Technische Universität München
Institut für Informatik
Dr. rer. nat. Tobias Weinzierl
Kaveh Rahnema

März 2013

Kompaktkurs Einführung in die Programmierung

Übungsblatt 4: Schleifen

Lernziele

- Schleifen und Abbruchbedingungen.
- Rechnen mit Gleitkommazahlen.
- Rundungsfehler und resultierende Probleme.

1) `for` und `while` (P)

Ändern Sie Ihr Programm von Blatt 2, Aufgabe 1 (Fakultät berechnen, Liste der vollkommen Zahlen) so, dass alle darin vorkommenden `for`-Schleifen durch `while`-Schleifen ersetzt werden. Entfernen Sie anschließend überflüssige `{}` von allen Blöcken, die nur eine Anweisung enthalten.

2) Fakultätsberechnung bis zum Äußersten (P)

Schreiben Sie ein Programm, das für $i = 1, 2, \ldots$ die Fakultät $i!$ berechnet und ausdruckt – solange, bis $(i + 1)!$ nicht mehr als `int`-Zahl darstellbar ist. Verwenden Sie dazu die Konstante `INT_MAX` für die größte `int`-Zahl, sie ist verfügbar mittels
`#include <climits>` (C++-Version) bzw.
`#include <limits.h>` (C-Version; auf der WWW-Seite finden Sie ein Beispielprogramm zur Inspiration).

Nun ist zu gegebenem i und $i!$ die Bedingung $(i+1)! \leq$`INT_MAX` zu formulieren – und zwar, ohne dass dabei als Zwischenergebnis Zahlen größer als `INT_MAX` auftreten können, insbesondere darf $(i + 1)!$ für den Vergleich noch nicht berechnet werden! Natürlich soll Ihr Programm auch unabhängig vom tatsächlichen Wert für `INT_MAX` funktionieren.

3) Schleifen und Gleitpunktzahlen (P)

Auf der WWW-Seite zur Vorlesung finden Sie das Programm `fp_table.cpp` welches eine Tabelle bestehend aus 2 Spalten druckt. In der ersten Spalte soll $x \in [0.0, 10.0]$ in Zehntelschritten ausgegeben werden, in der zweiten Spalte der korrespondierende Wert x^2. Es entsteht somit eine kleine Wertetabelle.

```
0       0
0.1     0.01
0.2     0.04
...     ...
...     ...
...     ...
9.8     96.04
9.9     98.01
10      100
```

i) Compilieren Sie das Programm und starten Sie es. Was passiert?
Hinweis: Die Tastenkombination <Strg> + C wird Ihnen nach dem Start des Programms behilflich sein!

ii) Formen Sie die Abbruchbedingung der `for`-Schleife so um, dass Sie den Differenzbetrag zwischen x und der oberen Grenze gegen eine geringe Konstante (z.B. `1e-3`) testen lassen. Den Absolutbetrag liefert Ihnen die Funktion `fabs()`. Kommentieren Sie die entsprechende Stelle aus und ersetzen Sie sie durch Ihre neue Kontrollstruktur. Das Problem aus Teilaufgabe a) sollte damit behoben werden können. Was ist bei diesem Vorgehen problematisch?

iii) Wie könnte das Problem der Ausgangsschleife (relativ einfach) noch behoben werden?

iv) Formulieren Sie die im Programm enthaltene `for`-Schleife so um, dass im Schleifenkopf nur mit ganzzahligen Variablen (`int`) gerechnet wird. Dabei soll die Ausgabe aus Teilaufgabe b) bzw. c) erhalten bleiben.

4) Rundungsfehler (P)

Schreiben Sie ein Programm `wurzel.cpp` welches zu fest vorgegebenem `n` die Zahl $x_{1,2} = (\sqrt{2})^n$ berechnet. Dabei soll zunächst x_1 über reine Multiplikationen von $\sqrt{2}$ in einer Schleife berechnet werden. Anschließend soll x_2 über halb so viele Multiplikationen von 2 in einer Schleife berechnet werden. Beachten Sie, dass für ungerades n eine zusätzliche Wurzelmultiplikation für x_2 notwendig wird. Somit ergibt sich

$$x_2 = \begin{cases} 2^{n/2}, & \text{wenn } n \text{ gerade,} \\ \sqrt{2} \cdot 2^{\lfloor n/2 \rfloor}, & \text{wenn } n \text{ ungerade.} \end{cases}$$

i) Berechnen Sie zunächst nur x_1. Codieren Sie dazu zunächst ihr eigenes n (z.B. $n = 120$). Zur Aufmultiplizierung von $\sqrt{2}$ in einer `for`-Schleife benutzen Sie bitte die Funktion `sqrt()` aus `math.h`. Machen Sie sich mit der formatierten Ausgabe von `std::cout` vertraut.
Über `std::setprecision(20)` können Sie anschließend x_1 (wie auch alle anderen Ergebnisse) auf 20 Stellen ausgeben lassen:
`std::cout << std::setprecision(20) << "x_1 = " << x1 << std::endl;`

ii) Berechnen Sie nun x_2 über $\frac{n}{2}$ Multiplikationen von 2. Achten Sie dabei auf ungerade n. Lassen Sie nun auch x_2 sowie die Differenz $x_1 - x_2$ drucken. Was können Sie, deutlich für größere n (z.B. $n = 120$), beobachten und warum?

5) Anzahl Primzahlen (★)

Basis dieser Aufgabe ist es, für eine Zahl n die Anzahl an Primzahlen zu berechnen. Die Berechnung soll mittels OpenMP parallelisiert werden.

i) Implementieren Sie zunächst sequentiell eine Funktion `int prime_number ( int n )` die Ihnen für gegebenes n die Anzahl an Primzahlen (bis zu n) zurückgibt. Zum Beispiel für $n = 1024$ sind das 172.

ii) Implementieren Sie nun eine Funktion

`void prime_number_sweep ( int n_lo, int n_hi, int n_factor )`

die im Intervall $[n_{lo}, n_{hi}]$ in Potenzschritten n_{factor} die Anzahl an Primzahlen ausgibt. Zum Beispiel soll für $n_{lo} = 1$, $n_{hi} = 131072$ und $n_{factor} = 2$ folgende Ausgabe erscheinen:

```
N         Primes
1         0
2         1
4         2
8         4
16        6
32        11
64        18
128       31
256       54
512       97
1024      172
2048      309
4096      564
8192      1028
16384     1900
32768     3512
65536     6542
131072    12251
```

iii) Modifizieren Sie nun Ihre Funktion `int prime_number ( int n )` so, dass die Berechnung der Primzahlen parallel mit Hilfe OpenMP abläuft.

iv) Erweitern Sie Ihre Ausgabe aus Teilaufgabe ii) so, dass eine dritte Spalte mit der benötigten Zeit angezeigt wird. Überprüfen Sie, ob Sie mit Hilfe mehrerer Threads eine zeitliche Beschleunigung der Berechnung erhalten. Hinweis: `omp_get_wtime ( )` wird Ihnen dazu hilfreich sein.