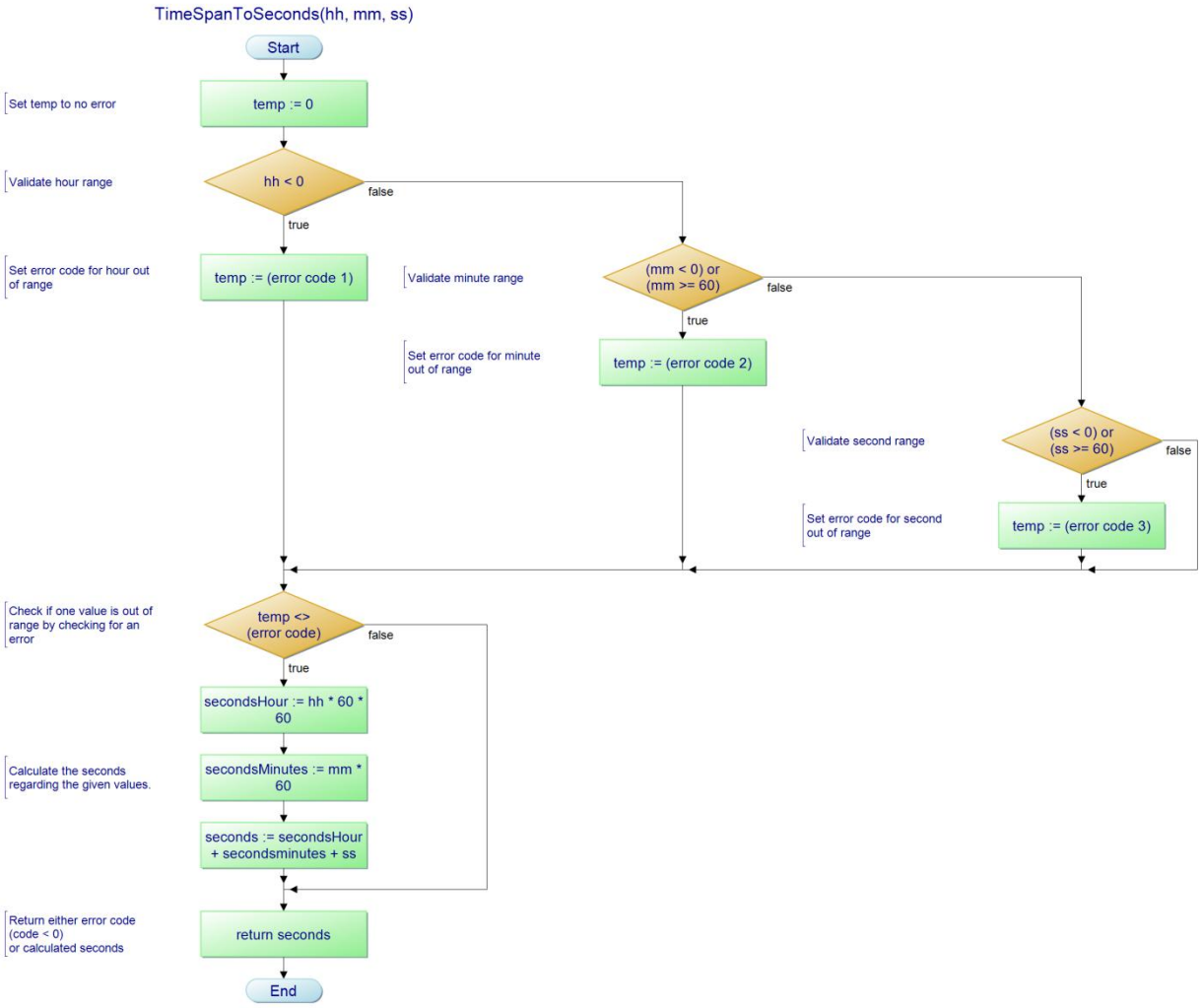
5. *TestSecondsToTimeSpan(sec: LONGINT);*

   Testet die SecondsToTimespan Funktion.

