Lösungen zu Übungsblatt 8

Ralph Gauges Ursula Rost Katja Wegner

18.12.2007

Aufgabe 1:

Schreiben Sie das Programm zur Berechnung der Fakultät einer Zahl aus Aufgabe 2 vom 31.10.2007 so um, dass das Programm den Wert, für den die Fakultät berechnet werden soll, als Kommandozeilenparameter entgegen nimmt.
Überprüfen Sie aber, ob der Benutzer nur einen Parameter angegeben hat, ob dieser das richtige Format hat und ob dieser im erlaubten Bereich liegt, so wie es in der ursprünglichen Aufgabe verlangt war.

```
1 #include <string>
2 using namespace std;
3 int main(int argc, char** argv)
4 {
5   if(argc==2)
6   {
7    string s=argv[1];
8    if(s.size() == 1 || s.size() == 2)
9    {
10     if(istEineZahl(s[0],(s.size()==1)?0:s[1])==true)
11     {
12       unsigned int zahlenWert=umwandelnInZahl(s[0],(s.size()==1)?0:s[1]);
13       if(zahlenWert==0)
14       {
15         cerr << "Bitte geben Sie eine Zahl zwischen 1 und 20 an." << endl;
16       }
17       else if(zahlenWert > 20)
18       {
19         cerr << "Bitte geben Sie eine Zahl zwischen 1 und 20 an." << endl;
20       }
21       else
```

```
22        {
23          cout <<  "Fakultaet von " << zahlenWert << " ist ";
24          cout << fakultaet(zahlenWert) << endl;
25        }
26      }
27      else
28      {
29        cerr << "Es handelt sich bei der Eingabe nicht um eine Zahl." << endl;
30      }
31     }
32     else
33     {
34       cerr << "Bitte geben Sie eine Zahl zwischen 1 und 20 an." << endl;
35     }
36   }
37   else
38   {
39     cerr << "Usage: fakultaet ZAHL" << endl;
40     cerr << "Die Zahl muss zwischen 1 und 20 liegen." << endl;
41   }
42   return 0;
43 }
```

Die Struktur dieser Version unterscheidet sich nicht sehr von der ursprünglichen Fassung. Die Funktion zum Überprüfen, ob es sich um eine Zahl handelt, die Funktion zum Umwandeln in einen Zahlenwert und die Funktion zur Berechnung der Fakultät sind gleich geblieben.

Zeile 2: Macht alle Symbole aus dem Namespace *std* im globalen Namespace verfügbar.

Zeile 5: Überprüft ob der Benutzer genau ein Argument an das Programm übergeben hat.

Zeile 7-8: Erzeugt einen C++ String aus dem zweiten Kommandozeilenargument. (Das erste Argument ist der Name des Programms.) Anschliessend wird überprüft ob der Benutzer nur ein oder zwei Zeichen eingegeben hat.

Zeile 10: Benutzt den Bedingungsoperator um der Funktion *istEineZahl* die richtigen Zeichen zu übergeben.

Zeile 12: Benutzt auch den Bedingungsoperator um der Funktion *umwandelnInZahl* die richtigen Zeichen zu übergeben.

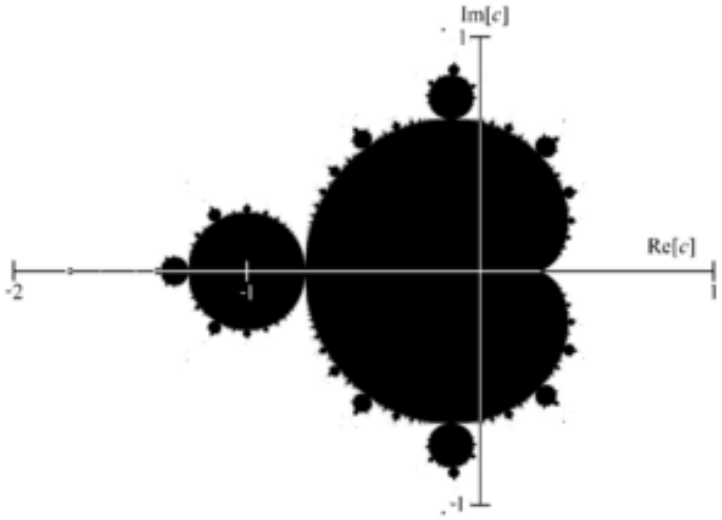

Abbildung 1: Mandelbrotmenge (Quelle: Wikipedia)

Der Rest des Programmes unterscheidet sich nur in kleinen Details von der ursprünglichen Version, z.B. werden die Fehlermeldungen auf *cerr* ausgegeben und nur die eigentliche Ausgabe auf *cout*. Das Programm

