IMS
Institut für Mikroelektronische Systeme
Leibniz Universität Hannover

Programmierpraktikum Technische Informatik (C++)
Programmiertutorium Vorlesung 9

Pullen Sie von ppti-common. Sie finden dann in ihrem Repository den Ordner `tutorials/Lecture09/`, in dem die folgenden Aufgaben zu bearbeiten sind.
Hinweis: Im Gegensatz zu den Übungsaufgaben sind für die Tutorien nicht unbedingt alle für die Lösung notwendigen Header bereits inkludiert.

Teilaufgabe 1

Implementieren Sie eine Funktion `alternateSum`, die einen `std::vector<int> vec` übergeben bekommt und die alternierende Summe seiner Elemente zurückgibt! Die alternierende Summe ist dabei wie folgt definiert:

$$altSum = \sum_{k=0}^{vec.size()} (-1)^k * vec[k] = vec[0] - vec[1] + vec[2] - vec[3] + vec[4] - vec[5] + ...$$

Verwenden Sie zur Implementation `std::accumulate`!

Teilaufgabe 2

Schreiben Sie eine Templatefunktion `sortIndirect`! Diese Funktion soll zwei beliebige Iteratoren gleichen Typs und einen (beliebigen) Prädikatsfunktor übergeben bekommen und die Elemente des durch die beiden Iteratoren beschriebenen Bereichs indirekt sortieren. Indirekt sortieren bedeutet in diesem Fall, dass die Elemente des Bereichs pointerartige Konstrukte (Pointer, Iteratoren, Smartpointer,...) sind und diese anhand der Reihenfolge der Werte, auf die sie zeigen, sortiert werden sollen. Verwenden Sie zum Vergleich der dereferenzierten Werte der Pointer die übergebene Prädikatsfunktion!

Den Sortieralgorithmus selbst sollen Sie nicht neu implementieren.

Teilaufgabe 3

Schreiben Sie die Templatefunktion `zip`, die die Werte zweier durch Iteratoren angegebener Bereiche paarweise zu `std::pair` zusammenfasst! Diese Funktion soll die folgenden

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

Parameter haben:

- `begin1`: Iterator auf den Beginn des ersten Bereichs
- `end1`: Iterator auf das Ende des ersten Bereichs
- `begin2`: Iterator auf den Beginn des zweiten Bereichs
- `beginOut`: Iterator auf den Beginn des Zielbereichs, in den die Paare geschrieben werden sollen

Rückgabewert der Funktion soll ein Iterator hinter den letzten in den Zielbereich geschriebenen Wert sein. Verwenden Sie zur Implementation den Algorithmus `std::inner_product`!

Teilaufgabe 4

Schreiben Sie eine Klasse `LinearInterpolator`, die eine lineare Interpolation zwischen zwei Werten durchführen soll! Diese Klasse soll im Konstruktor zwei Werte vom Typ `double` erhalten, die Start und Ende des zu interpolierenden Bereichs angeben. Desweiteren soll die Klasse einen überladenen Funktionsoperator besitzen, dem ein `double` zwischen `0.0` und `1.0` übergeben wird. Dieser Wert gibt die Position auf der Strecke zwischen Start und Ende an, wobei `0.0` die Position des Starts und `1.0` die Position des Ziels ist. Die Funktion soll den durch lineare Interpolation zwischen Start und Ende berechneten Wert an der übergebenen Position zurückgeben.

Teilaufgabe 5

Schreiben Sie eine Funktion `makeInterpolator`, die zwei `double`-Werte übergeben bekommt und eine Lambdafunktion zurückgibt, die eine lineare Interpolation zwischen den beiden Werten an einer gegebenen Position berechnet! Das Verhalten der Lambdafunktion bzgl. des Funktionsaufrufs soll dabei dem einer Instanz von `LinearInterpolator` entsprechen!

Teilaufgabe 6

Schreiben Sie die Klasse `multiInterpolator`! Diese Klasse soll eine stückweise lineare Interpolation über einen beliebig großen Wertebereich durchführen. Sowohl die Position als auch der dazugehörige Wert sollen als `double` vorliegen. Die Klasse soll das folgende Interface unterstützen:

- Eine Methode `addValue`, die eine Position und einen Wert übergeben bekommt und diese dem Interpolationsbereich hinzufügt

- Einen überladenen Funktionsoperator, der eine Position übergeben bekommt und den dazugehörigen stückweise linear interpolierten Wert zurückgibt. Für Positionen außerhalb des abgedeckten Bereichs soll dabei der Wert an der ersten bzw. letzten gespeicherten Position ausgegeben werden. Beachten Sie, dass die Position hier nicht auf `0.0` bis `1.0` normiert ist, sondern eine direkt mit den hinzugefügten Positionen vergleichbare Position angibt.

Hinweis: Sinnvollerweise sollten die Positionen mit ihren verknüpften Werten nach den Positionen sortiert in einem Container abgelegt werden.