3.1.1 TimeSpanToSeconds

Folgend ist die Lösungsidee des Algorithmus angeführt, welcher eine Zeitdarstellung in seine Sekundendarstellung konvertiert. Als Datentyp für den Rückgabewert wurde LONGING gewählt, da INTEGER nur einen Wertebereich von -32768 <= x <= 32767 zur Verfügung stellt, was für die Konvertierung zu Sekunden ein zu kleiner Wertebereich ist. Als Parameter für die Eingangswerte wurde INTEGER gewählt, da dieser für die Werte enthalten in der Zeitdarstellung völlig ausreichend ist. Sollte ein ungültiger Zeitwert detektiert werden, so wird ein entsprechender Error Code gesetzt, welcher in weiterer Folge in der Testprozedur ausgewertet wird.

Hier kann eine Funktion verwendet werden, weil es hier nur einen Rückgabewert gibt.

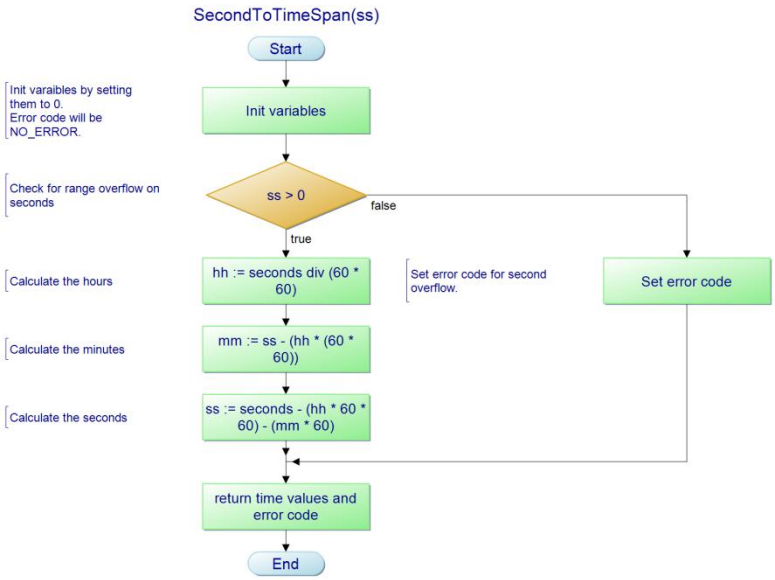

TimeSpanToSeconds(hh, mm, ss)
Start
Set temp to no error
temp := 0
Validate hour range
hh < 0
false
true
Set error code for hour out of range
temp := (error code 1)
Validate minute range
(mm < 0) or (mm >= 60)
false
true
Set error code for minute out of range
temp := (error code 2)
Validate second range
(ss < 0) or (ss >= 60)
false
true
Set error code for second out of range
temp := (error code 3)
Check if one value is out of range by checking for an error
temp <> (error code)
false
true
secondsHour := hh * 60 * 60
Calculate the seconds regarding the given values.
secondsMinutes := mm * 60
seconds := secondsHour + secondsminutes + ss
Return either error code (code < 0) or calculated seconds
return seconds
End

3.1.2 SecondsToTimeSpan

Folgend ist die Lösungsidee für den Algorithmus SecondToTimeSpan angeführt. Als Datentyp für den Eingangswert wurde LONGINT gewählt, da der Wertebereich für die Sekunden größer sein muss als der für die Zeitdarstellung. Da 93599 in der Sekundendarstellung die Zeitdarstellung 25:59:59 hat und schon bei weitem den Wertebereich eines INTEGER überschreitet. Sollte hier ein ungültiger Wert bei den Sekunden detektiert werden, so wird auch hier ein Error Code gesetzt, welcher in späterer Folge in der Testprozedur ausgewertet wird. Hierbei ist anzumerken, das jeweils immer der erste Fehler detektiert und der entsprechende Fehlercode gesetzt wird. Weitere Fehler werden nicht mehr detektiert. Dies konnte realisiert werden, in dem man ein Array für das Speichern der Fehlercodes verwendet und kein Integer. So könnte man in späterer Folge alle Fehler ausgeben. In diesem Fall wurde darauf verzichtet.

