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

```cpp
#ifndef COMPLEX_H
#define COMPLEX_H

class MyComplex{

private:
    double m_Real;
    double m_Image;

public:
    // con-/destructor
    MyComplex();
    MyComplex(const double re, const double im );
    MyComplex(const MyComplex & cplx);
    ~MyComplex();

    // copy operator =
    MyComplex & operator=(const MyComplex & cplx);

    // overload operator +-*/ (with complex number, real number on the left and on the right)
    const MyComplex operator+(const MyComplex & cplx) const;
    const MyComplex operator+(const double & real_number) const;
    friend const MyComplex operator+(const double & real_number, const MyComplex & cplx); // use friend to define out of class
    const MyComplex operator-(const MyComplex & cplx) const;
    const MyComplex operator-(const double & real_number) const;
```

```
    friend const MyComplex operator-(const double & real_number, const MyComplex & cplx);
    const MyComplex operator*(const MyComplex & cplx) const;
    const MyComplex operator*(const double & real_number) const;
    friend const MyComplex operator*(const double & real_number, const MyComplex & cplx);
    const MyComplex operator/(const MyComplex & cplx) const;
    const MyComplex operator/(const double & real_number) const;
    friend const MyComplex operator/(const double & real_number, const MyComplex & cplx);

    // overload operator ^
    const MyComplex operator^(const int & exp) const;

    const double real() const;
    const double imag() const;
    const double norm() const;
};

#endif /* COMPLEX_H */
```

2. complex.cpp

```
#include <iostream>
#include <cmath>

#include "complex.h"

using namespace std;

MyComplex::MyComplex(const double re, const double im ){ // constructor
    this->m_Real = re;
    this->m_Image = im;
}

MyComplex::MyComplex(const MyComplex & cplx){ // copy constructor
    this->m_Real = cplx.m_Real;
    this->m_Image = cplx.m_Image;
}

MyComplex::MyComplex(){this->m_Real = 0; this->m_Image = 0;} // default constructor
MyComplex::~MyComplex(){} // default destructor

```

```cpp
MyComplex & MyComplex::operator=(const MyComplex & cplx){ // overload = operator
    this->m_Real = cplx.m_Real;
    this->m_Image = cplx.m_Image;
}

const MyComplex MyComplex::operator+(const MyComplex & cplx) const{ // overload + operator (complex number)
    MyComplex temp;
    temp.m_Real = this->m_Real + cplx.m_Real;
    temp.m_Image = this->m_Image + cplx.m_Image;
    return temp;
}

const MyComplex MyComplex::operator+(const double & real_number) const{ // overload + operator (real number)
    MyComplex temp;
    temp.m_Real = this->m_Real + real_number;
    temp.m_Image = this->m_Image;
    return temp;
}

const MyComplex operator+(const double & real_number, const MyComplex & cplx){ // overload + operator (real number, static, left +)
    MyComplex temp;
    temp = cplx + real_number;
    return temp;
}

const MyComplex MyComplex::operator-(const MyComplex & cplx) const{ // overload - operator (complex number)
    MyComplex temp;
    temp.m_Real = this->m_Real - cplx.m_Real;
    temp.m_Image = this->m_Image - cplx.m_Image;
    return temp;
}

const MyComplex MyComplex::operator-(const double & real_number) const{ // overload - operator (real number)
    MyComplex temp;
    temp.m_Real = this->m_Real - real_number;
    temp.m_Image = this->m_Image;
    return temp;
```

```
}

const MyComplex operator-(const double & real_number, const MyComplex & cplx){ // overload - operator (real number, static, left -)
    MyComplex temp(real_number,0.);
    return temp - cplx;
}

const MyComplex MyComplex::operator*(const MyComplex & cplx) const{ // overload * operator (complex number)
    MyComplex temp;
    temp.m_Real = this->m_Real * cplx.m_Real - this->m_Image * cplx.m_Image;
    temp.m_Image = this->m_Image * cplx.m_Real + this->m_Real * cplx.m_Image;
    return temp;
}

const MyComplex MyComplex::operator*(const double & real_number) const{ // overload * operator (real number)
    MyComplex temp;
    temp.m_Real = this->m_Real * real_number;
    temp.m_Image = this->m_Image * real_number;
    return temp;
}

const MyComplex operator*(const double & real_number, const MyComplex & cplx){ // overload * operator (real number, static, left *)
    MyComplex temp;
    temp = cplx * real_number;
    return temp;
}

const MyComplex MyComplex::operator/(const MyComplex & cplx) const{ // overload / operator (complex number)
    MyComplex temp;
    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);
    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);
    return temp;
}

const MyComplex MyComplex::operator/(const double & real_number) const{ // overload / operator (real number)
```

```
    MyComplex temp;
    temp.m_Real = this->m_Real / real_number;
    temp.m_Image = this->m_Image / real_number;
    return temp;
}

const MyComplex operator/(const double & real_number, const MyComplex & cplx){ // overload / operator (real number, static, left /)
    MyComplex temp(real_number,0);
    return temp / cplx;
}

const MyComplex MyComplex::operator^(const int & exp) const{ // overload ^ operator
    MyComplex temp(1,0);
    for (int i = 0; i<exp; i++)
    {
        temp = temp * *this;
    }
    return temp;
}

const double MyComplex::real() const{
    return this->m_Real;
}

const double MyComplex::imag() const{
    return this->m_Image;
}

const double MyComplex::norm() const{
    return sqrt(this->m_Real * this->m_Real + this->m_Image * this->m_Image);
}
```

