Praktikumstermin Nr. 07, INF: HTML Datei aus Vorlage & Daten, Zeichenkette suchen rekursiv

Abgabe im GIP-INF Praktikum der Woche 22.-26.11.2021.

(Pflicht-) Aufgabe INF-07.01: Webseitendatei erstellen aus Vorlagendatei und Datendatei (Dateioperationen, Stringoperationen)

Hinweis:
Sie brauchen im Praktikum nur das resultierende Programm vorzuzeigen, welches alle Anforderungen der kompletten Aufgabe löst. Einzelne Zwischenversionen (z.B. gemäß den unten angegebenen Arbeitsschritten, falls Sie sich an diesen orientieren wollen) brauchen Sie nicht dauerhaft zu speichern bzw. im Praktikum vorzuzeigen.

Legen Sie in Visual Studio ein neues leeres C++ Projekt für diese Praktikumsaufgabe an. Den Namen des Projekts können Sie frei wählen. Legen Sie dann innerhalb des Projekts eine neue C++ Quelltextdatei (`.cpp` Datei) an. Auch hier können Sie den Namen der Datei frei wählen (Endung `.cpp`).

Öffnen Sie nun den Ordner mit den Dateien dieses Projekts im Windows Explorer: Klicken Sie dazu mit der rechten Maustaste in Visual Studio auf den Projektnamen, dann *"Ordner in Datei-Explorer öffnen"* (zweiter Eintrag von unten in dem Kontextmenü des Projekts) auswählen.

Laden Sie folgende Dateien aus dem GIP Praktikumsordner in Ilias herunter (erst einmal in einen beliebigen Zielordner):

`webseite.html.tmpl`, die Webseiten-Vorlagendatei (*„Vorlage" = „Template"*)
Achtung: Diese Datei heißt nach dem Herunterladen aus Ilias aus Sicherheitsgründen leider `webseitehtmltmpl.sec`. Sie müssen die heruntergeladene Datei also umbenennen in `webseite.html.tmpl`.

`personendaten.txt`, die Datendatei.

Kopieren oder verschieben Sie diese Dateien in den Ordner mit den Dateien Ihres Projekts.

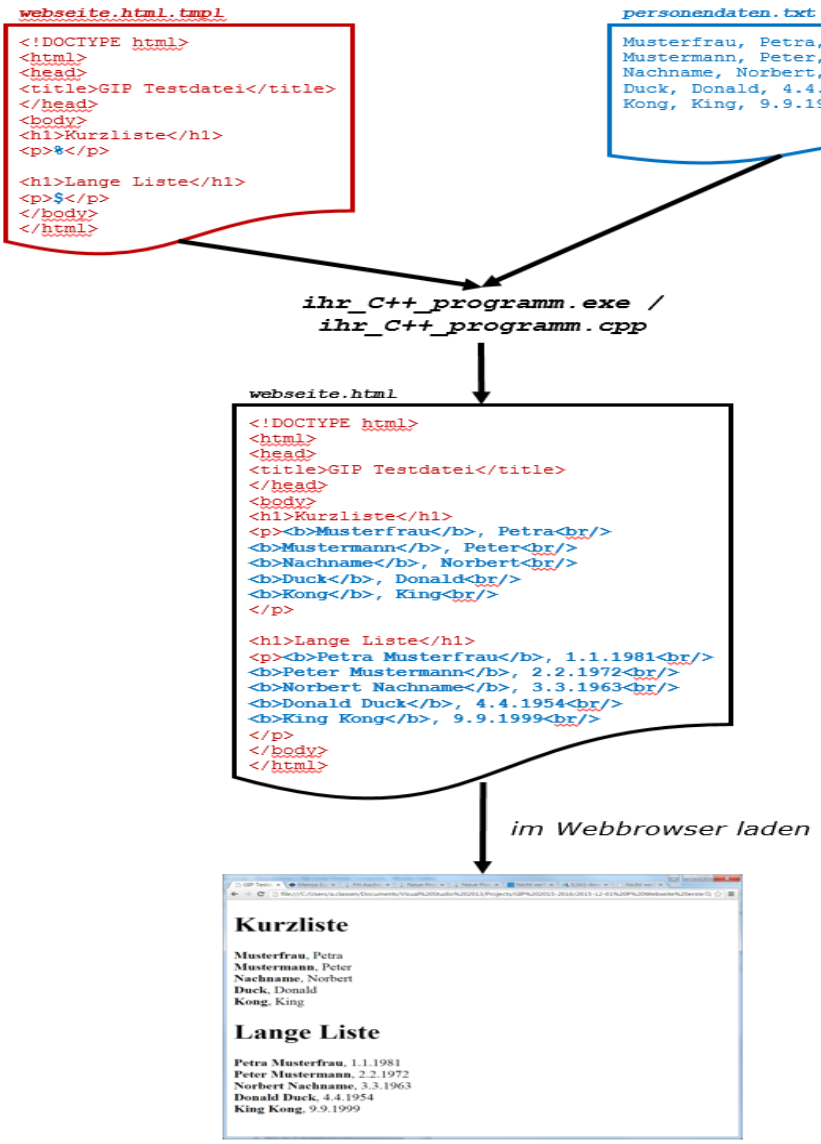
Gesamtaufgabe:
Schreiben Sie ein C++ Programm, welches die beiden Dateien einliest und