Hierbei kann keine Funktion sondern nur eine Prozedur zum Einsatz kommen, da hier mehrere Rückgabewerte gebraucht werden (Stunden, Minuten, Sekunden) und die Prozedur dies erlaubt, wobei eine Funktion nur einen Rückgabewert zulässt.

Folgend ist das Ablaufdiagram des Algorithmus SecondToTimeSpamn angeführt.



3.2 Source

Folgend ist der Source des Programms TimeConverter angeführt.

```
{
  This program is used to convert a second value to its time representation
  and to convert a time representation 'hh:mm:ss' to its second representation.

  Main function and procedure:
  ----------------------------
  1. SecondsToTimeSpan(sec: LONGINT; VAR hh, mm, ss, code: INTEGER);
     Converts the seconds to its time representation

  2. TimeSpanToSeconds(hh, mm, ss: INTEGER): LONGINT;
     Converts a time representation to its second representation

  Contained util function, procedures:
  -------------------------------------
  1. ToString(value: LONGINT): STRING;
     Converts a integer to its string representation

  2. ValidateRange(value, bottom, top: INTEGER): BOOLEAN;
     Validates if the given value is within the defined borders

  3. GetErrorMessage(code: INTEGER): STRING;
     Resoles the proper error message regarding the given code

  4. TestTimeSpanToSeconds(hh, mm, ss: INTEGER);
     Procedure to test the TimeSpanToSeconds algorithm.

  5. TestSecondsToTimeSpan(sec: LONGINT);
     Procedure to test the SecondsToTimespan algorithm
}
PROGRAM TimeConverter;

  {
    Constans available in whole program, because they are used in several functions and
procedures.
    These constans do define the maximum time part borders
    and the error codes if the time value or second value could not be converted
  }
  CONST
    hoursPerDay: INTEGER = 24;
    minutesPerHour: INTEGER = 60;
    secondsPerMinute: INTEGER = 60;
    invalidHours: INTEGER = -1;
    invalidMinutes: INTEGER = -2;
    invalidSeconds: INTEGER = -3;

  {
    Converts a integer to a string representation.
    @param value:
              the integer to be converted to a string
    @paaram the string representation of the given integer value
  }
  FUNCTION ToString(value: LONGINT): STRING;
  VAR
    s: STRING;
  BEGIN
   Str(value, s);
   ToString := s
  END;

  {
    Function which validates the range of the given value.

    @param value:
            the value which range shall be validated
    @param bottom:
            the bottom border of the givan 'value'
    @param top:
            the top border of the given 'value'
    @return true if the value is within the defined borders, false otherwise or if bottom >
top
  }
```

```
  FUNCTION ValidateRange(value, bottom, top: INTEGER): BOOLEAN;
  BEGIN
    ValidateRange := false;

    IF (bottom <= top) AND (value >= bottom) AND (value <= top) THEN
      ValidateRange := true
  END;

  {
    Function to resolve the error message regarding the error code.
    @param code:
            the code containg the error message
    @return the proper error message, 'undefined error occured: [UNKNOWN ERROR CODE]' if an
unknown error occured

  }
  FUNCTION GetErrorMessage(code: INTEGER): STRING;
  BEGIN
    GetErrorMessage := 'undefined error occured: ' + ToString(code);

    IF code = invalidSeconds THEN
      GetErrorMessage := 'Given seconds are out of range !!! error-code: ' + ToString(code)
    ELSE IF code = invalidMinutes THEN
      GetErrorMessage := 'Given minutes are out of range !!! error-code: '  + ToString(code)
    ELSE IF code = invalidHours THEN
      GetErrorMessage := 'Given hours are out of range !!! error-code: '  + ToString(code)
  END;


  {
    Procedure which converts the given seconds to hour, minute and second values.
    @param seconds:
              the seconds to convert to time representation
    @return
        hour: the calculated hours
        min : the calculated minutes
        sec : the calculated seconds which are the rest
  }
  PROCEDURE SecondsToTimeSpan(sec: LONGINT; VAR hh, mm, ss, code: INTEGER);
  BEGIN
    code := 0;
    hh := 0;
    mm := 0;
    ss := 0;

    {
      Calculate the time representation of the given 'seconds' when 'seconds > 0'.
      If 'seconds <= 0' then no calculation is necessary.
