

☐ Gr. 1, Dr. D. Auer☐ Gr. 2, Dr. G. Kronberger☒ Gr. 3, Dr. H. GruberName Andreas Roither Aufwand in h 10 h

Punkte _____ Kurzzeichen Tutor / Übungsleiter _____ / _____

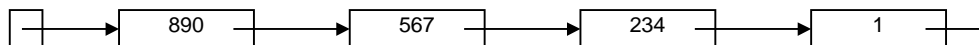
1. Rechnen mit wirklich großen Zahlen zur Basis 1000**(12 Punkte)**

Nachdem Sie in der Übung 4 ja schon mit Zahlen bis zur Basis 36 gerechnet haben, kann Sie nichts mehr erschüttern ;-) Also:

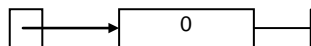
Beliebig große, nicht negative ganze Zahlen (*BigInts*) können realisiert werden, indem man sie (z. B.) zur Basis 1000 darstellt und die einzelnen "Ziffern" (*big digits*, Werte zwischen 0 und 999) in Form einer einfach-verketteten Liste so anordnet, dass die niederwertigste "Ziffer" am Anfang und die höchstwertigste am Ende der Liste steht.

Beispiele:

Die Zahl 1.234.567.890 (Tausenderpunkte hier nur zur Verbesserung der Lesbarkeit) wird durch folgende Liste mit vier Knoten dargestellt:



Die Zahl 0 wird durch die Liste mit einem 0-Knoten dargestellt:



Das Vorzeichen der Zahl kann durch einen Vorzeichenknoten (mit dem Wert +1 oder -1) am Anfang der Liste dargestellt werden. Realisieren Sie mit den unten angegebenen Datentypen *BigInts* und implementieren Sie die Operationen zum Addieren und Multiplizieren.

TYPE

NodePtr = ^Node;**Node** = RECORD**next**: NodePtr;**val**: INTEGER; (*one digit of BigInt: 0 <= val <= 999*)

END; (*RECORD*)

BigIntPtr = NodePtr;FUNCTION **Sum** (a, b: BigIntPtr): BigIntPtr; (*compute sum = a + b*)FUNCTION **Product**(a, b: BigIntPtr): BigIntPtr; (*compute product = a * b*)

Hinweis: Um Sie von „mühsamer Kleinarbeit“ zu entlasten, stehen für das Einlesen und das Ausgeben von *BigInts* folgende zwei Prozeduren (zusammen mit einigen Hilfsprozeduren und -funktionen) in einem Rahmenprogramm (*BigInts.pas*) fix und fertig zur Verfügung:

PROCEDURE **ReadBigInt** (VAR bi: BigIntPtr);PROCEDURE **WriteBigInt**(bi: BigIntPtr);

2. Christkind auf Rationalisierungsstrip

(2 + 4 + 6 Punkte)

Das Christkind bekommt jedes Jahr viele Wunschzettel von den Kindern, und bisher hat es diese von Hand ausgewertet. Weil auch das Christkind immer älter, die Wünsche aber immer mehr werden, muss es rationalisieren und will wie folgt vorgehen: Die von den Kindern handgeschriebenen Wunschzettel werden eingescannt, mit Schrifterkennungssoftware weiterbearbeitet und alle zusammen in einer Textdatei *Wishes.txt* so abgespeichert, dass in einer Zeile jeweils der Name (*name*) des Kinds (gefolgt von Doppelpunkt und Leerzeichen) und sein Wunsch (*item*) stehen. Die Kindernamen sind eindeutig und dürfen mehrfach vorkommen (für den häufigen Fall, dass ein Kind mehrere Wünsche hat):

Beispiel:

```
...
Christoph: Schlitten
Barbara: Barbie-Puppe
Barbara: Puppenküche
Christoph: Matchboxauto
Barbara: Blockflöte
Susi: Strolchi-Puppe
...
```

Nun wünscht sich das Christkind aber etwas von Ihnen: Ein Pascal-Programm *WLA* (*wish list analyzer*), das nach dem Einlesen der Wünschedatei folgendes ermittelt: eine *Bestellliste* mit den Bestellmengen für die einzelnen Gegenstände, damit es diese schnell und einfach besorgen kann, sowie eine *Zustellliste*, in der für jedes Kind alle Geschenke eingetragen sind, sodass es die Geschenke pünktlich ausliefern kann. Sie beschließen, das Programm *WLA*, mittels einfach-verketteter Listen auf Basis folgender Typdeklarationen zu realisieren:

```
(*for a:*)
WishNodePtr = ^WishNode;
WishNode = RECORD
    next: WishNodePtr;
    name: STRING;
    item: STRING;
END; (*RECORD*)
WishListPtr = WishNodePtr;
```

```
(*for b:*)
OrderNodePtr = ^OrderNode;
OrderNode = RECORD
    next: OrderNodePtr;
    item: STRING;
    n: INTEGER;
END; (*RECORD*)
OrderListPtr = OrderNodePtr;
```

```
(*for c:*)
DelivNodePtr = ^DelivNode;
DelivNode = RECORD
    next: DelivNodePtr;
    name: STRING;
    items: ItemListPtr;
END; (*RECORD*)
DelivListPtr = DelivNodePtr;
```

