

Wissenschaftliches Programmieren für Ingenieure

Übungsaufgabe

Name: Wen

Vorname: Yi

Immatrikulationsnummer: 2105066

Studiengang: Maschinenbau

> Aufgabe 1: Rechnen mit komplexen Zahlen

In *main_complex_beispiel.cpp* wird *complex.h* als Header File importiert. Hier werden Operatoren ($=$ $+$ $-$ $*$ $/$ sowie $^$) für zukünftige Rechnungen übergeladet. Dabei werden 3 Situationen berücksichtigt: komplexe Zahl (z.B) $+$ komplexe Zahl, komplexe Zahl $+$ reelle Zahl, reelle Zahl $+$ komplexe Zahl. Für dritte Situation wird *friend* als Attribut benutzt und wird das Überladen außer Klasse definiert.

1. complex.h

```
1. #ifndef COMPLEX_H
2. #define COMPLEX_H
3.
4. class MyComplex{
5.
6. private:
7.     double m_Real;
8.     double m_Image;
9.
10. public:
11.     // con-/destructor
12.     MyComplex();
13.     MyComplex(const double re, const double im );
14.     MyComplex(const MyComplex & cplx);
15.     ~MyComplex();
16.
17.     // copy operator =
18.     MyComplex & operator=(const MyComplex & cplx);
19.
20.     // overload operator +-* / (with complex number, real number on the left
        and on the right)
21.     const MyComplex operator+(const MyComplex & cplx) const;
22.     const MyComplex operator+(const double & real_number) const;
23.     friend const MyComplex operator+(const double & real_number, const
        MyComplex & cplx); // use friend to define out of class
24.     const MyComplex operator-(const MyComplex & cplx) const;
25.     const MyComplex operator-(const double & real_number) const;
```

```
26.     friend const MyComplex operator-(const double & real_number, const
      MyComplex & cplx);
27.     const MyComplex operator*(const MyComplex & cplx) const;
28.     const MyComplex operator*(const double & real_number) const;
29.     friend const MyComplex operator*(const double & real_number, const
      MyComplex & cplx);
30.     const MyComplex operator/(const MyComplex & cplx) const;
31.     const MyComplex operator/(const double & real_number) const;
32.     friend const MyComplex operator/(const double & real_number, const
      MyComplex & cplx);
33.
34.     // overload operator ^
35.     const MyComplex operator^(const int & exp) const;
36.
37.     const double real() const;
38.     const double imag() const;
39.     const double norm() const;
40. };
41.
42. #endif /* COMPLEX_H */
```

