Dr. Ole Klein, Dr. Steffen Müthing
IWR, Universität Heidelberg

Abgabe 22. Dezember 2017

Übung 1 *Häufigkeit von Buchstaben*

Bisher haben Sie in den Übungen nur die Container `std::array` und `std::vector` verwendet, um Daten zu speichern. Diese beiden Datentypen modellieren eine Liste fixer Länge, bei der jeder Eintrag über einen 0-basierten, konsekutiven Index addressiert wird.

In vielen Fällen sind diese Datentypen völlig ausreichend, manchmal ist es jedoch praktisch, andere Werte als Zahlen (oder nicht-konsekutive Zahlen) zu verwenden, um auf Einträge zuzugreifen.

Hierfür gibt es in C++ zwei Datentypen, die nun jeweils zwei Parameter in spitzen Klammern (sogenannte Template-Parameter) bekommen, einen für den Index-Typ (im folgenden `Key`) und einen für den Eintrag (im folgenden `Value`), analog zu den bisherigen Containern:

- `std::map<Key,Value>`[1] (Header `map`) speichert die Einträge in sortierter `Key`-Reihenfolge und benötigt deshalb Keys, die man sinnvoll anordnen kann.
- `std::unordered_map<Key,Value>`[2] (Header `unordered_map`) speichert die Einträge in zufälliger Reihenfolge; wenn die Einträge nicht sortiert sein müssen, ist sie bei einer großen Anzahl von Keys aber deutlich schneller.

Maps haben keine fixe Größe: Beim ersten Zugriff auf einen noch nicht existierenden Key wird der zugehörige Wert erzeugt, indem der Default-Konstruktor aufgerufen wird. Zahlen (die keinen Konstruktor haben) werden stattdessen mit 0 initialisiert.

```
1  // we want an alphabetically sorted shopping list, so
2  // we use std::map instead of std::unordered_map
3  std::map<std::string,int> shopping_list;
4  shopping_list["cookies"] = 3;
5  std::cout << shopping_list["cookies"] << std::endl;
6  // Next line outputs 0, because the entry gets created here
7  std::cout << shopping_list["biscuits"] << std::endl;
8  // Puts a total of 7 candles on the shopping list
9  for (int i = 0 ; i < 7 ; ++i)
10   shopping_list["candles"] += 1;
```

Zum Iterieren über die Einträge einer Map verwendet man am besten den verbesserten `for`-Loop. Hierbei gibt es jedoch ein Problem: Oft benötigt man sowohl den Key als auch den zugehörigen Wert (z.B. um unsere Einkaufsliste auszudrucken). C++ löst dieses Problem, indem es beim Iterieren statt des Werts ein `std::pair`[3] zurückgibt. Wie der Name andeutet, repräsentiert diese Klasse ein Paar von zwei Werten (in diesem Fall Key und Value). Da der Name dieses Typs sehr umständlich zu tippen ist, verwenden wir hier als Typ `auto`, um den Compiler den richtigen Typ raten zu lassen. Dies wird in einer kommenden Vorlesung genauer erklärt:

```
1  // Print the shopping list
2  for (auto entry : shopping_list)
3  {
```

[1] http://en.cppreference.com/w/cpp/container/map
[2] http://en.cppreference.com/w/cpp/container/unordered_map
[3] http://en.cppreference.com/w/cpp/utility/pair

```
  // entry.first gets you the key
  std::cout << entry.first << ": ";
  // entry.second gets you the value
  std::cout << entry.second << std::endl;
}
```

Wenn man die Values verändern will, muss man wie gewohnt eine Referenz verwenden:

```
// Buy one extra of each item on the list
for (auto& entry : shopping_list)
{
  // get the value by reference (the & above) and increase by 1
  entry.second += 1;
}
```

Diese neuen Datentypen verwenden wir nun, um die Häufigkeit von Buchstaben bzw. Wörtern in einem Text auszuwerten.

