CES Softwareentwicklungspraktikum

Analyse- und Entwurfsdokument – Berechnung Rationaler Zahlen

Center for Computational Engineering Science
RWTH Aachen University

Uwe Naumann

LuFG Informatik 12
email: naumann@stce.rwth-aachen.de

Inhaltsverzeichnis

Kapitel 1

Vorwort

1.1 Aufgabenstellung und Struktur des Dokuments

Kurzversion der Aufgabenstellung und Organisation des Dokuments (Wo steht was?)

Dieses Dokument stellt ein Trivialbeispiel für die im Rahmen des CES Softwareentwicklungpraktikums zu bearbeitenden Aufgaben dar. Es dient vor allem zur Illustration der angestrebten Arbeitsabläufe. Alle UML Diagramme wurden mit Rational Software Architect (RSA) entwickelt. Der C++ Rahmencode wurde RSA basierend auf dem Klassenmodell automatisch generiert.

1.2 Projektmanagement

Arbeitsaufteilung innerhalb der Gruppe (Wer macht was?)

Naumann hat alles allein gemacht.

1.3 Lob und Kritik

z.B. Danksagung an Betreuer

... nichts zu danken ...

Kapitel 2

Analyse

2.1 Anforderungsanalyse

2.1.1 Benutzeranforderungen

Ausführliche Beschreibung der Aufgabenstellung (max. 1 Seite; Ziel: Konsens zwischen Auftraggeber und Auftragnehmer); Zusatzmaterial im Anhang

Es soll eine Software zur Berechnung des Gleitkommawertes rationaler Zahlen entwickelt werden [1].

2.1.2 Anwendungsfallanalyse

Statik: Anwendungsfalldiagramme; Dynamik: Aktivitätsdiagramme; Textuelle Beschreibungen laut Vorlage

Beschreibung der Anwendungsfälle

1. Eingabe einer ganzen Zahl
    - *Ziel:* Der Wert der ganzen Zahl ist gesetzt.
    - *Einordung:* Hauptfunktion
    - *Vorbedingung:* Das Menü wird angezeigt.
    - *Nachbedingung:* Das Menü wird angezeigt.
    - *Nachbedingung im Fehlerfall:* Eine Fehlermeldung wird auf den Bildschirm ausgegeben und das Menü wird angezeigt.
    - *Hauptakteure:* Nutzer
    - *Nebenakteure:*
    - *Auslöser:* Der Nutzer möchte den Wert einer ganzen Zahl eingeben.
    - *Standardablauf:*

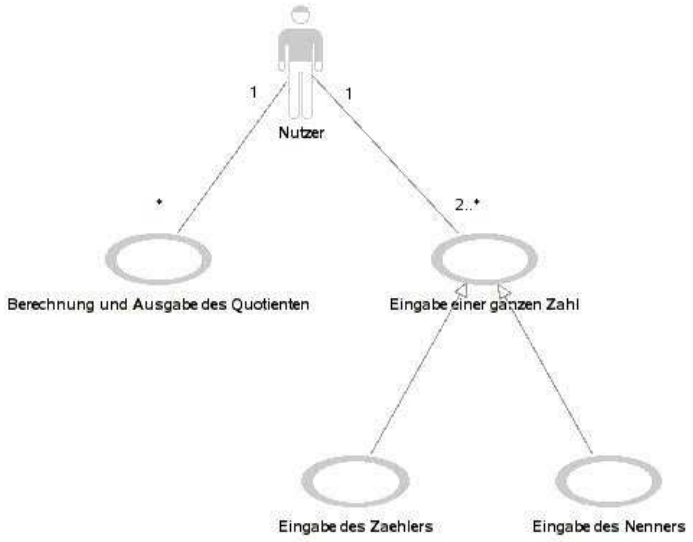

Abbildung 2.1: Anwendungsfalldiagramm

(1) Auswahl des Menüpunktes “Eingabe des Zählers” (bzw. “Eingabe des Nenners”) durch den Nutzer
(2) Eingabe der ganzen Zahl durch den Nutzer
(3) Speichern der ganzen Zahl durch das System
(4) Anzeige des Menüs durch das System

- *Verzweigungen:*

(2a) eingegebener Wert ist keine ganze Zahl
  * Ausgabe einer entsprechenden Fehlermeldung durch das System
  * Anzeige des Menüs durch das System

