PRG1x & ADE1x Üb. zu Einf. i d. Programmierung und Element. Alg. u. Datenstrukt. **WS 22/23, Übung 1**

Abgabetermin: Sa, 09.10.2022

☒ **Gr. 1,** DI (FH) G. Horn-Völlenkle, MSc **Name Andreas Neubauer** **Aufwand in h 6**

☐ **Gr. 2,** S. Schöberl, MSc

Punkte _____ **Tutor*in / Übungsleiter*in** ____ / ____

1. Algorithmus: Einnahmen/Ausgaben-Rechnung (4 + 4 Punkte)

Entwickeln Sie einen Algorithmus, der eine Folge ganzer Zahlen einliest und die Summe der positiven Zahlen (*Einnahmen*) sowie die Summe der negativen Zahlen (*Ausgaben*) bildet. Wird der Wert 0 gelesen, soll der Algorithmus das Einlesen beenden und die beiden Summen sowie eine Gesamtsumme (= Einnahmen – Ausgaben) ausgeben. Stellen Sie den Algorithmus mittels (a) *Pseudocode* und (b) *Ablaufdiagramm* dar.

Beispiele:

1. Eingabe: `10 20 -30 40 -50 60 70 80 -90 0`
   Ausgabe:
   ```
   Einnahmen:   280
   Ausgaben:   -170
   Gesamt:      110
   ```
2. Eingabe: `0`
   Ausgabe:
   ```
   Einnahmen:     0
   Ausgaben:      0
   Gesamt:        0
   ```

2. Pascal-Programm: Einnahmen/Ausgaben-Rechnung (6 Punkte)

Implementieren Sie den Algorithmus aus Aufgabe 1 in Pascal. Testen Sie Ihr Programm ausführlich und geben Sie aussagekräftige Testfälle ab.

3. Pascal-Programm: tägliche Zeiterfassung (6 Punkte)

Entwickeln Sie ein Pascal-Programm, welches die tägliche Arbeitszeit in Form ganzzahliger Werte für Stunden und Minuten einliest und aus ggf. geleisteten Überstunden den Anspruch auf Zeitausgleich ermittelt und ausgibt. Bezüglich der täglichen Arbeitszeit gelten (nur hier!) folgende Regeln

- Die ersten acht Stunden sind Normalarbeitszeit, es fallen keine Überstunden an.
- Die 9. und 10. Stunde kann als Zeitausgleich in Anspruch genommen werden.
- Die 11. und 12. Stunde kann mit Faktor 1,5 als Zeitausgleich in Anspruch genommen werden.
- Täglich dürfen höchstens 12 Stunden gearbeitet werden.

Beispiele:

1. Eingabe: `8 30`
   Ausgabe: `Anspruch auf Zeitausgleich: 0.50 Stunden`
2. Eingabe: `11 00`
   Ausgabe: `Anspruch auf Zeitausgleich: 3.50 Stunden`
3. Eingabe: `13 00`
   Ausgabe: `Taegliche Hoechstarbeitszeit ueberschritten`

4. Darstellungsformen (4 Punkte)

Diskutieren Sie die Vor- und Nachteile der Darstellungsformen für Algorithmen, die Sie in Aufgabe 1 und 2 verwendet haben.

Hinweise:

1. Lesen Sie die organisatorischen Hinweise im Moodle-Kurs.
2. Geben Sie für alle Ihre Lösungen immer eine „Lösungsidee“ an.
3. Dokumentieren und kommentieren Sie Ihre Algorithmen.
4. Bei Programmen: Geben Sie immer auch Testfälle ab, an denen man erkennen kann, dass Ihr Programm funktioniert, und dass es auch in Fehlersituation entsprechend reagiert.

