# Objektmengen (Sets)

Folgend ist die Dokumentation für die Aufgabenstellung Set angeführt.

## Lösungsidee

Da bereits in der Übung eine Liste implementiert wurde und diese sich nur durch die Eigenschaft dass sie Duplikate erlaubt von einem Set unterscheidet, soll diese Implementierung wiederverwendet werden, wobei eine Prüfung auf Duplikate erfolgen soll.

Wird versucht Duplikate einzufügen soll diese verhindert werden und eine dementsprechende Meldung auf die Konsole ausgegeben werden.

Da von der bereits implementierten Klasse List abgeleitet werden soll, sind auch keine Änderungen an den Klassen der MiniLib notwendig.

Da die Klasse List bereits eine Iterator Implementierung zur Verfügung stellt und diese alle notwendigen Operationen zur Verfügung stellt, mit den verwendeten Node Typen arbeiten kann, sowie mit dessen Variationen, gibt es keine Notwendigkeit einen neuen Iterator für den Set Container zu implementieren.

Da nur die Prüfungen auf Duplikate erfolgen müssen sollte die Implementierung des Set Containers trivial sein.

Die Methoden Intersect(Schniottmenge), Union(Vereinigung) und Complement(A ohne B) könnte wie folgt implementiert werden.

1. void target->Intersect(Set\* source)  
   Die Instanz über die die Methode aufgerufen wird spiegelt das Resultat der Operation wieder
2. void source->Intersect(Set\* target)  
   Die übergebene Set Instanz spiegelt das Resultat der Operation wieder.
3. Set\* util->(Set\* source, Set\* target)  
   Es wird ein neues Set erstellt, welches das Resultat der Operation wiederspiegelt.

Ich würde die Version 3 bevorzugen, jedoch folgt diese Version nicht der Anforderung.

Daher soll Version 1 implementiert werden.

## Implementierung

Folgend sind die Implementierungen für die erste Aufgabenstellung angeführt.

### List.h

Folgend ist die Spezifikation des List Containers angeführt.

### List.cpp

Folgend ist die Implementierung der Spezifikation List.h angeführt.

### Node.h

Folgend ist die Spezifikation der Container Node angeführt.

### Node.cpp

Folgend ist die Implementierung der Spezifikation Node.h angeführt.

### Set.h

Folgend ist die Spezifikation des Set Containers angeführt.

### Set.cpp

Folgend ist die Implementierung der Spezifikation Set.h angeführt.

## Tests

Folgend sind die Tests für die Set Implementierung angeführt.

Es wird hierbei davon ausgegangen das die List Implementierung korrekt funktioniert.

Sie wird in diesen Tests sowieso implizit getestet, bis auf die Möglichkeit Duplikate einzufügen.

Am Ende der Tests werden die Meta-Informationen ausgegeben.

# Objektbehälter (Bag)

Folgend ist die Dokumentation der zweiten Aufgabenstellung angeführt.

## Lösungsidee

Laut Aufgabenstellung soll dieser Container von Set ableiten und dieselben Funktionalitäten zur Verfügung stellen, was an sich zwar nicht in Ordnung ist, da Mengenoperationen auf Mengen angewendet werden sollen und der Bag Container nun mahl keine Menge darstellt, da er Duplikate erlaubt. Daher müssen die Mengenoperationen erneut implementiert und deren Verhalten spezifiziert werden. Dies wird später genauer erläutert.

Um Speicherschonend arbeiten zu können sollen die Duplikate nicht gespeichert sondern lediglich die Anzahl der Werte, die im Container verwaltet werden gespeichert werden. Dies stellt kein Problem dar, da die Duplikate exakt dem gespeicherten Wert entsprechen und sich von diesen unterschieden.

Daher soll die Klasse BagNode eingeführt werden, die von Node ableitet und sich die Anzahl der Werte im Container merkt.

Der Bag Container soll aber trotzdem die Anzahl aller gespeicherten Werte halten obwohl diese nicht die tatsächliche Anzahl der verwalteten Knoten wiederspiegelt. Da aber mittels Iterator über die Elemente iteriert wird und dieser unabhängig von der aktuellen Anzahl und der gehaltenen Größe ist, stellt dies kein Problem dar.

Bei einer Iteration über die Elemente soll nur über die aktuell verwalteten Knoten iteriert werden und der Aufrufer ist verantwortlich sich um die Duplikate, welche über die gehaltene Anzahl in der Klasse BagNode definiert ist, zu kümmern.

Bei der Konstruktion der Bag Container Instanz soll der Anker der gehaltenen Liste ausgetauscht werden und als BagNode definiert werden um Typ Probleme zu vermeiden.

### Spezifikation der Mengenoperationen

Im Folgenden wird die aktuelle Instanz über die die Operationen durchgeführt werden als Target und der übergebene Bag Container als Source bezeichnet.

Um die Performanz zu steigern sollen die Duplikate mittels Arithmetik hinzugefügt oder entfernt werden und nur der aktuell verwaltete Knoten gelöscht werden falls erforderlich. Die Remove Methode würde zwar funktionieren, jedoch währen zu viele Iterationen über die backing List erfolgen müssen, was hierbei nicht notwendig ist.

Das folgend spezifizierte Verhalten der Mengenoperationen betrachtet die Duplikate als wären Sie gültige Elemente einer Menge.

**Schnittmenge:**

Hierbei soll sich diese Operation wie folgt verhalten:

1. Knoten im Source, die mehr Duplikate definieren als der Knoten des Target, sollen die Anzahl der Duplikate des Target Knoten wiederspiegeln.
2. Knoten im Source, die weniger Duplikate definieren als die Knoten im Target, sollen so verbleiben und nicht geändert werden.
3. Knoten die im Target vorhanden sind aber nicht im Source sollen vollständig gelöscht werden.

**Vereinigung:**

Hierbei soll sich diese Operation wie folgt verhalten:

1. Knoten die im Source vorhanden sind aber nicht im Target sollen mit Ihren Duplikaten übernommen werden.
2. Knoten die Im Target vorhanden sind aber weniger Duplikate definieren, sollen die Anzahl der Duplikate des Knoten im Source wiederspiegeln.
3. Knoten die im Target vorhanden sind und mehr Duplikate definieren als die Knoten im Source sollen so verbleiben wie sie sind.

**Komplement:**

Hierbei soll sich die Operation wie folgt verhalten.

1. Knoten die im Target und Source vorhanden sind, sollen vollständig gelöscht werden.
2. Knoten die Target vorhanden sind und weniger Knoten definieren als die Knoten des Source, sollen vollständig entfernt werden.
3. Knoten die Im Target vorhanden sind und mehr Knoten definieren als die Knoten im Source sollen die Differenz: sourceNode.count – targetNode.count; defnieren.