

1. Uhrzeitkonvertierung

1.1 Lösungsidee

Die Lösung unterteilt sich in die Funktion `validateTimeSpan`, `TimespanToSeconds`, `SecondsToTimeSpan` und `TimeDifference`.

1.1.1 Konstanten

Es werden folgende Konstanten angelegt:

- `FactorMinutesToSeconds` mit dem Wert 60 (Umrechnungsfaktor Minuten zu Sekunden)
- `FactorHoursToSeconds` mit dem Wert 3600 (Umrechnungsfaktor Stunden zu Sekunden)
- `MinimumHours` mit dem Wert 0 (Wieviele Stunden gibt es mindestens)
- `InvalidValue` mit dem Wert -1 (-1 kann nur im Fehlerfall auftreten)

1.1.2 Typen

Es werden mehrere Typen angelegt.

- `MinutesAndHoursSpan` mit 0..59 (Range von 0 bis 59, da es minimal 0 Minuten und Sekunden geben kann und maximal 59 Minuten und Sekunden im Format HH:MM:SS)
- `TimeSpan` als Verbund. Dieser Verbund repräsentiert die Zeit und hat folgende Daten:
 - `hours` als `LONGWORD`. `LONGWORD` wurde gewählt, da die Stunden ziemlich groß werden können.
 - `minutes` als `BYTE`
 - `seconds` als `BYTE`

1.1.3 Funktion `validateTimeSpan`

Diese Funktion dient zur Validierung eines Datensatzes für den Typ `TimeSpan`. Sie liefert einen `BOOLEAN` zurück. Es wird überprüft, ob der Wert für `Seconds` und `Minutes` zwischen 0 und 59 liegen. Es wird `TRUE` für eine gültige Validierung zurückgeliefert, ansonsten `FALSE`. `Hours` wird übrigens nicht validiert, da z.B. im VS Code Flags beim kompilieren gesetzt werden, dass das Programm nicht kompiliert. Wenn man es mit dem `fpc` Kommando direkt kompiliert, dann ist es zwar möglich, wir bekommen bei einem Unterlauf aber den Maximalwert von einem `LONGWORD`, der Wert kann nie negativ werden.

1.1.4 Funktion TimespanToSeconds

In diese Funktion wird ein Parameter mit dem Datentyp TimeSpan übergeben. Diese Funktion liefert einen Wert mit dem Datentyp INT64 zurück. Es wurde hier INT64 gewählt, da dies der Datentyp ist, welcher am Größten ist und auch negative Werte speichern kann. LONGWORD, welches bei hours verwendet wurde, hat einen Maximalwert von 4294967295, wenn dies als Stundenanzahl angenommen wird und versucht wird, in Sekunden umzuwandeln, kann der INT64 diesen Problemlos speichern.

Zuerst wird mit der Funktion validateTimeSpan überprüft, ob timeSpan gültig ist, wenn diese gültig ist, werden die Sekunden berechnet und die Funktion liefert das Ergebnis zurück. Die Formel lautet:

`hours * FactorHoursToSeconds + minutes * FactorMinutesToSeconds + seconds`

Anderenfalls, wenn die Validierung FALSE zurückliefert, liefert die Funktion den Wert -1 zurück.

1.1.5 Funktion SecondsToTimeSpan

In diese Funktion wird ein Parameter mit dem Datentyp INT64, welche unsere Sekunden darstellt. Diese Funktion liefert ein Datenobjekt vom Typ TimeSpan zurück.

Es wird als Zwischenspeicher eine Variable outputTimeSpan mit dem Typ TimeSpan angelegt. Der Variable remainingSeconds werden die Sekunden zugewiesen, die übergeben wurden. Zuerst berechnen wir die Stunden, diese ergeben sich aus einer Ganzzahligen Division (Rest wird verworfen) von den Sekunden durch 3600. Der Rest ist der neue Wert für remainingSeconds. Als nächstes werden die Minuten berechnet, ebenfalls mit einer Ganzzahligen Division, diesmal aber mit remainingSeconds durch 60. Der Rest wird den Sekunden des outputTimeSpan zugewiesen, die Ergebnisse für die Stunden und Minuten ebenfalls.

1.1.6 Funktion TimeDifference

In diese Funktion werden zwei Parameter mit dem Datentyp TimeSpan übergeben, welche Zwei Zeitspannen darstellen, von welchen wir den Zeitunterschied auswerten wollen. Der Rückgabewert der Funktion ist vom Typ INT64. Es wird angenommen, dass der Rückgabewert für eine gültige Eingabe nur positiv sein kann, da niemand im echten Leben sagt, es ist -13 Sekunden Zeitunterschied. Negative Werte werden zur Fehlerbehandlung hergenommen.