2. Berechnung und Ausgabe des Quotienten

- *Ziel:* Der Wert des Quotienten wird auf den Bildschirm ausgegeben.
- *Einordung:* Hauptfunktion
- *Vorbedingung:* Zähler und Nenner sind initialisiert. Das Menü wird angezeigt.
- *Nachbedingung:* Der Wert des Quotienten wird als GKZ auf den Bildschirm ausgegeben. Das Menü wird angezeigt.
- *Nachbedingung im Fehlerfall:* Eine Fehlermeldung wurde auf den Bildschirm ausgegeben und das Menü wird angezeigt.
- *Hauptakteure:* Nutzer

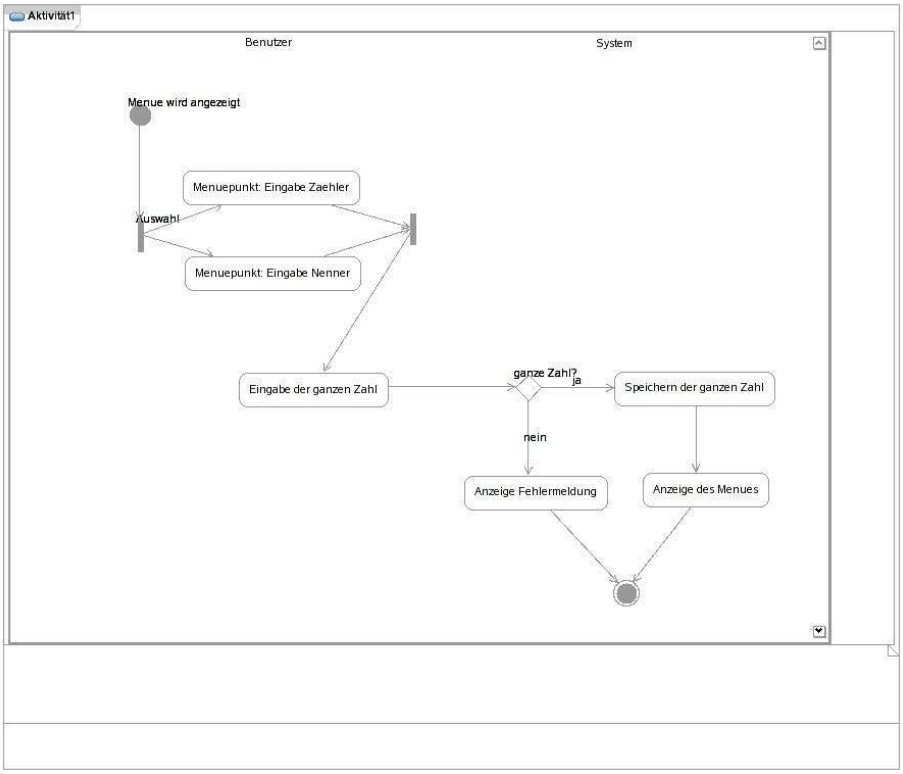

Abbildung 2.2: Aktivitätsdiagramm

- *Nebenakteure:*
- *Auslöser:* Der Nutzer möchte den Wert des Quotienten ermitteln.
- *Standardablauf:*
  (1) Auswahl des Menüpunktes “Berechnung und Ausgabe des Quotienten”durch den Nutzer
  (2) Lesen des gespeicherten Wertes z des Zählers durch das System
  (3) Lesen des gespeicherten Wertes n des Nenners durch das System
  (4) Berechnung von $q = z/n$ durch das System
  (5) Ausgabe von q im GKZ Format durch das System
  (6) Anzeige des Menüs durch das System
- *Verzweigungen:*
  (2a) Zähler ist nicht initialisiert
    * Ausgabe einer entsprechenden Fehlermeldung durch das System
    * Anzeige des Menüs durch das System
  (3a) Nenner ist nicht initialisiert
    * Ausgabe einer entsprechenden Fehlermeldung durch das System
    * Anzeige des Menüs durch das System
  (3b) Nenner ist gleich 0
    * Ausgabe einer entsprechenden Fehlermeldung durch das System
    * Anzeige des Menüs durch das System

3. Beenden des Programms

- *Ziel:* Das Programm soll beendet werden.
- *Einordung:* Hauptfunktion
- *Vorbedingung:* Das Menü wird angezeigt.
- *Nachbedingung:* Das Programm wurde beendet.
- *Nachbedingung im Fehlerfall:* Das Programm wurde beendet.
- *Hauptakteure:* Nutzer
- *Nebenakteure:*
- *Auslöser:* Der Nutzer möchte das Programm beenden.
- *Standardablauf:*
  (1) Auswahl Menüpunkt “Programm beenden”
- *Verzweigungen:*

