1. Übung des Programmierpraktikums

Abgabetermin: 21. März 2022, 23:59 Uhr

Die Übungen sind grundsätzlich selbst zu lösen.
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.

---

1. Legen Sie in der IDE (Integrated Development Environment) `Code::Blocks` ein neues Projekt (Typ: **Console Application**) `bsp_1` mit dem C++-Quelltextfile `main.cpp` (oder `bsp_1.cpp`) an. Der Ordner `bsp_1` wird dabei automatisch mit diesem Projekt neu angelegt.

   - Über die Menüeinträge $\longrightarrow$ „`Build`" $\longrightarrow$ „`Build and run`" wird dieses Programm übersetzt und ausgeführt. Testen Sie dies. (1 Pkt.)
   - Ersetzen Sie den gesamten Quelltext in `main.cpp` durch den Quelltext des Files demoVector.cpp[1], übersetzen und führen Sie das Programm aus.
   - Fügen Sie die Zeilen

     ```
     cout << exp(abs(bb[3])) << endl;

     char ch;
     cout << "Eingabe eines Zeichens: "
     cin >> ch;

     const double c5 = c.at(5);
     cout << "c[5] : " << c5 << endl;
     ```

     vor der `return`-Anweisung in Ihr geändertes Quelltextfile ein.
     - Übersetzen (Compilieren+Linken: *Build*) Sie Ihr Projekt. Lesen Sie die ersten Zeile des Compilerfehlers und korrigieren Sie den Syntaxfehler. Wiederholen Sie diesen Vorgang solange bis der Code fehlerfrei übersetzt wird.
     - Führen Sie das Programm aus (*Run*). Suchen und beheben Sie den Laufzeitfehler.
       Hinweis: In C/C++ beginnt die Indizierung von Vektoren etc. mit 0.
   - Setzen Sie einen Breakpoint in Zeile 17 des Quellfiles und nutzen Sie die Debugging[2]-Möglichkeiten der IDE, um das Programm schrittweise abzuarbeiten und die Veränderung der Variablen aa, cc, i im weiteren Programmverlauf zu beobachten.

2. Legen Sie ein neues (Consolen-)Projekt `bsp_2` an und schreiben Sie ein Programm welches den Widerstand[3] $R = \varrho \cdot \ell / A$ eines Leiters aus dem spezifischen Widerstand ϱ $[\frac{\Omega mm^2}{m}]$, der Länge ℓ $[m]$ und dem Querschnitt A $[mm^2]$ des Leiters berechnet.

   Wie nehmen Kupfer als Leitermaterial an, siehe www[4] für den konstant zu wählenden Wert ϱ. Die Werte für ℓ und A sind via `cin` auf `double`-Variablen einzulesen.
   Geben Sie das Ergebnis R via `cout` im Terminal aus.

   Testdaten (ℓ, A): $(10, 2.5)$, $(2.5, 10)$

   Achtung: Eine Gleit*komma*zahl heißt im englischen „floating *point* number", daher muß der Dezimal*punkt* statt des Kommas bei der Zahleneingabe angegeben werden.
   Wissenschaftliche Schreibweise: $17,86 * 10^{-3}$ $\longrightarrow$ Computer: `17.86e-3`

---

[1] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Beispiele/demoVector.cpp`
[2] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Download/kfu.html`
[3] `http://www.e-formel.info/elektrotechnik/leitungswiderstand.html`
[4] `https://www.xplore-dna.net/mod/page/view.php?id=1291`

