Dr. Ole Klein, Dr. Steffen Müthing
IWR, Universität Heidelberg

Abgabe 20. November 2017

Allgemeine Hinweise:

- **Am Freitag, den 17.11. findet die Vorlesung statt!**
- Da der Compiler auf den Rechnern im Pool veraltet ist, haben wir dort jetzt eine aktuelle Version des clang-Compilers installiert. Um diese verwenden zu können, müssen Sie in Ihrem Home-Verzeichnis den Installationspfad des Compilers hinterlegen:
  1. Öffnen Sie die Datei `.bash_profile` in Ihrem Home-Verzeichnis mit einem Texteditor. Unter UNIX sind Dateien, die mit einem `.` beginnen, versteckt. Um versteckte Dateien anzuzeigen, können Sie auf der Kommandozeile den Befehl `ls -a` (für "all") verwenden.
  2. Fügen Sie in die Datei folgende Zeile ein und speichern Sie die Datei anschliessend:

     ```
     export PATH=/opt/llvm/5.0/bin:$PATH
     ```

     Damit können Sie jetzt Programme ausführen, die im Verzeichnis `/opt/llvm/5.0/bin` liegen.
  3. Melden Sie sich einmal ab und wieder neu an.

  Diese Schritte müssen Sie nur einmal machen, danach ist der Compiler zukünftig immer verfügbar. Um den neuen Compiler zu verwenden, schreiben Sie auf der Kommandozeile den Befehl **`ipkc++`** statt `g++` oder `clang++`.
- **WICHTIG:** Von jetzt an müssen Sie auf den Pool-Rechnern `ipkc++` als Compiler verwenden, ansonsten funktionieren einige Übungen nicht!
- Wenn Sie mit der virtuellen Maschine oder einem eigenen Linux-Rechner arbeiten, müssen Sie beim Kompilieren die Option `-std=c++14` verwenden:

```
g++ -std=c++14 -Wall ...
clang++ -std=c++14 -Wall ...
```

- Für die Fortgeschrittenen: Der neue Compiler im Pool ist clang 5.0, und wir verwenden die alternative Standardbibliothek libc++ aus dem LLVM-Projekt.

Übung 1 *Listen von Zahlen*

Bevor Sie mit dieser Übung anfangen: Aktualisieren Sie Ihren Compiler wie oben beschrieben!

Bisher haben Sie in Ihren Programmen nur mit einzelnen Zahlen gearbeitet; oft muss man in Programmen aber grössere Mengen von Daten speichern. Hierfür gibt es in C++ sogenannte *Container*. Im folgenden lernen Sie den wichtigsten dieser Container 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:

```
#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 *Methoden*, 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:

```
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:

```
// 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<int>` 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 beide auf die Standardausgabe schreibt und testen Sie die Funktion mit verschiedenen Vektoren.

(c) Schreiben Sie eine Funktion `std::vector<int> reverse(std::vector<int> 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 ein Programm, das alle Einträge in einem `std::vector<double>` auf ganze Zahlen rundet und diese dann wieder im gleichen Vektor speichert. Zum Runden von Zahlen verwenden Sie folgendes:

```
#include <cmath>

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

(e) Schreiben Sie ein Programm, das die Reihenfolge der Einträge in einem Vektor umkehrt, aber das Ergebnis nun im *selben* Vektor speichert.

(f) Verändern Sie Ihr Programm aus der vorherigen Teilaufgabe so, dass es zum Vertauschen einzelner Einträge die Funktion `std::swap(a,b)` verwendet. Lesen Sie auf `cppreference.`

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

`com`[2] nach, was diese Funktion macht und welche `#include`-Anweisung Sie benötigen.

(Grundlagen)

Übung 2 *Berechnung n-ter Wurzeln*

In einer der vorherigen Aufgaben haben Sie Potenzen von Zahlen berechnet, wobei der Exponent n stets eine ganze Zahl war. Hier wollen wir die Aufgabenstellung quasi umdrehen: wir suchen die n-te Wurzel einer positiven Zahl $q \in \mathbb{R}^+$, definiert durch

$$q^{1/n} = a \in \mathbb{R}^+ \iff a^n = \prod_{i=1}^{n} a = q.$$

Im Gegensatz zur Potenzaufgabe nutzen wir hier Fließkommazahlen, da die so definierte Wurzel $q^{1/n}$ für die meisten Kombinationen von q und n keine ganze Zahl mehr ist. Eine Formel, mit der man diese n-te Wurzel $q^{1/n}$ näherungsweise berechnen kann, ist

$$a_{k+1} \coloneqq a_k + \frac{1}{n} \cdot \left(\frac{q}{a_k^{n-1}} - a_k \right),$$

wobei a_0 eine erste Schätzung ist, z.B. einfach $a_0 \coloneqq 1$, und die Folge von Werten $a_0, a_1, a_2, a_3, \ldots$ immer bessere Näherungen für den echten Wert von $q^{1/n}$ produziert.

Alle folgenden Funktionen sollen testen, ob die Eingabe gültig ist. Bei einem Fehler schreiben Sie eine Meldung nach `std::cout` und geben ggf. 0 zurück.

(a) Schreiben Sie eine Funktion `double root_iterative(double q, int n, int steps)`, die $q^{1/n}$ näherungsweise berechnet. Dabei ist `steps` die Anzahl an Schritten (und damit Anzahl an Näherungen), die das Programm berechnen soll.