3. makefile

```
PROG = complex_test

FLAGS = -O2

CC = g++

SRCS = complex.cpp main_complex_beispiel.cpp

```

```
OBJ = $(SRCS:.cpp=.o)

all: $(SRCS) $(PROG)

$(PROG): $(OBJ)
	$(CC) $(FLAGS) $(OBJ) -o $@

%.o:%.cpp
	$(CC)  $(FLAGS) -c $<

clean:
	rm -rf *.o $(PROG)

complex.o: complex.cpp complex.h

main_complex_beispiel.o: main_complex_beispiel.cpp complex.cpp complex.h
```

4. Berechnungsbeispiel

Gegeben sind $z1 = 2.+7.\,i$ und $z2 = 42.-9.\,i$ und $z3 = -11.+19.\,i$.

a) $z4 = z1 * z2$

b) $z5 = z1 + z2$

c) $z6 = (z1 + z2) * 2.$

d) $z7 = (z2 + z3) * z1$

e) $z8 = z1 + 5.$

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/Bedingunen die Anzahl der Iterationschritte bei Erreichen des Konvergenzradiuses rückgeben.

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

plot.gp ist ein GNUPLOT-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

```cpp
#include "complex.h"
#include <iostream>
#include <fstream>

using namespace std;

int IsConvergency(MyComplex z0, int exp, MyComplex c, double rc, int N_max)
{
    MyComplex z_temp = z0;
    for (int i = 0; i < N_max; i++)
    {
        z_temp = (z_temp ^ exp) + c;
        if (z_temp.norm()>=rc) return i;
    }
    return N_max;
}

void calcu(int Iter_num, double value_l_real, double value_l_imag, double value_r_real, double value_r_imag,
            int Nxmax, int Nymax, int exp, int N_max, double rc, char filename[], double c_real, double c_imag)
```

```
{
    char file[256];
    sprintf(file,filename);
    ofstream calc_output;

    double dx = (value_r_real - value_l_real) / Nxmax;
    double dy = (value_r_imag - value_l_imag) / Nymax;

    calc_output.open(file);

    switch(Iter_num)
    {
        case 1:
        {
            MyComplex c(c_real, c_imag);
            for(int i = 0; i<=Nxmax; i++)
            {
                for(int j = 0; j<=Nymax; j++)
                {
                    MyComplex z0(double(i)*dx + value_l_real, double(j)*dy + value_l_imag);
                    calc_output<<i<<" "<<j<<" "<<IsConvergency(z0, exp, c, rc, N_max)<<endl;
                }
                calc_output<<endl;
            }
            break;
        }
        case 2:
        case 3:
        {
            MyComplex z0(c_real, c_imag);
            for(int i = 0; i<=Nxmax; i++)
            {
                for(int j = 0; j<=Nymax; j++)
                {
                    MyComplex c(double(i)*dx + value_l_real, double(j)*dy + value_l_imag);
                    calc_output<<i<<" "<<j<<" "<<IsConvergency(z0, exp, c, rc, N_max)<<endl;
                }
                calc_output<<endl;
            }
```

```
            break;
        }
    }
    calc_output.close();
}

int main()
{
    int Iter_num;
    double value_l_real, value_l_imag, value_r_real, value_r_imag;
    int Nxmax, Nymax, exp, N_max;
    double rc;
    char filename[256];
    double c_real, c_imag;

    cout<<"input number of iteration process"<<endl;
    cin>>Iter_num;
    cout<<"input start value and end value of complex"<<endl;
    cin>>value_l_real>>value_l_imag>>value_r_real>>value_r_imag;
    cout<<"input Nxmax and Nymax"<<endl;
    cin>>Nxmax>>Nymax;
    cout<<"input exponent"<<endl;
    cin>>exp;
    cout<<"input Nmax"<<endl;
    cin>>N_max;
    cout<<"input convergency radius"<<endl;
    cin>>rc;
    cout<<"input output filename"<<endl;
    cin>>filename;
    cout<<"input start value of c / z"<<endl;
    cin>>c_real>>c_imag;

    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);

    return 0;
}
```

