

<input type="checkbox"/> Gr. 1, Dr. G. Kronberger	Name <u>Daniel Kreuzer</u>	Aufwand in h <u>11</u>
<input type="checkbox"/> Gr. 2, Dr. H. Gruber		
<input type="checkbox"/> Gr. 3, Dr. D. Auer	Punkte <u>23</u>	Kurzzeichen Tutor / Übungsleiter <u>JS</u> / _____

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.

1. Zeichenketten-Mengen als Klasse

1.1. Lösungsidee

- Die Klasse im Bsp1 besteht aus dem OBJECT SOS, dieses setzt sich aus den PROTECTED Elemente, dem Array mit den gespeicherten STRING Elementen und dem Zähler für die Anzahl der Elemente und PUBLIC Methoden zusammen
- Der CONSTRUCTOR initialisiert den Array und setzt den Zähler auf 0
- Der DESTRUCTOR hat in dieser Aufgabenstellung keine weiteren Aufgaben
- EMPTY überprüft ob der Zähler n gleich Null ist
- CARDINALITY gibt den Zähler n wieder
- ADD fügt einen neuen Eintrag in den Array hinzu und überprüft ob dieser schon enthalten ist. Dadurch wird sichergestellt, dass keine Werte doppelt vorkommen
- REMOVE entfernt einen bestimmten Eintrag.
- CONTAINS sucht einen bestimmten Eintrag und gibt zurück ob dieser vorhanden ist
- UNION fügt zwei Mengen zu einer neuen Menge zusammen
- INTERSECTION gibt die Überschneidungsmenge zweier Mengen wieder
- DIFFERENCE gibt die erste Menge ohne die zweite Menge wieder
- SUBSET überprüft ob eine Menge in der anderen enthalten ist
- GETSET gibt den Array wieder
- PRINT gibt die gespeicherten Elemente in der Konsole aus
- Die Methoden im PUBLIC Bereich sind auf VIRTUAL gesetzt um diese in weiterer Folge in Subklassen weiter bearbeiten zu können

Wie ??

1.2. Quelltext

1.2.1. Quelltext Klasse SOS (Als Bsp1 Abgegeben)

```

UNIT Bsp1;

INTERFACE

CONST
    MaxSize = 100;

TYPE
    wordSet = ARRAY [1..MaxSize] OF STRING;

    SOSPtr = ^SOS;
    SOS = OBJECT
        PROTECTED
            elements: wordSet;
            n: INTEGER;
        PUBLIC
            CONSTRUCTOR Init;
            DESTRUCTOR Done; VIRTUAL;
            FUNCTION Empty: BOOLEAN; VIRTUAL;
            FUNCTION Cardinality: INTEGER; VIRTUAL;
            PROCEDURE Add(w: STRING); VIRTUAL;
            PROCEDURE Remove(w: STRING); VIRTUAL;
            FUNCTION Contains(w: STRING): BOOLEAN; VIRTUAL;
            FUNCTION Union(set2: SOS): SOS; VIRTUAL;
            FUNCTION Intersection(set2: SOS): SOS; VIRTUAL;
            FUNCTION Difference(set2: SOS): SOS; VIRTUAL;
            FUNCTION Subset(set2: SOS): BOOLEAN; VIRTUAL;
            FUNCTION GetSet: wordSet; VIRTUAL;
            PROCEDURE Print; VIRTUAL;
    END;

IMPLEMENTATION

FUNCTION SOS.getSet: wordSet;
BEGIN
    getSet := elements;
END;

CONSTRUCTOR SOS.Init;
VAR
    i: INTEGER;
BEGIN
    (* Init n *)
    n := 0;
    (* Init wordSet *)
    FOR i := 1 TO MaxSize DO
    BEGIN
        elements[i] := '';
    END;
END;

DESTRUCTOR SOS.Done;
BEGIN
    (* nothing to do *)
END;

```

```

FUNCTION SOS.Empty: BOOLEAN;
BEGIN
    Empty := (n = 0);
END;

```

```

FUNCTION SOS.Contains (w: STRING): BOOLEAN;
VAR
    i: INTEGER;

```

```

BEGIN
    Contains := FALSE;
    FOR i := 1 TO maxSize DO
    BEGIN
        IF w = elements[i] THEN
            Contains := TRUE;
        END;
    END;
END;

```

unnötig wenn leer, liefert außerdem true wenn
letztes hinzugefügtes Wort gelöscht wurde ...

