Anmerkung: Alle Punkte dieses "Weihnachtsblatts" werden als Bonuspunkte gewertet. Bearbeiten Sie dieses Blatt auf jeden Fall ausgiebig, wenn Sie noch nicht genug Punkte für die Klausurzulassung haben!

ÜBUNG 10.1 ZELLULÄRE AUTOMATEN - CONWAYS GAME OF LIFE

Zelluläre Automaten sind ein formales Konzept mit dem man diskrete, räumliche Systeme beschreiben kann. Ein zellulärer Automat wird spezifiziert durch

- Eine Menge von Zellen (der *Zellraum*)
- Eine Menge von Zuständen, die die Zellen annehmen können (die *Zustandsmenge*)
- Eine *Übergangsfunktion*, die angibt wie sich aus dem Zustand der Zellen zum Zeitpunkt t deren Zustand zum Zeitpunkt $t+1$ ergibt. Dieser Übergang hängt dabei nur vom Zustand der Zellen in einer endlich großen *Nachbarschaft* der Zelle ab.

Wir wollen einen sehr bekannten zellulären Automaten, Conways Game of Life*, simulieren. Dabei gilt

- Die Zellen sitzen auf einem $n \times m$-Raster (der *Zellraum*)
- Zellen können entweder tot oder lebend sein (die *Zustandsmenge*)
- Die *Nachbarschaft* einer Zelle sind die acht Zellen, die sich direkt um die Zelle herum befinden.
- In jedem Schritt verändert sich der Zustand der Zellen nach folgenden Regeln (die *Übergangsfunktion*):
  - Eine tote Zelle mit exakt drei lebenden Zellen in ihrer Nachbarschaft wird wiederbelebt.
  - Eine lebende Zelle, mit weniger als zwei lebenden Zellen in ihrer Nachbarschaft stirbt (Vereinsamung).
  - Eine lebende Zelle, mit mehr als drei lebenden Zellen in ihrer Nachbarschaft stirbt (Überbevölkerung)
- Was genau am Rand des Rasters geschieht muss man definieren. Es gibt drei naheliegende Möglichkeiten:
  - Alle Nachbarfelder, die außerhalb des Rasters liegen, werden als tote Zellen betrachtet.
  - Alle Nachbarfelder, die außerhalb des Rasters liegen, werden als lebende Zellen betrachtet.
  - Das Raster wird als doppelt periodisch angenommen: Ist man an einer Zelle außerhalb des Rasters interessiert, so betrachtet man stattdessen die Zelle am gegenüberliegenden Rand†.

Schreiben Sie ein Programm, welches das Game of Life simuliert. Verwenden Sie dabei die auf der Vorlesungshomepage zur Verfügung stehende Vorlage. Diese verwendet das `TwoDBoolArray` von Blatt 9 als zugrundeliegende Datenstruktur. Sie können dabei gerne auch ihre eigene Implementierung verwenden. [*7 Punkte*]

Sie finden auf der Vorlesungshomepage ein Archiv mit Ausgangszuständen für das Game of Life. Experimentieren Sie mit diesen und schildern Sie ein paar bemerkenswerte Entdeckungen, die sie machen. Probieren Sie auch eigene Startzustände aus. Die Beschäftigung mit Conways Game of Life ist seit den 70er Jahren ein Sport unter Informatikern. Es ist inzwischen bewiesen, dass man durch ein Game of Life sogar eine Turingmaschine simulieren kann. Damit handelt es sich um eine turingvollständige Programmiersprache! [*3 Punkte*]

*`http://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens`

†Mathematisch gesehen ist der Zellraum dann ein Torus.

10 Punkte

ÜBUNG 10.2 WEIHNACHTSBÄUME

Schreiben Sie ein Programm, das einen Weihnachtsbaum auf dem Bildschirm ausgibt. Ihrer Kreativität sind dabei keine Grenzen gesetzt. So können Sie Ihren Baum zum Beispiel mit zufällig verteiltem Schmuck oder brennenden Kerzen verzieren, wie Sie es weiter unten auf diesem Blatt sehen.

Es werden alle Abgaben akzepiert, ob sie nun einfach aus kleinen Sternchen die Silhouette eines Baumes erzeugen oder durch geschickte Platzierung von Steuerzeichen des Terminals einen in hellen Farben erstrahlenden Baum ausgeben.

Punkte gibt es für:

- Ein funktionierendes Programm (d.h. die Ausgabe lässt sich zumindest mit Wohlwollen als Baum interpretieren) [*2 Punkte*]
- ein gut dokumentiertes und vor allem gut durchdachtes Programm (d.h. die einzelnen Einheiten des Programms sind sinnvoll in Klassen gekapselt, und die Gedankengänge, die zu dieser Kapselung führten, sind aus den Kommentaren ersichtlich) [*2 Punkte*]
- Kreativität und Funktionsumfang (z.B. randomisierte Platzierung des Schmucks, Sicherstellen, dass genügend freier Platz zwischen den Kugeln bleibt, farbige Ausgabe, variable Größe oder Breite des Baumes, oder was Ihnen eben sonst so einfällt) [*6 Punkte*]

Schreiben Sie zu Beginn Ihres Programmes einen großen Kommentar, in dem Sie auflisten, welche Besonderheiten Ihres Programmes Ihrer Meinung nach für die Bewertung der dritten Teilaufgabe von Belang sind.

