5 Übung 4

Vorbereitung

C++ Projekte bestehend aus mehreren Dateien

Unsere Aufgaben werden langsam komplexer: In den folgenden Aufgaben wird der Quellcode nicht nur aus einer Datei bestehen, sondern wir werden Klassen benutzen, die in separate Dateien ausgelagert werden.
Dieses Zusammenspiel müssen wir dem Compiler natürlich mitteilen. Dafür muss die `tasks.json` wie folgt angepasst werden:

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "build current",
            "type": "shell",
            "command": "g++",
            "args": [
                //"-g", "-Wall", "-std=c++11", "${file}"
                "-g", "-Wall", "-std=c++11", "${workspaceFolder}/Uebung4/Aufgabe4.1.cpp", "${workspaceFolder}/Uebung4/rationa
                //"-g", "-Wall", "-std=c++11", "${workspaceFolder}/Uebung4/Aufgabe4.2.cpp", "${workspaceFolder}/Uebung4/IntMe
                //"-g", "-Wall", "-std=c++11", "${workspaceFolder}/Uebung4/Aufgabe4.3.cpp", "${workspaceFolder}/Uebung4/Meine
                //"-g", "-Wall", "-std=c++11", "${workspaceFolder}/Uebung4/Aufgabe4.4.cpp", "${workspaceFolder}/Uebung4/Tasch
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            }

        }
    ]
}
```

Passen Sie ggf. die Pfade an Ihre eigene Ordnerstruktur an.
Kommentieren Sie jeweils die Zeile mit dem Compile-Argument zur jeweiligen Aufgabe ein (bzw. kommentieren Sie die Zeile von der vorherigen Aufgabe aus).

Die in der `tasks.json` angegebene Zeile zum Kompilieren der Aufgabe 4.1 entspricht dem Befehl

```
g++ -Wall -std=c++11 Aufgabe4.1cpp rational.cpp
```

im Terminal.

Hinweis: Verfahren Sie analog für alle folgenden Aufgaben dieses Kurses!

Visual Studio Code - IntelliSense

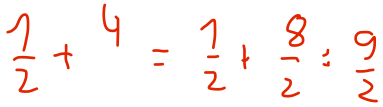
Dadurch, dass jetzt mit mehreren Dateien pro Aufgabe gearbeitet wird, benötigen wir eine verbesserte automatische Vervollständigung (IntelliSense). Erzeugen Sie dafür ein `c_cpp_properties.json` file indem Sie per `Strg + Shift + P` die Begriffe `"Edit configurations (JSON)"` eingeben. Überprüfen Sie den `compilerPath` in der Datei und speichern Sie anschließend das Dokument.

Aufgabe 4.1

Die folgenden beiden Aufgaben beziehen sich auf die in der Vorlesung vorgestellte Klasse Rational (Quellcode siehe unten).

Schreiben Sie die Methode `add(long a)`, um eine long-Zahl zu einer rationalen Zahl addieren zu können. Testen Sie anschließend Ihre Funktion.

Quellcode `rational.h`



```
// Klasse für rationale Zahlen
////////////////////////////////

#ifndef RATIONAL_H
#define RATIONAL_H

class Rational {
   public:
     Rational();
     Rational(long z, long n);   // allgemeiner Konstruktor

     // Abfragen
     long getZaehler() const;
     long getNenner()  const;

     // arithmetische Methoden
     // (werden später durch überladene Operatoren ergänzt)
     void add(const Rational& r);
     void sub(const Rational& r);
     void mult(const Rational& r);
     void div(const Rational& r);

     // weitere Methoden
     void set(long zaehler, long nenner);
     void eingabe();
     void ausgabe() const;
     void kehrwert();
     void kuerzen();
```

```
29
30    private:
31      long zaehler, nenner;
32 };
33
34 // inline Methoden
35 inline Rational::Rational()               : zaehler(0), nenner(1) {}
36 inline Rational::Rational(long z, long n) : zaehler(z), nenner(n) {}
37
38 inline long Rational::getZaehler() const {return zaehler;}
39 inline long Rational::getNenner()  const {return nenner;}
40
41 #endif
```

Quellcode `rational.cpp`