Schlechter Stil : Besser While w = elements[i]
AND i ≤ n ...
-0,5P

```

FUNCTION SOS.Cardinality: INTEGER;
BEGIN
    Cardinality := n;
END;

```

```

PROCEDURE SOS.Add (w: STRING);
BEGIN
    IF (n+1) > maxSize THEN
        WriteLn('Element ', w, ' not added, wordSet full!')
    ELSE
        BEGIN
            IF Contains(w) THEN
                WriteLn('Set contains ', w, ' already')
            ELSE
                BEGIN
                    Inc(n);
                    elements[n] := w;
                END;
            END;
        END;
END;

```

```

PROCEDURE SOS.Remove (w: STRING);
VAR
  i, pos: INTEGER;
BEGIN
  IF Empty THEN
    WriteLn('WordSet is empty!!')
  ELSE
    BEGIN
      pos := 0;
      FOR i := 1 TO maxSize DO
        BEGIN
          IF elements[i] = w THEN pos := i;
        END;
        IF pos <> 0 THEN
          BEGIN
            Dec(n);
            FOR i := pos TO maxSize DO
              BEGIN
                IF (i+1) > maxSize THEN
                  elements[i] := ''
                ELSE
                  elements[i] := elements[i + 1];
                END;
              END
            ELSE
              WriteLn('Element ', w, ' not removed');
            END;
          END;
        END;
      END;
    END;
  END;

```

n siehe oben

```

FUNCTION SOS.Union (set2: SOS): SOS;
VAR
  wordSetNew: SOS;
  i: INTEGER;
  setArr: wordSet;
BEGIN
  wordSetNew.Init;
  setArr := set2.GetSet;

  FOR i := 1 TO SELF.n DO
    BEGIN
      wordSetNew.Add(SELF.elements[i]);
    END;
  END;

  FOR i := 1 TO set2.n DO
    BEGIN
      wordSetNew.Add(setArr[i]);
    END;
  END;

  Union := wordSetNew;
END;

```




```

FUNCTION SOS.Intersection(set2: SOS): SOS;
VAR
    wordSetNew: SOS;
    i,j: INTEGER;
    setArr: wordSet;
BEGIN
    wordSetNew.Init;
    setArr := set2.GetSet;

    FOR i := 1 TO SELF.n DO
    BEGIN
        FOR j := 1 TO set2.n DO
        BEGIN
            IF SELF.elements[i] = setArr[j] THEN
                wordSetNew.Add(setArr[j]);
            END;
        END;

    Intersection := wordSetNew;
END;

```




```

FUNCTION SOS.Difference(set2: SOS): SOS;
VAR
    wordSetNew: SOS;
    i: INTEGER;
BEGIN
    wordSetNew.Init;

    wordSetNew := SELF;
    FOR i := 1 TO set2.n DO
    BEGIN
        IF wordSetNew.Contains(set2.elements[i]) THEN
            wordSetNew.Remove(set2.elements[i]);
        END;

    Difference := wordSetNew;
END;


```




```

FUNCTION SOS.Subset(set2: SOS): BOOLEAN;
VAR
    contain: BOOLEAN;
    i: INTEGER;
BEGIN
    contain := TRUE;
    FOR i := 1 TO set2.n DO
    BEGIN
        IF NOT SELF.Contains(set2.elements[i]) THEN
            contain := FALSE;
        END;
    Subset := contain;
END;

```



```
PROCEDURE SOS.Print;  
VAR  
  i: INTEGER;  
BEGIN  
  FOR i := 1 TO n DO  
    BEGIN  
      WriteLn(' - ', elements[i]);  
    END;  
  END;  
BEGIN  
END.
```



1.2.2. Quelltext Bsp1_Test

```
PROGRAM Bsp1_Test;

USES
  Bsp1;
VAR
  Set1: SOS;
  Set2: SOS;
  Set3: SOSPtr;
  Set4: SOSPtr;
  SOSNew: SOS;
BEGIN
  WriteLn('----    Test Bsp 1 - Set of Strings    ----');
  WriteLn;

  WriteLn('- Initialise dynamic and static SOS');
  Set1.Init;
  Set2.Init;
  New(Set3, Init);
  New(Set4, Init);
  WriteLn(' Completed!');
  WriteLn;

  WriteLn('- Test Empty and Cardinality');
  WriteLn(' Set1:');
  WriteLn(' Empty: ', Set1.Empty);
  WriteLn(' Cardinality: ', Set1.Cardinality);
  WriteLn(' Set2:');
  WriteLn(' Empty: ', Set2.Empty);
  WriteLn(' Cardinality: ', Set2.Cardinality);
  WriteLn(' Set3:');
  WriteLn(' Empty: ', Set3^.Empty);
  WriteLn(' Cardinality: ', Set3^.Cardinality);
  WriteLn(' Set4:');
  WriteLn(' Empty: ', Set4^.Empty);
  WriteLn(' Cardinality: ', Set4^.Cardinality);
  WriteLn;
```



```

WriteLn('- Add Test-String');
Set1.Add('Hallo');
(* Test double insert *)
Set1.Add('Hallo');
Set1.Add('Willkommen');
Set1.Add('ADE');
Set1.Add('Blabla');
Set1.Add('blabla');
Set1.Add('Stop');
Set1.Add('Begin');
Set2.Add('Hallo');
(* Test double insert *)
Set2.Add('Hallo');
Set2.Add('Willkommen');
Set2.Add('ADE');
Set2.Add('Blabla');
Set2.Add('balablaed');
Set2.Add('Stop it');
Set2.Add('Begin now!');
Set3^.Add('Hallo');
(* Test double insert *)
Set3^.Add('Hallo');
Set3^.Add('Willkommen');
Set3^.Add('ADE');
Set3^.Add('Blabla');
Set3^.Add('blabla');
Set3^.Add('Stop');
Set3^.Add('Begin');
Set4^.Add('Hallo');
(* Test double insert *)
Set4^.Add('Hallo');
Set4^.Add('Willkommen');
Set4^.Add('ADE');
Set4^.Add('Blabla');
Set4^.Add('balablaed');
Set4^.Add('Stop it');
Set4^.Add('Begin now!');
WriteLn;

WriteLn('- Test Print');
Set1.Print;
WriteLn;
Set2.Print;
WriteLn;
Set3^.Print;
WriteLn;
Set4^.Print;
WriteLn;

WriteLn('- Test Empty and Cardinality');
WriteLn(' Set1:');
WriteLn(' Empty: ', Set1.Empty);
WriteLn(' Cardinality: ', Set1.Cardinality);
WriteLn(' Set2:');
WriteLn(' Empty: ', Set2.Empty);
WriteLn(' Cardinality: ', Set2.Cardinality);
WriteLn(' Set3:');
WriteLn(' Empty: ', Set3^.Empty);
WriteLn(' Cardinality: ', Set3^.Cardinality);
WriteLn(' Set4:');
WriteLn(' Empty: ', Set4^.Empty);
WriteLn(' Cardinality: ', Set4^.Cardinality);
WriteLn;