- Bauen Sie beim Lesen der Wünschedatei die Wunschliste (*wish list*) gemäß obiger Deklarationen (links) so auf, dass die Reihenfolge der Wünsche in der Liste jener in der Datei entspricht.
- Entwickeln Sie eine Funktion *OrderListOf*, die auf Basis einer Wunschliste aus a) eine Bestellliste (*order list*) gemäß obiger Deklaration (Mitte) für das Christkind liefert. In der Bestellliste muss für jeden Gegenstand der Wunschliste (*item*) ein Knoten mit der Häufigkeit (*n*) des Auftretens dieses Gegenstands in der Wunschliste enthalten sein.
- Entwickeln Sie eine weitere Funktion *DeliveryListOf*, die auf Basis einer Wunschliste aus a) eine Zustellliste (*delivery list*) gemäß obiger Deklarationen (rechts) liefert. In der Zustellliste muss für jedes Kind (*name*) ein Knoten mit einer Liste aller Geschenke (*items*) für dieses Kind enthalten sein. Die Datentypen für die Geschenkliste pro Kind (*items*) müssen Sie selbst analog zu allen anderen deklarieren.

Hinweis: Um Ihnen das Abtippen der obigen Deklarationen zu ersparen und ein Rahmenprogramm zur Verfügung zu stellen, welches das Einlesen der Wünschedatei erledigt, haben wir *WLA.pas* und eine einfache Wünschedatei (*Wishes.txt*) in *WLA.zip* für Sie vorbereitet.

Übung 8

Aufgabe 1

Lösungsidee

Es sollen beliebig große ganze Zahlen addiert und multipliziert werden. Dazu wird eine Liste verwendet in der Nodes sind, die einen Integer mit maximal drei Stellen beinhalten. Diese Integer der Nodes zusammen gezählt ergeben die Zahl, wobei die erste Node die untersten drei Stellen der Zahl und die letzte Node die größten drei Stellen der Zahl beinhaltet. Für das Addieren und multiplizieren werden eigene Funktionen erstellt denen BigIntPtr übergeben werden. Damit auch negative Zahlen addiert werden können, wird eine Funktion HigherBigInt verwendet die zurückgibt welche Zahl größer ist bzw. ob beide gleich sind. Je nachdem was dies Funktion zurückliefert wird unterschiedlich gerechnet. Für das allgemeine Addieren werden die Integer der Nodes zusammen gezählt und mit einem overflow Integer zusammengezählt. Ist diese Summe größer als 999 muss bei der nächsten Node + 1 dazu gezählt werden, daher wird overflow auf 1 gesetzt. Bei negativen Zahlen wird überprüft ob die Summe unter 0 ist. Falls dies der Fall ist wird overflow auf -1 gesetzt und bei der nächsten Node subtrahiert.

```

1  (* BigInts:                                     F.Li, 1998-11-22
                                           HDO, 2000-11-13
3  Arithmetic for arbitrary size integers which are
   represented as singly-linked lists.
5  =====*)

7  PROGRAM BigInts;

9  (*$IFDEF WINDOWS, for Borland Pascal only*)
   USES
11     WinCrt,
     Math;
13  (*$ENDIF*)

15  (*$DEFINE SIGNED*)    (*when defined: first digit is sign +1 or -1*)

17  CONST
     base = 1000;        (*base of number system used in all*)
19                          (* calculations, big digits: 0 .. base - 1*)

21  TYPE
     NodePtr = ^Node;
23     Node = RECORD
         next: NodePtr;
25         val: INTEGER;
     END; (*RECORD*)
27     BigIntPtr = NodePtr;

29

31  FUNCTION NewNode(val: INTEGER): NodePtr;
     VAR
         n: NodePtr;
33  BEGIN
     New(n);
35     (*IF n = NIL THEN ... *)
     n^.next := NIL;

