Übungen zu Computergrundlagen WS 2016/2017

Übungsblatt 12: Programmieren in C

27. Januar 2017

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist **Freitag, 03.02.2017, 11:00 Uhr**.
- Schickt die Lösungen bitte per Email an Euren Tutor:
  - Montag 11:30 – 13:00 Uhr: Julian Michalowsky (`jmichalowsky@icp.uni-stuttgart.de`)
  - Montag 14:00 – 15:30 Uhr: Frank Uhlig (`fuhlig@icp.uni-stuttgart.de`)
  - Dienstag 14:00 – 15:30 Uhr: Patrick Kreissl (`pkreissl@icp.uni-stuttgart.de`)
  - Dienstag 15:45 – 17:15 Uhr: Kai Szuttor (`kai@icp.uni-stuttgart.de`)
  - Donnerstag 09:45 – 11:15 Uhr: Frank Maier (`fmaier@icp.uni-stuttgart.de`)
  - Donnerstag 15:45 – 17:15 Uhr: Evangelos Tzaras (`etzaras@icp.uni-stuttgart.de`)

Aufgabe 12.1: Berechnung von π mit C (7 Punkte)

Ziel dieses Aufgabenblattes ist es, π numerisch mit C-Programmen zu approximieren. Dabei sollen dieselben Algorithmen wie auf Blatt 9 verwendet werden und dieselbe Anzahl an Berechnungen durchgeführt werden (1.000.000 Monte-Carlo Versuche/Stützstellen).

- **12.1.1** Schreibt ein C-Programm, das π mit der Monte-Carlo Methode aus Blatt 9 abschätzt. Dazu sollen gleichverteilte Zufallszahlen aus den Einheitsquadrat gezogen werden. Das Verhältnis von Punkten, die innerhalb des Einheitskreises liegen, zu der Gesamtzahl an gezogenen Punkten geht für hinreichend viele Versuche gegen $\pi/4$. (3 Punkte)

  Hinweise:

  - Bedingte Ausführung in C ermöglicht ähnlich wie in Python der `if (...) {...}`-Befehl. Dabei wird der Code in den geschweiften Klammern nur ausgeführt, wenn die Bedingung in der runden Klammer erfüllt ist.
  - Benutzt Fließkommazahlen vom Typ `double`.
  - Wie schon in Python gilt auch in C: Wenn Eure Näherung für π Null ist, dann habt Ihr wahrscheinlich nicht darauf geachtet, das Verhältnis der Punktanzahlen in Fließkomma zu berechnen.
  - Fließkomma-Zufallszahlen zwischen 0 und 1 erzeugt in C die Funktion `drand48()`. Um diese benutzen zu können, müsst Ihr mit Hilfe des Befehls

    ```
    #include<stdlib.h>
    ```

    die Header der Standard-Bibliothek am Anfang des Programms einbinden. Hilfe zu dieser Funktion findet Ihr auf ihrer `man`-Seite.
  - Zum Kompilieren Eures C-Programms verwendet am besten den Befehl

    ```
    gcc -std=gnu99 -O3 -Wall -lm -o compute_pi_1 compute_pi_1.c
    ```

Beachtet, dass anstelle von "-std=c99" hier "-std=gnu99" gewählt ist. Das ist nötig, damit gcc auch Funktionen des POSIX-Standards wie `drand48()` in der Standardbibliothek freischaltet.

- **12.1.2** Schreibt nun ein C-Programm, dass π mit Hilfe der Monte-Carlo Integration des Einheitskreises approximiert, also die Fläche unter der Funktion $f(x) = \sqrt{1 - x^2}$ nähert. (2 Punkte) **Hinweis:** Die Wurzel berechnet die Funktion `sqrt(x)`. Um diese benutzen zu können, müsst Ihr zusätzlich zur `stdlib.h` auch den Header `math.h` einbinden.
- **12.1.3** Schreibt schließlich ein C-Programm, dass wie auf Blatt 9 die Integration des Einheitskreises mit gleichmäßig verteilten Stützstellen durchführt. (2 Punkte) **Hinweis:** Auch hier müsst Ihr darauf achten, rechtzeitig auf Fließkommazahlen zu wechseln. Eine einfache Möglichkeit, um eine Ganzzahl N in eine Fließkommazahl zu wandeln, ist etwa `1.0*N`.

Aufgabe 12.2: Laufzeitvergleich von Python und C (3 Punkte)

Da C-Programme verhältnismäßig kompliziert und länger sind als vergleichbare Programme in Python, muss es etwas geben, dass diesen Mehraufwand rechtfertigt. Der große Vorteil von C ist die Geschwindigkeit, die die kompilierten Programme erreichen. Dies wollen wir nun durch einen Vergleich der Laufzeiten Eurer C-Implementationen mit entsprechenden Pythonskripten zeigen.

- **12.2.1** Passt Eure Programme so an, dass die Berechnungen für π jeweils 100 mal durchgeführt werden und miss die Laufzeit für alle 3 Methoden, um π zu berechnen. (1 Punkt)

  Hinweis: Um die Laufzeit eines Programms zu bestimmen, könnt Ihr den Shell-Befehl `time` benutzen.
- **12.2.2** Messt nun ebenfalls die Laufzeit der 4 Pythonskripte in `/group/cgl/2015/12` mit derselben Anzahl an Stützstellen und Wiederholungen. (1 Punkt)
- Vergleicht die Laufzeiten der verschiedenen Implementierungen. Warum verhält sich `compute_pi_4.py` etwas anders als die anderen Pythonversionen? (1 Punkt)