```

```
WriteLn('- Test Contains');
  WriteLn(' Set 1 contains Hallo: ', Set1.Contains('Hallo'));
  WriteLn(' Set 2 contains Hallo: ', Set2.Contains('Hallo'));
  WriteLn(' Set 3 contains Hallo: ', Set3^.Contains('Hallo'));
  WriteLn(' Set 4 contains Hallo: ', Set4^.Contains('Hallo'));
WriteLn;

WriteLn('- Test Union');
  WriteLn(' Union of Set1 and Set2');
  SOSNew.Init;
  SOSNew := Set1.Union(Set2);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  SOSNew.Done;
  WriteLn;
  WriteLn(' Union of Set3 and Set4');
  SOSNew.Init;
  SOSNew := Set3^.Union(Set4^);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  WriteLn;
  SOSNew.Print;
  SOSNew.Done;
WriteLn;

WriteLn('- Test Intersection');
  WriteLn(' Intersection of Set1 and Set2');
  SOSNew.Init;
  SOSNew := Set1.Intersection(Set2);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  SOSNew.Done;
  WriteLn;
  WriteLn(' Intersection of Set3 and Set4');
  SOSNew.Init;
  SOSNew := Set3^.Intersection(Set4^);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  WriteLn;
  SOSNew.Print;
  SOSNew.Done;
WriteLn;

WriteLn('- Test Difference');
  WriteLn(' Difference of Set1 and Set2');
  SOSNew.Init;
  SOSNew := Set1.Difference(Set2);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  SOSNew.Done;
  WriteLn;
  WriteLn(' Difference of Set3 and Set4');
  SOSNew.Init;
  SOSNew := Set3^.Difference(Set4^);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  WriteLn;
  SOSNew.Print;
  SOSNew.Done;
WriteLn;
```

```

WriteLn('- Test Subset');
  WriteLn(' Subset of Set1 and Set1');
  WriteLn(' Is Set1 Subset of Set1?: ', Set1.Subset(Set1), ' expected:
                                         TRUE');

  WriteLn;
  WriteLn(' Subset of Set3 and Set4');
  WriteLn(' Is Set4 Subset of Set3?: ', Set3^.Subset(Set4^), ' expected:
                                         FALSE');

WriteLn;

WriteLn('- Test Remove');
  WriteLn(' Remove all elements in Set1');
  Set1.Remove('Hallo');
  (* Test double remove *)
  Set1.Remove('Hallo');
  Set1.Remove('Willkommen');
  Set1.Remove('ADE');
  Set1.Remove('Blabla');
  Set1.Remove('blabla');
  Set1.Remove('Stop');
  Set1.Remove('Begin');
  WriteLn(' Is Empty after remove?: ', Set1.Empty);
  WriteLn;

  WriteLn(' Remove all elements in Set3');
  Set3^.Remove('Hallo');
  (* Test double insert *)
  Set3^.Remove('Hallo');
  Set3^.Remove('Willkommen');
  Set3^.Remove('ADE');
  Set3^.Remove('Blabla');
  Set3^.Remove('blabla');
  Set3^.Remove('Stop');
  Set3^.Remove('Begin');
  WriteLn(' Is Empty after remove?: ', Set3^.Empty);
  Set3^.Print;
WriteLn;

WriteLn('- Set done');
  Set1.Done;
  Set2.Done;
  Dispose(Set3, Done);
  Dispose(Set4, Done);
  WriteLn(' Completed!');
WriteLn;

```

END.