Aufgaben:

(a) Schreiben Sie eine Funktion

```
std::map<char,int> get_frequencies();
```

die Buchstaben (Typ `char`) von der Standardeingabe liest, bis die Standardeingabe geschlossen wird, und zählt, wie oft die einzelnen Buchstaben vorkommen. Um alle Buchstaben von der Standardeingabe einzulesen, verwenden Sie folgenden Codeschnipsel:

```
while (true)
{
  unsigned char c;
  // read in character
  std::cin >> c;
  // abort if input closed
  if (not std::cin)
    break;
  // work with c
  // PUT YOUR CODE THAT PROCESSES c HERE
}
```

(b) Schreiben Sie eine Funktion

```
void print_frequencies(const std::map<char,int>& frequencies);
```

die eine Liste von Buchstaben und zugehörigen Häufigkeiten auf die Standardausgabe ausgibt.

(c) Schreiben Sie ein Programm `letterfrequencies`, das die beiden Funktionen aus (a) und (b) benutzt, um die Buchstabenhäufigkeit eines Textes zu untersuchen. Hierzu soll es die beiden Funktionen einfach nacheinander aufrufen und den Rückgabewert der ersten Funktion als Parameter an die zweite Funktion weiterreichen.

Sie können Ihr Programm entweder testen, indem Sie Text in die Konsole tippen und die Eingabe mit `CTRL+D` beenden, oder Sie lassen Ihr Programm den Inhalt einer Datei verarbeiten:

```
./letterfrequencies < dateiname
```

(d) Wir sind eigentlich nur an Buchstaben interessiert, aber im Moment gibt Ihr Programm auch die Häufigkeit von Ziffern und Sonderzeichen aus. Verwenden Sie die Funktion[4]

```
bool std::isalpha(char c)
```

[4] `http://en.cppreference.com/w/cpp/string/byte/isalpha`

die `true` zurückgibt, wenn `c` ein Buchstabe ist, und überspringen Sie mit ihrer Hilfe alle Zeichen, die kein Buchstabe sind.

(e) Ihr Programm zählt Großbuchstaben getrennt von Kleinbuchstaben. Beheben Sie dies, indem Sie die Buchstaben mit der Funktion[5]

```
1 char std::toupper(char c)
```

in Großbuchstaben umwandeln und so Groß- und Kleinbuchstaben zusammen zählen. Sie können die Funktion auch mit einem Großbuchstaben aufrufen, dieser wird unverändert zurückgegeben.

(f) Ergänzen Sie Ihr Programm so, dass es abschliessend noch die Gesamtanzahl an **mitgezählten** Buchstaben ausgibt.

(g) Um die Ergebnisse unterschiedlich langer Texte vergleichbar zu machen, verändern Sie die Ausgabe so, dass statt der Anzahl der Anteil an allen Buchstaben ausgegeben wird, d.h. für das Wort w geben Sie

$$p_w = \frac{f[w]}{\sum_{w' \in W} f[w']}$$

aus, wobei W die Menge aller Wörter ist und f die Map mit den Häufigkeiten. Wichtig: Vor der Division müssen Sie eine der beiden Zahlen in `double` umwandeln, sonst bekommen Sie immer 0 als Ergebnis.

Hinweise:

