Praktikumstermin Nr. 05, INF: Sudoku prüfen, Zeichenkette suchen rekursiv inkl. Unit Tests

Abgabe im GIP-INF Praktikum in der Woche 13.11.-17.11.2023

(Pflicht-) Aufgabe INF-05.01: Sudoku auf Gültigkeit prüfen (Schleifen, `if`, Strings, Funktionen)

Nutzen Sie Ihre Lösung der vorherigen Sudoku Praktikumsaufgabe (aus dem vorigen GIP-INF Praktikum) für das Einlesen des Sudokus.

Legen Sie im Projekt eine leere Datei `sudoku_pruefen_main.cpp` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Datei. Ergänzen Sie den Inhalt dieser Datei zu einem C++ Programm, welches ein Sudoku einliest *(nutzen Sie dafür gerne Ihre Lösung der Sudoku Praktikumsaufgabe aus dem vorigen GIP-INF Praktikum)* und in einem Array …

```
int sudoku[9][9] = {0};
```

… abspeichert. Anschließend soll das Programm prüfen, ob das Sudoku den Sudoku Regeln genügt: *„… jede Ziffer in jeder Spalte, in jeder Zeile und in jedem Block (3×3-Unterquadrat) genau einmal vorkommt*".

Das Ergebnis der Prüfung soll ausgegeben werden mitsamt Hinweisen auf mehrfach oder gar nicht vorkommenden Zahlen, siehe Testläufe.
Die Zeilen, Spalten und Blöcke sind dabei von 0 an durchnummeriert.
Innerhalb der Prüfung einer Zeile/Spalte/Blocks werden „Regelverletzungen" in aufsteigender Zeilen-/Spalten-/Block- und aufsteigender Zahlen-Reihenfolge ausgegeben.

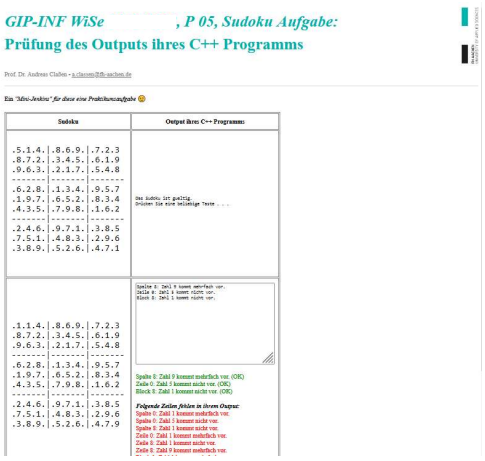
Die Prüfungen der Sudoku-Korrektheit und die Umwandlung der Eingabezeilen in die Sudoku-Werte (gemäß der Aufgabe aus dem letzten Praktikum) sind dabei zwecks Strukturierung des Gesamtprogramms in Funktionen ausgelagert. Da die Größe des Sudokus und die Anzahl der Eingabezeilen als globale Konstanten verfügbar sind, brauchen diese Werte auch nicht als weitere Funktionsparameter übergeben zu werden, wie das sonst üblich ist, wenn Arrays als Funktionsparameter übergeben werden.

Hinweis (muss aber nicht unbedingt genutzt werden):
Für Sudoku-Blöcke `0` … `8` liegen …
… die Zeilen des Blocks im Bereich
von `zeile = block/3*3` bis `zeile <= block/3*3+2`
… die Spalten des Blocks im Bereich
von `spalte = block%3*3` bis `spalte <= block%3*3+2`

Die Sudokus der beiden Testläufe finden Sie auch in einer Datei in Ilias.

Sollten bei nicht gültigen Sudokus die „Meldungen" ihres Programms in einer anderen Reihenfolge sein als in den Testläufen oder einzelne „Meldungen" über mehrfach vorhandene bzw. nicht vorhandene Zahlen bei ihnen mehrfach vorkommen, so ist das auch o.k.
Letztendlich müssen die einzelnen Meldungen (mal abgesehen von Wiederholungen und Reihenfolge) aber sein wie in den Testläufen …

In Ilias finden Sie eine HTML Datei `sudoku_check.html`, mittels der Sie den Outputs ihrer Praktikumslösung für drei gegebene Sudokus auf einfache Art prüfen können. Laden Sie die HTML Datei in ihren Webbrowser und copy-pasten Sie dann den Output ihres C++ Programms in den Textbereich rechts des jeweiligen Sudokus. Die Webseite sollte Ihnen dann Rückmeldung geben, in wieweit die Ausgaben ihres Programms von der gewünschten Lösung abweichen.


