EINFÜHRUNG IN DIE PROGRAMMIERUNG FÜR STUDIERENDE DER NATURWISSENSCHAFTEN (SOSE 2020)

–

INFORMATIONEN UND THEMENVORSCHLÄGE ZU DEN PROJEKTEN

1. Infos

Die Projekte sind wie die Übungsaufgaben in **Zweiergruppen** zu bearbeiten und im Rahmen eines **kurzen Vortrags (15–20 Minuten)** vorzustellen. Die Bearbeitung muss bis zum Vortragstag abgeschlossen sein. Alle Programme sind selbstverständlich selbst zu schreiben und gut zu dokumentieren. **Die Programme müssen zusammen mit einer kurzen schriftlichen Ausarbeitung des Vortrags (ca. 2 Seiten) abgegeben werden.**
Unten finden sich einige Themenvorschläge. Wählen Sie ein Thema aus oder lassen Sie sich von den Vorschlägen inspirieren: Um persönliche Interessen zu berücksichtigen sind **eigene Vorschläge** nicht nur erlaubt, sondern **explizit erwünscht**. Falls zu viele Gruppen das gleiche Thema bearbeiten wollen, behalten wir uns natürlich vor, auch Wünsche abzulehnen. Hier gilt das Windhund-Prinzip: Wer den Wunsch zuerst geäußert hat, wird auch zuerst berücksichtigt.
Sollten sich bei der Bearbeitung Fragen oder Probleme ergeben, bietet sich wie sonst auch das Forum an.
Schreiben Sie zur **Festlegung des Themas** bitte rechtzeitig vor dem 28.06. eine E-Mail mit ihrem Wunschthema an Ihren Tutor.

2. Vortragstermine

Für jede Übungsgruppe gibt es einen Präsentationstermin in der ersten Woche nach Ende der Vorlesungszeit. Die **Teilnahme an dem jeweiligen Termin ist verpflichtend** für alle Teilnehmer der entsprechenden Übungsgruppe. Die Vorträge werden voraussichtlich als Videokonferenzen in BigBlueButton stattfinden und sollten dementsprechend vorbereitet werden. Die genauen Termine können Sie der folgenden Tabelle entnehmen:

	3.8.	4.8.	5.8.
9–13 Uhr (s. t.)	Gruppe 5	Gruppe 3	Gruppe 1
14–18 Uhr (s. t.)	Gruppe 6	Gruppe 4	Gruppe 2

3. Themenvorschläge

Die Markierungen sind wie folgt zu verstehen: (A) = Anfängerthema, (F) = Fortgeschrittenes Thema, Keine Kennzeichung = mittlere Schwierigkeit, (M) = Mathematisches Thema. Die Anfängerthemen sind nicht weniger aufwendig zu bearbeiten, sie erfordern lediglich weniger Vorwissen.

- *Einfache Spiele*, textbasiert in der Konsole, Funktionen zum Speichern/Laden des Spielstands in/aus einer Datei, ggf. mit einfacher KI, z. B.:
  (1) (A) Kartenspiele (BlackJack, Mau-Mau, ...)
  (2) (A) Vier gewinnt

Date: June 10, 2020.

