Dr. U. Köthe *Heidelberg, 30. November 2016*

Übungsblatt 6

Anmerkung: Beachten Sie die Hinweise zum Programmierstil auf Blatt 5. Bei groben Verstößen können Ihnen ab dieser Woche Punkte abgezogen werden.

Aufgabe 6.1 Tabelle formatieren [13 Punkte]

In dieser Aufgabe wird der Umgang mit `std::transform()` geübt. Sie sollen einen Vektor reeller Zahlen so formatieren, dass die Zahlen in einer sauber ausgerichteten Tabellenspalte gedruckt werden können. Geben Sie die Lösung in der Datei `format_numbers.cpp` ab.

(a) Implementieren Sie als Vorübung eine Funktion

```
std::vector<int> vec_to_int(std::vector<double> v)
```

Diese Funktion soll `std::transform()` mit einer geeigneten Lambdafunktion aufrufen, um die Werte des Vektors `v` zu runden. Nachkommastellen sollen zur *nächstgelegenen* ganzen Zahl gerundet werden, die Rundung von 0.5 können Sie nach Belieben festlegen. Der resultierende Vektor ganzer Zahlen wird zurückgegeben.

Beispiel:
Eingabe: double-Vektor mit den Werten {`1.6, 19.5, 2.2`}.
Ergebnis: int-Vektor mit den Werten {`2, 20, 2`}.

(b) Implementieren Sie als weitere Vorübung eine Funktion

```
std::vector<double> vec_rounded(std::vector<double> v)
```

Diese Funktion soll `std::transform()` benutzen, um die Werte des Vektors `v` wie oben zu runden, aber diesmal auf zwei Nachkommastellen genau. Der resultierende double-Vektor wird zurückgegeben.

(c) Implementieren Sie die Funktion

```
std::string double_to_string(double x)
```

Diese Funktion soll `x` wie folgt in einen String umwandeln:

- `x` wird auf zwei Nachkommastellen gerundet: `12345.6789` wird zu `12345.68`.
- Das Ergebnis wird in einen String umgewandelt: `12345.68` wird zu `"12345.68"` (benutzen Sie `std::to_string()`). Stellen Sie sicher, dass der String *genau* zwei Nachkommastellen hat: `123.0` wird zu `"123.00"`.
- Nach jeder Tausenderstelle wird ein Apostroph eingefügt: `"12345.68"` wird zu `"12'345.68"` (aber `"123.45"` bleibt `"123.45"`).
- Der String wird links mit Leerzeichen aufgefüllt, bis er die Länge 16 hat: `"12'345.68"` wird zu `"       12'345.68"`. Eventuelle Vorzeichen müssen dabei natürlich mitgezählt werden.

(d) Implementieren Sie die Funktion

```
std::vector<std::string> format_numbers(std::vector<double> v)
```

Diese Funktion soll `std::transform()` aufrufen, um die Werte des Vektors `v` mit Hilfe der Funktion `double_to_string` umzuwandeln, und den resultierenden String-Vektor zurückgeben.

(e) Auf Moodle finden Sie die Datei `format_numbers.hpp` mit geeigneten Testdaten. Kopieren Sie die Datei in dasselbe Verzeichnis wie `format_numbers.cpp` und inkludieren Sie sie mittels `#include "format_numbers.hpp"`. Rufen Sie die Funktion `format_numbers()` in `main()` für diese Daten auf und geben Sie den resultierenden Vektor zeilenweise aus. Vergewissern Sie sich, dass alle Ausgaben exakt am Dezimalpunkt ausgerichtet sind.

Aufgabe 6.2 Versionsnummern sortieren [13 Punkte]

In dieser Aufgabe wird der Umgang mit `std::sort()` geübt. Sie sollen einen Vektor mit Versionsnummern sortieren. Diese Funktionalität verwenden Betriebssysteme und Versionsmanager (wie `apt-get` unter Linux), um z.B. die neueste Version eines Programms zu finden. Für die Aufgabe nehmen wir an, dass Versionsnummern als Strings gegeben sind, die genau eine natürliche Zahl enthalten, oder mehrere durch Punkte getrennte natürliche Zahlen. Die Sortierung erfolgt *lexikographisch* (siehe `https://de.wikipedia.org/wiki/Lexikographische_Ordnung`), also nach dem gleichen Prinzip wie bei Wörtern in einem Lexikon: Zwei Versionsnummern werden zuerst nach ihrer ersten Zahl sortiert, sind diese gleich, nach der zweiten Zahl usw. Besteht eine Version aus weniger Zahlen als die andere, werden die fehlenden Zahlen auf Null gesetzt. Es gilt also beispielsweise folgende Reihenfolge:

```
"1" = "1.0" < "1.1" < "1.1.16" < "2.0" < "2.3" = "2.3.0" < "2.3.4"
```

Geben Sie die Lösung in der Datei `sort_versions.cpp` ab.

(a) Implementieren Sie eine Funktion

```
std::vector<int> split_version(std::string version)
```

die einen Versionsstring bei `"."` splittet und die Substrings in ganze Zahlen umwandelt. Die Zahlen werden dann in einen `std::vector<int>` eingefügt, der als Ergebnis zurückgegeben wird. Zum Umwandeln eines Teilstrings `zahl` können Sie z.B. den Aufruf `std::atoi(zahl.c_str())` verwenden (`std::atoi()` befindet sich im Header `<cstdlib>`).

Beispiel:
Eingabe: `"2.12.5"`.
Ergebnis: int-Vektor mit den Werten {`2, 12, 5`}.

