IMS
Institut für Mikroelektronische Systeme
Leibniz Universität Hannover

Programmierpraktikum Technische Informatik (C++)
Aufgabe 3

Hinweise
Abgabe: Stand des Git-Repositories am 10.5.2022 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. Die Lösungen committen Sie bitte in Ihr lokales Repository und pushen sie in Ihr Repository auf dem Gitlab-Server.

Teilaufgabe 1 (2 Punkte)

Ziel der Aufgabe ist es, Eingaben des Spiels Schere, Stein, Papier entgegenzunehmen, auszuwerten und das Ergebnis der Runde auszugeben (s.a. `https://en.wikipedia.org/wiki/Rock_paper_scissors`). Die Dateien zur Aufgabe befinden sich im Verzeichnis rps. Über die Kommandozeile sollen die Eingaben der beiden Spieler übergeben werden. Ein Aufruf soll dabei so aussehen können:

```
$ ./rps rock paper
Player1: rock, Player2: paper, Winner: Player2
```

a) Implementieren Sie die Funktion `Move parseInput(const std::string& input)`! Diese Funktion soll entsprechend den als `std::string` übergebenen Zug als enum `Move` zurückgeben. Wenn keine gültige Eingabe übergeben wurde, soll der Move `Error` zurückgegeben werden. Als gültige Eingaben gelten `rock, paper und scissors` jeweils nur in Kleinbuchstaben.

Hinweise:

- Die Struktur `Move` ist in der Datei `rps.h` vorgegeben.
- Die Dokumentation von `enum class` können sie unter `https://en.cppreference.com/w/cpp/language/enum` nachlesen.

b) Implementieren sie die Funktion `Result rockPaperScissors(const std::string& p1, const std::string& p2)` und vervollständigen sie die `main`-Funktion derart, dass das Spiel funktionsfähig ist. Die Funktion `rockPaperScissors` bekommt die Eingaben aus der Komandozeile

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

übergeben, wertet diese gemäß der Regeln des Spieles dann aus und gibt das Ergebnis zurück. Dabei soll der Fall eines Siegers, eines Unentschiedens oder fehlerhafter Eingabe ermittelt werden. Beispiel einer fehlerhaften Eingabe:

```
$ ./rps brunnen rock
Invalid Input
```

In der `main`-Funktion soll das Ergebnis dann sinnvoll auf der Standardausgabe `std::cout` ausgegeben werden.

Hinweise:

- Die Kommandozeilenparameter werden in dem `std::vector<std::string> args` gespeichert und können dort auch abgerufen werden.
- In der Funktion `rockPaperScissors` sollen die Parameter zuerst über die Funktion `parseInput` in ein `Move` umgewandelt und diese dann ausgewertet werden.

Teilaufgabe 2 (2 Punkte)

Unter Unix existiert das Kommandozeilenprogramm `diff`, welches die Unterschiede zweier Dateien aufführt. Es gibt diejenigen Zeilen und Zeilennummern der Dateien aus, die sich unterscheiden. Schreiben Sie ein Programm, das diese Funktion in einfacher Form nachbildet.

a) Lesen Sie zuerst beide Dateien zeilenweise ein und speichern Sie diese in jeweils einem Vektor.

b) Implementieren Sie die Methode compareFilesLineByLine! Es sollen beide Dateien zeilenweise miteinander verglichen werden. Unterscheiden sich die Zeilen, sollen beide Zeilen ausgegeben werden. Dafür soll das folgende Format verwendet werden:

```
<output> = <line1> <line2> "\n"
<line-1> = "<<<" <linenum> "<<<" <line-file-1> "\n"
<line-2> = ">>>" <linenum> ">>>" <line-file-2> "\n"
```

`<linenum>` steht hierbei für die aktuelle Zeilennummer, `<line-file-1>` für die entsprechende Zeile der ersten Datei und `<line-file-2>` für die entsprechende Zeile der zweiten Datei.

Da eine Datei länger sein kann als die andere, sollen zum Schluss alle überschüssigen Zeilen in der längeren Datei in derselben Formatierung wie unterschiedliche Zeilen ausgegeben werden. Ist die erste Datei länger wird den

Zeilen also <<<*linenum*<<< vorangestellt, ansonsten >>>*linenum*>>>.