- Um die Aufgabe nicht zu schwierig zu machen, werden Umlaute nicht korrekt verarbeitet. Verwenden Sie deshalb am besten englische Texte.
- Auf der Website `https://gutenberg.org` können Sie viele ältere Bücher als Text-Datei herunterladen, z.B.
  - die King-James-Bibel: `https://www.gutenberg.org/files/30/30.txt`,
  - Krieg und Frieden: `https://www.gutenberg.org/files/2600/2600-0.txt`,
  - Grimms Märchen: `https://www.gutenberg.org/files/2591/2591-0.txt`,
  - Moby Dick: `https://www.gutenberg.org/files/2701/2701-0.txt`,
  - Sherlock Holmes: `https://www.gutenberg.org/files/1661/1661-8.txt`.
- Bei großen Texten kann es sich lohnen, das Programm vom Compiler optimieren zu lassen, um die Laufzeit zu verbessern.
  - Wenn Sie Ihr Programm von Hand an der Kommandozeile übersetzen: Fügen Sie beim Kompilieren die Option `"-O3"` hinzu.
  - Wenn Sie CMake verwenden: Löschen Sie das Build-Verzeichnis und rufen Sie CMake erneut auf. Geben Sie CMake dabei die Option `"-DCMAKE_BUILD_TYPE=Release"` mit.

(Grundlagen)

Übung 2 *Häufigkeit von Wörtern*

Basierend auf Ihren Erfahrungen in der letzten Aufgabe sollen Sie jetzt statt der Häufigkeit von Buchstaben die von Wörtern zählen.

(a) Schreiben Sie eine Funktion

[5] `http://en.cppreference.com/w/cpp/string/byte/toupper`

```
std::map<std::string,int> get_frequencies();
```

die Wörter (Typ `std::string`) von der Standardeingabe liest, bis die Standardeingabe geschlossen wird, und zählt, wie oft die einzelnen Buchstaben vorkommen. Hierzu können Sie sich am Gerüst der vorherigen Aufgabe orientieren und **`char`** durch `std::string` ersetzen.

Um Sonderzeichen und ähnliches zu entfernen, filtern Sie die Wörter mit Hilfe der Funktion `sanitize_word()` von der Vorlesungs-Homepage[6] [7]. Testen Sie nach dem Filtern, ob der resultierende String leer ist (`size() == 0`). Leere Strings sollen nicht mitgezählt werden!

(b) Schreiben Sie eine Funktion

```
void print_frequencies(const std::map<std::string,int>& frequencies);
```

die eine Liste von Wörtern und zugehörigen relativen Häufigkeiten auf die Standardausgabe ausgibt.

(c) Schreiben Sie ein Programm `wordfrequencies`, das die beiden Funktionen benutzt, um die Worthäufigkeit eines Textes zu untersuchen. Hierzu soll es die beiden Funktionen einfach nacheinander aufrufen und den Rückgabewert der ersten Funktion als Parameter an die zweite Funktion weiterreichen.

(d) Ersetzen Sie die `std::map` durch `std::unordered_map` und testen Sie, ob das Programm bei sehr großen Texten schneller wird.

(Grundlagen)

Übung 3 *Häufigkeiten für Fortgeschrittene*

(a) Bearbeiten Sie die beiden Grundlagen-Aufgaben zur Buchstaben- und Worthäufigkeit.

(b) Schreiben Sie eine Funktion

```
template <typename Map>
void print_frequencies_sorted(const Map& map);
```

- Die Funktion soll die Häufigkeiten der Einträge in der Map ausgeben.
- Die Ausgabe soll in absteigender Reihenfolge nach der **Häufigkeit der Einträge** sortiert sein.
- Die Funktion soll sowohl für Buchstaben- als auch für Worthäufigkeiten funktionieren.
- Die ausgegebenen Häufigkeiten sollen relativ sein.
- Die Funktion soll sowohl mit `std::map` als auch mit `std::unordered_map` funktionieren.

Tipp: Schauen Sie sich die Funktion `std::sort(RandomIt first, RandomIt last, Compare comp)`[8] an, bei der Sie eine Funktion `comp` übergeben können, mit der Sie eine von Ihnen definierte Sortierreihenfolge vorgeben.