2. complex.cpp

```
1. #include <iostream>
2. #include <cmath>
3.
4. #include "complex.h"
5.
6. using namespace std;
7.
8. MyComplex::MyComplex(const double re, const double im ){ // constructor
9.     this->m_Real = re;
10.    this->m_Image = im;
11. }
12.
13. MyComplex::MyComplex(const MyComplex & cplx){ // copy constructor
14.     this->m_Real = cplx.m_Real;
15.     this->m_Image = cplx.m_Image;
16. }
17.
18. MyComplex::MyComplex(){this->m_Real = 0; this->m_Image = 0;} // default
    constructor
19. MyComplex::~MyComplex(){} // default destructor
20.
```

```
21. MyComplex & MyComplex::operator=(const MyComplex & cplx){ // overload =  
    operator  
22.     this->m_Real = cplx.m_Real;  
23.     this->m_Image = cplx.m_Image;  
24. }  
25.  
26. const MyComplex MyComplex::operator+(const MyComplex & cplx) const{ //  
    overload + operator (complex number)  
27.     MyComplex temp;  
28.     temp.m_Real = this->m_Real + cplx.m_Real;  
29.     temp.m_Image = this->m_Image + cplx.m_Image;  
30.     return temp;  
31. }  
32.  
33. const MyComplex MyComplex::operator+(const double & real_number) const{ //  
    overload + operator (real number)  
34.     MyComplex temp;  
35.     temp.m_Real = this->m_Real + real_number;  
36.     temp.m_Image = this->m_Image;  
37.     return temp;  
38. }  
39.  
40. const MyComplex operator+(const double & real_number, const MyComplex &  
    cplx){ // overload + operator (real number, static, left +)  
41.     MyComplex temp;  
42.     temp = cplx + real_number;  
43.     return temp;  
44. }  
45.  
46. const MyComplex MyComplex::operator-(const MyComplex & cplx) const{ //  
    overload - operator (complex number)  
47.     MyComplex temp;  
48.     temp.m_Real = this->m_Real - cplx.m_Real;  
49.     temp.m_Image = this->m_Image - cplx.m_Image;  
50.     return temp;  
51. }  
52.  
53. const MyComplex MyComplex::operator-(const double & real_number) const{ //  
    overload - operator (real number)  
54.     MyComplex temp;  
55.     temp.m_Real = this->m_Real - real_number;  
56.     temp.m_Image = this->m_Image;  
57.     return temp;
```

```
58.}
59.
60.const MyComplex operator-(const double & real_number, const MyComplex &
    cplx){ // overload - operator (real number, static, left -)
61.    MyComplex temp(real_number,0.);
62.    return temp - cplx;
63.}
64.
65.const MyComplex MyComplex::operator*(const MyComplex & cplx) const{ //
    overload * operator (complex number)
66.    MyComplex temp;
67.    temp.m_Real = this->m_Real * cplx.m_Real - this->m_Image * cplx.m_Image;
68.    temp.m_Image = this->m_Image * cplx.m_Real + this->m_Real * cplx.m_Image;
69.    return temp;
70.}
71.
72.const MyComplex MyComplex::operator*(const double & real_number) const{ //
    overload * operator (real number)
73.    MyComplex temp;
74.    temp.m_Real = this->m_Real * real_number;
75.    temp.m_Image = this->m_Image * real_number;
76.    return temp;
77.}
78.
79.const MyComplex operator*(const double & real_number, const MyComplex &
    cplx){ // overload * operator (real number, static, left *)
80.    MyComplex temp;
81.    temp = cplx * real_number;
82.    return temp;
83.}
84.
85.const MyComplex MyComplex::operator/(const MyComplex & cplx) const{ //
    overload / operator (complex number)
86.    MyComplex temp;
87.    temp.m_Real = (this->m_Real * cplx.m_Real + this->m_Image * cplx.m_Image)
        / (cplx.m_Real * cplx.m_Real + cplx.m_Image * cplx.m_Image);
88.    temp.m_Image = (this->m_Image * cplx.m_Real - this->m_Real * cplx.m_Image)
        / (cplx.m_Real * cplx.m_Real + cplx.m_Image * cplx.m_Image);
89.    return temp;
90.}
91.
92.const MyComplex MyComplex::operator/(const double & real_number) const{ //
    overload / operator (real number)
```

```
93.     MyComplex temp;
94.     temp.m_Real = this->m_Real / real_number;
95.     temp.m_Image = this->m_Image / real_number;
96.     return temp;
97. }
98.
99. const MyComplex operator/(const double & real_number, const MyComplex &
    cplx){ // overload / operator (real number, static, left //)
100.     MyComplex temp(real_number,0);
101.     return temp / cplx;
102. }
103.
104. const MyComplex MyComplex::operator^(const int & exp) const{ // overload ^
    operator
105.     MyComplex temp(1,0);
106.     for (int i = 0; i<exp; i++)
107.     {
108.         temp = temp * *this;
109.     }
110.     return temp;
111. }
112.
113. const double MyComplex::real() const{
114.     return this->m_Real;
115. }
116.
117. const double MyComplex::imag() const{
118.     return this->m_Image;
119. }
120.
121. const double MyComplex::norm() const{
122.     return sqrt(this->m_Real * this->m_Real + this->m_Image * this->m_Image);
123. }
```

