Programmieren II
SS 2023
Prof. Dr. Elmar Böhler

Informatik/Game Engineering Bachelor
Hochschule Kempten

Übungsblatt 8

Betrügen Sie sich nicht selbst, indem Sie Programme zu den folgenden Aufgabenstellungen googeln und irgendwelches Zeug abschreiben, das Sie nicht verstehen.

1. Implementieren Sie eine Klasse `ComplexNumber`, mit deren Hilfe man in `C++` komplexe Arithmetik darstellen kann. Diese Klasse habe zwei private Attribute `real` und `img` vom Typ `double`. Fügen Sie Ihrer Klasse passende Konstruktoren hinzu.

   (a) Überladen Sie die Operatoren `+, -, *, /` gemäß der bekannten Rechenregeln, so dass diese Operatoren im Kontext `(ComplexNumber, ComplexNumber)` funktionieren. Erinnerung: Für zwei komplexe Zahlen $a + b\mathrm{i}$ und $c + d\mathrm{i}$ gilt:
   - $(a + b\mathrm{i}) + (c + d\mathrm{i}) = (a + c) + (b + d)\mathrm{i}$
   - $(a + b\mathrm{i}) - (c + d\mathrm{i}) = (a - c) + (b - d)\mathrm{i}$
   - $(a + b\mathrm{i}) \cdot (c + d\mathrm{i}) = (ac - bd) + (ad + bc)\mathrm{i}$
   - $(a + b\mathrm{i})/(c + d\mathrm{i}) = h \cdot (ac + bd) + h \cdot (bc - ad)\mathrm{i}$, wobei $h \stackrel{\text{def}}{=} 1/(c^2 + d^2)$.

   Überprüfen Sie Ihre Implementierung indem Sie folgende Gleichheiten in einem Testprogramm nachvollziehen:
   - $(3 + 5\mathrm{i}) + (7 + 2\mathrm{i}) = 10 + 7\mathrm{i}$
   - $(3 + 5\mathrm{i}) - (7 + 2\mathrm{i}) = -4 + 3\mathrm{i}$
   - $(0 + 1\mathrm{i}) \cdot (0 + 1\mathrm{i}) = -1 + 0\mathrm{i}$
   - $x/x = 1 + 0\mathrm{i}$ für alle komplexe Zahlen x (außer der 0).

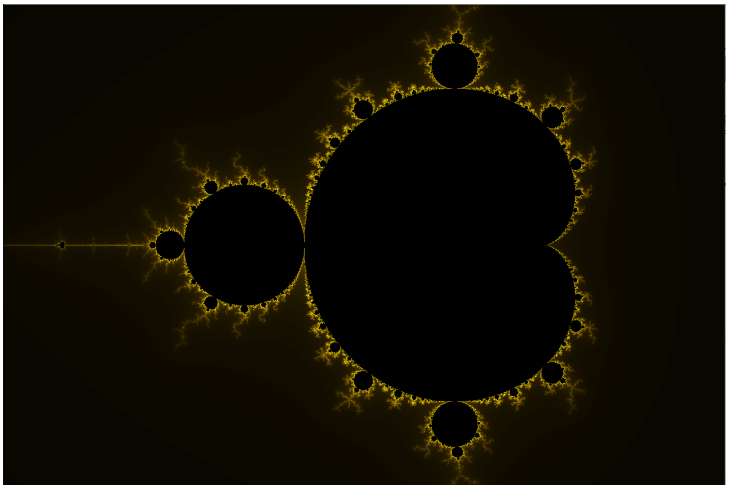
   (b) Sorgen Sie dafür, dass obige Operatoren auch in den Kontexten `(`T`, ComplexNumber)` und `(ComplexNumber,` T`)` funktionieren, wobei T ein beliebiger der defaultmäßig eingebauten Ganzzahl- oder Fließkommatypen ist.

   (c) Überladen Sie die Operatoren `+=, -=, *=, /=` gemäß der üblichen Semantik, nun aber für den Kontext `(ComplexNumber, ComplexNumber)`. Testen Sie die Funktionsweise dieser neuen Operatoren!