```

```

37     n^.val := val;
      NewNode := n;
39 END; (*NewNode*)

41 FUNCTION Zero: BigIntPtr;
BEGIN
43     Zero := NewNode(0);
END; (*Zero*)

45
PROCEDURE Append(VAR bi: BigIntPtr; val: INTEGER);
47     VAR
        n, last: NodePtr;
49 BEGIN
        n := NewNode(val);
51     IF bi = NIL THEN
        bi := n
53     ELSE BEGIN (*l <> NIL*)
        last := bi;
55         WHILE last^.next <> NIL DO BEGIN
            last := last^.next;
57         END; (*WHILE*)
        last^.next := n;
59     END; (*ELSE*)
END; (*Append*)

61
PROCEDURE Prepend(VAR bi: BigIntPtr; val: INTEGER);
63     VAR
        n: NodePtr;
65 BEGIN
        n := NewNode(val);
67     n^.next := bi;
        bi := n;
69 END; (*Prepend*)

71 FUNCTION Sign(bi: BigIntPtr): INTEGER;
BEGIN
73 (*$IFDEF SIGNED*)
        (*assert: bi <> NIL*)
75     Sign := bi^.val; (*results in +1 or -1*)
(*$ELSE*)
77     WriteLn('Error in Sign: no sign node available');
    Halt;
79 (*$ENDIF*)
END; (*Sign*)

81
FUNCTION CopyOfBigInt(bi: BigIntPtr): BigIntPtr;
83     VAR
        n: NodePtr;
85     cBi: BigIntPtr; (*cBi = copy of BigIntPtr*)
BEGIN
87     cBi := NIL;
        n := bi;
89     WHILE n <> NIL DO BEGIN
        Append(cBi, n^.val);
91     n := n^.next;
    END; (*WHILE*)
93     CopyOfBigInt := cBi;
END; (*CopyOfBigInt*)

```

```

95  PROCEDURE InvertBigInt(VAR bi: BigIntPtr);
96      VAR
97          iBi, next: NodePtr; (*iBi = inverted BigIntPtr*)
98  BEGIN
99      IF bi <> NIL THEN BEGIN
100          iBi := bi;
101          bi := bi^.next;
102          iBi^.next := NIL;
103          WHILE bi <> NIL DO BEGIN
104              next := bi^.next;
105              bi^.next := iBi;
106              iBi := bi;
107              bi := next;
108          END; (*WHILE*)
109          bi := iBi;
110      END; (*IF*)
111  END; (*InvertBigInt*)
112
113  PROCEDURE DisposeBigInt(VAR bi: BigIntPtr);
114      VAR
115          next: NodePtr;
116  BEGIN
117      WHILE bi <> NIL DO BEGIN
118          next := bi^.next;
119          Dispose(bi);
120          bi := next;
121      END; (*WHILE*)
122  END; (*DisposeBigInt*)
123
124  (* ReadBigInt: reads BigIntPtr, version for base = 1000
125     Input syntax: BigIntPtr = { digit }.
126                    BigIntPtr = [+ | -] digit { digit }.
127     The empty string is treated as zero, and as the whole
128     input is read into one STRING, max. length is 255.
129     -----*)
130  PROCEDURE ReadBigInt(VAR bi: BigIntPtr);
131      VAR
132          s: STRING; (*input string*)
133          iBeg, iEnd: INTEGER; (*begin and end of proper input *)
134          bigDig, decDig: INTEGER;
135          nrOfBigDigits, lenOfFirst: INTEGER;
136          sign, i, j: INTEGER;
137
138      PROCEDURE WriteWarning(warnPos: INTEGER);
139          BEGIN
140              WriteLn('Warning in ReadBigInt: ',
141                      'character ', s[warnPos],
142                      ' in column ', warnPos, ' is treated as zero');
143          END; (*WriteWarning*)
144
145      BEGIN (*ReadBigInt*)
146          IF base <> 1000 THEN BEGIN
147              WriteLn('Error in ReadBigInt: ',
148                      'procedure currently works for base = 1000 only');
149              Halt;
150          END; (*IF*)

```

```

153   ReadLn(s);
      iEnd := Length(s);
155   IF iEnd = 0 THEN
      bi := Zero
157   ELSE BEGIN

159   (*$IFDEF SIGNED*)
      IF s[1] = '-' THEN BEGIN
161         sign := -1;
         iBeg := 2;
163     END (*THEN*)
      ELSE IF s[1] = '+' THEN BEGIN
165         sign := 1;
         iBeg := 2;
167     END (*THEN*)
      ELSE BEGIN
169   (*$ENDIF*)
         sign := 1;
         iBeg := 1;
171   (*$IFDEF SIGNED*)
      END; (*ELSE*)
173   (*$ENDIF*)

175   WHILE (iBeg <= iEnd) AND
177       ((s[iBeg] < '1') OR (s[iBeg] > '9')) DO BEGIN
      IF (s[iBeg] <> '0') AND (s[iBeg] <> ' ') THEN
179         WriteWarning(iBeg);
         iBeg := iBeg + 1;
181     END; (*WHILE*)

183   (*get value from s[iBeg .. iEnd]*)
      IF iBeg > iEnd THEN
185         bi := Zero
      ELSE BEGIN
187         bi := NIL;
         nrOfBigDigits := (iEnd - iBeg) DIV 3 + 1;
189         lenOfFirst := (iEnd - iBeg) MOD 3 + 1;
         FOR i := 1 TO nrOfBigDigits DO BEGIN
191             bigDig := 0;
             FOR j := iBeg TO iBeg + lenOfFirst - 1 DO BEGIN
193                 IF (s[j] >= '0') AND (s[j] <= '9') THEN
                     decDig := Ord(s[j]) - Ord('0')
195                 ELSE BEGIN
                     WriteWarning(j);
                     decDig := 0;
197                 END; (*ELSE*)
                 bigDig := bigDig * 10 + decDig;
199             END; (*FOR*)
             Prepend(bi, bigDig);
             iBeg := iBeg + lenOfFirst;
201             lenOfFirst := 3;
203         END; (*FOR*)
205   (*$IFDEF SIGNED*)
      Prepend(bi, sign);
207   (*$ENDIF*)
      END; (*IF*)
209   END; (*ELSE*)
      END; (*ReadBigInt*)

