

<input type="checkbox"/> Gr. 1, Dr. G. Kronberger	Name <u>Andreas Roither</u>	Aufwand in h <u>6 h</u>
<input type="checkbox"/> Gr. 2, Dr. H. Gruber		
<input checked="" type="checkbox"/> Gr. 3, Dr. D. Auer	Punkte _____	Kurzzeichen Tutor / Übungsleiter _____ / _____

Pascal bietet zwar einen Datentyp zur Repräsentation von Mengen (*SET OF ...*), allerdings können die Elemente solcher Mengen nur ganze Zahlen aus dem Bereich von 0 .. 255, Zeichen (also Werte des Datentyps *CHAR*) oder Werte eines Aufzählungsdantentyps sein. – Wir haben diesen *SET*-Datentyp nicht behandelt, da er zu restriktiv ist und in anderen Sprachen (z. B. in C, C++ und Java) so nicht zur Verfügung steht.

Klassenbasierte objektorientierte Programmiersprachen (wie Borland Pascal, C++, Java und C#) bieten mit eben diesen Klassen aber die Möglichkeit, benutzerdefinierte Datentypen auf einfachere Art und Weise zu realisieren, als es bisher mit dem ADT-Konzept möglich war.

1. Zeichenketten-Menge als Klasse (10 Punkte)

Entwerfen Sie eine Klasse *SOS* (für *set of strings*), deren Objekte eine Realisierung des mathematischen Konzepts einer Menge von Elementen des Datentyps *STRING* darstellen. Verwenden Sie zur internen Speicherung der Elemente eine Datenkomponente *elements* (ein Feld von Zeichenketten) und eine Datenkomponente *n* (Zähler für die Anzahl der Elemente).

Stellen Sie mindestens folgende Methoden zur Verfügung:

- Einen Konstruktor und einen Destruktor,
- die Methoden *Empty*, *Cardinality*, *Add*, *Remove* und *Contains* sowie
- Methoden, welche die aus der Mathematik bekannten Mengenoperationen *Union*, *Intersection*, *Difference* und *Subset* implementieren.

Testen Sie Ihre Klasse ausführlich, indem Sie statische und dynamische Objekte anlegen und alle Operationen darauf ausführen.

2. Säcke (14 Punkte)

Jeder kennt einen Sack. – Denken Sie z. B. an einen Plastiksack: In einen solchen kann man mehrere, auch gleiche Dinge hineinstecken.

Im mathematischen Sinn gilt: Ein Sack (*bag*) ist eine Menge (*set*), die Elemente auch mehrfach enthalten kann. Somit könnte man auf die Idee kommen, die Klasse *BOS* (für *bag of strings*) von der Klasse *SOS* aus Aufgabe 1 abzuleiten¹.

Versuchen Sie, möglichst viel von der Basisklasse *SOS* zu nutzen, indem Sie in der abgeleiteten Klasse *BOS* nur eine weitere Datenkomponente *counters* (ein Feld von ganzen Zahlen) hinzufügen, welche für alle Elemente in *elements* angibt, wie oft diese Elemente im Sack vorkommen.

Passen Sie alle Mengenoperationen in *BOS* an die Semantik von Säcken an.

Testen Sie Ihre Klasse ausführlich, indem sie statische und dynamische Objekte anlegen und alle Operationen darauf ausführen.

¹ Ob das im Sinne der OOP besonders klug ist, werden wir in einem höheren Semester bei der Behandlung des Liskov'schen Substitutionsprinzips näher besprechen.

Übung 8

Aufgabe 1 und Aufgabe 2 befinden in einem .pas file: "SOSU.pas". Die Testfälle werden in eigenen .pas files behandelt. Lösungsidee und Testfälle sind auf den nachfolgenden Seiten. Die Implementationen der Klassen sind durch Kommentare gekennzeichnet.