Inhalt

1 Algorithmus: Einnahmen/Ausgaben-Rechnung

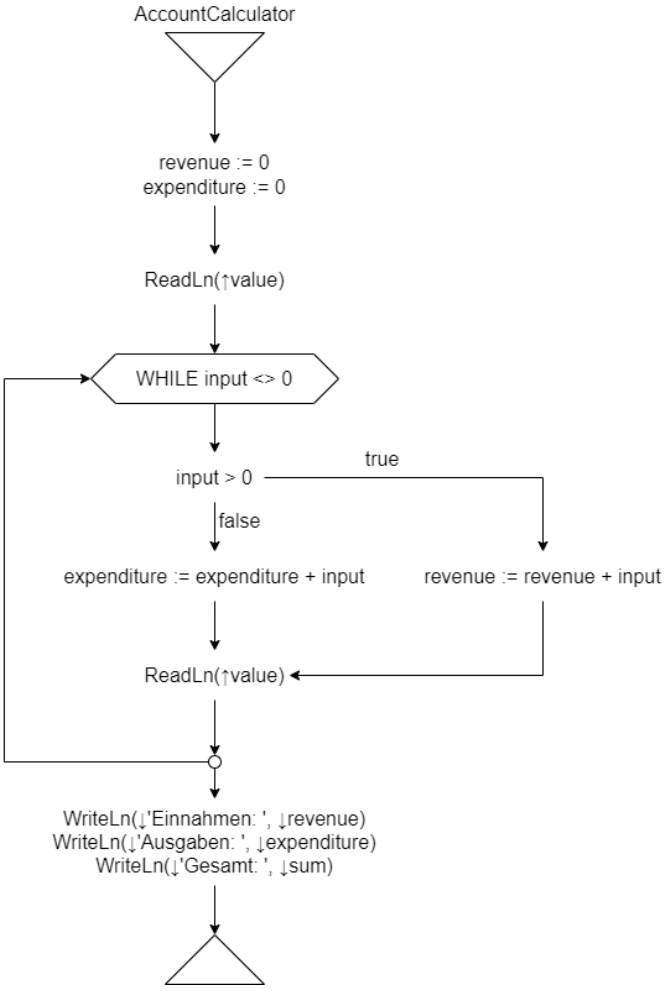
1.1 Lösungsidee

1. Erstellung der Variablen revenue und expenditure mit dem Wert 0.
2. Auslesen der Eingabe und Zuweisung des Wertes zur Variable input.
3. Wenn der Wert 0 ist, gehe zu Schritt 6
4. Wenn input größer als 0 ist, addiere den input zu revenue, sonst addiere den Wert zu expenditure
5. Gehe zu Schritt 2
6. Berechne sum mit revenue + expenditure
7. Ausgeben der Werte revenue, expenditure und sum

1.2 Pseudocode

```
AccountCalculator ()

BEGIN

  revenue     := 0
  expenditure := 0

  ReadLn(↑input)
  WHILE input <> 0
    IF input > 0 THEN revenue     := revenue      + input
    ELSE              expenditure := expenditure  + input
    ReadLn(↑input)
  END -- while

  sum := revenue + expenditure
  WriteLn(↓'Einnahmen: ', ↓revenue)
  WriteLn(↓'Ausgaben: ', ↓expenditure)
  WriteLn(↓'Gesamt: ', ↓sum)

END -- BEGIN
```

1.2 Ablaufdiagramm



2 Algorithmus: Einnahmen/Ausgaben-Rechnung

2.1 Code

```
PROGRAM AccountCalculator;

// Creating variables
// input        => User input
// revenue      => Positive input (more money)
// expenditure  => Negative input (less money)
// sum          => SUm of revenue + expenditure
VAR
  input: INTEGER;
  revenue : INTEGER;
  expenditure : INTEGER;
  sum: INTEGER;

// Statements
BEGIN
  // Set revenue and expenditure to 0
  revenue := 0;
  expenditure := 0;

  // Read the user input
  ReadLn(input);

  // Run the loop until the user input is 0
  // If the input is greater than 0, sum the input to the revenue
  // Else sum the input to the expenditure
  // At the last step read the next user input
  WHILE input <> 0 DO BEGIN
    IF input > 0 THEN revenue := revenue + input
    ELSE expenditure := expenditure + input;

    ReadLn(input);
  END;

  // Calculate the sum
  // In this case, you need to add the expenditure to the revenue
  // because the expenditure has a negative value
  sum := revenue + expenditure;

  // Output the data
  WriteLn('Einnahmen: ', revenue);
  WriteLn('Ausgaben: ', expenditure);
  WriteLn('Gesamt: ', sum)
END
.
```