2. makefile

```
PROG = convergency

FLAGS = -O2

```

```
CC = g++

SRCS = complex.cpp complex_convergency.cpp

OBJ = $(SRCS:.cpp=.o)

all: $(SRCS) $(PROG)

$(PROG): $(OBJ)
 $(CC) $(FLAGS) $(OBJ) -o $@

%.o:%.cpp
 $(CC)  $(FLAGS) -c $<

clean:
 rm -rf *.o $(PROG)

## dependencies

complex.o: complex.cpp complex.h

main_complex_beispiel.o: complex_convergency.cpp complex.cpp complex.h
```

3. plot.gp

```
file1A = sprintf("./ergebnis1A.dat")
file2A = sprintf("./ergebnis2A.dat")
file3A = sprintf("./ergebnis3A.dat")
file1B = sprintf("./ergebnis1B.dat")
file2B = sprintf("./ergebnis2B.dat")

set pm3d map

set term jpeg
set output '1A.jpg'
spl file1A u 1:2:3 with image
set term jpeg
set output '1B.jpg'
spl file1B u 1:2:3 with image
set term jpeg
set output '2A.jpg'
spl file2A u 1:2:3 with image
set term jpeg
set output '2B.jpg'
```

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

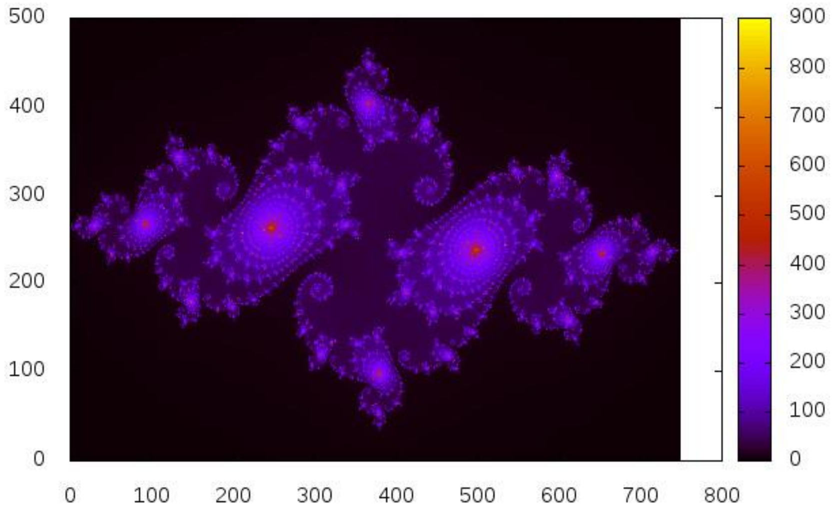
4. runs.sh

```
for i in $(seq 1 3)
do
echo "start with file: start${i}A.dat"
echo -e "\n"
./convergency < start${i}A.dat
echo -e "\n"
echo "finish!"
echo -e "\n"
done
for i in $(seq 1 2)
do
echo "start with file: start${i}B.dat"
echo -e "\n"
./convergency < start${i}B.dat
echo -e "\n"
echo "finish!"
echo -e "\n"
done
echo "
    load 'plot.gp'
    " | gnuplot
```

5. Ergebnisse

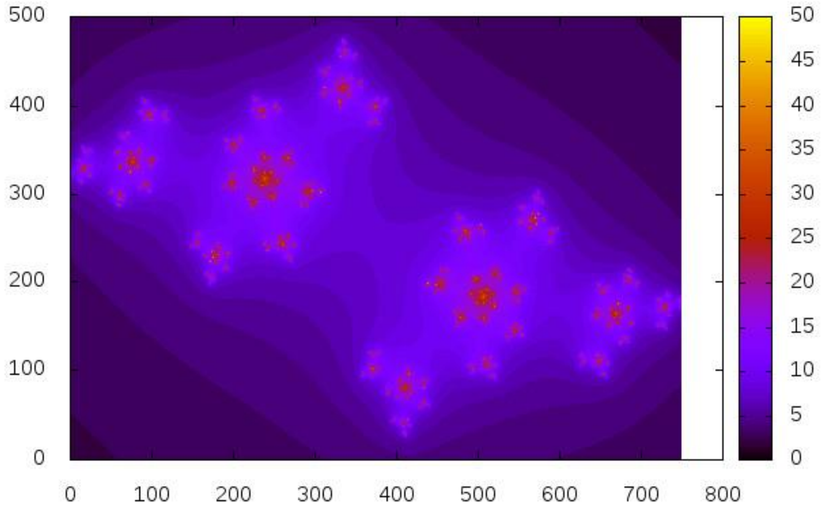
a) Beispiel 1A (Datei: start1A.dat)