Anmerkung: Farbige Ausgabe wird unter Unix durch Escape-Sequenzen[‡] gesteuert. Diese haben die Form

`\033[x;ym` oder `\033[ym` (hier ist $x = 0$) oder auch `\033[m` (hier sind $x = y = 0$),

wobei x und y für Codes stehen, die die Eigenschaften der Schrift beschreiben. So steht $x = 1$ für Fettdruck und $y = 34$ für Blau, so dass ein Text, der zwischen

`\033[1;34m` und `\033[m`

steht, in blauem Fettdruck gesetzt wird. `\033[m` setzt die Schrifteigenschaften wieder auf das Default. Die möglichen Farbkombinationen sind an vielen Stellen im Netz aufgelistet, z.B. unter `http://www.pixelbeat.org/docs/terminal_colours/`.

Um ein lesbares Programm zu erhalten, bietet es sich an, eine Klasse zu schreiben, die in der Lage ist, übergebene Strings "farbig" zurück zu geben, indem sie sie mit den passenden Steuersequenzen umgibt.

10 Punkte

ÜBUNG 10.3 AUFWAND UND LAUFZEITEN

Das Tupel $A = (a_1, ..., a_n)$ enthalte n reelle Zahlen. Gesucht seien weitere n Zahlen $M = (m_1, ..., m_n)$, die folgendermaßen definiert sind:

$$m_i := \frac{1}{i} \sum_{k=1}^{i} a_k,$$

d.h. m_i ist der Mittelwert der ersten i Zahlen aus dem Tupel A.

Wir wollen zwei Funktionen für diese Aufgabe schreiben und deren Laufzeitverhalten bei wachsender Anzahl n anhand einer kleinen Testreihe aus n_{Level} Versuchen beobachten. In der Datei

[‡] `http://de.wikipedia.org/wiki/Escape-Sequenz`

`mittelwerte.cc` finden Sie das Hauptprogramm, welches über die Kommandozeile drei Zahlen einliest: n_{Start}, n_{Level} und $n_{\text{Verbosity}}$. n_{Start} wird nach jedem Versuch verdoppelt, insgesamt $n_{\text{Level}} - 1$ mal. $n_{\text{Verbosity}}$ schaltet die Ausgabe der Daten ein ($\neq 0$) oder aus ($= 0$).

Die Funktion `repeatTest` soll für ein festes n die obige Aufgabe durchführen. Es legt ein dynamisches Feld `a` der Größe n an.

1. Schreiben Sie die fehlenden Funktionen

   (a) `initialize`, die das Feld `a` mit n möglichst verschiedenen reellen Zahlen auffüllt, die nicht zu schnell wachsen, z.B.:

   $$a_k = (-1)^{k+1} k \cos(k) \; .$$

   Mathematische Funktionen in C++ sind z.B. auf der Seite `http://en.cppreference.com/w/cpp/numeric/math` mit vielen Beispielen dokumentiert. Sie werden über `#include<cmath>` verfügbar gemacht. Wie hoch ist die Komplexität in Abhängigkeit von n? (1 Punkt)

   (b) `printArray`, die den Inhalt von `a` in einer einzigen Zeile ausgibt (mit zwei Leerzeichen Abstand zwischen jeder Zahl). Zeilenumbruch bitte erst einfügen, nachdem die n Zahlen ausgegeben sind. Wenn $n > 5$ ist, bitte nur die letzten 5 Zahlen ausgeben! (1 Punkt)

   (c) `meanValues`, die nacheinander die Zahlen m_i nach obiger Formel berechnet und in das Feld `meanArray1` steckt. Wie hoch ist die Komplexität in Abhängigkeit von n? (1 Punkte)

2. Wie lässt sich der Aufwand um eine Potenz reduzieren? (Hinweis: Bei der Berechnung der m_i dürfen bereits berechnete Vorgänger benutzt werden!) Schreiben Sie dafür die fehlende Funktion `meanValuesFast`. (2 Punkte)

Vergleichen Sie die Entwicklung der Laufzeiten der beiden Funktionen `meanValues` und `meanValueFast`, z.B. für $n_{\text{Start}} = 2000$ und $n_{\text{Level}} = 6$.

5 Punkte

ÜBUNG 10.4 KOMPLEXITÄT VON ALGORITHMEN

a) Gauss-Elimination:

```
for (k = 0; k < n - 1; k = k + 1)
  for (i = k + 1; i < n; i = i + 1)
    for (j = k + 1; j < n; j = j + 1)
      a[i][j] = a[i][j] - a[i][k] * a[k][j] / a[k][k];
```

Die Berechnung innerhalb der innersten Schleife ist von den Werten der Schleifenzähler unabhängig, d.h. benötigt eine konstante Zeit t_g. Wie ist die algorithmische Komplexität in Abhängigkeit von `n`? (2 Punkte)

b) Rekursiver Funktionsaufruf:

```
int f (int a, int b) {
  if (a >= b)
    return g(a);
  else
    return f(a, b - 1) + f(a + 1, b);
};
```

Überlegen Sie zuerst, welche Größe als Komplexitätsparameter sinnvoll ist! Die Zeit, um `g(a)` auszurechnen, soll vom Wert von `a` unabhängig sein. Wie ist die algorithmische Komplexität? (3 Punkte)

5 Punkte