Aufgabe 2:

Schreiben Sie ein Programm, welches die Mandelbrotmenge berechnet und in Form einer TARGA Grafik abspeichert. Zum Abspeichern der Bilddaten können Sie die Klasse aus Aufgabe 3 vom 05.12.2007 benutzen.
Zur Berechnung der Mandelbrotmenge benötigen Sie außerdem die Klasse für komplexe Zahlen aus Aufgabe 3 vom 28.11.2007 bzw. Aufgabe 1 vom 05.12.2007.
Die Mandelbrotmenge ist die Menge aller komplexen Zahlen c für die die rekursiv definierte Folge komplexer Zahlen z0,z1,z2,... mit dem Bildungsgesetz $z_{n+1} = z_n^2 + c$ und der Anfangsbedingung $z_0 = 0$ beschränkt bleibt, d.h. der Betrag der Folgenglieder wächst nicht über alle Grenzen.
Der Realteil der komplexen Zahlen, die zur Mandelbrotmenge gehören liegt dabei in etwa zwischen -2.0 und $+1.0$ der Imaginärteil der Zahlen zwischen $-1.0i$ und $+1.0i$.

Um herauszufinden, welche komplexen Zahlen zur Mandelbrotmenge gehören und welche nicht, muss man die komplexen Zahlen in diesem Rechteck($a \in [-2, 1]$ und $b \in [-1, 1]$ wobei $z = a + b_i$, siehe auch Abbildung 1) daraufhin untersuchen, ob die oben definierte Zahlenfolge immer weiter wächst, wenn man die zu untersuchende Zahl als c in die Gleichung einsetzt, oder ob die Folge beschränkt bleibt, d.h. immer unterhalb eines bestimmten Wertes. Man kann in unserem Beispiel davon ausgehen, dass eine Wertefolge beschränkt bleibt, wenn der berechnete Wert nach 100 Iterationen einen Betrag kleiner oder gleich zwei hat. Der Betrag der komplexen Zahl $z = a + bi$ ist $\sqrt{a + b^2}$. Obwohl die Wertefolge rekursiv definiert ist, ist es in diesem Fall geschickter, die Folge iterativ zu programmieren, da die Rekursionstiefe schnell so groß wird, dass der Speicher auf dem Stack nicht mehr ausreicht.
Nimmt man zum Beispiel die komplexe Zahl c mit dem Wert $-1.9 + 0.8i$ welche ziemlich weit am rechten oberen Rand des Rechtecks liegt. Für diese Zahl ergibt sich die Zahlenfolge:

$$
\begin{aligned}
z_0 &= 0 \\
z_1 &= z_0^2 + c = c \\
z_2 &= z_1^2 + c = c^2 + c = 2.97 + 3.04i
\end{aligned}
$$

Der Betrag von z2 ist somit $\sqrt{2.97^2 + 3.04^2} = 4.25$ und somit bereits nach der zweiten Iteration größer als zwei. Somit gehört der Wert $1.9 + 0.8i$ nicht zur Mandelbrotmenge.
Natürlich liegen in dem Rechteck unendlich viele Zahlen und man kann nicht alle testen. Wieviele Zahlen gestestet werden sollen, hängt von der Größe des Bildes ab, das man erstellen will. Da das Rechteck eine Breite von drei Einheiten hat und eine Höhe von zwei Einheiten, bietet es sich an, z.B. mit einer Bildgröße von 600x400 Punkten zu beginnen. D.h. man muss das Rechteck in der Breite in 600 Schritte teilen und in der Höhe in 400 Schritte. Die Schrittgröße beträgt für die Breite 3.0/600.0 und für die Höhe 2.0/400.0 (was jeweils 0.0005 ergibt). Wenn wir dann z.B. links oben im Rechteck bei $c = -2.0+1.0i$ beginnen, testen wir als zweites den Wert $c = -2.005 + 1.0i$ und fahren fort bis zum Wert $c = 1.0 + 1.0i$. Nun sind wir mit der ersten Reihe fertig und beginnen mit der nächsten Reihe, d.h. dem Wert $c = -2.0 + 0.995i$ usw. bis wir alle 400 Reihen durch haben.

