Praktikumstermin Nr. 04, INF: Caesar-Verschlüsselung, Sudoku einlesen

Abgabe in den Praktikumsterminen Di 25.10. - Do 3.11.2022:
Dienstags-GIP-INF-Praktikumsgruppen:
Abgabe Di 25.10.2022, Praktikumstermin Di 1.11. entfällt
Mittwochs-GIP-INF-Praktikumsgruppen:
Abgabe Mi 26.10.2022, Praktikumstermin Mi 2.11. entfällt
Donnerstags-GIP-INF-Praktikumsgruppen:
Praktikumstermin Do 27.10. entfällt, Abgabe Do 3.11.2022

(Pflicht-) Aufgabe INF-04.01: Caesar-Verschlüsselung / Verschlüsselung durch „Verschieben" (Schleife, C++ Strings)

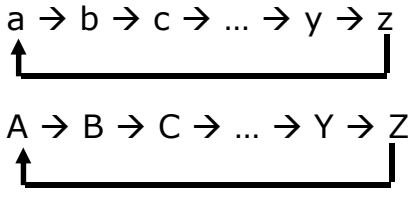
Schreiben Sie eine C++ Funktion …

```
string caesar(string original_text, unsigned int v)
```

… welche eine einzeilige Zeichenkette `text` als String einliest und diese Zeichenkette dann „durch Rechts-Verschiebung um `v` Positionen" verschlüsselt wieder ausgibt.

Ihre Funktion soll eingefügt werden in eine `.cpp` Datei mit Hauptprogramm, welche in Ilias zum Download vorgegeben ist. Dieses Hauptprogramm führt auch einige Tests mit ihrer Funktion aus, so dass Sie Hinweise bekommen, ob ihre Funktion korrekt zu funktionieren scheint oder nicht.

Die Verschlüsselung durch Verschieben („Caesar-Verschlüsselung" oder auch „Caesar-Chiffre") funktioniert dabei wie folgt:
Die Verschiebepositionen-Anzahl **v** legt fest, dass jeder Kleinbuchstabe und jeder Großbuchstabe aus dem Originaltext ersetzt wird durch seinen **v**-ten Nachfolger im Alphabet. Die Verschiebung wird dabei zyklisch fortgesetzt, d.h. es erfolgt ein sogenannter *Wrap-Around* bei den Kleinbuchstaben sowie bei den Großbuchstaben (man fängt vorne wieder an, wenn man am Ende angekommen ist).



Lassen Sie sich durch den Begriff „verschieben" nicht irritieren: Bei der Caesar-Verschlüsselung bleibt jeder Buchstaben an seinem Platz. Aber der Buchstabe wird ersetzt durch einen anderen Buchstaben, der sich durch die Verschiebung des Alphabets um die angegebene Anzahl an Positionen ergibt.

Beispiel: Bei einer Verschiebepositionen-Anzahl **v** von **2** würde der Text **apz** als **crb** verschlüsselt, weil sich das **c** aus dem **a** durch Verschieben um **2** ergibt, das **r** aus dem **p** durch Verschieben um **2** und das **b** aus dem **z** ergibt durch Verschieben um **2** (beim "Verschlüsseln" des Buchstabens **z** inklusive *Wrap-Around* beim ersten Verschiebe-Schritt, d.h. aus dem **z** wird beim ersten Verschiebe-Schritt ein **a** und aus diesem beim zweiten Verschiebe-Schritt ein **b**).

Alle Zeichen anders als Klein- und Großbuchstaben (z.B. Satzzeichen, Leerzeichen) sollen unverändert in den verschlüsselten Text übernommen werden.

Ihr Programm soll erst die Zeichenkette und dann die Anzahl der Verschiebepositionen **v** als (positive) ganze Zahl einlesen. Die **0** (Null, d.h. keine Verschiebung) ist als Eingabe erlaubt. Es sind auch Verschiebungen um mehr als **26** Positionen erlaubt, d.h. größer als der Umfang des Alphabets, siehe Testläufe. Der zu verschlüsselnde Originaltext darf auch leer sein.

Ihr Programm kann davon ausgehen, dass der Benutzer nur korrekte Eingaben macht.

