Erweitern Sie Ihre Implementierung des **ADS_set** um die Methode

```
std::pair<bool,key_type> z() const;
```

Diese soll den zweitgrößten Wert liefern, der mit **erase()** oder **clear()** aus dem **ADS_set** erfolgreich gelöscht wurde. Sollten zum Zeitpunkt des Aufrufs von **z()** bereits zumindest zwei unterschiedliche Werte mit **erase()** oder **clear()** gelöscht worden sein, so ist **z().first** gleich **true** und **z().second** enthält den zweitgrößten gelöschten Wert. Andernfalls ist **z().first** gleich **false** und **z().second** ist nicht definiert (d.h. es ist jeder beliebige Wert zulässig). Bitte beachten: Der zweitgrößte Wert ist kleiner als der größte Wert.

Zu diesem Zweck sind im **ADS_set** zusätzlich der größte und der zweitgrößte gelöschte Wert zu verwalten („Historie“) und bei den Löschoperationen **erase()** und **clear()** gegebenenfalls entsprechend anzupassen. Achtung: es sind nur „externe“ Aufrufe der Funktionen **erase()** und **clear()** zu berücksichtigen. Externe Aufrufe sind Aufrufe von außerhalb des **ADS_set** (also hier vom Unit-Test), nicht hingegen Aufrufe dieser Funktionen durch andere Funktionen des **ADS_set** („interne“ Aufrufe).

Beim Kopierkonstruktor und Kopierzuweisungsoperator

```
ADS_set(const ADS_set &other);
ADS_set &operator=(const ADS_set &other);
```

wird die Historie aus **other** übernommen.

Beim Initializer-List-Zuweisungsoperator

```
ADS_set &operator=(std::initializer_list<key_type> ilist);
```

wird die Historie in den Anfangszustand gesetzt („noch keine Werte gelöscht“).

Bei **swap**-Operationen wird die Historie der beiden **ADS_set**s getauscht.

Bei den Vergleichsoperationen

```
friend bool operator==(const ADS_set &lhs, const ADS_set &rhs);
friend bool operator!=(const ADS_set &lhs, const ADS_set &rhs);
```

wird die Historie ignoriert.

Die Zeit- und Speicherkomplexität von **z()** muss O(1) sein. Die Zeit- und Speicherkomplexität aller übrigen Funktionen (inklusive Methoden) müssen unverändert (spezifikationskonform) bleiben. Für die Implementierung und die Verwendung der STL gelten dieselben Regeln wie im übrigen Projekt. Insbesondere müssen alle Instanzvariablen **private** sein.

Zum Vergleich zweier Werte vom Typ **key_type** ist **std::less<key_type>** zu verwenden (oder der alias **key_compare** falls vorhanden). **std::less<key_type>{}(key1,key2)** liefert für die beiden Werte **key1** und **key2 true**, falls **key1** kleiner als **key2** ist und **false** sonst.

Achtung: die Verwendung von **std::less<key_type>** ist bei Hashverfahren wie bisher verboten und auch bei dieser Aufgabe nur im Zuge von externen (!) Aufrufen von **erase()** oder **clear()** erlaubt. Wie bereits oben erwähnt, ist auch die Historie nur bei externen Aufrufen von **erase()** oder **clear()** zu aktualisieren. Wenn diese Methoden auch intern verwendet werden, dann sind diese internen Aufrufe entsprechend anders zu behandeln (zB durch eigene private Varianten dieser Methoden oder ähnliches).