C++-Klausur SS16.1

Spiel des Lebens

In den Aufgaben A1, A2 und A3 geht es darum, das bekannte Spiel des Lebens (Conway's Game of Life) zu implementieren und ein paar Zellen zu kultivieren. Entworfen hat dieses System 1970 der Mathematiker John Horton Conway. Es basiert auf einem zweidimensionalen zellulären Automaten und beschreibt das Entstehen und Sterben von Zellen in Abhängigkeit von ihrer jeweiligen Umgebung.

Das rechteckige Spielfeld ist in Spalten (x Pos.) und Zellen (y Pos.) unterteilt. Die Nummerierung beginnt bei 0 und läuft von links nach rechts und von oben nach unten. An jeder Position befindet sich eine Zelle mit zwei Zuständen, die als tot oder lebendig bezeichnet werden. Wenn wir vereinbaren, dass das Feld keinen Rand besitzt sondern an der jeweils gegenüberliegenden Seite fortgesetzt wird (Torus), dann besitzt jede Zelle genau 8 Nachbarn. Beispiel: Bei einer Breite von 5 Feldern sind die gültigen x-Positionen des Feldes 0…4, ein x-Wert von 5 entspricht der Position 0, ein x-Wert von 6 der Position 1 und umgekehrt ein x-Wert von -1 der Position 4 usw. Entsprechend für die Zeilen. Die Zelle an Spalte 0 und Zelle 1 (0,1) besitzt demnach die Nachbarschaft { (4,0), (0,0), (1,0), (4,1), (1,1), (4,2), (0,2), (1,2) }.

Zu Beginn wird eine Anfangsgeneration (n=0) von Zellen auf dem Spielfeld verteilt und danach wird (im Prinzip gleichzeitig) für jede Position des Feldes jeweils anhand der Anzahl lebender Nachbarn in der aktuellen Generation n bestimmt, ob diese Zelle in der nächsten Generation n+a noch lebt (2 oder 3 lebende Nachbarn) oder stirbt (0, 1 oder mehr als 3 Nachbarn), oder ob an der Position einer toten Zelle eine lebende entsteht (genau 3 lebende Nachbarn). Diese so bestimmten „neuen" Zellen bilden dann die nächste Generation.

Aufgabe A1

In dieser Aufgabe geht es um Datenstrukturen für das Spiel des Lebens, speziell um 2er-Tupel, etwa (1,0), und Vektoren von 2er-Tupeln, also eines Vektors z.B. der Form { (3,3), (1,0), (0,1), (1,1), (2,1), (1,2) }.
Gedacht sind diese Vektoren, um in Aufgabe A2 und A3 die Dimension des Spielfeldes und die Zellenpositionen zu beschreiben. Dabei wir d das erste Tupel die Dimensionen und die restlichen Tupel die Positionen der lebenden Zellen beschreiben, Hier ist es aber erst einmal ein genereller Vektor von 2er-Tupeln.

Realisieren Sie folgende Anforderungen für das 2er-Tupel:

a) Die generische Klasse Txy besitzt zwei Template-Parameter T1 und T2
b) Sowie zwei öffentliche Member x und y vom Typ T1 bzw. T2.
c) Es gibt genau einen öffentlichen Konstruktor mit zwei Argumenten vom Typ T1 bzw. T2, der x bzw. y initialisiert.
d) Der globale Typ xy_t ist definiert als die konkrete Ausprägung von Txy mit T1=T2=int.

Punkte: [_____ / 2 + 2 + 2 + 2]

Alternativ mit Abzug: Bei Problemen nehmen Sie einen struct xy_t mit zwei int Membern x und y.

Realisieren Sie folgende Anforderungen für den beschriebenen Vektor:

e) Die Klasse xyListe erbt von vector<xy_t> und kann so 2er-Tupel speichern.
f) Sie besitzt einen privaten Konstruktor. Um Instanzen zu erzeugen existieren drei öffentliche Fabrikmethoden names „wandle“ mit unterschiedlichen Parametern, die jeweils eine Instanz erzeugen, entsprechend füllen und sie zurückgeben. Siehe auch das Beispiel in main.
g) wandle(x, y, initializer_list) für Parameter der Form (3, 3, { {1,0}, {0,1} })
h) wandle(dateiname) für Daten aus der Datei namens dateiname, siehe Beispieldaten in blinker.txt.
i) wandle(vector<int>) für Daten der Form vector<int>({3, 3, 0, 0, 0, 1, …, 0, 1})
j) Es existiert ein Ausgabeoperator op<<, der eine xyListe in der Form [(3,3) (1,0) (0,1)] ausgibt.