(Fortgeschritten)

[6] `https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2017/sanitizeword.hh`
[7] `https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2017/sanitizeword.cc`
[8] `http://en.cppreference.com/w/cpp/algorithm/sort`

Übung 4 *Histogramme*

Ein Histogramm ist eine (eigentlich graphische) Darstellung der Häufigkeitsverteilung einer Menge von Zahlen[9]. Gegeben eine Menge von N Zahlen $X = \{x_i\}_{i=1,\ldots,N}$, erstellen wir M **bins**, die jeweils ein Teilintervall I_m nach folgender Regel abdecken:

$$I_m = \left[\min X + (m-1)\frac{\max X - \min X}{M}, \min X + m\frac{\max X - \min X}{M}\right), i = 1, \ldots, M-1,$$
$$I_M = \left[\min X + (M-1)\frac{\max X - \min X}{M}, \min X + M\frac{\max X - \min X}{M}\right].$$

Nun zählen wir für jeden bin, wie viele der Zahlen in X in diesen bin fallen und können das Ergebnis als Histogramm ausgeben.

Aufgaben:

(a) Schreiben Sie eine Klasse `Histogram` in einer Header-Datei `histogram.hh` und zugehöriger Implementierungs-Datei, die folgendes Interface erfüllt:

```
class Histogram
{
public:
  // default-constructible
  Histogram();
  // add new number to data set
  void insert(double x);
  // return size of data set
  unsigned int size() const;
  // return smallest value in data set
  double min() const;
  // return largest value in data set
  double max() const;
  // classify the data into bin_count bins and
  // print the histogram for the data set to stdout
  void print(unsigned int bin_count);
}
```

Die Ausgabe des Histograms muss nicht graphisch erfolgen und könnte z.B. so aussehen:

```
Histogram of 36512 entries, min = 0.2, max = 1.2
0.2 -- 0.4: 6572
0.4 -- 0.6: 11684
0.6 -- 0.8: 10954
0.8 -- 1.0: 5476
1.0 -- 1.2: 1826
```

(b) Testen Sie Ihre Klasse, indem Sie mit den Funktionen von `io.hh` (Blatt 4) Zufallszahlen erzeugen und dafür Histogramme ausgeben. Verwenden Sie beide Verteilungen mit mehreren verschiedenen Parametern und unterschiedliche Mengen an bins.

(c) Ergänzen Sie die Klasse um eine Methode

```
void print_normalized(unsigned int bin_count);
```

Diese Methode soll die Werte bei der Ausgabe normalisieren, indem es jeden Wert durch die Summe der Werte aller bins teilt. Denken Sie daran, die Zahlen vorher nach **`double`** zu konvertieren! Ihre Ausgabe könnte dann so aussehen:

[9] `https://de.wikipedia.org/wiki/Histogramm`

```
Histogram of 36512 entries, min = 0.2, max = 1.2
0.2 -- 0.4: 0.18
0.4 -- 0.6: 0.32
0.6 -- 0.8: 0.30
0.8 -- 1.0: 0.15
1.0 -- 1.2: 0.05
```

(d) Wenn Sie die Aufgaben 1 und / oder 2 gemacht haben, erweitern Sie die Programme zur Buchstaben- und Worthäufigkeit, so dass zusätzlich ein Histogramm ausgegeben wird.

(Grundlagen)

Übung 5 *Histogramme für Fortgeschrittene*

(a) Bearbeiten Sie die Grundlagen-Aufgabe zu Histogrammen.

(b) Erweitern Sie die Funktion so, dass Sie das Histogramm als ASCII-Art mit folgender Formatierung (ungefähr) ausgibt:

```
Histogram of 36512 entries, min = 0.2, max = 1.2
0.2 -- 0.4: ***********                          0.23
0.4 -- 0.6: ************************             0.35
0.6 -- 0.8: ****                                 0.09
0.8 -- 1.0: **********                           0.20
1.0 -- 1.2: ******                               0.12
```

Um die Zahlen so zu formatieren, dass sie immer die gleiche Breite einnehmen, schauen Sie sich die I/O manipulators[10] an.

(Fortgeschritten)

[10] http://en.cppreference.com/w/cpp/io/manip