IMS
Institut für Mikroelektronische Systeme
Leibniz Universität Hannover

Programmierpraktikum Technische Informatik (C++)
Aufgabe 3

Hinweise
Abgabe: Stand des Git-Repositories am 4.5.2023 um 9 Uhr.
Die Dateien zur Bearbeitung dieser Aufgabe erhalten Sie, indem Sie die neue Aufgabe aus dem Aufgabenrepository in Ihr lokales mergen. Dies geschieht mit `git pull common main` innerhalb Ihres Repositories. Öffnen Sie ein Terminal in dem Verzeichnis der jeweiligen Aufgabe und geben Sie "`code .`" ein. Dann öffnet sich VSCode, in dem das CMake-Tools-Plugin installiert sein sollte. Sie können dann in der unteren blauen Zeile die Schaltflächen zum Kompilieren (Build), Starten und ggf. Testen (Run CTest) verwenden. Es sind nicht zu allen Aufgaben CTests vorgesehen. Die Lösungen committen Sie bitte in Ihr lokales Repository und pushen sie in Ihr Repository auf den Gitlab-Server.

Teilaufgabe 1 (3 Punkte)

Um eine geheime Nachricht zu codieren, wird in dieser Aufgabe eine Variante der Caesar-Verschlüsselung benutzt. Anstatt, dass jedes Zeichen um die gleiche Distanz verschoben wird, wird allerdings jedes Zeichen um eine zufällige Weite verschoben. Diese jeweilige Verschlüsselung ist in einer `std::map` vermerkt. Es existiert bereits die Funktion `std::tuple<std::string, std::map<char, char>> createCipherMap(std::istream& is)`, die den Inhalt einer Datei verschlüsselt und sowohl den sich daraus ergebenden verschlüsselten Text als `std::string` als auch die zugerörige `std::map` zurückgibt.

a) Implementieren Sie in der Datei `decipherer.cpp` die Funktion `std::string decipherMessage(const std::string& codedMessage, const std::map<char,char>& cipher)`! Diese Funktion soll einen übergebenen verschlüsselten String mit Hilfe der übergebenen `std::map` decodieren und danach zurückgeben.

   Hinweise:

   - Die übergebene Map enthält alle in dem String vorkommenden Zeichen als Schlüssel.
   - Die Werte, die mit den Schlüsseln assoziiert sind, liegen ebenfalls als `char` vor.
   - Die Decodierung ist jedoch nicht eine 1-zu-1-Übersetzung zwischen dem

Appelstraße 4 | 30167 Hannover
fon 0511 762-19691 | fax 0511 762-19694
www.ims.uni-hannover.de | info@ims.uni-hannover.de

Schlüssel und dem Wert!

- Ein einzelner Character wird decodiert, indem die Summe von dem Schlüssel und dem dazugehörigen Wert gebildet wird.
- Um auf einen Wert zuzugreifen, der zu einem Schlüssel gehört, ist die Methode `at` der `std::map` hilfreich.

b) Während der Übertragung sind Fehler in der Nachricht erschienen. Vereinzelt sind zusätzliche Zeichen übertragen worden. Es ist davon auszugehen, dass Zeichen, die unverfälscht übertragen werden, deutlich häufiger vorkommen als fehlerhafte.

Implementieren Sie in der Datei `decipherer.cpp` die Funktion `std::string removeErrors(const std::string& messageWithErrors)`! Diese Funktion soll das am seltensten erscheinende Zeichen im übergebenen String entfernen und einen korrigierten String zurückgeben.

Hinweise:

- Als erstes sollten alle Zeichen gezählt werden. Eine `std::map`, die die Zeichen als Schlüssel und die Häufigkeit als Werte beinhaltet, kann dafür hilfreich sein.
- Um ein neues Schlüssel-Wert-Paar der Map hinzuzufügen, besitzt die `std::map` die Methode `emplace`.
- Um zu überprüfen, ob die Map bereits einen bestimmten Schlüssel besitzt, gibt es die Methode `find`.
- Als nächstes muss das seltenste Zeichen bestimmt werden.
- Beim Durchlaufen einer `std::map` mit einer Range-based-for-Schleife erhält man die Elemente der Map als `std::pair<KeyType, ValueType>`.
- Auf die Elemente eines `std::pair` kann mit den Methoden `first` und `second` zugegriffen werden.
- Als letztes muss das seltenste Zeichen aus der fehlerhaften Nachricht enfernt werden und der so entstandene String zurückgegeben werden.

Achtung: In den Vorlesungen 3 und 4 wurden Regeln zur C++-Programmierung vorgestellt. Außerdem sollen beim Kompilieren keine Warnings mehr erscheinen. Eine Nichtbeachtung dieser Regeln führt zu Punktabzug, sofern keine überzeugende Begründung für die Missachtung geliefert wird.

Teilaufgabe 2 (2 Punkte)

In dieser Aufgabe soll eine Graphen-Datenstruktur eines gerichteten Graphen erstellt werden. In `graph.h` sind die entsprechenden Klassen `Vertex, Edge, Graph` bereits definiert. Bei Aufruf der kompilierten Datei wird ein Testgraph aufgebaut und ausgegeben. Die Ausgabe soll nach Abschluss der Aufgabe wie folgt aussehen:

```
$ ./graph
-----------------------------------------
Vertex Name: Vertex1
Input Edges:
Edge ID: 2
Output Edges:
Edge ID: 0
Edge ID: 1

Vertex Name: Vertex2
Input Edges:
Edge ID: 0
Output Edges:


Vertex Name: Vertex3
Input Edges:
Edge ID: 1
Output Edges:
Edge ID: 2

-----------------------------------------
```

Programmieren und vervollständigen sie in `graph.cpp` die Methoden und Konstruktoren von Vertex, Edge und Graph, die nicht bereits in `graph.h` inline definiert wurden!

Hinweise:

- Weitere Erläuterungen finden Sie in den Kommentaren in `graph.h`.
- Die ID eines Vertex ist sein Index in `Graph::vertices`. Die ID einer Edge ist ihr Index in `Graph::edges`.
- Die Member `inEdgeIds` und `outEdgeIds` der Klasse `Vertex` enthalten jeweils die IDs der ankommenden bzw. abgehenden Kanten. In `outEdgeIds` und `inEdgeIds` können die IDs unsortiert abgelegt werden
- Bei den Konstruktoren ist darauf zu achten, dass alle Datenmember initialisiert werden.

Verwenden Sie dafür Initialisierungslisten! Dies bedeutet nicht, dass alle Member zwingend im Konstruktor bereits ihren finalen Wert annehmen müssen.