Programmierkurs IPK SS23
Andreas Naumann
IWR, Universität Heidelberg

Übung 3
Abgabedatum —
ipk-exercises:2023-ss-1-ga451146

Übung 3

Aufgabe 1: Positive Potenzen von ganzen Zahlen

In dieser Aufgabe sollen Sie positive Potenzen $n \in \mathbb{N}_0$ von ganzen Zahlen $q \in \mathbb{Z}$ berechnen:

$$q^n = \prod_{i=1}^{n} q = \underbrace{q \cdot q \cdots q}_{n-\text{mal}}, \qquad q^0 = 1.$$

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

(a) Schreiben Sie eine Funktion `int iterative(int q, int n)`, die q^n mit Hilfe einer Schleife berechnet. Diese Funktion soll im namespace `power` liegen.

(b) Schreiben Sie eine Funktion `int recursive(int q, int n)`, die q^n berechnet, indem sie sich wiederholt selbst mit anderen Argumenten aufruft. Hier müssen Sie eine geeignete Abbruchbedingung finden, bei der die Funktion sich nicht mehr weiter selbst aufruft. Diese Funktion soll ebenfalls im namespace `power` liegen.

(c) Eine naive Implementierung obiger Funktionen muss $n-1$ Multiplikationen durchführen. Können Sie eine bessere Implementierung finden? Sie können die verbesserte Version entweder iterativ oder rekursiv implementieren, was immer Ihnen einfacher erscheint. Tipp: $q^{2n} = (q^n)^2$.

Hinweise:

- Ihr Hauptprogramm soll q und n von der Kommandozeile einlesen und das Ergebnis ausgeben.
- Achten Sie darauf, dass die Funktion `main(int argc, char** argv)` nicht innerhalb des namespace liegt, sonst funktioniert Ihr Programm nicht!

Aufgabe 2: Berechnung n-ter Wurzeln

In der vorherigen Aufgabe 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 (`double`), 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. Sollten Sie mit der letzten Aufgabe Schwierigkeiten gehabt haben dürfen sie auch `std::pow`[1] verwenden. Vergessen Sie nicht `#include <cmath>` mit einzufügen.

(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, `steps` 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.

Aufgabe 3: Fibonacci-Folge

Jedes Element der Fibonacci-Folge wird durch Addition der beiden vorherigen Folgen-Elemente gebildet, wobei die ersten beiden Elemente durch 0 und 1 gegeben sind:

$$\begin{aligned} f_1 &= 0, \\ f_2 &= 1, \\ f_n &= f_{n-2} + f_{n-1}. \end{aligned}$$

Damit ergibt sich als Beginn der Folge:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

(a) Implementieren Sie die Berechnung der Folgen-Elemente in einem C++-Programm.

i. Schreiben sie eine Funktion `int fibonacci(int number)`, welche f_N berechnet. Lesen Sie N von der Kommandozeile ein und geben sie f_N aus.

ii. Erweitern Sie Ihr Programm so, dass es nacheinander f_n für alle Werte von 0 bis N auf der Kommandozeile ausgibt.

iii. Probieren Sie Ihr Programm für verschiedene Werte von N aus. Was passiert, wenn Sie N groß werden lassen? Haben Sie eine Erklärung hierfür?

Hinweis: Sie können die Geschwindigkeit Ihres Programms verbessern, indem Sie beim Kompilieren die Option `-O3` angeben. Dadurch analysiert der Compiler Ihr Programm und erzeugt ein optimiertes Programm, das normalerweise deutlich schneller ist.

(b) Im folgenden geht es darum, das Programm aus dem ersten Teil zu verbessern.

Anmerkung: Es kann natürlich sein, dass Sie den vorherigen Aufgabenteil schon so implementiert haben, dass hier gar keine Probleme auftreten. In diesem Fall sind Sie bereits fertig.

i. Sorgen Sie dafür, dass Ihr Programm auch für grössere N korrekt funktioniert. Schauen Sie sich hierzu die Folien zu Variablentypen an.

ii. Die Laufzeit des Programms steigt für ungefähr $N = 40$ sehr stark an. Woran liegt das? Schreiben Sie eine alternative Version des Programms, die dieses Problem umgeht und die Folgenglieder bis mindestens $N = 90$ ohne nennenswerte Verzögerung ausgeben kann.

[1] https://en.cppreference.com/w/cpp/numeric/math/pow