4. Übung des Programmierpraktikums

Abgabetermin: 15. Mai 2023, 23:59 Uhr

Die Übungen sind grundsätzlich selbst zu lösen.
Zweiergruppen sind erlaubt, aber nur unter Angabe des Partners im gut sichtbaren Kommentar.

Abzugeben sind jeweils die sinnvoll dokumentierten Programmfiles (`*.cpp`, `*.h`) in einem separaten Ordner dessen Name `bsp_`*`nummer`* zu sein hat, d.h., der Ordner `bsp_2` gehört zu Aufgabe 2.
Speichern Sie die Funktionsdeklarationen jeweils in einem separaten Headerfile und die Funktionsdefinitionen in einem separaten Sourcefile, d.h., jedes Projekt enthält mindestens zwei **.cpp* und ein **.h* File(s). Alle eigenen Funktionen und Klassen sind so im Headerfile zu dokumentierten, daß die Funktionalitäten für Dritte ausreichend beschrieben werden und daß `doxygen` damit arbeiten kann.

---

18. Implementieren Sie das Ratespiel Mastermind[1], auch als *Bulls and Cows* im englischen bekannt [Str10, p.193], als Funktion. Die Funktion erhält als Inputparameter einen geheimen Vektor mit 4 ganzen Zahlen aus dem Intervall $[1, p]$ (p wird ebenfalls als Inputparameter übergeben). (4 Pkt.)
    Aufgabe des Spielers ist es, diese Zahlen durch wiederholtes Raten herauszufinden. Angenommen, die zu ratende Zahl ist 1234 und der Spieler rät 1359 (diese Zahl ist einzugeben), dann soll die Antwort lauten "1 Bulle und 1 Kuh", weil der Spieler eine Zahl (die 1) nicht nur richtig geraten, sondern auch an die richtige Position gesetzt hat (ein Bulle). Eine zusätzliche Zahl ist richtig getippt (die 3), steht aber an der falschen Position (eine Kuh). Das Ratespiel wird solange fortgesetzt, bis der Benutzer 4 Bullen bekommt, d.h., alle Zahlen in der richtigen Reihenfolge erraten hat.
    Entwerfen Sie **zuerst** ein Struktogramm **bevor** Sie zu programmieren beginnnen.

    Hinweise:

    - Die Summe von *bull* und *cow* darf nicht größer als 4 sein.
    - Beachten Sie, daß sowohl im Geheimvektor als auch im eingegebenen Vektor Zahlen mehrfach vorkommen können, oder daß ein *bull* nicht nochmals als *cow* gezählt werden darf.
    - Eine Funktion `countBullCow` wäre hilfreich, da sich diese separat testen läßt. Überlegen Sie sich Testdaten welche die möglichen Fälle (am besten alle) abdecken, z.B.

Geheimzahl	Spieler	*bull*	*cow*
1234	1315	1	1
1234	1335	2	0
1234	1331	2	0
1134	1331	2	1
1131	1331	3	0
1234	4433	1	1
4433	1234	1	1

[1] `http://de.wikipedia.org/wiki/Mastermind_(Spiel)`

19. Das Ziel dieser kleinen Projektarbeit besteht darin, daß Sie ein Projekt selbständig entwerfen und programmieren. Hierbei sollen Sie:

- die Source- und Headerfiles einer gegebenen Bibliothek in Ihr Projekt einfügen und die darin enthaltenen **Funktionen** auch **nutzen**,
- Funktionen für eine weitere, eigene Bibliothek (Source und Headerfile) bzgl. der gegebenen Aufgabenstellung programmieren,
- mit Eingabe- und Ausgabefiles arbeiten,
- Ihre einzelnen Module/Funktionen selbständig testen.
- Ihren Code ausreichend dokumentieren, in den Headerfiles die Parameterliste erläutern und beschreiben, was die jeweilige Funktion macht (doxygen[2]; kurze Einführung[3]).

Was soll Ihr Code können?

a) Für bestimmte Teilaufgaben sind Koeffizienten von dem gegebenen ASCII-File einzulesen bzw. Daten auszulesen. Hierzu können Sie das Modul *file_io*[4] benutzen, dessen Files sind als zusätzliche Files in Ihr Projekt einzubinden und **im Verzeichnis** `bsp_19` zu speichern (keine Unterverzeichnisse).

b) Für die gegebenen Funktionen $p(x)$, $q(x)$, $s(x)$ sind **alle reellen Nullstellen** im jeweils angegebenen Intervall numerisch zu bestimmen. Bei der Nullstellenbestimmung können Sie sich am Bisektionsverfahren[5] (auch im Skript) orientieren oder etwas anderes ausprobieren. Dafür sollen Sie ein Modul (Header-und Sourcefile) für die Nullenstellensuche schreiben wobei die jeweils auszuwertende Funktion als Funktionsparameter, z.B., als Funktional[6], übergeben werden muß.

c) Für jede gewählte Funktion $f(x)$ sind die gefundenen Nullstellen (x-Werte) in separaten ASCII-Files abzuspeichern.
In Matlab/Octave sind $p(x)$, $q(x)$, $s(x)$ gemeinsam mit den gefunden Nullstellen zu visualisieren und die Graphik als *jpg*-File zu speichern. Nutzen Sie die Matlab-Fkt. `importdata`[7] oder `readmatrix`[8] um jeweils die Nullstellen (Vektor) von einem ASCII-File einzulesen.