   (d) Überladen Sie den Operator `<<` im Kontext `(std::ostream, ComplexNumber)`, so dass die Ausgaben einer `ComplexNumber x` über Ausdrücke wie `std::cout << x << std::endl;` möglich wird.

   (e) Überladen Sie die einstelligen Operatoren `+, -, ~, !` für `ComplexNumber`, wobei $\sim x$ die zu x komplex konjugierte Zahl ist und $!x$ ihr Betrag. Der Betrag einer Komplexen Zahl $a + b\mathrm{i}$ ist gleich der reellen Zahl $\sqrt{a^2 + b^2}$. Die einstelligen `+` und `-` haben ihre übliche Bedeutung.

2. Benutzen Sie Ihre neuen Fähigkeiten mit komplexen Zahlen um ein Bild wie das folgende zu generieren:



Wie Sie dabei vorgehen, steht auf der nächsten Seite.

Für eine Komplexe Zahl x definieren wir die Mandelbrot-x-Folge[1] $m_0(x), m_1(x), \ldots$ gemäß der folgenden Induktion:

(IA) $m_0(x) \stackrel{\text{def}}{=} 0$
(IS) $m_{j+1}(x) \stackrel{\text{def}}{=} (m_j(x))^2 + x$ für alle $j > 0$

Beispiel: Ist $x = 1 + \frac{1}{2}\mathrm{i}$, so ist

- $m_0(x) = 0$
- $m_1(x) = (m_0(x))^2 + x = 0^2 + 1 + \frac{1}{2}\mathrm{i} = 1 + \frac{1}{2}\mathrm{i}$
- $m_2(x) = (m_1(x))^2 + x = (1 + \frac{1}{2}\mathrm{i})^2 + 1 + \frac{1}{2}\mathrm{i} = \frac{7}{4} + \frac{3}{2}\mathrm{i}$
- usw.

Wenn die Zahl x nahe 0 ist, so bleiben die Folgenglieder alle klein, d.h. der Betrag von $m_j(x)$ ist immer klein, für alle $j \in \mathbb{N}$. Für betragsmäßig größere x wächst der Betrag von $m_j(x)$ hingegen stetig an und geht (unterschiedlich schnell) gegen unendlich.

Die Idee bei der Erzeugung des obigen Bildes ist die folgende: Jeder Bildpunkt steht für eine Komplexe Zahl x. Für jede solche Zahl werden nun die ersten paar Elemente der obigen Mandelbrot-x-Folge berechnet. Bleiben alle so berechneten Folgenglieder klein, so wird der Bildpunkt schwarz eingefärbt. Ansonsten gibt es ein kleines j, so dass $m_j(x)$ größer als ein festgelegter Grenzwert ist. Der entsprechende Bildpunkt ist dann umso heller, je größer j ist.

(a) Schreiben Sie eine Funktion mit dem Kopf

```
unsigned int getMandelbrotIndex(ComplexNumber& x, double limes, int max)
```

die auf Eingabe der komplexen Zahl `x` den kleinsten Index j zurückliefert, für den $m_j($`x`$) >$ `limes` gilt oder `max`, falls dieses j nicht existiert oder größer als `max` ist.

(b) Schreiben Sie eine Funktion mit dem Kopf

```
void drawMandelbrotMenge(ViewPortGL& vp)
```

die ein Bild wie oben gezeigt produziert. Weisen Sie dazu jedem Pixel von `vp` eine Komplexe Zahl wie folgt zu: Der Realteil der Zahlen bewegt sich im Bereich $[-2, 1)$, ihr Imaginärteil im Bereich $[-1, 1)$. Hätten wir z.B. einen `ViewPortGL` mit einer Breite von 300 und einer Höhe von 200 Pixeln gegeben, dann hätten zwei Pixel in der Waagerechten einen „Abstand“ von $\frac{1}{100}$, denn der Bereich $[-2, 1)$ hat die Größe 3 und diese wird auf 300 Pixel aufgeteilt.

