FH AACHEN UNIVERSITY OF APPLIED SCIENCES

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

Abgabe im GIP-INF Praktikum in der Woche 7.11.-11.11.2022.

2022-11-04: *Der Fix führt (auf Windows Rechnern) zu einem neuen Problem, (nur) falls Sie Leerzeichen in Verzeichnisnamen haben sollten (was man besser nicht haben sollte, viele Software kriegt dann Probleme…). Sprich Pfade wie …* `C:\Users\Andreas Classen\Documents\Meine GIP Loesungen\...` *funktionieren nach dem Fix nicht, da die* ***`--%`*** *Option (wie am Ende dieses Dokuments erläutert) die Mechanismen „abschaltet", mit denen die problemlose Behandlung von Pfadnamen mit Leerzeichen möglich wird.*
Daher, ***falls Sie von diesem neuen Problem betroffen sein sollten****: Die* ***beiden Optionen `"--%", "-Wa,-mbig-obj",`*** *am besten* ***an vorletzter Position der `args` Liste einfügen*** *und nicht ganz vorne. Am besten *vor* der* `-pthread` *Option (wichtig: der letzte Eintrag der Liste darf nicht mit einem Komma enden, alle anderen Einträge davor müssen mit einem Komma enden). Wir werden eine Version 1.3.6 von VSCode-GIP veröffentlichen, die für neu angelegte Projekte das dann richtig macht bzw. eine andere Option verwendet, die dieses Problem nicht mehr auslöst).*

2022-11-03: *Entschuldigung für die unerwarteten* ***Probleme auf Windows Rechnern*** *(Fehlermeldung:* `file too big`*) beim Compilieren der "Zeichenkette finden" Lösung (Aufgabe INF-05-02) mit (Visual Studio Code und) dem g++ Compiler. Ich habe in meinem Berufsleben schon "tausende-mal" mit dem g++ compiliert (allerdings fast nur auf Unix-basierten Systemen) und hätte nicht gedacht, dass solche Probleme auftreten könnten. Genauer gesagt liegt das Problem im Assembler (*`as.exe`*), der den finalen Schritt in der Toolchain beim Compilieren des C++ Codes in Maschinencode darstellt.*

Lösung des Problems:

- *Entweder die neue Version 1.3.5 des GIP-VSCode installieren von https://git.fh-aachen.de/Erbach/gip-entwicklungsumgebung/-/releases (dort sind neben der Korrektur dieses Problems noch weitere Verbesserungen enthalten…)*
- *Oder Sie müssen in der* ***`tasks.json`*** *Datei ihres C++ Projekts in VS Code folgende zwei Optionen zu den* ***`args`*** *hinzufügen (beide mit Komma am Ende):* ***`"--%", "-Wa,-mbig-obj",`***
  Dann funktioniert das Compilieren wieder dauert aber immer noch jedes Mal etwas lange (Geduld gefragt). Wen es interessiert: Weitere Erläuterungen am Ende.

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

(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.

Schreiben Sie ein 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.

Benutzen Sie folgendes Programmgerüst, welches auch als Datei in Ilias verfügbar ist. Die Prüfungen und die Umwandlung der Eingabe 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.

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

const size_t sudoku_groesse = 9;   // 2022-11-03: geändert in size_t; int ist auch o.k.
const size_t eingabe_groesse = 11; // 2022-11-03: geändert in size_t; int ist auch o.k.

bool pruefe_spalten(int sudoku[][sudoku_groesse])
{
}

bool pruefe_zeilen(int sudoku[][sudoku_groesse])
{
}
```

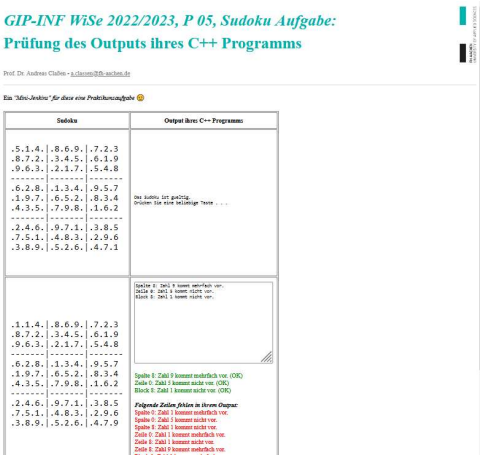
```
bool pruefe_bloecke(int sudoku[][sudoku_groesse])
{
}

void konvertiere(string eingabe[], int sudoku[][sudoku_groesse])
{
}

int main() {
    int sudoku[sudoku_groesse][sudoku_groesse] = { 0 };
    string eingabe[eingabe_groesse];

    cout << "Bitte geben Sie das Sudoku ein:" << endl;
    for (size_t i = 0; i < eingabe_groesse; i++)
        getline(cin, eingabe[i]);

    konvertiere(eingabe, sudoku);

    bool ps = pruefe_spalten(sudoku);
    bool pz = pruefe_zeilen(sudoku);
    bool pb = pruefe_bloecke(sudoku);

    if (ps && pz && pb)
        cout << "Das Sudoku ist gueltig." << endl;

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

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.



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. 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

Programmieren Sie in einer Datei `suchen_rekursiv.cpp` plus zugehöriger Headerdatei eine C++ Funktion

```
int zeichenkette_suchen_rekursiv(string text,
                                 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 `catch.h` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Headerdatei.

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.

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 (auch als Datei in Ilias vorgegeben):

```
// 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).
```

2022-11-03: *Weitere Erläuterungen zu der Korrektur: Im GIP-VSCode wird der g++ Compiler per Kommando in einer Powershell Kommandozeile gestartet. Dies führt aber leider dazu, dass die Powershell sich auch die Optionen des Compiler-Kommandos anschaut und versucht, ihre eigenen "Schreibweisen und Komfortmechanismen" auf das Kommando anzuwenden. Dies würde bei der zweiten Option* ***`-Wa,-mbig-obj`*** *zu Problemen führen, da diese Option ein Komma enthält und die Powershell dann meint, dies verbieten zu müssen. Die Option* ***`--%`*** *weist die Powershell an, den Rest des Kommandos (und damit alle weiteren Optionen) so zu belassen, wie sie sind und nicht eigene Regeln und Mechanismen darauf anzuwenden.*
Die zweite Option ***`-Wa,-mbig-obj`*** *setzt sich aus zwei Anteilen zusammen:* ***`-Wa`*** *besagt, dass die (ohne Leerzeichen folgenden, durch Komma voneinander getrennten) Optionen an den Assembler weitergegeben werden sollen. D.h. die Option* ***`-mbig-obj`*** *geht an den Assembler und löst dort das* `file too big` *Problem. Leider müssen wie gesagt die Optionen an den Assembler ohne Leerzeichen und mit Kommas getrennt geschrieben werden. Wenn die Powershell sich das anschaut, meint Sie, dies seien zwei separate Optionen, die durch Leerzeichen voneinander getrennt werden müssten und macht die Option dann "kaputt". Daher wie gesagt die Notwendigkeit der ersten Option.*