1.3. Testfälle

----- Test Bsp 1 - Set of Strings -----

- Initialise dynamic and static SOS
Completed!

- Test Empty and Cardinality

 - Set1:

 - Empty: TRUE

 - Cardinality: 0

 - Set2:

 - Empty: TRUE

 - Cardinality: 0

 - Set3:

 - Empty: TRUE

 - Cardinality: 0

 - Set4:

 - Empty: TRUE

 - Cardinality: 0

- Add Test-String

 - Set contains Hallo already

 - Set contains Hallo already

 - Set contains Hallo already

 - Set contains Hallo already

- Test Print

-Hallo
-Willkommen
-ADE
-Blabla
-blabla
-Stop
-Begin

-Hallo
-Willkommen
-ADE
-Blabla
-balablaed
-Stop it
-Begin now!

-Hallo
-Willkommen
-ADE
-Blabla
-blabla
-Stop
-Begin

-Hallo
-Willkommen
-ADE
-Blabla
-balablaed
-Stop it
-Begin now!

- Test Empty and Cardinality

Set1:
Empty: FALSE
Cardinality: 7
Set2:
Empty: FALSE
Cardinality: 7
Set3:
Empty: FALSE
Cardinality: 7
Set4:
Empty: FALSE
Cardinality: 7

- Test Contains

- Set 1 contains Hallo: TRUE
 - Set 2 contains Hallo: TRUE
 - Set 3 contains Hallo: TRUE
 - Set 4 contains Hallo: TRUE

- Test Union

- Union of Set1 and Set2
 - Set contains Hallo already
 - Set contains Willkommen already
 - Set contains ADE already
 - Set contains Blabla already
 - Cardinality of New Set: 10

- Union of Set3 and Set4
 - Set contains Hallo already
 - Set contains Willkommen already
 - Set contains ADE already
 - Set contains Blabla already
 - Cardinality of New Set: 10

- Hallo
 - Willkommen
 - ADE
 - Blabla
 - blabla
 - Stop
 - Begin
 - balablaed
 - Stop it
 - Begin now!

- Test Intersection

- Intersection of Set1 and Set2
 - Cardinality of New Set: 4

- Intersection of Set3 and Set4
 - Cardinality of New Set: 4

- Hallo
 - Willkommen
 - ADE
 - Blabla

- Test Difference
 - Difference of Set1 and Set2
 - Cardinality of New Set: 3

 - Difference of Set3 and Set4
 - Cardinality of New Set: 3

 - blabla
 - Stop
 - Begin
- Test Subset
 - Subset of Set1 and Set1
 - Is Set1 Subset of Set1?: TRUE expected: TRUE

 - Subset of Set3 and Set4
 - Is Set4 Subset of Set3?: FALSE expected: FALSE
- Test Remove
 - Remove all elements in Set1
 - Element Hallo not removed
 - Is Empty after remove?: TRUE

 - Remove all elements in Set3
 - Element Hallo not removed
 - Is Empty after remove?: TRUE
- Set done
 - Completed!

Drücken Sie eine beliebige Taste . . .



2. Säcke

2.1. Lösungsidee

- Die Klasse im Bsp2 erweitert die Klasse im Bsp1 mit dem OBJECT BOS ✓
- Das OBJECT BOS setzt sich aus dem Zähl-Array im PRIVATE Bereich und Methoden im PUBLIC Bereich zusammen ✓
- Der Zähl-Array ermöglicht, Elemente öfter in die Menge einzuspeichern, ohne dass diese im Array mit den STRING Elementen doppelt vorkommen ✓
- Der CONSTRUCTOR übernimmt die Funktionen des CONSTRUCTORS in der Basisklasse und initialisiert zusätzlich noch den Zähl-Array ✓
- Der DESTROCTOR übernimmt die Funktionen des DESTRUCTORS in der Basisklasse und führt keine weiteren Schritte durch
- Es wurden nicht alle Methoden der Basisklasse abgeändert, da manche ihren keine Änderung in BOS benötigen. Werden andere Datentypen im Methodenkopf verwendet, wie es bei den Mengenoperationen UNION, INTERSECTOIN, DIFFERENCE und SUBSET der Fall ist, wird das Schlüsselwort OVERLOAD verwendet. *noja,*
- CARDINALITY zählt in BOS die Einträge im Zähl-Array zusammen ✓
- ADD übernimmt die Funktion der Basisklasse und erhöht den Zähl-Array an der richtigen Stelle ✓
- REMOVE entfernt einen Eintrag, in BOS ist zu beachten das ein Eintrag in dem Array mit den STRING Elementen nur dann gelöscht werden darf, wenn der Eintrag den Wert Eins im Zähl-Array hat ✓
- UNION übernimmt die Funktionen der Basisklasse. Eine Lokale Variable übernimmt das Ergebnis und setzt ein BOS Ergebnis zusammen. Da in der Basisklasse kein Zähl-Array vorkommt wird dieser noch zusätzlich aktualisiert. Dazu wird die Anzahl der Einträge in Menge1 und Menge2 addiert ✓
- INTERSECTION übernimmt die Funktionen der Basisklasse. Wie bei UNION wird der Zähl-Array noch aktualisiert ✓
- DIFFERENCE übernimmt die Funktionen der Basisklasse. Hier wird der Zähl-Array nur an den Zähl-Array der ersten Menge angepasst ✓
- SUBSET übernimmt die Funktionen der Basisklasse. Zusätzlich muss noch überprüft werden ob die Anzahl in beiden Mengen noch übereinstimmt ✓
- GETCNT übergibt den Zähl-Array ✓
- PRINT gibt alle Elemente so oft aus, wie sie vorkommen ✓