Zuerst werden die Zeitspannen in Sekunden umgewandelt, dann wird überprüft, ob beide der Zeitspannen in Sekunden nicht den Wert -1 haben (Wert für eine Ungültige Eingabe). Wenn dies zutrifft, wird überprüft, ob die erste Zeitspanne größer ist als die zweite. Falls dies der Fall ist, wird die zweite Zeitspanne von der ersten subtrahiert, anderenfalls die erste von der zweiten. Das Ergebnis liefert die Funktion dann zurück.

Falls der Wert -1 auftritt bei der Bedingung im Absatz darüber, dann gibt die Funktion den Wert -1 zurück.

1.2 Code

```
PROGRAM TimeConversion;

CONST
    FactorMinutesToSeconds = 60;
    FactorHoursToSeconds = 3600;
    MinimumHours = 0;
    InvalidValue = -1;

TYPE
    MinutesAndHoursSpan = 0..59;
    TimeSpan = RECORD
        hours : LONGWORD;
        minutes: BYTE;
        seconds: BYTE;
    END;

FUNCTION ValidateTimeSpan (timeSpan : TimeSpan): BOOLEAN;
VAR
    validInput : BOOLEAN;

BEGIN

    validInput := TRUE;

    IF (timeSpan.seconds < Low(MinutesAndHoursSpan))
    OR (timeSpan.seconds > High(MinutesAndHoursSpan))
    THEN validInput := FALSE;

    IF (timeSpan.minutes < Low(MinutesAndHoursSpan))
    OR (timeSpan.minutes > High(MinutesAndHoursSpan))
    THEN validInput := FALSE;

    IF timeSpan.hours < MinimumHours
    THEN validInput := FALSE;

    ValidateTimeSpan := validInput;

END;
```

```

FUNCTION TimeSpanToSeconds (timeSpan : TimeSpan) : INT64;
BEGIN
    IF ValidateTimeSpan(timeSpan) THEN
        TimeSpanToSeconds := timeSpan.hours * FactorHoursToSeconds +
                               TimeSpan.minutes * FactorMinutesToSeconds +
                               TimeSpan.seconds
    ELSE
        TimeSpanToSeconds := InvalidValue;
    END;
END;

FUNCTION SecondsToTimeSpan (seconds : INT64) : TimeSpan;
VAR
    outputTimeSpan : TimeSpan;
    remainingSeconds : INT64;
BEGIN
    remainingSeconds := seconds;

    outputTimeSpan.hours := remainingSeconds DIV FactorHoursToSeconds;
    remainingSeconds := remainingSeconds MOD FactorHoursToSeconds;

    outputTimeSpan.minutes := remainingSeconds DIV FactorMinutesToSeconds;
    remainingSeconds := remainingSeconds MOD FactorMinutesToSeconds;

    outputTimeSpan.seconds := remainingSeconds;

    SecondsToTimeSpan := outputTimeSpan;
END;

FUNCTION TimeDifference (timeSpan1, timeSpan2 : TimeSpan) : INT64;
VAR
    totalSecondsForTimeSpan1 : INT64;
    totalSecondsForTimeSpan2 : INT64;
BEGIN
    totalSecondsForTimeSpan1 := TimeSpanToSeconds(timeSpan1);
    totalSecondsForTimeSpan2 := TimeSpanToSeconds(timeSpan2);

    IF (totalSecondsForTimeSpan1 <> InvalidValue)
    AND (totalSecondsForTimeSpan2 <> InvalidValue) THEN BEGIN
        IF totalSecondsForTimeSpan1 >= totalSecondsForTimeSpan2 THEN
            TimeDifference := totalSecondsForTimeSpan1 - totalSecondsForTimeSpan2
        ELSE
            TimeDifference := totalSecondsForTimeSpan2 - totalSecondsForTimeSpan1;
        END
    ELSE
        TimeDifference := InvalidValue;
    END;
END;

```

```

PROCEDURE OutputTestCase (original, backConverted : TimeSpan; seconds :
INT64);
BEGIN
    WriteLn('Hours: Original: ', original.hours, ' : BackConverted: ',
backConverted.hours);
    WriteLn('Minutes: Original: ', original.minutes, ' : BackConverted: ',
backConverted.minutes);
    WriteLn('Seconds: Original: ', original.seconds, ' : BackConverted: ',
backConverted.seconds);
    WriteLn('Total Seconds: ', seconds);
END;

```

```

VAR

```

```

    test1 : TimeSpan;
    test2 : TimeSpan;
    totalSeconds1 : INT64;
    totalSeconds2 : INT64;
    test1BackConverted : TimeSpan;
    test2BackConverted : TimeSpan;
    totalSeconds : INT64;

```

```

BEGIN

```

```

    // Code zum Aufrufen der Prozeduren
    WriteLn('1. Timespan:');
    ReadTimeSpan(test1);
    totalSeconds1 := TimeSpanToSeconds(test1);
    test1BackConverted := SecondsToTimeSpan(totalSeconds1);

    WriteLn('2. Timespan:');
    ReadTimeSpan(test2);
    totalSeconds2 := TimeSpanToSeconds(test2);
    test2BackConverted := SecondsToTimeSpan(totalSeconds2);

    totalSeconds := TimeDifference(test1, test2);
    WriteLn;
    WriteLn('Timedifference: ', totalSeconds, ' Seconds');
    WriteLn('1. Timespan:');
    OutputTestCase(test1, test1BackConverted, totalSeconds1);
    WriteLn;
    WriteLn('2. Timespan:');
    OutputTestCase(test2, test2BackConverted, totalSeconds2);

```

```

END.