      Set error code invalidSeconds if 'sec < 0' to indicate that invalid second value has
been detected
    }
    IF sec > 0 THEN BEGIN
      hh := sec DIV (secondsPerMinute * minutesPerHour);
      mm := (sec - (hh * (secondsPerMinute * minutesPerHour))) DIV minutesPerHour;
      ss := (sec - (hh * (secondsPerMinute * minutesPerHour)) - (mm * minutesPerHour));
    END ELSE IF sec < 0 THEN
      code := invalidSeconds
  END;

  {
    Function which calculates the second representation of the given time 'hh:mm:ss'.
    @param hh:
            the hours to be converted to seconds
    @param mm:
            the minutes to be cconverted to seconds
    @param ss:
            the seconds to be added to the converted seconds of 'hour' and 'min'
    @return the seconds representation of the given time.
            If there is an invalid time value given e.g.: 123:60:60 then regarding to the
first detected
            timer part a constan will be returned to indicate where the error occured.
    @see ValidateRange(value, range: INTEGER): BOOLEAN;
    @see CONST at program head
         invalidHours: INTEGER = -1;
         invalidMinutes: INTEGER = -2;
         invalidSeconds: INTEGER = -3;
```

```
  }
  FUNCTION TimeSpanToSeconds(hh, mm, ss: INTEGER): LONGINT;
  VAR
    temp: LONGINT;
  BEGIN
    temp := 0;

    IF hh < 0 THEN
      temp := invalidHours
    ELSE IF NOT ValidateRange(mm, 0, (minutesPerHour - 1)) THEN
      temp := invalidMinutes
    ELSE IF NOT ValidateRange(ss, 0, (secondsPerMinute - 1)) THEN
      temp := invalidSeconds;

    IF temp = 0 THEN
        temp := (hh * minutesPerHour * secondsPerMinute) + (mm * secondsPerMinute) + ss;

    TimeSpanToSeconds := temp;
  END;

  {
    Procedure to test the Function TimeSpanToSeconds.
    Prints the result or the error to the console.
    @param hh:
            the hours of the time
    @param mm:
            the minutes of the time
    @param ss:
            the seconds of the time
  }
  PROCEDURE TestTimeSpanToSeconds(hh, mm, ss: INTEGER);
  VAR
    seconds: LONGINT;
  BEGIN
    seconds := TimeSpanToSeconds(hh,mm,ss);
    WriteLn('Time: ', ToString(hh), ':', ToString(mm), ':', Tostring(ss), ' / ');

    IF seconds < 0 THEN
      WriteLn(GetErrorMessage(seconds))
    ELSE
      WriteLn('Seconds: ', seconds);

    WriteLn('--------------------------------------------------------------------------');
  END;

  {
    Tests the procedure SecondsToTimeSpan.
    Prints the result or the error to the console.
    @param sec:
             the seconds to convert
  }
  PROCEDURE TestSecondsToTimeSpan(sec: LONGINT);
    VAR
      hh, mm ,ss, code: INTEGER;
  BEGIN
    WriteLn('Seconds: ', sec);
    SecondsToTimeSpan(sec, hh, mm, ss, code);

    IF code < 0 THEN
      WriteLn(GetErrorMessage(code))
    ELSE
      WriteLn('Time: ', ToString(hh), ':', ToString(mm), ':', Tostring(ss), ' / ');

    WriteLn('--------------------------------------------------------------------------');
  END;

VAR
  input: LONGINT;
  hours, minutes, seconds, code: INTEGER;
BEGIN
  WriteLn('##########################################################################');
  WriteLn('#                    Tests for TimeSpanToSeconds                         #');
  WriteLn('##########################################################################');
  WriteLn();
  TestTimeSpanToSeconds(-1,0,0);
  TestTimeSpanToSeconds(0,-1,0);
```

```
  TestTimeSpanToSeconds(0,0,-1);
  TestTimeSpanToSeconds(0,60,0);
  TestTimeSpanToSeconds(0,0,60);
  TestTimeSpanToSeconds(0,0,0);
  TestTimeSpanToSeconds(1,0,0);
  TestTimeSpanToSeconds(0,1,0);
  TestTimeSpanToSeconds(0,0,1);
  TestTimeSpanToSeconds(25,0,0);
  TestTimeSpanToSeconds(0,59,0);
  TestTimeSpanToSeconds(0,0,59);
  TestTimeSpanToSeconds(25,59,59);

  WriteLn();
  WriteLn('#########################################################################');
  WriteLn('#                   Tests for SecondsToTimeSpan                         #');
  WriteLn('#########################################################################');
  WriteLn();
  TestSecondsToTimeSpan(-1);
  TestSecondsToTimeSpan(0);
  TestSecondsToTimeSpan(59);
  TestSecondsToTimeSpan(60);
  TestSecondsToTimeSpan(3559);
  TestSecondsToTimeSpan(3600);
END.
```