Systemanforderungen

Beschreibung funktionaler und nichtfunktionaler Anforderungen basierend auf Anwendungsfällen (Ziel: Konsens zwischen Auftraggeber und Auftragnehmer)

Funktionale Anforderungen

1. Anzeige eines Menüs
2. Eingabe von Menüoptionen
3. Eingabe ganzer Zahlen
4. Ausgabe von GKZ
5. Ausgabe von Fehlermeldungen
6. Speichern ganzer Zahlen
7. Initialisierungsstatus für Zähler und Nenner
8. Division ganzer Zahlen
9. Test auf ganze Zahl
10. Test auf 0

Nichtfunktionale Anforderungen

- Kommandozeilenprogramm

2.2 Begriffsanalyse

Identifikation von Klassenkandidaten basierend auf Anforderungen; Assoziationen zwischen Klassenkandidaten (Aggregation, Komposition, Vererbung) und Kardinalitäten

Klassenkandidaten

- **rationale Zahl**
- Zähler ($\hat{=}$ ganze Zahl)
- **Nenner**
- **ganze Zahl**
- Initialisierungsstatus (durch ganze Zahl implementiert)
- Ouotient ($\hat{=}$ rationale Zahl)
- Wert ($\hat{=}$ ganze Zahl, rationale Zahl)
- Gleitkommazahl (durch rationale Zahl implementiert)
- Menü (durch `main` implementiert)
- Menüoption (durch `main` implementiert)
- Fehlermeldung (durch `main` implementiert)

Kapitel 3

Entwurf

3.1 Pakete

Grobe Strukturierung in Pakete (Namensbereiche)

... ist hier nicht sinnvoll

3.2 Abstrakte Datentypen

Vollständig spezifizierte Datentypen: Funktionen, Axiome, Bedingungen

3.2.1 GanzeZahl

Funktionen

```
new → GanzeZahl
set GanzeZahl × int → GanzeZahl
get GanzeZahl → int
is_set GanzeZahl → bool
```

Axiome

```
∀ z : GanzeZahl; i : int :

get (new) = 0;
is_set (new) = false;
get ( set (z,i) ) = i;
is_set ( set (z,i) ) = true;
```

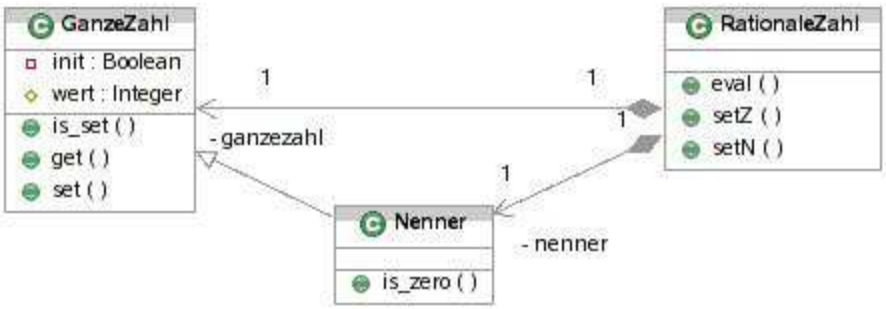

Abbildung 3.1: Klassenmodell

Bedingungen

3.2.2 RationaleZahl

Funktionen

```
new → RationaleZahl
setZ RationaleZahl × int → RationaleZahl
setN RationaleZahl × int → RationaleZahl
eval RationaleZahl → float
```

Axiome

∀ `r : RationaleZahl; z,n : int :`

```
q.setZ(z).setN(n).eval()=z/n
```

Bedingungen

3.3 Klassen

Klassenmodell (Statik: Klassendiagramm; Dynamik: Sequenzdiagramm, CRC Karten) inklusive (Klassen- bzw. Objekt-)Daten und (Klassen- bzw. Objekt-)Funktionen; Zugriffsrechte

Kapitel 4

Benutzerdokumentation

wohlstrukturierte und gut lesbare Dokumentation basierend auf den Anwendungsfällen

4.1 Installation

Die Software wird als `RaZa.zip` Archiv ausgeliefert. Zur Installation sind folgende Schritte notwendig.