2.2. Quelltext

2.2.1. Quelltext Klasse BOS (Als Bsp2 abgegeben)

```

UNIT Bsp2;

INTERFACE

USES
    Bsp1;

TYPE
    counterArr = ARRAY [1..MaxSize] OF INTEGER;

    BOSPtr = ^BOS;
    BOS = OBJECT(SOS)
        PRIVATE
            counters: counterArr;
        PUBLIC
            CONSTRUCTOR Init;
            DESTRUCTOR Done; VIRTUAL;
            FUNCTION Cardinality: INTEGER; VIRTUAL;
            PROCEDURE Add(w: STRING); VIRTUAL;
            PROCEDURE Remove(w: STRING); VIRTUAL;
            FUNCTION Union(set2: BOS): BOS; VIRTUAL; OVERLOAD;
            FUNCTION Intersection(set2: BOS): BOS; VIRTUAL; OVERLOAD;
            FUNCTION Difference(set2: BOS): BOS; VIRTUAL; OVERLOAD;
            FUNCTION Subset(set2: BOS): BOOLEAN; VIRTUAL; OVERLOAD;
            FUNCTION GetCnt: counterArr; VIRTUAL;
            PROCEDURE Print; Virtual;
    END;

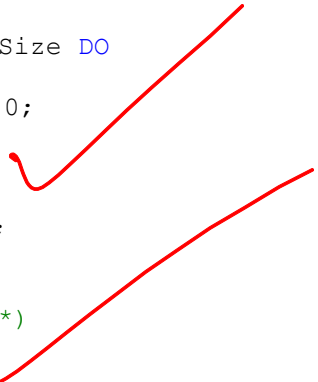
IMPLEMENTATION

FUNCTION BOS.GetCnt: counterArr;
BEGIN
    GetCnt := counters;
END;

CONSTRUCTOR BOS.Init;
VAR
    i: INTEGER;
BEGIN
    INHERITED Init;
    FOR i := 1 TO MaxSize DO
        BEGIN
            counters[i] := 0;
        END;
    END;

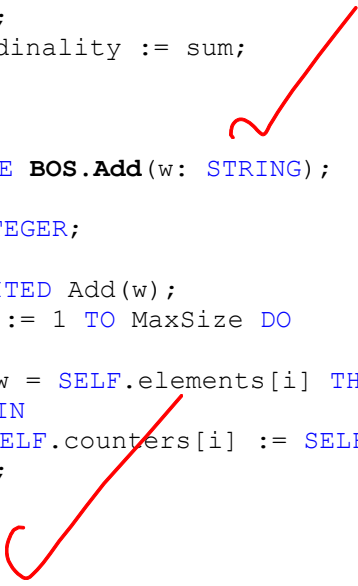
DESTRUCTOR BOS.Done;
BEGIN
    INHERITED Done;
    (* nothing to do *)
END;

```



```
FUNCTION BOS.Cardinality: INTEGER;
VAR
  i, sum: INTEGER;
BEGIN
  IF Empty THEN
    Cardinality := 0
  ELSE
    BEGIN
      sum := 0;
      FOR i := 1 TO MaxSize DO
        BEGIN
          sum := sum + counters[i];
        END;
      Cardinality := sum;
    END;
  END;
END;

PROCEDURE BOS.Add(w: STRING);
VAR
  i: INTEGER;
BEGIN
  INHERITED Add(w);
  FOR i := 1 TO MaxSize DO
    BEGIN
      IF w = SELF.elements[i] THEN
        BEGIN
          SELF.counters[i] := SELF.counters[i] + 1;
        END;
      END;
    END;
  END;
END;
```



```

PROCEDURE BOS.Remove (w: STRING);
VAR
  i, pos: INTEGER;
BEGIN
  IF Empty THEN
    WriteLn('WordSet is empty!!')
  ELSE
    BEGIN
      pos := 0;
      FOR i := 1 TO maxSize DO
        BEGIN
          IF elements[i] = w THEN pos := i;
        END;
        IF pos <> 0 THEN
          BEGIN
            IF counters[pos] = 1 THEN
              BEGIN
                Dec(n);
                FOR i := pos TO maxSize DO
                  BEGIN
                    IF (i+1) > maxSize THEN
                      counters[i] := 0;
                    ELSE
                      counters[i] := counters[i + 1];
                    END;
                    FOR i := pos TO maxSize DO
                      BEGIN
                        IF (i+1) > maxSize THEN
                          elements[i] := '';
                        ELSE
                          elements[i] := elements[i + 1];
                        END;
                      END;
                    END;
                  ELSE
                    BEGIN
                      counters[pos] := counters[pos] - 1;
                    END;
                  END;
                ELSE
                  WriteLn('Element ', w, ' not removed');
                END;
              END;
            END;
          END;
        END;
      END;
    END;
  END;