SOSU Unit

```

1  (* ===== *)
2  (* --          SOS UNIT 27.05.17          -- *)
3  (* ===== *)

5  UNIT SOSU;

7  INTERFACE
8      CONST
9          MAX = 10;

11     TYPE
12         (* ===== *)
13         (* --          SOS Class          -- *)
14         (* ===== *)

15         SOSPtr = ^SOS;
16         SOS = OBJECT
17             PRIVATE
18                 n: INTEGER;
19                 a: ARRAY[1..MAX] OF STRING;
20             PUBLIC
21                 CONSTRUCTOR Init;
22                 DESTRUCTOR Done;                                VIRTUAL;

23                 FUNCTION Empty: BOOLEAN;                        VIRTUAL;
24                 FUNCTION Contains(val: STRING): BOOLEAN;        VIRTUAL;
25                 FUNCTION ContainsPos(val: STRING): INTEGER;     VIRTUAL;
26                 FUNCTION Cardinality: INTEGER;                  VIRTUAL;
27                 PROCEDURE Add(val: STRING);                      VIRTUAL;
28                 PROCEDURE Remove(val: STRING);                  VIRTUAL;
29                 FUNCTION LookupPos(val: INTEGER): STRING;       VIRTUAL;
30                 PROCEDURE Print;                                VIRTUAL;

31                 (* set operations *)
32                 FUNCTION Union(b: SOSPtr): SOSPtr;              VIRTUAL;
33                 FUNCTION Intersection(b: SOSPtr): SOSPtr;       VIRTUAL;
34                 FUNCTION Difference(b: SOSPtr): SOSPtr;         VIRTUAL;
35                 FUNCTION Subset(b: SOSPtr): BOOLEAN;            VIRTUAL;

36     END;
37
38
39

```

```

41      (* ===== *)
42      (* --          Sack Class          -- *)
43      (* ===== *)

45      SackPtr = ^Sack;
46      Sack = OBJECT(SOS)
47      PRIVATE
48          counters: ARRAY[1..MAX] OF INTEGER;
49      PUBLIC
50          CONSTRUCTOR Init;
51          DESTRUCTOR Done;

53          FUNCTION Cardinality: INTEGER;
54          PROCEDURE Add(val: STRING);
55          PROCEDURE AddAmount(val: STRING; amount: INTEGER);
56          PROCEDURE Remove(val: STRING);
57          PROCEDURE LookupPos(val: INTEGER; VAR s: STRING; VAR amount:
INTEGER);
          PROCEDURE Print;

59          (* set operations *)
60          FUNCTION Union(b: SackPtr): SackPtr;
61          FUNCTION Difference(b: SackPtr): SackPtr;
62          FUNCTION Intersection(b: SackPtr): SackPtr;
63      END;

65 IMPLEMENTATION

67      (* ===== *)
68      (* --          Implementation of SOS          -- *)
69      (* ===== *)

71      CONSTRUCTOR SOS.Init;
72      BEGIN
73          n := 0;
74      END;

75      DESTRUCTOR SOS.Done;
76      BEGIN
77          (* do nothing *)
78      END;

79      FUNCTION SOS.Empty: BOOLEAN;
80      BEGIN
81          Empty := n = 0;
82      END;

83      (* Returns TRUE if found*)