```

```

211
213  (* WriteBigInt: writes BigIntPtr, version for base = 1000
    _____ *)
215  PROCEDURE WriteBigInt(bi: BigIntPtr);
    VAR
217      revBi: BigIntPtr;
        n: NodePtr;
219  BEGIN
    IF base <> 1000 THEN BEGIN
221        WriteLn('Error in WriteBigInt: ',
                'procedure currently works for base = 1000 only');
223        Halt;
    END; (* IF *)
225    IF bi = NIL THEN
        Write('0')
227    ELSE BEGIN
    (* $IFDEF SIGNED *)
229        IF Sign(bi) = -1 THEN
            Write('-');
231        revBi := CopyOfBigInt(bi^.next);
    (* $ELSE *)
233        revBi := CopyOfBigInt(bi);
    (* $ENDIF *)
235        InvertBigInt(revBi);
        n := revBi;
237        Write(n^.val); (* first big digit printed without leading zeros *)
        n := n^.next;
239        WHILE n <> NIL DO BEGIN
            IF n^.val >= 100 THEN
241                Write(n^.val)
            ELSE IF n^.val >= 10 THEN
243                Write('0', n^.val)
            ELSE (* n^.val < 10 *)
245                Write('00', n^.val);
            n := n^.next;
247        END; (* WHILE *)
        DisposeBigInt(revBi); (* release the copy *)
249    END; (* IF *)
    END; (* WriteBigInt *)
251
253  FUNCTION ANZ_Nodes(a : BigIntPtr): INTEGER;
    VAR count : INTEGER;
    BEGIN
255        count := 1;

257        WHILE a^.next <> NIL DO
            BEGIN
259                a := a^.next;
                count := count + 1;
261            END;

263        ANZ_Nodes := count;
    END;
265
    (* Returns 2,1,0 - 2,1 determines which is bigger 0 means they are equal *)
267  FUNCTION HigherBigInt(a, b: BigIntPtr) : INTEGER;
    VAR temp_a, temp_b : BigIntPtr;

```

```

269 BEGIN
    temp_a := CopyOfBigInt(a);
271 temp_b := CopyOfBigInt(b);
    InvertBigInt(temp_a);
273 InvertBigInt(temp_b);

275 WHILE (temp_a^.val = temp_b^.val) AND (temp_a^.next <> NIL) AND (temp_b^.next
    <> NIL) DO
    BEGIN
277 temp_a := temp_a^.next;
        temp_b := temp_b^.next;
279 END;

281 IF (temp_a^.next = NIL) AND (temp_b^.next = NIL) THEN HigherBigInt := 0
    ELSE IF (temp_a^.next <> NIL) AND (temp_b^.next = NIL) THEN HigherBigInt := 1
283 ELSE IF (temp_a^.next = NIL) AND (temp_b^.next <> NIL) THEN HigherBigInt :=
        2
    ELSE IF temp_a^.val > temp_b^.val THEN HigherBigInt := 1
285 ELSE HigherBigInt := 2;
END;

287 FUNCTION Sum (a, b: BigIntPtr) : BigIntPtr; (*compute sum = a + b*)
289 VAR result : BigIntPtr;
    VAR sign_a, sign_b, overflow, temp, anz_a, anz_b, ishigher : Integer;
291 BEGIN
293 IF a^.val = 0 THEN result := CopyOfBigInt(b);
295 IF b^.val = 0 THEN result := CopyOfBigInt(a)
    ELSE
297 BEGIN
        result := NIL;
299 overflow := 0;
        ishigher := 0;
301 sign_a := Sign(a);
        sign_b := Sign(b);
303 anz_a := ANZ_Nodes(a);
        anz_b := ANZ_Nodes(b);
305
        IF anz_a > anz_b THEN ishigher := 1
307 ELSE IF anz_b > anz_a THEN ishigher := 2
        ELSE ishigher := HigherBigInt(a,b);
309
        (* Set the sign of the result *)
311 IF ((sign_a = 1) OR (sign_a = 0)) AND ((sign_b = 1) OR (sign_a = 0)) THEN
            Append(result,1)
        ELSE IF (sign_a = -1) AND (sign_b = -1) THEN Append(result,-1)
313 ELSE IF (sign_a = -1) AND (sign_b = 1) THEN
            IF ishigher = 1 THEN Append(result,-1) ELSE Append(result,1)
315 ELSE
            IF ishigher = 1 THEN Append(result,1) ELSE Append(result,-1);
317
        IF (ishigher = 0) AND (sign_a <> sign_b) THEN
319 BEGIN
            result^.val := 1;
321 Append(result,0);
        END
323 ELSE

