3. Übung des Programmierpraktikums

Abgabetermin: 29. April 2024, 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.

---

13. Übernehmen Sie die Funktion `geheimZahl`[1] (docu[2]) aus der Vorlesung welche einen `vector` zurückgibt, in dem alle Rateversuche gespeichert sind. (2 Pkt.)

    Entsprechend des Hinweises über dem Strich, sollen Sie für diese (und alle weiteren) Aufgaben eine Trennung in Headerfile und Sourcefile, wie in **§7.5 des Skriptes** beschrieben, vornehmen. Ihr Projekt enthält damit die folgenden Files:

    - Das File mit dem Hauptprogramm `main()`: *main.cpp* oder *bsp_13.cpp*.
      Das Headerfile der Funktionen, s.u., muß statt der Funktionsdeklaration inkludiert werden (`#include "bsp_fkt_13.h"`).
    - Das Headerfile mit den Funktionsdeklarationen und deren Dokumentationen: *bsp_fkt_13.h*.
    - Das Sourcefile mit den Funktionsdefinitionen, d.h., den Funktionsimplementierungen: *bsp_fkt_13.cpp*.

    Nutzen Sie die Möglichkeiten von Code::Blocks zum Erzeugen einer Headerdatei (header guarding[3]) und fügen Sie diese und das Sourcefile zum Projekt hinzu, d.h., Ihr Projekt besteht aus den obigen drei Files.

    Testen Sie die Funktion `geheimZahl` im Hauptprogramm mit einer Geheimzahl $x \in [a, b]$ mit $a < 0 < b$ und geben Sie sowohl die Anzahl der Versuche als auch alle geratenen Zahlen im Hauptprogramm aus.

    Schreiben Sie **eine weitere Funktion**, welche den Minimalwert und den Maximalwert dieses Vektors zurückgibt, diese Werte müssen ebenfalls im Hauptprogramm ausgegeben werden. Entwerfen und programmieren Sie diese Funktion **ohne** Nutzung der STL-Algorithmen, fügen Sie Deklaration und Definition in Header- bzw. Sourcefile hinzu.

---

[1] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS24/v_3a.zip`
[2] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS24/v_3a/html`
[3] `https://de.wikipedia.org/wiki/Include-Guard`

14. Schreiben Sie eine Funktion welche die (Partialsummen-)Folge

$$x_n = \frac{1}{n^2} \sum_{k=1}^{n} \frac{2k^3 - 4k^2 - 6k}{n + k + nk + 1} \qquad \forall n = 1, \ldots, m$$

für die ersten m Glieder berechnet und diesen Vektor an das aufrufende Programm zurückgibt.

- Testen Sie die Funktion aus dem Hauptprogramm heraus (Ausgabe der letzten 3 Folgenglieder im Hauptprogramm). (2 Pkt.)
- Schreiben Sie eine zweite Funktion für obige Folge welche mit möglichst wenig Arithmetik auskommt. Hier ist Mathematik/Matlab/sage sehr nützlich, um die Summation über k durch eine Formel zu ersetzen.
- Konvergiert obige Folge, und wenn ja gegen welchen Grenzwert?
  Beweisen Sie Ihre Vermutung!
- Es sind wieder mindestens 3 Files abzugeben.

  Eingabedaten: $m = 100$, $m = 10\,000$, $m = 1\,000\,000$

15. Schreiben Sie eine Funktion für das Spiel „Stein, Schere, Papier [, Brunnen]“. Wenn Sie das Spiel nicht kennen, recherchieren Sie im Internet.
  Eine der Möglichkeiten zum Lösen der Aufgabe beinhaltet die `switch`[4]-Anweisung, es geht aber auch ganz kompakt ohne `switch`- oder `if`-Anweisungen, oder auch mit einer kleinen Matrix. Der Rechner, als Gegenspieler des Benutzers, soll zufällige Züge machen (d.h. nach dem Zufallsprinzip entweder den Stein, das Papier, die Schere [oder den Brunnen] für den nächsten Zug auswählen) [Str10, p.154]. Sie geben Ihre Wahl per Tastatur ein. (4 Pkt.)
  Spielen Sie n-mal gegen den Computer, wobei Sie jedesmal den Sieger ausgeben. Geben Sie den Gesamtsieger aus indem Sie die von Mensch bzw. Computer gewonnenen Spiele summieren.

