C/C++ Programmierung
Übungsblatt 10

Florian Oetke

Hinweis zum Arbeiten mit Dateien
Das Verzeichnis in dem Ihr Programm nach Dateien sucht, die Sie mit relativen Pfaden angegeben haben, hängt u.a. von ihrer IDE und der Projektkonfiguration ab.

Wenn Sie sich unsicher sind oder Dateien nicht gefunden werden, können Sie für diese Aufgaben entweder mit absoluten Pfaden arbeiten oder sich das entsprechende Verzeichnis mit `std::cout << std::filesystem::current_path()<< "\n";` ausgeben lassen (benötigt `#include <filesystem>`).

1 Klassenhierarchie für mathematische Funktionen (Teil 2)

In dieser Aufgabe soll die `Function`, sowie die daraus abgeleiteten Klassen, aus dem letzten Übungsblatt um die Möglichkeit erweitert werden die Ausdrücke auszugeben.

Ergänzen Sie dazu die Basisklasse `Function` um eine rein virtuelle `print`-Methode, die eine Referenz auf einen `std::ostream` als Parameter bekommt. Implementieren Sie diese Methode anschließend in den abgeleiteten Klassen so, dass der Ausdruck, wie in der Beispielausgabe, auf dem Stream ausgegeben wird.

Darüber hinaus sollte es für Ihre Klasse einen `operator<<` geben, der wie in der Vorlesung gezeigt definiert ist und Objekte der Basisklasse auf einem Stream ausgibt, in dem die `print`-Methode aufgerufen wird.

Als Basis für die Implementierung sollten Sie Ihre Lösung vom letzten Übungsblatt verwenden und die `test`-Funktion wie folgt abändern, bzw. aus dem Handout übernehmen:

```
1 void test(const Function& e) {
2     std::cout << e << "\n";
3     std::cout << "    x= 2 : " << e.calculate( 2.f) << "\n";
4     std::cout << "    x=-2 : " << e.calculate(-2.f) << "\n\n";
5 }
```

Die Ausgabe Ihres Programms sollte am Ende wie folgt aussehen:

```
1x + 3
    x= 2 : 5
    x=-2 : 1

1x^2 + 2x + -1
    x= 2 : 7
    x=-2 : -1
```

2 Eine einfache Binärdatei schreiben und lesen

Im Handout finden Sie eine einfache Binärdatei, die einige Zeichenketten enthält, wobei das erste Byte vor jeder Zeichenkette jeweils die Anzahl der folgenden Zeichen enthält.

Implementieren Sie die `read()` Funktion, welche die Binärdatei öffnen und solange einlesen soll, bis das Ende der Datei erreicht wurde. Lesen Sie jeweils zuerst das nächste Byte mit der unformatierten Eingabe ein, das die Länge der folgenden Zeichenkette angibt. Lesen Sie anschließend, wie in der Vorlesung gezeigt, entsprechend viele Zeichen/Bytes ein und geben Sie die so eingelesene Zeichenkette mit `std::cout` aus.

Implementieren Sie anschließend auch die `write()` Funktion, die eine Binärdatei schreiben soll, die identisch zu der aus dem Handout ist.

3 Wetterdaten Eingabe

In dieser Aufgabe soll ein einfaches Programm geschrieben werden, das Wetterdaten von der Standardeingabe liest und sie zu einer CSV-Datei hinzufügt. Hierbei besteht ein Eintrag in der CSV-Datei aus der Jahreszahl, dem Tag im jeweiligen Jahr, der durchschnittlichen Temperatur an diesem Tag in °C und der Niederschlagsmenge in $\frac{L}{m^2}$.

Die Interaktion mit dem Benutzer sollte dabei wie im folgenden Beispiel aussehen, wobei alle Zeilen, die mit `>` anfangen, vom Benutzer eingegeben wurden. Das heißt, Sie sollen immer die gesamte nächste Zeile einlesen, aus dieser dann (z.B. mit `std::istringstream`) die einzelnen Werte bestimmen und ggf. eine Fehlermeldung für den ersten fehlenden oder ungültigen Wert ausgeben. Es sollen beliebig viele Einträge eingegeben werden können und das Programm soll erst abbrechen, wenn eine leere Zeile eingegeben wurde oder das Lesen von `std::cin` fehlgeschlagen ist.

```
Please input:
year day_of_year average_temperature precipitation

> 2020 12   2.5 0
> 2020 13 2.0 5
> 2020 14
Invalid temperature
> 2020 14 2.8
Invalid precipitation
> 2019 103 1.0 3.0
>
Done
```