```

1.3 Testfälle

```
// Beispiele aus der Angabe
```

```
1. Timespan:
```

```
Hours: 0
```

```
Minutes: 0
```

```
Seconds: 0
```

```
2. Timespan:
```

```
Hours: 4
```

```
Minutes: 13
```

```
Seconds: 23
```

```
Timedifference: 15203 Seconds
```

```
1. Timespan:
```

```
Hours: Original: 0 : BackConverted: 0
```

```
Minutes: Original: 0 : BackConverted: 0
```

```
Seconds: Original: 0 : BackConverted: 0
```

```
Total Seconds: 0
```

```
2. Timespan:
```

```
Hours: Original: 4 : BackConverted: 4
```

```
Minutes: Original: 13 : BackConverted: 13
```

```
Seconds: Original: 23 : BackConverted: 23
```

```
Total Seconds: 15203
```

```
// Beweis, dass die Timespans auch getauscht werden können
```

```
Timedifference: 15203 Seconds
```

```
1. Timespan:
```

```
Hours: Original: 4 : BackConverted: 4
```

```
Minutes: Original: 13 : BackConverted: 13
```

```
Seconds: Original: 23 : BackConverted: 23
```

```
Total Seconds: 15203
```

```
2. Timespan:
```

```
Hours: Original: 0 : BackConverted: 0
```

```
Minutes: Original: 0 : BackConverted: 0
```

```
Seconds: Original: 0 : BackConverted: 0
```

```
Total Seconds: 0
```

```
// Ungültige Eingaben
```

```
1. Timespan:
```

```
Hours: 1
```

```
Minutes: 62
```

```
Seconds: 17
```

```
2. Timespan:
```

```
Hours: 1
```

```
Minutes: 17
```

```
Seconds: 62
```

```

Timedifference: -1 Seconds
1. Timespan:
Hours: Original: 1 : BackConverted: 0
Minutes: Original: 62 : BackConverted: 0
Seconds: Original: 17 : BackConverted: 255
Total Seconds: -1

2. Timespan:
Hours: Original: 1 : BackConverted: 0
Minutes: Original: 17 : BackConverted: 0
Seconds: Original: 62 : BackConverted: 255
Total Seconds: -1

// Ungültige Stunden, das Programm wirft auch einen Error, wenn ein Negativer
Wert für Minuten oder Sekunden gesetzt wird
1. Timespan:
Hours: -1
Runtime error 106 at $004017FA
    $004017FA
    $00401AC0
    $004086E7

// Ungültige Eingabe (weil character)
1. Timespan:
Hours: A
Runtime error 106 at $004017FA
    $004017FA
    $00401AC0
    $004086E7

// Stunden werden zu groß
// Fehler, weil überlauf
1. Timespan:
Hours: 4294967295
Minutes: 59
Seconds: 59

2. Timespan:
Hours: 4294967296
Runtime error 106 at $004017FA
    $004017FA
    $00401B2C
    $004086E7

// Bei der Konvertierung von Sekunden zu TimeSpan kann es ebenfalls dazu
kommen, dass die Stunden überlaufen, da Sekunden vom Typ INT64 ist und Hours
von Typ LONGWORD

```

2. Erzeugen von Balkendiagrammen

2.1 Lösungsidee

Die Lösung wird unterteilt in eine Prozedur `BarChart`, welche als Hauptprogramm agiert und welche die Ausgabe des Charts delegiert. Die Funktion `LineIsRequired` gibt einen Boolean zurück, ob eine Zeile ausgegeben werden soll. Die Prozedur `PrintChartLine` gibt eine einzelne Zeile des Charts aus und die Prozedur `PrintChartTail` gibt die X-Achse des Charts aus. Die Prozedur `ReadData` wird zum Einlesen der Daten verwendet.

2.1.1 Konstanten

Es werden mehrere Konstanten angelegt, welche für die verbesserte Lesbarkeit des Codes verwendet werden.

- `PipeCharacter`: `'|'`
- `EmptyString`: `""` (In dem String ist wirklich nichts drinnen)
- `TrailBegin`: `'+'` (Für das Ende des Charts)
- `SpaceCharacter`: `' '` (Ein Leerzeichen)

2.1.2 Typen

Es werden mehrere Typen angelegt.

- `BarValue`: 1..10 (Wertebereich von mindestens 1 und maximal 10, welche ein Balken zugewiesen werden kann.)
- `BarRange`: 1..40 (Wertebereich von 1 bis 40, dieser Type wird für das Array als Anzahl der Feldelemente verwendet, in welche die Werte der Balken gespeichert werden.)
- `BarChartData`: Array mit 40 Feldelementen (zum Ansprechen wird die `BarRange` verwendet).

