2. Übung des Programmierpraktikums

Abgabetermin: 15. 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 mit Namen `bsp_`*`nummer`*, d.h., der Ordner `bsp_2` gehört zu Aufgabe 2.

---

7. Berechnen Sie die alternierende Summe aller Kehrwerte der natürlichen Zahlen zwischen $n1$ und $n2$ ($n2 >= n1$) für selbstgewählte $n1$ und $n2$, also $\sum_{k=n_1}^{n_2} \frac{(-1)^k}{k}$ (2 Pkt.)

   über die folgenden 4 Konstrukte:

   a) einer `for`-Schleife,
   b) einer `while`-Schleife,
   c) einer `do while`-Schleife,
   d) einer `for`-Schleife in umgekehrter Summationsreihenfolge, d.h. $\sum_{k=n_2}^{n_1} \frac{(-1)^k}{k}$.

   Lesen Sie dazu §4.4-4.6 des Skriptes[1].

   Weshalb sind die Resultate von a) und d) für **float**-Daten und $n_2 \approx 10^6$ (mit z.B., $n_1 = 1$) unterschiedlich?
   Welches Resultat ist genauer?

8. Fünf Personen haben versucht, die Summe der Zahlen 1 bis 100 zu berechnen [Bre17[2],p.83]. Beurteilen Sie die folgenden Lösungsvorschläge (ohne diese zu programmieren!) bzgl. Korrektheit und weiterer möglicher Fallen. Begründen Sie Ihre Einschätzung! (1 Pkt.)

   (a)
   ```
   int  n = 1, sum = 0;
   while(n <= 100) {
     ++n;
     sum += n;
   }
   ```

   (b)
   ```
   int  n = 1, sum = 1;
   while(n < 100) {
     n += 1;
     sum += n;
   }
   ```

   (c)
   ```
   int  n = 100;
   int  sum = n*(n+1)/2;
   ```

   (d)
   ```
   int  n = 1;
   while(n < 100) {
     int sum = 0;
     n = n + 1;
     sum = sum + n;
   }
   ```

   (e)
   ```
   int  n = 1, sum = 0;
   while(n <= 0100) {
     sum += n;
     ++n;
   }
   ```

   Geben Sie ein File *main.cpp* (im Ordner *bsp_8*) ab, in welches Sie Ihre Einschätzungen per C++-Kommentar hineinschreiben.

---

[1] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf`
[2] `http://www.cppbuch.de/`

9. Programmieren Sie eine Funktion (oder mehrere)

```
bool is_palindrom(const string &s);
```

welche an das aufrufende Hauptprogramm zurückgibt, ob der String `s` ein Palindrom[3] ist, d.h., vorwärts und rückwärts gelesen das gleiche ergibt. Testen Sie Ihre Funktion im Hauptprogramm. (1 Pkt.)
Hierbei sind (wie immer) keine `break`, `continue`, `goto` erlaubt.

- Wir nehmen an, daß nur Kleinbuchstaben verwendet werden (nicht darauf testen!).
- Finden Sie eine Lösung, die möglichst wenige Tests auf Gleichheit einzelner Buchstaben erfordert.
- Pflicht: Programmieren Sie die Lösung in mit einem selbstgeschriebenen Loop, ohne Nutzung der STL-Algorithmen.
- Kür: Nutzen sie den STL-Algorithmus `equal`[4] für den Vergleich.
- Sie die n die (gerade) Anzahl der Buchstaben (`char`) im String. Dann benötigt die ideale Lösung $<= n/2$ Vergleiche und keinerlei Kopieroperationen.

Eingabedaten (mystring): radar, abca, rentner, otto, abrakadabra

10. Wir schreiben Funktionen welche argtanh(x) (Umkehrfunktion von Tangens hyperbolicus), $|x| < 1$ mittels einer Reihe approximieren:

$$\text{argtanh}x \approx t_n := \texttt{reihe}(x,n) := x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \cdots + \frac{x^{2n-1}}{2n-1}$$

Entwerfen Sie **zuerst** ein Struktogramm der jeweiligen Funktion **bevor** Sie zu programmieren beginnen.

- Schreiben Sie eine Funktion, welche `reihe`(x,n) für ein fixes n berechnet, d.h., x, n sind Input der Funktion und der resultierende Wert stellt den Output dar. Die math. Voraussetzung an x ist in der Funktion zu überprüfen.
- Testen Sie Ihre Funktion vom Hauptprogramm heraus.
- Schreiben Sie **unter Nutzung obiger Funktion** eine zweite Funktion, welche für ein übergebenes x und ε, mit $n = 2$ beginnend, die Approximationen $t_n := \texttt{reihe}(x,n)$ berechnet und n solange erhöht, bis $|\texttt{reihe}(x,n-1) - \texttt{reihe}(x,n)| < \varepsilon$ erfüllt ist. Speichern Sie die berechneten t_n in einem dynamischen[5] Vektor `vector<double>`[6] (siehe auch § 2.3.4 des Skriptes). Die Funktion soll diesen dynamischen Vektor an das aufrufende Programm zurückliefern. (2 Pkt.)
- Beide Funktionen sind für die Verarbeitung mit doxygen zu **dokumentieren**!
- Geben Sie im Hauptprogramm x ein und rufen Sie die zweite Funktion mit $\varepsilon = 10^{-6}$ auf. Geben Sie anschließend den Abbruchindex n und die letzten beiden Werte des Vektors t im Hauptprogramm aus.