2.2 Testfälle

1. Zufällige Werte

```
10
20
-50
100
40
70
-30
-20
0
Einnahmen: 240
Ausgaben: -100
Gesamt: 140
```

2. Positive Werte

```
10
50
70
20
30
100
0
Einnahmen: 280
Ausgaben: 0
Gesamt: 280
```

3. Negative Werte

```
-10
-200
-30
-60
-25
0
Einnahmen: 0
Ausgaben: -325
Gesamt: -325
```

4. Nur null

```
0
Einnahmen: 0
Ausgaben: 0
Gesamt: 0
```

5. Extrem große Zahlen

```
32767
1
Runtime error 201 at $0040163F
  $0040163F  main,  line 28 of c:/Users/Andre/Desktop/fh/PascalWorkspace/Units/uebung1/AccountCalculator.pas
  $00408E07
```

   Die Eingabe bis 32767 ist möglich, danach stürzt das Programm ab, da ein Integer[1] maximal den Wert 32767 haben muss.

6. Extrem kleine Zahlen

```
-32768
-1
Runtime error 201 at $00401672
  $00401672  main,  line 29 of c:/Users/Andre/Desktop/fh/PascalWorkspace/Units/uebung1/AccountCalculator.pas
  $00408E07
```

   Die Eingabe bis -32768 ist möglich, danach stürzt das Programm ab, da ein Integer[1] mindestens den Wert -32768 haben muss.

Anmerkungen:

[1]) Der Integer hat eine Größe von 2 Bytes (16 Bits). Hier sind insgesamt 65 536 verschiedene Werte möglich. Dieser teilt sich auf einen Wertebereich von -32 768 bis 32 767 auf. 0 ist dabei als positive Zahl zu sehen.

3 Pascal-Programm: tagliche Zeiterfassung

3.1 Lösungsidee

Main:

1. Anlegen der Variablen overtime, hours und minutes
2. Algorithmus zum Einlesen der Benutzereingabe, dieser liefert die Werte von hours und minutes zurück.
3. Algorithmus zum Berechnen der Überstunden, dieser liefert den Wert für overtime zurück.
4. Wenn overtime mehr als 5 ist, wird Taegliche Hoechstarbeitszeit ueberschritten ausgegeben. (5 ergibt sich daraus, da hier die Sollstundenzahl von 8 Überstunden subtrahiert wird. Wenn die Überstunden mehr als 2 ist, wird alles über die 2 Stunden hinaus mit dem Wert 1,5 multipliziert. Die Maximale Stundenanzahl pro Tag beträgt 12 Stunden, wenn wir die Formel anwenden, dann kommen wir auf 5 Stunden.
   Beispiel:
   12 Stunden – 8 Stunden = 4 Stunden Overtime
   4 Stunden Overtime – 2 Stunden = 2 Stunden
   2 Stunden * 1,5 = 3 Stunden
   Overtime = 2 Stunden + 3 Stunden = 5 Stunden
)
   Wenn overtime mehr als 0 ist, wird Anspruch auf Zeitausgleich: <Zahl> ausgegeben
   Wenn overtime weniger als 0 ist, wird Ausstehende Stunden <Zahl> ausgegeben
   Ansonsten wird Kein Anspruch auf Zeitausgleich ausgegeben.

