Hinweise zur Bearbeitung der C++-Projekte

- Abgabe als einzelne zip/tgz-Datei (die Ihren Namen trägt) per Email an mich bis zum **15.10.2017, 23:59**.
- Bitte sehen Sie ab von Benutzung von Clouds zur Übermittlung des Projekts. Wenn Ihr Projekt wirklich mehrere Dutzend Megabyte groß wird, benutzen Sie BoxUP, nicht irgendwelche kommerziellen Cloud-Anbieter.
- Jede Datei muss einen Vorspann mit Autor, Matrikelnummer, Compiler, Compilerversion, verwendetes Betriebssystem und Zweck der Datei enthalten
- Jedes **private und öffentliche** Klassenelement (auch Datentypen und Instanzvariablen) muss mit doxygen-Kommentaren kommentiert sein (vgl. www.doxygen.org).
- Kommentieren Sie auch den Quelltext Ihrer Funktionen ausführlich! Daumenregel: jede dritte Code-Zeile wird kommentiert.
- Ihrer Lösung muss eine doxygen-Datei zur Erzeugung der html-Dokumentation beiliegen (aber nicht das erzeugte HTML!) [seit einiger Zeit gibt es eine Alternative zu doxygen: doxypress. Wenn Sie Lust haben, probieren Sie dies aus].
- Strukturieren Sie Ihr Projekt in Unterverzeichnisse. Bewährt hat sich eine Struktur mit den Verzeichnissen include (.h./hpp-Dateien), src (.cpp-Dateien), doc (.pdf, .doxygen), data (Eingabe/Testdateien) und bin (ausführbares Programm).
- Ebenfalls beiliegen muss ein Testprogramm, welches Ihre Klasse unter Ausgabe von Kommentaren beispielhaft demonstriert. **Legen Sie auf jeden Fall Testdaten verschiedener Größen bei.**
- Geben Sie nur Quelldateien und Testdaten ab (neben Makefile, Doxygen-Config usw.). Keine html-Dokumention (diese wird durch die Make-Datei mittels doxygen erzeugt), keine ausführbaren Dateien, keine Objekt-Dateien, keine Backup-Dateien, keine MacOS Ordner usw).
- Legen Sie ein Makefile bei, das Ihr Testprogramm und die Dokumentation beim Aufruf von make erzeugt. Verwenden Sie make-Variablen, um mit zwei verschiedenen Compilern zu übersetzen (z.B. g++ und clang). Achten Sie darauf, dass alles glatt durchläuft. Also bitte keine Dateien mit Syntaxfehlern, solche Projekte berücksichtige ich nicht weiter.
- Ihr Makefile muss auch ein test-Target enthalten, wo mit `make test` einige Tests mit Ihrem Demo-Programm durchgeführt werden.
- Beschreiben Sie in einer zweiseitigen Dokumentation kurz die Verwendungsweise Ihres Programms. Orientieren Sie sich dabei an den Linux **man**-Seiten (z.B. http://www.linuxmanpages.com/man1/ls.1.php). Dazu gehört auch die Angabe von Aufrufbeispielen. Dokumentieren Sie auch die Formate von Eingabe- und Ausgabedateien.
- Dazu gehört auch ein Verzeichnis der von Ihnen verwendeten Literatur.

Weitere Hinweise:

- Denken Sie objektorientiert! Lange bevor Sie die erste Zeile schreiben, überlegen Sie, wie Sie die Aufgabe in Klassen strukturieren können. Ein Chart-Parser wird beispielsweise Klassen für die Grammatik, die Regeln und den Parser selbst beinhalten. Definieren Sie dann die öffentliche Schnittstelle dieser Klassen und entwerfen Sie dann erst die Datenstrukturen dafür.
- Schreiben Sie pro C++-Klasse eine .hpp-Datei.
- Halten Sie die internen Datenstrukturen flexibel, lassen Sie sie ggf. mitwachsen (Verwendung von STL-Klassen).
- Berücksichtigen Sie die ***goldenen Regeln*** der C++-Programmierung!
- Schreiben Sie keine Monsterfunktionen! Teilen Sie das Problem in sinnvolle Subfunktionen auf. Faustregel: max. eine Bildschirmseite pro Funktion.
- Machen Sie keine unnötigen Ausgaben! Folgen Sie dem Unix-Konzept: Alles so geräuschlos wie möglich. Wenn sich jedoch Fehler ergeben, dann geben Sie detaillierte Meldungen auf `std::cerr` aus.
- Programmieren Sie defensiv, d.h. antizipieren Sie Fehlerquellen so weit wie möglich und behandeln Sie diese!
- Testen Sie Ihre Klasse(n) ausführlich, auch mit unsinnigen/zufälligen Eingabedaten und Grenzfällen. **Ihr Programm darf auf keinen Fall abstürzen.** Widmen Sie dem Programmtest mind. 30% der Gesamtzeit.
- Führen Sie auch Stresstests mit großen Datenmengen aus. Achten Sie dabei auf Zeit- und Speicherplatzverbrauch (für solche Tests sollten Sie Ihr Programm immer mit voller Optimierung kompilieren (je nach Compiler -O2 /Ox))
- Kodieren Sie keine Daten (z.B. Dateinamen) ins Programm. Machen Sie Gebrauch von argc/argv.
- Prüfen Sie argc/argv-Argumente auf Korrektheit. Geben Sie im Falle von inkorrekter Bedienung eine kurze Programmsynopsis aus, wo sie jeden Programmparameter und ggf. jede Option noch kurz erläutern. Verwenden Sie bei komplizierteren Parameterstrukturen einen Befehlszeilenparser (z.B. http://tclap.sourceforge.net/)
- Verwenden Sie asserts so oft wie möglich, um während der Entwicklung Logikfehler zu entdecken. Deaktivieren Sie asserts im abgegebenen Code mit `#define NDEBUG` (vor `#include <cassert>`).