Dokumentation

Übung 1 (Summarizer)

Thomas Herzog

04.10.2013

1. Lösungsidee

1.1 Verbale Prosa

Folgend ist die Lösungsidee der Übung 1 (Algorithmus Summarizer) beschrieben.

Für diesen Algorithmus sollen vier Integer Variablen verwendet werden:

1. *countGreater*

   Variable, die die Summe für die Werteeingaben größer gleich der ersten Werteeingabe hält

2. *countSmaller*

   Variable, die die Summe aller Werteeingaben kleiner der ersten Werteeingabe hält

3. *value*

   Variable, die die aktuelle Werteeingabe hält

4. *first*

   Variable, die die erste Werteeingabe über den Lebenszyklus des Programms hält

Als Datentyp der Variablen soll Integer verwendet werden, da laut Spezifikation nur ganzzahlige Werteeingaben dürfen.

Zu Programmbegin sollen die Variablen *countGreater*, *countSmaller* mit 0 und die Variablen *value* und *first* mit der ersten Werteeingabe initialisiert werden. Es wird keine Typprüfung durchgeführt, daher ist mit Exceptions zu rechnen, wenn ungültige Zahlenwerte, die nicht den Datentyp Integer entsprechen, eingegeben werden. Die Summierung der Werteeingaben soll mit Hilfe einer Abbruchschleife erfolgen, da dieser Schleifentyp den beinhalteten Code nur dann ausführt wenn die Bedingung (*value* <> 0) erfüllt ist. Anders als bei einer Durchlaufschleife, wo zusätzlich eine Prüfung mit Hilfe einer IF-Verzweigung erfolgen müsste, um einen Durchlauf zu unterbinden, wenn als erste Werteeingabe eine 0 erfolgt. Dies muss erfolgen da dieser Schleifentyp mindestens einmal den beinhalteten Code ausführt. Die verwendete Abbruchschleife soll den Code enthalten, der die Summen über die Werteeingaben bildet. Die Entscheidung auf welche der Variablen die aktuelle Werteeingabe aufsummiert werden soll, soll mittels einer IF-Verzweigung erfolgen, die prüft ob die erfolgte Werteeingabe größer gleich der ersten Werteeingabe ist. Je nach Ergebnis der Prüfung soll die entsprechende Summe gebildet werden. Zusätzlich soll die vorherig beschriebene Prüfung der Werteeingaben auch noch in einer IF-Verzweigung eingeschlossen werden, die prüft ob die aktuelle Werteeingabe eine positive Zahl darstellt oder nicht, da laut Spezifikation nur positive ganzzahlige Zahlenwerte aufsummiert werden dürfen und das Programm nur bei einer Werteeingabe von 0 beendet werden darf.

Bevor das Programm beendet wird, sollen die Werte der gebildeten Summen ausgegeben werden.

1.2 Stilisierte Prosa

Im folgenden ist die stilisierte Prosa angeführt, die den Algorithmus der Übung 1 beschreibt.

Algorithmus Summarizer:

Step 1. Initialisierung:

Initialisierung der Variablen vom Typ Integer: *countGreater, countSmaller* mit dem Wert 0.

Initialisierung der Variablen vom Type Integer: *first, value* mit der ersten Werteeingabe.

Step 2. Abbruch ?

Wenn *value* = 0, dann gehe zu Step 4, sonst gehe zu Step 3.

Step 3. Summenbildung:

Wenn *value* > 0 dann:

Wenn *value* >= *first* dann:

Addiere *value* mit *countGreater*.

sonst:

Addiere *value* mit *countSmaller*

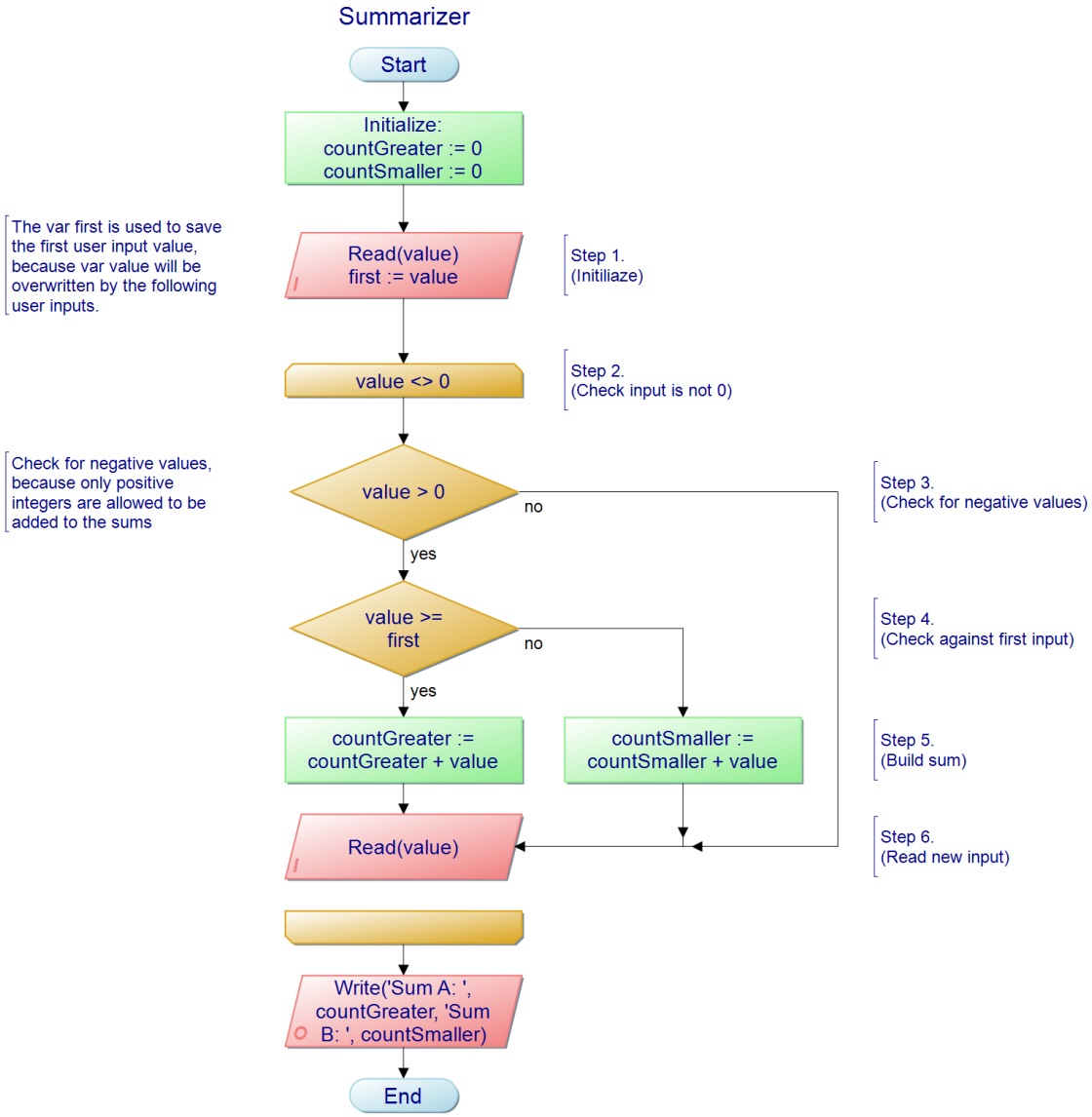
Setze *value* mit der neuen Werteeingabe.

Gehe zu Step 2.

Step 4th Ausgabe:

Gib den Wert der Variablen *countGreater* und *countSmaller* aus.

1.3 Ablaufdiagram



2 Source

```
{
  This program is used to summarize positive Integer input.
  All integer input greater or equal to the first input value will
  be added to the 'countGreater' variable.
  All integer input smaller then the first input value will
  be added to the 'countSmaller' variable.
  Negative values will be excluded and not added to any sum.
  When 0 is entered, the program will exit.
}
PROGRAM Summarizer;

VAR
  countSmaller, countGreater, first, value: INTEGER;

BEGIN
  countSmaller := 0;
  countGreater := 0;
  ReadLn(value);
  {
    The var first is used to save the first read input value over the
    program lifetime. Need var first to be able to compare the following
    read input against the first input.
  }
  first := value;

  WHILE value <> 0 DO BEGIN
    {
      Exclude all negative input, because as specified only
      positive integer greater than 0 are supposed to be added to the a sum
    }
    IF value > 0 THEN BEGIN
      {
        No need to check for the first read input, because when the first
        input was entered then it is guaranteed that the conditions fits
        and that the first input is added to the right sum
      }
      IF value >= first THEN
        countGreater := countGreater + value
      ELSE
        countSmaller := countSmaller + value
    END;
    ReadLn(value)
  END;

  Write('Sum A: ', countGreater, ' / Sum B: ', countSmaller)
END.
```

3 Testfälle

Folgend sind die Testfälle für den Algorithmus 'Summarizer' beschrieben.

3.1 Schreibtischtest

Folgend ist der Schreibtischtest angeführt, der die erste Beispielzahlenfolge der Übungsbeschreibung testet.