Dann wäre in diesem Beispiel

- dem Pixel mit den Koordinaten $(0, 0)$ die Zahl $-2 + 1\mathrm{i}$ zugeordnet,
- dem Pixel mit den Koordinaten $(1, 0)$ die Zahl $-2 + \frac{1}{100} + 1\mathrm{i}$ zugeordnet,
- dem Pixel mit den Koordinaten $(2, 0)$ die Zahl $-2 + \frac{2}{100} + 1\mathrm{i}$ zugeordnet,
- dem Pixel mit den Koordinaten $(167, 0)$ die Zahl $-2 + \frac{167}{100} + 1\mathrm{i}$ zugeordnet,
- dem Pixel mit den Koordinaten $(299, 0)$ die Zahl $(1 - \frac{1}{100}) + 1\mathrm{i}$ zugeordnet,
- dem Pixel mit den Koordinaten $(0, 1)$ die Zahl $-2 + (1 - \frac{1}{100})\mathrm{i}$ zugeordnet,
- dem Pixel mit den Koordinaten $(0, 199)$ die Zahl $-2 - (1 + \frac{1}{100})\mathrm{i}$ zugeordnet und
- dem Pixel mit den Koordinaten $(299, 199)$ die Zahl $(1 - \frac{1}{100}) - (1 + \frac{1}{100})\mathrm{i}$ zugeordnet.

Sie sollten Ihren Code so gestalten, dass er auch mit ViewPorts zurechtkommt, die eine andere Größe als 300×200 Pixel besitzt. Die schönsten Resultate erhalten Sie aber, wenn die Seitenverhältnisse des ViewPort mit dem abzubildenden komplexen Bereich (im obigen Beispiel also $3 : 2$) übereinstimmen.

Es ist nun also jedem Koordinatenpaar (x, y) aus dem Grafikfenster eine komplexe Zahl Paar von $(a + b\mathrm{i})$ zugeordnet.

Berechnen Sie für jeden Punkt (x, y) des Grafikfensters die zugehörige komplexe Zahl $(a + b\mathrm{i})$ und rufen Sie jeweils `getMandelbrotIndex` mit dieser Zahl als Argument auf. Als weitere Argumente übergeben Sie z.B. 100 und 255 (mit diesen Werten können Sie dann ein bisschen spielen). Speichern Sie das Ergebnis des

[1] Benoît Mandelbrot, 1924 – 2010, französicher Mathematiker

Aufrufs in einer Variablen `n`. Färben Sie den Punkt (x, y) schwarz, falls `n` mindestens den Wert 255 besitzt. Ansonsten färben Sie den Punkt (x, y) mit der Farbe, deren Rot-, Grün- und Blaukomponente durch `(n,n,0)` gegeben ist.

Sie sind natürlich herzlich eingeladen mit Farben bzw. Abbildungen in den Farbraum herumzuspielen. Zum Beispiel erhalten Sie ein etwas hübscheres Ergebnis (mit mehr bunten Pixeln), wenn Sie jeden Punkt mit der Farbe `(`f`(n),`f`( n) ,0)` färben, wobei $f(x) \stackrel{\text{def}}{=} \frac{2x^2}{510} + 2x$.

Die Erstellung eines solchen Mandelbrot-Bildes dauert ein bisschen, weil ja für jeden Punkt die Mandelbrot-Folge ansatzweise berechnet werden muss; falls der Rechner also ein paar Sekunden nichts zu tun scheint, heißt das nicht zwangsläufig, dass sich das Programm aufgehängt hat. Vielleicht testen Sie Ihr Programm zunächst mit kleinen ViewPorts.

(c) Können Sie die oben beschriebene Funktion in eine mit dem Kopf

```
void drawMandelbrotMenge(ViewPortGL& vp, const ComplexNumber& upperLeft,
                         const ComplexNumber& lowerRight)
```

abändern, die einen anderen Ausschnitt der komplexen Ebene anzeigt, wobei der Ausschnitt durch `upperLeft, lowerRight` gegeben ist?