3. makefile

```
1. PROG = complex_test
2.
3. FLAGS = -O2
4.
5. CC = g++
6.
7. SRCS = complex.cpp main_complex_beispiel.cpp
8.
```

```
9. OBJ = $(SRCS:.cpp=.o)
10.
11. all: $(SRCS) $(PROG)
12.
13. $(PROG): $(OBJ)
14. $(CC) $(FLAGS) $(OBJ) -o $@
15.
16. %.o: %.cpp
17. $(CC) $(FLAGS) -c $<
18.
19. clean:
20. rm -rf *.o $(PROG)
21.
22. complex.o: complex.cpp complex.h
23.
24. main_complex_beispiel.o: main_complex_beispiel.cpp complex.cpp complex.h
```

4. Berechnungsbeispiel

Gegeben sind $z_1 = 2.+7.i$ und $z_2 = 42. - 9.i$ und $z_3 = -11.+19.i$.

- a) $z_4 = z_1 * z_2$
- b) $z_5 = z_1 + z_2$
- c) $z_6 = (z_1 + z_2) * 2.$
- d) $z_7 = (z_2 + z_3) * z_1$
- e) $z_8 = z_1 + 5.$

Ergebnis:

```
> z1 : (2, 7)
> z2 : (42, -9)
> z3 : (-11, 19)
> z4=z1*z2 =: (147, 276)
> z5=(z1+z2) =: (44, -2)
> z6=(z1+z2)*2. = : (88, -4)
> z7=(z2+z3)*z1 = : (-8, 237)
> z8=z1+5. = : (7, 7)
> z8=z8*10. = : (70,70)
```

> Aufgabe 2: Untersuchung der Konvergenz komplexer Zahlenfolgen

Hier wird *complex.h* als Header File in *complex_convergency.cpp* importiert.

IsConvergency() kann mittels gegebenen Parametern/Bedingungen die Anzahl der Iterationschritte bei Erreichen des Konvergenzradiuses rückgeben.

calcu() kann mittels gegebenen Parametern/Bedingungen eine Datei erstellen, deren Inhalt durch *GNU PLOT* dargestellt wird.

plot.gp ist ein GNU PLOT-Skript, mit dem die Rechnungsergebnisse von *complex_convergency.cpp* plottet und als jpg-Datei exportiert werden können.

runs.sh kann die ganze Vorgänge mit formatierten Dateien als Eingabe automatisch durchführen.