Wertefolge: 6, 11, 4, 6, 2, 0

Steps (Siehe auch Ablaufdiagram):

Step 1. Initialize

Step 2. value <> 0

Step 3. value > 0

Step 4. value >= first

Step 5. Build sum

Step 6. Read new input

Step	First	Value	countGreater	countSmaller	value<>0	value>0	value>=first
1	6	6	0	0	-	-	-
2	6	6	0	0	True	-	-
3	6	6	0	0	-	True	-
4	6	6	0	0	-	-	True
5	6	6	6	0	-	-	-
6	6	11	6	0	-	-	-
2	6	11	6	0	True	-	-
3	6	11	6	0	-	True	-
4	6	11	6	0	-	-	True
5	6	11	17	0	-	-	-
6	6	4	17	0	-	-	-
2	6	4	17	0	True	-	-
3	6	4	17	0	-	True	-
4	6	4	17	0	-	-	False
5	6	4	17	4	-	-	-
6	6	6	17	4	-	-	-
2	6	6	17	4	True	-	-
3	6	6	17	4	-	True	-
4	6	6	17	4	-	-	True
5	6	6	23	4	-	-	-
6	6	2	23	4	-	-	-
2	6	2	23	4	True	-	-
3	6	2	23	4	-	True	-
4	6	2	23	4	-	-	False
5	6	2	23	6	-	-	-
6	6	0	23	6	-	-	-
2	6	0	23	6	False	-	-

3.2 0 als erste Eingabe

Dieser Test testet das Verhalten des Algorithmus wenn als erste Werteeingabe eine 0 erfolgt.

```
C:\Users\cchet\Documents\FH-Hagenaberg\Semester 1\ADE\Uebungen\Uebung1>uebung_1
0
Sum A: 0 / Sum B: 0
```

Wenn als erste Werteeingabe 0 erfolgt, dann wird lediglich der Initialwert der verwendeten Variablen für die Summenbildung ausgegeben, welcher natürlich 0 ist, da keine Summenbildung durchgeführt werden konnte.

3.3 Einzelne Eingabe

Dieser Test testet das Verhalten des Algorithmus wenn nur eine Werteeingabe größer 0 erfolgt und danach das Programm mittels der Werteeingabe von 0 beendet wird.

```
C:\Users\cchet\Documents\FH-Hagenaberg\Semester 1\ADE\Uebungen\Uebung1>uebung_1
10
0
Sum A: 10 / Sum B: 0
```

Wenn ein Wert > 0 eingeben wird und danach das Programm mittels der Eingabe von 0 beendet wird, kann lediglich die Summe für die Werteeingaben, die größer gleich der ersten Werteeingabe ist gebildet werden.

3.4 0 < Eingabewerte < 0

Dieser Test testet das Verhalten des Algorithmus wenn neben positiven auch negative Werteeingaben erfolgen.

```
C:\Users\cchet\Documents\FH-Hagenaberg\Semester 1\ADE\Uebungen\Uebung1>uebung_1
10
11
11
12
3
-1
-99
5
0
Sum A: 44 / Sum B: 8
```

Wenn negative Werteeingaben erfolgen, so werden diese nicht in die Summen mit aufgenommen, da, wie spezifiziert, nur ganze positive Zahlen summiert werden dürfen. Ebenso erfolgt auch kein Programmabbruch, da das Programm nur bei der Eingabe einer 0 beendet werden darf. Da nicht genau spezifiziert, habe ich mich dazu entschlossen, negative Werteeingaben lediglich nicht in die Summen mit aufzunehmen und keine Fehlermeldung auszugeben.

3.5 Wertebereichüberschreitung

Dieser Test testet das Verhalten des Algorithmus wenn eine Werteeingabe erfolgt, welche den zur Verfügung gestellten Wertebereich der verwendeten Integer Variablen überschreitet, oder wenn eine der gebildeten Summen diesen Wertebereich überschreitet.

```
C:\Users\cchet\Documents\FH-Hagenaberg\Semester 1\ADE\Uebungen\Uebung1>uebung_1
1
2
32765
Runtime error 201 at $004014A0
  $004014A0
  $00406F81

C:\Users\cchet\Documents\FH-Hagenaberg\Semester 1\ADE\Uebungen\Uebung1>uebung_1
32768
Runtime error 201 at $0040142A
  $0040142A
  $00406F81
```

Der verwendete Integer Datentyp stellt 16 bit zur Verfügung, wobei bei einer Eingabe von 32767<Wert<-32768 oder wenn die Gesamtsumme einer der gebildeten Summen den Wertebereich überschreitet, ein Overflow auftritt und der Range Check eine Exception wirft.