```

```
FUNCTION SOS.Contains(val: STRING): BOOLEAN;
89   VAR i: INTEGER;
BEGIN
91   Contains := FALSE;

93   FOR i := 1 TO n DO BEGIN
       IF a[i] = val THEN BEGIN
95       Contains := TRUE;
           break;
97       END;
       END;
99   END;

101  (* returns pos of the found element *)
FUNCTION SOS.ContainsPos(val: STRING): INTEGER;
103   VAR i: INTEGER;
BEGIN
105   ContainsPos := 0;
       FOR i := 1 TO n DO BEGIN
107       IF a[i] = val THEN BEGIN
           ContainsPos := i;
109       break;
           END;
111       END;
       END;
113
FUNCTION SOS.Cardinality: INTEGER;
115 BEGIN
       Cardinality := n;
117 END;

119  (* Adds to value to array, returns error msg if array is full *)
PROCEDURE SOS.Add(val: STRING);
121 BEGIN
       IF n < MAX THEN BEGIN
123         n := n + 1;
           a[n] := val;
125       END ELSE WriteLn('== ERROR ADD: Full! ==', #13#10);
       END;
127
PROCEDURE SOS.Remove(val: STRING);
129   VAR i,j: INTEGER;
BEGIN
131   IF NOT Empty THEN BEGIN
       FOR i := 0 TO n DO BEGIN
133       IF a[i] = val THEN BEGIN
           FOR j := i TO n - 1 DO BEGIN
135             a[j] := a[j + 1];
```

```

137         END;
        n := n - 1;
        break;
139     END;
    END;
141 END ELSE WriteLn('Empty');
END;

143
(* Returns value from array at position * *)
145 FUNCTION SOS.LookupPos(val: INTEGER): STRING;
BEGIN
147     LookupPos := '';
    IF NOT Empty THEN BEGIN
149         IF (val <= n) AND (val >= 1) THEN LookupPos := a[val];
    END;
151 END;

153
(* Print all elements to the console *)
PROCEDURE SOS.Print;
155     VAR
        i: INTEGER;
157 BEGIN
    FOR i := 1 TO n DO Write(a[i], ' ');
159     WriteLn;
END;

161
(* Union of two sos objects
   Returns SOSPtr *)
163 FUNCTION SOS.Union(b: SOSPtr): SOSPtr;
165 VAR
    i: INTEGER;
    s: SOSPtr;
167 BEGIN
    New(s, Init);
    IF (Cardinality + b^.Cardinality) <= MAX THEN BEGIN
169         FOR i := 1 TO n DO BEGIN
            s^.Add(a[i]);
171         END;
            FOR i := 1 TO b^.Cardinality DO BEGIN
173                 s^.Add(b^.LookupPos(i));
175             END;
        END
177     ELSE BEGIN
179         WriteLn('== Union Error ==');
        WriteLn(' Too big', #13#10' Cardinality m1: ', Cardinality, ' m2: '
181             , b^.Cardinality, '- MAX size: ', MAX);
    END;
    Union := s;
183

```

```

END;

185
(* Intersection of two sos objects
187   Returns SOSPtr *)
FUNCTION SOS.Intersection(b: SOSPtr): SOSPtr;
189   VAR
        i: INTEGER;
191        s: SOSPtr;
BEGIN
193   New(s, Init);
   FOR i := 1 TO n DO BEGIN
195       IF b^.Contains(a[i]) THEN s^.Add(a[i]);
   END;
197   IF s^.Cardinality = 0 THEN BEGIN
       WriteLn('== Intersection ==');
199       WriteLn(' Nothing in common');
   END;
201   Intersection := s;
END;

203
(* Difference of two sos objects
205   Returns SOSPtr *)
FUNCTION SOS.Difference(b: SOSPtr): SOSPtr;
207   VAR
        i: INTEGER;
209        s: SOSPtr;
BEGIN
211   New(s, Init);
   FOR i := 1 TO n DO BEGIN
213       IF NOT b^.Contains(a[i]) THEN s^.Add(a[i]);
   END;
215   IF s^.Cardinality = 0 THEN BEGIN
       WriteLn('== Difference ==');
217       WriteLn(' No difference');
   END;
219   Difference := s;
END;

221
(* True if calling object is subset of b *)
223 FUNCTION SOS.Subset(b: SOSPtr): BOOLEAN;
   VAR
225     i: INTEGER;
BEGIN
227   Subset := TRUE;
   FOR i := 1 TO n DO BEGIN
229       IF NOT b^.Contains(a[i]) THEN BEGIN
           Subset := FALSE;
231       break;