```



```

BEGIN
325 REPEAT
    IF a^.next <> NIL THEN a := a^.next ELSE a^.val := 0;
327 IF b^.next <> NIL THEN b := b^.next ELSE b^.val := 0;

329 IF ((sign_a = 1) AND (sign_b = 1)) OR ((sign_a = -1) AND (sign_b = -1))
THEN
    BEGIN
331 temp := a^.val + b^.val + overflow;
        overflow := 0;
333 END
    ELSE
335 BEGIN
        IF ishigher = 1 THEN
337 BEGIN
            IF a^.val >= b^.val THEN
339 BEGIN
                temp := a^.val - b^.val + overflow;
341 overflow := 0;
            END
            ELSE BEGIN
343 temp := (1000 + a^.val) - b^.val + overflow;
345 overflow := -1;
            END;
        END
        ELSE
347 BEGIN
            IF b^.val >= a^.val THEN
351 BEGIN
                temp := b^.val - a^.val + overflow;
353 overflow := 0;
            END
            ELSE BEGIN
355 temp := (1000 + b^.val) - a^.val + overflow;
357 overflow := -1;
            END;
        END;
    END;
359 END;

361 IF temp >= 1000 THEN
363 BEGIN
        overflow := 1;
        temp := temp - 1000;
365 END
    ELSE IF temp < 0 then
367 BEGIN
        temp := 1000 + temp;
        overflow := - 1;
369 END;
371 END;

373 IF (temp = 0) AND ((a^.next = NIL) AND (b^.next = NIL)) THEN ELSE
Append(result ,temp);

375 UNTIL ((a^.val = 0) AND (b^.val = 0)) AND ((a^.next = NIL) AND (b^.next =
NIL));
377 END;
END;

```

```

379   Sum := result;
END;
381
(* Product of two Big Ints *)
383 FUNCTION Product(a, b: BigIntPtr): BigIntPtr; (*compute product = a * b*)
    VAR
385         result : BigIntPtr;
        sum_a, sum_b, sum_ab : int64;
387         i : INTEGER;
    BEGIN
389
        IF (a^.val = 0) OR (b^.val = 0) THEN
391            BEGIN
                result := NewNode(1);
393                Append(result, 0);
            END
        ELSE IF Sign(a) <> Sign(b) THEN
395            result := NewNode(-1)
        ELSE
397            result := NewNode(1);
399
            i := 0;
401            sum_a := 0;
            sum_b := 0;
403
            a := a^.next;
405            b := b^.next;
407
            WHILE a <> NIL DO BEGIN
                sum_a := sum_a + (a^.val * (base ** i));
409                a := a^.next;
                i := i + 1;
411            END;
413
            i := 0;
415
            WHILE b <> NIL DO BEGIN
                sum_b := sum_b + (b^.val * (base ** i));
417                b := b^.next;
                i := i + 1;
419            END;
421
            sum_ab := sum_a * sum_b;
423
            WHILE sum_ab <> 0 DO
                BEGIN
425                    Append(result, (sum_ab MOD base));
                    sum_ab := sum_ab DIV base;
427                END;
                Product := result;
429            END;
431
(*==== main program, for test purposes ====*)
433
    VAR
        bi : BigIntPtr;
435        bi2: BigIntPtr;
        bi_temp : BigIntPtr;

```

```

437     bi2_temp : BigIntPtr;
        sumbi : BigIntPtr;
439     probi : BigIntPtr;

441 BEGIN (* BigInts *)

443     WriteLn(chr(205), chr(205), chr(185), ' BigInt ', chr(204), chr(205), chr(205));

445     (* tests for ReadBigInt and WriteBigInt only *)
447     Write('big int > ');
        ReadBigInt(bi);

449     Write('big int > ');
        ReadBigInt(bi2);

451     bi_temp := CopyOfBigInt(bi);
453     bi2_temp := CopyOfBigInt(bi2);

455     WriteLn;

457     Write('Sum : ');
        sumbi := Sum(bi, bi2);
459     WriteBigInt(sumbi);
        WriteLn;

461     Write('Product : ');
463     probi := Product(bi_temp, bi2_temp);
        WriteBigInt(probi);

465 END. (* BigInts *)

```

BigInts.pas

```

C:\WINDOWS\system32\cmd.exe
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung08>BigInts.exe
== BigInt ==
big int > 123456
big int > 1234

Sum : 124690
Product : 152344704
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung08>BigInts.exe
== BigInt ==
big int > -1200
big int > 300

Sum : -900
Product : -360000
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung08>BigInts.exe
== BigInt ==
big int > 0
big int > 101

Sum : 101
Product : 0
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung08>

```

Abbildung 1: Testfälle BigInts

Testfälle

Die Testfälle zeigen die Addition / Multiplikation mit positiven und negativen Zahlen. Das Addieren funktioniert solange genug Speicher für die Liste vorhanden ist. Beim Multiplizieren wird ein Laufzeitfehler verursacht wenn der Maximale Wertebereich von Int64 unter/überschritten wird. Alternativ kann hier mit einem String gearbeitet werden, jedoch ist dieser auch “beschränkt“ auf 255 Zeichen, “unendlich lange“ Zahlen können damit aber nicht verwirklicht werden.

