Übungsblatt 1 19.09.2023

Organisatorisches

Bildet für die Bearbeitung dieses Übungsblatt und für alle folgenden Übungsblätter nach Möglichkeit feste Kleingruppen.

Als Übersicht und Nachschlagewerk zur Sprache C++ und zur Standardbibliothek empfehlen wir `https://en.cppreference.com/` oder `https://devdocs.io/cpp/`.

Aufgabe 0

a) Richtet euch eure Arbeitsumgebung ein. Unter Linux oder MacOSX könnt ihr einfach die benötigten Pakete für `g++` (oder `clang`), `make` und `emacs` nachinstallieren.[1]

   Wenn ihr noch Windows nutzt, könnt ihr das Windows Subsystem for Linux[2] nutzen oder euch Cygwin[3] installieren und bei Bedarf die benötigten Pakete für Emacs[4] und g++[5] nachrüsten.

b) Entpackt sodann das in Stud.IP vorgegebene Archiv `ueb1.zip` mit den Dateien für die folgenden Übungen.

Aufgabe 1

In der Datei `aufgabe_1/hello-world.cc` findet ihr ein einfaches C++-Programm mit mehreren syntaktischen Fehlern.

Versucht, alle Fehler zu beheben und das Programm mit `g++ -Wall -Werror -o hello-world hello-world.cc` zu übersetzen. Ruft es anschließend auf der Kommandozeile auf, indem ihr in dem betreffenden Verzeichnis `./hello-world` eintippt und Enter drückt.

Aufgabe 2

Schreibt ein Programm, das zwei rationale Zahlen (Datentyp `double`) über `cin` einliest und anschließend deren Produkt auf `cout` ausgibt. Falls beim Einlesen ein Fehler auftritt, soll auf dem Stream `cerr` eine Fehlermeldung ausgegeben und das Programm beendet werden.

---

[1] Wer möchte, kann statt Emacs natürlich auch eine andere Entwicklungsumgebung nutzen.

[2] `https://docs.microsoft.com/en-us/windows/wsl/install-win10` und `http://www.omgubuntu.co.uk/2016/08/enable-bash-windows-10-anniversary-update`.

[3] `https://cygwin.org`

[4] `http://cygwin.org/packages/summary/emacs-w32.html`

[5] `http://cygwin.org/packages/summary/gcc-g++.html`, `http://cygwin.org/packages/summary/make.html`

Aufgabe 3

Schreibt ein Programm `zaehlen.cc`, das mittels `getline()` einen Text von `cin` einliest und für jeden Buchstaben ohne Berücksichtigung der Groß-/Kleinschreibung protokolliert, wie oft dieser Buchstabe in dem Text vorkommt. Die Werte sollen während der Verarbeitung in einer `map` gespeichert werden, deren Schlüssel (ausschließlich) Großbuchstaben darstellen. (Für die Umwandlung eines Buchstabens in einen Großbuchstaben könnt ihr die Funktion `toupper()` nehmen, ob ein Zeichen ein Buchstabe ist, verrät `isalpha()`.)

Nachdem der gesamte Text analysiert worden ist, soll eine Übersicht über die enthaltenen Buchstaben und deren Vorkommen ausgegebenen werden.

Hinweis: Zum Testen eures Programms könnt ihr auf der Kommandozeile die Eingabeumlenkung mittels „<“ verwenden. Eine Datei mit dem Namen „fasel“ würde beispielsweise mit folgendem Kommando als Eingabe für das Programm „zaehlen“ genutzt: `./zaehlen <fasel`.

Aufgabe 4

Schaut euch in einem Editor die im Archiv `ueb1.zip` enthaltene Datei `aufgabe_4/rot13.cc` an.

a) Erläutert(!) die Funktionsweise des Programms. Was bewirkt es? Überprüft eure These, indem ihr das Programm auf der Kommandozeile mit `make` übersetzt und anschließend mittels `./rot13 <rot13.cc` ausführt.

   Hinweis: Recherchiert, welche Aufgabe die benutzten Bibliotheksfunktionen `isupper()` und `islower()` haben und was die Funktion `transform()` bewirkt.

b) Erweitert die Funktionen `f()` und `rot()` um einen Parameter „`ofs`“, der ein vorzeichenloser Integer (`unsigned int`) sein soll. Die Funktion soll nun alle Buchstaben durch den Buchstaben ersetzen, der (modulo 26) `ofs` Positionen später im Alphabet folgt (d. h. $f(x) = (x + ofs)\%26$). Legt den Parameter `ofs` für `rot` beim Aufruf über `transform` mittels `bind`[6] fest wie folgt (hier wird `ofs` auf den Wert 3 festgelegt):

   ```
   transform(buf.begin(), buf.end(), buf.begin(), bind(rot, placeholders::_1, 3));
   ```

c) Testet die neue Funktionsweise mit sinnvollen Werten für `ofs`. Wie lautet das Ergebnis für $ofs = 5174$?

[6] Für `bind()` benötigt ihr die Header-Datei `functional`.