2.1.3 Prozedur `BarChart`

Diese Prozedur agiert als Hauptalgorithmus und delegiert die Ausgabe des Charts.

1. In die Prozedur werden die Variablen `ch`, `n` und `data` übergeben. Diese Namen ergeben sich aus der Angabe.
`ch` ist ein Character, mit den die Balken gezeichnet werden. `n` beschreibt die Anzahl der eingegebenen Balken. `data` hat den Typ `BarChartData`. Es handelt sich dabei um ein Array, welches 40 Feldelemente hat, diese Elemente sind vom Typ `BarValue` (Wertebereich von 1 bis 10).
2. Zuerst wird die Variable `yIndexWidth` auf 1 gesetzt. Diese Variable ist dafür verantwortlich, wie viele Characters die Legende auf der Y-Achse sein soll. Dies betrifft nur die Ziffern von 1 bis 10, nicht die Pipe-Characters. Dies hat nur rein grafische Hintergründe.
3. Es wird überprüft, ob die Variable `n` (Anzahl der Balken) kleiner als 10 ist. Wenn dies der Fall ist, wird die Variable `barItemWidth` auf den Wert 2 gesetzt, ansonsten auf 3. Diese ist dafür verantwortlich, wie breit (also wie viele Character) ein einzelner Datensatz auf der X-Achse ist. Dies hat nur rein grafische Hintergründe.
4. Variable `i` wird auf die höchste Ordinalzahl von `BarValue` gesetzt (also 10)
5. Überprüfe, ob `i` kleiner gleich die minimale Ordinalzahl von `BarValue` hat (also 1 oder kleiner), wenn dies nicht der Fall ist, gehe zu Schritt 9
6. Überprüft mit der Funktion `LineIsRequired`, ob die aktuelle Zeile benötigt wird oder nicht. Dies hat den Hintergrund, dass leere Zeilen übersprungen werden. Man könnte es auch mit einem Array realisieren und dieses auswerten, aber bei mit der derzeitigen Idee braucht man

dieses Array nicht.

Das Ergebnis von `LineIsRequired` wird in die Variable `lineRequired` gespeichert. Es wird dann überprüft, ob die Variable `i` den Höchsten Ordinalwert von `BarValue` hat (10). Wenn dies zutrifft und `lineRequired` auch `true` ist, wird die Variable `yIndexWidth` auf 2 gesetzt.

7. Es wird nun überprüft, ob die Variable `lineRequired` `true` ist, wenn dies der Fall ist, wird die Prozedur `PrintChartLine` aufgerufen, welche eine einzelne Zeile des Charts ausgibt.
8. Dekrementiere die Variable `i` um 1 und gehe zu Schritt 5.
9. Die Prozedur `PrintChartTail` wird aufgerufen, welche den Rest des Charts ausgibt. Im Endeffekt handelt es sich hierbei um die X-Achse des Charts sowie die Achsenbeschriftung.

2.1.4 Funktion `LineIsRequired`

Die Funktion `LineIsRequired` gibt einen Boolean zurück, ob eine Zeile ausgegeben werden soll. Der Grund warum `LineIsRequired` als Funktion angelegt wird ist, da es alle Bedingungen für eine Funktion erfüllt. Es gibt keine Seiteneffekte, keine Übergangsparameter und nur einen Ausgangsparameter.

1. In die Funktion werden die Parameter `lineNumber`, `amountOfData` und `data` übergeben. `lineNumber` ist die aktuelle Zeilennummer, `amountOfData` repräsentiert `n`, also die Anzahl der Balken und `data` ist das Array mit den Balkenwerten.
2. Die Variable `isRequired` wird auf `FALSE` gesetzt
3. Iteration, welche über `data` von der kleinsten Ordinalzahl von `data` bis zu `amountOfData` das Array durchgeht. Es wird in der Schleife überprüft, ob der Balken mit dem aktuellen index größer oder gleich der `lineNumber` ist. Wenn dies der Fall ist, wird die Variable `isRequired` auf `TRUE` gesetzt.
4. Das Ergebnis der Funktion ist die Variable `isRequired` und liefert den Wert zurück.

2.1.5 Prozedur `PrintChartLine`

Diese Prozedur gibt eine einzelne Zeile des BarCharts aus.

1. In die Prozedur werden die Parameter `lineNumber`, `amountOfData`, `data`, `character`, `yIndexWidth` und `barItemWidth` übergeben.
2. Es wird die `lineNumber` mit einem Pipe-Zeichen ausgegeben. Die `lineNumber` hat die Breite (Anzahl von Characters), welche in `yIndexWidth` festgelegt wurden.
3. Es wird über die geringste Ordinalzahl von `data` drüberiteriert, bis der index den Wert von `amountOfData` erreicht hat. Es wird dabei über `data` iteriert. Wenn der Wert des aktuellen Balkens größer oder gleich der `lineNumber` ist, wird der übergebene `character` ausgegeben, ansonsten werden Leerzeichen ausgegeben.
4. Am Ende wird eine neue Zeile ausgegeben, dass nicht alles in einer Wurst im Terminal steht.

