Übung 8

a) Programmieren Sie eine Funktion, welche ein Dreieck in ASCII-Art wie nebenstehend gezeigt erzeugt (Beispiel für 4 Zeilen, das linke untere „+“ soll direkt am linken Rand der Konsole anfangen).
Die Signatur sieht wie folgt aus: `void muster(int anz_zeilen);`
Ist das Argument kleiner als 1, soll natürlich nichts dargestellt werden.

b) Suchen Sie in einem Integer-Feld beliebiger Größe das kleinste sowie das größte darin enthaltene Element und geben Sie beides auf der Konsole aus. Achten Sie darauf, dass auch negative Werte im Feld enthalten sein können und testen Sie daher auch diesen Fall.

c) Schreiben Sie eine Funktion `bool firstLast8(int arr[], int n)`, die *true* zurückgibt, wenn die Zahl 8 entweder an erster oder letzter Stelle des Feldes steht, andernfalls wird *false* zurückgegeben. Das Argument `n` sei dabei die übergebene Länge des Feldes `arr`.

d) Programmieren Sie die Funktion `int countOdds(int f[], int n)`, die die Anzahl der ungeraden Zahlen im Feld `f` zurückgibt. Testen Sie Ihre Funktion mit Feldern verschiedener Länge und Belegung (gerne dürfen Sie auch Zufallszahlen dafür verwenden).

e) Eine 1, unmittelbar gefolgt von einer 3, gelte in einem Feld als eine "unglückliche" 1. Schreiben Sie eine Funktion `bool unlucky1(int arr[], int n)`, die *true* zurückgibt, wenn das gegebene Array mindestens eine unglückliche 1 enthält.
Testen Sie mit möglichst vielen Varianten in *main()*. Beispielsweise soll ein Array mit der Belegung {2, 1, 3, 4, 5} *true* liefern, während die Belegung {1, 1, 8} den Wert *false* liefert.

f) Implementieren Sie zwei Funktionen `int kleinstes(int arr[], int n)` sowie `int zweitkleinstes(int arr[], int n)`, die den Index des kleinsten bzw. zweitkleinsten Elements eines Feldes zurückliefern. Testen Sie alles gründlich in *main()*.

g) Schreiben Sie eine Funktion `void reverse(int arr[], int n)`, die den Inhalt des übergebenen Feldes so umdreht, dass das zuvor erste Element nun an letzter Stelle steht, das zuvor zweite Element sich an vorletzter Stelle befindet usw. Testen Sie sowohl mit Feldern gerader als auch ungerader Länge, indem Sie die geänderten Felder in *main()* ausgeben.

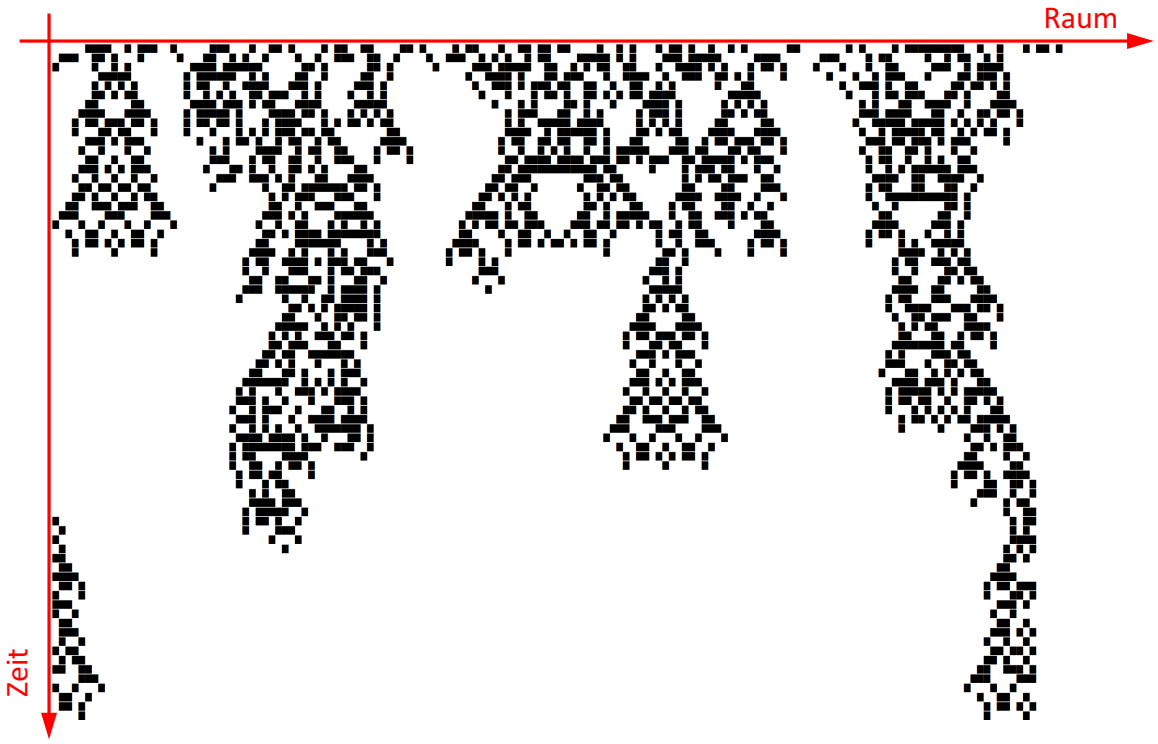
h) Implementieren Sie eine Funktion `void swap(int arr1[], int arr2[], int len)`, so dass der Aufruf `swap(a, b)` den Inhalt beider Arrays `a` und `b` vertauscht. Überlegen Sie sich zunächst, wie man grundsätzlich beim Vertauschen zweier „normaler“ Variablen vorgehen muss (Tipp: Sie benötigen eine Hilfsvariable zum Zwischenspeichern).