  Hinweise:

  - Zerlegen Sie die Aufgabe zuerst in kleinere Teilaufgaben (Blöcke) für welche Sie sich grob überlegen, welche Input/Output-Daten sie dafür jeweils benötigen.
  - Wie stellen Sie die Auswahl Schere, Stein, Papier [, Brunnen] dar (Datentyp,Bereich)?
  - Wie realisieren Sie zufällige Züge des Computers?
    Tip: alte C-Generatoren `rand`, `srand` oder neue C++-Generatoren wie `uniform_int_distribution` oder `minstd_rand0` verwenden, siehe §13.7 des Skriptes[5].
  - Schreiben Sie eine komfortable Eingabefunktion, die Ihnen mitteilt welche Auswahl Sie haben und die Eingabe auf den zulässigen Bereich einschränkt.
  - Tip: Nutzen Sie Indizes (evtl. in Verbindung mit Vektoren) zur Speicherung der Auswahl und der Bestimmung des Gewinners.
  - Es sind wieder mindestens 3 Files abzugeben.

[4] `http://www.tutorialspoint.com/cplusplus/cpp_switch_statement.htm`
[5] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf`

16. Die Anzahl der möglichen Kombinationen für k zu tippende Zahlen aus n Zahlen wird durch den folgenden Binomialkoeffizienten ausgedrückt: (2 Pkt.)

$$C_n^{(k)} = \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

- Schreiben Sie eine **Funktion** `binom` zur Berechnung obigen Binomialkoeffizientens (der Funktionsname ist frei wählbar). Die Funktion besitzt **ganzzahlige Input- und Outputparameter**.
  - Beachten Sie, daß die Fakultät ($n!$) in C++ nicht verfügbar ist. Es gibt (mind.) zwei Möglichkeiten bzgl. des Binomialkoeff. dies zu beheben.
    Hinweis: Wie berechnet man im Kopf $\binom{9}{2}$ bzw. $\binom{9}{7}$ ohne Fakultät?
  - Geben Sie die Anzahl der möglichen Kombinationen $C_n^{(k)}$ für die Eingabedaten im `main()` aus und kontrollieren Sie Ihr Ergebnis (Internet, Matlab[6]).

  (*) Vorsicht bei Verwendung der Gamma[7]-Funktion:
  Falsches Ergebnis kann auftreten, z. B. bei $\binom{45}{1}$ oder $\binom{45}{6}$, ist einfach vermeidbar.

  Eingabedaten (n, k): $(9, 2)$, $(66, 3)$, $(45, 1)$, $(45, 6)$, $(50, 45)$
- Schreiben Sie eine **weitere Funktion** welche die k-te Zeile eines Pascalschen Dreiecks als Vektor zurückliefert.
  - Funktion IN: k OUT: vector<Ganzzahl>.
  - Geben Sie die 9. und die 61. Zeile des Pascalschen Dreiecks in `main()` aus.
  - Geben Sie die berechneten Zeilensummen der 9. und die 61. Zeile in `main()` aus.
- Es sind wieder mindestens 3 Files abzugeben.

17. Wir spielen Lotto und wollen wissen, wie groß die Wahrscheinlichkeit $p \in [0, 1]$ ist, daß bei einem Tipschein mit k zu tippenden Zahlen aus n Zahlen, r Zahlen richtig sind und z Zusatzzahlen richtig getippt wurden ($z + r \leq k$). Wir nehmen im weiteren an, daß es nur eine Zusatzzahl gibt. Die entsprechende Formel[8] für die Wahrscheinlichkeit ist (2 Pkt.)

$$p = \frac{\binom{k}{r}\binom{1}{z}\binom{n-k-1}{k-r-z}}{\binom{n}{k}} \tag{2}$$

und diese soll in einer **Funktion** implemeniert werden (Funktionen aus Aufg. 16 dürfen/sollten verwendet werden).

- Erstellen Sie für die Eingabedaten eine Übersicht über die Wahrscheinlichkeit, daß Sie r-Zahlen und z Zusatzzahlen richtig haben, mit $r \in [0, k]$ und $z \in [0, 1]$.
- Vergleichen Sie bei 6 aus 49 mit der zweiten Tabelle in §5.1 der *Lotto-Webseite*[9]. Achtung, in der Internettabelle müssen die beiden Tabelleneinträge der Wahrscheinlichkeit für "6 Richtige, Superzahl falsch"und der Wahrscheinlichkeit für "6 Richtige mit Superzahl"addiert werden.
- Nutzen Sie Ihre Funktionen aus Aufg. 16 in dieser Aufgabe, Sie können dieses Files per Copy-Paste benutzen.
- Es sind wieder mindestens 3 Files abzugeben.

Eingabedaten (n, k): $(49, 6)$, $(90, 6)$, $(20, 4)$, $(10, 2)$

[6] `https://de.mathworks.com/help/symbolic/nchoosek.html`
[7] `https://en.cppreference.com/w/cpp/numeric/math/tgamma`
[8] `http://de.wikipedia.org/wiki/Lotto`
[9] `http://de.wikipedia.org/wiki/Lotto`

Die Abgabe der Lösungen (`*.cpp`-Files, `*.h`, (`Makefile*`,) ...,) erfolgt an der UNI Graz über Moodle, siehe dazu die Hinweise auf der LV-Homepage[10].

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://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS24`