Punkte: [_____ / 2 + 2 + 6 + 8 + 8 + 2]

Alternativ mit Abzug: Bei generellen Problemen mit der Klasse arbeiten Sie mit einer Liste vom Typ vector<xy_t> und globalen Methoden.
Kommentieren Sie die Ausgabe in main ein.

Aufgabe A2

In dieser Ausgabe geht es um Spielfeld-Datenstrukturen. **Realisieren Sie:**

a) Die Klasse FeldBasis besitzt einen Konstruktor mit einem vector<xy_t> als Parameter. Dieser speichert die Dimensionen des Feldes, welche in dem ersten Eintrag des Vektors (x und y) gegeben sind, in zwei Membervariablen size_x und size_y vom Typ int.

b) Des Weiteren legt der Konstruktor dynamisch ein int-Feld der Größe 2*size_x*size_y an und speichert die Adresse in einem int-Zeiger feld (alternativ smart Pointer). Dadurch, dass in Aufgabe A3 immer aus der aktuellen Generation von Zellen eine nächste Generation berechnet wird, reich es aus, Platz für nur zwei Generationen zu reservieren und die Rollen der aktiven und der nächsten Generation bei jeder Berechnung zu tauschen. Siehe Bedeutung der Variablen no in A3 und Indexberechnung in d).

c) Das Feld feld wird zunächst mit 0 initialisiert (Zelle tot). Die restlichen Tupel (x,y) des übergebenen Vektors beschreiben die Startpopulation und diese Zellen werden in dem int-Feld mit 1 markiert (Zelle lebt). Am Ende der Initialisierung befinden sich daher nur 0en und 1en in dem Feld. Nutzen Sie zur Adressierung der Zellen immer den Operator aus d).

d) Da der []-Operator nur ein Argument besitzen darf, nutzen wir hier den ()-Operator, um auf eine Zelle lesend bzw. schreibend zuzugreifen. Es gibt folglich einen ()-Operator (bzw. einen zum Lesen (const) und einen zum Schreiben) mit drei Parametern. Das erste Argument k (0 oder 1) gibt an, auf welches der beiden möglichen zwei Felder zugegriffen werden soll. Die beiden folgenden Parameter x und y geben die Position der Zelle an. Der Index einer Zelle ergibt sich daher zu k*size_x*size_y + y*size_x + x. Wählen Sie die Rückgabetypen geeignet. Beachten Sie hier jeweils auch den Überlauf (Torus).

e) Es gibt einen Ausgabeoperator, der ein Feld vergleichbar der Darstellung in der Beispieldatei blinker.txt bzw. dem Muster in main ausgibt.

f) Es gibt (in jedem Fall) einen Destruktor, der je nach Notwendigkeit Daten wieder freigibt oder auch leer bleiben kann.

Punkte: [_____ / 8 + 2 + 2]

Alternativ mit Abzug: Bei generellen Problemen mit der Klasse arbeiten Sie mit einem oder zwei globalen Feldern und globalen Methoden.
Kommentieren Sie die Ausgabe in main ein.

Aufgabe A3

In dieser Aufgabe geht es um das Erzeugen einer neuen Population, einmal seriell und einmal parallel. **Implementieren Sie:**

a) Die Klasse FeldBasis wird um eine rein virtuelle Methode schritt() ergänzt.
b) Es gibt eine Membervariable no vom Typ int, die die gerade aktive Population speichert (0 oder 1) – wichtig für d) und e), siehe main.
c) Des Weiteren um eine Methode teilschritt(), die vier int-Parameter x0, x1, y0, y1 erhält und eine neue Population auf dem jeweils nächsten, d.h. dem anderen Spielfeld (1-no) für die Zellenindizes x0..x1, y0..y1 (exklusive x1 und y1) entsprechend den angegebenen Regeln berechnet.

   Punkte: [_____ / 2 + 2 + 8]

d) Zur nicht-parallelen Berechnung gibt es eine Klasse FieldSeriell, die von FeldBasis erbt und die Methode schritt() zur Berechnung eines kompletten Spielfeldes implementiert. Aktualisieren Sie no zu 1-no (alterniert zwischen 0 und 1).
e) Zur parallelen Berechnung gibt es eine Klasse FeldParallel, die von FeldBasis erbt und die Methode schritt() zur parallelen Berechnung der vier Viertel eines Spielfeldes mittels Threads implementiert. Aktualisieren Sie no.
f) Nutzen Sie Lambda-Funktionen zur Angabe der Worker(Thread)funktion.

   Punkte: [_____ / 2 + 4 + 4]

Alternativ mit Abzug: Bei generellen Problemen arbeiten Sie mit globalen Methoden. Kommentieren Sie die Ausgabe und die Schritte in main ein.