```

```

        END;
233    END;
    END;

235
    (* ===== *)
237    (* --      Implementation of Sack      -- *)
    (* ===== *)

239
    CONSTRUCTOR Sack.Init;
241    BEGIN
        INHERITED Init;
243    END;

245
    DESTRUCTOR Sack.Done;
    BEGIN
247        INHERITED Done;
    END;

249
    PROCEDURE Sack.Remove(val: STRING);
251    VAR i,j: INTEGER;
    BEGIN
253        i := ContainsPos(val);
        INHERITED Remove(val);
255        IF i > 0 THEN BEGIN
            FOR j := i TO n - 1 DO BEGIN
257                counters[j] := counters[j + 1];
            END;
259        END;
    END;

261
    (* Adds to value to array, returns error msg if array is full *)
263    PROCEDURE Sack.Add(val: STRING);
        VAR
265        pos: INTEGER;
    BEGIN
267        pos := ContainsPos(val);
        IF pos = 0 THEN BEGIN
269            INHERITED Add(val);
            counters[n] := counters[n] + 1;
271        END
        ELSE counters[pos] := counters[pos] + 1;
273    END;

275
    (* Adds to value to array, returns error msg if array is full *)
    PROCEDURE Sack.AddAmount(val: STRING; amount: INTEGER);
277    VAR
        pos: INTEGER;
279    BEGIN

```

```

pos := ContainsPos(val);
281 IF pos = 0 THEN BEGIN
    INHERITED Add(val);
283     counters[n] := counters[n] + amount;
    END
285 ELSE counters[pos] := counters[pos] + amount;
END;

287
FUNCTION Sack.Cardinality: INTEGER;
289     VAR
        i: INTEGER;
291 BEGIN
    Cardinality := 0;
293     FOR i := 1 TO n DO BEGIN
        Cardinality := Cardinality + counters[i];
295     END;
END;

297
(* Returns value from array at position * *)
299 PROCEDURE Sack.LookupPos(val: INTEGER; VAR s: STRING; VAR amount:
    INTEGER);
301 BEGIN
    IF NOT Empty THEN BEGIN
303         IF (val <= n) AND (val >= 1) THEN BEGIN
            s := a[val];
305             amount := counters[val];
        END;
307     END;
END;

309
PROCEDURE Sack.Print;
311     VAR
        i: INTEGER;
313 BEGIN
    FOR i := 1 TO n DO Write(a[i], ' x', counters[i], ' ');
315     WriteLn;
END;

317
(* Union of two sos objects
319     Returns SackPtr *)
FUNCTION Sack.Union(b: SackPtr): SackPtr;
321 VAR
    i, amount: INTEGER;
323     s: SackPtr;
    str: STRING;
325 BEGIN
    New(s, Init);
327     str := '';

```



```

amount := 0;

329
IF (Cardinality + b^.Cardinality) <= MAX THEN BEGIN
331   FOR i := 1 TO n DO BEGIN
       s^.AddAmount(a[i], counters[i]);
333   END;
       FOR i := 1 TO b^.Cardinality DO BEGIN
335         b^.LookupPos(i, str, amount);
           s^.AddAmount(str, amount);
337       END;
     END
339   ELSE BEGIN
       WriteLn('== Union Error ==');
341       WriteLn(' Too big', #13#10' Cardinality m1: ', Cardinality, ' m2: '
           , b^.Cardinality, '- MAX size: ', MAX);
343     END;
     Union := s;
345   END;

347   (* Difference of two sos objects
       Returns SackPtr *)
349   FUNCTION Sack.Difference(b: SackPtr):SackPtr;
       VAR
351         i: INTEGER;
           s: SackPtr;
353     BEGIN
       New(s, Init);
355       FOR i := 1 TO n DO BEGIN
           IF NOT b^.Contains(a[i]) THEN s^.AddAmount(a[i], counters[i]);
357       END;
       IF s^.Cardinality = 0 THEN BEGIN
359         WriteLn('== Difference ==');
           WriteLn(' No difference');
361       END;
       Difference := s;
363     END;

365   (* Intersection of two sos objects
       Returns SOSPtr *)
367   FUNCTION Sack.Intersection(b: SackPtr): SackPtr;
       VAR
369         i: INTEGER;
           s: SackPtr;
371     BEGIN
       New(s, Init);
373       FOR i := 1 TO n DO BEGIN
           IF b^.Contains(a[i]) THEN s^.Add(a[i]);
375     END;