i) Programmieren Sie das Spiel *Tic Tac Toe* (s. Folien Nr. 7 – 10) für den einfachen Fall, dass zwei Spieler vor dem Rechner sitzen und sich abwechseln.[1]
Wem das so zu langweilig ist, darf natürlich gerne einen Single Player Mode implementieren.

[1] Für Ästheten: Unter Linux können Sie mit folgendem Befehl die Konsole „clearen“, so dass der Output nicht weiterwandert: `printf("\x1b[H\x1b[J");` Unter Windows wird es leider ziemlich dreckig.

j) Weihnachtsaufgabe *(freiwillige Zusatzaufgabe)*: **Zellulärer Automat**

Ein sog. zellulärer Automat ist ein „Modell-Universum“ bestehend aus einer Raum- und einer Zeitdimension (vgl. Abb.).



Implementieren Sie mit Hilfe von Feldern einen einfachen zellulären Automaten, bei dem das Verhalten von „Lebewesen“ in einer eindimensionalen Welt simuliert und dargestellt wird.
Die besagte Welt besteht dabei aus *n* aneinandergereihten Zellen bzw. Feldelementen (oder „Grundstücken“), die entweder leer (0) oder voll (1) sind.

1	**0**	**1**	**0**	**1**	**1**

Pro Simulations- bzw. Zeitschritt wird dargestellt, wie sich die Welt weiterentwickelt, d.h. auf welchen Grundstücken sich bereits ein Lebewesen angesiedelt hat oder wo inzwischen keines mehr wohnt. Um die einzelnen Zeitschritte darzustellen, wird auf der Konsole jeweils eine neue Zeile hinzugefügt.
Je nachdem, wie viele benachbarte Lebewesen (oder besser gesagt Vorgänger) existieren, ist die zeitlich nachfolgende Zelle voll oder leer (d.h. also, es wird ein neues Lebewesen erzeugt, bleibt am Leben oder stirbt).
Für den Simulationsverlauf ist die Anzahl der Nachbarn (bzw. Vorgänger) wichtig: Eine Zelle (im Bild z.B. *C*) kann zu einem Zeitpunkt *i+1* je 0 bis 5 volle Nachbarzellen haben (also Zellen A bis E zum Zeitpunkt *i*).

1	**1**	**1**	**0**	**1**	**0**	*Zeit i*
A	***B***	***C***	***D***	***E***	***F***	*Zeit i+1*
						…

Die „Welt“ zu einem bestimmten Zeitpunkt (im Rechner repräsentiert als eindimensionales Feld) wird zeilenweise so dargestellt, dass für die Ausgabe pro Zeitschritt genau eine Zeile (ohne Zeilenumbruch) verwendet wird.
Stellen Sie hierbei die beiden Zustände auf der Konsole folgendermaßen dar: Unterstrich `"_"` für 0 und Buchstabe `"X"` für 1.
Die Anzahl der dargestellten Simulationsschritte soll frei wählbar sein, wobei das Verfahren natürlich möglichst speicherschonend (mit nur zwei eindimensionalen Feldern) umgesetzt werden soll. Die Felder dürfen Sie initial aber mit einer festen Maximalgröße (z.B. 80) anlegen.

Der Anfangszustand ergibt sich aus einer zufällig generierten Startzeile. Mehr zu Zufallszahlen finden Sie im Anhang des aktuellen Foliensatzes zum Thema Felder und Strings.

Erzeugen Sie die jeweils nächste Zeile nach der folgenden Regel: Hat eine Zelle 2 oder 4 (volle) Vorgänger, so wird sie auf Zustand 1 gesetzt, ansonsten auf Zustand 0. Für diese Berechnung wird eine Zeile zudem als geschlossen (also „ringförmig“ verbunden) angenommen.
Damit ergibt sich z.B. für die oben gezeigte Situation, dass Zelle B 3 Vorgänger hat, A und C haben 4, und D, E sowie F haben wieder 3 Vorgänger.

In den Schleifen sollten Sie versuchen, möglichst ohne Fallunterscheidung auszukommen. Verwenden Sie stattdessen zur Berechnung der Anzahl eine geeignete Indizierung mit Modulorechnung und nutzen Sie ggfs. für die nachfolgende Zuweisung den ternären Operator.