3. Legen Sie ein neues Projekt `bsp_3` an, sie können den Vorlesungscode (intro_function) benutzen.

   - Schreiben Sie vor die Funktion `int main()` in `main.cpp` eine **Funktion** welche den Widerstand eines kreisförmigen Leiters analog zu Bsp. 2 aus dessen Länge $[m]$ und Durchmesser d $[mm]$ berechnet.
     Input: ϱ, ℓ, d
     Output: R
     Der berechnete Wert für R ist nicht in der Funktion, sondern nur im Hauptprogramm auszugeben. Die Testdaten können Sie im Hauptprogramm als Konstanten definieren, das erspart das wiederholte Eingeben.
     Testdaten (ϱ, ℓ, d): (1.786e-2, 10, 1.784), (27.8e-3, 1000, 17.841)
   - Scheiben Sie eine zweite **Funktion** welche den Widerstand in Abhängigkeit von der Temperatur T $[°C]$ für einen kreisförmigen Kupferleiter berechnet. Die Formel

     $$R(T) = R_{T_0} \cdot (1 + \alpha \cdot (T - T_0))$$

     und die Materialdaten für α, ϱ sind im im www[5] zu finden, T_0 wird mit $20°C$ angenommen.
     Input: ℓ, d, T
     Output: R
     Verwenden Sie als R_{T_0} die erste Funktion dieser Aufgabe, d.h. rufen Sie diese erste Funktion in Ihrer zweiten Funktion auf. Die Werte von α, ϱ können als Konstanten innerhalb dieser zweiten Funktion definiert werden.
     Der berechnete Wert für R ist nicht in der Funktion, sondern nur im Hauptprogramm auszugeben.
     Testdaten (ℓ, d, T): (10, 0.8, 40), (1000, 7.98, -20)

   Achtung: **Keine Leerzeichen oder Umlaute** in Variablennamen oder Filenamen benutzen.

4. Berechnungen des arithmetischen, des geometrischen und des harmonischen Mittels:

   (a) Sie können mit dem Code (intro_function) beginnen und ein **Funktion** hinzufügen, welche (2 Pkt.)
   - drei ganze Zahlen als Inputparameter über Inputparameterliste erhält,
   - die drei Mittel berechnet, und
   - die drei Werte über die Parameterliste an das aufrufende Programm zurückgibt.

   (b) Rufen Sie die Funktion von Ihrem Hauptprogramm auf und geben Sie dort die Inputgrößen und die Resultate aus.

   (c) Inputdaten $(1, 4, 16)$ ergeben die drei Mittel $(7, 4, 2.28571)$.

   (d) Inputdaten $(2, 3, 5)$ ergeben die drei Mittel $(3.33333, 3.10723, 2.90323)$.

   (e) Überprüfen Sie die Korrektheit für $(1000, 4000, 16000)$, abgesehen von der beschränkten Genauigkeit der Gleitkommazahlen.

   (f) Die zweite **Funktion** besitzt dieselbe Funktionalität wie 4a) aber mit einem Vektor beliebiger Länge als Inputparameter statt der drei einzelnen Zahlen, siehe §2.3.3, §7.4 des Skriptes[6]. Generieren Sie sich einen Vektor mit mind. 10 Elementen (des Datentyps `int` oder `double`) und überprüfen Sie die Ergebnisse.

   Hints: `#include <cmath>`, `pow`, `#include <vector>`, `vector`[7], `#include <string>`, `string`

[5] `https://www.xplore-dna.net/mod/page/view.php?id=1291`
[6] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf`
[7] `https://en.cppreference.com/w/cpp/container/vector`

5. Überprüfen Sie, ob die folgenden Rechenregeln der Mathematik auch im Computer gelten. Die verfügbaren math. Funktionen sind unter *cmath*[8] zu finden und werden über `#include <cmath>` in Ihren Quelltext eingebunden, π wird von den meisten Compilern in diesem Headerfile als `M_PI` bereitgestellt. Lesen Sie dazu §3.6 des Skriptes[9] und auf cplusplus[10].

$$\begin{aligned}
\frac{(\mathrm{e}^a)^b}{\mathrm{e}^2} &= \mathrm{e}^{a\cdot b-2} \\
\frac{b}{\sqrt{a+b}-\sqrt{a}} &= \sqrt{a+b}+\sqrt{a} \\
\pi/2-\arcsin a &= \arctan\frac{\sqrt{1-a^2}}{a} \\
\cosh(\mathrm{arsinh}(b)) &= \sqrt{b^2+1}
\end{aligned}$$