Algorithmus zum Einlesen der Benutzereingabe:

1. In die Prozedur werden die Variablen hours und minutes übergeben. Es werden zusätzlich die Variablen input und validInputFormat angelegt.
2. Der Wert von validInputFormat wird initial auf FALSE festgelegt. In die Prozedur werden hours und minutes übergeben.
3. Wertzuweisung von input durch Benutzereingabe
4. Die Werte für minutes und hours werden auf -1 festgelegt.
5. Algorithmus zur Überprüfung der Benutzereingabe, diese liefert dann die Werte für hours, minutes und validInputFormat zurück.
6. Wenn validInputFormat FALSE ist, dann wiederhole Schritt 3
7. Die Prozedur endet und liefert die Werte für hours und minutes zurück.

Algorithmus zur Überprüfung der Benutzereingabe:

1. In die Prozedur werden die Variablen input, validInputFormat, hours und minutes übergeben. Anlegen der Variablen INPUT_LENGTH und DELIMITER_POSITION
2. Auslesen der Länge von input und Zuweisung zur Variable INPUT_LENGTH
3. Wenn INPUT_LENGTH nicht 4 und nicht 5 ist, dann wird die Prozedur abgebrochen. Die Werte für validInputFormat, hours und minutes werden nicht überschrieben. Dies ist auf das Format H:MM (4 Zeichen) oder HH:MM (5 Zeichen) zurückzuführen.

4. Wenn die INPUT_LENGTH = 4 ist, dann wird DELIMITER_POSITION der Wert 2 zugewiesen, ansonsten wird der Wert 3 zugewiesen.
5. Überprüfe, ob sich der Delimiter Leertaste oder Doppelpunkt an der DELIMITER_POSITION vom input befinden, wenn ein anderer Wert drinnen steht, dann wird die Prozedur abgebrochen. Die Werte für validInputFormat, hours und minutes werden nicht überschrieben
6. Algorithmus zum Parsen der Ziffern. Der Rückgabewert wird in hours gespeichert. Es muss der input übergeben werden, außerdem eine Start- und End-Position. Diese sind in diesem Fall 1 als Start Wert und DELIMITER_POSITION – 1 als End Wert. Damit kann das Format HH, aber auch nur H behandelt werden. Schlussendlich muss auch noch hours übergeben werden, damit dieser Wert überschrieben werden kann.
7. Algorithmus zum Parsen der Ziffern. Der Rückgabewert wird in minutes gespeichert. Es muss der input übergeben werden, außerdem eine Start- und End-Position. Diese sind in diesem Fall DELIMITER_POSITION + 2 als Start Wert und DELIMITER_POSITION + 2 als End Wert. Damit kann das Format MM behandelt werden. Schlussendlich muss auch noch minutes übergeben werden, damit dieser Wert überschrieben werden kann.
8. Überprüfung, ob hours oder minutes nicht überschrieben wurde. Wenn dies der Fall ist, dann wird die Prozedur abgebrochen. Der Wert für validInputFormat wird nicht überschrieben.
9. Überprüfung, ob minutes größer als 59 ist. Wenn dies der Fall ist, dann wird die Prozedur abgebrochen. Der Wert für validInputFormat wird nicht überschrieben.
10. Setze die Variable validInputFormat auf TRUE.
11. Die Prozedur endet und liefert die Werte für validInputFormat , hours und minutes zurück.

Algorithmus zum Parsen der Ziffern:

1. In die Prozedur werden die Variablen input, START_INDEX, STOP_INDEX und result übergeben. Die Variable input ist die Benutzereingabe, der START_INDEX ist, wo begonnen werden soll zum Auswerten und der STOP_INDEX, wo die Auswertung beendet werden soll. result wird dann zurückgegeben, dies ist dann das Ergebnis.
   Es werden zusätzlich die Variablen i und number angelegt. Der Variable i wird der Wert von START_INDEX zugewiesen.
2. Weise der Variable number den Wert von Ord(input[i]) - Ord(‘0‘). Hier wird der Ascii Wert von input[i] minus der Ascii Wert von 0 subtrahiert, damit eine sinnvolle Zahl rauskommt.
3. Wenn die Variable number kleiner als 0 ist, dann beende die Prozedur. Der Wert für result bleibt unverändert.
4. Wenn die Variable number größer als 9 ist, dann beende die Prozedur. Der Wert für result bleibt unverändert.
5. Wenn i den gleichen Wert wie STOP_INDEX hat, addiere number zu result. Ansonsten addiere number * 10 zu result.
6. Addiere 1 zu i.
7. Überprüfe, ob i kleiner oder gleich STOP_INDEX ist. Wenn dies der Fall ist, gehe zu Schritt 2. Die Schleife wird aber aufgrund vorhergehender Validierungen maximal 1 Mal wiederholt.
8. Addiere 1 zu result, da der Start-Wert für Result -1 ist.