mittels der Daten aus den beiden Dateien eine neue Ausgabedatei `webseite.html` schreibt, gemäß den unten beschriebenen Regeln.

Beim Inhalt der Ausgabedatei `webseite.html` wird es sich um eine Webseite im HTML Format handeln. Öffnen Sie zur Überprüfung, ob Ihr Programm richtig gearbeitet hat, die Ausgabedatei in einem Webbrowser (z.B. in Firefox per Tastendruck `Strg-O`), um sie entsprechend aufbereitet anzeigen zu lassen.
Sollten Sie Ihr C++ Programm erneut starten, so können Sie den aktualisierten Inhalt der Datei mittels "*Reload*" im Webbrowser neu anzeigen lassen.

Das Vorgehen Ihres C++ Programms bei der Erzeugung der Ausgabedatei sei dabei wie folgt:

- Im Prinzip wird der Inhalt der Vorlagendatei gelesen und meist unverändert in die Ausgabedatei geschrieben.
- Kommt in einer Zeile der Vorlagendatei das Zeichen `%` vor, so soll dieses einzelne Zeichen ersetzt werden durch folgenden längeren Text:
  Den Text für die Kurz-Liste der Personen. Jeder „kurze" Personeneintrag der Liste besteht aus dem Nachnamen (aus der Datendatei) in Fettschrift, gefolgt von einem Komma und einem Leerzeichen, gefolgt vom Vornamen in normaler Schrift.
  Wie dieser Text für den Inhalt der Liste genau gebildet wird, ist weiter unten beschrieben.
- Kommt in einer Zeile der Vorlagendatei das Zeichen **`$`** vor, so soll dieses Zeichen ersetzt werden durch folgenden Text:
  Den Text für die „Langform-Liste". Jeder „lange Personeneintrag" besteht aus dem Vornamen gefolgt vom Nachnamen, beides in Fettschrift, gefolgt von einem Komma und einem Leerzeichen, gefolgt vom Geburtsdatum. Außer dem Vor- und dem Nachnamen sei der ganze Text in normaler Schrift.
  Wie der Text für diese Liste genau gebildet wird, ist weiter unten beschrieben.



Die Listentexte (siehe blaue Sektionen in `webseite.html` im Diagramm) ergeben sich wie folgt:

- Jede Listenzeile entspricht einer Datenzeile der Datendatei.
- Jede Listenzeile wird abgeschlossen mit dem Text `<br/>`.
- Der Text der Listenzeile ergibt sich aus den Zeilen der Datendatei, die wie in der Aufgabenstellung oben beschrieben umgewandelt werden.
- Ein Text wird in Fettdruck ausgegeben, wenn er innerhalb die Zeichen `<b>...</b>` steht.

Testlauf:

Keine sichtbaren Eingaben und Ausgaben des Programms. Reines Lesen und Schreiben der Dateien.

Arbeitsschritte (Sie müssen diese Schritte nicht unbedingt befolgen, dies ist nur ein "Angebot"; ihr Programm muss auch nicht unbedingt den hier gemachten Vorschlägen entsprechen):

1. Schreiben Sie ein C++ Programm, welches die Datendatei `personendaten.txt` zeilenweise einliest und jede eingelesene Zeile wieder auf den Bildschirm ausgibt. *Pseudo-Code:*

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
    string eingabezeile;

    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;

    solange(eingabezeile aus Datendatei lesen) {
        eingabezeile ausgeben;
    }