GIP-INF WiSe , P 05, Sudoku Aufgabe:
Prüfung des Outputs ihres C++ Programms

Eine unterschiedliche Reihenfolge der Ausgabezeilen wird dabei von der Webseite „ignoriert", d.h. auch bei anderer Reihenfolge werden Sie eine positive Rückmeldung bekommen, wenn die Zeilen an sich stimmen und vollständig sind …

Testlauf: *(Benutzereingaben diesmal **nicht** unterstrichen, da sonst zu unübersichtlich)*

```
Bitte geben Sie das Sudoku ein:
.5.1.4.|.8.6.9.|.7.2.3
.8.7.2.|.3.4.5.|.6.1.9
.9.6.3.|.2.1.7.|.5.4.8
-------|-------|-------
.6.2.8.|.1.3.4.|.9.5.7
.1.9.7.|.6.5.2.|.8.3.4
.4.3.5.|.7.9.8.|.1.6.2
-------|-------|-------
.2.4.6.|.9.7.1.|.3.8.5
.7.5.1.|.4.8.3.|.2.9.6
.3.8.9.|.5.2.6.|.4.7.1
Das Sudoku ist gueltig.
Drücken Sie eine beliebige Taste . . .
```

Im folgenden Sudoku wiederholen sich bei den „Meldungen" bestimmte Muster. Dadurch ist es leichter zu kontrollieren, ob von ihrem Programm die richtigen Meldungen generiert werden. Ich habe durch Farbgebung versucht, die sich wiederholenden Muster hervorzuheben …

```
Bitte geben Sie das Sudoku ein:
.1.1.4.|.8.6.9.|.7.2.3
.8.7.2.|.3.4.5.|.6.1.9
.9.6.3.|.2.1.7.|.5.4.8
-------|-------|-------
.6.2.8.|.1.3.4.|.9.5.7
.1.9.7.|.6.5.2.|.8.3.4
.4.3.5.|.7.9.8.|.1.6.2
-------|-------|-------
.2.4.6.|.9.7.1.|.3.8.5
.7.5.1.|.4.8.3.|.2.9.6
.3.8.9.|.5.2.6.|.4.7.9
Spalte 0: Zahl 1 kommt mehrfach vor.
Spalte 0: Zahl 5 kommt nicht vor.
Spalte 8: Zahl 1 kommt nicht vor.
Spalte 8: Zahl 9 kommt mehrfach vor.
Zeile 0: Zahl 1 kommt mehrfach vor.
Zeile 0: Zahl 5 kommt nicht vor.
Zeile 8: Zahl 1 kommt nicht vor.
Zeile 8: Zahl 9 kommt mehrfach vor.
Block 0: Zahl 1 kommt mehrfach vor.
Block 0: Zahl 5 kommt nicht vor.
Block 8: Zahl 1 kommt nicht vor.
Block 8: Zahl 9 kommt mehrfach vor.
Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Aufgabe INF-05.02: Zeichenkette suchen rekursiv (Rekursion, String-operationen) inkl. Unit Tests

Legen Sie im Projekt eine leere Datei `suchen_rekursiv.cpp` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Datei. Programmieren Sie in dieser Datei plus einer zugehörigen Header-datei eine C++ Funktion

```
int zeichenkette_suchen_rekursiv(std::string text,
                                 std::string zkette,
                                 size_t text_pos = 0,
                                 size_t text_such_pos = 0,
                                 size_t 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`.

Die Zeichenkette `zkette` darf nicht der leere String sein.
Ihre Funktion kann aber davon ausgehen, dass dies immer der Fall ist und muss dies nicht prüfen.
Der Text `text` darf auch der leere String sein.
In einem leeren Text kann natürlich nichts gefunden werden.