3.2 Code

```
PROGRAM TimeTracker;


// Procedure in order to validate the single digits of a input
// and assemble them into an integer value
PROCEDURE validateDigits (INPUT : STRING; START_INDEX, STOP_INDEX : INTEGER;
VAR result : INTEGER);

// Variables (i => index for-loop, number => as helper variable)
VAR
  i       : INTEGER;
  number  : INTEGER;

// Statements
BEGIN
  // For loop in order to validate the numbers for the hours and minutes
  FOR i := START_INDEX TO STOP_INDEX DO BEGIN

    // Get the number (based by ascii value)
    number := Ord(input[i]) - Ord('0');

    // Check if the digit is between 0 and 9
    IF number < 0 THEN EXIT;
    IF number > 9 THEN EXIT;

    // Calculate the result integer
    IF i = STOP_INDEX THEN result := result + number
    ELSE result := result + number * 10;
  END;

  // Add 1 to the result, because the value started with -1
  result := result + 1
END;



// Procedure to validate the time
PROCEDURE validateTime (input : STRING; VAR validInputFormat : BOOLEAN; VAR
hours, minutes: INTEGER);

// Define variables for the procedure
// INPUT_LENGTH is the amount of characters of the input and the
// DELIMITER_POSITION is the index of the delimiter
VAR
  INPUT_LENGTH      : INTEGER;
  DELIMITER_POSITION: INTEGER;

// Statements
BEGIN
```

```
  // Get the length of the input and check if the input has a length of 4 or
5,
  // otherwise exit the procedure and repeate the user input
  INPUT_LENGTH := Length(input);
  IF (INPUT_LENGTH <> 4) AND (INPUT_LENGTH <> 5) THEN EXIT;

  // Set the DELIMITER_POSITION, if the length is 4, the delimiter must be at
position 2
  // Otherwise the DELIMITER_POSITION is 3
  IF INPUT_LENGTH = 4 THEN DELIMITER_POSITION := 2
  ELSE DELIMITER_POSITION := 3;

  // Check if the delimiter (space or colon) is at the expected position
  // Otherwise exit the procedure
  IF (input[DELIMITER_POSITION] <> ' ') AND (input[DELIMITER_POSITION] <> ':')
THEN EXIT;

  // Assemble the hours and minutes and check if the values for hours and
minutes are set,
  // otherwise exit this procedure and repeat the user input reading
  validateDigits(input, 1, DELIMITER_POSITION - 1, hours);
  validateDigits(input, DELIMITER_POSITION + 1, DELIMITER_POSITION + 2,
minutes);
  if (hours = -1) OR (minutes = -1) THEN EXIT();
  if (minutes > 59) THEN EXIT();

  // Set the value for validInputFormat to true
  validInputFormat := TRUE
END;



// Procedure to read the input
PROCEDURE readInput (VAR hours, minutes : INTEGER);

// Define variables for the procedure
VAR
  input           : STRING;
  validInputFormat: BOOLEAN;

BEGIN
  // Assign the value false to validInputFormat
  validInputFormat  := FALSE;

  // DO-While loop in order to read the input
  REPEAT
    // Read the user input
    WriteLn('Enter Time:');
    ReadLn(input);
```

```
    // Assign values to the variables
    // hours and minutes will be set -1 in order to check, if the digits has
been
    // set correctly
    hours             := -1;
    minutes           := -1;

    // Call method in order to validate the input
    validateTime(input, validInputFormat, hours, minutes)
  UNTIL validInputFormat;
END;



// Procedure to calculate the overtime
PROCEDURE calculateOvertime (hours, minutes : INTEGER; VAR overtime : REAL);
VAR
  workingTime : REAL;
BEGIN
  workingTime := hours + (minutes / 60);
  overtime := workingTime - 8;

  if overtime > 2 THEN overtime := 2 + (overtime - 2) * 1.5;
END;


// Main

// Variables for minutes, hours and the amount of overtime
VAR
  minutes : INTEGER;
  hours   : INTEGER;
  overtime: REAL;

BEGIN
  // Read the input
  readInput(hours, minutes);

  // Calculate the overtime
  calculateOvertime(hours, minutes, overtime);

  // Output messages
  IF      overtime > 5 THEN WriteLn('Taegliche Hoechstarbeitszeit
ueberschritten')
  ELSE IF overtime > 0 THEN WriteLn('Anspruch auf Zeitausgleich:
'                  , overtime:2:2        , ' Stunden')
  ELSE IF overtime < 0 THEN WriteLn('Ausstehende Stunden:
'                         , (overtime * -1):2:2 , ' Stunden')
  ELSE                      WriteLn('Kein Anspruch auf Zeitausgleich')
```

```
END.
```