1. complex_convergency.cpp

```
1. #include "complex.h"
2. #include <iostream>
3. #include <fstream>
4.
5. using namespace std;
6.
7. int IsConvergency(MyComplex z0, int exp, MyComplex c, double rc, int N_max)
8. {
9.     MyComplex z_temp = z0;
10.    for (int i = 0; i < N_max; i++)
11.    {
12.        z_temp = (z_temp ^ exp) + c;
13.        if (z_temp.norm() >= rc) return i;
14.    }
15.    return N_max;
16.}
17.
18. void calcu(int Iter_num, double value_l_real, double value_l_imag, double
    value_r_real, double value_r_imag,
19.            int Nxmax, int Nymax, int exp, int N_max, double rc, char
    filename[], double c_real, double c_imag)
```



```
20.{
21.    char file[256];
22.    sprintf(file,filename);
23.    ofstream calc_output;
24.
25.    double dx = (value_r_real - value_l_real) / Nxmax;
26.    double dy = (value_r_imag - value_l_imag) / Nymax;
27.
28.    calc_output.open(file);
29.
30.    switch(Iter_num)
31.    {
32.        case 1:
33.        {
34.            MyComplex c(c_real, c_imag);
35.            for(int i = 0; i<=Nxmax; i++)
36.            {
37.                for(int j = 0; j<=Nymax; j++)
38.                {
39.                    MyComplex z0(double(i)*dx + value_l_real, double(j)*dy +
value_l_imag);
40.                    calc_output<<i<<" "<<j<<" "<<IsConvergency(z0, exp, c, rc,
N_max)<<endl;
41.                }
42.                calc_output<<endl;
43.            }
44.            break;
45.        }
46.        case 2:
47.        case 3:
48.        {
49.            MyComplex z0(c_real, c_imag);
50.            for(int i = 0; i<=Nxmax; i++)
51.            {
52.                for(int j = 0; j<=Nymax; j++)
53.                {
54.                    MyComplex c(double(i)*dx + value_l_real, double(j)*dy +
value_l_imag);
55.                    calc_output<<i<<" "<<j<<" "<<IsConvergency(z0, exp, c, rc,
N_max)<<endl;
56.                }
57.                calc_output<<endl;
58.            }
```

```
59.         break;
60.     }
61. }
62. calc_output.close();
63.}
64.
65.int main()
66.{
67.    int Iter_num;
68.    double value_l_real, value_l_imag, value_r_real, value_r_imag;
69.    int Nxmax, Nymax, exp, N_max;
70.    double rc;
71.    char filename[256];
72.    double c_real, c_imag;
73.
74.    cout<<"input number of iteration process"<<endl;
75.    cin>>Iter_num;
76.    cout<<"input start value and end value of complex"<<endl;
77.    cin>>value_l_real>>value_l_imag>>value_r_real>>value_r_imag;
78.    cout<<"input Nxmax and Nymax"<<endl;
79.    cin>>Nxmax>>Nymax;
80.    cout<<"input exponent"<<endl;
81.    cin>>exp;
82.    cout<<"input Nmax"<<endl;
83.    cin>>N_max;
84.    cout<<"input convergency radius"<<endl;
85.    cin>>rc;
86.    cout<<"input output filename"<<endl;
87.    cin>>filename;
88.    cout<<"input start value of c / z"<<endl;
89.    cin>>c_real>>c_imag;
90.
91.    calcu(Iter_num, value_l_real, value_l_imag, value_r_real, value_r_imag,
        Nxmax, Nymax, exp, N_max, rc, filename, c_real, c_imag);
92.
93.    return 0;
94.}
```

2. makefile

1. **PROG** = convergency
- 2.
3. **FLAGS** = -O2
- 4.

```
5. CC = g++
6.
7. SRCS = complex.cpp complex_convergency.cpp
8.
9. OBJ = $(SRCS:.cpp=.o)
10.
11.all: $(SRCS) $(PROG)
12.
13.$(PROG): $(OBJ)
14. $(CC) $(FLAGS) $(OBJ) -o $@
15.
16.%.o:%.cpp
17. $(CC) $(FLAGS) -c $<
18.
19.clean:
20. rm -rf *.o $(PROG)
21.
22.## dependencies
23.
24.complex.o: complex.cpp complex.h
25.
26.main_complex_beispiel.o: complex_convergency.cpp complex.cpp complex.h
```

3. plot.gp

```
1. file1A = sprintf("./ergebnis1A.dat")
2. file2A = sprintf("./ergebnis2A.dat")
3. file3A = sprintf("./ergebnis3A.dat")
4. file1B = sprintf("./ergebnis1B.dat")
5. file2B = sprintf("./ergebnis2B.dat")
6.
7. set pm3d map
8.
9. set term jpeg
10.set output '1A.jpg'
11.spl file1A u 1:2:3 with image
12.set term jpeg
13.set output '1B.jpg'
14.spl file1B u 1:2:3 with image
15.set term jpeg
16.set output '2A.jpg'
17.spl file2A u 1:2:3 with image
18.set term jpeg
19.set output '2B.jpg'
```

```
20. spl file2B u 1:2:3 with image
21. set term jpeg
22. set output '3A.jpg'
23. spl file3A u 1:2:3 with image
```