```

↑
a .

```

FUNCTION BOS.Union(set2: BOS): BOS;
VAR
  bosSetNew: BOS;
  sosTemp: SOS;
  i, j: INTEGER;
BEGIN
  sosTemp.Init;
  bosSetNew.Init;
  sosTemp := INHERITED Union(set2);
  bosSetNew.elements := sosTemp.GetSet;
  bosSetNew.n := sosTemp.Cardinality;

  FOR i := 1 TO bosSetNew.n DO
  BEGIN
    FOR j := 1 TO SELF.n DO
    BEGIN
      IF SELF.elements[j] = bosSetNew.elements[i] THEN
        bosSetNew.counters[i] := bosSetNew.counters[i] +
                                SELF.counters[j];
    END;
    FOR j := 1 TO set2.n DO
    BEGIN
      IF set2.elements[j] = bosSetNew.elements[i] THEN
        bosSetNew.counters[i] := bosSetNew.counters[i] +
                                set2.counters[j];
    END;
  END;
  Union := bosSetNew;
END;

```

Sollte privat sein... - 0,5P

✓

```

FUNCTION BOS.Intersection(set2: BOS): BOS;
VAR
  bosSetNew: BOS;
  sosTemp: SOS;
  i, j: INTEGER;
BEGIN
  sosTemp.Init;
  bosSetNew.Init;
  sosTemp := INHERITED Intersection(set2);
  bosSetNew.elements := sosTemp.GetSet;
  bosSetNew.n := sosTemp.Cardinality;

  FOR i := 1 TO bosSetNew.n DO
  BEGIN
    FOR j := 1 TO SELF.n DO
    BEGIN
      IF SELF.elements[j] = bosSetNew.elements[i] THEN
        bosSetNew.counters[i] := bosSetNew.counters[i] +
                                SELF.counters[j];
    END;
    FOR j := 1 TO set2.n DO
    BEGIN
      IF set2.elements[j] = bosSetNew.elements[i] THEN
        bosSetNew.counters[i] := bosSetNew.counters[i] +
                                set2.counters[j];
    END;
  END;
  Intersection := bosSetNew;
END;

```

✓

```

FUNCTION BOS.Difference(set2: BOS): BOS;
VAR
    wordSetNew: BOS;
    i: INTEGER;
BEGIN
    wordSetNew.Init;

    wordSetNew := SELF;
    FOR i := 1 TO set2.n DO
    BEGIN
        IF wordSetNew.Contains(set2.elements[i]) THEN
            wordSetNew.Remove(set2.elements[i]);
        END;
    END;

    Difference := wordSetNew;
END;

FUNCTION BOS.Subset(set2: BOS): BOOLEAN;
VAR
    checks, verify: BOOLEAN;
    i, j: INTEGER;
BEGIN
    checks := INHERITED.Subset(set2);
    IF checks THEN
    BEGIN
        verify := TRUE;
        FOR i := 1 TO set2.n DO
        BEGIN
            FOR j := 1 TO SELF.n DO
            BEGIN
                IF SELF.elements[j] = Set2.elements[i] THEN
                BEGIN
                    IF SELF.counters[j] <> Set2.counters[i] THEN
                        verify := FALSE;
                    END;
                END;
            END;
        END;
        Subset := verify;
    END
    ELSE
        Subset := FALSE;
    END;
END;

PROCEDURE BOS.Print;
VAR
    i, j: INTEGER;
BEGIN
    FOR i := 1 TO n DO
    BEGIN
        FOR j := 1 TO counters[i] DO
            WriteLn('-', elements[i]);
        END;
    END;
END;

BEGIN
END.