Alle eingegebenen vollständigen/gültigen Einträge sollen sofort in eine CSV-Datei `"weather.csv"` geschrieben werden, welche Sie direkt beim Start des Programms öffnen/anlegen, wobei die neuen Einträge hinten an die Datei angehängt werden sollten, damit bestehende Einträge in der Datei erhalten bleiben. In der CSV-Datei soll ein Eintrag pro Zeile stehen, wobei die einzelnen Werte der Einträge mit `;` getrennt sind. Der erste Eintrag soll, wie im Beispiel unten, der Header der Tabelle sein, welchen Sie ggf. ebenfalls noch in die Datei schreiben müssen, falls diese zu Beginn Ihres Programms noch leer ist oder neu angelegt wurde[1]. Ihre CSV-Datei sollte also nach der obigen Eingabe so aussehen:

```
year; day_of_year; average_temperature; precipitation
2020;12;2.5;0
2020;13;2;5
2019;103;1;3
```

Als Basis für die Implementierung können Sie den im Handout bereitgestellten Programmcode verwenden.

[1] Ob die Datei aktuell leer ist, können Sie z.B. herausfinden, indem Sie `tellp()` aufrufen und prüfen, ob der Wert > 0 ist, während Sie sich am Ende des Streams befinden (z.B. weil Sie ihn u.a. mit `std::ios::app` geöffnet haben).

4 Wetterdaten Ausgabe

In dieser Aufgabe sollen Sie ein Programm schreiben, das die CSV-Dateien der letzten Aufgabe einliest, daraus jährliche Statistiken generiert und diese als Tabelle ausgibt.

Die erste Funktion, die Sie hierfür schreiben müssen ist `read_data()`, welche den Pfad zur CSV-Datei als Parameter bekommt und eine Map (`Weather_data` im Handout) mit den enthaltenen Daten zurückliefert.

Das Format der CSV-Dateien entspricht dem der letzten Aufgabe und Sie finden im Handout auch nochmal eine Beispieldatei, die Sie als Eingabe für Ihr Programm verwenden können. Beim Einlesen der Datei können Sie davon ausgehen, dass diese nur vollständige und gültige Einträge enthält und brauchen dies nicht zu prüfen. Beachten Sie allerdings, dass die Datei in der ersten Zeile den Header mit den Spaltennamen enthält, den Sie beim Einlesen überspringen müssen.

Falls die Datei mehrere Einträge für denselben Tag enthält, sollen die Niederschlagswerte alle Einträge aufaddiert und die Temperatur des Eintrags übernommen werden, der später in der Datei steht (bereits in `Weather::update` vorgegeben).

Basierend auf den eingelesenen Werten sollen Sie anschließend in der Funktion `print_statistics()` für jedes Jahr, das in der Datei vorgekommen ist, die durchschnittliche Temperatur (d.h. den Durchschnitt aller Tageswerte), den Gesamtniederschlag und die Anzahl der Tage, an denen es Niederschlag gab, berechnen und als Tabelle ausgeben.

Das Format der Ausgabe steht Ihnen frei, Sie können sich aber an dem Beispiel aus der Vorlesung (S.24) orientieren. Um die Ausgabe der `float`-Zahlen genauer zu steuern, können Sie sich die unterschiedlichen Zahlendarstellungen und `std::setprecision` zunutze machen[2].

Für die Beispieldatei im Handout könnte die Ausgabe z.B. so aussehen:

```
| Year | Avg. Temp. | Precipitation | Rain days |
================================================
| 2019 |    7.80    |     87.63     |     12    |
| 2020 |   18.07    |     30.00     |      1    |
```

[2]Achtung: Die Bedeutung des Parameters von `std::setprecision` unterscheidet sich abhängig vom gewählten Darstellungsformat. Bei `std::defaultfloat` wird damit die Gesamtzahl der Stellen vor und nach dem Komma festgelegt, bei `std::fixed` bezieht sich das dagegen nur auf die Stellen hinter dem Komma.