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)	



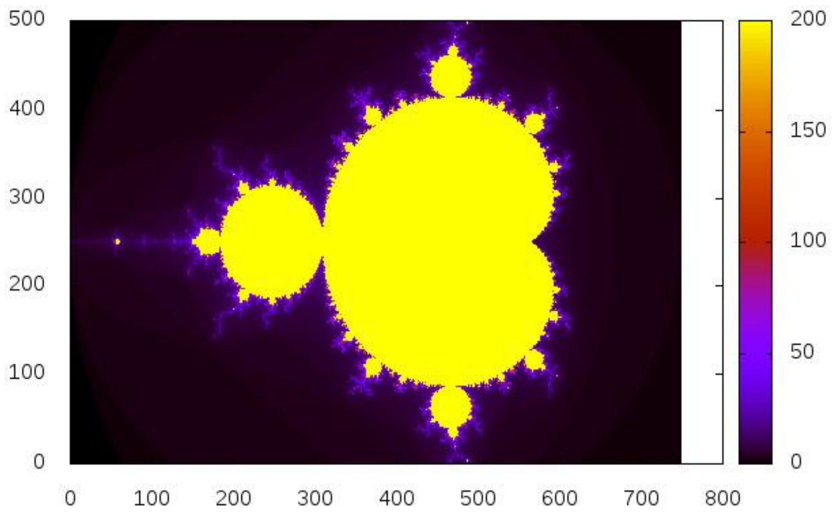
b) Beispiel 1B (Datei: start1B.dat)

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)	



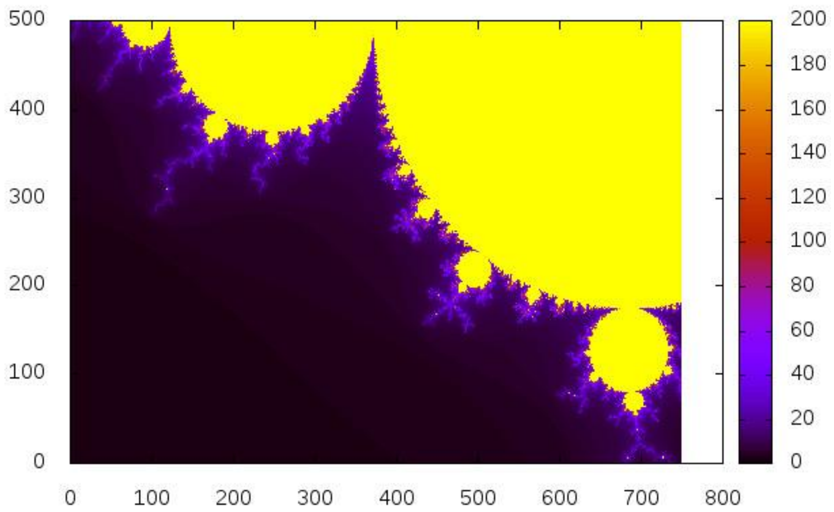
c) Beispiel 2A (Datei: start2A.dat)

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.)	



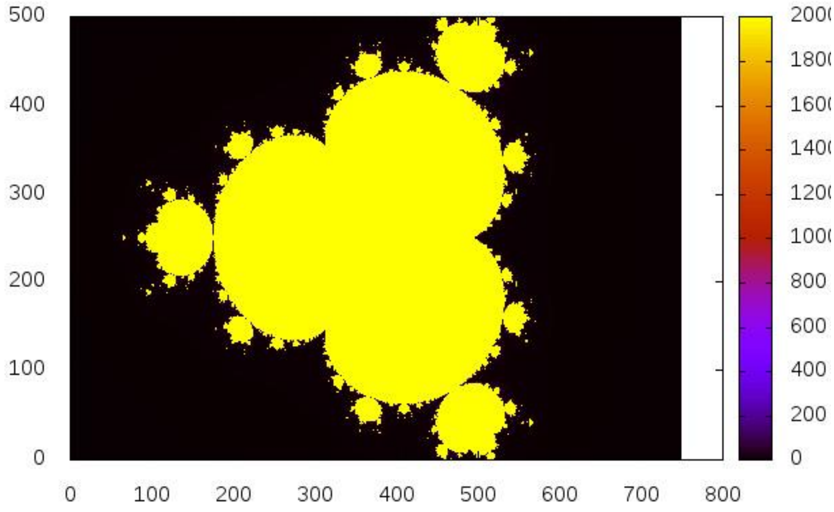
d) Beispiel 2B (Datei: start2B.dat)

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.)	


500
400
300
200
100
0
0
100
200
300
400
500
600
700
800
2000
1800
1600
1400
1200
1000
800
600
400
200
0