Programmierkurs IPK SS23
Andreas Naumann
IWR, Universität Heidelberg

Übung 4
Abgabedatum —
ipk-exercises:2023-ss-1-ga451146

Übung 4

Aufgabe 1: std::vector

Im folgenden lernen Sie den wichtigsten Container der C++-Standardbibliothek kennen: `std::vector<T>`, eine indizierte Liste mit Einträgen vom Typ `T`. `T` kann hierbei ein (fast) beliebiger Datentyp sein, z.B. `int` oder `double`. Einen `std::vector` können Sie auf verschiedene Weisen anlegen:

```cpp
#include <vector> // vector in Ihrem Programm verfügbar machen

int main(int argc, char** argv)
{
  // Ein leerer vector für ganze Zahlen
  std::vector<int> v1;
  // Ein vector für ganze Zahlen mit 10 Einträgen
  std::vector<int> v2(10);
  // Ein vector mit den Einträgen 3,8,7,5,9,2
  std::vector<int> v3 = {{ 3, 8, 7, 5, 9, 2 }};
}
```

Ein `vector` ist ein *Objekt* und hat sogenannte *member functions*, das sind spezielle Funktionen, die das Objekt verändern. Eine vollständige Referenz finden Sie auf der Website `cppreference.com`[1], die wichtigsten Methoden für diese Aufgabe sind:

```cpp
std::vector<int> v = {{ 3, 8, 7, 5, 9, 2 }};
// Gibt die Anzahl der Einträge zurück
std::cout << v.size() << std::endl; // 6
// Verändert die Länge der Liste
v.resize(42);
```

Um auf einen Eintrag des Vektors zuzugreifen, schreiben Sie den Index des Eintrags in eckigen Klammern hinter den Variablennamen. **Die Nummerierung der Einträge beginnt bei 0, nicht bei 1**. Um einen Eintrag zu verändern, weisen Sie dem Eintrag einfach einen neuen Wert zu:

```cpp
// Zugriff auf einzelne Einträge - Index ist 0-basiert!
std::cout << v[2] << std::endl; // 7
v[0] = v[0] * 2;
std::cout << v[0] << std::endl; // 6
```

Aufgaben:

(a) Legen Sie einen `vector<double>` mit jeder der oben beschriebenen Methoden an und geben Sie jeweils alle Einträge mit einer `for`-Schleife aus. Welchen Wert haben Einträge, für die Sie keinen expliziten Wert angegeben haben?

(b) Schreiben Sie eine Funktion, die den grössten und den kleinsten Wert in einem Vektor findet und als `std::pair` zurückgibt (erst den kleinsten, dann den größten Wert). Testen Sie die Funktion mit verschiedenen Vektoren.

[1] http://en.cppreference.com/w/cpp/container/vector

(c) Schreiben Sie eine Funktion `std::vector<double> reversed(const std::vector<double>& v)`, die einen Vektor mit Einträgen $x_0, x_1, \ldots, x_{n-1}$ als Parameter nimmt und einen neuen Vektor mit den Einträgen in umgekehrter Reihenfolge $x_{n-1}, x_{n-2}, \ldots, x_0$ zurückgibt. Testen Sie die Funktion mit verschiedenen Vektoren, insbesondere auch mit einem leerem.

(d) Schreiben Sie eine Funktion, die alle Einträge in einem `std::vector<double>` auf ganze Zahlen rundet und diese dann wieder im **selben** Vektor speichert. Zum Runden von Zahlen verwenden Sie folgendes:

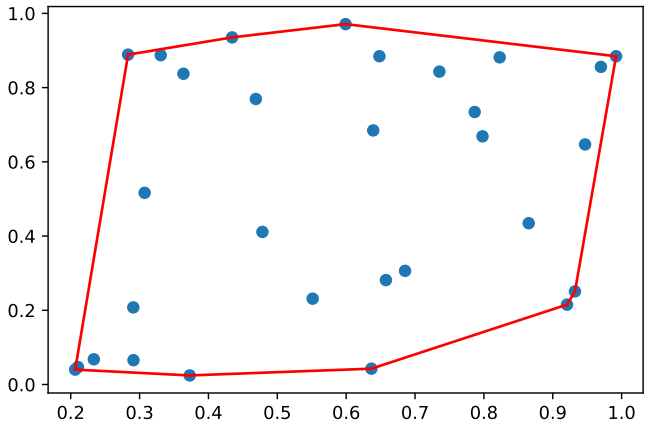
```cpp
#include <cmath>

int main()
{
  double x = 2.71;
  double x_rounded = std::round(x);
}
```

Testen Sie die Funktion mit verschiedenen Vektoren.

(e) Schreiben Sie eine Funktion, das die Reihenfolge der Einträge in einem Vektor umkehrt, aber das Ergebnis nun im **selben** Vektor speichert. Verwenden Sie zum Vertauschen einzelner Einträge die Funktion `std::swap(a,b)`. Lesen Sie auf cppreference.com[2] nach, was diese Funktion macht und welche `#include`-Anweisung Sie benötigen. Testen Sie die Funktion mit leeren Vektoren sowie mit welchen, die eine gerade bzw. eine ungerade Grösse haben.

