Tutorium Programmierkurs

Sommer 2022 **Übung 3**

Donald Knuth.

Donald Ervin Knuth (* 10. Januar 1938 in Milwaukee, Wisconsin) ist ein US-amerikanischer Informatiker. Er ist emeritierter Professor an der Stanford University, Autor des Standardwerks The Art of Computer Programming und Urheber des Textsatzsystems TeX.
Knuth ist der Sohn eines Lehrers für Buchhaltung, der daneben noch eine kleine Druckerei unterhielt. Er besuchte die Milwaukee Lutheran High School und begann sein Physikstudium am Case Institute of Technology (heute bekannt als Case Western Reserve University) im September 1956. Aus zweierlei Gründen schlug er ab seinem zweiten Studienjahr jedoch den Weg zur Mathematik ein: Zum einen löste er ein Problem eines seiner Mathematikprofessoren, was ihm eine 1,0 als Note einbrachte, zum anderen fand er wenig Gefallen an den physikalischen Praktika.
Er erhielt einen Bachelor- und gleichzeitig einen Master-Abschluss 1960 an der Case Western Reserve University. 1963 erhielt er seinen Ph.D. vom California Institute of Technology bei Marshall Hall, wo er dann auch nach der Promotion Assistant Professor und 1966 Associate Professor und schließlich Professor wurde. 1968 wurde er Professor für Informatik an der Stanford University. Ab 1977 war er dort Fletcher Jones Professor of Computer Science und ab 1990 Professor of the Art of Computer Programming. Seit 1993 ist er Professor Emeritus.

Aufgabe 5.1: Zahlen einlesen

Anstatt sich auf die eingebauten Funktionen von C++ zu verlassen, schreiben Sie im folgenden eine Reihe von Funktionen, die einen String (z.B. `"1346"`) in eine Zahl umwandeln sowie ein Hauptprogramm, das diese Funktionen testet.

Für den Anfang können Sie davon ausgehen, dass Sie nur gültige Zeichenfolgen als Eingabe bekommen.

(a) Schreiben Sie eine Funktion `int parse_int(std::string number)`, die den String `number` in eine Zahl umwandelt und diese zurückgibt. Sie können davon ausgehen, dass die Eingabe kein Vorzeichen enthält.

Hinweise:

- Ein `std::string`[1] ist auch ein Container! Sie können ihn sich konzeptuell als etwas ähnliches wie ein `std::vector<char>` vorstellen. Ein einzelnes Zeichen

[1] `http://en.cppreference.com/w/cpp/string/basic_string`

können Sie in einer Variablen vom Typ `char` speichern. Das folgende Beispiel zeigt Ihnen, wie Sie an die Information in `std::string` herankommen:

```
std::string s = "47218";    // Zuweisung
std::getline(std::cin,s);   // Zeile einlesen und in s speichern, nicht mit
int size = s.size();        // Gibt die Länge des Strings zurück
char c = s[0];              // erstes Zeichen (0-basierter Index)
char d = s[size-1];         // letztes Zeichen
```

- Eine Variable vom Typ `char` enthält eine Zahl, die das zugehörige Zeichen repräsentiert. Hierfür gibt es verschiedene Standards, welche Zahl für welches Zeichen steht, für unsere Zwecke reicht ASCII[2]. Dabei ist z.B. der Wert des Zeichens `'3'` nicht 3, sondern 51. Sie müssen den Wert des Zeichens `'0'` (= 48) abziehen, um zum korrekten Ziffernwert zu kommen:

```
char three = '3';
std::cout << three << std::endl; // Ausgabe: 51
three = three - '0';             // alternativ: three - 48
std::cout << three << std::endl; // Ausgabe: 3
```

- Sie können für diesen Aufgabenteil davon ausgehen, dass der gesamte String nichts ausser der einzulesenden Zahl enthält.

(b) Erweitern Sie Ihre Funktion so, dass sie ein optionales `'+'` oder `'-'` am Anfang des Strings korrekt einliest und auf die Zahl anwendet.

(c) Erweitern Sie Ihre Funktion so, dass sie eventuelle Leerzeichen am Anfang des Strings ignoriert und die Zahl so lange weiter einliest, bis ein anderes Zeichen als eine Ziffer kommt. Beispiel: Der String `"  -7628.8text"` soll in die Zahl -7628 umgewandelt werden (`"."` ist kein gültiges Zeichen innerhalb einer ganzen Zahl, also stoppt das Einlesen dort).

Hinweis:

- Um lange Ketten von `if`-Statements zu vermeiden, kann es hier hilfreich sein, stattdessen ein `switch`-Statement[3] zu verwenden, dass in einem Statement mehrere Alternativen abhandeln kann:

```
char c = ...;
switch (c) {
  case ' ':
  // handle blank spaces
  break; // leave switch statement
  case '+':
  case '-':
```

[2] American Standard Code for Information Interchange, `https://en.wikipedia.org/wiki/ASCII`
[3] `https://en.cppreference.com/w/cpp/language/switch`

```
  // handle plus or minus
  break; // leave switch statement
  case '0':
  case '1':
  ...
  case '9':
  // handle digits
  break; // leave switch statement
  default:
  // handle any other (invalid) symbol
}
```

- Sie müssen wahrscheinlich mit einigen `bool`-Variablen nachverfolgen, ob gerade noch ein Leerzeichen oder ein Vorzeichen kommen darf.

(d) **1 Bonuspunkt:** Erweitern Sie Ihre Funktion so, dass sie statt einem `int` ein `std::pair<int,int>` zurückgibt, wobei der erste Eintrag die eingelesene Zahl sein soll und der zweite der Index des ersten Zeichens, das nicht mehr eingelesen wurde.

(e) **1 Bonuspunkt:** Erweitern Sie Ihre Funktion so, dass sie im Fehlerfall (also, wenn der String z.B. mit einem Buchstaben anfängt) eine Exception vom Typ `std::invalid_argument`[4] wirft.

Aufgabe 5.2: 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.

In dieser Aufgabe verwenden wir stattdessen Maps, um die Häufigkeit von Buchstaben bzw. Wörtern in einem Text auszuwerten.

Aufgaben:

[label= ()]Schreiben Sie eine Funktion

1. `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:

[4] `http://en.cppreference.com/w/cpp/error/invalid_argument`

```cpp
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
}
```

2. Schreiben Sie eine Funktion

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

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

3. 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
```

4. 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[5]

```cpp
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.

5. Ihr Programm zählt Großbuchstaben getrennt von Kleinbuchstaben. Beheben Sie dies, indem Sie die Buchstaben mit der Funktion[6]

```cpp
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.

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

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

7. 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:

```
int b = 3;
int sum_all_b = ...;
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.
- Auf der Vorlesungs-Homepage [7] können Sie ein paar frei verfügbare Texte als Beispiele herunterladen.
- 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.

[7] `https://moodle.uni-heidelberg.de/course/view.php?id=8916`