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 1 *Mandelbrot-Menge*

Ein äußerst beliebtes Motiv auf Büchern, Postern und Bildschirmhintergründen ist die Mandelbrot-Menge. Diese Menge wird wie folgt definiert: Für jede komplexe Zahl $c \in \mathbb{C}$ wird die Folge z_i von Elementen aus $\mathbb{C}$ rekursiv definiert als

$$z_0 := 0$$
$$z_{n+1} := f(z_n, c) := z_n^2 + c \, .$$

Die Zahl c gehört zur Mandelbrot-Menge M, wenn die dazugehörigen Folgeglieder eine Bahn erzeugen, die ganz innerhalb eines Kreises mit endlichem Radius bleiben [†], d.h. es existiert ein $C \in \mathbb{R}$, so dass $|z_n| := \sqrt{\mathrm{Re}(z_n)^2 + \mathrm{Im}(z_n)^2} \leq C$ für alle $n \in \mathbb{N}$ gilt. Die Mandelbrot-Menge wollen wir im Folgenden visualisieren.

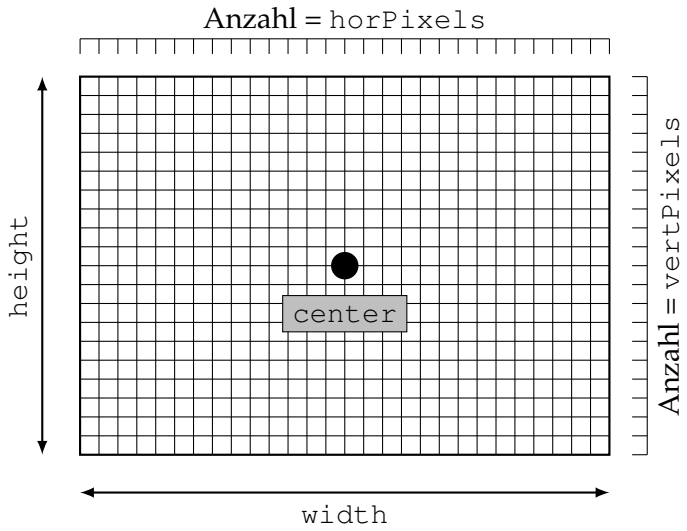
a) Für die Berechnung der Trajektorie soll hier der zusammengesetzte Datentyp `struct Complex` verwendet werden, der in der Vorlesung behandelt wurde. Vervollständigen Sie die Implementierung von `add_complex`, `multiply_complex`[‡] und `betrag` in `mandelbrot.cc`. Diese Funktionen sollen als Prozeduren implementiert werden, indem sie *in-place* operieren, d.h. statt eine neue komplexe Zahl zu erzeugen, soll das Ergebnis im ersten Argument abgespeichert werden. **[(2 Teilpunkte)]**

b) Zur Visualisierung der Mandelbrot-Menge werden wir ein Bild erzeugen, welches den Betrag von z_n für ein $n \in \mathbb{N}$ darstellt. Zur digitalen Repräsentation von Bildern nutzen wir Pixel (*picture elements*). In dieser Aufgabe entspricht ein Pixel einem `int`, wobei Werte gleich 0 schwarze Pixel darstellen und Werte größer als 0 Grauwerte entsprechen[§]. Den Ausschnitt der komplexen Zahlenebene zur Visualisierung und die Anzahl der horizontalen bzw. vertikalen Pixel wird in einer Klasse `Canvas` (Leinwand) gespeichert, die die Grauwerte der Pixel eines Bildes mit zugehöriger Geometrieinformation beinhaltet. Dies stellen wir durch folgende Member-Variablen dar:

   - Eine Anzahl an Pixeln in x- und in y-Richtung.
   - Zwei `double`-Werten, die die Koordinaten der Bildmitte angeben.
   - Zwei `double`-Werten für die physikalische Höhe und Breite des Bildes.

   Dies ist auf dem folgenden Bild noch einmal illustriert:

[†] `https://de.wikipedia.org/wiki/Mandelbrot-Menge`
[‡] $z_1 z_2 := \{\mathrm{Re}(z_1)\mathrm{Re}(z_2) - \mathrm{Im}(z_1)\mathrm{Im}(z_2)\} + \{\mathrm{Re}(z_1)\mathrm{Im}(z_2) + \mathrm{Im}(z_1)\mathrm{Re}(z_2)\}i$
[§] `https://de.wikipedia.org/wiki/Portable_Anymap`



Sie finden bereits in der Datei `canvas.hh` eine angefangene Implementierung dieser Leinwand.

```
class Canvas
{
public :
  // Konstruktor, erzeuge int* _pixels
  Canvas(double center_x, double center_y,
         double width, double height,
         int horPixels, int vertPixels);

  // Desktruktor, raeume int* _pixels auf
  ~Canvas();

  // gibt die Breite des Bildes zurueck
  double width();

  // gibt die Hoehe des Bildes zurueck
  double height();

  // gibt die Anzahl an horizontalen Pixeln
  int horPixels();

  // gibt die Anzahl an vertikalen Pixeln
  int vertPixels();

  // gebe die Koordinaten des Pixels (i,j) als Complex zurueck
  Complex coord(int i, int j);

  // schreibe value an den Pixel (i,j)
  // Ueberlegen Sie wie aus (i,j) den flachen Index bei row-major bekommen
  void writePixel(int i, int j, int value);

  // Zugang zum Pixel (i,j) im 1D Array
  // Ueberlegen Sie wie aus (i,j) den flachen Index bei row-major bekommen
  int operator()(int i, int j);

  // schreibe Bild mit Dateinamen filename
  void write(std::string filename);
private :
  double _center_x;
  double _center_y;
  double _width;
  double _height;
  int _horPixels;
  int _vertPixels;
  int* _pixels;
};
```