2.1.6 Prozedur `PrintChartTail`

Diese Prozedur gibt die X-Achse des Charts aus.

1. In die Prozedur werden die Parameter `amountOfData`, `data`, `yIndexWidth` und `barItemWidth` übergeben.
2. Es wird `amountOfData` mit der `barItemWidth` multipliziert und in die Variable `columnsToPrint` gespeichert. Diese Variable gibt an, wie viele Minus-Characters ausgegeben werden müssen für die Y-Achse.
3. Es werden Leerzeichen ausgegeben in der Länge von `yIndexWidth` und die Konstante `TailBegin`, welche ein `+` ist. Somit ist das Plus direkt unter den Pipe-Characters.

4. Iteration beginnt bei 1 und iteriert bis columnsToPrint, dabei wird bei jedem durchlauf ein Minus-Character ausgegeben. Danach wird eine neue Zeile begonnen.
5. Es werden Leerzeichen ausgegeben in der Länge von yIndexWidth + 1.
6. Iteration beginnt bei 1 und iteriert bis amountOfData, dabei wird unter dem Trennstrich der Index der einzelnen Balken ausgegeben.

2.1.7 Prozedur ReadData

Diese Prozedur liest die Werte für ch, n und data ein. Wenn n größer als 40 ist oder kleiner als 1, dann wird amountOfData auf 0 gesetzt. Wenn für einen einzelnen Datenwert der Wert von 1 unterschritten oder von 10 überschritten wird, wird amountOfData auch auf 0 gesetzt. Wenn amountOfData 0 ist, wissen wir, dass die Eingabe ungültig ist.

2.1.8 Hauptprogramm

Im Hauptprogramm wird lediglich zuerst die Prozedur ReadData aufgerufen, mit welchen die Variablen ch (character), n (amountOfData) und data befüllt werden. Wenn wir als amountOfData 0 zurückbekommen, dann wissen wir, dass wir eine ungültige Eingabe haben. Hier wird dann „Invalid Input“ ausgegeben. Wenn amountOfData aber größer als 0 ist, wird die Prozedur BarChart aufgerufen.

2.2 Code

```
PROGRAM CreateBarChart;

CONST
    PipeCharacter    = '|';
    EmptyString      = '';
    TrailBegin       = '+';
    MinusCharacter    = '-';
    SpaceCharacter    = ' ';

TYPE
    BarValue         = 1..10;
    BarRange          = 1..40;
    BarChartData      = ARRAY[BarRange] OF BarValue;

// amountOfData represents n
FUNCTION LineIsRequired (lineNumber, amountOfData : INTEGER; data :
BarChartData) : BOOLEAN;
    VAR
        isRequired   : BOOLEAN;
        i             : INTEGER;
BEGIN
    isRequired := FALSE;

    FOR i := Low(data) TO amountOfData DO BEGIN
        IF data[i] >= lineNumber THEN isRequired := TRUE;
    END;

    LineIsRequired := isRequired;
END;

PROCEDURE PrintChartLine (lineNumber, amountOfData : INTEGER; data :
BarChartData; character : CHAR; yIndexWidth, barItemWidth : INTEGER);
    VAR
        i : INTEGER;
BEGIN
    Write(lineNumber : yIndexWidth, PipeCharacter);

    FOR i := Low(data) TO amountOfData DO BEGIN
        IF data[i] >= lineNumber THEN Write(character : barItemWidth)
        ELSE Write(EmptyString : barItemWidth);
    END;

    WriteLn;
END;
```

```

PROCEDURE PrintChartTail (amountOfData : INTEGER; data : BarChartData;
yIndexWidth, barItemWidth : INTEGER);
    VAR
        columnsToPrint, i : INTEGER;
BEGIN
    columnsToPrint := amountOfData * barItemWidth;
    Write(EmptyString : yIndexWidth, TrailBegin);

    FOR i := Low(data) TO columnsToPrint DO Write(MinusCharacter);
    WriteLn;

    Write(EmptyString : yIndexWidth, SpaceCharacter);
    FOR i := Low(data) TO amountOfData DO Write(i : barItemWidth);
END;

PROCEDURE BarChart (ch: CHAR; n : INTEGER; data : BarChartData);
    VAR
        i : INTEGER;
        lineRequired : BOOLEAN;
        yIndexWidth : INTEGER;
        barItemWidth : INTEGER;
BEGIN
    yIndexWidth := 1;

    IF n < 10 THEN barItemWidth := 2
    ELSE barItemWidth := 3;

    FOR i:= High(BarValue) DOWNTO Low(BarValue) DO BEGIN
        lineRequired := LineIsRequired(i, n, data);
        IF (lineRequired = TRUE) AND (i = High(BarValue)) THEN yIndexWidth := 2;
        IF lineRequired THEN PrintChartLine(i, n, data, ch, yIndexWidth,
barItemWidth) ;
    END;
    PrintChartTail(n, data, yIndexWidth, barItemWidth);
END;