- Schreiben Sie ein oder zwei kleine Funktionen welche den absoluten und den relativen Fehler[11] zweier einfach genauer[12] Zahlen (`float`) a und b (Input-Parameter) als Output-Parameter zurückgeben.
- Testen Sie obige Rechenregeln indem Sie die rechte und die linke Seite der Gleichungen jeweils einer Variablen zuweisen und neben dem Wert auch absolute und relative Genauigkeit ausgeben. (1 Pkt.)
- Geben Sie die Anzahl von Byte zur Speicherung in einer Variablen vom gewählten Datentyp unter Nutzung von `sizeof`[13] an und geben Sie die relative Genauigkeit (Maschinen-ε[14]) füur Ihren Datentyp analog zum Beispiel `Ex390.cpp`[15] an.

Hinweise: `exp`, `log`, `sqrt`, `pow`, `asin`, `atan`, `asinh`, `cosh`, `M_PI`, `sizeof`, `numeric_limits`
Testdaten (a,b): (0.9876, 1.762e-6) bzw. (0.678, 98),

Für Fortgeschrittene: Packen Sie obige Aufgabe in eine Template-Funktion welche Sie sowohl für `float` als auch für `double` und `long double` ausprobieren, siehe §10.1.1 im Skript[16] oder auf LernCpp.com[17].

[8] https://en.cppreference.com/w/cpp/header/cmath
[9] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf
[10] http://www.cplusplus.com/reference/cmath/?kw=cmath
[11] https://de.wikipedia.org/wiki/Fehlerschranke#Absoluter_Fehler
[12] https://de.wikipedia.org/wiki/Einfache_Genauigkeit
[13] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Beispiele/DataTypes.cpp
[14] https://de.wikipedia.org/wiki/Maschinengenauigkeit
[15] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Beispiele/Ex390.cpp
[16] https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf
[17] http://www.learncpp.com/cpp-tutorial/131-function-templates/

6. Berechnen Sie die folgenden Ausdrücke für gegebene komplexe Zahlen a, und b. Die verfügbaren math. Funktionen sind unter *complex*[18] zu finden und werden über `#include <complex>` wie in §2.3.2 des Skriptes[19] in Ihren Quelltext eingebunden. (1 Pkt.)
   Benutzen Sie doppelt genaue komplexe Zahlen, sodaß $-3.5+2i \Longrightarrow$ `complex<double> b(-3.5,2)`.
   Geben Sie für die gegebenen Zahlen a und b die Polarkoordinaten, d.h., den Winkel φ (`arg()`) und den Betrag r (`abs()`), in der Gaußschen Zahlenebene aus.

$$\begin{aligned} \mathrm{e}^{\pi \cdot a} &= \\ a^3 &= \\ \sqrt{b} &= \\ b + \overline{b} &= \\ \mathrm{e}^{\mathbf{i}\varphi_b} - (\cos\varphi_b + \mathbf{i} \cdot \sin\varphi_b) &= \\ 3 \cdot a - b &= \end{aligned}$$

   Testdaten (a,b): $(0 + \mathbf{i},\ -3.5 + 2 \cdot \mathbf{i})$, bzw. $(0 + \frac{2}{3} \cdot \mathbf{i},\ -3 - 4 \cdot \mathbf{i})$,

   Hinweise: `pow, log, log10, acos, atan, exp, sqrt, arg, abs, conj`

Die Abgabe der Lösungen (`*.cpp`-Files, `*.h`, (`Makefile*`,) ...,) erfolgt über KFU-Moodle, siehe dazu die Hinweise auf der LV-Homepage[20].

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. Durch die automatische Abgabekontrolle zählen andere Verzeichnisnamen als nicht abgegeben.
Keine Leerzeichen, Sonderzeichen oder Umlaute in File- und Verzeichnisnamen benutzen (Portabilität).

[18] `http://www.cplusplus.com/reference/std/complex/`
[19] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf`
[20] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Download/kfu.html`