Der Programmaufruf könnte wie folgt aussehen:

```
./newdiff file1.txt file2.txt
<<<3<<< Test-3
>>>3>>> Test 3

<<<5<<< Test.5
>>>5>>> Test 5

>>>7>>> Test 7
>>>8>>> Test 8
```

Teilaufgabe 3 (1 Punkt)

Unter Unix existiert das Kommandozeilenprogramm `word count` mit dem Aufruf `wc` *`filename`*. Dieses gibt für die unter *`filename`* angegebene Datei die Anzahl der Zeilen, Wörter und Buchstaben, sowie den Dateinamen aus. Schreiben Sie in `newWc/newwc.cpp` ein Programm, das diese Funktionalität nachbildet! Die Ausgabe ihres Programmes soll dabei dem Orginal entsprechend die Syntax *`linecount wordcount bytecount filename`* einhalten. Für die vorgegebene Datei `hamlet.txt` würde Aufruf und Ausgabe also wie folgt aussehen:

```
$ ./newwc hamlet.txt
5302 32230 178848 hamlet.txt
```

Zeilen werden dabei durch einen Zeilenumbruch (`'\n'`) getrennt, Wörter durch Leerzeichen (`' '`). Leere Zeilen zwischen aufeinanderfolgenden Zeilenumbrüchen sollen dabei mitgezählt werden, leere Wörter zwischen aufeinanderfolgenden Leerzeichen allerdings nicht.

a) Implementieren Sie eine Funktion `getWordCount`, die einen Input-Stream (`std::istream`) und einen Namen übergeben bekommt. Diese soll die Anzahl der Zeilen, Wörter und Zeichen im übergebenen Input-Stream zählen und das Ergebnis als `WCResult` zurückgeben.
Hinweise:

- Zur Trennung von Wörtern inerhalb einer Zeile müssen an dieser Stelle lediglich Leerzeichen berücksichtigt werden.
- `std::getline` überspringt das angegebene Trennzeichen ohne es in den erzeugten String hinzuzufügen. Je nach Vorgehen muss dies eventuell beim für das Zählen der Zeichen beachtet werden.
- In der Datei `newwc.cpp` sind zwei `split`-Funktionen bereits definiert. Diese

stammen aus der Lösung zum letzten Übungsblatt und können Ihnen möglicherweise bei der Lösung helfen.

b) Vervollständigen Sie die Main-Funktion mit Hilfe der von Ihnen geschriebenen Funktion derart, dass das Programm die oben beschriebene Funktionalität erfüllt.
Hinweise:

- Die Memberfunktion `output` von `WCResult` schreibt die enthaltenen Daten in der gewünschten Formatierung in den übergebenen Output-Stream.
- `wc` trennt Wörter auch bei Tabulatorzeichen und zählt die Anzahl der Zeilenumbrüche statt der Anzahl der Zeilen. Aus diesem Grund können die von `newwc` ausgegebenen Zahlen für bestimmte Eingaben von durch `wc` generierten Werten abweichen. Für `hamlet.txt` ist dies allerdings nicht der Fall.

Bonusaufgabe (1 Punkt)

Achtung: Diese Aufgabe ist aufwändiger als ihre Punktzahl vermuten lässt.

Bisher deckt das Programm `newwc` nur einen Teil der Funktionalität von `wc` ab. Dies soll nun geändert werden. Implementieren Sie dazu die folgenden Features von `wc` auch für `newwc`:

a) Wird beim Aufruf von `wc` keine Datei angegeben, so wertet das Programm die Standardeingabe bis zum Auftreten des End-Of-File-Zeichens aus und gibt dafür die Anzahl Zeilen, Wörter und Zeichen ohne Angabe eines Namens aus.
Hinweis: End-Of-File kann in der Kommandozeile mittels `Ctrl D` eingegeben werden.

b) Werden beim Aufruf von `wc` mehrere Dateinamen als Argumente angegeben, so werden die entsprechenden Informationen jeweils in einer Zeile pro Datei ausgegeben. In diesem Fall (und nur dann) wird zusätzlich eine Extrazeile für die Summe der Zeilen, Wörter und Zeichen über alle Dateien ausgegeben.
Beispiel:

```
$ ./newwc hamlet.txt macbeth.txt
5302  32230 178848 hamlet.txt
3196  20146 125897 macbeth.txt
8498  52376 304745 total
```

c) Neben Leerzeichen interpretiert `wc` auch Tabulatorzeichen (`'\t'`) und Carriage-Return (`'\r'`) als Trenner für Wörter.
Hinweise:

- Die Memberfunktion `find_first_of` von `std::string` könnte zu diesem Zweck nützlich sein.

- Beachten Sie, dass leere Wörter zwischen aufeinanderfolgenden Trennzeichen (Leerzeichen, Tabulatoren oder Zeilenumbrüchen) nicht gezählt werden. Dies gilt insbesondere auch für leere Wörter am Ende einer Zeile.
- Sie können die Datei `bonus.txt` zur Überprüfung ihrer Implementation verwenden. Die Ausgabe sollte mit der von `wc` übereinstimmen.