```

```
377     IF s^.Cardinality = 0 THEN BEGIN
        WriteLn('== Intersection ==');
        WriteLn(' Nothing in common');
379     END;
    Intersection := s;
381 END;
END.
```

SOSU.pas

Aufgabe 1

Lösungsidee

Es werden öffentliche Methoden und Funktionen erstellt mit deren Hilfe Mengenoperationen durchgeführt werden können. Die Mengenoperationen werden als Funktionen implementiert, da die Ursprungsmengen nicht verändert werden sollen damit weitere Operationen auf dieselben durchgeführt werden können. Zurückgegeben wird ein Pointer auf ein SOS Objekt. In den Testfällen wird dieser verwendet um die Ergebnisse der Mengenoperationen auszugeben. Es wurden zusätzlich zu den vorgegebenen Operationen neue wie "LookupPos" hinzugefügt um das Einfügen und Verwalten in das Array zu vereinfachen.

Testfälle

Zum Testen werden alle Operationen ausgeführt. Bei Union wird eine Fehlermeldung ausgegeben falls das resultierende Objekt der Union zu groß wäre. Auch Operationen wie "Add" haben eine Fehlerüberprüfung.

```

2  (* ===== *)
3  (* --          SOS TEST 27.05.17          -- *)
4  (* ===== *)
5
6  PROGRAM SOSTest;
7      USES SOSU;
8
9      VAR
10         s, s1, s2, s3: SOSPtr;
11 BEGIN
12     WriteLn('=== SOSTest ===');
13
14     New(s, Init);
15     New(s1, Init);
16     New(s2, Init);
17
18     s^.Add('Hallo');
19     s^.Add('ich');
20
21     s1^.Add('Hallo');
22     s1^.Add('ich');
23     s1^.Add('bin');
24     s1^.Add('eine');
25     s1^.Add('Unit');
26
27     s2^.Add('Hallo');
28     s2^.Add('its');
29     s2^.Add('me');
30     s2^.Add('bin');
31     s2^.Add('Unit');