    Datendatei schließen;

    return 0;
}
```

2a. Erweitern Sie dieses C++ Programm:

- Definieren Sie in einer separaten Headerdatei `person.h` einen strukturierten Datentyp für Personen.

- Definieren Sie in einer separaten Implementierungsdatei `person.cpp` eine Funktion …

  `Person extrahiere_person(string eingabezeile)`

  … welche aus einer Eingabezeile die Personendaten extrahiert:
  Die Eingabezeile soll am ersten Komma aufgespalten werden und der Teil bis zum Komma als Nachname verwendet werden.

Der zweite Teil der Aufspaltung soll wieder am ersten Komma aufgespalten werden. Der Teil bis zum Komma soll als Vorname benutzt werden, der Teil dahinter als Geburtsdatum.

Benutzen Sie zum Aufspalten die Funktion `spalte_ab_erstem()` aus den Offline-Aufgaben. Platzieren Sie diese in separaten Dateien `texte.h` und `texte.cpp`.

Die Funktion `extrahiere_person()` soll dann den so erstellten `Person`-Wert zurückgeben.

Lassen Sie von Ihrem C++Hauptprogramm die Daten der erhaltenen Person ausgeben. *Pseudo-Code:*

```
// In person.h: ////////////////////////
struct Person { ... };

// In person.cpp: /////////////////////////
Person extrahiere_person(string eingabezeile)
{
    Person p;
    string rest;
    spalte_ab_erstem(eingabezeile, ',', p.nachname, rest);
    spalte_ab_erstem(rest, ',', p.vorname, p.geburtsdatum);
    return p;
}

// Datei main.cpp: ////////////////////////
#include <iostream>
#include <string>
#include <fstream>
// ... ggfs. weitere include ...
using namespace std;

int main()
{
    string eingabezeile = "";

    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;

    solange(eingabezeile aus Datendatei lesen) {
        Person person = extrahiere_person(eingabezeile);
        // Dann person.vorname, person.nachname,
        //      person.geburtsdatum ausgeben
    }

    Datendatei schließen;

    return 0;
}
```

2b. Fällt Ihnen auf, dass die einzelnen Personendaten noch unnötige Leerzeichen am Anfang und Ende der Einzelwerte der Person enthalten können?

Falls ja: Erweitern Sie die Funktion `extrahiere_person()`, so dass bei der Rückgabe des Werts `p` die Komponentenwerte `vorname`, `nachname`, `geburtsdatum` "gesäubert" werden. Benutzen Sie dazu die `string trimme(string s)` Funktion aus den Offline-Aufgaben (Code der Funktion so ändern, dass statt Pluszeichen jetzt Leerzeichen entfernt werden), die Sie ebenfalls in `texte.h` bzw. `texte.cpp` platzieren:

```
Person extrahiere_person(string eingabezeile)
{
    Person p; string rest = "";
    spalte_ab_erstem(eingabezeile, ',', p.nachname, rest);
    spalte_ab_erstem(rest, ',', p.vorname, p.geburtsdatum);
    p.nachname = trimme(p.nachname);
    p.vorname = trimme(p.vorname);
    p.geburtsdatum = trimme(p.geburtsdatum);
    return p;
}
```

3. Erweitern Sie nun Ihr Hauptprogramm, um den Kurztext gemäß den Vorgaben erstellen zu lassen. Die bisherige Ausgabe der Personenwerte kann wieder weggelassen werden.

```
int main()
{
    string eingabezeile = "", kurztext = "", langtext = "";

    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;

    solange(eingabezeile aus Datendatei lesen) {
        Person person = extrahiere_person(eingabezeile);
        // kurztext ausgeben lassen ...
        kurztext += br(
                        b(person.nachname) + ", " + person.vorname
                    ) + "\n";
    }

    Datendatei schließen;

    return 0;
}
```

Es werden dabei zwei Hilfsfunktionen benötigt:
`string br(string s)` hängt an den String `s` den Text `<br/>` an.
`string b(string s)` hängt vor den String `s` den Text `<b>` und dahinter den Text `</b>`. Diese beiden Funktionen können Sie in der `main.cpp` platzieren.

4. Erweitern Sie nun Ihr Hauptprogramm, um den Langtext gemäß den Vorgaben erstellen zu lassen.