Aufgabe 2: Konvexe Hülle

Die konvexe Hülle einer Menge von Punkten $\{x_i \in \mathbb{R}^2\}$ ist definiert als das kleinste konvexe Polygon, in dem alle Punkte x_i enthalten sind [3]. Im folgenden Beispiel sehen Sie eine Menge von Punkten mit der konvexen Hülle in rot:



Schreiben Sie ein Programm, das für eine Menge von Punkten die konvexe Hülle berechnet. Das Programm soll die Punkte aus einer Datei, mit der folgenden Form einlesen:

```
0.3 0.7
1.2 3.4
9.3 4.8
2.8 7.2
```

Dabei steht in jeder Zeile ein Punkt mit seiner x- und y-Koordinate. Auf der Übungsseite finden Sie eine Beispiel-Datei in diesem Format[4].

[2] http://en.cppreference.com/w/cpp/algorithm/swap
[3] https://de.wikipedia.org/wiki/Konvexe_Hülle
[4] https://conan.iwr.uni-heidelberg.de/

(a) Schreiben Sie eine Funktion

```
1 std::vector<std::array<double, 2>> read_points_from_file(std::string filename)
```

Diese Funktion soll die Punkte aus der Datei `filename` einlesen und die Punkte in einem `std::vector` von `std::array` zurückgeben. **Hinweis:** Dies lässt sich sehr elegant mit `std::ifstream` lösen. Sie dürfen davon ausgehen, dass die Datei keine Fehler enthält.

(b) Berechnen Sie die konvexe Hülle der Punkte. Verwenden Sie dafür den Graham-Scan [5] und gehen Sie in folgenden Schritten vor:

- Legen Sie die Funktion

```
1 void convex_hull(std::vector<std::array<double, 2>>& points)
```

an und führen Sie die folgenden Schritte in dieser Funktion aus.

- Sortieren Sie die Punkte nach ihren y-Koordinaten:

```
1 std::sort(points.begin(), points.end(), sort_by_y);
```

Dafür müssen Sie eine Funktion

```
1 bool sort_by_y(std::array<double, 2> a, std::array<double, 2> b)
```

implementieren, welche `true` zurückgibt falls $a < b$ im obigen Sortieralgorithmus gelten soll.

$$\texttt{a<b} = \begin{cases} \texttt{true}, \text{ falls } \texttt{a[1]<b[1]} \\ \texttt{true}, \text{ falls } \texttt{a[1]==b[1] and a[0]<b[0]} \\ \texttt{false}, \text{ ansonsten} \end{cases}$$

- Sei p_0 der erste Punkt. Sortieren Sie alle Punkte hinter p_0 nach dem Winkel zwischen dem Vektor $p_0 \to p$ und der x-Achse. Dies können Sie beispielsweise folgendermaßen erreichen:
  - Subtrahieren Sie p_0 von allen Punkten.
  - Sortieren Sie

```
1 std::sort(points.begin()+1, points.end(), sort_by_angle);
```

so, dass gilt:

$$\texttt{a<b} = \begin{cases} \texttt{true}, \text{ falls } \texttt{a[0]*b[1]-a[1]*b[0] > 0} \\ \texttt{true}, \text{ falls } \texttt{a[0]*b[1]-a[1]*b[0] == 0 and std::abs(a[0])> std::abs(b[0])} \\ \texttt{false}, \text{ ansonsten} \end{cases}$$

  - Addieren Sie p_0 auf alle Punkte

- Implementieren Sie einen der Graham-Scan Algorithmen von https://de.wikipedia.org/wiki/Graham_Scan. Dafür müssen Sie bestimmen, ob ein Punkt C links oder rechts der Geraden durch zwei andere Punkte A und B liegt, wozu Sie die folgende Relation verwenden können:

$$\begin{vmatrix} x_B - x_A & y_B - y_A \\ x_C - x_A & y_C - y_A \end{vmatrix} = (x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A) = \begin{cases} < 0 & C \text{ ist rechts von } AB \\ = 0 & C \text{ liegt auf } AB \\ > 0 & C \text{ ist links von } AB \end{cases}$$

(c) Schreiben Sie eine Funktion

[5] https://de.wikipedia.org/wiki/Graham_Scan

```
1 void read_points_from_file(std::string filename, std::vector<std::array<double, 2>>& points)
```

welche die Punkte `points` im selben Format, wie die oben beschriebene Eingabe in eine Datei schreibt. **Hinweis:** Hierfür bietet sich `std::ofstream` an.

Um herauszufinden, ob Ihr Programm richtig arbeitet, können Sie von der Website ein kleines Python-Skript herunterladen, mit der Sie die berechnete konvexe Hülle als PDF-Datei plotten können. Für dieses Skript müssen die Python-Pakete `numpy` und `matplotlib` installiert sein.

Mithilfe von

```
1 python3 convex-hull-plot.py points.txt hull.txt
```

können Sie damit einen Plot erstellen. Dabei enthält `points.txt` alle Punkte und `hull.txt` die Punkte, durch die die konvexe Hülle beschrieben wird.