3.3 Tests

Folgend sind die Tests der beiden Algorithmen angeführt.

3.3.1 TimeSpanToSeconds

Folgend sind die Tests angeführt, welche den Algorithmus für die Konvertierung einer Zeitdarstellung in seine Sekundendarstellung testen.

Testszenarien:

1. Grenzüberschreitungen
2. Gültige Werte

Grenzüberschreitungen:

Folgend sind die Tests angeführt, die testen wie sich der Algorithmus verhält, wenn ungültige Werte für entweder Stunden, Minuten oder Sekunden auftreten.

```
Time: -1:0:0 /
Given hours are out of range !!! error-code: -1
-------------------------------------------------------------------------
Time: 0:-1:0 /
Given minutes are out of range !!! error-code: -2
-------------------------------------------------------------------------
Time: 0:0:-1 /
Given seconds are out of range !!! error-code: -3
-------------------------------------------------------------------------
Time: 0:60:0 /
Given minutes are out of range !!! error-code: -2
-------------------------------------------------------------------------
Time: 0:0:60 /
Given seconds are out of range !!! error-code: -3
-------------------------------------------------------------------------
Time: 0:60:60 /
Given minutes are out of range !!! error-code: -2
-------------------------------------------------------------------------
```

In diesem Fall wird ein Error Code gesetzt, der in weiterer Folge in der Testprozedur abgefangen und ausgewertet wird. Es wird jeweils nur der erste Fehler detektiert egal ob weitere ungültige Fehlereingaben vorhanden sind.

Gültige Werte:

Folgend sind die Tests angeführt, welche das erwartende Verhalten des Algorithmus testen, wenn nur gültige Werte angegeben werden.

```
Time: 0:0:0 /
Seconds: 0
Time: 1:0:0 /
Seconds: 3600
Time: 0:1:0 /
Seconds: 60
Time: 0:0:1 /
Seconds: 1
Time: 25:0:0 /
Seconds: 90000
Time: 0:59:0 /
Seconds: 3540
Time: 0:0:59 /
Seconds: 59
Time: 25:59:59 /
Seconds: 93599
```

3.3.2 SecondsToTimeSpan

Folgend sind die Tests angeführt, welche den Algorithmus für die Konvertierung einer Sekundendarstellung in seine Zeitdarstellung testen.

Testszenarien:

1. Grenzüberschreitung
2. Gültige Werte

Grenzüberschreitung:

Folgend ist der Test angeführt, der testet wie sich der Algorithmus verhält, wenn ein ungültiger Wert für die Sekunden angegeben wird.

```
Seconds: -1
Given seconds are out of range !!! error-code: -3
```

Hierbei wird der entsprechende Fehlercode gesetzt und in der Testprozedur ausgewertet.

Gültige Werte:

Folgend sind die Tests angeführt, welche das zu erwartende Verhalten des Algorithmus testen, wenn nur gültige Werte für die Sekunden angegeben werden.

```
Seconds: 0
Time: 0:0:0 /
Seconds: 59
Time: 0:0:59 /
Seconds: 60
Time: 0:1:0 /
Seconds: 3559
Time: 0:59:19 /
Seconds: 3600
Time: 1:0:0 /
```