1. Erstellen eines neuen Unterverzeichnisses (z.B. `./RaZa`);
2. Speichern von `RaZa.zip` in `./RaZa`;
3. Entpacken von `RaZa.zip` in `./RaZa` mittels `unzip RaZa.zip`;
4. Erstellung des ausführbaren Programms `RaZa` in `./RaZa` unter Verwendung von GNU Make[1] durch Eingabe von `make`.

Die Eingabe von `make clean` in `./RaZa` löscht alle generierten Dateien und stellt den Ausgangszustand des Codes wieder her. Die Bestimmung des Unterverzeichnisses `./RaZa/doc` wird in Kapitel 5 erläutert.

4.2 Beispielsitzung

Nach Start des Programms wird das Menü angezeigt.

```
++++++++++++++++++++++++++
1: Eingabe Zaehler
2: Eingabe Nenner
3: Ausgabe des Quotienten
0: Beenden des Programms
++++++++++++++++++++++++++
```

[1] `www.gnu.org/software/make`

Der Nutzer wählt Menüpunkt 1 aus und gibt anschliessend z.B. die ganze Zahl 3 ein. Es wird wiederum das Menü angezeigt.

```
1
3
++++++++++++++++++++++++++
1: Eingabe Zaehler
2: Eingabe Nenner
3: Ausgabe des Quotienten
0: Beenden des Programms
++++++++++++++++++++++++++
```

Der Nutzer wählt Menüpunkt 2 aus und gibt anschliessend z.B. die ganze Zahl 4 ein. Es wird wiederum das Menü angezeigt.

```
2
4
++++++++++++++++++++++++++
1: Eingabe Zaehler
2: Eingabe Nenner
3: Ausgabe des Quotienten
0: Beenden des Programms
++++++++++++++++++++++++++
```

Der Nutzer wählt Menüpunkt 3 aus. Es wird das Resultat der Division in Gleitkommadarstellung angezeigt gefolgt von der Anzeige des Menüs.

```
3
0.75
++++++++++++++++++++++++++
1: Eingabe Zaehler
2: Eingabe Nenner
3: Ausgabe des Quotienten
0: Beenden des Programms
++++++++++++++++++++++++++
```

Der Nutzer beendet das Programm durch Auswahl von Menüpunkt 0. Das Programm verabschiedet sich höflich...

```
0
Hope you had a good time! See you later.
```

4.3 Fehlersituationen

Eine entsprechende Fehlermeldung wird angezeigt (gefolgt von der Anzeige des Menüs)

1. wenn ein Menüpunkt $\notin \{0, 1, 2, 3\}$ gewählt wird;

2. wenn keine ganze Zahl für Zähler bzw. Nenner eingegeben wird (nicht implementiert);
3. bei Ausgabe des Quotienten falls entweder Zähler oder Nenner nicht gesetzt sind (nicht implementiert);
4. bei Ausgabe des Quotienten falls der Wert des Nenners gleich 0 ist.

Kapitel 5

Entwicklerdokumentation

wohlstrukturierte und gut lesbare Dokumentation der Software aus Entwicklersicht; zahlreiche Referenzen nach Kapitel 3 und in den Quellcode in Kapitel A

5.1 Codestruktur

Der Code besteht aus drei Klassen und einem Hauptprogramm. Alle Quelldateien befinden sich im selben Verzeichnis (z.B. `./RaZa`. Siehe auch Kapitel 4 für eine Installationsanleitung.

5.1.1 Klasse GanzeZahl

Die Klasse `GanzeZahl` implementiert den gleichnamigen ADT aus Sektion 3.2.1. Der gesamte Quellcode befindet sich in Sektion A.1.

5.1.2 Klasse Nenner

Die Klasse `Nenner` implementiert eine Spezialisierung des ADT `GanzeZahl` aus Sektion 3.2.1. Zusätzlich zur Grundfunktionalität ist ein Test des Wertes auf Null enthalten, um Divisionen durch Null zu vermeiden. Der gesamte Quellcode befindet sich in Sektion A.2.

5.1.3 Klasse RationaleZahl

Die Klasse `RationaleZahl` implementiert den gleichnamigen ADT aus Sektion 3.2.2. Ihre Hauptfunktion `float eval();` berechnet den Gleitkommawert der repäsentierten rationalen Zahl. Der gesamte Quellcode befindet sich in Sektion A.3.