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

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 `gip_mini_catch.h` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Headerdatei. Verfahren Sie ebenso mit den Dateien `aufruf_visualisieren.h` und `aufruf_visualisieren.cpp`.

Legen Sie im Projekt eine leere Datei `test_suchen_rekursiv.cpp` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Datei. Ergänzen Sie die Unit Tests in dieser Datei um die Testfälle, die in den unten angegebenen Testläufen vorgegeben sind.

Legen Sie im Projekt eine leere Datei `suchen_rekursiv_main.cpp` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Datei. Ergänzen Sie das C++ Hauptprogramm so, dass es Eingaben entgegennimmt, die Funktion aufruft und das Ergebnis ausgibt (siehe Testläufe).

Testläufe: *(Benutzereingaben sind zur Verdeutlichung unterstrichen)*

```
Alle Tests erfolgreich (26 REQUIREs in 7 Test Cases)
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 . . .
```

```
Alle Tests erfolgreich (26 REQUIREs in 7 Test Cases)
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 . . .
```

```
Alle Tests erfolgreich (26 REQUIREs in 7 Test Cases)
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 . . .
```

```
Alle Tests erfolgreich (26 REQUIREs in 7 Test Cases)
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 . . .
```

```
Alle Tests erfolgreich (26 REQUIREs in 7 Test Cases)
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`
Wenn Sie in der Datei `suchen_rekursiv.cpp` den Aufruf der Funktion `aufruf_visualisieren(…)` „aktivieren", dann sehen Sie auf dem Bildschirm erläuternde Programmausgaben, die Ihnen helfen sollen nachzuvollziehen, welche rekursiven Aufrufe in ihrem Programm stattfinden und welchen „Suchsituationen" diese entsprechen. Die blau eingefärbten Zeilen werden nicht von der `aufruf_visualisieren(…)` Funktion ausgegeben, sondern diese habe ich hier ergänzt, um Ihnen ein besseres Verständnis des rekursiven Algorithmus zu ermöglichen. Ihre Lösung muss aber nicht genauso vorgehen und „entscheiden", wie dies „mein" Algorithmus gemäß der blauen Ausgabezeilen tut.

```
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cda",
          text_pos: 0,
          text_such_pos: 0,
          zkette_such_pos: 0)
0123456789
abcdabcde
cda ((zkette ab 'text_pos' 0, d.h. ab dort im text wird aktuell verglichen))
^ ((hier wird gerade verglichen))
Suche an dieser Pos. erfolglos ('a' ungleich 'c'), schiebe zkette eine Pos. weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cda",
          text_pos: 1,
          text_such_pos: 1,
          zkette_such_pos: 0)
0123456789
abcdabcde
 cda ((zkette ab 'text_pos' 1, d.h. ab dort im text wird aktuell verglichen))
 ^ ((hier wird gerade verglichen))
Suche an dieser Pos. erfolglos ('b' ungleich 'c'), schiebe zkette eine Pos. weiter
========================================================================
```

```
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cda",
          text_pos: 2,
          text_such_pos: 2,
          zkette_such_pos: 0)
0123456789
abcdabcde
  cda ((zkette ab 'text_pos' 2, d.h. ab dort im text wird aktuell verglichen))
  ^ ((hier wird gerade verglichen))
Zeichen c gefunden, mache weiter
==========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cda",
          text_pos: 2,
          text_such_pos: 3,
          zkette_such_pos: 1)
0123456789
abcdabcde
  cda ((zkette ab 'text_pos' 2, d.h. ab dort im text wird aktuell verglichen))
   ^ ((hier wird gerade verglichen))
Zeichen d gefunden, mache weiter
==========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cda",
          text_pos: 2,
          text_such_pos: 4,
          zkette_such_pos: 2)
0123456789
abcdabcde
  cda ((zkette ab 'text_pos' 2, d.h. ab dort im text wird aktuell verglichen))
    ^ ((hier wird gerade verglichen))
Zeichen a gefunden und damit Zeichenkette komplett gefunden
Die Zeichenkette 'cda' ist in dem Text 'abcdabcdef' enthalten.
Sie startet ab Zeichen 2 (bei Zaehlung ab 0).