Die Punkte des Bildes, welche zu den komplexen Zahlen gehören, die beschränkt bleiben, färbt man meist schwarz. Dafür wie man die restlichen Punkte, welche nicht zur Mandelbrotmenge gehören, einfärbt, gibt es mehrere Verfahren. Das einfachste ist es natürlich, allen Punkten, die nicht zur Mandelbrotmenge gehören, dieselbe Farbe zu geben. Andere Verfahren bestimmen die Farbwerte der restlichen Punkte anhand der Anzahl der Iterationen, welche gebraucht wurden, um den Grenzwert zu überschreiten. Es lohnt sich sicherlich mit der Farbgebung etwas herumzuspielen und sich verschiedene Verfahren auszudenken.
Ein weiterer Reiz der Mandelbrotmenge besteht darin, bestimmte Bereiche zu vergrößern. D.h. man berechnet ein Bild für einen bestimmten Ausschnitt der Mandelbrotmenge. Versuchen Sie doch einmal, das Programm so umzuschreiben, dass der Benutzer die Grenzen für einen Bereich eingeben kann und anschließend für diesen Bereich das Bild berechnet wird. Man muss dabei beachten, dass je nachdem welchen Bereich man wählt, man die Anzahl der Iterationen erhöhen muss und dass ab einer gewissen Anzahl an Iterationen die Genauigkeit von **double** Werten nicht mehr ausreicht und man unter Umständen auf **long double** umsteigen muss. Irgendwann wird auch der **long double** Datentyp nicht mehr ausreichen.

```
1 #include "ComplexNumber.h"
2 #include "Image.h"
3
4 #include <math.h>
5
6 int main()
7 {
8   unsigned int width=600;
9   unsigned int height=400;
10  double minX=-2.0l;
11  double minY=-1.0l;
12  double maxX=1.0l;
13  double maxY=1.0l;
14  unsigned int maxIterations=1000;
15  double maxBetrag=2.0;
16  double stepSizeX=(maxX-minX)/(double)width;
17  double stepSizeY=(maxY-minY)/(double)height;
18  ComplexNumber c;
19  ComplexNumber z;
20  Image image(width,height);
21  unsigned int i,j;
22  for(i=0;i<height;++i)
```

```
23    {
24      for(j=0;j<width;++j)
25      {
26        z=ComplexNumber(0.0,0.0);
27        double betrag=0.0;
28        c=ComplexNumber(minX+j*stepSizeX,minY+i*stepSizeY);
29        unsigned int k;
30        for(k=0;k < maxIterations && betrag < maxBetrag;++k)
31        {
32          z=z*z+c;
33          betrag=sqrt(z.getRealPart()*z.getRealPart()+z.getImaginaryPart()*z.getImagin
34        }
35        if(k==maxIterations)
36        {
37          image.setPixel(j,i,0,0,0);
38        }
39        else
40        {
41          image.setPixel(j,i,0,0,255);
42        }
43      }
44    }
45    image.saveTGA("mandelbrot.tga");
46  }
```

Zeile 1-4: Importiert die Deklarationen der Klassen zum Rechnen mit komplexen Zahlen und zum Arbeiten mit Bilddaten, sowie die Deklaration der *sqrt* Funktion zur Berechnung der Quadratwurzel einer Zahl.

Zeile 8-21: Deklariert die Variablen die für die Berechnung benötigt werden und initialisiert eine Instanz der Klasse *Image* mit der gewünschten Höhe und Breite.

Zeile 22: Schleife über alle Zeilen des Bildes.

Zeile 24: Schleife über alle Spalten des Bildes.

Zeile 26-29: Initialisiert die beiden komplexen Zahlen z und c. Die Position von c ergibt sich aus der aktuellen Zeile i und der aktuellen Spalte j, sowie den Schrittweiten in den einzelnen Dimensionen und den Minimalwerten der beiden Dimensionen.

Zeile 30: Schleife welche entweder so lange läuft bis k den Wert von *maxIterations* erreicht, oder der berechnete Betreag (*betrag*) größer ist als der Wert von *maxBetrag*.

Zeile 32-33: Berechnet den nächsten Wert von z und den neuen Wert für *betrag*.

Zeile 35-42: Überprüft, ob k gleich der maximalen Anzahl der Iterationen ist, wenn ja, dann wird der entsprechnde Pixel auf die Farbe Schwarz gesetzt, da der Punkt zur Mandelbrotmenge gehört, andernfalls wird der Pixel auf die Farbe Weiß gesetzt.

Zeile 45: Speichert das Bild in der Datei mandelbrot.tga

Das Programm läßt sich sicherlich an einigen Stellen verbessern, so kann man z.B. die Farbgebung etwas interessanter gestalten, oder man kann den Benutzer einige der Variablen eingeben lassen oder dem Programm als Kommandozeilenargumente übergeben. Ausserdem braucht man die *sqrt* Funktion nicht wirklich und kann das Programm sogar noch etwas beschleunigen, wenn man sie nicht verwendet.