- Studierendenausweis auf dem Platz hinterlegen
- Elektronische Geräte aller Art **ausschalten** und **unerreichbar verstauen**.
- Einloggen
- Lesen und Beachten der Regeln, die auf der Webseite angeführt sind.
- Änderungen in Ihrem Programm durchführen (Edit Button)
- Hochladen und Testen (Abschicken Button)
- Sichern ohne Test ist ebenfalls möglich (Speichern Button rechts oben)
- Um mit Ctrl-F im Programmtext zu suchen, müssen Sie zuerst in das Editorfenster klicken, da sonst eventuell die Browsersuche aktiviert wird.
- Hilfreiche Tastenkombinationen: Alt-Pos1 (Startseite), Strg-W (Fenster schließen) – Achtung: nicht gespeicherte Änderungen gehen bei beiden Tastenkombinationen verloren
- Sobald Sie SUCCESS erreichen, melden Sie sich bitte **unbedingt** bei der Prüfungsaufsicht, bevor Sie Ihren Arbeitsplatz verlassen. Andernfalls ist Ihr Ergebnis ungültig.
- Wenn kein SUCCESS erreicht wird, kann die Klausur zum nächsten Termin wiederholt werden.
- Programme, die SUCCESS erreichen, werden noch auf Korrektheit (z.B. Ordnungsverhalten) geprüft. Ihr Projekt wird ebenfalls noch einem Korrektheits- (richtige Datenstruktur, Einhaltung der Spezifikation) und einem Plagiatscheck unterzogen. Falls all diese Prüfungen auch erfolgreich durchlaufen werden, werden die Punkte für das Projekt vergeben.

**Verwendung zusätzlicher Hilfsmittel bzw. Kommunikation mit NachbarInnen wird als Versuch gewertet, die Leistung zu erschleichen.**

Erweitern Sie Ihre Klasse **ADS_set** um die Methode
**size_type z() const;**
Diese soll die Anzahl der in der Datenstruktur gespeicherten Werte retournieren, die kleiner sind, als der zuletzt erfolgreich mittels **erase** gelöschte Wert. Wurde zum Zeitpunkt des Aufrufs der Methode **z** noch nie ein Wert erfolgreich mittels **erase** gelöscht, so ist eine Exception vom Typ **std::runtime_error** (**#include <stdexcept>**) zu werfen. Es sind nur externe Aufrufe von **erase** zu berücksichtigen. Eventuelle interne Aufrufe durch andere Methoden von **ADS_set** (z.B. **clear**) sind zu ignorieren.
Zum Vergleich zweier Werte vom Typ **key_type** ist **std::less<key_type>** zu verwenden. Der Aufruf **std::less<key_type>{}(key1,key2)** für die beiden Werte **key1** und **key2** liefert **true**, falls **key1** kleiner als **key2** ist und **false** sonst.

Die Verwendung von Iteratoren (und damit auch die Verwendung einer range based for loop) zum Iterieren über die in der Datenstruktur gespeicherten Werte ist nicht erlaubt. Die Zeitkomplexität der Funktion **z** muss O(n) sein, die Speicherkomplexität O(1). Es ist also beispielsweise nicht erlaubt, die Werte zu sortieren, auf ein eventuell vorhandenes Werte-Cache zuzugreifen, oder zusätzliche Felder mit einer nicht konstanten Größe zu verwenden.

Beispiel:
Gespeicherte Werte seien: C={7,1,5,3,0} der zuletzt erfolgreich gelöschte Wert sei 3 (3 wurde offenbar danach wieder eingefügt). Ein Aufruf von z muss dann 2 liefern (da 0 und 1 kleiner als 3 sind).

Eine mathematische Formulierung der Aufgabenstellung für jene, die diese Art der Beschreibung bevorzugen: Sei $C$ die Menge der im Container gespeicherten Werte und $g$ der zuletzt mittels **erase** erfolgreich gelöschte Wert ($g = null$ , falls noch nie ein Wert mittels **erase** erfolgreich gelöscht wurde). Dann liefert

$$z(): \begin{cases} |\{x | x \in C \ \wedge std :: less < key\_type > \{\}(x, g) = true\}| \ falls \ g \neq null \\ eine \ Exception \ vom \ Typ \ std :: runtime\_error \ sonst \end{cases}$$

Tipp:
Sie benötigen *zumindest* eine weitere Instanzvariable in Ihrer Klasse, um sich zu merken, ob bereits ein Wert erfolgreich mittels **erase** entfernt wurde und wenn ja, welcher Wert das war. Diese Information ist bei jedem externen Aufruf von **erase** entsprechend zu pflegen und kann dann in der Methode **z** verwendet werden. **z** wirft eine Exception vom Typ **std::runtime_error** , falls noch nie ein Wert erfolgreich mittels **erase** gelöscht wurde. Andernfalls werden alle in der Datenstruktur gespeicherten Werte mit dem zuletzt erfolgreich gelöschten Wert verglichen. Immer wenn der gespeicherte Wert kleiner ist, wird jeweils ein Zähler erhöht. Der Wert des Zählers wird am Ende von **z** retourniert. (Achtung: Die Verwendung von Iteratoren oder einer range based for loop zum Durchlaufen der gespeicherten Werte ist nicht erlaubt!)
Die Informationen bezüglich des zuletzt gelöschten Wertes werden beim Kopieren von Objekten (also im Kopierkonstruktur und Kopierzuweisungsoperator) oder bei swap-Operationen nicht übernommen. Die Informationen werden im neu erstellten Objekt auf den Anfangszustand gesetzt (Kopierkonstruktor) bzw. bleiben unverändert (Kopierzuweisungsoperator und swap-Operation).