SWE 4x Übung zu Softwareentwicklung mit modernen Plattformen 4 — SS 2015, Übung 1

Abgabetermin: Ostersamstag in der KW 14

☑ **Gr. 1, E. Pitzer** ☐ **Gr. 2, F. Gruber-Leitner**

Name Stefan Kert **Aufwand in h** 8

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

STL Hashtable (12 + 12 Punkte)

Da wir nun bereits viele Containerklassen der STL kennengelernt haben wird es an der Zeit zu versuchen selbst einen generischen Container zu implementieren. Dazu bietet sich eine einfache Hashtabelle an, die einigermaßen flexibel sein soll. Wir wollen, der Einfachheit halber, lineare Verkettung zur Auflösung von Kollisionen verwenden und nur die wichtigsten Operationen anbieten, sowie die Ausgabe auf einen Stream ermöglichen.

Außerdem wollen wir die eigentliche Hashfunktion sowie die Gleichheit von Elementen konfigurierbar machen. Dazu können wir im ersten Schritt gleich die Funktoren `std::hash()` und `std::equal_to()` der STL verwenden. Da es sich bei diesen Funktionen sowohl um einfache C Funktionszeiger als auch um C++ Funktoren handeln kann, müssen die Typen dieser Funktionen als Templateparameter deklariert werden. Es kommen also zusätzlich zum Templateparameter `V` für den Wertetype (value) noch die Templateparameter `H` für die Hashfunktion sowie `C` für die Vergleichsfunktion (compare) dazu.

Als krönenden Abschluss brauchen wir noch die Implementierung eines einfachen Iterators um auch die bereits bestehenden Algorithmen der STL auf unsere Hashtabelle anwenden zu können.

1. Implementieren Sie dir Grundfunktionalität einer STL-kompatiblen Hashtabelle, die mindestens die folgende Schnittstelle erfüllen muss und testen Sie Ihre Implementierung ausführlich.

```
template<typename V, typename H, typename C>
class hashtable {
public:
  void insert(const V &value);
  void erase(const V &value);
  bool contains(const V &value);
  void rehash(size_t new_n_buckets);

  double load_factor() const;
  size_t size() const;
  size_t capacity() const;
  bool empty() const;
};

template<typename V, typename H, typename C>
std::ostream & operator << (std::ostream & os, const hashtable<V, H, C> &ht);
```

Überlegen Sie sich dazu eine geeignete Komposition aus STL Containern um diese Funktionalität in Form einer Hashtabelle mit linearer Verkettung zu erhalten.

2. Erweitern Sie Ihre Implementierung nun auch um einen einfachen Iterator den Sie z.B. von `std::iterator` ableiten können.

```
typedef std::iterator <std::bidirectional_iterator_tag,
                       value_type,
                       difference_type,
                       const_pointer,
                       const_reference> iterator_base;

class const_iterator : public iterator_base {
public:
  bool operator == (const_iterator const & rhs) const;
  bool operator != (const_iterator const & rhs) const;
  reference operator *  () const;
  pointer   operator -> () const;

  const_iterator & operator ++ ();
  const_iterator & operator -- ();

  const_iterator operator ++ (int);
  const_iterator operator -- (int);
};
```

Abschließend können Sie diesen Iterator dann auch noch verwenden um den Vergleichsoperator für Hashtabellen so zu überschreiben, dass zwei Hashtabellen dann gleich sind, wenn sie die gleichen Elemente enthalten.

```
bool hashtable::operator == (const hashtable &other) const;
```

Hinweis: Um unnötige Stolperfallen im Umgang mit der STL zu vermeiden, verwenden Sie am besten die Vorlage `hashtable_template.hpp` die Sie nur mehr erweitern müssen.