SWE 4x	**Übung zu Softwareentwicklung mit modernen Plattformen 4**	**SS 2016, Übung 2**

Abgabetermin: SA in der KW 14

☐ **Gr. 1**, Dr. E. Pitzer **Name** _____ **Aufwand in h** _____

☐ **Gr. 2**, DI F. Gruber-Leitner

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

Skip Lists (24 Punkte)

Zur Realisierung der abstrakten Datenstrukturen sortierte Menge *(set)* und sortiertes assoziatives Feld *(dictionary, map)* werden meistens binäre Suchbäume verwendet. Damit die Operationen auf diese Datenstrukturen (Finden, Einfügen und Löschen) in allen Fällen eine logarithmische Laufzeitkomplexität aufweisen, muss der Suchbaum bei jeder schreibenden Operation ausbalanciert werden. Die Implementierung von balancierten Bäumen (Rot-Schwarz-Baum) ist allerdings sehr aufwändig.

In einer schon relativ alten Arbeit aus dem Jahr 1990 stellt William Pugh sogenannte *Skip Lists* vor (das vollständige Paper finden Sie unter ftp://ftp.cs.umd.edu/pub/skipLists/skiplists.pdf), welche sich ebenfalls zur Implementierung der oben angesprochenen abstrakten Datenstrukturen eignen. Skip-Lists haben zwar im Durchschnitt dasselbe asymptotische Laufzeitverhalten wie balancierte Suchbäume, sind aber bei Weitem einfacher zu implementieren.

a) Studieren Sie das Paper von Pugh und erstellen Sie auf Basis der daraus gewonnenen Erkenntnisse eine C++-Schablone *skip_set*, welche die abstrakte Datenstruktur Menge implementiert, allerdings intern Skip Lists zur Repräsentation der Knoten verwendet. Die Schnittstelle von *skip_set* soll zunächst folgenermaßen aussehen:

```
template<typename T, const int MAXLEVEL=32>
class skip_set {
public:
  skip_set();
  ~skip_set();
  int size() const;
  bool find(T value);
  void insert(T value);
  bool erase(T value);
};
```

b) Erweitern Sie *skip_set* um einen Vorwärts-Iterator. Ein Vorwärts-Iterator muss die Operatoren ==, !=, ++ (Prä- u. Postinkrement), * und -> unterstützen. Die Schnittstelle von *skip_set* muss dafür natürlich auch erweitert und teilweise angepasst werden:

```
template<typename T, const int MAXLEVEL=32>
class skip_set {
public:
  typedef skip_set_iterator iterator;
  ...
  iterator find(T value);
  ...
  iterator begin();
  iterator end();
};
```

c) Erweitern Sie den in b) entwickelten Iterator zu einem bidirektionalen Iterator. Sie müssen dafür leichte Modifikationen an der internen Datenstruktur vornehmen.

d) Führen Sie für die Mengen-Implementierung in der STL und Ihre Implementierung ausführliche Laufzeitanalysen durch. Übersetzen Sie dazu Ihre Quelltexte mit aktivierter Optimierungsoption. Überprüfen Sie zunächst, ob das erwartete asymptotische Laufzeitverhalten tatsächlich eintritt und belegen Sie Ihre Beobachtungen mit einer Grafik. Präzisieren Sie die Unterschiede im Laufzeitverhalten, indem Sie aus Ihrem Datenmaterial eine Regressionsfunktion für beide Implementierungen ermitteln.