```
int main()
{
    string eingabezeile = "", kurztext = "", langtext = "";

    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;

    solange(eingabezeile aus Datendatei lesen) {
        Person person = extrahiere_person(eingabezeile);
        // kurztext erstellen ...
        kurztext += br(
                        b(person.nachname) + ", " + person.vorname
                    ) + "\n";
        // langtext erstellen ...
        langtext += br(
                        b(person.vorname + " " + person.nachname) +
                        ", " +
                        person.geburtsdatum
                    ) + "\n";
    }

    Datendatei schließen;

    return 0;
}
```

5. Erweitern Sie Ihr Hauptprogramm um das Lesen der Vorlagendatei und das Schreiben der Ausgabedatei. Benutzen Sie dabei die `ersetze()` Funktion aus den Offline-Aufgaben.

```
int main()
{
    string eingabezeile = "", kurztext = "", langtext = "";

    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;

    solange(eingabezeile aus Datendatei lesen) {
        Person person = extrahiere_person(eingabezeile);
        // kurztext erstellen ...
        kurztext += br(
                        b(person.nachname) + ", " + person.vorname
                    ) + "\n";
        // langtext erstellen ...
        langtext += br(
                        b(person.vorname + " " + person.nachname) +
                        ", " +
                        person.geburtsdatum
                    ) + "\n";
    }

    Datendatei schließen;

    Templatedatei "webseite.html.tmpl" als textuelle Eingabedatei öffnen;

    Ausgabedatei "webseite.html" als textuelle Datei zum Schreiben öffnen;

    solange(eingabezeile aus Templatedatei lesen) {
        eingabezeile = ersetze(eingabezeile, '%', kurztext);
        eingabezeile = ersetze(eingabezeile, '$', langtext);

        Schreibe eingabezeile + "\n" in die Ausgabedatei;
    }

    Templatedatei schließen;

    Ausgabedatei schließen;

    return 0;
}
```

(Pflicht-) Aufgabe INF-07.02: Zeichenkette suchen rekursiv (Rekursion, Stringoperationen)

Programmieren Sie in einer Datei `suchen_rekursiv.cpp` (mit zugehöriger Headerdatei) eine C++ Funktion

```
int zeichenkette_suchen_rekursiv(string text,
                                 string zkette,
                                 unsigned int text_pos = 0,
                                 unsigned int text_such_pos = 0,
                                 unsigned int zkette_such_pos = 0
);
```

welche durch Rekursion ermittelt, ob die Zeichenkette `zkette` in dem einzeiligen Text `text` (ggf. mit Leerzeichen und/oder Satzzeichen) vorkommt.
Die Funktion **darf keinerlei Schleifen benutzen**.

`text_pos` bezeichne die Position innerhalb von `text`, ab der gerade versucht wird, die Zeichenkette `zkette` zu finden.
Wenn an einer `text_pos` das linkeste Zeichen der gesuchten Zeichenkette gefunden wird, so werden ab dieser Stelle mittels der Positionszähler `text_such_pos` (aktuelle Vergleichsposition im einzeiligen Text) und `zkette_such_pos` (aktuelle Vergleichsposition in der zu suchenden Zeichenkette) auch alle weiteren Zeichen des Textes und der Such-Zeichenkette rekursiv verglichen.

Sollte die Zeichenkette nicht in dem Text vorkommen, so soll der Wert `-1` zurückgegeben werden.
Anderenfalls wir die Startposition zurückgegeben, ab der das erste (linkeste) Vorkommen von `zkette` in `text` beginnt.
Die Zählung der Positionen im Text beginne bei `0`.

Schreiben Sie ferner ein C++ Hauptprogramm, welches die Eingaben entgegennimmt, die Funktion aufruft und das Ergebnis ausgibt (siehe Testlauf).

Es ist ihnen *nicht* erlaubt, Systemfunktionen zur Suche von Zeichenketten o.ä. zu verwenden. *Schleifen auch nicht erlaubt, siehe oben.*

Legen Sie im Projekt eine leere Headerdatei `catch.h` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Headerdatei.

Legen Sie im Projekt eine Datei `test_suchen_rekursiv.cpp` an mit folgendem Inhalt:

```
// Datei: test_suchen_rekursiv.cpp

#include "catch.h"
#include "suchen_rekursiv.h"

TEST_CASE("Zeichenkette suchen, Text mit Laenge groesser 1, Zeichenkette mit Laenge
groesser 1") {
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "cda") == 2);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "de") == 7);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "dex") == -1);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "xyz") == -1);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "abcdabcd") == 0);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "abcdabcdx") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge groesser 1, Zeichenkette mit Laenge
1") {
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "a") == 0);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "c") == 2);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "e") == 8);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdabcde", "x") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge 1") {
    REQUIRE(zeichenkette_suchen_rekursiv("a", "a") == 0);
    REQUIRE(zeichenkette_suchen_rekursiv("a", "c") == -1);
    REQUIRE(zeichenkette_suchen_rekursiv("a", "xy") == -1);
    REQUIRE(zeichenkette_suchen_rekursiv("a", "aa") == -1);
}