[3] `https://de.wikipedia.org/wiki/Palindrom`
[4] `https://en.cppreference.com/w/cpp/algorithm/equal`
[5] `https://en.cppreference.com/w/cpp/container/vector/push_back`
[6] `https://en.cppreference.com/w/cpp/container/vector`

11. **Summation of specified numbers**:

Write a function with input parameter n that adds all those positive integers less or equal n which are a multiples of 3 or of 5 (including or!). (2 Pkt.)

- The easiest approach uses a for-loop.
- Test your function in the main function with various parameters:
  - $n = 15$ results in 60.
  - $n = 1001$ results in 234 168.
  - $n = 1432987$ results in 479 139 074 204.
- Derive a formula for calculating the required sum without executing a loop. Implement it in a second function and test it.
- Compare the run time of your two functions by using the `chrono`[7] functions for time measurement, see also §13.2 of the Script[8].
  Run each function at least 1000 times to get some measurable timings.
  Alternatively, you can include my header file *timing.h*[9]. See its documentation[10] and Listing 13.2 in the script for its use.

Hints: `#include "timing.h"`, `std::chrono::system_clock::now()`, `std::chrono::duration<double>`, `std::chrono::duration<double>.count()`

[7] `https://en.cppreference.com/w/cpp/chrono/system_clock/now`
[8] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf`
[9] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Beispiele/utils/timing.h`
[10] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Beispiele/utils/html`

12. Schreiben Sie eine **Funktion** welche die quadratische Gleichung

$$a \cdot x^2 + b \cdot x + c = 0$$

für **beliebige** $a, b, c \in \mathbb{R}$ mit $|a| + |b| \neq 0$ löst und geben Sie die entsprechende(n) Lösung(en) in $\mathbb{R}$ (d.h., wir betrachten keine komplexen Lösungen) an das aufrufende Programm zurück. (2 Pkt.)

- Überlegen Sie zuerst, welche Fälle und damit welche Lösungsmengen es gibt.
- Zeichnen Sie ein Struktogramm (mit abgeben oder zur Übung mitbringen).
- Überlegen Sie welche Input- und Output-Parameter ihre Funktion haben muß.
  Achtung: Anzahl der Lösungen ist flexibel.
  $|a| + |b| \neq 0 \Longrightarrow$ Output-Parameter an die aufrufende Funktion nötig.
- Jetzt (und nicht eher!) dürfen Sie die IDE starten.
- Testen Sie den Code mit **allen** gegebenen **Testdaten**, geben Sie die Lösungen im Hauptprogramm (oder besser in einer weiteren Testfunktion) aus und überlegen Sie sich weitere Testdaten, sodaß alle Verzweigungen Ihres Codes getestet werden.
- **Dokumentieren** Sie Ihre Funktion entsprechend des Vorlesungsbeispiels (Source[11] (zweites Bsp.[12]), resultierendes html[13]) und der doxygen-Dokumentation[14].
  Erzeugen Sie die html-Dokumentation mit doxygen[15] (Einstellung beim Expert-Build: `EXTRACT_ALL`) aus der IDE heraus.

(∗) Überprüfen Sie, ob Sie für (1, -3-fLarge, 3*fLarge) mit fLarge=2e+16 die Lösungen $\{3, 2e{+}16\}$ erhalten.
Wenn nicht, dann den Satz von Vieta[16] benutzen, um die Stellenauslöschung bei der Subtraktion großer Zahlen zu vermeiden [Gander2014, p.55].
Auch mit (1, fLarge-3, -3*fLarge) ausprobieren ($\Longrightarrow$ {-2e+16, 3})

Eingabedaten (a,b,c):
$(1.0, -4.0, 1.0)$, $(2.0, -100, 1200)$, $(0.0, 3.0, 4.5)$, $(3.0, -6.0, 51.0)$, $(0, 0, 4)$

Literatur

[Gander2014] Gander, Walter and Gander, Martin and Kwok, Felix. Scientific Computing. Springer, TCSE 11, (2014).

Die Abgabe der Lösungen (`*.cpp`-Files, `*.h`, (`Makefile*`,) ...,) erfolgt an der UNI Graz über Moodle, siehe dazu die Hinweise auf der LV-Homepage[17].

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).

---

[11] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS24/v_1b/main.cpp`
[12] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS24/v_2a/main.cpp`
[13] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS24/v_1b/html/main_8cpp.html`
[14] `https://www.doxygen.nl/manual/docblocks.html#cppblock`
[15] `http://www.doxygen.org/index.html`
[16] `https://de.wikipedia.org/wiki/Satz_von_Vieta`
[17] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Download/kfu.html`