Auszug aus Runtime Errors:

201	Range check error

3.6 Ungültiges Eingabeformat

Dieser Test testet das Verhalten des Algorithmus wenn eine ungültige Werteeingabe erfolgt.

```
C:\Users\cchet\Documents\FH-Hagenaberg\Semester 1\ADE\Uebungen\Uebung1>uebung_1
00a
Runtime error 106 at $00401416
  $00401416
  $00406F81
```

Wenn ein Werteeingabe erfolgt, so wird eine Exception geworfen, die aussagt, dass die Eingabe ein ungültiges Zahlenformat darstellt. Der Algorithmus kann nur Werteeingaben verarbeiten, die dem Datentype Integer entsprechen.

Auszug aus Runtime Errors:

106	Invalid numeric format

3.7 Normaler Programmdurchlauf

Dieser Test testet das Verhalten des Algorithmus bei einer erwarteten Nutzung.

```
C:\Users\cchet\Documents\FH-Hagenaberg\Semester 1\ADE\Uebungen\Uebung1>uebung_1
10
10
9
8
12
0
Sum A: 32 / Sum B: 17
```

Dieser Test beweist, dass sich dieser Algorithmus wie spezifiziert verhält.

4 Diskussion

Im folgenden werden die verschiedenen Darstellungsformen des Algorithmus diskutiert.

4.1 Verbale Beschreibung der Lösungsidee

Bei der verbalen Beschreibung der Lösungsidee ist für mich eine klare Struktur schwerer zu erkennen als bei der stilisierten Prosa oder bei einem Ablaufdiagramm. Ebenso ist es schwer den Algorithmus kurz und prägnant zu beschreiben, wobei darauf zu achten ist, für welche Personen diese Beschreibung verfasst wird. Bei technischen versierten Personen sollte man auf eine umfangreiche Beschreibung verzichten, wobei nicht technisch versierten Personen, Kunden als Beispiel, eher auf eine umfangreichere Beschreibung zu achten wäre. Eine verbale Beschreibung lässt sich nur sequenziell lesen. Damit ist gemeint das es schwer ist zu einen früheren beschriebenen zusammenhängenden Schritt zurückzukehren, da dieser aus der Gesamtbeschreibung erst wieder herausgefunden werden muss.

4.2 Stilisierte Prosa

Bei der stilisierten Prosa ist gegenüber den verbalen Beschreibung schon mehr Struktur zu erkennen, da man sich hier bewusst auf spezifizierte Schritte beschränkt und man davon ausgeht, dass der Leser dieser Form der Beschreibung entweder technisch versiert ist, oder zumindest den Zusammenhang des Algorithmus bzw. der Funktionalität, die dieser abbildet, bewusst ist. Jedoch muss darauf geachtet werden wie die einzelnen Schritte definiert werden. Damit ist gemeint welche Funktionen sollen in einem Schritt zusammengefasst werden. In meiner verfassten stilisierten Prosa ist als Beispiel zwar die Prüfung ob die Abbruchschleife abgearbeitet werden soll in einem eigenen Schritt definiert, obwohl die Prüfungen der IF-Verzweigungen in dem darauffolgenden Schritt zusammengefasst wurden. Einerseits sehe ich die Bedingung der Abbruchschleife als so wichtig an, das diese es wert ist in einem eigenen Schritt zu sein, andererseits würden die IF-Verzweigungen in eigenen Schritten nur die Struktur verkomplizieren und weiters sehe ich die IF-Verzweigungen als Teil der Summierungslogik an, die in einem einzigen Schritt zusammengefasst wurde.

4.3 Ablaufdiagram

Bei dem Ablaufdiagram ist eine besonders klare Struktur ersichtlich, welche in den anderen beiden Darstellungsformen fehlt, wobei auch immer auf die Zielgruppe geachtet werden muss. Diese Darstellungsform lässt auch schon Rückschlusse auf die Implementierung des Algorithmus zu. Hier ist es leicht von einem Schritt zum anderen vorwärts wie rückwärts zu gehen ohne den Überblick zu verlieren. Selbst das gegenüberstellen von einzelnen Schritten ist in der Vorstellung möglich, da diese gut strukturiert in einem Gesamtüberblick des Algorithmus enthalten sind. Bei einem

umfangreicheren Algorithmus sollte man Algorithmus in logische zusammenhängende Teile auftrennen und die einzelnen Teile in einzelnen Ablaufdiagrammen abbilden und diese dann in einem gesamten Ablaufdiagramm, welches den Algorithmus abbildet, anführen. Eine riesiges Ablaufdiagram, welches alle Einzelheiten des Algorithmus enthält wurde an Übersichtlichkeit verlieren.

5 Literaturverzeichnis

Frameworkpascal. (28. 09 2013). Abgerufen am 28. 09 2013 von http://www.frameworkpascal.com/helphtml/run_errors.htm