Aufgabe 2

Lösungsidee

Bei dieser Aufgabe wird eine Wish List erstellt die Wünsche von einem Text Dokument einliest. Aufgrund dieser Liste wird eine Order Liste erstellt. In dieser sind alle Dinge die sich die Kinder wünschen Mengenmäßig enthalten. Danach wird eine Delivery Liste erstellt, in der alle Wünsche für das jeweilige Kind enthalten ist. Es werden für die Ausgabe, Node-erstellung und anfügen an eine Liste Funktionen bzw. Prozeduren erstellt.

```

1  (* WLA:                                     HDO, 2016-12-06
3      Wish list analyzer for the Christkind.
=====*)
5 PROGRAM WLA;
7  (*$IFDEF WINDOWS*)
8      USES
9          WinCrt;
10  (*$ENDIF*)
11
12  TYPE
13
14      WishNodePtr = ^WishNode;
15      WishNode = RECORD
16          next: WishNodePtr;
17          name: STRING;
18          item: STRING;
19      END; (*RECORD*)
20      WishListPtr = WishNodePtr;
21
22      OrderNodePtr = ^OrderNode;
23      OrderNode = RECORD
24          next: OrderNodePtr;
25          item: STRING;
26          n: INTEGER;
27      END; (*RECORD*)
28      OrderList = OrderNode;
29
30      ItemNodePtr = ^ItemNode;
31      ItemNode = RECORD
32          next : ItemNodePtr;
33          item : STRING;
34      END;
35
36      DelivNodePtr = ^DelivNode;
37      DelivNode = RECORD
38          next: DelivNodePtr;
39          name: STRING;
40          items: ItemNodePtr;
41      END; (*RECORD*)
42      DelivListPtr = DelivNodePtr;
43
44  (*#####*)
45  (*  Wishes  *)

```

```

(*#####*)
49 (* New WishNode Function *)
51 FUNCTION newWishNode(line : STRING): WishListPtr;
VAR node : WishNodePtr;
53 BEGIN
    New(node);
55     node^.name := Copy(line,1,pos(':',line)-1);
    node^.item := Copy(line,pos(' ',line)+1,length(line));
57     node^.next := NIL;
    newWishNode := node;
59 END;

61 (* Append to a Wish List*)
PROCEDURE appendToWishList(VAR list : WishNodePtr; node : WishNodePtr);
63 VAR wishList : WishNodePtr;
BEGIN
65     IF list = NIL THEN list := node
    ELSE
67         BEGIN
            wishList := list;
69
            WHILE (wishList^.next <> NIL) DO
71                 wishList := wishList^.next;

73         wishList^.next := node;
    END;
75 END;

77 (*#####*)
(* ORDER *)
79 (*#####*)

81 (* New Order Node*)
FUNCTION newOrderNode(item : STRING; anz : INTEGER): OrderNodePtr;
83 VAR node : OrderNodePtr;
BEGIN
85     New(node);
    node^.item := item;
87     node^.n := anz;
    node^.next := NIL;
89     NewOrderNode := node;
    END;

91
(* Append to an Order List *)
93 PROCEDURE appendToOrder(VAR list : OrderNodePtr; node : OrderNodePtr);
VAR orderList : OrderNodePtr;
95 BEGIN
    IF list = NIL THEN list := node
97     ELSE
    BEGIN
99         orderList := list;

101         WHILE (orderList^.next <> NIL) DO
            orderList := orderList^.next;

103         orderList^.next := node;
105     END;

```

```

END;
107
(* Increase the number of items per order *)
109 PROCEDURE increaseOrder(VAR list : OrderNodePtr; s_item : STRING);
VAR temp_orderlist : OrderNodePtr;
111 BEGIN
    IF list = NIL THEN appendToOrder(list , newOrderNode(s_item , 1))
113 ELSE
    BEGIN
115         temp_orderlist := list;
117         WHILE(temp_orderlist <> NIL) DO
        BEGIN
119             IF(temp_orderlist^.item = s_item) THEN
121             BEGIN
                temp_orderlist^.n := temp_orderlist^.n + 1;
123                 exit;
                END;
125             temp_orderlist := temp_orderlist^.next;
127         END;
129         appendToOrder(list , newOrderNode(s_item , 1));
        END;
131 END;

133 (* Generate Order List from Wish List*)
FUNCTION orderListOf(wishlist : WishNodePtr): OrderNodePtr;
135 VAR result : OrderNodePtr;
BEGIN
137     result := NIL;
    WHILE wishlist <> NIL DO
139     BEGIN
        increaseOrder(result , wishlist^.item);
141         wishlist := wishlist^.next;
        END;
143     orderListOf := result;
    END;
145

147 (*#####*)
(* Items *)
149 (*#####*)

151 (* New Item Node *)
FUNCTION newItemNode(item : STRING): ItemNodePrt;
153 VAR node : ItemNodePrt;
BEGIN
155     New(node);
    node^.item := item;
157     node^.next := NIL;
    newItemNode := node;
159 END;

161 (* Append To Item List *)
PROCEDURE appendItem(VAR list : ItemNodePrt; element : ItemNodePrt);
163 VAR tmp : ItemNodePrt;