TEST_CASE("Zeichenkette suchen, leerer Text") {
    REQUIRE(zeichenkette_suchen_rekursiv("", "") == -1);
    REQUIRE(zeichenkette_suchen_rekursiv("", "a") == -1);
    REQUIRE(zeichenkette_suchen_rekursiv("", "abc") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge 20 Zeichen") {
    REQUIRE(zeichenkette_suchen_rekursiv("abcdefghij1234567890", "90") == 18);
    REQUIRE(zeichenkette_suchen_rekursiv("12345678901234567890", "90") == 8);
    REQUIRE(zeichenkette_suchen_rekursiv("abcdefghij1234567890", "9012") == -1);
}

// Vorgegebene Testläufe müssen selbst als Testcases programmiert werden ...
TEST_CASE("Vorgegebene Testlaeufe") {

}
```

Ergänzen Sie die Unit Tests in dieser Datei um die Testfälle, die in den unten angegebenen Testläufen vorgegeben sind.

Schreiben Sie ferner ein C++ Hauptprogramm, welches die Eingaben entgegennimmt, die Funktion aufruft und das Ergebnis ausgibt (siehe Testläufe). Ergänzen Sie dabei den folgenden Programmrahmen:

```
// Datei: suchen_rekursiv_main.cpp

#define CATCH_CONFIG_RUNNER
#include "catch.h"

#include <iostream>
using namespace std;

#include "suchen.h"

int main()
{
    Catch::Session().run();

    // Ihr Code ab hier ...


    system("PAUSE");
    return 0;
}
```

Testläufe: *(Benutzereingaben sind zur Verdeutlichung unterstrichen)*

```
Bitte geben Sie den Text ein: abcdefg
Bitte geben Sie die zu suchende Zeichenkette ein: bcd99
Die Zeichenkette 'bcd99' ist NICHT in dem Text 'abcdefg' enthalten.
Drücken Sie eine beliebige Taste . . .
```

```
Bitte geben Sie den Text ein: xy abc abcdefgh
Bitte geben Sie die zu suchende Zeichenkette ein: abcde
Die Zeichenkette 'abcde' ist in dem Text 'xy abc abcdefgh'
enthalten.
Sie startet ab Zeichen 7 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
Bitte geben Sie den Text ein: xyz 123 456 abc
Bitte geben Sie die zu suchende Zeichenkette ein: 123 4
Die Zeichenkette '123 4' ist in dem Text 'xyz 123 456 abc'
enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
Bitte geben Sie den Text ein: abc defg
Bitte geben Sie die zu suchende Zeichenkette ein: abc d
Die Zeichenkette 'abc d' ist in dem Text 'abc defg' enthalten.
Sie startet ab Zeichen 0 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
Bitte geben Sie den Text ein: abcdefg
Bitte geben Sie die zu suchende Zeichenkette ein: efg
Die Zeichenkette 'efg' ist in dem Text 'abcdefg' enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

Zum Verständnis:
Interner Beispielablauf der Rekursion: Text `abcabcdef`*, Zeichenkette* `abcd`

```
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 0, 0, 0)
Vergleiche a und a: gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 0, 1, 1)
Vergleiche b und b: gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 0, 2, 2)
Vergleiche c und c: gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 0, 3, 3)
Vergleiche a und d: nicht gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 1, 1, 0)
Vergleiche b und a: nicht gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 2, 2, 0)
Vergleiche c und a: nicht gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 3, 3, 0)
Vergleiche a und a: gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 3, 4, 1)
Vergleiche b und b: gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 3, 5, 2)
Vergleiche c und c: gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 3, 6, 3)
Vergleiche d und d: gleich
zeichenkette_suchen_rekursiv("abcabcdef", "abcd", 3, 7, 4)
Die Zeichenkette 'abcd' ist in dem Text 'abcabcdef' enthalten.
Sie startet ab Zeichen 3 (bei Zaehlung ab 0).
```

Für das Aufgaben-Tutorium am Freitag 19.11.2021, als freiwillige Aufgabe:

Aufgabe INF-07.03 (war auch schon INF-06.03)**: "Animation"**

Für GIP-WI und GIP-MCD ist diese Thematik noch neu …