3.3 Testfälle

1. Tageshöchstzahl überschritten

```
Enter Time:
13 00
Taegliche Hoechstarbeitszeit ueberschritten
```

2. Tageshöchstzahl deutlich überschritten

```
Enter Time:
50 00
Taegliche Hoechstarbeitszeit ueberschritten
```

3. Mehrere ungültige Angaben mit einer anschließend gültigen
   1. Zu hohe Minutenanzahl (> 59)
   2. Buchstabenmix
   3. Zu lange Werte
   4. Zu kurze Werte
   4. 10 Stunden und 59 Minuten

```
Enter Time:
10 60
Enter Time:
ab 10
Enter Time:
10 ab
Enter Time:
abcdef
Enter Time:
10 000
Enter Time:
1 1
Enter Time:
10 59
59
Anspruch auf Zeitausgleich: 3.48 Stunden
```

4. 8 Stunden und 0 Minuten in den Formaten HH:MM und H:MM

```
Enter Time:
8 00
Kein Anspruch auf Zeitausgleich
```

```
Enter Time:
08 00
Kein Anspruch auf Zeitausgleich
```

5. Weniger als 8 Stunden

```
Enter Time:
06 00
Ausstehende Stunden: 2.00 Stunden
Enter Time:
0 00
Ausstehende Stunden: 8.00 Stunden
```

6. Versuch von negativen Werten

```
Enter Time:
-1 00
Enter Time:
```

4. Darstellungsformen

4.1 Ablaufdiagramm

4.1.1 Vorteile

- Ist sehr übersichtlich und ansprechend für das menschliche Auge.
- Kann einfach in Ablaufsprachen umgesetzt werden.
- Sind einfach zu verstehen

4.1.2 Nachteile

- Erstellung ist viel aufwändiger als Code oder Pseudocode.
- Schwerer umsetzbar in einer Programmiersprache als Pseudocode

4.2 Pseudocode

4.2.1 Vorteile

- Einfach zu schreiben
- Ähnlich wie Programmiercode
- Man muss sich nicht an die Syntax von Programmiersprachen halten und kann z.B. Dinge weglassen
- Sind einfach zu überarbeiten

4.2.2 Nachteile

- Ist nicht typisiert, dadurch braucht man ein bisschen, bis man den Pseudocode versteht.
- Es können Verwirrungen auftreten
- Kann nicht von einer Maschine direkt ausgeführt werden

4.1 Code

4.2.1 Vorteile

- Kann ausgeführt werden
- Es gibt eine vorgegebene Syntax
- Es werden direkt die Features entwickelt.

4.2.2 Nachteile

- Es gibt eine Syntax, an die man sich halten muss
- Abhängig der Programmiersprache ist das Verständnis und die Umsetzung aufwändiger.
- Schwieriger zu überarbeiten