Testläufe: *(Benutzereingaben sind zur Verdeutlichung unterstrichen, Blaufärbung auch nur zur Verdeutlichung, ihr Programm braucht den Text nicht einzufärben!)*

```
Verschluesselungs-Test erfolgreich (test case passed)! abc  ==(2)==>  cde
Verschluesselungs-Test erfolgreich (test case passed)! ABC  ==(2)==>  CDE
Verschluesselungs-Test erfolgreich (test case passed)! abc.,!?  ==(4)==>  efg.,!?
Verschluesselungs-Test erfolgreich (test case passed)! apz  ==(2)==>  crb
Verschluesselungs-Test erfolgreich (test case passed)! xYz. ABC  ==(5)==>  cDe. FGH
Verschluesselungs-Test erfolgreich (test case passed)! ., !?  ==(4)==>  ., !?
Verschluesselungs-Test erfolgreich (test case passed)! abcXYZ!  ==(0)==>  abcXYZ!
Verschluesselungs-Test erfolgreich (test case passed)!  ==(7)==>
Verschluesselungs-Test erfolgreich (test case passed)! aBxYz!  ==(26)==>  aBxYz!
Verschluesselungs-Test erfolgreich (test case passed)! aB!  ==(28)==>  cD!
Verschluesselungs-Test erfolgreich (test case passed)! aBxYz!  ==(52)==>  aBxYz!
Verschluesselungs-Test erfolgreich (test case passed)! aBxYz!  ==(78)==>  aBxYz!
Verschluesselungs-Test erfolgreich (test case passed)! aBxYz!  ==(99)==>  vWsTu!
Verschluesselungs-Test erfolgreich (test case passed)! abc def xyz!ABC XYZ?  ==(3)==>  def ghi abc!DEF ABC?
Verschluesselungs-Test erfolgreich (test case passed)! Abc Zyx!  ==(55)==>  Def Cba!
Bitte geben Sie den zu verschluesselnden Text ein: ? abc def xyz!ABC XYZ?
Bitte geben Sie die Anzahl Verschiebepositionen ein (als positive ganze Zahl): ? 3
Verschluesselt: def ghi abc!DEF ABC?
Drücken Sie eine beliebige Taste . . .
```

```
Verschluesselungs-Test erfolgreich (test case passed)! abc  ==(2)==>  cde
Verschluesselungs-Test erfolgreich (test case passed)! ABC  ==(2)==>  CDE
Verschluesselungs-Test erfolgreich (test case passed)! abc.,!?  ==(4)==>  efg.,!?
Verschluesselungs-Test erfolgreich (test case passed)! apz  ==(2)==>  crb
Verschluesselungs-Test erfolgreich (test case passed)! xYz. ABC  ==(5)==>  cDe. FGH
Verschluesselungs-Test erfolgreich (test case passed)! ., !?  ==(4)==>  ., !?
Verschluesselungs-Test erfolgreich (test case passed)! abcXYZ!  ==(0)==>  abcXYZ!
Verschluesselungs-Test erfolgreich (test case passed)!   ==(7)==>
Verschluesselungs-Test erfolgreich (test case passed)! aBxYz!  ==(26)==>  aBxYz!
Verschluesselungs-Test erfolgreich (test case passed)! aB!  ==(28)==>  cD!
Verschluesselungs-Test erfolgreich (test case passed)! aBxYz!  ==(52)==>  aBxYz!
Verschluesselungs-Test erfolgreich (test case passed)! aBxYz!  ==(78)==>  aBxYz!
Verschluesselungs-Test erfolgreich (test case passed)! aBxYz!  ==(99)==>  vWsTu!
Verschluesselungs-Test erfolgreich (test case passed)! abc def xyz!ABC XYZ?  ==(3)==>  def ghi abc!DEF ABC?
Verschluesselungs-Test erfolgreich (test case passed)! Abc Zyx!  ==(55)==>  Def Cba!
Bitte geben Sie den zu verschluesselnden Text ein: ? Abc Zyx!
Bitte geben Sie die Anzahl Verschiebepositionen ein (als positive ganze Zahl): ? 55
Verschluesselt: Def Cba!
Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Aufgabe INF-04.02: Sudoku einlesen (zweidimensionale Arrays)

Sudoku ist ein Logikrätsel. In der üblichen Version ist es das Ziel, ein 9×9-Gitter mit den Ziffern **1** bis **9** so zu füllen, dass jede Ziffer in jeder Spalte, in jeder Zeile und in jedem Block (3×3-Unterquadrat) genau einmal vorkommt.

Schreiben Sie ein C++ Programm, welches ein „gelöstes" Sudoku einliest und in einem Array …

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

… abspeichert. D.h. in dem Array sollen nur die 9x9 Zahlwerte abgespeichert werden, die „optischen Blocktrennzeichen" der Eingabe sollen in diesem Array nicht mehr vorkommen. Außerdem sollen die realen Zahlwerte abgespeichert werden, nicht die ASCII Codes der Zahlziffern (also eine „3" soll auch als Wert **3** abgespeichert werden, nicht als ASCII Wert **51**).

Lesen Sie dazu die 11 Eingabezeilen zuerst einmal in ein Array …