```

```
32  (* Cardinality Test *)
    WriteLn;
34  WriteLn('Cardinality s1: ', s1^.Cardinality);
    WriteLn('Cardinality s2: ', s2^.Cardinality);
36  WriteLn;

38  (* Successful union *)
    WriteLn('-- Union --');
40  s3 := s1^.Union(s2);
    s3^.Print();
42  s3^.Remove('Hallo');
    WriteLn('-- Removed Hallo --');
44  s3^.Print();

46  (* unsuccessful union *)
    WriteLn;
48  s2^.Add('T');
    s2^.Add('T2');
50  s2^.Add('T3');
    s2^.Add('T4');
52  s2^.Add('T5');
    s2^.Add('T6');
54  s3 := s1^.Union(s2);
    s3^.Print();

56

58  (* Difference Test *)
    WriteLn('-- Difference --');
    s3 := s1^.Difference(s2);
60  s3^.Print();
    WriteLn;

62

64  (* Intersection Test *)
    WriteLn('-- Intersection --');
    s3 := s1^.Intersection(s2);
66  s3^.Print();
    WriteLn;

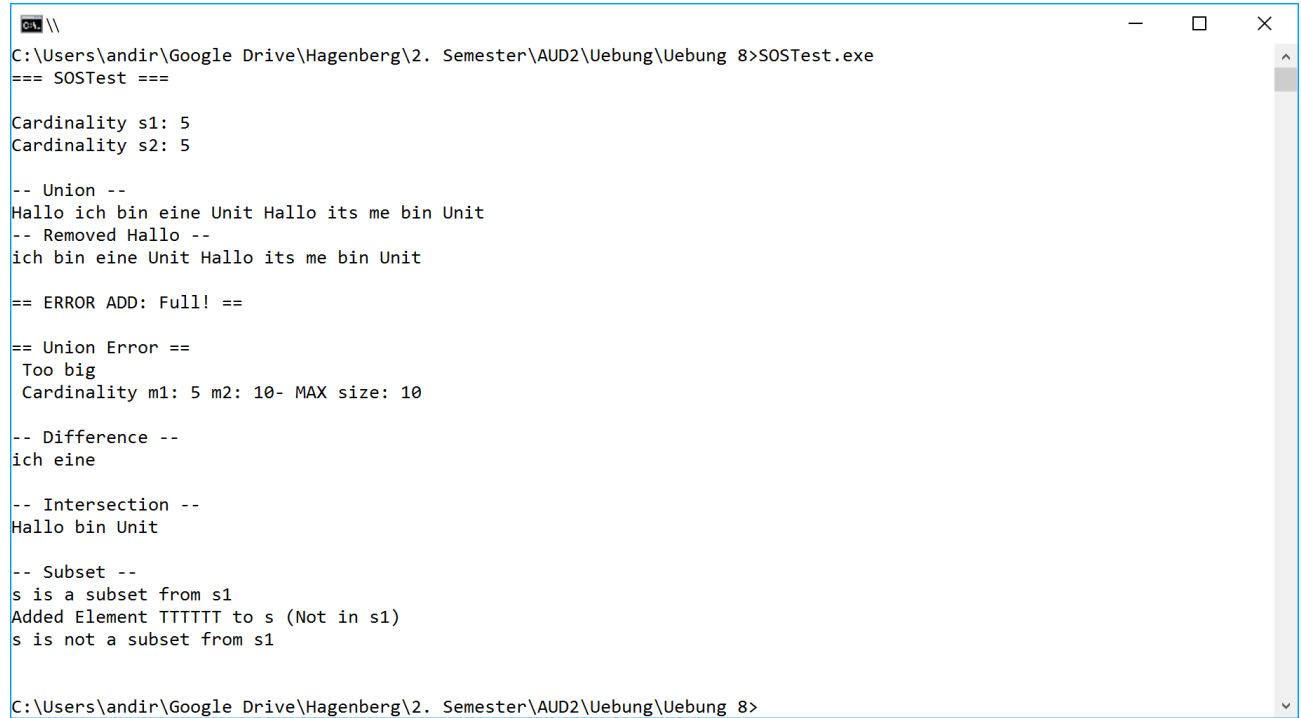
68

70  (* Subset Test, first true, second false *)
    WriteLn('-- Subset --');
    IF s^.Subset(s1) THEN WriteLn('s is a subset from s1')
72  ELSE WriteLn('s is not a subset from s1');
    s^.Add('TTTTTT');
74  WriteLn('Added Element TTTTTT to s (Not in s1)');
    IF s^.Subset(s1) THEN WriteLn('s is a subset from s1')
76  ELSE WriteLn('s is not a subset from s1');
    WriteLn;

78  Dispose(s, Done);
```

```
80  Dispose(s1, Done);
    Dispose(s2, Done);
82  Dispose(s3, Done);
END.
```