5.1.4 Hauptprogramm

Das Hauptprogramm implementiert die Benutzeroberfläche. Der gesamte Quellcode befindet sich in Sektion A.4.

5.1.5 makefile

Das makefile steuert den Übersetzungsprozess. Der gesamte Quellcode befindet sich in Sektion A.5.

5.2 Detailierte Dokumentation des Codes

Im Unterverzeichnis `./RaZa/doc` befindet sich eine Konfigurationsdatei für das Quellcodedokumentationsprogramm `doxygen`[1] mit dem Namen `Doxyfile`. Durch Aufruf von `doxygen Doxyfile` werden navigierbare Dokumentationen des Codes in HTML (Unterverzeichnis `./RaZa/doc/html`) und LaTeX(Unterverzeichnis `./RaZa/doc/latex`) erstellt. Zur Generierung eines entsprechenden Dokuments im PDF-Format reicht ein `make` im Unterverzeichnis `./RaZa/doc/latex`. Die HTML-Dokumentation kann mit einem beliebigen Browser durch Öffnen der Datei `./RaZa/doc/html/index.html` angesehen werden.

Alternativ kann die Code Dokumentation auch mittels `make doc` erstellt werden. Siehe dazu auch Kapitel 4. Die Eingabe von `make cleandoc` löscht die beiden Unterverzeichnisse `./RaZa/doc/html` und `./RaZa/doc/latex` wieder.

5.3 Software Tests

Alle Tests wurden unter Linux (`i386-redhat-linux` mit `gcc version 4.1.2 20070925 (Red Hat 4.1.2-33)` durchgeführt.

5.3.1 Eingabebedingungen

1. Hauptmenü erlaubt ganzzahlige Eingaben $0, \ldots, 3$.
2. Zähler und Nenner sind ganze Zahlen.
3. Nenner is ungleich 0.

5.3.2 Äquivalenzklassenbildung und Testfälle

1. Hauptmenü
   (a) gültig
   - 0 (ok)
   - 3 (ok)

[1] `www.doxygen.org`

   (b) ungültig
      - -1 (ok)
      - 4 (ok)
      - a (nicht korrekt behandelt)

2. Zähler (Nenner gültig, z.B. = 1)
   (a) gültig
      - INT_MIN=$-2147483648 = -2^{31}$ (ok)
      - 0 (ok)
      - INT_MAX=$2147483647 = 2^{31} - 1$ (ok)
   (b) ungültig
      - INT_MIN-1 (nicht korrekt behandelt)
      - INT_MAX+1 (nicht korrekt behandelt)

3. Nenner (Zähler gültig, z.B. = 1)
   (a) gültig
      - INT_MIN (ok)
      - -1 (ok)
      - 1 (ok)
      - INT_MAX (ok)
   (b) ungültig
      - INT_MIN-1 (nicht korrekt behandelt)
      - 0 (ok)
      - INT_MAX+1 (nicht korrekt behandelt)

Offensichtlich hat die Software noch einige Sicherheitslücken, die es zu schliessen gilt.

Literaturverzeichnis

[1] Adam Ries. *Rechenung auff der Linihen und Federn.* Annaberg, 1522.

Anhang A

Quellcode

einfach referenzierbare Version des Quelltexts

A.1 GanzeZahl

Listing A.1: `GanzeZahl.h`

```
1 #ifndef GANZEZAHL_H
2 #define GANZEZAHL_H
3
4 /** Implmentierung des ADT GanzeZahl */
5
6 class GanzeZahl {
7
8 private:
9         bool init;
10
11 protected:
12         int wert;
13
14 public:
15         bool is_set();
16         int get();
17         void set(int v);
18
19         GanzeZahl(void);
20         ~GanzeZahl(void);
21 };
22
23 #endif /* GANZEZAHL_H */
```

Listing A.2: `GanzeZahl.cpp`

```
1 #include "GanzeZahl.h"
```

```
2
3 bool GanzeZahl::is_set(){
4   return init;
5 }
6
7 int GanzeZahl::get(){
8   return wert;
9 }
10
11 void GanzeZahl::set(int v){
12   wert=v; init=true;
13 }
14
15 GanzeZahl::GanzeZahl(void) : init(false)
16 {
17 }
18
19 GanzeZahl::~GanzeZahl(void)
20 {
21 }
```

