Programmierkurs IPK SS23
Andreas Naumann
IWR, Universität Heidelberg

Projektaufgabe zur Abgabe
Abgabedatum 3. April 2023, 13:00 Uhr
ipk-exercises:2023-ss-5-gff45c3f

Projektaufgabe zur Abgabe

1 Abgabeformat

- Erstellen sie ein privates Projekt auf https://edu.ziti.uni-heidelberg.de/
- Fügen Sie die Tutoren und den Dozenten zu dem Projekt mit der Rolle `Maintainer` hinzu.
- Erstellen Sie einen Branch mit passenden Namen. (z.B. Abgabe)
- Erstellen Sie eine `CMakeLists.txt`-Datei, die alle Lösungen enthält
- Schreiben Sie für jede Aufgabe ein Programm mit dem Namen `loesung_X` wobei X durch die die Aufgabennummer ersetzt wird.
- Falls Sie Implementierungen für mehrere Lösungen nutzen, verwenden Sie eine Bibliothek. (shared oder static spielt keine Rolle)
- Taggen finalen Stand in git mit dem Tag `final`.
- Erstellen Sie einen Merge-Request für diesen Branch und weisen diesen einem Tutor oder dem Dozenten zu.
- Nur vollständig erzeugbare Programme können Punkte bekommen. (cmake und make müssen erfolgreich durch laufen)
- Es gibt höchstens 24 Punkte. Zur Zulassung sind 9 Punkte notwendig.

Aufgabe 1: Histogramme (Punkte: 7)

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

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

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.h` und zugehöriger Implementierungs-Datei, die folgendes Interface erfüllt:

[1] https://de.wikipedia.org/wiki/Histogramm

```
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.h` (Blatt 5) 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:

```
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
```

Aufgabe 2: Häufigkeit von Buchstaben (Punkte: 7)

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.

In dieser Aufgabe verwenden wir stattdessen Maps, 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. Das Ergebnis soll sie im Rückgabewert speichern.

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[2]

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

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[3]

```
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.

[2] http://en.cppreference.com/w/cpp/string/byte/isalpha
[3] http://en.cppreference.com/w/cpp/string/byte/toupper

(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 den Buchstaben b geben Sie

$$p_b = \frac{f[b]}{\sum_{b' \in B} f[b']}$$

aus, wobei B die Menge aller gefundenen Buchstaben 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, weil der Compiler eine Ganzzahl-Division durchführt. Um einen Wert in `double` umzuwandeln, verwenden Sie folgende Syntax:

```
1 int b = 3;
2 int sum_all_b = ...;
3 p_b = static_cast<double>(b) / sum_all_b;
```

Hinweise:

- Um die Aufgabe nicht zu schwierig zu machen, werden Umlaute nicht korrekt verarbeitet. Verwenden Sie deshalb am besten englische Texte. ChatGPT oder Lorem-lipsum sind dafür gute Quellen.
- 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.

Aufgabe 3: Häufigkeit von Wörtern (Punkte: 5)

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

```
1 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[4] [5]. Testen Sie nach dem Filtern, ob der resultierende String leer ist (`size()` == 0). Leere Strings sollen nicht mitgezählt werden!

(b) Schreiben Sie eine Funktion

```
1 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.

Aufgabe 4: Häufigkeiten für Fortgeschrittene (Punkte: 5) (Bonus)

Teilaufgaben:

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

(b) Schreiben Sie eine Funktion.

```
1 template <typename Map>
2 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 bei Wörtern Wort- und Buchstabenhäufigkeit, bei Buchstaben nur die Buchstabenhäufigkeit zählen.
- Die ausgegebenen Häufigkeiten sollen relativ sein.
- Die Funktion soll sowohl mit `std::map` als auch mit `std::unordered_map` funktionieren. Es reicht, wenn die Funktion für diese Maps funktioniert.

Tipps:

- Tipp: Schauen Sie sich die Funktion `std::sort(RandomIt first, RandomIt last, Compare comp)`[6] an, bei der Sie eine Funktion `comp` übergeben können, mit der Sie eine von Ihnen definierte Sortierreihenfolge vorgeben.

[4] https://scoop.iwr.uni-heidelberg.de/teaching/2023ss/grundkurscpp/sources/sanitizeword.h
[5] https://scoop.iwr.uni-heidelberg.de/teaching/2023ss/grundkurscpp/sources/sanitizeword.cc
[6] http://en.cppreference.com/w/cpp/algorithm/sort

- Nutzen Sie Überladung oder `std::is_same` um zwischen Wörtern und Buchstaben zu unterscheiden.