```
 1 // Klasse für rationale Zahlen
 2 // (Definition der Methoden)
 3 ////////////////////////////////
 4 #include"rational.h"
 5 #include<iostream>
 6 #include<cassert>
 7 using namespace std;
 8
 9 // Elementfunktionen
10
11 void Rational::add(const Rational& r) {
12      zaehler = zaehler*r.nenner + r.zaehler*nenner;
13      nenner  = nenner*r.nenner;
14      kuerzen();
15 }
16
17 void Rational::sub(const Rational& s) {
18      Rational r = s;
19      r.zaehler *=-1;
20      add(r);
21 }
22
23 void Rational::mult(const Rational& r) {
24      zaehler = zaehler*r.zaehler;
25      nenner  = nenner *r.nenner;
26      kuerzen();
27 }
28
29 void Rational::div(const Rational& n) {
30      Rational r = n;
31      r.kehrwert();
32      mult(r);
33 }
34
35 void Rational::set(long z, long n) {
36      zaehler = z;
37      nenner  = n;
38      assert(nenner != 0);
39      kuerzen();
40 }
41
42 void Rational::eingabe() {
43      cout << "Zähler :";
44      cin >> zaehler;
45      cout << "Nenner :";
46      cin >> nenner;
47      assert(nenner != 0);
48      kuerzen();
49 }
50
51 void Rational::ausgabe() const {
52      cout << zaehler << "/" << nenner << endl;
53 }
54
55 void Rational::kehrwert() {
56      long temp = zaehler;
57      zaehler = nenner;
58      nenner  = temp;
59      assert(nenner != 0);
60 }
61
62 long ggt(long x, long y) {
63     long rest;
64     while (y > 0) {
65        rest = x % y;
66        x = y;
67        y = rest;
68     }
69     return x;
70 }
71
72 void Rational::kuerzen() {
73      int sign = 1;
74      if (zaehler < 0) { sign=-sign; zaehler = -zaehler;}
75      if (nenner  < 0) { sign=-sign; nenner  = -nenner;}
76      long teiler = ggt(zaehler, nenner);
77      zaehler = sign*zaehler/teiler;
78      nenner = nenner/teiler;
79 }
```

Stellen wir uns vor, dass wir nicht nur `long`-Zahlen addieren wollen, sondern auch noch subtrahieren, multiplizieren und dividieren. Sie müssten jetzt für jede dieser Funktionen eine weitere Überladung implementieren. Ganz schön aufwändig, nicht? :)

In der Vorlesung haben Sie den sogenannten Typumwandlungskonstruktor kennengelernt. Dieser kann es Ihnen ersparen diese zusätzlichen Funktionen zu implementieren (auch das von Ihnen zuvor implementierte `add(long a)`). Kommentieren Sie also Ihre neu implementierte Funktion aus. Ihre `main()` sollte nun nicht mehr kompilierbar sein.
Schreiben Sie nun einen Typumwandlungskonstruktor, der automatisch eine rationale Zahl aus einer `long`-Zahl erzeugt. Dies sollte dazu führen, dass Ihre `main()` ohne weitere Änderungen wieder kompiliert. Wieso ist das Fall? Geben Sie Ihre Begründung als Kommentar in `rational.h` über dem Typumwandlungskonstruktor an.

Aufgabe 4.2

Schreiben Sie eine Klasse `IntMenge`, bestehend aus den zwei Dateien `IntMenge.h` und `IntMenge.cpp`, sowie ein Testprogramm `main.cpp`. Die Klasse soll eine mathematische Menge für ganze Zahlen nachbilden. Es sollen nur die folgenden einfachen Funktionen möglich sein, auf Operationen mit zwei Mengen wie Vereinigung und Durchschnitt werde verzichtet:

- void `hinzufuegen(int el)`: Element `el` hinzufügen, falls es noch nicht existiert, andernfalls nichts tun.
- `void entfernen(int el)`: Element `el` entfernen, falls es vorhanden ist, andernfalls nichts tun.
- `bool istMitglied(int el)`: Gibt an, ob `el` in der Menge enthalten ist.
- `size_t size()`: Gibt die Anzahl der gespeicherten Elemente zurück.
- `void anzeigen()`: Gibt alle Elemente auf der Standardausgabe aus.
- `void loeschen()`: Alle Elemente löschen.
- `int getMax()` und `int getMin()`: Geben das größte bzw. kleinste Element zurück.

Benutzen Sie intern zum Speichern der Werte ein `vector`-Objekt. Schreiben Sie auch einen Kopier-Konstruktor, der die echte Kopie der Ausgangsmenge erzeugt. Ein Auszug einer Anwendung könnte etwa wie folgt aussehen:

```
1  IntMenge menge;
2  menge.hinzufuegen(2); // ok
3  menge.hinzufuegen(-9); // ok
4  menge.hinzufuegen(2); // keine Wirkung, 2 gibt es schon
5  menge.entfernen(99); // keine Wirkung, nicht vorhanden
6  menge.entfernen(-9); // ok
7  menge.anzeigen();
8  menge.loeschen();
9  for(int i=17; i < 33; ++i) {
10     menge.hinzufuegen(i * i);
11 }
12 cout << "Anzahl=" << menge.size() << "Minimum=" <<
13 menge.getMin();
14 if(menge.istMitglied(-11)) { ? }
15 Menge mengeB(menge);
16 if(mengeB.istMitglied(-11)) { ? }
17 ...
```

Sie sollten nach diesem Kurs in der Lage sein, C++-Programme im Terminal zu kompilieren und nicht nur per IDE auszuführen.

Wie lautet der einzeilige Kompilierbefehl, der aus der Datei Aufgabe4.2cpp und Ihrer Abhängigkeit zur Klasse IntMenge eine ausführbare Datei a.out (Linux) / a.exe (Windows) erzeugt?

1.

Auswerten

Aufgabe 4.3

Wie können Sie mit C++ erreichen, dass ein Attribut direkt, also ohne Einsatz einer Methode, zwar gelesen, aber nicht durch eine Zuweisung verändert werden kann? Dabei soll das Attribut aber immer noch innerhalb der Klasse (oder mit der Methode `aendern(int)`) veränderbar sein.

Beispiel

```
1 int main() {
2     MeineKlasse objekt;
3     objekt.readonlyAttribut = 999; // Fehler! nicht erlaubte Aktion
4     objekt.aendern(999);              // OK, Setter funktioniert!
5     // erlaubter direkter lesender Zugriff:
6     cout << "objekt.readonlyAttribut="
7     << objekt.readonlyAttribut << endl;    // ok
8 }
```

Wie sieht die Realisierung in der Klasse `MeineKlasse` aus?

static extern int readonly Attribut

Aufgabe 4.4

Schreiben Sie auf Basis der Lösung der Aufgabe „Taschenrechner“ aus Übung 3 eine Klasse `Taschenrechner`, die einen eingegebenen `String` verarbeitet. Die Anwendung könnte wie folgt aussehen:

```
1  int main() {
2      while(true) {
3          // Abbruch mit break
4          cout << "Bitte einen mathematischen Ausdruck eingeben , z.B. 4*(12+3)"
5                 << "\n(Abbruch durch Eingabe einer Leerzeile) : " ;
6          string anfrage;
```

```
 7          getline(cin, anfrage);
 8          if(anfrage.length() > 0) {
 9              Taschenrechner tr(anfrage);
10              cout << "Das Ergebnis der Anfrage '"
11                   << tr.getAnfrage() << ' ist " << tr.getErgebnis() << endl;
12          }
13          else break;
14      }
15  }
```

Die ggf. verwendete Funktion `cin.get(c);` muss dabei durch den Aufruf einer Funktion ersetzt werden, die das jeweils nächste Zeichen des übergebenen Anfrage-Strings holt!