```

```

BEGIN
165 IF list = NIL THEN list := element
    ELSE
167 BEGIN
        tmp := list;
169
        WHILE (tmp^.next <> NIL) DO
171             tmp := tmp^.next;

173     tmp^.next := element;
    END;
175 END;

177 (*#####*)
178 (*  Delivery  *)
179 (*#####*)

181 (* Get New DeliveryList *)
FUNCTION newDelivNode(name,item : STRING): DelivNodePtr;
183 VAR node : DelivNodePtr;
BEGIN
185     New(node);
        node^.name := name;
187     node^.items :=NewItemNode(item);
        node^.next := NIL;
189     NewDelivNode := node;
END;

191
(* Append to a Deliverylist *)
193 PROCEDURE appendToDelivery(VAR list : DelivNodePtr; element : DelivNodePtr);
VAR tmp : DelivNodePtr;
195 BEGIN
    IF list = NIL THEN list := element ELSE
197 BEGIN
        tmp := list;
199
        WHILE (tmp^.next <> NIL) DO
201             tmp := tmp^.next;

203     tmp^.next := element;
    END;
205 END;

207 (* Append Item to Kids Item List *)
PROCEDURE appendItemKid(VAR list : DelivNodePtr; name, item : STRING);
209 VAR tmp : DelivNodePtr;
BEGIN
211     IF list = NIL THEN list := NewDelivNode(name, item)
        ELSE
213 BEGIN

215     tmp := list;
        WHILE tmp <> NIL DO BEGIN
217             IF tmp^.name = name THEN BEGIN
                AppendItem(tmp^.items ,NewItemNode(item));
219                 exit;
                END;
            END;
221             tmp := tmp^.next;

```



```

    END;
223     appendToDelivery(list , NewDelivNode(name, item));
225     END;
END;
227
(* Generate Delivery based on wish list *)
229 FUNCTION DeliveryListOf(wishlist : WishNodePtr): DelivNodePtr;
VAR result : DelivNodePtr;
231 BEGIN
    result := NIL;
233     WHILE wishlist <> NIL DO BEGIN
        AppendItemKid(result , wishlist^.name, wishlist^.item);
235         wishlist := wishlist^.next;
    END;
237     DeliveryListOf := result;
END;
239
(*#####*)
241 (* Printing to Console *)
(*#####*)
243
(* Write Wish List to Console *)
245 PROCEDURE writeWishList(wishList : WishNodePtr);
BEGIN
247     WriteLn(chr(205),chr(205), ' Wish list ',chr(205),chr(205));

249     WHILE wishList <> NIL DO
        BEGIN
251             Writeln(wishList^.name, ' : ' , wishList^.item);
            wishList := wishList^.next;
253         END;

255     WriteLn(chr(205),chr(205), ' End Wish list ',chr(205),chr(205));
END;
257
(* Write OrderList to Console *)
259 PROCEDURE writeOrderList(list : OrderNodePtr);
BEGIN
261     WriteLn(chr(205),chr(205), ' Order list ',chr(205),chr(205));

263     WHILE list <> NIL DO
        BEGIN
265             Writeln(list^.item, ' : ' , list^.n);
            list := list^.next;
267         END;

269     WriteLn(chr(205),chr(205), ' End Order list ',chr(205),chr(205));
END;
271
(* Write Delivery List *)
273 PROCEDURE writeDelivList(list : DelivListPtr);
VAR citeM: ItemNodePrt;
275 BEGIN
    WriteLn(chr(205),chr(205), ' Delv list ',chr(205),chr(205));

277     WHILE list <> NIL DO BEGIN
279         Write(list^.name, ' : ');

