IMS
Institut für Mikroelektronische Systeme
Leibniz Universität Hannover

Programmierpraktikum Technische Informatik (C++)
Aufgabe 04

Hinweise
Abgabe: Stand des Git-Repositories am 11.5.2023 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 (1,5 Punkte)

Beantworten Sie folgende Fragen in einer von Ihnen erzeugten Datei im Repo!
Hinweis: `int *p` ist identisch mit `int* p`.

a) Was gibt der folgende Code aus? Begründen Sie ihre Antwort!

```
std::vector<int> v = {10, 9};
int* p1 = &v[0];
*p1    = --*p1 * *(p1 + 1);
std::cout << v[0]<<", "<<v[1] << std::endl;
```

b) Erklären Sie jede der folgenden Definitionen! Ist eine davon illegal? Wenn ja, warum? `int i = 0;`

   a) `short* p1 = &i;`

   b) `int*  p2 = 0;`

   c) `int*  p3 = i;`

   d) `int*  p4 = &i;`

c) Sei `p` ein Pointer auf `int`. Unter welcher Bedingung wird *`Code1`* und unter welcher *`Code2`* ausgeführt? Welche Probleme können dabei auftreten?

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

```
if (p) {Code1}
if (*p) {Code2}
```

d) Es sei ein Pointer `p` gegeben. Kann man herausfinden, ob `p` auf ein gültiges Objekt zeigt? Wenn ja, wie? Wenn nein, warum nicht?

Teilaufgabe 2 (1,5 Punkte)

Welche der folgenden jeweils einzelnen `unique_ptr`-Deklarationen sind illegal oder führen möglicherweise im Folgenden zu Programmfehlern? Erklären Sie, worin die Probleme jeweils bestehen!

```
double e   = 2.7182;
double* dp = &e;
double* dp2 = new double(3.1415);
double* dp3 = new double(1.618);
double& dr1 = *dp3;
std::vector<double> v = {1.5, 2.5};
using DoubleP = std::unique_ptr<double>;
```

a) `DoubleP pd0(std::make_unique<double>(3.1415));`

b) `DoubleP pd1(dp2);`

c) `DoubleP pd2(dp);`

d) `DoubleP pd3(pd1.get());`

e) `DoubleP pd4(&e);`

f) `DoubleP pd5(e);`

g) `DoubleP pd6(pd0);`

h) `DoubleP pd7(&v[0]);`

i) `DoubleP pd8(&dr1);`

Teilaufgabe 3 (2 Punkte)

Unter Unix existiert das Kommandozeilenprogramm `paste` mit dem Aufruf `paste` *`filename {filename}`*. Dieses fügt die unter *`filename`* angegebenen Dateien horizontal zusammen und gibt das Ergebnis aus. Beispielsweise enthält die erste Zeile des Ergebnisses die erste Zeile jeder angegebenen Dateien, wobei Inhalte verschiedener

Dateien durch einen Tabulator getrennt werden. Für die vorgegebenen Dateien `names.txt` und `numbers.txt` würden Aufruf und Ausgabe also wie folgt aussehen:

```
./newpaste names.txt numbers.txt
Mark Smith      555-1234
Bobby Brown     555-9876
Sue Miller      555-6743
Jenny Igotit    867-5309
                007-0815
```

In `newPaste/examples.txt` finden Sie einige Beispielausgaben für verschiedene Aufrufe des originalen `paste`, die Ihr Programm abdecken sollte.
Schreiben Sie in `newPaste/newpaste.cpp` ein Programm, das diese Funktionalität nachbildet!

a) Schreiben Sie eine Hilfsfunktion `readFile`, die einen Input-Stream (`std::istream`) übergeben bekommt. Diese soll den Inhalt des Streams zeilenweise in einen Vektor (`std::vector<std::string>`) einlesen und diesen zurückgeben

b) Implementieren Sie eine Funktion `combineLines`! Diese Funktion soll die Zeilen mehrerer Zeilen-Vektoren entsprechend dem Verhalten von `paste` zu einem Ergebnisvektor kombinieren.
Hinweis: Wieviele Zeilen-Vektoren `combineLines` auf einmal verarbeitet, ist Ihnen freigestellt. Sie können entscheiden, ob `combineLines` alle Zeilen-Vektoren auf einmal kombiniert, oder ob `combineLines` jeweils nur ein oder zwei Zeilen-Vektoren verarbeitet und in der `main`-Funktion in einer Schleife für alle Eingabedaten aufgerufen wird.

c) Vervollständigen Sie die Main-Funktion mit Hilfe der von Ihnen geschriebenen Funktionen derart, dass das Programm die oben und in der `examples.txt` beschriebene Funktionalität erfüllt. Beschränken Sie dabei die Anzahl möglicher Parameter nicht auf zwei, sondern ermöglichen Sie es, beliebig viele Dateinamen übergeben zu können. Weiterhin sollte Ihr Programm auch die in der `examples.txt` gezeigte Fehlermeldung ausgeben, falls eine Datei nicht gefunden werden kann.

   Hinweise:

   - Um festzustellen, ob eine Datei gefunden werden konnte, können Sie die Methode `is_open` des `std::ifstream` verwenden.
   - Für Aufrufe mit mehr als zwei Dateinamen muss die Ausgabe Ihres Programms nicht mit der des Originals übereinstimmen.