==========================================================================
==========================================================================
==========================================================================
```

Weiterer Suchfall:

```
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 0,
          text_such_pos: 0,
          zkette_such_pos: 0)
0123456789
abcdabcdef
cde ((zkette ab 'text_pos' 0, d.h. ab dort im text wird aktuell verglichen))
^ ((hier wird gerade verglichen))
Suche an dieser Pos. erfolglos ('a' ungleich 'c'), schiebe zkette eine Pos. weiter
==========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 1,
          text_such_pos: 1,
          zkette_such_pos: 0)
0123456789
```

```
abcdabcdef
 cde ((zkette ab 'text_pos' 1, d.h. ab dort im text wird aktuell verglichen))
 ^ ((hier wird gerade verglichen))
Suche an dieser Pos. erfolglos ('b' ungleich 'c'), schiebe zkette eine Pos. weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 2,
          text_such_pos: 2,
          zkette_such_pos: 0)
0123456789
abcdabcdef
  cde ((zkette ab 'text_pos' 2, d.h. ab dort im text wird aktuell verglichen))
  ^ ((hier wird gerade verglichen))
Zeichen c gefunden, mache weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 2,
          text_such_pos: 3,
          zkette_such_pos: 1)
0123456789
abcdabcdef
  cde ((zkette ab 'text_pos' 2, d.h. ab dort im text wird aktuell verglichen))
   ^ ((hier wird gerade verglichen))
Zeichen d gefunden, mache weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 2,
          text_such_pos: 4,
          zkette_such_pos: 2)
0123456789
abcdabcdef
  cde ((zkette ab 'text_pos' 2, d.h. ab dort im text wird aktuell verglichen))
    ^ ((hier wird gerade verglichen))
Suche an dieser Pos. erfolglos ('a' ungleich 'e'), schiebe zkette eine Pos. weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 3,
          text_such_pos: 3,
          zkette_such_pos: 0)
0123456789
abcdabcdef
   cde ((zkette ab 'text_pos' 3, d.h. ab dort im text wird aktuell verglichen))
   ^ ((hier wird gerade verglichen))
Suche an dieser Pos. erfolglos ('d' ungleich 'c'), schiebe zkette eine Pos. weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 4,
          text_such_pos: 4,
          zkette_such_pos: 0)
0123456789
abcdabcdef
    cde ((zkette ab 'text_pos' 4, d.h. ab dort im text wird aktuell verglichen))
    ^ ((hier wird gerade verglichen))
```

```
Suche an dieser Pos. erfolglos ('a' ungleich 'c'), schiebe zkette eine Pos. weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 5,
          text_such_pos: 5,
          zkette_such_pos: 0)
0123456789
abcdabcdef
     cde ((zkette ab 'text_pos' 5, d.h. ab dort im text wird aktuell verglichen))
     ^ ((hier wird gerade verglichen))
Suche an dieser Pos. erfolglos ('b' ungleich 'c'), schiebe zkette eine Pos. weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 6,
          text_such_pos: 6,
          zkette_such_pos: 0)
0123456789
abcdabcdef
      cde ((zkette ab 'text_pos' 6, d.h. ab dort im text wird aktuell verglichen))
      ^ ((hier wird gerade verglichen))
Zeichen c gefunden, mache weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 6,
          text_such_pos: 7,
          zkette_such_pos: 1)
0123456789
abcdabcdef
      cde ((zkette ab 'text_pos' 6, d.h. ab dort im text wird aktuell verglichen))
       ^ ((hier wird gerade verglichen))
Zeichen d gefunden, mache weiter
========================================================================
zeichenkette_suchen_rekursiv(
          text: "abcdabcdef",
          zkette: "cde",
          text_pos: 6,
          text_such_pos: 8,
          zkette_such_pos: 2)
0123456789
abcdabcdef
      cde ((zkette ab 'text_pos' 6, d.h. ab dort im text wird aktuell verglichen))
        ^ ((hier wird gerade verglichen))
Zeichen e gefunden und damit Zeichenkette komplett gefunden
Die Zeichenkette 'cde' ist in dem Text 'abcdabcdf' enthalten.
Sie startet ab Zeichen 6 (bei Zaehlung ab 0).
```