```
string eingabe[11];
```

… ein (es müssen zwei Einträge mehr sein als die neun Zeilen eines Sudoku wegen der beiden Zeilen mit den horizontalen Trennstrichen). Bitte beachten Sie, dass in der Eingabe keine Leerzeichen vorkommen, sondern anstelle dieser jeweils ein Punkt. Die Zahlen im Sudoku-Eingabetext sind immer einstellig.
Ihr Programm kann davon ausgehen, dass der Benutzer nur richtig formatierte Eingaben macht. *Die Gültigkeit der Zahlen (ob einstellig und gültig gemäß den Sudoku Regeln) muss daher nicht geprüft werden.*

Der Benutzer muss das Eingabe-Sudoku komplett selbst eingeben, inklusive der Punkte und Striche (senkrechte Striche und Minuszeichen).

Anschließend soll das Sudoku wieder ausgegeben werden, und zwar mit anderen „Trennzeichen" als bei der Eingabe, siehe Testläufe.
Die Erzeugung der Ausgabe darf nur unter Verwendung der Werte in `int sudoku[9][9]` geschehen, auf den Eingabetext `eingabe[]` darf dann nicht mehr zurückgegriffen werden.

Hinweis:
Kopieren Sie sich die Beispiel-Sudokus aus den Testläufen in einen separaten Editor oder in einen Blockkommentar in Ihrem C++ Quelltext. Dann können Sie von dort aus per *copy-paste* die Eingaben für Ihre Testläufe machen, ohne die Sudokus jedes Mal neu eintippen zu müssen.

Testläufe: *(Benutzereingaben diesmal **nicht** unterstrichen, sondern blau gefärbt, 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 lautet:
;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
Drücken Sie eine beliebige Taste . . .
```

```
Bitte geben Sie das Sudoku ein:
.9.4.6.|.3.1.8.|.2.7.5
.1.2.3.|.7.5.6.|.9.4.8
.5.8.7.|.2.4.9.|.6.3.1
-------|-------|-------
.8.1.4.|.9.2.5.|.7.6.3
.2.7.5.|.1.6.3.|.8.9.4
.6.3.9.|.8.7.4.|.1.5.2
-------|-------|-------
.3.6.8.|.5.9.2.|.4.1.7
.4.5.1.|.6.8.7.|.3.2.9
.7.9.2.|.4.3.1.|.5.8.6

Das Sudoku lautet:
;9;4;6;//;3;1;8;//;2;7;5
;1;2;3;//;7;5;6;//;9;4;8
;5;8;7;//;2;4;9;//;6;3;1
=======//=======//=======
;8;1;4;//;9;2;5;//;7;6;3
;2;7;5;//;1;6;3;//;8;9;4
;6;3;9;//;8;7;4;//;1;5;2
=======//=======//=======
;3;6;8;//;5;9;2;//;4;1;7
;4;5;1;//;6;8;7;//;3;2;9
;7;9;2;//;4;3;1;//;5;8;6
Drücken Sie eine beliebige Taste . . .
```