```

2.2.2. Quelltext Bsp2_Test

```
PROGRAM Bsp2_Test;

USES
  Bsp1, Bsp2;
VAR
  Set1: BOS;
  Set2: BOS;
  Set3: BOSPtr;
  Set4: BOSPtr;
  SOSNew: BOS;
BEGIN
  WriteLn('----    Test Bsp 2 - Bag of Strings    ----');
  WriteLn;

  WriteLn('- Initialise dynamic and static BOS');
  Set1.Init;
  Set2.Init;
  New(Set3, Init);
  New(Set4, Init);
  WriteLn(' Completed!');
  WriteLn;

  WriteLn('- Test Empty and Cardinality');
  WriteLn(' Set1:');
  WriteLn(' Empty: ', Set1.Empty);
  WriteLn(' Cardinality: ', Set1.Cardinality);
  WriteLn(' Set2:');
  WriteLn(' Empty: ', Set2.Empty);
  WriteLn(' Cardinality: ', Set2.Cardinality);
  WriteLn(' Set3:');
  WriteLn(' Empty: ', Set3^.Empty);
  WriteLn(' Cardinality: ', Set3^.Cardinality);
  WriteLn(' Set4:');
  WriteLn(' Empty: ', Set4^.Empty);
  WriteLn(' Cardinality: ', Set4^.Cardinality);
  WriteLn;
```

```

WriteLn('- Add Test-String');
Set1.Add('Hallo');
(* Test double insert *)
Set1.Add('Hallo');
Set1.Add('Willkommen');
Set1.Add('ADE');
Set1.Add('Blabla');
Set1.Add('blabla');
Set1.Add('Stop');
Set1.Add('Begin');
Set2.Add('Hallo');
(* Test double insert *)
Set2.Add('Hallo');
Set2.Add('Willkommen');
Set2.Add('ADE');
Set2.Add('Blabla');
Set2.Add('balablaed');
Set2.Add('Stop it');
Set2.Add('Begin now!');
Set3^.Add('Hallo');
(* Test double insert *)
Set3^.Add('Hallo');
Set3^.Add('Willkommen');
Set3^.Add('ADE');
Set3^.Add('Blabla');
Set3^.Add('blabla');
Set3^.Add('Stop');
Set3^.Add('Begin');
Set4^.Add('Hallo');
(* Test double insert *)
Set4^.Add('Hallo');
Set4^.Add('Willkommen');
Set4^.Add('ADE');
Set4^.Add('Blabla');
Set4^.Add('balablaed');
Set4^.Add('Stop it');
Set4^.Add('Begin now!');
WriteLn;

WriteLn('- Test Print');
Set1.Print;
WriteLn;
Set2.Print;
WriteLn;
Set3^.Print;
WriteLn;
Set4^.Print;
WriteLn;

WriteLn('- Test Empty and Cardinality');
WriteLn(' Set1:');
WriteLn(' Empty: ', Set1.Empty);
WriteLn(' Cardinality: ', Set1.Cardinality);
WriteLn(' Set2:');
WriteLn(' Empty: ', Set2.Empty);
WriteLn(' Cardinality: ', Set2.Cardinality);
WriteLn(' Set3:');
WriteLn(' Empty: ', Set3^.Empty);
WriteLn(' Cardinality: ', Set3^.Cardinality);
WriteLn(' Set4:');
WriteLn(' Empty: ', Set4^.Empty);
WriteLn(' Cardinality: ', Set4^.Cardinality);
WriteLn;

```

```
WriteLn('- Test Contains');
  WriteLn(' Set 1 contains Hallo: ', Set1.Contains('Hallo'));
  WriteLn(' Set 2 contains Hallo: ', Set2.Contains('Hallo'));
  WriteLn(' Set 3 contains Hallo: ', Set3^.Contains('Hallo'));
  WriteLn(' Set 4 contains Hallo: ', Set4^.Contains('Hallo'));
WriteLn;

WriteLn('- Test Union');
  WriteLn(' Union of Set1 and Set2');
  SOSNew.Init;
  SOSNew := Set1.Union(Set2);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  SOSNew.Done;
  WriteLn;
  WriteLn(' Union of Set3 and Set4');
  SOSNew.Init;
  SOSNew := Set3^.Union(Set4^);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  WriteLn;
  SOSNew.Print;
  SOSNew.Done;
WriteLn;

WriteLn('- Test Intersection');
  WriteLn(' Intersection of Set1 and Set2');
  SOSNew.Init;
  SOSNew := Set1.Intersection(Set2);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  SOSNew.Done;
  WriteLn;
  WriteLn(' Intersection of Set3 and Set4');
  SOSNew.Init;
  SOSNew := Set3^.Intersection(Set4^);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  WriteLn;
  SOSNew.Print;
  SOSNew.Done;
WriteLn;

WriteLn('- Test Difference');
  WriteLn(' Difference of Set1 and Set2');
  SOSNew.Init;
  SOSNew := Set1.Difference(Set2);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  SOSNew.Done;
  WriteLn;
  WriteLn(' Difference of Set3 and Set4');
  SOSNew.Init;
  SOSNew := Set3^.Difference(Set4^);
  WriteLn(' Cardinality of New Set: ', SOSNew.Cardinality);
  WriteLn;
  SOSNew.Print;
  SOSNew.Done;
WriteLn;
```



```

WriteLn('- Test Subset');
  WriteLn(' Subset of Set1 and Set1');
  WriteLn(' Is Set1 Subset of Set1?: ', Set1.Subset(Set1), ' expected:
                                         TRUE');

  WriteLn;
  WriteLn(' Subset of Set3 and Set4');
  WriteLn(' Is Set4 Subset of Set3?: ', Set3^.Subset(Set4^), ' expected:
                                         FALSE');

WriteLn;

WriteLn('- Test Remove');
  WriteLn(' Remove all elements in Set1');
  Set1.Remove('Hallo');
  (* Test double remove *)
  Set1.Remove('Hallo');
  Set1.Remove('Willkommen');
  Set1.Remove('ADE');
  Set1.Remove('Blabla');
  Set1.Remove('blabla');
  Set1.Remove('Stop');
  Set1.Remove('Begin');
  WriteLn(' Is Empty after remove?: ', Set1.Empty);
  WriteLn;

  WriteLn(' Remove all elements in Set3');
  Set3^.Remove('Hallo');
  (* Test double insert *)
  Set3^.Remove('Hallo');
  Set3^.Remove('Willkommen');
  Set3^.Remove('ADE');
  Set3^.Remove('Blabla');
  Set3^.Remove('blabla');
  Set3^.Remove('Stop');
  Set3^.Remove('Begin');
  WriteLn(' Is Empty after remove?: ', Set3^.Empty);
  Set3^.Print;
  Set1.Print;
WriteLn;

WriteLn('- Set done');
  Set1.Done;
  Set2.Done;
  Dispose(Set3, Done);
  Dispose(Set4, Done);
  WriteLn(' Completed!');
WriteLn;