Der Konstruktor

```
Canvas(double center_x, double center_y,
       double width, double height,
       int horPixels, int vertPixels);
```

erzeugt eine Leinwand mit Zentrum (`center_x`, `center_y`), mit `horPixels` pro Zeile und `vertPixels` pro Spalte. Während die Reihenfolge der Pixel prinzipiell beliebig gewählt werden kann, folgen wir der *row-major* Konvention, in welcher Pixel Zeile-für-Zeile angeordnet sind. Wir legen uns hier auf eine Nummerierung beginnend vom linken unteren Eck fest. Demnach wird ein Bild mit Anzahl 2 an vertikalen Pixeln und Anzahl 4 an horizontalen Pixeln zum Beispiel folgendermaßen dargestellt:

$$\begin{array}{cccc} \text{pixel}_4 & \text{pixel}_5 & \text{pixel}_6 & \text{pixel}_7 \\ \text{pixel}_0 & \text{pixel}_1 & \text{pixel}_2 & \text{pixel}_3 \end{array}$$

So entspricht z.B. $(i, j) = (0, 2)$ dem Pixel pixel_2 und $(i, j) = (1, 2)$ dem Pixel pixel_6.

Implementieren Sie die (fehlenden) Methoden der Klasse `Canvas`. **[(4 Teilpunkte)]**

c) Schreiben Sie in der Datei `mandelbrot.cc` eine Funktion

```
void mandelbrot(Canvas& canvas, double threshold,
                int maxIt, std::string filename)
```

die einen `canvas` mit einer Visualisierung der Mandelbrot-Menge füllen und damit eine Bilddatei erzeugen soll. Dabei ist `threshold` ein Radius, ab dem Punkte als "unendlich weit weg" gelten sollen (sonst wäre das Problem nicht in endlicher Zeit lösbar) und `maxIt` ist die Anzahl an maximalen durchzuführenden Iterationen. Bahnen von Testpunkten $c \in \mathbb{C}$, die nach `maxIt` Iterationen noch einen Betrag kleiner als `threshold` haben, gelten als beschränkt.

Implementieren die Iterationsvorschrift und berechnen Sie für jeden Testpunkt aus dem Pixelraster die Bahn. Verwenden Sie dabei die Methoden aus den Aufgabenteilen a) und b). Das Befüllen der Pixel im `canvas` soll dabei wie folgt geschehen:

- Punkte c mit "beschränkten" Bahnen sollen schwarz dargestellt werden (Zahlenwert 0).
- Punkte mit "unbeschränkten" Bahnen sollen eine Graustufe zugeordnet bekommen, die um so heller ist, je länger es dauert, bis der betrachtete Punkt "entkommt". Wählen Sie daher den Logarithmus (`std::log()`) der Iterationszahl als Grauwert und multiplizieren Sie diesen mit 100, um genügend Abstufungen zu erhalten.

Die Datei `pgm.hh` beinhaltet eine vorgefertigte Implementierung, die die Pixelwerte einer Leinwand in eine `pgm`-Bilddatei schreibt. Diese wird durch die Klasse `Canvas` verwendet. Sie brauchen also nur die entsprechende Methode in `canvas` aufrufen um ein Bild unter dem übergebenen Dateinamen zu speichern.

Benutzen Sie zum Testen Ihres Programms zunächst eine kleine Auflösung, bis Sie sicher sind, dass Ihr Code im Wesentlichen das Richtige tut. Als Zentrum Ihres Bildes sollten Sie dabei den Punkt (-1,0) verwenden. Erstellen Sie dann ein hochaufgelöstes Bild, indem Sie für die horizontale Auflösung eine Pixelanzahl von 4000 wählen, sowie 3000 für die vertikale. Variieren Sie die Werte `threshold` und `maxIt`, bis Sie mit dem Ergebnis zufrieden sind. Als Startwert können Sie jeweils mit 1000 beginnen und dann variieren. Welche Effekte haben größere und kleinere Werte jeweils? **[(2 Teilpunkte)]**

(8 Punkte)

Übung 2 *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. Zur Repräsentation der Bool-Werte auf dem Raster können Sie die Leinwandimplementierung aus der Mandelbrot-Aufgabe verwenden, oder auch gerne Ihre eigene Implementierung verwenden. Denken Sie an die Rule of Three, falls Sie einen Copy-Konstruktor, Destruktor oder Zuweisungsoperator implementieren, dann müssen Sie die beiden anderen Methoden ebenfalls schreiben. **[(6 Teilpunkte)]**

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! **[(2 Teilpunkte)]**

(8 Punkte)

Übung 3 *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.

[¶] `http://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens`

[||] Mathematisch gesehen ist der Zellraum dann ein Torus.

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) **[(1 Teilpunkte)]**
- 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) **[(1 Teilpunkte)]**
- 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 Teilpunkte)]**

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. **(8 Punkte)**

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