```

```

PROCEDURE ReadData (VAR character : CHAR; VAR amountOfData : INTEGER; VAR data
: BarChartData);
    VAR
        i : INTEGER;
        inputData : INTEGER;
BEGIN
    Write('ch: ');
    Read(character);

    Write('n: ');
    ReadLn(amountOfData);

    IF (Low(BarRange) <= amountOfData) AND (amountOfData <= High(BarRange)) THEN
BEGIN
    i := Low(data);

    WHILE (i <= amountOfData) DO BEGIN
        Write('data item ', i, ':');
        ReadLn(inputData);
        IF (inputData < Low(BarValue)) OR (High(BarValue) < inputData)
            THEN amountOfData := 0
            ELSE
                data[i] := inputData;
                Inc(i);
            END;
        END
    ELSE amountOfData := 0;

END;

VAR
    data, test : BarChartData;
    character : CHAR;
    amountOfData : INTEGER;
BEGIN
    ReadData(character, amountOfData, data);

    IF amountOfData = 0 THEN WriteLn('Invalid input!')
    ELSE BarChart(character, amountOfData, data);
END.

```

2.3 Testfälle

```
// Test mit character 1 mit 2 Werten (5 und 10)
```

```
ch: 1
```

```
n: 2
```

```
data item 1: 5
```

```
data item 2: 10
```

```
10| 1
```

```
9| 1
```

```
8| 1
```

```
7| 1
```

```
6| 1
```

```
5| 1 1
```

```
4| 1 1
```

```
3| 1 1
```

```
2| 1 1
```

```
1| 1 1
```

```
+----
```

```
1 2
```

```
-----  
// Test mit character E mit 10 Werten (1 bis 10)
```

```
ch: E
```

```
n: 10
```

```
data item 1: 1
```

```
data item 2: 2
```

```
data item 3: 3
```

```
data item 4: 4
```

```
data item 5: 5
```

```
data item 6: 6
```

```
data item 7: 7
```

```
data item 8: 8
```

```
data item 9: 9
```

```
data item 10: 10
```

```
10| E
```

```
9| E E
```

```
8| E E E
```

```
7| E E E E
```

```
6| E E E E E
```

```
5| E E E E E E
```

```
4| E E E E E E E
```

```
3| E E E E E E E E
```

```
2| E E E E E E E E E
```

```
1| E E E E E E E E E E
```

```
+-----
```

```
1 2 3 4 5 6 7 8 9 10
```

```
-----  
// Eingabe eines ungültigen characters  
// Hier kommt ein Runtime Error raus
```

```
ch: XE  
n: Runtime error 106 at $004019FB  
    $004019FB  
    $00401B3D  
    $004084D7
```

```
-----  
// Eingabe einer ungültigen Anzahl von Daten  
// Es wird Invalid Input ausgegeben, es werden keine Daten eingelesen
```

```
ch: X  
n: 41
```

```
Invalid input!
```

```
-----  
// Eingabe einer ungültigen Anzahl von Daten  
// Es wird Invalid Input ausgegeben, es werden keine Daten eingelesen
```

```
ch: F  
n: -1
```

```
Invalid input!
```

```
-----  
// Eingabe eines ungültigen Data Items  
// Die Eingabe wird sofort abgebrochen und es wird Invalid input ausgegeben
```

```
ch: E  
n: 5  
data item 1: 10  
data item 2: 20
```

```
Invalid input!
```

```
-----  
// Eingabe eines ungültigen Data Items  
// Die Eingabe wird sofort abgebrochen und es wird Invalid input ausgegeben
```

```
ch: F  
n: 4  
data item 1: -100
```

```
Invalid input!
```

```
-----  
// Test mit character H mit 40 Werten
```

```
ch: H
```

```
n: 40
```

```
data item 1: 1
```

```
data item 2: 2
```

```
data item 3: 3
```

```
data item 4: 4
```

```
data item 5: 5
```

```
data item 6: 6
```

```
data item 7: 7
```

```
data item 8: 8
```

```
data item 9: 9
```

```
data item 10: 10
```

```
data item 11: 1
```

```
data item 12: 2
```

```
data item 13: 3
```

```
data item 14: 4
```

```
data item 15: 5
```

```
data item 16: 6
```

```
data item 17: 7
```

```
data item 18: 8
```

```
data item 19: 9
```

```
data item 20: 10
```

```
data item 21: 1
```

```
data item 22: 2
```

```
data item 23: 3
```

```
data item 24: 4
```

```
data item 25: 5
```

```
data item 26: 6
```

```
data item 27: 7
```

```
data item 28: 8
```

```
data item 29: 9
```

```
data item 30: 10
```

```
data item 31: 1
```

```
data item 32: 2
```

```
data item 33: 3
```

```
data item 34: 4
```

```
data item 35: 5
```

```
data item 36: 6
```

```
data item 37: 7
```

```
data item 38: 8
```

```
data item 39: 9
```

```
data item 40: 10
```


[illegible]

3. Matrizenmultiplikation

3.1 Lösungsidee

Die Lösung unterteilt sich in die Funktionen TurnMatrix, GetScalarProduct, MultiplyMatrix und die Prozeduren ReadMatrix und PrintMatrix.