Zu benutzende Funktionen:

- $p(x) := \cos(x) \cdot \left(x^4 - \frac{x^3}{2} + \frac{x^2}{2} - \frac{x}{2} - \frac{1}{2}\right)$ im Intervall $[-2.5, 2]$. Ohne den Kosinusfaktor ergeben sich die reellen Nullstellen $x = -0.5$ und $x = 1$.
- $q(x) := \sin(1/x)$ im Intervall $[\frac{1}{800}, 1]$.
  Wieviele Nullstellen gibt es im Intervall (analytisch bestimmbar!)?
- $s(x) := \sum_{k=0}^{n} w_k \cdot x^k$ im Intervall $[-6, 15]$ wobei die Koeffizienten w_k in der Reihenfolge $w_0, w_1, \ldots, w_n$ im Programm vom File *input_1.txt*[9] eingelesen werden müssen.
  Es gibt 7 Nullstellen, eine ist $x = -1.23456789$.
  Die Funktion $s(x)$ muß über eine Lambda-Funktion oder über die Methode einer Klasse realisiert werden.

[2] `http://www.doxygen.nl/manual/`
[3] `http://kapo-cpp.blogspot.com/2007/10/documenting-code-using-doxygen.html`
[4] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/file_io.zip`
[5] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Beispiele/Bisect3.cpp`
[6] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Beispiele/Bisect3.cpp`
[7] `https://de.mathworks.com/help/matlab/ref/importdata.html`
[8] `https://de.mathworks.com/help/matlab/ref/readmatrix.html`
[9] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/input_1.txt`

Wie bestimme ich alle Nullstellen im Interval $[a, b]$?

Es gibt keinen allgemeingültigen Algorithmus zur globalen Bestimmung aller Nullstellen einer allgemeinen Funktion $f(x)$ im Interval $[a, b]$ $(a < b)$.

Eine Lösungsidee (brute force):

(a) Wenn sich für eine stetige Funktion $f(x)$ in einem (Teil-)Intervall $[x_i, x_{i+1}] \subset [a, b]$ die Vorzeichen von $f(x_i)$ und $f(x_{i+1})$ unterscheiden, dann muß mindestens eine Nullstelle im Intervall zu finden sein.
Dann kann man, z.B., mit Bisektion, eine dieser Nullstellen bestimmen.

(b) $\Longrightarrow$ Unterteilung von $[a, b]$ in n (gleichgroße) Teilintervalle und Bestimmung max. einer Nullstelle pro Teilintervall.

(c) Erhöhung der Intervallanzahl n ($*4$ oder $*10$) und wie unter (b) verfahren.

(d) Die Schritte (c) und (b) solange wiederholen bis sich die Anzahl der gefundenen Nullstellen nicht mehr ändert.

Auch dieser Algorithmus garantiert nicht, daß alle Nullstellen gefunden werde, er ist aber eine brauchbare Heuristik.
Bei mehr Information über die Funktion $f(x)$ und Ausnutzung dieser vorhandenen Strukturen kann man bessere Algorithmen zur Nullstellenbestimmung ableiten.

Abgabe und Bewertung

Geben Sie alle Files (inkl. des/der Matlab-Files und Grafiken) in einem Projektfolder *bsp_19* ab.
Es werden insgesamt 12 Punkte vergeben welche *in etwa* so verteilt werden: (12 Pkt.)

1 Pkt. Module korrekt benutzt,

6 Pkt. korrekte Nullstellenberechnung,

2 Pkt. Dokumentation Ihres Modules,

2 Pkt. Speicherung der berechneten Daten in Files und deren Visualisierung,

1 Pkt. Lambda-Funktion mit Capture[10] bei $s(x)$ (oder als Methode einer Klasse und mit `bind`[11]).

Hinweise:
Die Schlüsselwörter `continue, goto` dürfen nicht benutzt werden, `break` nur in Verbindung mit der `switch`-Anweisung. Desgleichen darf ein `return` nur als letzte Anweisung einer Funktion vorkommen.

Die Abgabe der Lösungen (`*.cpp`-Files, `*.h`, (`Makefile*`,) ...,) erfolgt an der KFU über Moodle, siehe dazu die Hinweise auf der LV-Homepage[12].
Die Verzeichnisnamen mit den Files für die jeweilige Aufgabe **müssen** dem Schema `bsp_nummer` folgen, z.B. enthält das Verzeichnis (der Ordner) `bsp_1` alle Files für Beispiel 1. Andere Verzeichnisnamen zählen als nicht abgegeben.
Keine Leerzeichen, Sonderzeichen oder Umlaute in File- und Verzeichnisnamen benutzen (Portabilität).

[10] `https://www.heise.de/developer/artikel/C-Core-Guidelines-Funktionsobjekte-und-Lambdas-3848097.html`
[11] `https://stackoverflow.com/questions/16016112/stdbind-of-class-member-function`
[12] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/index.html`