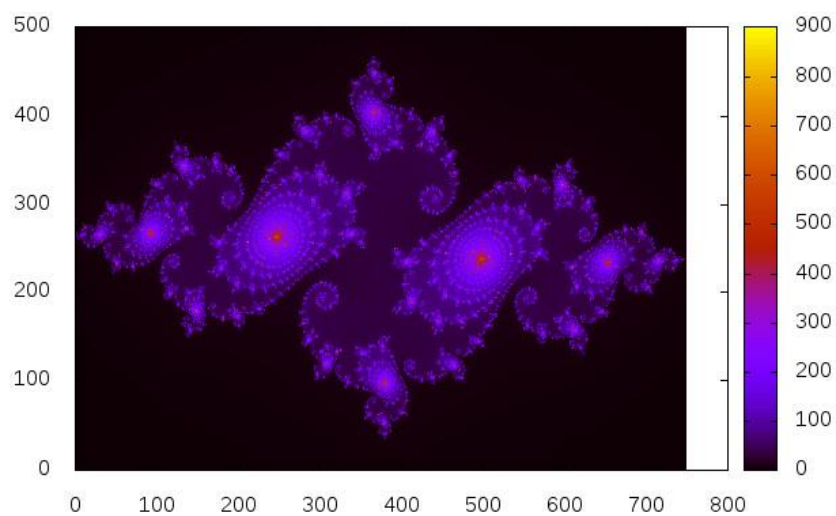
4. runs.sh

```
1. for i in $(seq 1 3)
2.
3. do
4.
5. echo "start with file: start${i}A.dat"
6. echo -e "\n"
7.
8. ./convergency < start${i}A.dat
9. echo -e "\n"
10. echo "finish!"
11. echo -e "\n"
12.
13. done
14.
15. for i in $(seq 1 2)
16.
17. do
18.
19. echo "start with file: start${i}B.dat"
20. echo -e "\n"
21.
22. ./convergency < start${i}B.dat
23. echo -e "\n"
24. echo "finish!"
25. echo -e "\n"
26.
27. done
28.
29. echo "
30.     load 'plot.gp'
31.     " | gnuplot
```

5. Ergebnisse

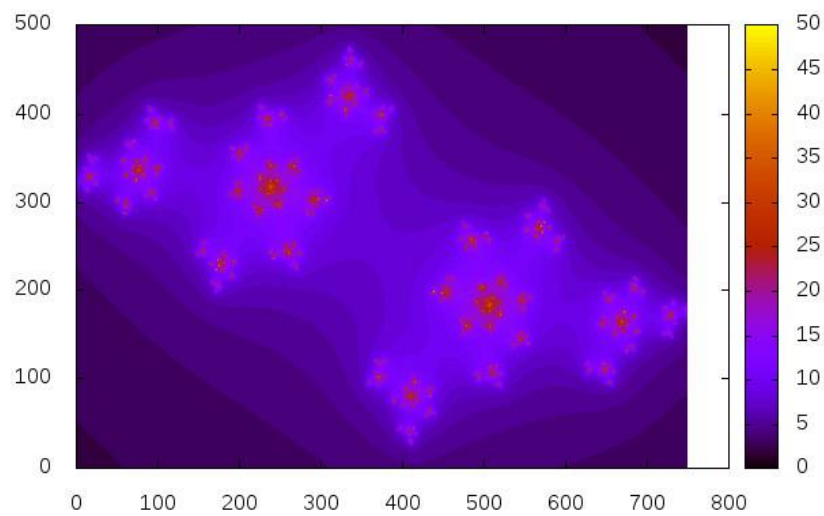
a) Beispiel 1A (Datei: start1A.dat)

Nummer der Iterationsvorschrift	1	
Wertebereich	$(-1.5, -1.)$	$(1.5, 1)$
Unterteilung (Nxmax,Nymax)	750	500
Exponent	2	
Nmax	2000	
Rc	100	
Dateiname	ergebnis1A.dat	
Komplexe Konstante c0	$(-0.75, 0.1)$	