3.1.1 Typen

Es wurden folgende Typen angelegt:

- MatrixLength mit der Range 1 bis 3, diese wird für die Länge eines MatrixItems, sprich einer Zeile hergenommen, aber auch für die Anzahl der Zeilen, weil es sich um eine 3x3 Matrix lt. Angabe handelt
- MatrixItem, welches ein Array mit MatrixLength Feldelementen vom Typ Real sind
- Matrix, welches ein Array mit MatrixLength mit Feldelementen von MatrixItem ist.

3.1.2 ReadMatrix

Es handelt sich hierbei um eine Prozedur, da es hier Seiteneffekte gibt. Dabei wird Zeile für Zeile eingelesen. Für jede Zeile wird Item für Item eingelesen. Diese Prozedur hat einen Rückgabeparameter outputMatrix, welcher die Eingabewerte zugewiesen werden.

3.1.3 PrintMatrix

Es handelt sich hierbei um eine Prozedur, da es hier Seiteneffekte gibt. Es wird in die Prozedur eine Matrix übergeben und dabei wird Zeile für Zeile ausgegeben. Für jede Zeile wird Item für Item ausgegeben. Somit wird die Matrix dargestellt.

3.1.4 MultiplyMatrix

In der Angabe steht zwar, dass eine Prozedur zu implementieren ist, aber es wurde sich bewusst für eine Funktion entschieden, da es 0 bis n Übergabeparameter gibt, keine Seiteneffekte und 1 Ausgangsparameter. In die Funktion werden zwei Matrizen übergeben. Zuerst wird die zweite Matrize gedreht und der Variable turnedMatrixB zugewiesen. Dann wird über jede Matrixzeile der outputMatrix drüberiteriert, wo über jede Spalte iteriert wird. Es wird dann der Wert für das Item der Zeile über die Funktion GetScalarProduct berechnet. Zur Berechnung wird die matrixA Zeile und die turnedMatrixB Spaltennummer übergeben. Als Ergebnis wird die Matrix outputMatrix übergeben.

3.1.5 TurnMatrix

Diese Funktion dreht Spalten und Zeilen einer Matrix, damit die Verarbeitung einfacher gestaltet werden kann. Dabei sind die Spalten die neuen Zeilen und die Zeilen die neuen Spalten

3.1.6 GetScalarProduct

In dieser Funktion wird das Skalarprodukt von zwei MatrixItems gebildet und zurückgegeben.

3.2 Code

```
PROGRAM MatrixCalculator;

TYPE
  MatrixLength = 1..3;
  MatrixItem = ARRAY[MatrixLength] OF REAL;
  Matrix = ARRAY[MatrixLength] OF MatrixItem;
```

```

FUNCTION TurnMatrix (inputMatrix : Matrix) : Matrix;
VAR
    outputMatrix : Matrix;
    i, j : INTEGER;
BEGIN
    FOR i := Low(MatrixLength) TO High(MatrixLength) DO BEGIN
        FOR j := Low(MatrixLength) TO High(MatrixLength) DO BEGIN
            outputMatrix[j, i] := inputMatrix[i, j];
        END;
    END;
    TurnMatrix := outputMatrix;
END;

FUNCTION GetScalarProduct (matrixRowA, matrixRowB : MatrixItem) : REAL;
VAR
    sum : REAL;
    column : INTEGER;
BEGIN
    sum := 0;
    FOR column := Low(MatrixLength) TO High(MatrixLength) DO
        sum := sum + matrixRowA[column] * matrixRowB[column];
    END;
    GetScalarProduct := sum;
END;

FUNCTION MultiplyMatrix (matrixA, matrixB : Matrix) : Matrix;
VAR
    turnedMatrixB : Matrix;
    outputMatrix : Matrix;
    row, column : INTEGER;
    sum : REAL;
BEGIN
    turnedMatrixB := TurnMatrix(matrixB);
    FOR row := Low(MatrixLength) TO High(MatrixLength) DO BEGIN
        FOR column := Low(MatrixLength) TO High(MatrixLength) DO BEGIN
            outputMatrix[row, column] := GetScalarProduct(matrixA[row],
                                                            turnedMatrixB[column]);
        END;
    END;
    MultiplyMatrix := outputMatrix;
END;

PROCEDURE ReadMatrix (VAR outputMatrix : Matrix);
VAR
    row, column : INTEGER;
BEGIN