```

```

    citem := list^.items;
281
    WHILE citem <> NIL DO BEGIN
283        write(citem^.item, ' ');
        citem := citem^.next;
285    END;
    writeln();
287    list := list^.next;
    END;
289
    WriteLn(chr(205),chr(205), ' End Delv list ',chr(205),chr(205));
291 END;

293 VAR
    wishesFile: TEXT;
295    s: STRING;
    wish_list : WishNodePtr;
297    order_list : OrderNodePtr;
    delv_list : DelivNodePtr;
299
    BEGIN (*WLA*)
301
    WriteLn(chr(205),chr(205),chr(185), ' Lists for Xmas ',chr(204),chr(205),chr
        (205));
303    WriteLn;

305    (* Read everyline from txt, appendtowishlist and close file *)
    Assign(wishesFile, 'Wishes.txt');
307    Reset(wishesFile);

309    REPEAT
        ReadLn(wishesFile, s);
311        appendToWishList(wish_list,newWishNode(s));
    UNTIL Eof(wishesFile);
313    Close(wishesFile);

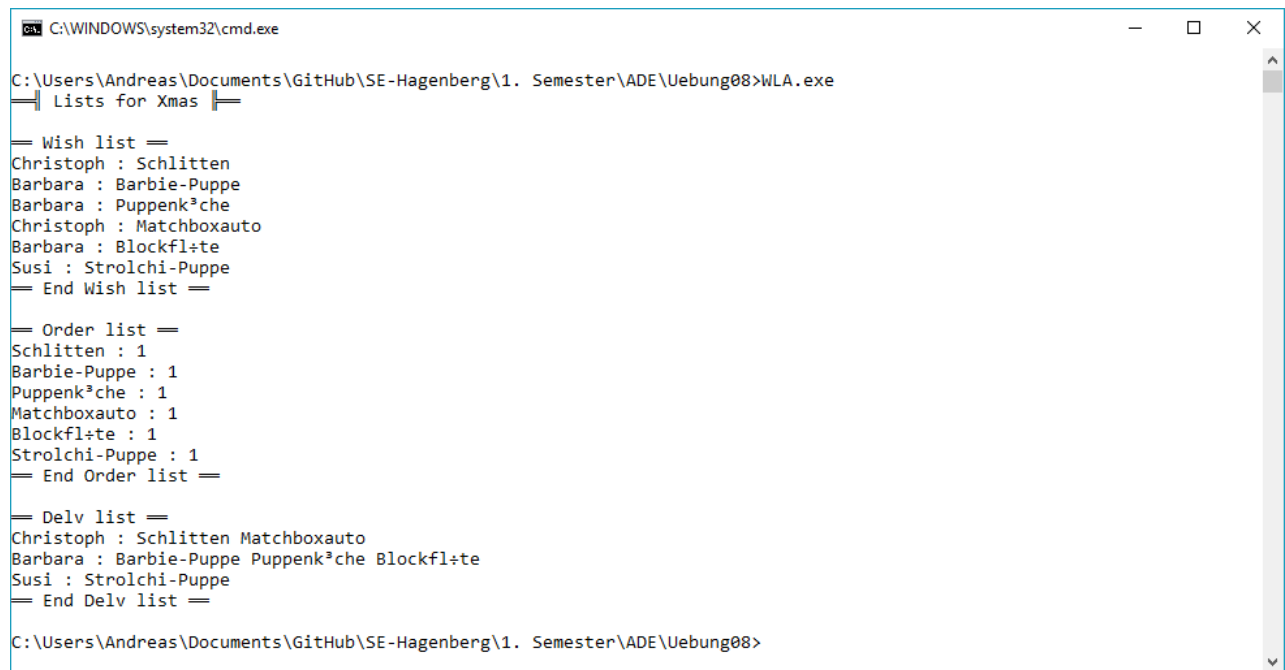
315    (* Write wishlist to console *)
    writeWishList(wish_list);
317    WriteLn;

319    (* Generating & Writing OrderList to Console *)
    order_list := orderListOf(wish_list);
321    writeOrderList(order_list);
    WriteLn;
323

    (* Generating & Writing DeliveryList to Console *)
325    delv_list := DeliveryListOf(wish_list);
    writeDelivList(delv_list);
327
    END. (*WLA*)

```

WLA.pas



```
C:\WINDOWS\system32\cmd.exe

C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung08>WLA.exe
== Lists for Xmas ==

== Wish list ==
Christoph : Schlitten
Barbara : Barbie-Puppe
Barbara : Puppenk³che
Christoph : Matchboxauto
Barbara : Blockfl³te
Susi : Strolchi-Puppe
== End Wish list ==

== Order list ==
Schlitten : 1
Barbie-Puppe : 1
Puppenk³che : 1
Matchboxauto : 1
Blockfl³te : 1
Strolchi-Puppe : 1
== End Order list ==

== Delv list ==
Christoph : Schlitten Matchboxauto
Barbara : Barbie-Puppe Puppenk³che Blockfl³te
Susi : Strolchi-Puppe
== End Delv list ==

C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung08>
```

Abbildung 2: Testfälle WLA

Testfall

Hier wird gezeigt, dass die WishList die Wünsche der Kinder aus dem Textdokument enthält. Die OrderList wird aufgrund dieser Liste erzeugt. Die DeliveryList enthält Kinder mit ihren Wünschen basierend auf der WishList.