(b) Zum Berechnen einer Iteration $a_k \to a_{k+1}$ benötigen Sie die $(n-1)$-te Potenz von a_k. Eine passende Funktion haben Sie bereits geschrieben; Sie müssen lediglich darauf achten, dass sich die Datentypen von Ein- und Ausgabe geändert haben.

(c) Schreiben Sie eine Funktion `void test_root(double q, int n, int steps)`, die die Genauigkeit Ihrer Wurzelberechnung testet. Dazu soll die Funktion wie oben beschrieben eine Näherung $\tilde{a} \approx q^{1/n}$ berechnen, die Potenz $\tilde{a}^n$ bestimmen, und dann die folgenden Werte ausgeben: q, n, $\tilde{a}$, $\tilde{a}^n$ und $q - \tilde{a}^n$.

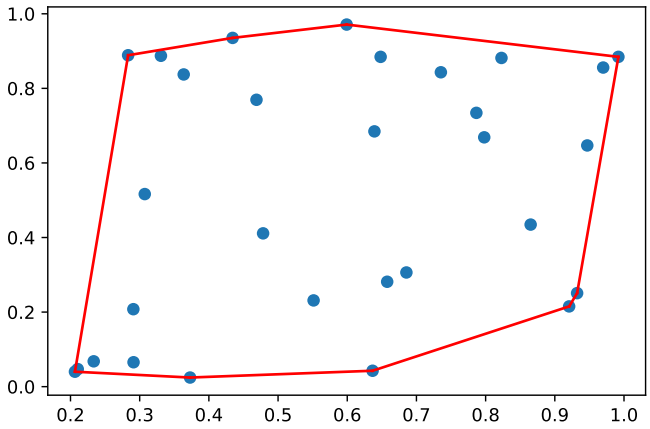
Testen Sie Ihr Programm für mehrere zehnstellige Ganzzahlen q als Eingabe, mit $n \in \mathbb{N}$ einstellig. Überprüfen Sie insbesondere, dass für $n = 1$ die Eingabe reproduziert wird ($q^1 = q$), und für $n = 2$ die gewohnte Quadratwurzel berechnet wird. Die Schrittzahl `steps` kann dabei in der Größenordnung von 100 gewählt werden.

(Grundlagen)

Übung 3 *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:

[2] `http://en.cppreference.com/w/cpp/algorithm/swap`
[3] `https://de.wikipedia.org/wiki/Konvexe_Hülle`



Schreiben Sie ein Programm, das für eine Menge von Punkten die konvexe Hülle berechnet. Das Programm soll die Punkte in folgendem Format von der Standardeingabe einlesen:

```
4
0.3 0.7
1.2 3.4
9.3 4.8
2.8 7.2
```

Hierbei gibt die erste Zeile der Datei die Anzahl der Punkte an, und in den nächsten Zahlen stehen die Punkte mit ihren x- und y-Koordinaten. Auf der Übungsseite finden Sie eine Beispiel-Datei in diesem Format[4]. Um Ihr Programm mit dieser Datei zu testen, verwenden Sie die Eingabeumleitung:

```
./convex-hull < convex-hull-points.txt
```

(a) Schreiben Sie eine Klasse `Point`, die einen Punkt mit den Koordinaten x und y repräsentiert. Die Klasse soll zwei Konstruktoren haben, `Point()` und `Point(double x, double y)`, wobei die Koordinaten im ersten Fall mit 0 initialisiert werden sollen. Verwenden Sie hier eine *constructor initalizer list*.

(b) Schreiben Sie Operatoren, mit denen Sie `Point` direkt aus einem `std::istream` initialisieren und auf die Standardausgabe ausgeben können:

```
Point p;
std::cin >> p;
std::cout << p << std::endl;
```

(c) Schreiben Sie eine Funktion `std::vector<Point> read_problem()`, die ein Problem von der Standardeingabe einliest und eine Liste mit den eingelesenen Punkten zurückgibt.

(d) Berechnen Sie die konvexe Hülle der Punkte. Hierzu können Sie einen der Algorithmen auf der Wikipedia-Seite zur konvexen Hülle verwenden, am einfachsten den Graham-Scan[5]. Hierzu müssen Sie bestimmen, ob ein Punkt C links oder rechts der Geraden durch zwei andere Punkte A und B liegt, wozu Sie folgende Relation verwenden können:

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

(e) Schreiben Sie die Indizes der Punkte auf der konvexen Hülle auf die Standardausgabe im gleichen Format wie die Eingabe (in der ersten Zeile die Anzahl an Punkten, danach ein Index-Wert pro Zeile). Die Indizes müssen 0-basiert sein, so dass mit Ihnen die passenden Koordinaten in

[4] https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2017/convex-hull-points.txt
[5] https://de.wikipedia.org/wiki/Graham_Scan

dem anfangs eingelesenen `std::vector<Point>` nachgeschaut werden können. Die Punkte selbst müssen so sortiert sein, dass sie in aufsteigender Reihenfolge entweder im oder gegen den Uhrzeigersinn auf dem Rand der konvexen Hülle liegen.

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 [6]. Für dieses Skript müssen die Python-Pakete `numpy` und `matplotlib` installiert sein. Danach können Sie Ihr Programm `convex-hull` auf folgende Weise testen:

```
./convex-hull < eingabe.txt > ausgabe.txt
python plot-hull.py eingabe.txt ausgabe.txt
```

(Fortgeschritten)

[6] `https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2017/plot-hull.py`