A.2 Nenner

Listing A.3: `Nenner.h`

```
1 #ifndef NENNER_H
2 #define NENNER_H
3
4 #include "GanzeZahl.h"
5
6 /** Implementierung des ADT Nenner */
7
8 class Nenner : public GanzeZahl {
9
10 public:
11         bool is_zero();
12
13         Nenner(void);
14         ~Nenner(void);
15 };
16
17 #endif /* NENNER_H */
```

Listing A.4: `Nenner.cpp`

```
1 #include "Nenner.h"
2
3 bool Nenner::is_zero(){
4    return wert==0;
5 }
```

```
6
7 Nenner::Nenner(void)
8 {
9 }
10
11 Nenner::~Nenner(void)
12 {
13 }
```

A.3 Rationale Zahl

Listing A.5: `RationaleZahl.h`

```
1 #ifndef RATIONALEZAHL_H
2 #define RATIONALEZAHL_H
3
4 #include "Nenner.h"
5
6 /** Implementierung des ADT RationaleZahl */
7
8 class RationaleZahl {
9
10 private:
11         Nenner nenner;
12         GanzeZahl zaehler;
13
14 public:
15         float eval();
16         void setZ(int v);
17         void setN(int v);
18
19         RationaleZahl(void);
20         ~RationaleZahl(void);
21 };
22
23 #endif /* RATIONALEZAHL_H */
```

Listing A.6: `RationaleZahl.cpp`

```
1 #include <iostream>
2 #include "Nenner.h"
3 #include "GanzeZahl.h"
4 #include "RationaleZahl.h"
5
6 using namespace std;
7
8 float RationaleZahl::eval(){
9   if (!zaehler.is_set()) {
10     cerr << "ERROR: Nominator not set!" << endl;
11     return 0;
```

```
12   }
13   if (!nenner.is_set()) {
14     cerr << "ERROR: Denominator not set!" << endl;
15     return 0;
16   }
17   if (nenner.get()==0) {
18     cerr << "ERROR: Division by 0!" << endl;
19     return 0;
20   }
21   return zaehler.get()/(float) nenner.get();
22 }
23
24 void RationaleZahl::setZ(int v){
25   zaehler.set(v);
26 }
27
28 void RationaleZahl::setN(int v){
29   nenner.set(v);
30 }
31
32 RationaleZahl::RationaleZahl(void)
33 {
34 }
35
36 RationaleZahl::~RationaleZahl(void)
37 {
38 }
```

A.4 Hauptprogramm

Listing A.7: Hauptprogramm

```
1 #include <iostream>
2 using namespace std;
3
4 #include "RationaleZahl.h"
5
6 void display_menue() {
7   cout << "++++++++++++++++++++++++" << endl;
8   cout << "1: Eingabe Zaehler" << endl;
9   cout << "2: Eingabe Nenner" << endl;
10  cout << "3: Ausgabe des Quotienten" << endl;
11  cout << "0: Beenden des Programms" << endl;
12  cout << "++++++++++++++++++++++++" << endl;
13 }
14
15 int main() {
16   RationaleZahl q;
17   int choice=1;
18   while (choice) {
```

```
19      display_menue();
20      cin >> choice;
21      switch (choice) {
22        case 0: {
23          cout << "Hope you had a good time! See you later." <<
                endl;
24          break;
25        }
26        case 1: {
27          int z;
28          cin >> z;
29          q.setZ(z);
30          break;
31        }
32        case 2: {
33          int n;
34          cin >> n;
35          q.setN(n);
36          break;
37        }
38        case 3: {
39          cout << q.eval() << endl;
40          break;
41        }
42        default: {
43          cout << "ERROR: Invalid Option!" << endl;
44          break;
45        }
46      }
47    }
48    return 0;
49  }
```

A.5 makefile

Listing A.8: makefile

```
1  RaZa : main.o GanzeZahl.o Nenner.o RationaleZahl.o
2          g++ -o $@ $^
3
4  %.o : %.cpp
5          g++ -O4 -c $<
6
7  clean :
8          rm -f *.o RaZa
9
10 doc :
11         cd doc && doxygen Doxyfile
12
13 cleandoc:
```

```
14          cd doc && rm -fr html latex
15
16 .PHONY: clean doc cleandoc
17
18 # dependencies generated by g++ with -M option
19
20 GanzeZahl.o: GanzeZahl.h
21 Nenner.o: Nenner.h GanzeZahl.h
22 RationaleZahl.o: RationaleZahl.h Nenner.h GanzeZahl.h
23 main.o: RationaleZahl.h Nenner.h GanzeZahl.h
```