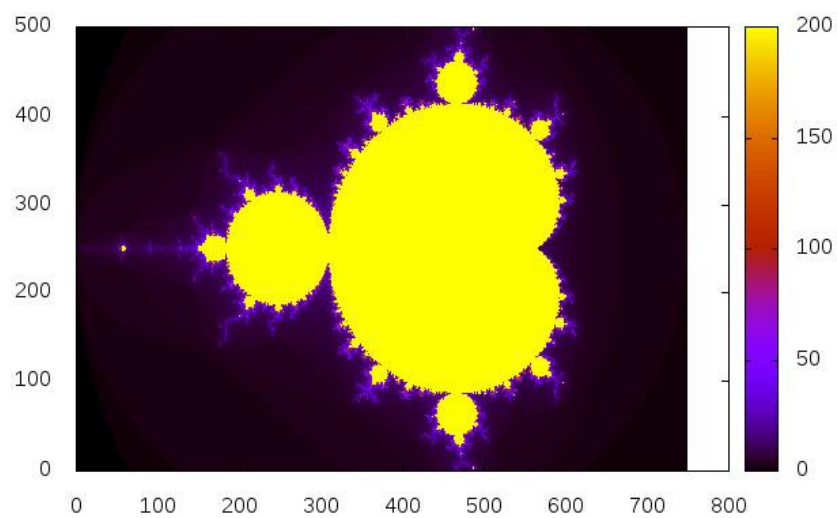
b) Beispiel 1B (Datei: start1B.dat)

Nummer der Iterationsvorschrift	1	
Wertebereich	$(-1.5, -1.)$	$(1.5, 1)$
Unterteilung (Nxmax,Nymax)	750	500
Exponent	2	
Nmax	2000	
Rc	100	
Dateiname	ergebnis1B.dat	
Komplexe Konstante c0	$(-0.75, 0.55)$	



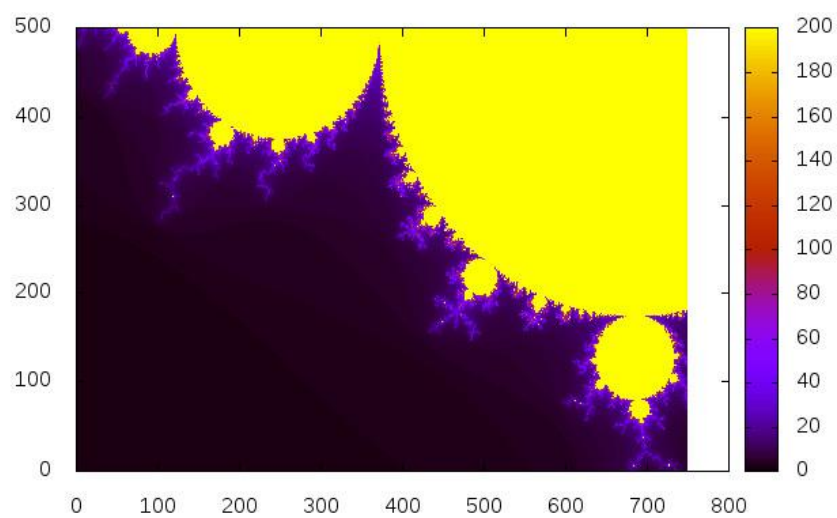
c) Beispiel 2A (Datei: start2A.dat)

Nummer der Iterationsvorschrift	2	
Wertebereich	$(-2, -1.)$	$(1, 1)$
Unterteilung (Nxmax,Nymax)	750	500
Exponent	2	
Nmax	200	
Rc	2	
Dateiname	ergebnis2A.dat	
Komplexe Konstante c0	$(0,0.)$	



d) Beispiel 2B (Datei: start2B.dat)

Nummer der Iterationsvorschrift	2	
Wertebereich	$(-1.5, 1.)$	$(0., 0.)$
Unterteilung (Nxmax,Nymax)	750	500
Exponent	2	
Nmax	200	
Rc	2	
Dateiname	ergebnis2B.dat	
Komplexe Konstante c0	$(0., 0.)$	



e) Beispiel 3A (Datei: start3A.dat)

Nummer der Iterationsvorschrift	3	
Wertebereich	$(-1.5, -1.)$	$(1.5, 1)$
Unterteilung (Nxmax,Nymax)	750	500
Exponent	4	
Nmax	2000	
Rc	200	
Dateiname	Ergebnis3A.dat	
Komplexe Konstante c0	$(0., 0.)$	

