Übungsblatt 2:
C++ Libraries, Git & CMake

Sommersemester 2016

Programmiertechniken

Prof. Dr. L. Pollet

J. Greitemann, D. Hügel, J. Nespolo, T. Pfeffer

---

1) Vektoren der C++ Standardbibliothek `std`

In dieser Aufgabe sollen Sie sich mit der Behälterklasse `<vector>` der C++ Standardbibliothek vertraut machen. Dazu wird die erste Aufgabe des letzten Übungsblattes mit Hilfe eines `std::vector` Behälters gelöst.

(a) Es gibt verschiedene Deklarations- und Initialisierungsmöglichkeiten für einen `std::vector`. Deklarieren und initialisieren Sie einen `std::vector<int>` v mit 10 Einträgen $v_i = i$ auf unterschiedliche Weise.

(b) Da die Behälterklasse `<vector>` eine abstrakte Definition für beliebige Datenstrukturen ist, beinhaltet die Standardbibliothek keine Funktion zur Ausgabe des Vektorinhaltes. Benutzen sie die Standard I/0 Funktionen um eine eigene Funktion `print_vector` zu implementieren, welche für einen beliebigen `std::vector<double>` den Vektorinhalt in das Terminal ausgibt.

(c) Lesen sie nun die Temperaturdaten `https://db.tt/kET7HUlZ` wie im letzten Übungsblatt ein und speichern sie die Daten in einem `std::vector`. Erweitern sie die Länge des Vektors dynamisch mit jedem neuen Datenpunkt. Berechnen Sie nun die Durchschnittstemperatur und Standardabweichung der Daten mit Hilfe von Bibliotheksfunktionen der Standardbibliothek und geben Sie die Daten im Terminal aus.

(d) Die dynamische Erweiterung der Länge des Vektors beim Einlesen von großen Datenmengen ist nicht effizient, da der Vektorinhalt immer wieder auf neu alokierten Speicher kopiert werden muss. Initialisieren Sie eine `std::vector<double>` mit 1000 Einträgen und finden Sie die Anzahl der Einträge heraus um welche der Vektor erweitert werden kann ohne dabei kopiert werden zu müssen. Wie können sie das dynamische erweitern von Vektoren effizient gestalten, wenn Sie die maximal zu erreichende Größe des Vektors kennen.

(e) Initialisieren Sie zwei Vektoren x und y. Wie ist es moeglich effizient die Namen der Vektoren zu tauschen?

2) `<algorithm>` der C++ Standardbibliothek

Das Suchen eines Elements in einer Liste L ist eine der Standardaufgaben, welche in vielen Anwendungen benötigt wird. Bei der naivsten Implementation skaliert die Rechenzeit linear mit der Größe N der Liste. Kann eine Ordnung $L_i < L_j$ der Listenelemente definiert werden, so ist es möglich einen binären Suchalgorithmus zu verwenden und damit die Rechenzeit auf $\log(N)$ zu reduzieren.

1. Bestimmen Sie die niedrigste und höchste Temperatur.
2. Verwenden Sie `std::sort` aus der C++ Standardbibliothek um die Temperaturwerte aus Aufgabe 1) zu sortieren.
3. Nun kann die Funktion `std::binary_search` auf die sortierten Temperaturwerte angewendet werden um festzustellen ob sich ein Temperaturwert in der Liste der Tagestemperaturen befindet.
4. Mit der Standardkonfiguration der Function `std::binary_search` ist es nicht möglich den Tag herauszufinden an welchem die von ihnen spezifizierten Temperatur gemessen wurde. Dazu muss auch die Permutation der Elemente der Ausgangsliste bekannt sein. Speichern Sie die Temperaturwerte mit

dem zugehörigen Tag in ein `std::pair<int,double>` und implementieren sie eine Vergleichsfunktion welche die Paare ordnet. Schreiben Sie sodann eine Funktion welche es ihnen ermöglicht für eine vorgegebene Temperatur den Tag im Jahr 2015 herauszufinden an dem diese Temperatur gemessen wurde.

5. Löschen Sie die niedrigste und höchste Temperatur aus der Liste und berechnen Sie erneut die Durchschnittstemperatur.