Programmierkurs (Wintersemester 2017) **Aufgabenblatt 2**

Dr. Ole Klein, Dr. Steffen Müthing Abgabe 13. November 2017

IWR, Universität Heidelberg

Allgemeine Hinweise:

- Beginnend mit diesem Übungsblatt werden wir Ihre Mitarbeit mit einem *Votiersystem* überprüfen. Dies funktioniert folgendermassen:
  - Freitags wird ein neues Übungsblatt ausgegeben.
  - In den Übungen der darauf folgenden Woche können Sie am Übungsblatt arbeiten und sich von den Übungsleiter und Tutoren dabei helfen lassen.
  - In den Übungen der zweiten Woche (wenn Sie schon am nächsten Übungsblatt arbeiten), lassen wir einen Zettel herumgehen, bei dem Sie jede Aufgabe ankreuzen, die Sie in der vorherigen Übung und zu Hause bearbeitet haben.
  - Im weiteren Verlauf der Übung kommen die Übungsleiter zu einigen von Ihnen und lassen sich von Ihnen eine der angekreuzten Übungen zeigen (nach Wahl des Übungsleiters).
  - Sollte der Übungsleiter den Eindruck bekommen, dass Sie die Lösung nur kopiert und sich nicht wirklich selbst damit beschäftigt haben, werden Ihnen alle Kreuze für den Tag aberkannt.
  - Für die Klausurzulassung benötigen Sie 50% angekreuzte Aufgaben und müssen mindestens eine Aufgabe vorgestellt haben.
- Alle Aufgaben werden von jetzt an mit **Grundlagen** oder **Fortgeschritten** markiert.
  - Falls Sie in eine der normalen Übungsgruppen eingetragen sind, brauchen Sie für die Zulassung nur 50% der Grundlagen-Aufgaben. Sie dürfen natürlich gerne trotzdem Punkte mit Fortgeschrittenen-Aufgaben sammeln.
  - Fall Sie in einer der Helfer-Gruppen eingetragen sind, brauchen Sie 50% der Fortgeschrittenen-Aufgaben und können auch nur diese ankreuzen.
- Um eine Aufgabe ankreuzen zu können, müssen Sie **keine** perfekte Lösung haben! Das Ziel des Kurses ist, Ihnen Programmierpraxis zu geben — es ist völlig in Ordnung, wenn Sie nach der Hälfte der Aufgabe nicht weiter wußten und uns erklären können, wie Sie zu diesem Punkt gekommen sind.
- Aufgabe 4 ist dafür gedacht, die C++-Kenntnisse der Studierenden in den Helfergruppen einzuordnen. Wenn Sie es bis (c) schaffen, sind Sie als Helfer richtig!

Übung 1 *Die Collatz-Vermutung*

Schreiben Sie eine Funktion `void collatz(int zahl)`, die folgendes tut:

- Gib `zahl` auf dem Bildschirm aus.
- Falls `zahl` gerade ist, teile die Zahl durch 2.
- Andernfalls multipliziere die Zahl mit 3 und addiere 1.
- Wiederhole diese Schritte, bis einer der folgenden Zahlenwerte erreicht wird:
  $1, 0, -1, -5$ oder -17.

Hinweis:
Um herauszufinden, ob eine Zahl x gerade ist, testen Sie, ob $(x \mod 2) = 0$. Hierfür gibt es in C++ den Operator `x % y`, der den Rest der Ganzzahl-Division von x durch y berechnet:

```
int x = 23 / 5; // 4 (runden nach −∞)
int y = 23 % 5; // 3 (vorzeichenbehafteter Divisionsrest)
```

Rufen Sie die obige Funktion aus der `main`-Funktion auf, wobei Sie den ersten Wert für `zahl` von der Tastatur einlesen. Auf diese Weise können Sie die entstehenden Zahlenfolgen für verschiedene Startwerte untersuchen. Warum kann der Wert 0 nur auf eine Weise erreicht werden? Und was haben alle Zahlen gemeinsam, die zum Wert 1 führen? Informieren Sie sich unter `https://en.wikipedia.org/wiki/Collatz_conjecture` über die mathematische Vermutung hinter diesen Zahlenfolgen.

(Grundlagen)

Übung 2 *Primzahlen*

Eine natürliche Zahl $p \neq 1$ heißt *prim*, wenn Sie nur durch sich selbst und 1 teilbar ist.

(a) Schreiben Sie eine Funktion `bool isPrime(int number)`, die für eine gegebene Zahl testet, ob diese eine Primzahl ist. In diesem Fall soll sie `true` zurück geben, ansonsten `false`. Beachten Sie dabei insbesondere, dass nicht-positive Zahlen nach Definition nie prim sind.