(b) Implementieren Sie eine Funktion

```
bool version_less(std::string v1, std::string v2)
```

die genau dann `true` zurückgibt, wenn die Versionsnummer `v1` kleiner als die Versionsnummer `v2` ist. Die Funktion soll zuerst `split_version()` aufrufen, um die Versionsnummern in zwei Vektoren von Zahlen umzuwandeln, und dann über die beiden Vektoren iterieren, um den lexikographischen Vergleich durchzuführen.

(c) Inkludieren Sie mittels `#include "sort_versions.hpp"` (diese Datei finden Sie auf Moodle) geeignete Testdaten und rufen Sie in `main()` die Funktion `sort()` mit der Vergleichsfunktion `version_less` für diese Daten auf. Geben Sie den sortierten Vektor auf die Konsole aus. Vergewissern Sie sich, dass die Versionen wie gewünscht sortiert sind.

Aufgabe 6.3 Text dechiffrieren [14 Punkte]

In dieser Aufgabe werden die Ein- und Ausgabe von Dateien sowie der Umgang mit `std::map` geübt.

In Moodle finden Sie die Datei `encrypted_text.txt`. Diese Datei enthält einen englischen Text, der mit einer Substitutionschiffre verschlüsselt wurde. Zum Verschlüsseln wurde ein Alphabet mit zufälliger Reihenfolge der Buchstaben erstellt und anschließend jeder Buchstabe des normalen Alphabets durch den korrespondierenden Buchstaben (d.h. an derselben Stelle) im neuen Alphabet ersetzt. Weitere Informationen finden Sie auf `http://de.wikipedia.org/wiki/Monoalphabetische_Substitution` unter „Einfache monoalphabetische Substitution“. Da sich bei dieser Substitution die Häufigkeit der Buchstaben nicht ändert, lässt sich eine solche Verschlüsselung leicht knacken: Man zählt einfach die Häufigkeit der vorkommenden Buchstaben, vergleicht diese mit den Häufigkeiten eines normalen englischen Textes und ersetzt Buchstaben mit gleicher Häufigkeit. Ihre Aufgabe ist es, ein Programm `decrypt.cpp` zu schreiben, das den gegebenen Text auf diese Weise entschlüsselt.

(a) *Lesen Sie die Datei ein:* C++ bietet dafür Filestreams an (im Header `<fstream>`), die analog zu `std::cin` funktionieren. Um eine Textdatei zeilenweise einzulesen und in einen String zusammenzusetzen, verwenden Sie folgenden Code

```
std::ifstream infile("encrypted_text.txt"); // Datei öffnen
std::string text; // leerer String für den eingelesenen Text
std::string line; // leerer String für die aktuelle Zeile
while(infile)
{
    std::getline(infile, line); // nächste Zeile einlesen ...
    text += line + "\n";      // ... und an den Text anhängen
}
```

(b) *Bestimmen Sie die Häufigkeit der Buchstaben:* Legen Sie eine Variable `counts` vom Typ `std::map<char, int>` an, deren Schlüssel die Buchstaben und deren Werte die Häufigkeiten (mit Null initialisiert) sind. Iterieren Sie über den Text und inkrementieren Sie für jeden gefundenen Buchstaben den entsprechenden Eintrag der Map. Beachten Sie dabei, dass Groß- und Kleinschreibung ignoriert werden müssen.

(c) *Sortieren nach aufsteigender Häufigkeit:* Legen Sie eine Variable `sorted` vom Typ `std::map<int, char>` an, wo die Rollen von Schlüssel und Wert vertauscht sind. Iterieren Sie über die Map `counts` aus (b) und fügen Sie alle Schlüssel/Wert-Paare umgekehrt in die Map `sorted` ein. Da eine `std::map` die Einträge immer nach den Schlüsseln ordnet, sind die Buchstaben jetzt automatisch nach ansteigender Häufigkeit sortiert.

Die entsprechende Sortierung bei einem normalen britisch-englischen Text lautet
`z j q x k v b y g p w f m c u l d r h s n i o a t e`

(`z` ist der seltenste, `e` der häufigste Buchstabe). Speichern Sie diese Sortierung in einem `std::vector<char> letters`.

(d) *Zuordnung von Klarbuchstaben zu verschlüsselten Buchstaben:* Legen Sie eine Variable `decrypt` vom Typ `std::map<char, char>` an, deren Schlüssel die verschlüsselten Buchstaben und deren Werte die zugehörigen entschlüsselten Buchstaben sind. Iterieren Sie simultan über die Map `sorted` und den Vector `letters` aus (c) und legen Sie für jedes Buchstabenpaar einen entsprechenden Eintrag in der Map `decrypt` an. Beachten Sie, dass die neue Map die Zuordnung für Klein- *und* für Großbuchstaben enthalten muss.

(e) *Text entschlüsseln:* Iterieren Sie über den verschlüsselten Text und ersetzen Sie jeden Buchstaben *in-place* durch seinen Klarbuchstaben. Leerzeichen, Zahlen und Satzzeichen sollen bei der Entschlüsselung erhalten bleiben. Schreiben Sie das Ergebnis mit folgendem Code in eine Datei:

```
std::ofstream outfile("decrypted_text.txt"); // Datei öffnen
outfile << text; // text enthält jetzt den entschlüsselten Text
```

Geben Sie die Datei `decrypted_text.txt` zusammen mit `decrypt.cpp` ab.

Bitte laden Sie Ihre Lösung spätestens bis 7. Dezember 2016, 9:00 Uhr in Moodle hoch.