```

END.

2.3. Testfälle

----- Test Bsp 2 - Bag of Strings -----

- Initialise dynamic and static BOS
Completed!

- Test Empty and Cardinality

Set1:

Empty: TRUE

Cardinality: 0

Set2:

Empty: TRUE

Cardinality: 0

Set3:

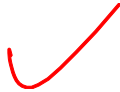
Empty: TRUE

Cardinality: 0

Set4:

Empty: TRUE

Cardinality: 0



- Add Test-String

Set contains Hallo already

Set contains Hallo already

Set contains Hallo already

Set contains Hallo already

- Test Print
 - Hallo
 - Hallo
 - Willkommen
 - ADE
 - Blabla
 - blabla
 - Stop
 - Begin

 - Hallo
 - Hallo
 - Willkommen
 - ADE
 - Blabla
 - balablaed
 - Stop it
 - Begin now!

 - Hallo
 - Hallo
 - Willkommen
 - ADE
 - Blabla
 - blabla
 - Stop
 - Begin

 - Hallo
 - Hallo
 - Willkommen
 - ADE
 - Blabla
 - balablaed
 - Stop it
 - Begin now!
- Test Empty and Cardinality
 - Set1:
 - Empty: FALSE
 - Cardinality: 8
 - Set2:
 - Empty: FALSE
 - Cardinality: 8
 - Set3:
 - Empty: FALSE
 - Cardinality: 8
 - Set4:
 - Empty: FALSE
 - Cardinality: 8

- Test Contains

- Set 1 contains Hallo: TRUE
 - Set 2 contains Hallo: TRUE
 - Set 3 contains Hallo: TRUE
 - Set 4 contains Hallo: TRUE

- Test Union

- Union of Set1 and Set2
 - Set contains Hallo already
 - Set contains Willkommen already
 - Set contains ADE already
 - Set contains Blabla already
 - Cardinality of New Set: 16

- Union of Set3 and Set4
 - Set contains Hallo already
 - Set contains Willkommen already
 - Set contains ADE already
 - Set contains Blabla already
 - Cardinality of New Set: 16

- Hallo
 - Hallo
 - Hallo
 - Hallo
 - Willkommen
 - Willkommen
 - ADE
 - ADE
 - Blabla
 - Blabla
 - blabla
 - Stop
 - Begin
 - balablaed
 - Stop it
 - Begin now!

- Test Intersection

Intersection of Set1 and Set2
Cardinality of New Set: 10

Intersection of Set3 and Set4
Cardinality of New Set: 10

-Hallo
-Hallo
-Hallo
-Hallo
-Willkommen
-Willkommen
-ADE
-ADE
-Blabla
-Blabla

- Test Difference

Difference of Set1 and Set2
Cardinality of New Set: 4

Difference of Set3 and Set4
Cardinality of New Set: 4

-Hallo
-blabla
-Stop
-Begin

- Test Subset

Subset of Set1 and Set1
Is Set1 Subset of Set1?: TRUE expected: TRUE

Subset of Set3 and Set4
Is Set4 Subset of Set3?: FALSE expected: FALSE

- Test Remove

Remove all elements in Set1
Is Empty after remove?: TRUE

Remove all elements in Set3
Is Empty after remove?: TRUE

- Set done

Completed!

Drücken Sie eine beliebige Taste . . .