(b) Schreiben Sie außerdem eine Funktion `void printPrimes(int upto)`, die alle Primzahlen bis einschließlich `upto` auf dem Bildschirm ausgibt. Falls `upto` zu klein ist, um überhaupt eine Zahl auszugeben, soll stattdessen eine entsprechende Fehlermeldung erscheinen. Greifen Sie hierbei auf die von Ihnen implementierte Funktion `isPrime()` von oben zurück.

Testen Sie die beiden Funktionen in einem Programm, das für verschiedene Werte von `upto`, davon mindestens einer negativ, einer gleich Null, und einer größer als 50, die Funktion `printPrimes` aufruft.

(Grundlagen)

Übung 3 *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 pow_iterative(int q, int n)`, die q^n mit Hilfe einer Schleife berechnet.

(b) Schreiben Sie eine Funktion `int pow_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.

(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: $q^4 = (q^2)^2$.

Übung 4 *Zahlen einlesen*

Anstatt sich auf die eingebauten Funktionen von C++ zu verlassen, schreiben Sie im folgenden eine Reihe von Funktionen, die einen String (z.B. `"1346"`) in eine Zahl umwandeln sowie ein Hauptprogramm, dass diese Funktionen testet.

Für den Anfang können Sie davon ausgehen, dass Sie nur gültige Zeichenfolgen als Eingabe bekommen.

(a) Schreiben Sie eine Funktion `int parse_int(std::string number)`, die den String `number` in eine Zahl umwandelt und diese zurückgibt. Sie können davon ausgehen, dass die Eingabe kein Vorzeichen enthält.

Hinweise:

- Ein `std::string`[1] enthält eine Folge von Zeichen. Ein einzelnes Zeichen können Sie in einer Variablen vom Typ `char` speichern. Das folgende Beispiel zeigt Ihnen, wie Sie an die Information in `std::string` herankommen:

```
1 std::string s = "47218";    // Zuweisung
2 std::getline(std::cin,s); // Zeile einlesen und in s speichern
3 int size = s.size();        // Gibt die Länge des Strings zurück
4 char c = s[0];              // erstes Zeichen (0-basierter Index)
5 char d = s[size-1];         // letztes Zeichen
```

- Eine Variable vom Typ `char` enthält eine Zahl, die das zugehörige Zeichen repräsentiert. Hierfür gibt es verschiedene Standards, für unsere Zwecke reicht ASCII[2]. Dabei ist z.B. der Wert des Zeichens `'3'` nicht 3, sondern 51. Sie müssen den Wert des Zeichens `'0'` (= 48) abziehen, um zum korrekten Ziffernwert zu kommen:

```
1 char three = '3';
2 std::cout << three << std::endl; // Ausgabe: 51
3 three = three - '0';             // alternativ: three - 48
4 std::cout << three << std::endl; // Ausgabe: 3
```

(b) Erweitern Sie Ihre Funktion so, dass sie ein optionales `'+'` oder `'-'` am Anfang des Strings korrekt einliest und auf die Zahl anwendet.

(c) Erweitern Sie Ihre Funktion so, dass sie eventuelle Leerzeichen am Anfang des Strings ignoriert und die Zahl so lange weiter einliest, bis ein anderes Zeichen als eine Ziffer kommt. Beispiel: Der String `"  -7628text"` soll in die Zahl -7628 umgewandelt werden.

(d) Erweitern Sie Ihre Funktion so, dass sie im Fehlerfall (also, wenn der String z.B. mit einem Buchstaben anfängt) eine Exception vom Typ `std::invalid_argument`[3] wirft.

(e) Erweitern Sie Ihre Funktion so, dass sie statt einem `int` ein `std::pair<int,int>` zurückgibt, wobei der erste Eintrag die eingelesene Zahl sein soll und der zweite der Index des ersten Zeichens, das nicht mehr eingelesen wurde.

Bonus-Votierpunkt: Schreiben Sie eine Funktion `parse_double(std::string number)`, die alle oben aufgeführte Funktionalität enthält und folgende Strings als Fließkommazahl einlesen kann: `"173"`, `"17.3"`, `"17."`, `".342"`, `"-0.342e4"`, `"2.3813E-71"`.

[1] `http://en.cppreference.com/w/cpp/string/basic_string`
[2] American Standard Code for Information Interchange, `https://en.wikipedia.org/wiki/ASCII`
[3] `http://en.cppreference.com/w/cpp/error/invalid_argument`