(3) (A) Mäxchen
(4) (A) Kniffel

- Spiele mit grafischer Ausgabe in der Konsole unter Verwendung der Bibliothek *ncurses* (Einführung: `https://de.wikibooks.org/wiki/Ncurses`), z. B.:
  (5) (F) Snake
  (6) (F) TicTacToe
  (7) (F) Schiffe versenken
  (8) (F) Minesweeper
  (9) (F) Tetris
  (10) (F) Sokoban
  (11) (F) Conway's game of life
- *Numerische Simulationen* (jeweils geeignete Plots der Simulationen erstellen, etwa durch einlesen der gespeicherten Daten in MATLAB/GnuPlot), z. B.:
  (12) (A,M) SIR-Modell implementieren, evtl. Parameter-Matching auf Corona-Epidemie
  (13) (A,M) Räuber-Beute-Modell (mit/ohne Erweiterungen) implementieren
  (14) (M) Doppelpendel simulieren
  (15) (A,M) Zwei-Körper-Problem (Planetenbahnen) simulieren
  (16) (A,M) LGS-Löser (exaktes Gaussverfahren vs. Jacobi/Gauss-Seidel/(SOR))
  Die relevanten Gleichungen/Verfahren finden sich z.B. auf Wikipedia.
- *Rekursives Backtracking* zur Lösung eines Problems in C++ implementieren (siehe `https://de.wikipedia.org/wiki/Backtracking`), z. B.:
  (17) Sudokulöser
  (18) Damenproblem
  (19) Springertour (Knight's Tour)
  (20) (F) rekursiver Spiele-Algorithmus (vgl. v. Rimscha - *Algorithmen kompakt und verständlich*)
- *Bildverarbeitung* (Skript dazu auf Anfrage):
  (21) ppm-Format: Datei einlesen, Umwandlung Farbe $\leftrightarrow$ S/W, Weichzeichnen, Datei speichern
- *Kryptografie/Verschlüsselung*:
  (22) Codebreaker für Caesar-Verschlüsselung mittels Häufigkeitsanalyse
  (23) (M) RSA-Verfahren implementieren (s. Haftendorn - *Mathematik sehen und verstehen*)
  (24) (M) Primzahltests: Fermat-, Solovay-Strassen-, Miller-Rabin-Test (Wikipedia)
- *Sortierverfahren*:
  (25) Vergleich Quicksort mit Mergesort/Bubblesort (Theorie + Implementierung)
- *Andere Programmiersprachen*:
  (26) (F) Vorstellung einer anderen Programmiersprache anhand von Beispielen, Vergleich mit C++ (Benchmarks,...)
  (27) Einfache eigene Programmiersprache entwerfen einen Interpreter dafür in C++ schreiben
  (28) (F) Brainfuck-Interpreter schreiben
  (29) (F) Parallelisierter Code mit MPI (ggf. Vergleich mit OpenMP)
- Speziell für *Lehramts-Studierende*:
  (30) (A) fischertechnik-Roboter aufbauen und programmieren: Ist vielleicht interessant für den Einsatz im Schulunterricht (insbes. NwT). Es gibt 6 Roboterkästen in der Abteilung für Didaktik der Mathematik, die ausgeliehen werden können. *Hinweis:* Da der dortige PC-Pool jedoch in diesem Semester nicht an einem gemeinsamen Termin genutzt werden kann, müsstet ihr euch selbst um die Installation der nötigen

Software auf eurem Rechner kümmern (kann im Internet heruntergeladen werden). Dementsprechend können wir bei technischen Problemen auch nur eingeschränkt helfen, es gibt jedoch Anleitungen im Internet.

(31) (A,M) Numerische Verfahren die auch für den Schulunterricht geeignet sind programmieren und miteinander vergleichen, z. B. Bisektion vs. Sekantenverfahren vs. Regula falsi zum Auffinden von Nullstellen ...

(32) (A,M) ... oder auch Monte-Carlo-Integration vs. Trapezregel

(33) (A,M) Stochastische Simulationen (z. B. alte Abituraufgaben simulieren oder Probleme betrachten, die mit Schulmathematik nicht lösbar sind, jedoch einfach simuliert werden können)

- *Sonstige Ideen*:

(34) (F) BigInteger-Klasse implementieren (um mit Zahlen zu rechnen, die für `long` zu groß sind), z. B. mit `std::vector` für die einzelnen Ziffern, Funktionen/Methoden bspw. für Summe, Differenz, Produkt, usw. implementieren oder die entsprechenden Operatoren überladen.

(35) Maze-Running-Algorithmus vorstellen und implementieren (v. Rimscha - *Algorithmen kompakt und verständlich*)

(36) (A) Code-Sonne: Umwandlung von DNA-Sequenzen in Aminosäuresequenzen (mit Datei-Ein-/Ausgabe)