SOSTest.pas



```
C:\Users\andir\Google Drive\Hagenberg\2. Semester\AUD2\Uebung\Uebung 8>SOSTest.exe
=== SOSTest ===

Cardinality s1: 5
Cardinality s2: 5

-- Union --
Hallo ich bin eine Unit Hallo its me bin Unit
-- Removed Hallo --
ich bin eine Unit Hallo its me bin Unit

== ERROR ADD: Full! ==

== Union Error ==
Too big
Cardinality m1: 5 m2: 10- MAX size: 10

-- Difference --
ich eine

-- Intersection --
Hallo bin Unit

-- Subset --
s is a subset from s1
Added Element TTTTTT to s (Not in s1)
s is not a subset from s1

C:\Users\andir\Google Drive\Hagenberg\2. Semester\AUD2\Uebung\Uebung 8>
```

Abbildung 1: Console Output SOSTest

Aufgabe 2

Lösungsidee

Die vererbten Methoden bzw. Funktionen werden teilweise überschrieben je nachdem ob neue Datentypen verwendet werden oder nicht. Die Mengenoperationen funktionieren fast gleich mit der Ausnahme dass das Feld "counters" berücksichtigt wird. Die Operationen bei Sack werden nicht mehr als "Virtual" bezeichnet da vom Objekt "Sack" nicht mehr abgeleitet wird.

Testfälle

Das ausführen der Operationen soll genau gleich funktionieren und dieselben Fehlermeldungen anzeigen wie beim SOS Test. Durchgeführt werden dieselben Operationen.

```

1  (* ===== *)
2  (* --          Sack TEST 27.05.17          -- *)
3  (* ===== *)

5  PROGRAM SackTest;
6      USES SOSU;

7
8      VAR
9          s, s1, s2, s3: SackPtr;
10 BEGIN
11     WriteLn('=== SackTest ===');

12
13     New(s, Init);
14     New(s1, Init);
15     New(s2, Init);

16
17     s^.Add('Hallo');
18     s^.Add('ich');

19
20     s1^.Add('Hallo');
21     s1^.Add('ich');
22     s1^.Add('bin');
23     s1^.Add('eine');
24     s1^.Add('Unit');

25
26     s2^.Add('Hallo');
27     s2^.Add('me');
28     s2^.Add('bin');
29     s2^.Add('Unit');

30
31     (* Cardinality Test *)
32     WriteLn;
33     WriteLn('Cardinality s1: ', s1^.Cardinality);
34     WriteLn('Cardinality s2: ', s2^.Cardinality);

```

```
35 WriteLn;

37 (* Successful union *)
WriteLn('-- Union --');
39 s3 := s1^.Union(s2);
s3^.Print();
41 s3^.Remove('Hallo');
WriteLn('-- Removed Hallo --');
43 s3^.Print();

45 (* unsuccessful union, s3 gets overwritten with new object -> no output *)
WriteLn;
47 s2^.Add('T');
s2^.Add('T2');
49 s2^.Add('T3');
s2^.Add('T4');
51 s2^.Add('T5');
s2^.Add('T6');
53 s3 := s1^.Union(s2);
s3^.Print();

55 (* Difference Test *)
WriteLn('-- Difference --');
57 s3 := s1^.Difference(s2);
s3^.Print();
59 WriteLn;

61 (* Intersection Test *)
WriteLn('-- Intersection --');
63 s3 := s1^.Intersection(s2);
s3^.Print();
65 WriteLn;

67 (* Subset Test, first true, second false *)
WriteLn('-- Subset --');
69 IF s^.Subset(s1) THEN WriteLn('s is a subset from s1')
71 ELSE WriteLn('s is not a subset from s1');
s^.Add('TTTTTT');
73 WriteLn('Added Element TTTTTT to s (Not in s1)');
IF s^.Subset(s1) THEN WriteLn('s is a subset from s1')
75 ELSE WriteLn('s is not a subset from s1');
WriteLn;

77 Dispose(s, Done);
79 Dispose(s1, Done);
Dispose(s2, Done);
81 Dispose(s3, Done);
END.
```

SackTest.pas

Abbildung 2: Console Output SackTest