```

```

FOR row := Low(MatrixLength) TO High(MatrixLength) DO BEGIN
    WriteLn('Enter values for row ', row, ':');
    FOR column := Low(MatrixLength) TO High(MatrixLength) DO BEGIN
        Write('Value for column ', column, ': ');
        ReadLn(outputMatrix[row, column]);
    END;
    WriteLn;
END;
END;

PROCEDURE PrintMatrix (matrix : Matrix);
VAR
    i, j : INTEGER;
BEGIN
    FOR i := Low(MatrixLength) TO High(MatrixLength) DO BEGIN
        FOR j := Low(MatrixLength) TO High(MatrixLength) DO BEGIN
            Write(matrix[i, j] :8:2, ' ');
        END;
        WriteLn;
    END;
END;

VAR
    a, b, c : Matrix;
BEGIN
    WriteLn('Enter values for Matrix A:');
    ReadMatrix(a);
    WriteLn;
    WriteLn('Enter values for Matrix B:');
    c := MultiplyMatrix(a, b);
    PrintMatrix(c);
END.

```

3.3 Testfälle

```
Values for Matrix A:
Enter values for row 1:
Value for column 1: 1
Value for column 2: 2
Value for column 3: 3
```

```
Enter values for row 2:
Value for column 1: 4
Value for column 2: 5
Value for column 3: 6
```

```
Enter values for row 3:
Value for column 1: 7
Value for column 2: 8
Value for column 3: 9
```

```
Values for Matrix B:
Enter values for row 1:
Value for column 1: 1
Value for column 2: 2
Value for column 3: 3
```

```
Enter values for row 2:
Value for column 1: 4
Value for column 2: 5
Value for column 3: 6
```

```
Enter values for row 3:
Value for column 1: 7
Value for column 2: 8
Value for column 3: 9
```

30.00	36.00	42.00
66.00	81.00	96.00
102.00	126.00	150.00

```
// Negative Values for a single matrix
```

```
Values for Matrix A:
Enter values for row 1:
Value for column 1: 1
Value for column 2: 2
Value for column 3: 3
```

```
Enter values for row 2:
Value for column 1: 4
Value for column 2: 5
Value for column 3: 6
```

```
Enter values for row 3:
Value for column 1: 7
Value for column 2: 8
Value for column 3: 9
```

```
Values for Matrix B:
Enter values for row 1:
Value for column 1: -1
Value for column 2: -2
Value for column 3: -3

Enter values for row 2:
Value for column 1: -4
Value for column 2: -5
Value for column 3: -6

Enter values for row 3:
Value for column 1: -7
Value for column 2: -8
Value for column 3: -9

-30.00  -36.00  -42.00
-66.00  -81.00  -96.00
-102.00 -126.00 -150.00
```

```
// Only Negative values
```

```
Values for Matrix A:
Enter values for row 1:
Value for column 1: -1
Value for column 2: -2
Value for column 3: -3

Enter values for row 2:
Value for column 1: -4
Value for column 2: -5
Value for column 3: -6

Enter values for row 3:
Value for column 1: -7
Value for column 2: -8
Value for column 3: -9
```

```
Values for Matrix B:
Enter values for row 1:
Value for column 1: -1
Value for column 2: -2
Value for column 3: -3

Enter values for row 2:
Value for column 1: -4
Value for column 2: -5
Value for column 3: -6

Enter values for row 3:
Value for column 1: -7
Value for column 2: -8
Value for column 3: -9
```

30.00	36.00	42.00
66.00	81.00	96.00
102.00	126.00	150.00

// Eingabe von einem String für einen Wert

Values for Matrix A:

Enter values for row 1:

Value for column 1: a

Runtime error 106 at \$004018B6

\$004018B6

\$00401A32

\$0040AC97

Values for Matrix A:

Enter values for row 1:

Value for column 1: -1-1

Runtime error 106 at \$004018B6

\$004018B6

\$00401A32

\$0040AC97

// Overflow eines Wertes

// Overflows und Underflows können passieren, das Programm arbeitet damit

// Und es wurde keine Input Validation gefordert.

Values for Matrix A:

Enter values for row 1:

Value for column 1: 50000

Value for column 2: 10

Value for column 3: 11

Enter values for row 2:

Value for column 1: -1

Value for column 2: -5

Value for column 3: 1

Enter values for row 3:

Value for column 1: 4

Value for column 2: 1

Value for column 3: 5

Values for Matrix B:

Enter values for row 1:

Value for column 1: 1

Value for column 2: 2

Value for column 3: 3

Enter values for row 2:

Value for column 1: 4

Value for column 2: 56

Value for column 3: 7

Enter values for row 3:

Value for column 1: 81

Value for column 2: 9

Value for column 3: 2

50931.00	100659.00	150092.00
60.00	-273.00	-36.00
413.00	109.00	29.00